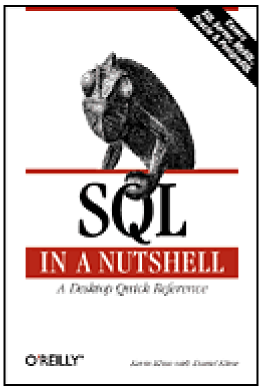

SQL in a Nutshell

By Kevin Kline with Daniel Kline, Ph.D.
January 2001
1-56592-744-3, Order Number: 7443
224 pages, $29.95

Chapter 4
SQL Functions

A *function* is a special type of command word in the SQL99 command set. In effect, functions are one-word commands that return a single value. The value of a function can be determined by input parameters, as with a function that averages a list of database values. But many functions do not use any type of input parameter, such as the function that returns the current system time, *CURRENT_ TIME*.

The SQL99 standard supports a number of useful functions. This chapter covers those functions, providing detailed descriptions and examples. In addition, each database vendor maintains a long list of their own internal functions that are outside of the scope of the SQL standard. Lists and descriptions are provided for each database implementation's internal functions.

In addition, most database vendors support the ability to create user-defined functions (UDF). For more information on UDFs, refer to the *CREATE FUNCTION* command in *Chapter 3, SQL Statements Command Reference*.

Deterministic and Nondeterministic Functions

Functions can be either deterministic or nondeterministic. A *deterministic* function always returns the same results if given the same input values. A *nondeterministic* function returns different results every time it is called, even when the same input values are provided.

Why is this important? It is important because of how functions may be used within views, user-defined functions, and stored procedures. The restrictions vary across implementations, but these objects sometimes allow only deterministic functions within their defining code. For example, Microsoft SQL Server allows the creation of an index on a column expression--as long as the expression does not contain nondeterministic functions. Rules and restrictions vary between the vendors, so check their documentation when using functions.

Types of Functions

There are several basic types and categories of functions in SQL99 and vendor implementations of SQL. The basic types of functions are:

Aggregate functions
: Operate against a collection of values, but return a single, summarizing value.

Scalar functions
: Operate against a single value, and return a single value based on the input value. Some scalar functions, *CURRENT_TIME* for example, do not require any arguments.

Aggregate Functions

Aggregate functions return a single value based upon a set of other values. If used among many other expressions in the item list of a *SELECT* statement, the *SELECT* must have a *GROUP BY* clause. No *GROUP BY* clause is required if the aggregate function is the only value retrieved by the *SELECT* statement. The supported aggregate functions and their syntax are listed in Table 4-1.

Table 4-1: SQL99 Aggregate Functions

Function	**Usage**
AVG(expression)	Computes the average value of a column by the expression
COUNT(expression)	Counts the rows defined by the expression
COUNT(*)	Counts all rows in the specified table or view
MIN(expression)	Finds the minimum value in a column by the expression
MAX(expression)	Finds the maximum value in a column by the expression
SUM(expression)	Computes the sum of column values by the expression

Technically speaking, *ANY*, *EVERY*, and *SOME* are considered aggregate functions. However, they have been discussed as range search criteria since they are most often used that way. Refer to the *SELECT* . . . *WHERE* topic in the previous chapter for more information on these functions.

The number of values processed by an aggregate varies depending on the number of rows queried from the table. This behavior makes aggregate functions different from scalar functions, which require a fixed number and fixed type of parameters.

The general syntax of an aggregate function is:

```
aggregate_function_name ( [ALL | DISTINCT] expression )
```

The aggregate function name may be *AVG*, *COUNT*, *MAX*, *MIN*, or *SUM*. The *ALL* clause, which is the default behavior and does not actually need to be specified, evaluates all rows when aggregating the value of the function. The *DISTINCT* clause uses only distinct values when evaluating the function.

AVG and SUM

The *AVG* function computes the average of values in a column or an expression. *SUM* computes the sum. Both functions work with numeric values and ignore NULL values. They also can be used to compute the average or sum of all *distinct* values of a column or expression.

AVG and *SUM* are supported by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL.

Example

The following query computes average year-to-date sales for each type of book:

```
SELECT   type, AVG( ytd_sales ) AS "average_ytd_sales"
FROM     titles
GROUP BY type;
```

This query returns the sum of year-to-date sales for each type of book:

```
SELECT   type, SUM( ytd_sales )
FROM     titles
GROUP BY type;
```

COUNT

The *COUNT* function has three variations. *COUNT(*)* counts all the rows in the target table whether they include nulls or not. *COUNT(expression)* computes the number of rows with non-NULL values in a specific column or expression. *COUNT(DISTINCT expression)* computes the number of distinct non-NULL values in a column or expression.

Examples

This query counts all rows in a table:

```
SELECT COUNT(*) FROM publishers;
```

The following query finds the number of different countries where publishers are located:

```
SELECT COUNT(DISTINCT country) "Count of Countries"
FROM   publishers
```

MIN and MAX

MIN(expression) and MAX(expression) find the minimum and maximum value (string, datetime, or numeric) in a set of rows. *DISTINCT* or *ALL* may be used with these functions, but they do not affect the result.

MIN and *MAX* are supported by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL.

MySQL also supports the functions *LEAST( )* and *GREATEST( )*, providing the same capabilities.

Examples

The following query finds the best and worst sales for any title on record:

```
SELECT  'MIN' = MIN(ytd_sales), 'MAX' = MAX(ytd_sales)
FROM    titles;
```

Aggregate functions are used often in the *HAVING* clause of queries with *GROUP BY*. The following query selects all categories (types) of books that have an average price for all books in the category higher than $15.00:

```
SELECT  type 'Category', AVG( price ) 'Average Price'
FROM    titles
GROUP BY type
HAVING AVG(price) > 15
```

Scalar Functions

Scalar functions fall into the categories listed in Table 4-2.

Table 4-2: Categories of Scalar Functions

Function Category	Explanation
Built-in	Performs operations on values or settings built into the database. Oracle uses the term "built-in" to describe all the specialty functions that are provided by Oracle, and thus "built into" their DBMS. This is a distinct and separate usage from the built-in functions described here.
Date & Time	Performs operations on datetime fields and returns values in datetime format.
Numeric	Performs operations on numeric values and returns numeric values.
String	Performs operations on character values (*char* , *varchar*, *nchar*, *nvarchar*, *and CLOB*) and returns a string or numeric value.

Note that *CASE* and *CAST* are both functions. However, they are detailed in Chapter 3 because of their complexity and frequent usage in SQL-data statements.

Built-in Scalar Functions

SQL99 built-in scalar functions identify the current user session, and also characteristics of the current user session, such as the current session privileges. Built-in scalar functions are almost always nondeterministic. The first three functions listed in Table 4-3 are built-in functions that fall into the date-and-time category of functions. Although the four vendors provide many additional functions beyond these SQL built-ins, the SQL standard declares only those listed in Table 4-3.

Table 4-3: SQL99 Built-in Scalar Functions

Function	Usage
CURRENT_DATE	Identifies the current date.
CURRENT_TIME	Identifies the current time.
CURRENT_TIMESTAMP	Identifies the current date and time.
CURRENT_USER	Identifies the currently active user within the database server.
SESSION_USER	Identifies the currently active Authorization ID, if it differs from the user.
SYSTEM_USER	Identifies the currently active user within the host operating system.

Microsoft SQL Server supports all the built-in scalar functions. Oracle does not support the built-in scalar functions shown above; however, it supports *USER* as a synonym of *CURRENT_USER* and *SYSDATE* as a synonym of *CURRENT_TIMESTAMP*. MySQL supports all the SQL99 built-in scalar functions, plus both of Oracle's variants. PostgreSQL supports *USER*, as defined in SQL99, as a synonym for *CURRENT_USER*. In addition, MySQL supports *NOW( )* and *UNIX_TIMESTAMP( )* as synonyms of the function *CURRENT_TIMESTAMP*. PostgreSQL supports all the SQL99 built-in scalar functions except *SESSION_USER*.

Example

The following queries retrieve the values from built-in functions. Notice that the various vendors return dates in their native formats:

```
/* On MySQL */
SELECT CURRENT_TIMESTAMP;
-> '2001-12-15 23:50:26'

/* On Microsoft SQL Server */
SELECT CURRENT_TIMESTAMP
GO
-> 'Dec 15,2001 23:50:26'

/* On Oracle */
SELECT USER FROM dual;
-> dylan
```

Numeric Scalar Functions

The list of official SQL99 numeric functions is rather small. The various vendors provide quite a large supplement of mathematical and statistical functions. MySQL supports many of these commands in its SQL99 incarnations. The other database products offer the same capabilities of numeric scalar functions through their own internally defined functions, but they do not share the same name as those declared by the SQL standard. The supported numeric functions and syntax are listed in Table 4-4.

Table 4-4: SQL99 Numeric Functions

Function	**Usage**
BIT_LENGTH(expression)	Returns an integer value representing the number of bits in an expression.
CHAR_LENGTH(expression)	Returns an integer value representing the number of characters in an expression.
EXTRACT(datetime_ expression datepart FROM expression)	Allows the datepart to be extracted (*YEAR*, *MONTH*, *DAY*, *HOUR*, *MINUTE*, *SECOND*, *TIMEZONE_HOUR*, or *TIMEZONE_ MINUTE*) from an expression.
OCTET_LENGTH(expression)	Returns an integer value representing the number of octets in an expression. This value is the same as *BIT_LENGTH*/8.
POSITION(starting_string *IN search_string)*	Returns an integer value representing the starting position of a string within the search string.

BIT_LENGTH, CHAR_LENGTH, and OCTET_LENGTH

The closest any of the vendors get to the *BIT_LENGTH* function is Oracle. Oracle supports the *LENGTHB* function, which returns an integer value representing the number of bytes in an expression.

MySQL and PostgreSQL support *CHAR_LENGTH* and the SQL99 synonym *CHARACTER_LENGTH( )*. PostgreSQL also supports *EXTRACT( )*, *OCTET_LENGTH( )*, and *POSITION( )* as per the SQL99 standard. The other two vendors each have a similar function that provides identical functionality. SQL Server provides the *LEN* function and Oracle provides the *LENGTH* function.

MySQL and PostgreSQL also fully support the *OCTET_LENGTH* function.

Example

The following example determines the length of a string and a value retrieved from a column:

```
/* On MySQL and PostgreSQL */
SELECT CHAR_LENGTH('hello');
SELECT OCTET_LENGTH(book_title) FROM titles;

/* On Microsoft SQL Server */
SELECT DATALENGTH(title)
FROM titles
WHERE type = 'popular_comp'
GO

/* On Oracle */
SELECT LENGTH('HORATIO') "Length of characters"
FROM dual;
```

EXTRACT

The *EXTRACT* function is not supported by the database vendors, except for PostgreSQL and MySQL.

Each vendor supports a separate command to accomplish the same functionality. Oracle uses the *TO_CHAR* function to extract a portion of a date into a character string. SQL Server uses the *CONVERT* function to extract a portion of a date.

MySQL implementation is extended somewhat beyond the SQL99 standard. The SQL99 standard does not have a provision for returning multiple fields in the same call to *EXTRACT( )* (e.g., "DAY_HOUR"). The MySQL extensions try to accomplish what the combination *DATE_TRUNC( )* and *DATE_PART( )* do in PostgreSQL. MySQL supports the dateparts listed in Table 4-5.

Table 4-5: MySQL Dateparts

Type value	Meaning	Expected format
SECOND	Seconds	SECONDS
MINUTE	Minutes	MINUTES
HOUR	Hours	HOURS
DAY	Days	DAYS
MONTH	Months	MONTHS
YEAR	Years	YEARS
MINUTE_SECOND	Minutes and seconds	"MINUTES:SECONDS"
HOUR_MINUTE	Hours and minutes	"HOURS:MINUTES"
DAY_HOUR	Days and hours	"DAYS HOURS"
YEAR_MONTH	Years and months	"YEARS-MONTHS"

HOUR_SECOND	Hours, minutes, seconds	"HOURS:MINUTES:SECONDS"
DAY_MINUTE	Days, hours, minutes	"DAYS HOURS:MINUTES"
DAY_SECOND	Days, hours, minutes, seconds	"DAYSHOURS:MINUTES:SECONDS"

Example

This example extracts dateparts from several datetime values:

```
/* On MySQL  */
SELECT EXTRACT(YEAR FROM "2013-07-02");
-> 1999
SELECT EXTRACT(YEAR_MONTH FROM "2013-07-02 01:02:03");
-> 199907
SELECT EXTRACT(DAY_MINUTE FROM "2013-07-02 01:02:03");
-> 20102
```

POSITION

The *POSITION* function returns an integer that indicates the starting position of a string within the search string. MySQL and PostgreSQL support the *POSITION* function with no variation from the SQL99 syntax. PostgreSQL has a synonymous function, *TEXTPOS*, while MySQL has the synonymous function, *LOCATE*.

Oracle's equivalent function is called *INSTR*. Microsoft SQL Server has both *CHARINDEX* and *PATINDEX*. The *CHARINDEX* and *PATINDEX* are very similar, except that *PATINDEX* allows the use of wildcard characters in the search criteria. For example:

```
/* On MySQL */
SELECT LOCATE('bar', 'foobar');
-> 4

/* On MySQL and PostgreSQL */
SELECT POSITION('fu' IN 'snafhu');
-> 0

/* On Microsoft SQL Server */
SELECT CHARINDEX( 'de', 'abcdefg' )
GO
-> 4
SELECT PATINDEX( '%fg', 'abcdefg' )
GO
-> 6
```

String Functions

Basic string functions offer a number of capabilities and return a string value as a result set. Some string functions are dyadic, indicating that they operate on two strings at once. SQL99 supports the string functions listed in Table 4-6.

Table 4-6: SQL String Functions

Function	Usage

CONCATENATE (expression || expression)	Appends two or more literal expressions, column values, or variables together into one string.
CONVERT	Converts a string to a different representation within the same character set.
LOWER	Converts a string to all lowercase characters.
SUBSTRING	Extracts a portion of a string.
TRANSLATE	Converts a string from one character set to another.
TRIM	Removes leading characters, trailing characters, or both from a character string.
UPPER	Converts a string to all uppercase characters.

CONCATENATE

SQL99 defines a concatenation operator (||), which joins two distinct strings into one string value. The *CONCATENATE* function appends two or more strings together, producing a single output string. PostgreSQL and Oracle support the double-pipe concatenation operator. Microsoft SQL Server uses the plus sign (+) concatenation operator.

MySQL supports a similar function, *CONCAT( )*. Refer to the "Concatenation Operators" section *Chapter 3, SQL Statements Command Reference*, for more information on concatenation within Oracle, PostgreSQL, and Microsoft SQL Server.

SQL99 Syntax

```
CONCATENATE('string1' || 'string2')
```

MySQL Syntax

```
CONCAT(str1, str2, [,...n])
```

If any of the concatenation values are null, the entire returned string is null. Also, if a numeric value is concatenated, it is implicitly converted to a character string:

```
SELECT CONCAT('My ', 'bologna ', 'has ', 'a ', 'first ', 'name...');
-> 'My bologna has a first name...'
SELECT CONCAT('My ', NULL, 'has ', 'first ', 'name...');
-> NULL
```

CONVERT and TRANSLATE

The *CONVERT* function alters the representation of a character string within its character set and collation. For example, *CONVERT* might be used to alter the number of bits per character.

TRANSLATE alters the character set of a string value from one base-character set to another. Thus, *TRANSLATE* might be used to translate a value from the English character set to a Kanji (Japanese) or Russian character set. The translation must already exist, either by default or having been created using the *CREATE TRANSLATION* command.

SQL99 Syntax

```
CONVERT (char_value target_char_set USING form_of_use source_char_name)
```

```
TRANSLATE(char_value target_char_set USING translation_name)
```

Among the database vendors, only Oracle supports *CONVERT* and *TRANSLATE* with the same meaning as SQL99. Oracle's implementation of *TRANSLATE* is very similar to SQL99, but not identical. In its implementation, Oracle accepts only two arguments and allows translating only between the database character set or the national language support character set.

MySQL's implementation of the *CONVERT* function only translates numbers from one base to another. In contrast, Microsoft SQL Server's implementation of *CONVERT* is a very rich utility that alters the base datatype of an expression, but is otherwise dissimilar to the SQL99 *CONVERT* function. PostgreSQL does not support *CONVERT*, and its implementation of *TRANSLATE* serves to morph any occurrence of a character string to any other character string.

MySQL Syntax and Variations

```
CONV(int, from_base, to_base)
```

MySQL does not support *TRANSLATE*. This implementation of *CONVERT* returns a string value representing the number as it is converted from the *from_base* value to the *to_base* value. If any of the numbers are NULL, then the function returns NULL. Following are some examples:

```
SELECT CONV("a",16,2);
-> '1010'
SELECT CONV("6E",18,8);
-> '172'
SELECT CONV(-17,10,-18);
-> '-H'
```

Microsoft SQL Server Syntax and Variations

```
CONVERT (data_type[(length) | (precision,scale)], expression[,style])
```

Microsoft SQL Server does not support *TRANSLATE*. Microsoft's implementation of the *CONVERT* function does not follow the SQL99 specification. Instead, it is functionally equivalent to the *CAST* function. The *style* clause is used to define the format of a date conversion. Refer to the vendor documentation for more information. Following is an example:

```
SELECT title, CONVERT(char(7), ytd_sales)
FROM titles
ORDER BY title
GO
```

Oracle Syntax and Variations

```
CONVERT('char_value', target_char_set, source_char_set)
```

```
TRANSLATE('char_value', 'from_text', 'to_text')
```

Under Oracle's implementation, the *CONVERT* function returns the *char_value* in the target character set. The *char_value* is the string being converted, while the *target_char_set* is the name of the character set where the *char_value* is converted. *Source_char_set* is the name of the character set where the *char_value* was originally stored.

Both the target and source character set can be either literals strings, variables, or columns containing the name of the character set. Note that inadequate replacement characters might be substituted when converting from or to a character set that does not support a representation of all the characters used in the conversion.

Oracle supports several common character sets including *US7ASCII*, *WE8DECDEC*, *WE8HP*, *F7DEC*, *WE8EBCDIC500*, *WE8PC850*, and *WE8ISO8859P1*. For example:

```
SELECT CONVERT('Groß', 'US7ASCII', 'WE8HP')
FROM DUAL;
->Gross
```

PostgreSQL Syntax and Variations

```
TRANSLATE (character_string, from_text, to_text)
```

PostgreSQL does not support *CONVERT*. PostgreSQL's implementation of the *TRANSLATE* function offers a large superset of functions compared to that found in the SQL99 specification. Instead, it converts any occurrence of one text string to another within another specified string. Here is an example:

```
SELECT TRANSLATE('12345abcde', '5a', 'XX');
-> 1234XXbcde

SELECT TRANSLATE(title, 'Computer', 'PC')
FROM  titles
WHERE type = 'Personal_computer'
```

LOWER and UPPER

The functions *LOWER* and *UPPER* allow the case of a string to be altered quickly and easily, so that all the characters are lower- or uppercase, respectively. These functions are supported in all the database implementations covered in this book.

Example

```
SELECT LOWER('You Talkin To ME?'), UPPER('you talking to me?!');
-> you talking to me?, YOU TALKIN TO ME?!
```

The various database vendors also support a variety of other text formatting functions that are specific to their implementation.

SUBSTRING

The *SUBSTRING* function allows one character string to be extracted from another.

SQL99 Syntax

```
SUBSTRING(extraction_string FROM starting_position [FOR length]
[COLLATE collation_name])
```

If any of the inputs are NULL, the *SUBSTRING* function returns a NULL. The *extraction_string* is where the character value is extracted from. It may be a literal string, a column in a table with a character datatype, or a variable with a character datatype. The *starting_position* is an integer value telling the function at which position to perform the extract. The optional *length* is an integer value that tells the function how many characters to extract, starting at the *starting_position*.

MySQL Syntax and Variations

```
SUBSTRING(extraction_string FROM starting_position)
```

MySQL's implementation assumes that the characters are to be extracted from the starting position continuing to the end of the character string.

Microsoft SQL Server Syntax and Variations

```
SUBSTRING(extraction_string [FROM starting_position] [FOR length])
```

Microsoft SQL Server largely supports the SQL99 standard, except that it does not allow the *COLLATE* clause. Microsoft allows this command to be applied to text, image, and binary datatypes; however, the *starting_position* and *length* represent the number of bytes rather than the number of characters to count.

Oracle Syntax and Variations

```
SUBSTR(extraction_string, starting_position [, length])
```

Oracle's implementation, *SUBSTR*, largely functions the same way as SQL99. It does not support the *COLLATE* clause. When a *starting_value* is a negative number, Oracle counts from the end of the *extraction_string*. If *length* is omitted, the remainder of the string (starting at *starting_position*) is returned.

PostgreSQL Syntax and Variations

```
SUBSTRING(extraction_string [FROM starting_position] [FOR length])
```

PostgreSQL largely supports the SQL99 standard, except that it does not accept the *COLLATE* clause.

Examples

These examples generally work on any one of the four database vendors profiled in this book. Only the second Oracle example, with a negative starting position, fails on the others (assuming, of course, that Oracle's *SUBSTR* is translated into *SUBSTRING*):

```
/* On Oracle, counting from the left */
SELECT SUBSTR('ABCDEFG',3,4) FROM DUAL;
-> CDEF

/* On Oracle, counting from the right */
SELECT SUBSTR('ABCDEFG',-5,4) FROM DUAL;
-> CDEF

/* On MySQL */
SELECT SUBSTRING('Be vewy, vewy quiet',5);
-> 'wy, vewy quiet''

/* On PostgreSQL or SQL Server */
SELECT au_lname, SUBSTRING(au_fname, 1, 1)
FROM authors
```

```
WHERE au_lname = 'Carson'
-> Carson     C
```

TRIM

The *TRIM* function removes leading spaces, trailing characters, or both from a specified character string. This function also removes other types of characters from a specified character string. The default function is to trim the specified character from both sides of the character string. If no removal string is specified, *TRIM* removes spaces by default.

SQL99 Syntax

```
TRIM( [ [{LEADING | TRAILING | BOTH}] [removal_string] FROM ]
  target_string
  [COLLATE collation_name])
```

The *removal_string* is the character string to be stripped out. The *target _string* is the character string from which characters are to be taken. If a *removal_string* is not specified, then *TRIM* strips out spaces. The *COLLATE* clause forces the result set of the function into another preexisting collation set.

MySQL, PostgreSQL, and Oracle support the SQL99 syntax of *TRIM*.

Microsoft SQL Server (and the other vendors for that matter) provide the functions *LTRIM* and *RTRIM* to trim off leading spaces or trailing spaces, respectively. *LTRIM* and *RTRIM* cannot be used to trim other types of characters.

Examples

```
SELECT TRIM('   wamalamadingdong  ');
-> 'wamalamadingdong'

SELECT TRIM(LEADING '19' FROM '1976 AMC GREMLIN');
-> '76 AMC GREMLIN'

SELECT TRIM(BOTH 'x' FROM 'xxxWHISKEYxxx');
-> 'WHISKEY'

SELECT TRIM(TRAILING 'snack' FROM 'scooby snack');
-> 'scooby '
```

Vendor Extensions

The following section provides a full listing and description of each vendor-supported function. These functions are vendor-specific. Thus, a MySQL function, for example, is not guaranteed to be supported by any other vendor. MySQL functions are provided to give an idea of the capabilities available within the various products. Refer to the vendor's documentation for exact syntax usage.

Microsoft SQL Server-Supported Functions

Table 4-7 provides an alphabetical listing of Microsoft SQL Server-supported functions.

Table 4-7: Microsoft SQL Server-Supported Functions

Function	**Description**

abs(numeric_expression)	Returns absolute value.
acos(float_expression)	Returns angle (in radians) whose cosine is the specified argument.
app_name( )	Returns application name for current session; set by application.
ascii(character_expression)	Converts character to a numeric ASCII code.
asin(float_expression)	Returns angle (in radians) whose sine is the specified argument.
atan(float_expression)	Returns angle (in radians) whose tangent is the specified argument.
atn2(float_expression, float_ expressioin)	Returns angle (in radians) whose tangent is argument1/argument1.
avg([All| Distinct] Expression)	Computes average of a column.
binary_checksum(| expression [,...n])*	Returns binary checksum for list of expressions or row of a table.
cast(Expression as Data Type)	Converts a valid SQL Server expression to the specified datatype.
ceiling(numeric_expression)	Returns smallest integer greater than or equal to the argument.
char(integer_expression)	Converts a numeric ASCII code to a character.
charindex(expression1, expression2 [, start_location])	Returns position of the first occurrence of a substring in a string.
checksum(| expression [,...n])*	Returns checksum value (computed over row values or expressions provided).
checksum_agg([ALL | Distinct] expression)	Returns checksum of the values in group.
coalesce(expression [,...n])	Returns the first non-NULL argument from a list of arguments.
col_length(`table', `column')	Returns column length in bytes.
col_name(table_id, column_ id)	Returns column name, given table ID and column ID.
contains({column | }, `contains_search_condition'})	Searches columns on exact or "fuzzy" matches of *contains_seach_ criteria*. It is an elaborate function used to perform full-text searches. Refer to the vendor documentation for more information.
containsable(table, column, contains_search_condition)	Returns a table with exact and "fuzzy" matches of *contains_search_ condition*. It is an elaborate function used to perform full-text searches. Refer to the vendor documentation for more information.
convert(data_type [(length)], expression [, style])	Converts data from one datatype to another.
cos(float_expression)	Returns cosine.
cot(float_expression)	Returns cotangent.
*count({[All | Distinct] expression]| *})*	Counts rows.
count(*)	Computes the number of rows, including those with NULL values.

count(DISTINCT expression)	Calculates the number of distinct non-NULL values in a column or expression. Each group of rows with the same value of *expression* adds 1 to the result.
count(expression)	Returns the number of rows with non-NULL values in a certain column or expression.
count_big([All | Distinct] expression)	Same as *count* except returns big integer.
current_timestamp	Returns current date and time.
current_user	Returns username in the current database of the current session.
datalength(expression)	Returns number of bytes in a character or binary string.
databasepropertyex(database, property)	Returns database option or property.
dateadd(datepart, number, date)	Adds a number of dateparts (e.g., days) to a datetime value.
datediff(datepart, startdate, enddate)	Calculates difference between two datetime values expressed in certain dateparts.
datename(datepart, date)	Returns name of a datepart (e.g., month) of a datetime argument.
datepart(datepart, date)	Returns value of a datepart (e.g., hour) of a datetime argument.
day(date)	Returns an integer value representing the day of the date provided as a parameter.
db_id(`[database_name]')	Returns database ID and given name.
db_name(database_id)	Returns the database name.
degrees(numeric_expression)	Converts radians to degrees.
difference(character_ expression, character_ expression)	Compares how two arguments sound and returns a number from 0 to 4. Higher result indicates better phonetic match.
exp(float_expression)	Returns exponential value.
floor(numeric_expression)	Returns largest integer less than or equal to the argument.
file_id(`file_name')	Returns the file ID for the logical filename.
file_name(file_id)	Returns the logical filename for file ID.
filegroup_id (`filegroup_name')	Returns filegroup ID for the logical filegroup name.
filegroup_name (filegroup_id)	Returns the logical filegroup name for filegroup ID.
filegroupproperty (filegroup_name, property)	Returns filegroup property value for the specified property.
fileproperty (file, property)	Returns file property value for the specified property.
fulltextcatalog property(catalog_name, property)	Returns full-text catalog properties.

fulltextservice property(property)	Returns full-text service level properties.
formatmessage (msg_number, param_value [,... n])	Constructs a message from an existing message in **SYSMESSAGES** table (similar to RAISERROR).
freetexttable(table { column |}, `freetext_string' [, top_n_by_rank])*	Used for full-text search; returns a table with columns that match the meaning but don't exactly match value of *freetext_string*.
getdate( )	Returns current date and time.
getansinull([`database'])	Returns default nullability setting for new columns.
getutcdate( )	Returns Universal Time Coordinate (UTC) date.
grouping(column_name)	Returns 1 when the row is added by CUBE or ROLLUP; otherwise, returns 0.
host_id( )	Returns workstation ID of a given process.
host_name( )	Returns process hostname.
ident_incr (`table_or_view')	Returns identity-column increment value.
ident_seed (`table_or_view')	Returns identity seed value.
ident_current (`table_name')	Returns the last identity value generated for the specified table.
identity(data_type [, seed, increment]) As column_name	Used in *SELECT INTO* statement to insert an identity column into the destination table.
index_col(`table', index_id, key_id)	Returns index column name, given table ID, index ID, and column sequential number in the index key.
indexproperty(table_id, index, property)	Returns index property (such as Fillfactor).
isdate(expression)	Validates if a character string can be converted to DATETIME.
is_member({`group' | `role'})	Returns true or false (1 or 0) depending on whether user is a member of NT group or SQL Server role.
is_srvrolemember (`role' [,'login'])	Returns true or false (1 or 0) depending on whether user is a member of specified server role.
isnull(check_expression, replacement_value)	Returns the first argument if it is not NULL; otherwise, returns the second argument.
isnumeric(expression)	Validates if a character string can be converted to NUMERIC.
left(character_expression, integer_expression)	Returns a portion of a character expression, starting at *integer_ expression* from left.
len(string_expression)	Returns the number of characters in the expression.
log(float_expression)	Returns natural logarithm.
log10(float_expression)	Returns base-10 logarithm.
lower(character_expression)	Converts a string to lowercase.
ltrim(character_expression)	Trims leading-space characters.

max([All | Distinct] expression)	Finds maximum value in a column.
min([All | Distinct] expression)	Finds minimum value in a column.
month(date)	Returns month part of the date provided.
nchar(integer_expression)	Returns the unicode character with the given integer code.
newid( )	Creates a new unique identifier of type *uniqueidentifier*.
nullif(expression, expression)	Returns NULL if two specified expressions are equivalent.
object_id(`object')	Returns object ID and given name.
object_name(object_id)	Returns object name and given ID.
objectproperty (id, property)	Returns properties of objects in the current database.

MySQL-Supported Functions

Table 4-8 provides an alphabetical listing of MySQL-supported functions.

Table 4-8: MySQL-Supported Functions

Function	Description
abs(X)	Returns the absolute value of *X*.
acos(X)	Returns the arc cosine of *X*, i.e., the value whose cosine is *X*; returns NULL if *X* is not in the range -1 to 1.
ascii(str)	Returns the ASCII code value of the leftmost character of the string *str*; returns 0 if *str* is the empty string; returns NULL if *str* is NULL.
asin(X)	Returns the arc sine of *X*, i.e., the value whose sine is *X*; returns NULL if *X* is not in the range -1 to 1.
atan(X)	Returns the arctangent of *X*, i.e., the value whose tangent is *X*.
atan2(X,Y)	Returns the arctangent of the two variables *X* and *Y*.
avg(expr)	Returns the average value of *expr*.
benchmark(count,expr)	Executes the expression *expr count* times. It may be used to time how fast MySQL processes the expression. The result value is always 0.
binary	Casts the string following it to a binary string.
bin(N)	Returns a string representation of the binary value of *N*, where *N* is a long (*BIGINT*) number.
bit_count(N)	Returns the number of bits that are set in the argument *N*.
bit_and(expr)	Returns the bitwise *AND* of all bits in *expr*. The calculation is performed with 64-bit (*BIGINT*) precision.
bit_or(expr)	Returns the bitwise *OR* of all bits in *expr*. The calculation is performed with 64-bit (*BIGINT*) precision.

CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN result ...] [ELSE result] END *CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END*	The first version returns the result where *value=compare-value*. The second version returns the result for the first condition that is true. If there is no matching result value, then the result after *ELSE* is returned. If there is no *ELSE* part, NULL is returned.
ceiling(X)	Returns the smallest integer value not less than *X*.
char(N,...)	Interprets the arguments as integers and returns a string consisting of the characters given by the ASCII code values of those integers. NULL values are skipped.
coalesce(list)	Returns first non-NULL element in the list.
concat(str1,str2,...)	Returns the string that results from concatenating the arguments.
concat_ws(separator, str1, str2,...)	Stands for CONCAT With Separator and is a special form of *CONCAT( ).* The first argument is the separator for the rest of the arguments. The separator and the rest of the arguments can be a string. If the separator is NULL, the result is NULL. The function skips any NULLs and empty strings after the separator argument. The separator is added between the strings to be concatenated.
connection_id( )	Returns the connection ID (*thread_id*) for the connection. Every connection has its own unique ID.
conv(N,from_base,to_base)	Converts numbers between different number bases; returns a string representation of the number *N*, converted from base *from_base* to base *to_base* ; returns NULL if any argument is NULL.
cos(X)	Returns the cosine of *X*, where *X* is given in radians.
cot(X)	Returns the cotangent of *X*.
count(DISTINCT expr,[expr...])	Returns a count of the number of different values.
count(expr)	Returns a count of the number of non-NULL values in the rows retrieved by a *SELECT* statement.
curdate( ) *current_date*	Returns today's date as a value in `YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.
curtime( ) *current_time*	Returns the current time as a value in `HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.
database( )	Returns the current database name.

date_add(date,INTERVAL expr type)
 	
date_sub(date,INTERVAL expr type)
	These functions perform date arithmetic. *ADDDATE( )* and *SUBDATE( )* are synonyms for *DATE_ADD( )* and *DATE_SUB( ). date* is a *DATETIME* or *DATE* value specifying the starting

adddate(date,INTERVAL expr type)
 	
subdate(date,INTERVAL expr type)
	interpreted.

date_ format (date, format)	Formats the date value according to the format string.
dayname(date)	Returns the name of the weekday for date.
dayofmonth(date)	Returns the day of the month for date, in the range 1 to 31.
dayofweek(date)	Returns the weekday index for date (1 = Sunday, 2 = Monday, . . . 7 = Saturday).
dayofyear(date)	Returns the day of the year for date, in the range 1 to 366.
decode(crypt_str, pass_str)	Decrypts the encrypted string *crypt_str* using *pass_str* as the password. *crypt_str* should be a string returned from *ENCODE( ).*
degrees(X)	Returns the argument *X*, converted from radians to degrees.
elt(N,str1,str2,str3,...)	Returns *str1* if *N* = 1, *str2* if *N* = 2, and so on. Returns NULL if *N* is less than 1 or greater than the number of arguments. *ELT( )* is the complement of *FIELD( ).*
encode(str,pass_str)	Encrypts *str* using *pass_str* as the password. To decrypt the result, use *DECODE( ).* The result is a binary string the same length as the string.
encrypt(str[,salt])	Encrypts *str* using the Unix *crypt( )* system call. The *salt* argument should be a string with two characters.
exp(X)	Returns the value of *e* (the base of natural logarithms) raised to the power of *X*.
export_set (bits,on,off,[separator, [number_of_bits]])	Returns a string where every bit set in `bit' gets an `on' string and every reset bit gets an `off ' string. Each string is separated with `separator' (default `,') and only `number_of_bits' (default 64) of `bits' is used.
field(str,str1,str2,str3,...)	Returns the index of *str* in the *str1*, *str2*, *str3*, . . . list. Returns 0 if *str* is not found. *FIELD( )* is the complement of *ELT( ).*
find_in_set(str,strlist)	Returns a value 1 to *N* if the string *str* is in the list *strlist* consisting of *N* substrings. A string list is a string composed of substrings separated by `,' characters. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns NULL if either argument is NULL. This function does not work properly if the first argument contains a `,'.
floor(X)	Returns the largest integer value not greater than *X*.
format(X,D)	Formats the number *X* to a format like `#,###,###.##', rounded to *D* decimals. If *D* is 0, the result has no decimal point or fractional part.
from_days(N)	Given a daynumber *N*, returns a *DATE* value. Not intended for use with values that precede the advent of the Gregorian calendar (1582), due to the days lost when the calendar was changed.

from_unixtime(unix_ timestamp)	Returns a representation of the *unix_timestamp* argument as a value in `YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.
from_unixtime*(unix_ timestamp,format)*	Returns a string representation of the *unix_timestamp*, formatted according to the format string. Format may contain the same specifiers as those listed in the entry for the *DATE_FORMAT( )* function.
get_lock(str,timeout)	Tries to obtain a lock with a name given by the string *str*, with a timeout of *timeout* seconds. Returns 1 if the lock is obtained successfully, 0 if the attempt times out, or NULL if an error occurs.
greatest(X,Y,...)	Returns the largest (maximum-valued) argument.
hex(N)	Returns a string representation of the hexadecimal value of *N*, where *N* is a long (*BIGINT*) number. This is equivalent to *CONV (N,10,16).* Returns NULL if *N* is NULL.
interval(N,N1,N2,N3,...)	Returns 0 if *N* < *N1*, 1 if *N* < *N2*, and so on. All arguments are treated as integers. It is required that *N1* < *N2* < *N3* < . . . < *Nn* for this function to work correctly.
hour(time)	Returns the hour for time, in the range 0 to 23.
if(expr1,expr2,expr3)	If *expr1* is TRUE (*expr1* <> 0 and *expr1* <> NULL), then *IF( )* returns *expr2*, else it returns *expr3*. *IF( )* returns a numeric or string value, depending on the context in which it is used.
ifnull(expr1,expr2)	If *expr1* is not NULL, *IFNULL( )* returns *expr1*; otherwise it returns *expr2*. *IFNULL( )* returns a numeric or string value, depending on the context in which it is used.
isnull(expr)	If *expr* is NULL, *ISNULL( )* returns 1; otherwise it returns 0.
insert(str,pos,len,newstr)	Returns the string *str*. The substring begins at position *pos* and is 10 characters long, replaced by the string *newstr*.
instr(str,substr)	Returns the position of the first occurrence of substring *substr* in string *str*.
last_insert_id([expr])	Returns the last automatically generated value that was inserted into an *AUTO_INCREMENT* column.

lcase(str)
 	
lower(str)
	Returns the string *str* with all characters changed to lowercase according to the current character-set mapping (default is ISO-8859-1 Latin1).

least(X,Y,...)	With two or more arguments, returns the smallest (minimum-valued) argument.
left(str,len)	Returns the leftmost *len* characters from the string *str*.

length(str)
 	
octet_length(str)
 	
char_length(str)
 	
character_length(str)
	These functions return the length of the string *str*.

load_ file(file_name)	Reads the file and returns the file contents as a string. The file must be on the server, and the user must specify the full pathname to the file and have the file privilege.

locate(substr,str)
 	
position(substr IN str)
	Returns the position of the first occurrence of substring *substr* in string *str*. Returns 0 if *substr* is not in *str*.

locate(substr,str,pos)	Returns the position of the first occurrence of substring *substr* in string *str*, starting at position *pos*; returns 0 if *substr* is not in *str*.
log(X)	Returns the natural logarithm of *X*.
log10(X)	Returns the base-10 logarithm of *X*.
lpad(str,len,padstr)	Returns the string *str*, left-padded with the string *padstr* until *str* is 10 characters long.
ltrim(str)	Returns the string *str* with leading-space characters removed.
make_set(bits,str1,str2, . . .)	Returns a set (a string containing substrings separated by `,' characters) consisting of the strings that have the corresponding bits in bit set. *str1* corresponds to bit 0, *str2* to bit 1, etc. NULL strings in *str1, str2, . . .* are not appended to the result.
md5(string)	Calculates a MD5 *checksum* for the string. Value is returned as a 32-long hex number.

min(expr)
 	
max(expr)
	Returns the minimum or maximum value of *expr*. *MIN( )* and *MAX ( )* may take a string argument; in such cases they return the minimum or maximum string value.

minute(time)	Returns the minute for time, in the range 0 to 59.
mod(N,M)	% Modulo (like the % operator in C); returns the remainder of *N* divided by *M*.
month(date)	Returns the month for date, in the range 1 to 12.
monthname(date)	Returns the name of the month for date.

now( )
 	
sysdate( )
 	
current_timestamp
	Returns the current date and time as a value in `YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

nullif(expr1,expr2)	If *expr1 = expr2* is true, returns NULL; otherwise returns *expr1*.
oct(N)	Returns a string representation of the octal value of *N*, where *N* is a long number. This is equivalent to *CONV(N,10,8).* Returns NULL if *N* is NULL.
ord(str)	If the leftmost character of the string *str* is a multibyte character, returns the code of multibyte character by returning the ASCII code value of the character in the format of: `((first byte ASCII code)*256+(second byte ASCII code))[*256+third byte ASCII code...]` If the leftmost character is not a multibyte character, returns the same value as the *ASCII( )* function does.

password(str)	Calculates a password string from the plain-text password *str*. This is the function that is used for encrypting MySQL passwords for storage in the **Password** column of the user grant table.
period_add(P,N)	Adds *N* months to period *P* (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument *P* is not a date value.
period_diff(P1,P2)	Returns the number of months between periods *P1* and *P2*. *P1* and *P2* should be in the format YYMM or YYYYMM. Note that the period arguments *P1* and *P2* are not date values.
pi( )	Returns the value of π.

pow(X,Y)
 	
power(X,Y)
	Returns the value of *X* raised to the power of *Y*.

quarter(date)	Returns the quarter of the year for date, in the range 1 to 4.
radians(X)	Returns the argument *X*, converted from degrees to radians.

rand( )
 	
rand(N)
	Returns a random floating-point value in the range 0 to 1.0.
	
	If an integer argument *N* is specified, it is used as the seed value.

release_lock(str)	Releases the lock named by the string *str* that was obtained with *GET_LOCK( )*. Returns 1 if the lock is released, 0 if the lock isn't locked by this thread (in which case the lock is not released), and NULL if the named lock doesn't exist.
repeat(str,count)	Returns a string consisting of the string *str* repeated *count* times. If *count* <= 0, returns an empty string. Returns NULL if *str* or *count* are NULL.
replace(str, from_str,to_str)	Returns the string *str* with all occurrences of the string *from_str* replaced by the string *to_str*.
reverse(str)	Returns the string *str* with the order of the characters reversed.
right(str,ten)	Returns the rightmost 10 characters from the string *str*.
round(X)	Returns the argument *X*, rounded to an integer.
round(X,D)	Returns the argument *X*, rounded to a number with *D* decimals. If *D* is 0, the result has no decimal point or fractional part.
rpad(str,len,padstr)	Returns the string *str*, right-padded with the string *padstr* until *str* is ten characters long.
rtrim(str)	Returns the string *str* with trailing space characters removed.
sec_to_time(seconds)	Returns the seconds argument, converted to hours, minutes, and seconds, as a value in `HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.
second(time)	Returns the second for time, in the range 0 to 59.
sign(X)	Returns the sign of the argument as -1, 0, or 1, depending on whether *X* is negative, zero, or positive.
sin(X)	Returns the sine of *X*, where *X* is given in radians.

soundex(str)	Returns a *soundex* string from *str*. Two strings that sound "about the same" should have identical *soundex* strings. A "standard" *soundex* string is four characters long, but the *SOUNDEX( )* function returns an arbitrarily long string. A *SUBSTRING( )* can be used on the result to get a "standard" *soundex* string. All non-alphanumeric characters are ignored in the given string. All international alphabetic characters outside the A-Z range are treated as vowels.
space(N)	Returns a string consisting of *N* space characters.
sqrt(X)	Returns the nonnegative square root of *X*.

std(expr)
 	
stddev(expr)
	Returns the standard deviation of *expr*. The *STDDEV( )* form of this function is provided for Oracle compatability.

strcmp(expr1,expr2)	*STRCMP( )* returns 0 if the strings are the same, -1 if the first argument is smaller than the second according to the current sort order, and 1 otherwise.

substring(str,pos,len)
 	
substring(str FROM pos FOR len)
 	
mid(str,pos,len)
	Returns a substring 10 characters long from string *str*, starting at position *pos*. The variant form that uses *FROM* is ANSI SQL92 syntax.

substring_index (str,delim,count)	Returns the substring from string *str* after *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned.

substring(str,pos)
 	
substring(str FROM pos)
	Returns a substring from string *str* starting at position *pos*.

sum(expr)	Returns the sum of *expr*. Note that if the return set has no rows, it returns NULL.
tan(X).	Returns the tangent of *X*, where *X* is given in radians.
time_ format (time, format)	This is used like *DATE_FORMAT( )*, but the format string may contain only those format specifiers that handle hours, minutes, and seconds. Other specifiers produce a NULL value or 0.
time_to_sec(time)	Returns the time argument, converted to seconds.
to_days(date)	Given a date, returns a daynumber (the number of days since year 0) .
trim([[BOTH | LEADING | TRAILING] [remstr] FROM] str)	Returns the string *str* with all *remstr* prefixes and/or suffixes removed. If none of the specifiers *BOTH*, *LEADING*, or *TRAILING* are given, *BOTH* is assumed. If *remstr* is not specified, spaces are removed.
truncate(X,D)	Returns the number *X*, truncated to *D* decimals. If *D* is 0, the result has no decimal point or fractional part.

ucase(str)
 	
upper(str)
	Returns the string *str* with all characters changed to uppercase according to the current character set mapping (default is ISO-8859-1 Latin1).

unix_timestamp( ) *unix_timestamp(date)*	If called with no argument, returns a Unix timestamp (seconds since `1970-01-01 00:00:00' GMT). If *UNIX_TIMESTAMP( )* is called with a date argument, it returns the value of the argument as seconds since `1970-01-01 00:00:00' GMT.

user( )
 	
system_user( )
 	
session_user( )
	These functions return the current MySQL username.

version( )	Returns a string indicating the MySQL server version.

week(date)
 	
week(date, first)
	With a single argument, returns the week for date, in the range 0 to 53. (The beginning of a week 53 is possible during some years.) The two-argument form of WEEK( ) allows the user to specify whether the week starts on Sunday (0) or Monday (1).

weekday(date)	Returns the weekday index for date (0 = Monday, 1 = Tuesday, . . . 6 = Sunday).
year(date)	Returns the year for date, in the range 1000 to 9999.

yearweek(date)
 	
yearweek(date, first)
	Returns year and week for a date. The second argument works exactly like the second argument to *WEEK( )*. Note that the year may be different from the year in the date argument for the first and the last week of the year.

Oracle SQL-Supported Functions

Table 4-9 provides an alphabetical listing of the SQL functions specific to Oracle.

Table 4-9: Oracle-Supported Functions

Function	Description
abs(number)	Returns the absolute value of *number*.
acos(number)	Returns the arc cosine of *number* ranging from -1 to 1. The result ranges from 0 to π and is expressed in radians.
add_months(date, int)	Returns the date *date* plus *int* months.
ascii(string)	Returns the decimal value in the database character set of the first character of *string*; returns an ASCII value when the database character set is 7-bit ASCII; returns EBCDIC values if the database character set is EBCDIC Code Page 500.
asin(number)	Returns the arc sine of *number* ranging from -1 to 1. The resulting value ranges from -π/2 to π /2 and is expressed in radians.
atan(number)	Returns the arctangent of any *number*. The resulting value ranges from -π /2 to p/2 and is expressed in radians.

atan2(number,nbr)	Returns the arctangent of *number* and *nbr*. The values for *number* and *nbr* are not restricted, but the results range from -π to π and are expressed in radians.
avg([DISTINCT] expression) over (analytics)	Returns the average value of *expr*. It can be used as an aggregate or analytic function (analytic functions are beyond the scope of this text).
bfilename (`directory','filename')	Returns a *BFILE* locator associated with a physical LOB binary *filename* on the server's filesystem in *directory*.
ceil(number)	Returns smallest integer greater than or equal to *number*.
chartorowid(char)	Converts a value from a character datatype (*CHAR* or *VARCHAR2* datatype) to *ROWID* datatype.
chr(number [USING NCHAR_ CS])	Returns the character having the binary equivalent to *number* in either the database character set (if *USING NCHAR_CS* is not included) or the national character set (if *USING NCHAR_CS* is included).
concat(string1, string2)	Returns *string1* concatenated with *string2*. It is equivalent to the concatenation operator (||).
convert(char_value, target_ char_set, source_char_set)	Converts a character string from one character set to another; returns the *char_value* in the *target_char_set* after converting *char_value* from the *source_char_set.*
corr(expression1, expression2) over (analytics)	Returns the correlation coefficient of a set of numbered pairs (*expressions* 1 and 2). It can be used as an aggregate or analytic function (analytic functions are beyond the scope of this text).
cos(number)	Returns the cosine of *number* as an angle expressed in radians.
cosh(number)	Returns the hyperbolic cosine of *number*.
count	Returns the number of rows in the query; refer to the earlier section on *COUNT* for more information.
covar_pop(expression1, expression2) over (analytics)	Returns the population covariance of a set of number pairs (*expressions* 1 and 2). It can be used as an aggregate or analytic function (analytic functions are beyond the scope of this text).
covar_samp(expression1, expression2) over(analytics)	Returns the sample covariance of a set of number pairs (*expressions* 1 and 2). It can be used as an aggregate or analytic function (analytic functions are beyond the scope of this text).
cume_dist( ) ([OVER (query)] ORDER BY...)	The cumulative distribution function computes the relative position of a specified value in a group of values.
decode(expr search , result [,. n] [,default])	Compares *expr* to the search value; if *expr* is equal to a search, returns the result. Without a match, *DECODE* returns default, or *NULL* if default is omitted. Refer to Oracle documentation for more details.
dense_rank( ) ([OVER (query)] ORDER BY...)	Computes the rank of each row returned from a query with respect to the other rows, based on the values of the *value_exprs* in the *ORDER_BY_clause.*
deref(expression)	Returns the object reference of *expression*, where *expression* must return a *REF* to an object.

dump(expression [,return_ format [, starting_at [,length]]])	Returns a *VARCHAR2* value containing a datatype code, length in bytes, and internal representation of *expression*. The resulting value is returned in the format of *return_ format*.
empth[B | C]lob( )	Returns an empty LOB locator that can be used to initialize a LOB variable. It can also be used to initialize a LOB column or attribute to empty in an *INSERT* or *UPDATE* statement.
exp(number)	Returns *E* raised to the *number* ed power, where E = 2.71828183.
first_value(expression) over (analytics)	Returns the first value in an ordered set of values.
floor(number)	Returns largest integer equal to or less than *number*.
greatest(expression [,...n])	Returns the greatest of the list of *expressions*. All *expressions* after the first are implicitly converted to the datatype of the first *expression* before the comparison.
grouping(expression)	Distinguishes null cause by a super-aggregation in *GROUP BY* extension from an actual null value.
hextoraw(string)	Converts *string* containing hexadecimal digits into a raw value.
initcap(string)	Returns *string*, with the first letter of each word in uppercase and all other letters in lowercase.
instr(string1, string2, start_at, occurrence)	Searches one character string for another character string. *INSRT* search *char1* with a starting position of *start_at* (an integer) looking for the numeric *occurrence* within *string2*. Returns the position of the character in *string1* that is the first character of this occurrence.
instrb(string1, string2, [start_a [t, occurrence]])	The same as *INSTR*, except that *start_at* and the return value are expressed in bytes instead of characters.
lag(expression [,offset] [,default]) over(analytics)	Provides access to more than one row of a table at the same time without a self join; refer to the vendor documentation for more information.
last_day(date)	Returns the date of the last day of the month that contains *date*.
last_value(expression) over (analytics)	Returns the last value in an ordered set of values; refer to the vendor documentation for more information.
lead(expression [,offset] [,default]) over(analytics)	Provides access to more than one row of a table at the same time without a self join. Analytic functions are beyond the scope of this text.
least(expression [,...n])	Returns the least of the list of *expressions*.
length(string)	Returns the integer length of *string*, or null if *string* is null.
lengthb(string)	Returns the length of *char* in bytes; otherwise, the same as *LENGTH.*
ln(number)	Returns the natural logarithm of *number*, where the *number* is greater than 0.
log(base_number, number)	Returns the logarithm of any *base_number* of *number*.

lower(string)	Returns *string* in the same datatype as it was supplied with all characters lowercase.
lpad(string1, number [,string2])	Returns *string1*, left-padded to length *number* using characters in *string2*; *string2* defaults to a single blank.
ltrim(string[, set])	Removes all characters in *set* from the left of *string. Set* defaults to a single blank.
make_ref({table_name | view_ name} , key [,...n])	Creates a reference (*REF*) to a row of an object view or a row in an object table whose object identifier is primary key-based.
max([DISTINCT] expression) over (analytics)	Returns maximum value of *expression*. It can be used as an aggregate or analytic function (analytic functions are beyond the scope of this text).
min([DISTINCT] expression) over (analytics)	Returns minimum value of *expression*. It can be used as an aggregate or analytic function (analytic functions are beyond the scope of this text).
mod(dividend, divider)	Returns remainder of *dividend* divided by *divider* ; returns the *dividend* if *divider* is 0.
months_between (date1, date2)	Returns number of months between dates *date1* and *date2*. When *date1* is later than *date2*, the result is positive. If it is earlier, the result is negative.
new_time(date, time_zone1, time_zone2)	Returns the date and time in *time_zone2* when date and time in *time_zone1* are *date*. *Time_zones* 1 and 2 may be any of these text strings: • AST, ADT: Atlantic Standard or Daylight Time • BST, BDT: Bering Standard or Daylight Time • CST, CDT: Central Standard or Daylight Time • EST, EDT: Eastern Standard or Daylight Time • GMT: Greenwich Mean Time • HST, HDT: Alaska-Hawaii Standard Time or Daylight Time • MST, MDT: Mountain Standard or Daylight Time • NST: Newfoundland Standard Time • PST, PDT: Pacific Standard or Daylight Time • YST, YDT: Yukon Standard or Daylight Time
next_day(date, string)	Returns the date of the first weekday named by *string* that is later than *date*. The argument *string* must be either the full name or the abbreviation of a day of the week in the date language of the session.

nls_charset_decl_len(bytecnt, csid)	Returns the declaration width (*bytecnt*) of an *NCHAR* column using the character set ID (*csid*) of the column.
nls_charset_id(text)	Returns the NLS character set ID number corresponding to *text.*
nls_charset_name(number)	Returns the *VARCHAR2* name for the NLS character set corresponding to the ID *number.*
nls_initcap(string [,'nlsparameter'])	Returns *string* with the first letter of each word in uppercase and all other letters in lowercase. The *nlsparameter* offers special linguistic sorting features.
nls_lower(string, [,'nlsparameter'])	Returns *string* with all letters lowercase. The *nlsparameter* offers special linguistic sorting features.
nlssort(string [,'nlsparameter'])	Returns the string of bytes used to sort *string.* The *nlsparameter* offers special linguistic sorting features.
nls_upper string [,'nlsparameter'])	Returns *string* with all letters uppercase. The *nlsparameter* offers special linguistic sorting features.
ntile(expression) over (query_partition ORDER BY...)	Divides an ordered data set into a number of buckets numbered 1 to *expression* and assigns the appropriate bucket number to each row.
numtodsinterval (number, `string')	Converts *number* to an *INTERVAL DAY TO SECOND* literal, where *number* is a number or an expression resolving to a number, such as a numeric datatype column.
numtoyminterval (number, `string')	Converts *number* to an *INTERVAL DAY TO MONTH* literal, where *number* is a number or an expression resolving to a number, such as a numeric datatype column.
nvl(expression1, expression2)	If *expression1* is null, *expression2* is returned in the place of a null value. Otherwise, *expression1* is returned. The expressions may be any datatype.
nvl2(expression1, expression2, expression3)	Similar to *NLV,* except that if *expression1* is not null, *expression2* is returned. If *expression1* is null, *expression3* is returned. The expressions may be any datatype, except *LONG*.
percent_rank( ) over (query_partition ORDER BY...)	Similar to the *CUME_DIST* analytical function. Rather than return the cumulative distribution, it returns the percentage rank of a row compared to the others in its result set. Refer to the vendor documentation for more assistance.
power(number, power)	Returns *number* raised to the nth *power*. The base and the exponent can be any numbers, but if *number* is negative, *power* must be an integer.
rank (value_expression) over (query_partition ORDER BY ...)	Computes the rank of each row returned from a query with respect to the other rows returned by the query, based on the values of the *value_expression* in the *ORDER_BY_clause*.
ratio_to_report (value_exprs) over (query_partition)	Computes the ratio of a value to the sum of a set of values. If *values_expr* is null, the ratio-to-report value also is null.
rawtohex(raw)	Converts a *raw* value to a string (character datatype) of its hexadecimal equivalent.

ref(table_alias)	*REF* takes a table alias associated with a row from a table or view. A special reference value is returned for the object instance that is bound to the variable or row.
reftohex(expression)	Converts argument *expression* to a character value containing its hexadecimal equivalent.
regr_ xxx(expression1, expression2) over (analytics)	Linear regression functions fit an ordinary-least-squares regression line to a set of number pairs where *expression1* is the dependent variable and *expression2* is the independent variable. The linear regression functions are: • REGR_SLOPE: returns the slope of the line • REGR_INTERCEPT: returns the y-intercept of the regression line • REGR_COUNT: returns the number of non-null pairs fitting the regression line • REGR_R2: returns the coefficient of determination for the regression • REGR_AVGX: returns the average of the independent variable • REGR_AVGY: returns the average of the dependent variable • REGR_SXX: calculates *REGR_COUNT(exp1, exp2) * VAR_POP(exp2)* • REGR_SYY: calculates *REGR_COUNT(exp1, exp2) * VAR_POP(exp1)* • REGR_SXY: calculates *REGR_COUNT(exp1, exp2) * COVAR_POP(exp1, exp2)* These can be used as aggregate or analytic functions.
replace(string, search_string [,replacement_string])	Returns *string* with every occurrence of *search_string* replaced with *replacement_string.*
round (number, decimal)	Returns *number* rounded to *decimal* places right of the decimal point. When *decimal* is omitted, *number* is rounded to 0 places. Note that *decimal*, an integer, can be negative to round off digits left of the decimal point.
round (date[, format])	Returns the *date* rounded to the unit specified by the format model *format*. When *format* is omitted, *date* is rounded to the nearest day.
row_number ( ) over (query_partition ORDER BY ...)	Assigns a unique number to each row where it is applied in the ordered sequence of rows specified by the *ORDER_BY_clause*, beginning with 1.

rowidtochar(rowid)	Converts a *rowid* value to *VARCHAR2* datatype, 18 characters long.
rpad(string1, number [, string2])	Returns *string1*, right-padded to length *number* with the value of *string2*, repeated as needed. *String2* defaults to a single blank.
rtrim(string[,set])	Returns *string*, with all the rightmost characters that appear in *set* removed; *set* defaults to a single blank.
sign(number)	When *number* < 0, returns -1. When *number* = 0, returns 0. When *number* > 0, returns 1.
sin(number)	Returns the sine of *number* as an angle expressed in radians.
sinh(number)	Returns the hyperbolic sine of *number*.
soundex(string)	Returns a character string containing the phonetic representation of *string*. This function allows words that are spelled differently but sound alike in English to be compared for equality.
sqrt(number)	Returns square root of *number*, a nonnegative number.
stddev([DISTINCT] expression) over (analytics)	Returns sample standard deviation of a set of numbers shown as *expression.*
stdev_pop(expression) over (analytics)	Computes the population standard deviation and returns the square root of the population variance.
seddev_samp(expression) over (analytics)	Computes the cumulative sample standard deviation and returns the square root of the sample variance.
substr(extraction_string [FROM starting_position] [FOR length])	Refer to the earlier section on *SUBSTR*.
substrb(extraction_string [FROM starting_position] [FOR length])	*SUBSTRB* is the same as *SUBSTR*, except that the arguments *m starting_position* and *length* are expressed in bytes, rather than in characters.
sum([DISTINCT] expression) over (analytics)	Returns sum of values of *expr* ; refer to vendor documentation for assistance with analytics and the *OVER* subclause.
sys_context (`namespace','attribute' [,length])	Returns the value of *attribute* associated with the context *namespace*, usable in both SQL and PL/SQL statements.
sys_guid( )	Generates and returns a globally unique identifier（*RAW* value) made up of 16 bytes.
sysdate	Returns the current date and time, requiring no arguments.
tan(number)	Returns the tangent of *number* as an angle expressed in radians.
tanh(number)	Returns the hyperbolic tangent of *number*
to_char (date [, format [, `nls_parameter']])	Converts *date* to a *VARCHAR2* in the format specified by the date format *format*. When *fmt* is omitted, *date* is converted to the default date format. The *nls_parameter* option offers additional control over formatting options.
	Converts *number* to a *VARCHAR2* in the format specified by the number format *format*. When *fmt* is omitted, *number* is converted

	to a string long enough to hold the *number*. The *nls_parameter* option offers additional control over formatting options.
to_date(string [, format [, `nls_parameter']])	Converts *string* (in *CHAR* or *VARCHAR2)* to a *DATE* datatype. The *nls_parameter* option offers additional control over formatting options.
to_lob(long_column)	Usable only by *LONG* or *LONG RAW* expressions, it converts *LONG* or *LONG RAW* values in the column *long_column* to LOB values. It is usable only in the *SELECT* list of a subquery in an *INSERT* statement.
to_multi_byte(string)	Returns *string* with all of its single-byte characters converted to their corresponding multi-byte characters.
to_number(string [, format [,'nls_parameter']])	Converts a numeric *string* (of *CHAR* or *VARCHAR2* datatype) to a value of a NUMBER datatype in the format specified by the optional format model *format*. The *nls_parameter* option offers additional control over formatting options.
to_single_byte(string)	Returns *string* with all of its multi-byte characters converted to their corresponding single-byte characters.
translate(`char_value', `from_text', `to_text')	Returns *char_value* with all occurrences of each character in *from_text* replaced by its corresponding character in *to_text*; refer to the section "CONVERT and TRANSLATE" earlier in this chapter for more information on *TRANSLATE*.
translate (text USING [CHAR_CS | NCHAR_CS])	Converts *text* into the character set specified for conversions between the database character set or the national character set.

*trim({[LEADING | TRAILING | BOTH] trim_char | trim_char }
 	
FROM trim_source})*
	Enables leading or trailing characters (or both) to be trimmed from a character string.

trunc (base [, number])	Returns *base* truncated to *number* decimal places. When *number* is omitted, *base* is truncated to 0 places. *Number* can be negative to truncate (make zero) *number* digits left of the decimal point.
trunc (date [, format])	Returns *date* with any time data truncated to the unit specified by *format*. When *format* is omitted, *date* is truncated to the nearest whole day.
uid	Returns an integer that uniquely identifies the session user who logged on. No parameters are needed.
upper(string)	Returns *string* with all letters in uppercase.
user	Returns the name of the session user who logged on in *VARCHAR2.*
userenv(option)	Returns information about the current session in *VARCHAR2.*
value(table_alias)	Takes as a table alias associated with a row in an object table and returns object instances stored within the object table.
var_pop(expression) over (analytics)	Returns the population variance of a set of numbers after discarding the nulls in the *expression* number set. Analytic functions are covered in the vendor documentation.

var_samp(expression) over (analytics)	Returns the sample variance of a set of numbers after discarding the nulls in the *expression* number set. Analytic functions are covered in the vendor documentation.
variance([DISTINCT] expression) over (analytics)	Returns variance of *expression* calculated as follows: • 0 if the number of rows in *expression* = 1 • *VAR_SAMP* if the number of rows in *expression* > 1
vsize(expression)	Returns the number of bytes in the internal representation of *expression*. When *expression* is null, it returns null.

PostgreSQL-Supported Functions

Table 4-10 lists the functions specific to PostgreSQL.

Table 4-10: PostgreSQL-Supported Functions

Function	Description
abstime(timestamp)	Converts to abstime
abs(float8)	Returns absolute value
acos(float8)	Returns arccosine
age(timestamp)	Preserves months and years
age(timestamp, timestamp)	Preserves months and years
area(object)	Returns area of item
asin(float8)	Returns arcsine
atan(float8)	Returns arctangent
atan2(float8,float8)	Returns arctangent
box(box,box)	Returns intersection box
box(circle)	Converts circle to box
box(point,point)	Returns points to box
box(polygon)	Converts polygon to box
broadcast(cidr)	Constructs broadcast address as text
broadcast(inet)	Constructs broadcast address as text
CASE WHEN expr THEN expr [...] ELSE expr END	Returns expression for first true WHEN clause
cbrt(float8)	Returns cube root
center(object)	Returns center of item
char(text)	Converts text to char type
char(varchar)	Converts varchar to char type
char_length(string)	Returns length of string
character_length(string)	Returns length of string

circle(box)	Converts to circle
circle(point,float8)	Converts point to circle
COALESCE(list)	Returns first non-NULL value in list
cos(float8)	Returns cosine
cot(float8)	Returns cotangent
date_part(text,timestamp)	Returns portion of date
date_part(text,interval)	Returns portion of time
date_trunc(text,timestamp)	Truncates date
degrees (float8)	Converts radians to degrees
diameter(circle)	Returns diameter of circle
exp(float8)	Raises e to the specified exponent
float(int)	Converts integer to floating point
float4(int)	Converts integer to floating point
height(box)	Returns vertical size of box
host(inet)	Extracts host address as text
initcap(text)	Converts first letter of each word to uppercase
interval(reltime)	Converts to interval
integer(float)	Converts floating point to integer
isclosed(path)	Returns a closed path
isopen(path)	Returns an open path
isfinite(timestamp)	Returns a finite time
isfinite(interval)	Returns a finite time
length(object)	Returns length of item
ln(float8)	Returns natural logarithm
log(float8)	Returns base-10 logarithm
lower(string)	Converts string to lowercase
lseg(box)	Converts box diagonal to lseg
lseg(point,point)	Converts points to lseg
lpad(text,int,text)	Returns left-pad string to specified length
ltrim(text,text)	Returns left-trim characters from text
masklen(cidr)	Calculates netmask length
masklen(inet)	Calculates netmask length
netmask(inet)	Constructs netmask as text
npoint(path)	Returns number of points
NULLIF(input,value)	Returns NULL if input = value, else returns input
octet_length(string)	Returns storage length of string
path(polygon)	Converts polygon to path

pclose(path)	Converts path to closed
pi( )	Returns fundamental constant
polygon(box)	Returns 12-point polygon
polygon(circle)	Returns 12-point polygon
polygon(npts,circle)	Returns npts polygon
polygon(path)	Converts path to polygon
point(circle)	Returns center
point(lseg,lseg)	Returns intersection
point(polygon)	Returns center
position(string in string)	Returns location of specified substring
pow (float8,float8)	Raises a number to the specified exponent
popen(path)	Converts path to open path
reltime(interval)	Converts to reltime
radians(float8)	Converts degrees to radians
radius(circle)	Returns radius of circle
round(float8)	Rounds to nearest integer
rpad(text,int,text)	Converts right pad string to specified length
rtrim(text,text)	Converts right trim characters from text
sin(float8)	Returns sine
sqrt(float8)	Returns square root
substring(string [from int] [for int])	Extracts specified substring
substr(text,int[,int])	Extracts specified substring
tan(float8)	Returns tangent
text(char)	Converts char to text type
text(varchar)	Converts varchar to text type
textpos(text,text)	Locates specified substring
timestamp(date)	Converts to timestamp
timestamp(date,time)	Converts to timestamp
to_char(timestamp, text)	Converts timestamp to string
to_char(int, text)	Converts int4/int8 to string
to_char(float, text)	Converts float4/float8 to string
to_char(numeric, text)	Converts numeric to string
to_date(text, text)	Converts string to date
to_number(text, text)	Converts string to numeric
to_timestamp(text, text)	Converts string to timestamp
translate(text,from,to)	Converts character in string
trim([leading|trailing| both] [string] from string)	Trims characters from string

trunc(float8)	Truncates (towards zero)
upper(text)	Converts text to uppercase
varchar(char)	Converts char to varchar type
varchar(text)	Converts text to varchar type
width(box)	Returns horizontal size

Back to: SQL in a Nutshell