12.15 Miscellaneous Functions

Table 12.19 Miscellaneous Functions

Name	Description
DEFAULT()	Return the default value for a table column
GET_LOCK()	Get a named lock
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
IS_FREE_LOCK()	Checks whether the named lock is free
IS_USED_LOCK()	Checks whether the named lock is in use. Return connection identifier if true.
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
NAME_CONST()	Causes the column to have the given name
RAND()	Return a random floating-point value
RELEASE_LOCK()	Releases the named lock
SLEEP()	Sleep for a number of seconds
UUID()	Return a Universal Unique Identifier (UUID)
VALUES()	Defines the values to be used during an INSERT

- **DEFAULT(*col_name*)**

  Returns the default value for a table column. Starting with MySQL 5.0.2, an error results if the column has no default value.

  ```
  mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
  ```

- **FORMAT(*X*,*D*)**

  Formats the number ***X*** to a format like `'#,###,###.##'`, rounded to ***D*** decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions”.

- **GET_LOCK(*str*,*timeout*)**

  Tries to obtain a lock with a name given by the string ***str***, using a timeout of ***timeout*** seconds. Returns **1** if the lock was obtained successfully, **0** if the attempt timed out (for example, because another client has previously locked the name), or **NULL** if an error occurred (such as running out of memory or the thread was killed with **mysqladmin kill**). If you have a lock obtained with**GET_LOCK()**, it is released when you execute **RELEASE_LOCK()**, execute a new **GET_LOCK()**, or your connection terminates (either normally or abnormally). Locks obtained with **GET_LOCK()** do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

  GET_LOCK() can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked within one session, **GET_LOCK()** blocks any request by another session for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or

deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form ***db_name.str*** or ***app_name.str***.

```
mysql> SELECT GET_LOCK('lock1',10);
        -> 1
mysql> SELECT IS_FREE_LOCK('lock2');
        -> 1
mysql> SELECT GET_LOCK('lock2',10);
        -> 1
mysql> SELECT RELEASE_LOCK('lock2');
        -> 1
mysql> SELECT RELEASE_LOCK('lock1');
        -> NULL
```

The second **RELEASE_LOCK()** call returns **NULL** because the lock **'lock1'** was automatically released by the second **GET_LOCK()** call.

If multiple clients are waiting for a lock, the order in which they will acquire it is undefined. Applications should not assume that clients will acquire the lock in the same order that they issued the lock requests.

> **Note**
> If a client attempts to acquire a lock that is already held by another client, it blocks according to the ***timeout*** argument. If the blocked client terminates, its thread does not die until the lock request times out. This is a known bug (fixed in MySQL 5.5).

- **INET_ATON(*expr*)**

Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address in network byte order (big endian). **INET_ATON()** returns**NULL** if it does not understand its argument.

```
mysql> SELECT INET_ATON('10.0.5.9');
        -> 167773449
```

For this example, the return value is calculated as $10\times256^3 + 0\times256^2 + 5\times256 + 9$.

INET_ATON() may or may not return a non-**NULL** result for short-form IP addresses (such as**'127.1'** as a representation of **'127.0.0.1'**). Because of this, **INET_ATON()**a should not be used for such addresses.

> **Note**
> To store values generated by **INET_ATON()**, use an **INT UNSIGNED** column rather than **INT**, which is signed. If you use a signed column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See Section 11.2.6, “Out-of-Range and Overflow Handling”.

- **INET_NTOA(*expr*)**

  Given a numeric IPv4 network address in network byte order, returns the dotted-quad representation of the address as a binary string. **INET_NTOA()** returns **NULL** if it does not understand its argument.

```
mysql> SELECT INET_NTOA(167773449);
        -> '10.0.5.9'
```

- **IS_FREE_LOCK(*str*)**

  Checks whether the lock named ***str*** is free to use (that is, not locked). Returns **1** if the lock is free (no one is using the lock), **0** if the lock is in use, and **NULL** if an error occurs (such as an incorrect argument).

- **IS_USED_LOCK(*str*)**

  Checks whether the lock named ***str*** is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns **NULL**.

- **MASTER_POS_WAIT(*log_name*,*log_pos*[,*timeout*])**

  This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns **NULL**if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns **-1** if the timeout has been exceeded. If the slave SQL thread stops while **MASTER_POS_WAIT()** is waiting, the function returns **NULL**. If the slave is past the specified position, the function returns immediately.

  If a ***timeout*** value is specified, **MASTER_POS_WAIT()** stops waiting when ***timeout*** seconds have elapsed. ***timeout*** must be greater than 0; a zero or negative ***timeout*** means no timeout.

- **NAME_CONST(*name*,*value*)**

  Returns the given value. When used to produce a result set column, **NAME_CONST()** causes the column to have the given name. The arguments should be constants.

```
mysql> SELECT NAME_CONST('myname', 14);
+--------+
| myname |
+--------+
|     14 |
+--------+
```

  This function was added in MySQL 5.0.12. It is for internal use only. The server uses it when writing statements from stored programs that contain references to local program variables, as described inSection 18.6, “Binary Logging of Stored Programs”, You might see this function in the output from**mysqlbinlog**.

For your applications, you can obtain exactly the same result as in the example just shown by using simple aliasing, like this:

```
mysql> SELECT 14 AS myname;
+--------+
| myname |
+--------+
|     14 |
+--------+
1 row in set (0.00 sec)
```

See Section 13.2.8, “SELECT Syntax”, for more information about column aliases.

- **RELEASE_LOCK(*str*)**

  Releases the lock named by the string ***str*** that was obtained with **GET_LOCK()**. Returns **1** if the lock was released, **0** if the lock was not established by this thread (in which case the lock is not released), and **NULL** if the named lock did not exist. The lock does not exist if it was never obtained by a call to**GET_LOCK()** or if it has previously been released.

  The **DO** statement is convenient to use with **RELEASE_LOCK()**. See Section 13.2.3, “DO Syntax”.

- **SLEEP(*duration*)**

  Sleeps (pauses) for the number of seconds given by the ***duration*** argument, then returns 0. If**SLEEP()** is interrupted, it returns 1. The duration may have a fractional part. This function was added in MySQL 5.0.12.

- **UUID()**

  Returns a Universal Unique Identifier (UUID) generated according to “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706,http://www.opengroup.org/public/pubs/catalog/c706.htm).

  A UUID is designed as a number that is globally unique in space and time. Two calls to **UUID()** are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

  A UUID is a 128-bit number represented by a **utf8** string of five hexadecimal numbers in **aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee** format:

  - The first three numbers are generated from a timestamp.
  - The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
  - The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number

is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
        -> '6ccd780c-baba-1026-9564-0040f4311e29'
```

> **Warning**
> Although `UUID()` values are intended to be unique, they are not necessarily unguessable or unpredictable. If unpredictability is required, UUID values should be generated some other way.

> **Note**
> `UUID()` does not work with statement-based replication.

- `VALUES(col_name)`

In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of ***col_name*** that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause of `INSERT` statements and returns `NULL` otherwise.
See Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```