CHAPTER

44

SQL Useful Functions

SQL has many built-in functions for performing processing on string or numeric data. Following is the list of all useful SQL built-in functions:

- SQL COUNT Function - The SQL COUNT aggregate function is used to count the number of rows in a database table.
- SQL MAX Function - The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
- SQL MIN Function - The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- SQL AVG Function - The SQL AVG aggregate function selects the average value for certain table column.
- SQL SUM Function - The SQL SUM aggregate function allows selecting the total for a numeric column.
- SQL SQRT Functions - This is used to generate a square root of a given number.
- SQL RAND Function - This is used to generate a random number using SQL command.
- SQL CONCAT Function - This is used to concatenate any string inside any SQL command.
- SQL Numeric Functions - Complete list of SQL functions required to manipulate numbers in SQL.
- SQL String Functions - Complete list of SQL functions required to manipulate strings in SQL.

SQL COUNT Function

SQL **COUNT** function is the simplest function and very useful in counting the number of records, which are expected to be returned by a SELECT statement.

To understand **COUNT** function, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+--------+--------+---------------+--------------------------+
| id     | name | work_date   | daily_typing_pages |
+--------+--------+---------------+--------------------------+
|     1 | John | 2007-01-24 |                250 |
|     2 | Ram  | 2007-05-27 |                220 |
|     3 | Jack | 2007-05-06 |                170 |
|     3 | Jack | 2007-04-06 |                100 |
|     4 | Jill | 2007-04-06 |                220 |
|     5 | Zara | 2007-06-06 |                300 |
|     5 | Zara | 2007-02-06 |                350 |
+--------+--------+---------------+--------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to count total number of rows in this table, then you can do it as follows:

```
SQL>SELECT COUNT(*) FROM employee_tbl ;
```

```
+-------------+
| COUNT(*) |
+-------------+
|         7 |
+-------------+
1 row in set (0.01 sec)
```

Similarly, if you want to count the number of records for Zara, then it can be done as follows:

```
SQL>SELECT COUNT(*) FROM employee_tbl
    -> WHERE name="Zara";
+-------------+
| COUNT(*) |
+-------------+
|         2 |
+-------------+
1 row in set (0.04 sec)
```

NOTE: All the SQL queries are case insensitive, so it does not make any difference if you give ZARA or Zara in WHERE CONDITION.

SQL MAX Function

SQL **MAX** function is used to find out the record with maximum value among a record set.

To understand **MAX** function, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+--------+--------+----------------+--------------------------+
| id    | name | work_date   | daily_typing_pages |
+--------+--------+----------------+--------------------------+
|     1 | John | 2007-01-24 |                250 |
|     2 | Ram  | 2007-05-27 |                220 |
|     3 | Jack | 2007-05-06 |                170 |
|     3 | Jack | 2007-04-06 |                100 |
|     4 | Jill | 2007-04-06 |                220 |
|     5 | Zara | 2007-06-06 |                300 |
|     5 | Zara | 2007-02-06 |                350 |
+--------+--------+----------------+--------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to fetch maximum value of daily_typing_pages, then you can do so simply using the following command:

```
SQL> SELECT MAX(daily_typing_pages)
    -> FROM employee_tbl;
+----------------------------------+
| MAX(daily_typing_pages) |
+----------------------------------+
|                      350 |
+----------------------------------+
1 row in set (0.00 sec)
```

You can find all the records with maxmimum value for each name using **GROUP BY** clause as follows:

```
SQL> SELECT id, name, MAX(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+--------+--------+----------------------------------+
| id    | name | MAX(daily_typing_pages) |
+--------+--------+----------------------------------+
|     3 | Jack |                      170 |
|     4 | Jill |                      220 |
```

```
|    1 | John |                     250 |
|    2 | Ram  |                     220 |
|    5 | Zara |                     350 |
+------+------+---------------------------------+
5 rows in set (0.00 sec)
```

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

```
SQL> SELECT MIN(daily_typing_pages) least, MAX(daily_typing_pages) max
    -> FROM employee_tbl;
+---------+--------+
| least | max  |
+---------+--------+
|   100 |  350 |
+---------+--------+
1 row in set (0.01 sec)
```

SQL MIN Function

SQL **MIN** function is used to find out the record with minimum value among a record set.

To understand **MIN** function, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+------+------+------------+-------------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------------+
|    1 | John | 2007-01-24 |                 250 |
|    2 | Ram  | 2007-05-27 |                 220 |
|    3 | Jack | 2007-05-06 |                 170 |
|    3 | Jack | 2007-04-06 |                 100 |
|    4 | Jill | 2007-04-06 |                 220 |
|    5 | Zara | 2007-06-06 |                 300 |
|    5 | Zara | 2007-02-06 |                 350 |
+------+------+------------+-------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to fetch minimum value of daily_typing_pages, then you can do so simply using the following command:

```
SQL> SELECT MIN(daily_typing_pages)
    -> FROM employee_tbl;
+---------------------------------+
| MIN(daily_typing_pages) |
+---------------------------------+
|                     100 |
+---------------------------------+
1 row in set (0.00 sec)
```

You can find all the records with minimum value for each name using **GROUP BY** clause as follows:

```
SQL> SELECT id, name, work_date, MIN(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+------+---------------------------------+
| id   | name | MIN(daily_typing_pages) |
+------+------+---------------------------------+
|    3 | Jack |                     100 |
|    4 | Jill |                     220 |
|    1 | John |                     250 |
```

```
|    2 | Ram  |                     220 |
|    5 | Zara |                     300 |
+------+------+---------------------------------+
5 rows in set (0.00 sec)
```

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

```
SQL> SELECT MIN(daily_typing_pages) least,
    -> MAX(daily_typing_pages) max
    -> FROM employee_tbl;
+---------+--------+
| least | max  |
+---------+--------+
|   100 |  350 |
+---------+--------+
1 row in set (0.01 sec)
```

SQL AVG Function

SQL **AVG** function is used to find out the average of a field in various records.

To understand **AVG** function, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+------+------+------------+-------------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+-------------------------+
|    1 | John | 2007-01-24 |                 250 |
|    2 | Ram  | 2007-05-27 |                 220 |
|    3 | Jack | 2007-05-06 |                 170 |
|    3 | Jack | 2007-04-06 |                 100 |
|    4 | Jill | 2007-04-06 |                 220 |
|    5 | Zara | 2007-06-06 |                 300 |
|    5 | Zara | 2007-02-06 |                 350 |
+------+------+------------+-------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to calculate average of all the dialy_typing_pages, then you can do so by using the following command:

```
SQL> SELECT AVG(daily_typing_pages)
    -> FROM employee_tbl;
+---------------------------------+
| AVG(daily_typing_pages) |
+---------------------------------+
|                 230.0000 |
+---------------------------------+
1 row in set (0.03 sec)
```

You can take average of various records set using **GROUP BY** clause. Following example will take average all the records related to a single person and you will have average typed pages by every person.

```
SQL> SELECT name, AVG(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+---------------------------------+
| name | AVG(daily_typing_pages) |
+------+---------------------------------+
| Jack |                135.0000 |
| Jill |                220.0000 |
```

```
| John |                250.0000 |
| Ram  |                220.0000 |
| Zara |                325.0000 |
+--------+--------------------------------+
5 rows in set (0.20 sec)
```

SQL SUM Function

SQL **SUM** function is used to find out the sum of a field in various records.

To understand **SUM** function, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+--------+--------+---------------+--------------------------+
| id     | name | work_date   | daily_typing_pages |
+--------+--------+---------------+--------------------------+
|     1 | John | 2007-01-24 |                250 |
|     2 | Ram  | 2007-05-27 |                220 |
|     3 | Jack | 2007-05-06 |                170 |
|     3 | Jack | 2007-04-06 |                100 |
|     4 | Jill | 2007-04-06 |                220 |
|     5 | Zara | 2007-06-06 |                300 |
|     5 | Zara | 2007-02-06 |                350 |
+--------+--------+---------------+--------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to calculate total of all the dialy_typing_pages, then you can do so by using the following command:

```
SQL> SELECT SUM(daily_typing_pages)
    -> FROM employee_tbl;
+---------------------------------+
| SUM(daily_typing_pages) |
+---------------------------------+
|                    1610 |
+---------------------------------+
1 row in set (0.00 sec)
```

You can take sum of various records set using **GROUP BY** clause. Following example will sum up all the records related to a single person and you will have total typed pages by every person.

```
SQL> SELECT name, SUM(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+--------+---------------------------------+
| name | SUM(daily_typing_pages) |
+--------+---------------------------------+
| Jack |                      270 |
| Jill |                      220 |
| John |                      250 |
| Ram  |                      220 |
| Zara |                      650 |
+--------+---------------------------------+
5 rows in set (0.17 sec)
```

SQL SQRT Function

SQL **SQRT** function is used to find out the square root of any number. You can Use SELECT statement to find out squre root of any number as follows:

```
SQL>  select SQRT(16);
+-------------+
| SQRT(16)  |
+-------------+
| 4.000000  |
+-------------+
1 row in set (0.00 sec)
```

You are seeing float value here because internally SQL will manipulate square root in float data type.

You can use SQRT function to find out square root of various records as well. To understand **SQRT**function in more detail, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+--------+--------+---------------+--------------------------+
| id     | name | work_date   | daily_typing_pages |
+--------+--------+---------------+--------------------------+
|     1 | John | 2007-01-24 |                250 |
|     2 | Ram  | 2007-05-27 |                220 |
|     3 | Jack | 2007-05-06 |                170 |
|     3 | Jack | 2007-04-06 |                100 |
|     4 | Jill | 2007-04-06 |                220 |
|     5 | Zara | 2007-06-06 |                300 |
|     5 | Zara | 2007-02-06 |                350 |
+--------+--------+---------------+--------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to calculate square root of all the dialy_typing_pages, then you can do so by using the following command:

```
SQL> SELECT name, SQRT(daily_typing_pages)
    -> FROM employee_tbl;
+--------+-----------------------------------+
| name | SQRT(daily_typing_pages) |
+--------+-----------------------------------+
| John |                15.811388 |
| Ram  |                14.832397 |
| Jack |                13.038405 |
| Jack |                10.000000 |
| Jill |                14.832397 |
| Zara |                17.320508 |
| Zara |                18.708287 |
+--------+-----------------------------------+
7 rows in set (0.00 sec)
```

SQL RAND Function

SQL has a **RAND** function that can be invoked to produce random numbers between 0 and 1:

```
SQL>  SELECT RAND( ), RAND( ), RAND( );
+-----------------------+----------------------+-----------------------+
| RAND( )           | RAND( )          | RAND( )           |
+-----------------------+----------------------+-----------------------+
| 0.45464584925645 | 0.1824410643265 | 0.54826780459682 |
+-----------------------+----------------------+-----------------------+
1 row in set (0.00 sec)
```

When invoked with an integer argument, RAND( ) uses that value to seed the random number generator. Each time you seed the generator with a given value, RAND( ) will produce a repeatable series of numbers:

```
SQL>  SELECT RAND(1), RAND( ), RAND( );
+-----------------------+-----------------------+-----------------------+
| RAND(1 )              | RAND( )               | RAND( )               |
+-----------------------+-----------------------+-----------------------+
| 0.18109050223705 | 0.75023211143001 | 0.20788908117254 |
+-----------------------+-----------------------+-----------------------+
1 row in set (0.00 sec)
```

You can use **ORDER BY RAND()** to randomize a set of rows or values as follows:

To understand **ORDER BY RAND()** function, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+--------+--------+---------------+-------------------------+
| id     | name | work_date   | daily_typing_pages |
+--------+--------+---------------+-------------------------+
|     1 | John | 2007-01-24 |                 250 |
|     2 | Ram  | 2007-05-27 |                 220 |
|     3 | Jack | 2007-05-06 |                 170 |
|     3 | Jack | 2007-04-06 |                 100 |
|     4 | Jill | 2007-04-06 |                 220 |
|     5 | Zara | 2007-06-06 |                 300 |
|     5 | Zara | 2007-02-06 |                 350 |
+--------+--------+---------------+-------------------------+
7 rows in set (0.00 sec)
```

Now, use the following commands:

```
SQL> SELECT * FROM employee_tbl ORDER BY RAND();
+--------+--------+---------------+-------------------------+
| id     | name | work_date   | daily_typing_pages |
+--------+--------+---------------+-------------------------+
|     5 | Zara | 2007-06-06 |                 300 |
|     3 | Jack | 2007-04-06 |                 100 |
|     3 | Jack | 2007-05-06 |                 170 |
|     2 | Ram  | 2007-05-27 |                 220 |
|     4 | Jill | 2007-04-06 |                 220 |
|     5 | Zara | 2007-02-06 |                 350 |
|     1 | John | 2007-01-24 |                 250 |
+--------+--------+---------------+-------------------------+
7 rows in set (0.01 sec)

SQL> SELECT * FROM employee_tbl ORDER BY RAND();
+--------+--------+---------------+-------------------------+
| id     | name | work_date   | daily_typing_pages |
+--------+--------+---------------+-------------------------+
|     5 | Zara | 2007-02-06 |                 350 |
|     2 | Ram  | 2007-05-27 |                 220 |
|     3 | Jack | 2007-04-06 |                 100 |
|     1 | John | 2007-01-24 |                 250 |
|     4 | Jill | 2007-04-06 |                 220 |
|     3 | Jack | 2007-05-06 |                 170 |
|     5 | Zara | 2007-06-06 |                 300 |
+--------+--------+---------------+-------------------------+
7 rows in set (0.00 sec)
```

SQL CONCAT Function

SQL **CONCAT** function is used to concatenate two strings to form a single string. Try out the following example:

```
SQL> SELECT CONCAT('FIRST ', 'SECOND');
+-------------------------------------+
| CONCAT('FIRST ', 'SECOND') |
+-------------------------------------+
| FIRST SECOND                |
+-------------------------------------+
1 row in set (0.00 sec)
```

To understand **CONCAT** function in more detail, consider an **employee_tbl** table, which is having the following records:

```
SQL> SELECT * FROM employee_tbl;
+--------+--------+---------------+--------------------------+
| id   | name | work_date  | daily_typing_pages |
+--------+--------+---------------+--------------------------+
|    1 | John | 2007-01-24 |                250 |
|    2 | Ram  | 2007-05-27 |                220 |
|    3 | Jack | 2007-05-06 |                170 |
|    3 | Jack | 2007-04-06 |                100 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
|    5 | Zara | 2007-02-06 |                350 |
+--------+--------+---------------+--------------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to concatenate all the names employee ID and work_date, then you can do it using the following command:

```
SQL> SELECT CONCAT(id, name, work_date)
    -> FROM employee_tbl;
+-------------------------------------+
| CONCAT(id, name, work_date) |
+-------------------------------------+
| 1John2007-01-24             |
| 2Ram2007-05-27              |
| 3Jack2007-05-06             |
| 3Jack2007-04-06             |
| 4Jill2007-04-06             |
| 5Zara2007-06-06             |
| 5Zara2007-02-06             |
+-------------------------------------+
7 rows in set (0.00 sec)
```

SQL Numeric Function

SQL numeric functions are used primarily for numeric manipulation and/or mathematical calculations. The following table details the numeric functions:

Name	Description
ABS()	Returns the absolute value of numeric expression.
ACOS()	Returns the arccosine of numeric expression. Returns NULL if the value is not in the range -1 to 1.
ASIN()	Returns the arcsine of numeric expression. Returns NULL if value is not in the range -1 to 1

ATAN()	Returns the arctangent of numeric expression.
ATAN2()	Returns the arctangent of the two variables passed to it.
BIT_AND()	Returns the bitwise AND all the bits in expression.
BIT_COUNT()	Returns the string representation of the binary value passed to it.
BIT_OR()	Returns the bitwise OR of all the bits in the passed expression.
CEIL()	Returns the smallest integer value that is not less than passed numeric expression
CEILING()	Returns the smallest integer value that is not less than passed numeric expression
CONV()	Convert numeric expression from one base to another.
COS()	Returns the cosine of passed numeric expression. The numeric expression should be expressed in radians.
COT()	Returns the cotangent of passed numeric expression.
DEGREES()	Returns numeric expression converted from radians to degrees.
EXP()	Returns the base of the natural logarithm (e) raised to the power of passed numeric expression.
FLOOR()	Returns the largest integer value that is not greater than passed numeric expression.
FORMAT()	Returns a numeric expression rounded to a number of decimal places.
GREATEST()	Returns the largest value of the input expressions.
INTERVAL()	Takes multiple expressions exp1, exp2 and exp3 so on.. and returns 0 if exp1 is less than exp2, returns 1 if exp1 is less than exp3 and so on.
LEAST()	Returns the minimum-valued input when given two or more.
LOG()	Returns the natural logarithm of the passed numeric expression.
LOG10()	Returns the base-10 logarithm of the passed numeric expression.
MOD()	Returns the remainder of one expression by diving by another expression.
OCT()	Returns the string representation of the octal value of the passed numeric expression. Returns NULL if passed value is NULL.
PI()	Returns the value of pi
POW()	Returns the value of one expression raised to the power of another expression
POWER()	Returns the value of one expression raised to the power of another expression
RADIANS()	Returns the value of passed expression converted from degrees to radians.
ROUND()	Returns numeric expression rounded to an integer. Can be used to round an expression to a number of decimal points
SIN()	Returns the sine of numeric expression given in radians.
SQRT()	Returns the non-negative square root of numeric expression.
STD()	Returns the standard deviation of the numeric expression.
STDDEV()	Returns the standard deviation of the numeric expression.

TAN()	Returns the tangent of numeric expression expressed in radians.
TRUNCATE()	Returns numeric exp1 truncated to exp2 decimal places. If exp2 is 0, then the result will have no decimal point.

ABS(X)

The ABS() function returns the absolute value of X. Consider the following example:

```
SQL> SELECT ABS(2);
+-------------------------------------------------------------------------+
| ABS(2)                                                                  |
+-------------------------------------------------------------------------+
| 2                                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT ABS(-2);
+-------------------------------------------------------------------------+
| ABS(2)                                                                  |
+-------------------------------------------------------------------------+
| 2                                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

ACOS(X)

This function returns the arccosine of X. The value of X must range between -1 and 1 or NULL will be returned. Consider the following example:

```
SQL> SELECT ACOS(1);
+-------------------------------------------------------------------------+
| ACOS(1)                                                                 |
+-------------------------------------------------------------------------+
| 0.000000                                                                |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

ASIN(X)

The ASIN() function returns the arcsine of X. The value of X must be in the range of -1 to 1 or NULL is returned.

```
SQL> SELECT ASIN(1);
+-------------------------------------------------------------------------+
| ASIN(1)                                                                 |
+-------------------------------------------------------------------------+
| 1.5707963267949                                                         |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

ATAN(X)

This function returns the arctangent of X.

```
SQL> SELECT ATAN(1);
+-------------------------------------------------------------------------+
```

```
| ATAN(1)                                                                    |
+----------------------------------------------------------------------------+
| 0.78539816339745                                                           |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

ATAN2(Y,X)

This function returns the arctangent of the two arguments: X and Y. It is similar to the arctangent of Y/X, except that the signs of both are used to find the quadrant of the result.

```
SQL> SELECT ATAN2(3,6);
+----------------------------------------------------------------------------+
| ATAN2(3,6)                                                                 |
+----------------------------------------------------------------------------+
| 0.46364760900081                                                           |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

BIT_AND(expression)

The BIT_AND function returns the bitwise AND of all bits in expression. The basic premise is that if two corresponding bits are the same, then a bitwise AND operation will return 1, while if they are different, a bitwise AND operation will return 0. The function itself returns a 64-bit integer value. If there are no matches, then it will return 18446744073709551615. The following example performs the BIT_AND function on the PRICE column grouped by the MAKER of the car:

```
SQL> SELECT
          MAKER, BIT_AND(PRICE) BITS
          FROM CARS GROUP BY MAKER
+----------------------------------------------------------------------------+
|MAKER           BITS                                                        |
+----------------------------------------------------------------------------+
|CHRYSLER        512                                                         |
|FORD            12488                                                       |
|HONDA           2144                                                        |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

BIT_COUNT(numeric_value)

The BIT_COUNT() function returns the number of bits that are active in numeric_value. The following example demonstrates using the BIT_COUNT() function to return the number of active bits for a range of numbers:

```
SQL> SELECT
          BIT_COUNT(2) AS TWO,
          BIT_COUNT(4) AS FOUR,
          BIT_COUNT(7) AS SEVEN
+-------+--------+----------+
| TWO | FOUR | SEVEN |
+-------+--------+----------+
|   1 |    1 |     3 |
+-------+--------+----------+
1 row in set (0.00 sec)
```

BIT_OR(expression)

The BIT_OR() function returns the bitwise OR of all the bits in expression. The basic premise of the bitwise OR function is that it returns 0 if the corresponding bits match and 1 if they do not. The function returns a 64-bit integer, and if there are no matching rows, then it returns 0. The following example performs the BIT_OR() function on the PRICE column of the CARS table, grouped by the MAKER:

```
SQL> SELECT
          MAKER, BIT_OR(PRICE) BITS
          FROM CARS GROUP BY MAKER
+-------------------------------------------------------------------------+
|MAKER           BITS                                                     |
+-------------------------------------------------------------------------+
|CHRYSLER        62293                                                    |
|FORD            16127                                                    |
|HONDA           32766                                                    |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

CEIL(X)

CEILING(X)

These functions return the smallest integer value that is not smaller than X. Consider the following example:

```
SQL> SELECT CEILING(3.46);
+-------------------------------------------------------------------------+
| CEILING(3.46)                                                           |
+-------------------------------------------------------------------------+
| 4                                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT CEIL(-6.43);
+-------------------------------------------------------------------------+
| CEIL(-6.43)                                                             |
+-------------------------------------------------------------------------+
| -6                                                                      |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

CONV(N,from_base,to_base)

The purpose of the CONV() function is to convert numbers between different number bases. The function returns a string of the value N converted from from_base to to_base. The minimum base value is 2 and the maximum is 36. If any of the arguments are NULL, then the function returns NULL. Consider the following example, which converts the number 5 from base 16 to base 2:

```
SQL> SELECT CONV(5,16,2);
+-------------------------------------------------------------------------+
| CONV(5,16,2)                                                            |
+-------------------------------------------------------------------------+
| 101                                                                     |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

COS(X)

This function returns the cosine of X. The value of X is given in radians.

```
SQL>SELECT COS(90);
+-------------------------------------------------------------------------+
| COS(90)                                                                 |
+-------------------------------------------------------------------------+
| -0.44807361612917                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

COT(X)

This function returns the cotangent of X. Consider the following example:

```
SQL>SELECT COT(1);
+-------------------------------------------------------------------------+
| COT(1)                                                                  |
+-------------------------------------------------------------------------+
| 0.64209261593433                                                        |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

DEGREES(X)

This function returns the value of X converted from radians to degrees.

```
SQL>SELECT DEGREES(PI());
+-------------------------------------------------------------------------+
| DEGREES(PI())                                                           |
+-------------------------------------------------------------------------+
| 180.000000                                                              |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

EXP(X)

This function returns the value of e (the base of the natural logarithm) raised to the power of X.

```
SQL>SELECT EXP(3);
+-------------------------------------------------------------------------+
| EXP(3)                                                                  |
+-------------------------------------------------------------------------+
| 20.085537                                                               |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

FLOOR(X)

This function returns the largest integer value that is not greater than X.

```
SQL>SELECT FLOOR(7.55);
+-------------------------------------------------------------------------+
| FLOOR(7.55)                                                             |
+-------------------------------------------------------------------------+
```

```
| 7                                                                          |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

FORMAT(X,D)

The FORMAT() function is used to format the number X in the following format: ###,###,###.## truncated to D decimal places. The following example demonstrates the use and output of the FORMAT() function:

```
SQL>SELECT FORMAT(423423234.65434453,2);
+----------------------------------------------------------------------------+
| FORMAT(423423234.65434453,2)                                               |
+----------------------------------------------------------------------------+
| 423,423,234.65                                                             |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

GREATEST(n1,n2,n3,.....)

The GREATEST() function returns the greatest value in the set of input parameters (n1, n2, n3, a nd so on). The following example uses the GREATEST() function to return the largest number from a set of numeric values:

```
SQL>SELECT GREATEST(3,5,1,8,33,99,34,55,67,43);
+----------------------------------------------------------------------------+
| GREATEST(3,5,1,8,33,99,34,55,67,43)                                        |
+----------------------------------------------------------------------------+
| 99                                                                         |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

INTERVAL(N,N1,N2,N3,.....)

The INTERVAL() function compares the value of N to the value list (N1, N2, N3, and so on). The function returns 0 if N < N1, 1 if N < N2, 2 if N <N3, and so on. It will return -1 if N is NULL. The value list must be in the form N1 < N2 < N3 in order to work properly. The following code is a simple example of how the INTERVAL() function works:

```
SQL>SELECT INTERVAL(6,1,2,3,4,5,6,7,8,9,10);
+----------------------------------------------------------------------------+
| INTERVAL(6,1,2,3,4,5,6,7,8,9,10)                                           |
+----------------------------------------------------------------------------+
| 6                                                                          |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

INTERVAL(N,N1,N2,N3,.....)

The INTERVAL() function compares the value of N to the value list (N1, N2, N3, and so on). The function returns 0 if N < N1, 1 if N < N2, 2 if N <N3, and so on. It will return -1 if N is NULL. The value list must be in the form N1 < N2 < N3 in order to work properly. The following code is a simple example of how the INTERVAL() function works:

```
SQL>SELECT INTERVAL(6,1,2,3,4,5,6,7,8,9,10);
+----------------------------------------------------------------------------+
| INTERVAL(6,1,2,3,4,5,6,7,8,9,10)                                           |
+----------------------------------------------------------------------------+
| 6                                                                          |
+----------------------------------------------------------------------------+
```

```
1 row in set (0.00 sec)
```

Remember that 6 is the zero-based index in the value list of the first value that was greater than N. In our case, 7 was the offending value and is located in the sixth index slot.

LEAST(N1,N2,N3,N4,...)

The LEAST() function is the opposite of the GREATEST() function. Its purpose is to return the least-valued item from the value list (N1, N2, N3, and so on). The following example shows the proper usage and output for the LEAST() function:

```
SQL>SELECT LEAST(3,5,1,8,33,99,34,55,67,43);
+-------------------------------------------------------------------------+
| LEAST(3,5,1,8,33,99,34,55,67,43)                                        |
+-------------------------------------------------------------------------+
| 1                                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LOG(X)

LOG(B,X)

The single argument version of the function will return the natural logarithm of X. If it is called with two arguments, it returns the logarithm of X for an arbitrary base B. Consider the following example:

```
SQL>SELECT LOG(45);
+-------------------------------------------------------------------------+
| LOG(45)                                                                 |
+-------------------------------------------------------------------------+
| 3.806662                                                                |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL>SELECT LOG(2,65536);
+-------------------------------------------------------------------------+
| LOG(2,65536)                                                            |
+-------------------------------------------------------------------------+
| 16.000000                                                               |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LOG10(X)

This function returns the base-10 logarithm of X.

```
SQL>SELECT LOG10(100);
+-------------------------------------------------------------------------+
| LOG10(100)                                                              |
+-------------------------------------------------------------------------+
| 2.000000                                                                |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

MOD(N,M)

This function returns the remainder of N divided by M. Consider the following example:

```
SQL>SELECT MOD(29,3);
+-------------------------------------------------------------------------------+
| MOD(29,3)                                                                     |
+-------------------------------------------------------------------------------+
| 2                                                                             |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

OCT(N)

The OCT() function returns the string representation of the octal number N. This is equivalent to using CONV(N,10,8).

```
SQL>SELECT OCT(12);
+-------------------------------------------------------------------------------+
| OCT(12)                                                                       |
+-------------------------------------------------------------------------------+
| 14                                                                            |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

PI()

This function simply returns the value of pi. SQL internally stores the full double-precision value of pi.

```
SQL>SELECT PI();
+-------------------------------------------------------------------------------+
| PI()                                                                          |
+-------------------------------------------------------------------------------+
| 3.141593                                                                      |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

POW(X,Y)

POWER(X,Y)

These two functions return the value of X raised to the power of Y.

```
SQL> SELECT POWER(3,3);
+-------------------------------------------------------------------------------+
| POWER(3,3)                                                                    |
+-------------------------------------------------------------------------------+
| 27                                                                            |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

RADIANS(X)

This function returns the value of X, converted from degrees to radians.

```
SQL>SELECT RADIANS(90);
+-----------------------------------------------------------------------------+
| RADIANS(90)                                                                 |
+-----------------------------------------------------------------------------+
|1.570796                                                                     |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

ROUND(X)

ROUND(X,D)

This function returns X rounded to the nearest integer. If a second argument, D, is supplied, then the function returns X rounded to D decimal places. D must be positive or all digits to the right of the decimal point will be removed. Consider the following example:

```
SQL>SELECT ROUND(5.693893);
+-----------------------------------------------------------------------------+
| ROUND(5.693893)                                                             |
+-----------------------------------------------------------------------------+
| 6                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL>SELECT ROUND(5.693893,2);
+-----------------------------------------------------------------------------+
| ROUND(5.693893,2)                                                           |
+-----------------------------------------------------------------------------+
| 5.69                                                                        |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SIGN(X)

This function returns the sign of X (negative, zero, or positive) as -1, 0, or 1.

```
SQL>SELECT SIGN(-4.65);
+-----------------------------------------------------------------------------+
| SIGN(-4.65)                                                                 |
+-----------------------------------------------------------------------------+
| -1                                                                          |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL>SELECT SIGN(0);
+-----------------------------------------------------------------------------+
| SIGN(0)                                                                     |
+-----------------------------------------------------------------------------+
| 0                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL>SELECT SIGN(4.65);
+-----------------------------------------------------------------------------+
| SIGN(4.65)                                                                  |
+-----------------------------------------------------------------------------+
| 1                                                                           |
+-----------------------------------------------------------------------------+
```

```
1 row in set (0.00 sec)
```

SIN(X)

This function returns the sine of X. Consider the following example:

```
SQL>SELECT SIN(90);
+-------------------------------------------------------------------------+
| SIN(90)                                                                 |
+-------------------------------------------------------------------------+
| 0.893997                                                                |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SQRT(X)

This function returns the non-negative square root of X. Consider the following example:

```
SQL>SELECT SQRT(49);
+-------------------------------------------------------------------------+
| SQRT(49)                                                                |
+-------------------------------------------------------------------------+
| 7                                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

STD(expression)

STDDEV(expression)

The STD() function is used to return the standard deviation of expression. This is equivalent to taking the square root of the VARIANCE() of expression. The following example computes the standard deviation of the PRICE column in our CARS table:

```
SQL>SELECT STD(PRICE) STD_DEVIATION FROM CARS;
+-------------------------------------------------------------------------+
| STD_DEVIATION                                                           |
+-------------------------------------------------------------------------+
| 7650.2146                                                               |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

TAN(X)

This function returns the tangent of the argument X, which is expressed in radians.

```
SQL>SELECT TAN(45);
+-------------------------------------------------------------------------+
| TAN(45)                                                                 |
+-------------------------------------------------------------------------+
| 1.619775                                                                |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

TRUNCATE(X,D)

This function is used to return the value of X truncated to D number of decimal places. If D is 0, then the decimal point is removed. If D is negative, then D number of values in the integer part of the value is truncated. Consider the following example:

```
SQL>SELECT TRUNCATE(7.536432,2);
+-------------------------------------------------------------------------+
| TRUNCATE(7.536432,2)                                                    |
+-------------------------------------------------------------------------+
| 7.53                                                                    |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SQL String Function

SQL string functions are used primarily for string manipulation. The following table details the important string functions:

Name	Description
ASCII()	Returns numeric value of left-most character
BIN()	Returns a string representation of the argument
BIT_LENGTH()	Returns length of argument in bits
CHAR_LENGTH()	Returns number of characters in argument
CHAR()	Returns the character for each integer passed
CHARACTER_LENGTH()	A synonym for CHAR_LENGTH()
CONCAT_WS()	Returns concatenate with separator
CONCAT()	Returns concatenated string
CONV()	Converts numbers between different number bases
ELT()	Returns string at index number
EXPORT_SET()	Returns a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Returns the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Returns the index position of the first argument within the second argument
FORMAT()	Returns a number formatted to specified number of decimal places
HEX()	Returns a string representation of a hex value
INSERT()	Inserts a substring at the specified position up to the specified number of characters
INSTR()	Returns the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Returns the leftmost number of characters as specified

LENGTH()	Returns the length of a string in bytes
LOAD_FILE()	Loads the named file
LOCATE()	Returns the position of the first occurrence of substring
LOWER()	Returns the argument in lowercase
LPAD()	Returns the string argument, left-padded with the specified string
LTRIM()	Removes leading spaces
MAKE_SET()	Returns a set of comma-separated strings that have the corresponding bit in bits set
MID()	Returns a substring starting from the specified position
OCT()	Returns a string representation of the octal argument
OCTET_LENGTH()	A synonym for LENGTH()
ORD()	If the leftmost character of the argument is a multi-byte character, returns the code for that character
POSITION()	A synonym for LOCATE()
QUOTE()	Escapes the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replaces occurrences of a specified string
REVERSE()	Reverses the characters in a string
RIGHT()	Returns the specified rightmost number of characters
RPAD()	Appends string the specified number of times
RTRIM()	Removes trailing spaces
SOUNDEX()	Returns a soundex string
SOUNDS LIKE	Compares sounds
SPACE()	Returns a string of the specified number of spaces
STRCMP()	Compares two strings
SUBSTRING_INDEX()	Returns a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING(), SUBSTR()	Returns the substring as specified
TRIM()	Removes leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Converts each pair of hexadecimal digits to a character
UPPER()	Converts to uppercase

ASCII(str)

Returns the numeric value of the leftmost character of the string str. Returns 0 if str is the empty string. Returns NULL if str is NULL. ASCII() works for characters with numeric values from 0 to 255.

```
SQL> SELECT ASCII('2');
+-----------------------------------------------------------------------+
| ASCII('2')                                                            |
+-----------------------------------------------------------------------+
| 50                                                                    |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT ASCII('dx');
+-----------------------------------------------------------------------+
| ASCII('dx')                                                           |
+-----------------------------------------------------------------------+
| 100                                                                   |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

BIN(N)

Returns a string representation of the binary value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,2). Returns NULL if N is NULL.

```
SQL> SELECT BIN(12);
+-----------------------------------------------------------------------+
| BIN(12)                                                               |
+-----------------------------------------------------------------------+
| 1100                                                                  |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

BIT_LENGTH(str)

Returns the length of the string str in bits.

```
SQL> SELECT BIT_LENGTH('text');
+-----------------------------------------------------------------------+
| BIT_LENGTH('text')                                                    |
+-----------------------------------------------------------------------+
| 32                                                                    |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

CHAR(N,... [USING charset_name])

CHAR() interprets each argument N as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

```
SQL> SELECT CHAR(77,121,83,81,'76');
+-----------------------------------------------------------------------+
| CHAR(77,121,83,81,'76')                                               |
+-----------------------------------------------------------------------+
| SQL                                                                   |
+-----------------------------------------------------------------------+
```

```
1 row in set (0.00 sec)
```

CHAR_LENGTH(str)

Returns the length of the string str measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, LENGTH() returns 10, whereas CHAR_LENGTH() returns 5.

```
SQL> SELECT CHAR_LENGTH("text");
+-----------------------------------------------------------------------------+
| CHAR_LENGTH("text")                                                         |
+-----------------------------------------------------------------------------+
| 4                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

CHARACTER_LENGTH(str)

CHARACTER_LENGTH() is a synonym for CHAR_LENGTH().

CONCAT(str1,str2,...)

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

```
SQL> SELECT CONCAT('My', 'S', 'QL');
+-----------------------------------------------------------------------------+
| CONCAT('My', 'S', 'QL')                                                     |
+-----------------------------------------------------------------------------+
| SQL                                                                        |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

CONCAT_WS(separator,str1,str2,...)

CONCAT_WS() stands for Concatenate With Separator and is a special form of CONCAT(). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.

```
SQL> SELECT CONCAT_WS(',','First name','Last Name' );
+-----------------------------------------------------------------------------+
| CONCAT_WS(',','First name','Last Name' )                                    |
+-----------------------------------------------------------------------------+
| First name, Last Name                                                       |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

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 to_base. Returns NULL if any argument is NULL. The argument N is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If to_base is a negative number, N is regarded as a signed number. Otherwise, N is treated as unsigned. CONV() works with 64-bit precision.

```
SQL> SELECT CONV('a',16,2);
+-----------------------------------------------------------------------------+
| CONV('a',16,2)                                                              |
+-----------------------------------------------------------------------------+
| 1010                                                                        |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

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().

```
SQL> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
+-----------------------------------------------------------------------------+
| ELT(1, 'ej', 'Heja', 'hej', 'foo')                                          |
+-----------------------------------------------------------------------------+
| ej                                                                          |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

EXPORT_SET(bits,on,off[,separator[,number_of_bits]])

Returns a string such that for every bit set in the value bits, you get an on string and for every bit not set in the value, you get an off string. Bits in bits are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the separator string (the default being the comma character .,.). The number of bits examined is given by number_of_bits (defaults to 64).

```
SQL> SELECT EXPORT_SET(5,'Y','N',',',4);
+-----------------------------------------------------------------------------+
| EXPORT_SET(5,'Y','N',',',4)                                                 |
+-----------------------------------------------------------------------------+
| Y,N,Y,N                                                                     |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

FIELD(str,str1,str2,str3,...)

Returns the index (position starting with 1) of str in the str1, str2, str3, ... list. Returns 0 if str is not found.

```
SQL> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
+-----------------------------------------------------------------------------+
| FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')                              |
+-----------------------------------------------------------------------------+
| 2                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

FIND_IN_SET(str,strlist)

Returns a value in the range of 1 to N if the string str is in the string list strlist consisting of N substrings.

```
SQL> SELECT FIND_IN_SET('b','a,b,c,d');
+-----------------------------------------------------------------------------+
| SELECT FIND_IN_SET('b','a,b,c,d')                                           |
+-----------------------------------------------------------------------------+
| 2                                                                           |
+-----------------------------------------------------------------------------+
```

```
1 row in set (0.00 sec)
```

FORMAT(X,D)

Formats the number X to a format like '#,###,###.##', rounded to D decimal places, and returns the result as a string. If D is 0, the result has no decimal point or fractional part.

```
SQL> SELECT FORMAT(12332.123456, 4);
+--------------------------------------------------------------------------+
| FORMAT(12332.123456, 4)                                                  |
+--------------------------------------------------------------------------+
| 12,332.1235                                                              |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

HEX(N_or_S)

If N_or_S is a number, returns a string representation of the hexadecimal value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,16).

If N_or_S is a string, returns a hexadecimal string representation of N_or_S where each character in N_or_S is converted to two hexadecimal digits.

```
SQL> SELECT HEX(255);
+--------------------------------------------------------------------------+
| HEX(255)                                                                 |
+--------------------------------------------------------------------------+
| FF                                                                       |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT 0x616263;
+--------------------------------------------------------------------------+
| 0x616263                                                                 |
+--------------------------------------------------------------------------+
| abc                                                                      |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

INSERT(str,pos,len,newstr)

Returns the string str, with the substring beginning at position pos and len characters long replaced by the string newstr. Returns the original string if pos is not within the length of the string. Replaces the rest of the string from position pos if len is not within the length of the rest of the string. Returns NULL if any argument is NULL.

```
SQL> SELECT INSERT('Quadratic', 3, 4, 'What');
+--------------------------------------------------------------------------+
| INSERT('Quadratic', 3, 4, 'What')                                        |
+--------------------------------------------------------------------------+
| QuWhattic                                                                |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

INSTR(str,substr)

Returns the position of the first occurrence of substring substr in string str. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.

```
SQL> SELECT INSTR('foobarbar', 'bar');
+--------------------------------------------------------------------------+
| INSTR('foobarbar', 'bar')                                                |
+--------------------------------------------------------------------------+
| 4                                                                        |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LCASE(str)

LCASE() is a synonym for LOWER().

LEFT(str,len)

Returns the leftmost len characters from the string str, or NULL if any argument is NULL.

```
SQL> SELECT LEFT('foobarbar', 5);
+--------------------------------------------------------------------------+
| LEFT('foobarbar', 5)                                                     |
+--------------------------------------------------------------------------+
| fooba                                                                    |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LENGTH(str)

Returns the length of the string str measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, LENGTH() returns 10, whereas CHAR_LENGTH() returns 5.

```
SQL> SELECT LENGTH('text');
+--------------------------------------------------------------------------+
| LENGTH('text')                                                           |
+--------------------------------------------------------------------------+
| 4                                                                        |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LOAD_FILE(file_name)

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full pathname to the file, and you must have the FILE privilege. The file must be readable by all and its size less than max_allowed_packet bytes.

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns NULL.

As of SQL 5.0.19, the character_set_filesystem system variable controls interpretation of filenames that are given as literal strings.

```
SQL> UPDATE table_test
    -> SET blob_col=LOAD_FILE('/tmp/picture')
        -> WHERE id=1;
.....................................................
```

LOCATE(substr,str), LOCATE(substr,str,pos)

The first syntax returns the position of the first occurrence of substring substr in string str. The second syntax 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.

```
SQL> SELECT LOCATE('bar', 'foobarbar');
+-----------------------------------------------------------------------+
| LOCATE('bar', 'foobarbar')                                            |
+-----------------------------------------------------------------------+
| 4                                                                     |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LOWER(str)

Returns the string str with all characters changed to lowercase according to the current character set mapping.

```
SQL> SELECT LOWER('QUADRATICALLY');
+-----------------------------------------------------------------------+
| LOWER('QUADRATICALLY')                                                |
+-----------------------------------------------------------------------+
| quadratically                                                         |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LPAD(str,len,padstr)

Returns the string str, left-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characters.

```
SQL> SELECT LPAD('hi',4,'??');
+-----------------------------------------------------------------------+
| LPAD('hi',4,'??')                                                     |
+-----------------------------------------------------------------------+
| ??hi                                                                  |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

LTRIM(str)

Returns the string str with leading space characters removed.

```
SQL> SELECT LTRIM('  barbar');
+-----------------------------------------------------------------------+
| LTRIM('  barbar')                                                     |
+-----------------------------------------------------------------------+
| barbar                                                                |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

MAKE_SET(bits,str1,str2,...)

Returns a set value (a string containing substrings separated by .,. characters) consisting of the strings that have the corresponding bit in bits set. str1 corresponds to bit 0, str2 to bit 1, and so on. NULL values in str1, str2, ... are not appended to the result.

```
SQL> SELECT MAKE_SET(1,'a','b','c');
+-------------------------------------------------------------------------------+
| MAKE_SET(1,'a','b','c')                                                       |
+-------------------------------------------------------------------------------+
| a                                                                             |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

MID(str,pos,len)

MID(str,pos,len) is a synonym for SUBSTRING(str,pos,len).

OCT(N)

Returns a string representation of the octal value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,8). Returns NULL if N is NULL.

```
SQL> SELECT OCT(12);
+-------------------------------------------------------------------------------+
| OCT(12)                                                                       |
+-------------------------------------------------------------------------------+
| 14                                                                            |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

OCTET_LENGTH(str)

OCTET_LENGTH() is a synonym for LENGTH().

ORD(str)

If the leftmost character of the string str is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code . 256)
+ (3rd byte code . 2562) ...
```

If the leftmost character is not a multi-byte character, ORD() returns the same value as the ASCII() function.

```
SQL> SELECT ORD('2');
+-------------------------------------------------------------------------------+
| ORD('2')                                                                      |
+-------------------------------------------------------------------------------+
| 50                                                                            |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

POSITION(substr IN str)

POSITION(substr IN str) is a synonym for LOCATE(substr,str).

QUOTE(str)

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotes and with each instance of single quote (' ‘ '), backslash ('\'), ASCII NUL, and Control-Z preceded by a backslash. If the argument is NULL, the return value is the word 'NULL' without enclosing single quotes.

```
SQL> SELECT QUOTE('Don\'t!');
+-----------------------------------------------------------------------------+
| QUOTE('Don\'t!')                                                            |
+-----------------------------------------------------------------------------+
| 'Don\'t!'                                                                   |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

NOTE: Please check if your installation has any bug with this function then don't use this function.

expr REGEXP pattern

This function performs a pattern match of expr against pattern. Returns 1 if expr matches pat; otherwise it returns 0. If either expr or pat is NULL, the result is NULL. REGEXP is not case sensitive, except when used with binary strings.

```
SQL> SELECT 'ABCDEF' REGEXP 'A%C%%';
+-----------------------------------------------------------------------------+
| 'ABCDEF' REGEXP 'A%C%%'                                                     |
+-----------------------------------------------------------------------------+
| 0                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Another example is:

```
SQL> SELECT 'ABCDE' REGEXP '.*';
+-----------------------------------------------------------------------------+
|  'ABCDE' REGEXP '.*'                                                        |
+-----------------------------------------------------------------------------+
| 1                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Let's see one more example:

```
SQL> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
+-----------------------------------------------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line'                                       |
+-----------------------------------------------------------------------------+
| 1                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

REPEAT(str,count)

Returns a string consisting of the string str repeated count times. If count is less than 1, returns an empty string. Returns NULL if str or count are NULL.

```
SQL> SELECT REPEAT('SQL', 3);
+-----------------------------------------------------------------------------+
```

```
| REPEAT('SQL', 3)                                                    |
+---------------------------------------------------------------------+
| SQLSQLSQL                                                           |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```

REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str replaced by the string to_str. REPLACE() performs a case-sensitive match when searching for from_str.

```
SQL> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
+---------------------------------------------------------------------+
| REPLACE('www.mysql.com', 'w', 'Ww')                                 |
+---------------------------------------------------------------------+
| WwWwWw.mysql.com                                                    |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```

REVERSE(str)

Returns the string str with the order of the characters reversed.

```
SQL> SELECT REVERSE('abcd');
+---------------------------------------------------------------------+
| REVERSE('abcd')                                                     |
+---------------------------------------------------------------------+
| dcba                                                                |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```

RIGHT(str,len)

Returns the rightmost len characters from the string str, or NULL if any argument is NULL.

```
SQL> SELECT RIGHT('foobarbar', 4);
+---------------------------------------------------------------------+
| RIGHT('foobarbar', 4)                                               |
+---------------------------------------------------------------------+
| rbar                                                                |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```

RPAD(str,len,padstr)

Returns the string str, right-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characters.

```
SQL> SELECT RPAD('hi',5,'?');
+---------------------------------------------------------------------+
| RPAD('hi',5,'?')                                                    |
+---------------------------------------------------------------------+
| hi???                                                               |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```

RTRIM(str)

Returns the string str with trailing space characters removed.

```
SQL> SELECT RTRIM('barbar   ');
+-------------------------------------------------------------------------+
| RTRIM('barbar   ')                                                      |
+-------------------------------------------------------------------------+
| barbar                                                                  |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SOUNDEX(str)

Returns a soundex string from str. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the SOUNDEX() function returns an arbitrarily long string. You can use SUBSTRING() on the result to get a standard soundex string. All non-alphabetic characters in str are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

```
SQL> SELECT SOUNDEX('Hello');
+-------------------------------------------------------------------------+
| SOUNDEX('Hello')                                                        |
+-------------------------------------------------------------------------+
| H400                                                                    |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

expr1 SOUNDS LIKE expr2

This is the same as SOUNDEX(expr1) = SOUNDEX(expr2).

SPACE(N)

Returns a string consisting of N space characters.

```
SQL> SELECT SPACE(6);
+-------------------------------------------------------------------------+
| SELECT SPACE(6)                                                         |
+-------------------------------------------------------------------------+
|       '                                                                 |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

STRCMP(str1, str2)

Compares two strings and returns 0 if both strings are equal, it returns -1 if the first argument is smaller than the second according to the current sort order otherwise it returns 1.

```
SQL> SELECT STRCMP('MOHD', 'MOHD');
+-------------------------------------------------------------------------+
| STRCMP('MOHD', 'MOHD')                                                  |
+-------------------------------------------------------------------------+
| 0                                                                       |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Another example is:

```
SQL> SELECT STRCMP('AMOHD', 'MOHD');
+-----------------------------------------------------------------------------+
| STRCMP('AMOHD', 'MOHD')                                                     |
+-----------------------------------------------------------------------------+
| -1                                                                          |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Let's see one more example:

```
SQL> SELECT STRCMP('MOHD', 'AMOHD');
+-----------------------------------------------------------------------------+
| STRCMP('MOHD', 'AMOHD')                                                     |
+-----------------------------------------------------------------------------+
| 1                                                                           |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SUBSTRING(str,pos)

SUBSTRING(str FROM pos)

SUBSTRING(str,pos,len)

SUBSTRING(str FROM pos FOR len)

The forms without a len argument return a substring from string str starting at position pos. The forms with a len argument return a substring len characters long from string str, starting at position pos. The forms that use FROM are standard SQL syntax. It is also possible to use a negative value for pos. In this case, the beginning of the substring is pos characters from the end of the string, rather than the beginning. A negative value may be used for pos in any of the forms of this function.

```
SQL> SELECT SUBSTRING('Quadratically',5);
+-----------------------------------------------------------------------------+
| SSUBSTRING('Quadratically',5)                                               |
+-----------------------------------------------------------------------------+
| ratically                                                                   |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT SUBSTRING('foobarbar' FROM 4);
+-----------------------------------------------------------------------------+
| SUBSTRING('foobarbar' FROM 4)                                               |
+-----------------------------------------------------------------------------+
| barbar                                                                      |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT SUBSTRING('Quadratically',5,6);
+-----------------------------------------------------------------------------+
| SUBSTRING('Quadratically',5,6)                                              |
+-----------------------------------------------------------------------------+
| ratica                                                                      |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SUBSTRING_INDEX(str,delim,count)

Returns the substring from string str before 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_INDEX() performs a case-sensitive match when searching for delim.

```
SQL> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
+-------------------------------------------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', 2)                |
+-------------------------------------------------------------------------+
| www.mysql                                                               |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

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

TRIM([remstr FROM] str)

Returns the string str with all remstr prefixes or suffixes removed. If none of the specifiers BOTH, LEADING, or TRAILING is given, BOTH is assumed. remstr is optional and, if not specified, spaces are removed.

```
SQL> SELECT TRIM('  bar   ');
+-------------------------------------------------------------------------+
| TRIM('  bar   ')                                                        |
+-------------------------------------------------------------------------+
| bar                                                                     |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
+-------------------------------------------------------------------------+
| TRIM(LEADING 'x' FROM 'xxxbarxxx')                                      |
+-------------------------------------------------------------------------+
| barxxx                                                                  |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
+-------------------------------------------------------------------------+
| TRIM(BOTH 'x' FROM 'xxxbarxxx')                                         |
+-------------------------------------------------------------------------+
| bar                                                                     |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
+-------------------------------------------------------------------------+
| TRIM(TRAILING 'xyz' FROM 'barxxyz')                                     |
+-------------------------------------------------------------------------+
| barx                                                                    |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

UCASE(str)

UCASE() is a synonym for UPPER().

UNHEX(str)

Performs the inverse operation of HEX(str). That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

```
SQL> SELECT UNHEX('4D7953514C');
+--------------------------------------------------------------------------+
| UNHEX('4D7953514C')                                                      |
+--------------------------------------------------------------------------+
| SQL                                                                      |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

The characters in the argument string must be legal hexadecimal digits: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. If UNHEX() encounters any non-hexadecimal digits in the argument, it returns NULL.

UPPER(str)

Returns the string str with all characters changed to uppercase according to the current character set mapping.

```
SQL> SELECT UPPER('Allah-hus-samad');
+--------------------------------------------------------------------------+
| UPPER('Allah-hus-samad')                                                 |
+--------------------------------------------------------------------------+
| ALLAH-HUS-SAMAD                                                          |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```