- **MySQL is the most popular Open Source Relational SQL database management system.**
- **MySQL is one of the best RDBMS being used for developing web based software applications.**

What is Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching, and replicating the data it holds.

Other kinds of data stores can be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems.

So now a days we use relational database management systems (RDBMS) to store and manager huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as foreign keys.

A Relational DataBase Management System (RDBMS)

is a software that:

- Enables you to implement a database with tables, columns, and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Updates the indexes automatically.
- Interprets an SQL query and combines information from various tables.

RDBMS Terminology:

- **Database:** A database is a collection of tables, with related data.
- **Table:** A table is a matrix with data. A table in a database looks like a simple spreadsheet.

- **Column:** One column (data element) contains data of one and the same kind.
- **Row:** A row (= tuple, entry or record) is a group of related data.
- **Redundancy:** Storing data twice, redundantly to make the system faster.
- **Primary Key:** A primary key is unique. A key value can not occur twice in one table. With a key you can find at most one row.
- **Foreign Key:** A foreign key is the linking pin between two tables.
- **Compound Key:** A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
- **Index:** An index in a database resembles an index at the back of a book.
- **Referential Integrity:** Referential Integrity makes sure that a foreign key value always points to an existing row.

MySQL Database:

MySQL is a fast, easy-to-use RDBMS used being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons.

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB.

Administrative MySQL Command:

Here is the list of important MySQL command which you will use time to time to work with MySQL database:

- **USE *Databasename*** : This will be used to select a particular database in MySQL workarea.
- **SHOW DATABASES:** Lists the databases that are accessible by the MySQL DBMS.
- **SHOW TABLES:** Shows the tables in the database once a database has been selected with the use command.
- **SHOW COLUMNS FROM *tablename:*** Shows the attributes, types of attributes, key information, whether NULL is permitted, defaults, and other information for a table.

- **SHOW INDEX FROM *tablename*:** Presents the details of all indexes on the table, including the PRIMARY KEY.
- **SHOW TABLE STATUS LIKE *tablename*\G:** Reports details of the MySQL DBMS performance and statistics.

MySQL PHP Syntax

MySQL works very well in combination of various programming languages like PERL, C, C++, JAVA and PHP. Out of these languages, PHP is the most popular one because of its web application development capabilities..

PHP provides various functions to access MySQL database and to manipulate data records inside MySQL database. You would require to call PHP functions in the same way you call any other PHP function.

The PHP functions for use with MySQL have the following general format:

```
mysql_function(value,value,...);
```

The second part of the function name is specific to the function, usually a word that describes what the function does. The following are two of the functions which we will use in our tutorial

```
mysqli_connect($connect);
mysqli_query($connect,"SQL statement");
```

Following example shows a generic sysntax of PHP to call any MySQL function.

```
<html>
<head>
<title>PHP with MySQL</title>
</head>
<body>
<?php
   $retval = mysql_function(value, [value,...]);
   if( !$retval )
   {
       die ( "Error: a related error message" );
   }
   // Otherwise MySQL  or PHP Statements
?>
</body>
</html>
```

Starting from next chapter we will see all the important MySQL functionality along with PHP.

MySQL Connection using PHP Script:

PHP provides **mysql_connect()** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax:

```
mysql_connect(server,user,passwd,new_link,client_flag);
```

Parameter	Description
server	Optional - The host name running database server. If not specified then default value is **localhost:3036**.
user	Optional - The username accessing the database. If not specified then default is the name of the user that owns the server process.
passwd	Optional - The password of the user accessing the database. If not specified then default is an empty password.
new_link	Optional - If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.
client_flags	Optional - A combination of the following constants: • MYSQL_CLIENT_SSL - Use SSL encryption • MYSQL_CLIENT_COMPRESS - Use compression protocol • MYSQL_CLIENT_IGNORE_SPACE - Allow space after function names • MYSQL_CLIENT_INTERACTIVE - Allow interactive timeout seconds of inactivity before closing the connection

Disconnect from MySQL database using PHP Script

You can disconnect from MySQL database anytime using PHP function **mysql_close()**. This function takes a single parameter.

Syntax:

```
mysql_close ($link_identifier );
```

If a resource is not specified then last opened database is closed. This function returns true if it closes connection successfully otherwise it returns false.

Example:

Try out following example to connect to a MySQL server:

```
<?php

   $conn = mysql_connect("localhost","user","password");
   if(! $conn )
   {
     die('Could not connect: ' . mysql_error());
   }
   echo 'Connected successfully';
   mysql_close($conn);
?>
```

Create Database using PHP Script:

PHP uses **mysql_query** function to create or delete a MySQL database. This function takes two parameters.

Syntax:

```
bool mysql_query( sql, connection );
```

Parameter	Description
sql	Required - SQL query to create or delete a MySQL database
connection	Optional - if not specified then last opened connection by mysql_connect will be used.

Example:

Try out following example to create a database:

```
<?php
$conn = mysql_connect("localhost","user","password");
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$create = 'CREATE DATABASE organization';
$return = mysql_query( $create, $conn );
```

```
if(!$return)
{
  die('Could not create database: ' . mysql_error());
}
echo "Database organization created successfully\n";
mysql_close($conn);
?>
```

Drop Database using PHP Script:

Syntax:

```
mysql_query( sql, connection );
```

Parameter	Description
sql	Required - SQL query to create or delete a MySQL database
connection	Optional - if not specified then last opened connection by mysql_connect will be used.

Example:

Try out following example to delete a database:

```
<?php
$conn = mysql_connect("localhost","user","password");
if(!$conn)
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$drop = 'DROP DATABASE TUTORIALS';
$return = mysql_query( $drop, $conn );
if(!$return)
{
  die('Could not delete database: ' . mysql_error());
}
echo "Database TUTORIALS deleted successfully\n";
mysql_close($conn);
?>
```

WARNING: While deleting a database using PHP script, it does not prompt you for any confirmation. So be careful while deleting a MySQL database.

Once you get connection with MySQL server, it is required to select a particular database to work with. This is because there may be more than one database available with MySQL Server.

Selecting MySQL Database Using PHP Script:

PHP provides function **mysql_select_db** to select a database.

Syntax:

```
mysql_select_db( db_name, connection );
```

Parameter	Description
db_name	Required - MySQL Database name to be selected
connection	Optional - if not specified then last opened connection by mysql_connect will be used.

Example:

Here is the example showing you how to select a database.

```
<?php

$conn = mysql_connect("localhost","user","password");

if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_select_db( 'organization' );
mysql_close($conn);
?>
```

MySQL uses many different data types, broken into three categories: numeric, date and time, and string types.

Numeric Data Types:

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions.

- **INT** - A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** - A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** - A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** - A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** - A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 11 digits.
- **FLOAT(M,D)** - A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
- **DOUBLE(M,D)** - A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** - An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

String Types:

Although numeric and date types are fun, most data you'll store will be in string format. This list describes the common string datatypes in MySQL.

- **CHAR(M)** - A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** - A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). You must define a length when creating a VARCHAR field.
- **BLOB or TEXT** - A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as

images or other types of files. Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.

- **TINYBLOB or TINYTEXT** - A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** - A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** - A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.
- **ENUM** - An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

Date and Time Types:

The MySQL date and time datatypes are:

- **DATE** - A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** - A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** - A timestamp between midnight, January 1, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** - Stores the time in HH:MM:SS format.
- **YEAR(M)** - Stores a year in 2-digit or 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4.

Create MySQL Tables

The table creation command requires:

- Name of the table
- Names of fields
- Definitions for each field

Syntax:

Here is generic SQL syntax to create a MySQL table:

```
CREATE TABLE table_name (column_name column_type);
```

Now we will create following table in **ORGANIZATION** database.

Here few items need explanation:

- Field Attribute **NOT NULL** is being used because we do not want this field to be NULL. SO if user will try to create a record with NULL value then MySQL will raise an error.
- Field Attribute **AUTO_INCREMENT** tells to MySQL to go ahead and add the next available number to the id field.
- Keyword **PRIMARY KEY** is used to define a column as primary key. You can use multiple columns separated by comma to define a primary key.

Creating Tables Using PHP Script:

To create new table in any existing database you would need to use PHP function **mysql_query()**. You will pass its second argument with proper SQL command to create a table.

Example:

Here is an example to create a table using PHP script:

```
<?php
$conn = mysql_connect("localhost","user","password");
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$ctable = "CREATE TABLE employee( ".
       "emp_id INT NOT NULL AUTO_INCREMENT, ".
       "emp_name VARCHAR(100) NOT NULL, ".
       "emp_post VARCHAR(40) NOT NULL, ".
       "join_date DATE, ".
       "PRIMARY KEY ( emp_id )); ";
```

```
mysql_select_db( 'organization' );
$return = mysql_query( $table, $conn );
if(! $retval )
{
  die('Could not create table: ' . mysql_error());
}
echo "Table created successfully\n";
mysql_close($conn);
?>
```

It is very easy to drop an existing MySQL table. But you need to be very careful while deleting any existing table because data lost will not be recovered after deleting a table.

Syntax:

Here is generic SQL syntax to drop a MySQL table:

```
DROP TABLE table_name ;
```

Dropping Tables from Command Prompt:

This needs just to execute **DROP TABLE** SQL command at mysql> prompt.

Example:

Dropping Tables Using PHP Script:

To drop an existing table in any database you would need to use PHP function **mysql_query()**. You will pass its second argument with proper SQL command to drop a table.

Example:

```
<body>
<?php
$conn = mysql_connect("localhost", "user", "password");
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$dtable = "DROP TABLE employee";
```

```
mysql_select_db( 'TUTORIALS' );
$return = mysql_query( $dtable, $conn );
if(! $retval )
{
  die('Could not delete table: ' . mysql_error());
}
echo "Table deleted successfully\n";
mysql_close($conn);
?>
```

To insert data into MySQL table you would need to use SQL **INSERT INTO** command. You can insert data into MySQL table.

Syntax:

Here is generic SQL syntax of INSERT INTO command to insert data into MySQL table:

```
INSERT INTO table_name ( field1, field2,...fieldN )
                       VALUES
                       ( value1, value2,...valueN );
```

To insert string data types it is required to keep all the values into double or single quote, for example:- **"value"**.

Inserting Data Using PHP Script:

You can use same SQL INSERT INTO command into PHP function **mysql_query()** to insert data into a MySQL table.

Example:

This example will take three parameters from user and will insert them into MySQL table:

```
<?php
if(isset($_POST['add']))
{
;
$conn = mysql_connect("localhost","user","password");
if(! $conn )
{
```

```
  die('Could not connect: ' . mysql_error());
}

if(! get_magic_quotes_gpc() )
{
   $tutorial_title = addslashes ($_POST['tutorial_title']);
   $tutorial_author = addslashes ($_POST['tutorial_author']);
}
else
{
   $tutorial_title = $_POST['tutorial_title'];
   $tutorial_author = $_POST['tutorial_author'];
}
$submission_date = $_POST['submission_date'];

$sql = "INSERT INTO tutorials_tbl ".
       "(tutorial_title,tutorial_author, submission_date) ".
       "VALUES ".
       "('$tutorial_title','$tutorial_author','$submission_date')";
mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not enter data: ' . mysql_error());
}
echo "Entered data successfully\n";
mysql_close($conn);
}
else
{
?>
<form method="post" action="<?php $_PHP_SELF ?>">
<table width="600" border="0" cellspacing="1" cellpadding="2">
<tr>
<td width="250">Tutorial Title</td>
<td>
<input name="tutorial_title" type="text" id="tutorial_title">
</td>
</tr>
<tr>
<td width="250">Tutorial Author</td>
<td>
<input name="tutorial_author" type="text" id="tutorial_author">
</td>
</tr>
<tr>
<td width="250">Submission Date [ yyyy-mm-dd ]</td>
<td>
<input name="submission_date" type="text" id="submission_date">
</td>
</tr>
<tr>
<td width="250"> </td>
<td> </td>
</tr>
<tr>
```

```
<td width="250"> </td>
<td>
<input name="add" type="submit" id="add" value="Add Tutorial">
</td>
</tr>
</table>
</form>
<?php
}
?>
</body>
</html>
```

While doing data insert its best practice to use function **get_magic_quotes_gpc()** to check if current configuration for magic quote is set or not. If this function returns false then use function **addslashes()** to add slashes before quotes.

You can put many validations around to check if entered data is correct or not and can take appropriate action.

Here is generic SQL syntax of SELECT command to fetch data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE Clause]
[OFFSET M ][LIMIT N]
```

- You can use one or more tables separated by comma to include various condition using a WHERE clause. But WHERE clause is an optional part of SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (*) in place of fields. In this case SELECT will return all the fields
- You can specify any condition using WHERE clause.
- You can specify an offset using **OFFSET** from where SELECT will start returning records. By default offset is zero
- You can limit the number of returned using **LIMIT** attribute.

Fetching Data from Command Prompt:

This will use SQL SELECT command to fetch data from MySQL table employee

Fetching Data Using PHP Script:

You can use same SQL SELECT command into PHP function **mysql_query()**. This function is used to execute SQL command and later another PHP function **mysql_fetch_array()** can be used to fetch all the selected data. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Below is a simple example to fetch records from **tutorials_tbl** table.

Example:

Try out following example to display all the records from tutorials_tbl table.

```
<?php
$conn = mysql_connect("localhost","user","password");
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}

$show = 'SELECT emp_id, emp_name,
               Join_date FROM employee';

mysql_select_db('organization');
$return = mysql_query( $show, $conn );
if(! $return )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($return, MYSQL_ASSOC))
{
    echo "EMP ID :{$row['emp_id']}  <br> ".
         "NAME: {$row['emp_name']} <br> ".
         "DATE OF JOINING:"Joining Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

The content of the rows are assigned to the variable $row and the values in row are then printed.

NOTE: Always remember to put curly brackets when you want to insert an array value directly into a string.

In above example the constant **MYSQL_ASSOC** is used as the second argument to PHP function **mysql_fetch_array()**, so that it returns the row as an associative array. With an associative array you can access the field by using their name instead of using the index.

PHP provides another function called **mysql_fetch_assoc()** which also returns the row as an associative array.

Example:

Try out following example to display all the records from tutorial_tbl table using mysql_fetch_assoc() function.

```
<?php
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
                tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_assoc($retval))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

You can also use the constant **MYSQL_NUM**, as the second argument to PHP function mysql_fetch_array(). This will cause the function to return an array with numeric index.

Example:

Try out following example to display all the records from tutorials_tbl table using MYSQL_NUM argument.

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
```

```
$sql = 'SELECT tutorial_id, tutorial_title,
                tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_NUM))
{
    echo "Tutorial ID :{$row[0]}  <br> ".
         "Title: {$row[1]} <br> ".
         "Author: {$row[2]} <br> ".
         "Submission Date : {$row[3]} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

All the above three examples will produce same result.

Releasing Memory:

It's a good practice to release cursor memory at the end of each SELECT statement. This can be done by using PHP function **mysql_free_result()**. Below is the example to show how it has to be used.

Example:

Try out following example

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
                tutorial_author, submission_date
        FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
```

```
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_NUM))
{
    echo "Tutorial ID :{$row[0]}  <br> ".
         "Title: {$row[1]} <br> ".
         "Author: {$row[2]} <br> ".
         "Submission Date : {$row[3]} <br> ".
         "--------------------------------<br>";
}
mysql_free_result($retval);
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

While fetching data you can write as complex SQL as you like. Procedure will remain same as mentioned above.

We have seen SQL **SELECT** command to fetch data from MySQL table. We can use a conditional clause called **WHERE** clause to filter out results. Using WHERE clause we can specify a selection criteria to select required records from a table.

Syntax:

Here is generic SQL syntax of SELECT command with WHERE clause to fetch data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE condition1 [AND [OR]] condition2.....
```

- You can use one or more tables separated by comma to include various condition using a WHERE clause. But WHERE clause is an optional part of SELECT command.
- You can specify any condition using WHERE clause.
- You can specify more than one conditions using **AND** or **OR** operators.
- A WHERE clause can be used alongwith DELETE or UPDATE SQL command also to specify a condition.

The **WHERE** clause works like a if condition in any programming language. This clause is used to compare given value with the field value available in MySQl table. If given value from outside is equal to the available field value in MySQL table then it returns that row.

Here is the list of operators which can be used with **WHERE** clause.

Assume field A holds 10 and field B holds 20 then:

Operator	Description	Example
=	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A = B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The WHERE clause is very useful when you want to fetch selected rows from a table, Specially when you use **MySQL Join**. Joins are discussed in another chapter.

It is a common practice to search records using **Primary Key** to make search fast.

If given condition does not match any record in the table then query would not return any row.

Fetching Data Using PHP Script:

You can use same SQL SELECT command with WHERE CLAUSE into PHP function **mysql_query()**. This function is used to execute SQL command and later another PHP function **mysql_fetch_array()** can be used to fetch all the selected data. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Example:

Following example will return all the records from **tutorials_tbl** table for which author name is **Sanjay**:

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
                tutorial_author, submission_date
        FROM tutorials_tbl
        WHERE tutorial_author="Sanjay"';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

There may be a requirement where existing data in a MySQL table need to be modified. You can do so by using SQL **UPDATE** command. This will modify any field value of any MySQL table.

Syntax:

Here is generic SQL syntax of UPDATE command to modify data into MySQL table:

```
UPDATE table_name SET field1=new-value1, field2=new-value2
[WHERE Clause]
```

- You can update one or more field all together.
- You can specify any condition using WHERE clause.

- You can update values in a single table at a time.

Updating Data Using PHP Script:

You can use SQL UPDATE command with or without WHERE CLAUSE into PHP function **mysql_query()**. This function will execute SQL command in similar way it is executed at mysql> prompt.

Example:

Try out following example to update **tutorial_title** field for a record having tutorial_id as 3.

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'UPDATE tutorials_tbl
        SET tutorial_title="Learning JAVA"
        WHERE tutorial_id=3';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not update data: ' . mysql_error());
}
echo "Updated data successfully\n";
mysql_close($conn);
?>
```

If you want to delete a record from any MySQL table then you can use SQL command **DELETE FROM**. You can use this command at mysql> prompt as well as in any script like PHP.

Syntax:

Here is generic SQL syntax of DELETE command to delete data from a MySQL table:

```
DELETE FROM table_name [WHERE Clause]
```

- If WHERE clause is not specified then all the records will be deleted from the given MySQL table.
- You can specify any condition using WHERE clause.
- You can delete records in a single table at a time.

The WHERE clause is very useful when you want to delete selected rows in a table.

Deleting Data Using PHP Script:

You can use SQL DELETE command with or without WHERE CLAUSE into PHP function **mysql_query()**. This function will execute SQL command in similar way it is executed at mysql> prompt.

Example:

Try out following example to delete a record into tutorial_tbl whose tutorial_id is 3.

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'DELETE FROM tutorials_tbl
        WHERE tutorial_id=3';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete data: ' . mysql_error());
}
echo "Deleted data successfully\n";
mysql_close($conn);
?>
```

We have seen SQL **SELECT** command to fetch data from MySQL table. We can also use a conditional clause called **WHERE** clause to select required records.

A WHERE clause with equal sign (=) works fine where we want to do an exact match. Like if "tutorial_author = 'Sanjay'". But there may be a requirement where we want to filter out all the results where tutorial_author name should contain "jay". This can be handled using SQL **LIKE** clause alongwith WHERE clause.

If SQL LIKE clause is used along with % characters then it will work like a meta character (*) in Unix while listing out all the files or directories at command prompt.

Without a % character LIKE clause is very similar to equal sign alongwith WHERE clause.

Syntax:

Here is generic SQL syntax of SELECT command along with LIKE clause to fetch data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'
```

- You can specify any condition using WHERE clause.
- You can use LIKE clause alongwith WHERE clause.
- You can use LIKE clause in place of equal sign.
- When LIKE is used alongwith % sign then it will work like a meta character search.
- You can specify more than one conditions using **AND** or **OR** operators
- A WHERE...LIKE clause can be used alongwith DELETE or UPDATE SQL command also to specify a condition.

Using LIKE clause inside PHP Script:

You can use similar syntax of WHERE...LIKE clause into PHP function **mysql_query()**. This function is used to execute SQL command and later another PHP function **mysql_fetch_array()** can be used to fetch all the selected data if WHERE...LIKE clause is used along with SELECT command.

But if WHERE...LIKE clause is being used with DELETE or UPDATE command then no further PHP function call is required.

Example:

Try out following example to return all the records from **tutorials_tbl** table for which author name contains **jay**:

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
```

```
                tutorial_author, submission_date
        FROM tutorials_tbl
        WHERE tutorial_author LIKE "%jay%"';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

We have seen SQL **SELECT** command to fetch data from MySQL table. When you select rows, the MySQL server is free to return them in any order, unless you instruct it otherwise by saying how to sort the result. But you sort a result set by adding an ORDER BY clause that names the column or columns you want to sort by.

Syntax:

Here is generic SQL syntax of SELECT command along with GROUP BY clause to sort data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
GROUP BY field1, [field2...] [ASC [DESC]]
```

- You can sort returned result on any field provided that filed is being listed out.
- You can sort result on more than one field.
- You can use keyword ASC or DESC to get result in ascending or descending order. By default its ascending order.
- You can use WHERE...LIKE clause in usual way to put condition.

Verify all the author names are listed out in ascending order.

Using ORDER BY clause inside PHP Script:

You can use similar syntax of ORDER BY clause into PHP function **mysql_query()**. This function is used to execute SQL command and later another PHP function **mysql_fetch_array()** can be used to fetch all the selected data.

Example:

Try out following example which returns result in descending order of tutorial author.

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
                tutorial_author, submission_date
        FROM tutorials_tbl
        ORDER BY  tutorial_author DESC';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

Thus far we have only been getting data from one table at a time. This is fine for simple takes, but in most real world MySQL usage you will often need to get data from multiple tables in a single query.

You can use multiple tables in your single SQL query. The act of joining in MySQL refers to smashing two or more tables into a single table.

You can use JOINS in SELECT, UPDATE and DELETE statements to join MySQL tables. We will see an example of LEFT JOIN also which is different from simple MySQL JOIN.

Using Joins in PHP Script:

You can use any of the above mentioned SQL query in PHP script. You only need to pass SQL query into PHP function **mysql_query()** and then you will fetch results in usual way.

Example:

Try out following example:

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT a.tutorial_id, a.tutorial_author, b.tutorial_count
        FROM tutorials_tbl a, tcount_tbl b
        WHERE a.tutorial_author = b.tutorial_author';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Author:{$row['tutorial_author']}  <br> ".
         "Count: {$row['tutorial_count']} <br> ".
         "Tutorial ID: {$row['tutorial_id']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

MySQL LEFT JOIN:

A MySQL left join is different from a simple join. A MySQL LEFT JOIN gives extra consideration to the table that is on the left.

If I do a LEFT JOIN, I get all records that match in the same way and IN ADDITION I get an extra record for each unmatched record in the left table of the join - thus ensuring (in my example) that every AUTHOR gets a mention:

Example:

Try out following example to understand LEFT JOIN:

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> SELECT a.tutorial_id, a.tutorial_author, b.tutorial_count
    -> FROM tutorials_tbl a LEFT JOIN tcount_tbl b
    -> ON a.tutorial_author = b.tutorial_author;
+-------------+-----------------+----------------+
| tutorial_id | tutorial_author | tutorial_count |
+-------------+-----------------+----------------+
|           1 | John Poul       |              1 |
|           2 | Abdul S         |           NULL |
|           3 | Sanjay          |              1 |
+-------------+-----------------+----------------+
3 rows in set (0.02 sec)
```

You would need to do more practice to become familiar with JOINS. This is a but complex concept in MySQL/SQL and will become more clear while doing real examples.

We have seen SQL **SELECT** command along with **WHERE** clause to fetch data from MySQL table. But when we try to give a condition which compare field or column value to **NULL** it does not work properly.

To handle such situation MySQL provides three operators

- **IS NULL:** operator returns true of column value is NULL.
- **IS NOT NULL:** operator returns true of column value is not NULL.
- <=> operator compare values, which (unlike the = operator) is true even for two NULL values

Conditions involving NULL are special. You cannot use = NULL or != NULL to look for NULL values in columns. Such comparisons always fail because it's impossible to tell whether or not they are true. Even NULL = NULL fails.

To look for columns that are or are not NULL, use IS NULL or IS NOT NULL.

Handling NULL Values in PHP Script:

You can use *if...else* condition to prepare a query based on NULL value.

Example:

Following example take tutorial_count from outside and then compare it with the value available in the table.

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
if( isset($tutorial_count ))
{
   $sql = 'SELECT tutorial_author, tutorial_count
           FROM  tcount_tbl
           WHERE tutorial_count = $tutorial_count';
}
else
{
   $sql = 'SELECT tutorial_author, tutorial_count
           FROM  tcount_tbl
           WHERE tutorial_count IS $tutorial_count';
}

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Author:{$row['tutorial_author']}  <br> ".
         "Count: {$row['tutorial_count']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

You have seen MySQL pattern matching with **LIKE ...%**. MySQL supports another type of pattern matching operation based on regular expressions and the **REGEXP** operator. If you are aware of PHP or PERL then its very simple for you to understand because this matching is very similar to those scripting regular expressions.

Following is the table of pattern which can be used along with **REGEXP** operator.

Pattern	What the pattern matches

^	Beginning of string
$	End of string
.	Any single character
[...]	Any character listed between the square brackets
[^...]	Any character not listed between the square brackets
p1|p2|p3	Alternation; matches any of the patterns p1, p2, or p3
*	Zero or more instances of preceding element
+	One or more instances of preceding element
{n}	n instances of preceding element
{m,n}	m through n instances of preceding element

Examples:

Now based on above table you can device various type of SQL queries to meet your requirements. Here I'm listing few for your understanding. Consider we have a table called person_tbl and its having a field called name:

Query to find all the names starting with 'st'

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP '^st';
```

Query to find all the names ending with 'ok'

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP 'ok$';
```

Query to find all the names which contains 'mar'

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP 'mar';
```

Query to find all the names starting with a vowel and ending with 'ok'

```
mysql> SELECT name FROM person_tbl WHERE name REGEXP '^[aeiou]|ok$';
```

A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit. In other words, a transaction will never be complete unless each individual operation within the group is successful. If any operation within the transaction fails, the entire transaction will fail.

Practically you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transactions:

Transactions have the following four standard properties, usually referred to by the acronym ACID:

- **Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
- **Consistency:** ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation:** enables transactions to operate independently of and transparent to each other.
- **Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

In MySQL, transactions begin with the statement BEGIN WORK and end with either a COMMIT or a ROLLBACK statement. The SQL commands between the beginning and ending statements form the bulk of the transaction.

COMMIT and ROLLBACK:

These two keywords **Commit** and **Rollback** are mainly used for MySQL Transactions.

- When a successful transaction is completed, the COMMIT command should be issued so that the changes to all involved tables will take effect.
- If a failure occurs, a ROLLBACK command should be issued to return every table referenced in the transaction to its previous state.

You can control the behavior of a transaction by setting session variable called **AUTOCOMMIT**. If AUTOCOMMIT is set to 1 (the default), then each SQL statement (within a transaction or not) is considered a complete transaction, and committed by default when it finishes. When AUTOCOMMIT is set to 0, by issuing the SET AUTOCOMMIT=0 command, the subsequent series of statements acts like a transaction, and no activities are committed until an explicit COMMIT statement is issued.

You can execute these SQL commands in PHP by using **mysql_query()** function.

Generic Example on Transaction

This sequence of events is independent of the programming language used; the logical path can be created in whichever language you use to create your application.

You can execute these SQL commands in PHP by using **mysql_query()** function.

1. Begin transaction by issuing SQL command **BEGIN WORK**
2. Issue one or more SQL commands like SELECT, INSERT, UPDATE or DELETE
3. Check if there is no error and everything is according to your requirement.
4. If there is any error then issue ROLLBACK command otherwise issue a COMMIT command.

Transaction-Safe Table Types in MySQL:

You can not use transactions directly, you can but they would not be save and guaranteed. If you plan to use transactions in your MySQL programming then you need to create your tables in a special way. There are many type of tables which support transactions but most popular one is **InnoDB**.

Support for InnoDB tables requires a specific compilation parameter when compiling MySQL from source. If your MySQL version does not have InnoDB support, ask your Internet Service Provider to build a version of MySQL with support for InnoDB table types, or download and install the MySQL-Max binary distribution for Windows or Linux/UNIX and work with the table type in a development environment.

If your MySQL installation supports InnoDB tables, simply add a **TYPE=InnoDB** definition to the table creation statement. For example, the following code creates an InnoDB table called tcount_tbl:

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> create table tcount_tbl
    -> (
    -> tutorial_author varchar(40) NOT NULL,
    -> tutorial_count  INT
    -> ) TYPE=InnoDB;
Query OK, 0 rows affected (0.05 sec)
```

You can use other table type like **GEMINI** or **BDB** but it depends on your installation if it supports these two types.

MySQL **ALTER** command is very useful when you want to change a name of your table, any table field or if you want to add or delete an existing column in a table.

Lets begin with creation of a table called **testalter_tbl**

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> create table testalter_tbl
    -> (
    -> i INT,
    -> c CHAR(1)
    -> );
Query OK, 0 rows affected (0.05 sec)
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| c     | char(1) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Dropping, Adding, or Repositioning a Column:

Suppose you want to drop an existing column **i** from above MySQL table then you will use **DROP** clause along with **ALTER** command as follows

```
mysql> ALTER TABLE testalter_tbl  DROP i;
```

A **DROP** will not work if the column is the only one left in the table.

To add a column, use ADD and specify the column definition. The following statement restores the **i** column to testalter_tbl

```
mysql> ALTER TABLE testalter_tbl ADD i INT;
```

After issuing this statement, testalter will contain the same two columns that it had when you first created the table, but will not have quite the same structure. That's because new columns are added to the end of the table by default. So even though **i** originally was the first column in mytbl, now it is the last one:

```
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

To indicate that you want a column at a specific position within the table, either use FIRST to make it the first column, or AFTER col_name to indicate that the new column should be placed after col_name. Try the following ALTER TABLE statements, using SHOW COLUMNS after each one to see what effect each one has:

```
ALTER TABLE testalter_tbl DROP i;
ALTER TABLE testalter_tbl ADD i INT FIRST;
ALTER TABLE testalter_tbl DROP i;
ALTER TABLE testalter_tbl ADD i INT AFTER c;
```

The FIRST and AFTER specifiers work only with the ADD clause. This means that if you want to reposition an existing column within a table, you first must DROP it and then ADD it at the new position.

Changing a Column Definition or Name:

To change a column's definition, use **MODIFY** or **CHANGE** clause along with ALTER command. For example, to change column **c** from CHAR(1) to CHAR(10), do this:

```
mysql> ALTER TABLE testalter_tbl MODIFY c CHAR(10);
```

With CHANGE, the syntax is a bit different. After the CHANGE keyword, you name the column you want to change, then specify the new definition, which includes the new name. Try out following example:

```
mysql> ALTER TABLE testalter_tbl CHANGE i j BIGINT;
```

If you now use CHANGE to convert j from BIGINT back to INT without changing the column name, the statement be as expected:

```
mysql> ALTER TABLE testalter_tbl CHANGE j j INT;
```

The Effect of ALTER TABLE on Null and Default Value Attributes:

When you MODIFY or CHANGE a column, you can also specify whether or not the column can contain NULL values, and what its default value is. In fact, if you don't do this, MySQL automatically assigns values for these attributes.

Here is the example where NOT NULL column will have value 100 by default.

```
mysql> ALTER TABLE testalter_tbl
    -> MODIFY j BIGINT NOT NULL DEFAULT 100;
```

If you don't use above command then MySQL will fill up NULL values in all the columns.

Changing a Column's Default Value:

You can change a default value for any column using ALTER command. Try out following example.

```
mysql> ALTER TABLE testalter_tbl ALTER i SET DEFAULT 1000;
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | 1000    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

You can remove default constraint from any column by using DROP clause along with ALTER command.

```
mysql> ALTER TABLE testalter_tbl ALTER i DROP DEFAULT;
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Changing a Table Type:

You can use a table type by using **TYPE** clause alongwith ALTER command. Try out following example to change testalter_tbl to **MYISAM** table type.

To find out the current type of a table, use the SHOW TABLE STATUS statement.

```
mysql> ALTER TABLE testalter_tbl TYPE = MYISAM;
mysql>  SHOW TABLE STATUS LIKE 'testalter_tbl'\G
*************************** 1. row ***************
           Name: testalter_tbl
           Type: MyISAM
     Row_format: Fixed
           Rows: 0
 Avg_row_length: 0
    Data_length: 0
Max_data_length: 25769803775
   Index_length: 1024
```

```
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2007-06-03 08:04:36
    Update_time: 2007-06-03 08:04:36
     Check_time: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

Renaming a Table:

To rename a table, use the **RENAME** option of the ALTER TABLE statement. Try out following example to rename testalter_tbl to alter_tbl

```
mysql> ALTER TABLE testalter_tbl RENAME TO alter_tbl;
```

You can use ALTER command to create and drop INDEX on a MySQL file. We will see this feature in next chapter.

A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

While creating index it should be considered that what are the columns which will be used to make SQL queries and create one or more indexes on those columns.

Practically, Indexes are also type of tables which keeps primary key or index field and a pointer to each record in to the actual table.

The users cannot see the indexes, they are just used to speed up queries and will be used by Database Search Engine to locate records very fast.

INSERT and UPDATE statements takes more time on tables having indexes where as SELECT statements become fast on those tables. The reason is that while doing insert or update, database need to inert or update index values as well.

Simple and Unique Index:

You can create a unique index on a table. A unique index means that two rows cannot have the same index value. Here is the syntax to create an Index on a table

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...);
```

You can use one or more columns to create an index. For example we can create an index on tutorials_tbl using tutorial_author

```
CREATE UNIQUE INDEX AUTHOR_INDEX
ON tutorials_tbl (tutorial_author)
```

You can creates a simple index on a table. Just omit UNIQUE keyword from the query to create simple index. Simple index allows duplicate values in a table.

If you want to index the values in a column in descending order, you can add the reserved word DESC after the column name:

```
mysql> CREATE UNIQUE INDEX AUTHOR_INDEX
ON tutorials_tbl (tutorial_author DESC)
```

ALTER command to add and drop INDEX:

There are four types of statements for adding indexes to a table:

- **ALTER TABLE tbl_name ADD PRIMARY KEY (column_list)** : This statement adds a PRIMARY KEY, which means that indexed values must be unique and cannot be NULL.
- **ALTER TABLE tbl_name ADD UNIQUE index_name (column_list):** This statement creates an index for which values must be unique (with the exception of NULL values, which may appear multiple times).
- **ALTER TABLE tbl_name ADD INDEX index_name (column_list):** This adds an ordinary index in which any value may appear more than once.
- **ALTER TABLE tbl_name ADD FULLTEXT index_name (column_list):** This creates a special FULLTEXT index that is used for text-searching purposes.

Here is the example to add index in an existing table.

```
mysql> ALTER TABLE testalter_tbl ADD INDEX (c);
```

You can drop any INDEX by using DROP clause along with ALTER command. Try out following example to drop above created index.

```
mysql> ALTER TABLE testalter_tbl DROP INDEX (c);
```

You can drop any INDEX by using DROP clause along with ALTER command. Try out following example to drop above created inde x.

ALTER Command to add and drop PRIMARY KEY:

You can add primary key as well in the same way. But make sure Primary Key works on columns which are NOT NULL.

Here is the example to add primary key in an existing table. This will make a column NOT NULL first and then add it as a primary key.

```
mysql> ALTER TABLE testalter_tbl MODIFY i INT NOT NULL;
mysql> ALTER TABLE testalter_tbl ADD PRIMARY KEY (i);
```

You can use ALTER command to drop a primary key as follows:

```
mysql> ALTER TABLE testalter_tbl DROP PRIMARY KEY;
```

To drop an index that is not a PRIMARY KEY, you must specify the index name.

Displaying INDEX Information:

You can use SHOW INDEX command to list out all the indexes associated with a table. Vertical-format output (specified by \G) often is useful with this statement, to avoid long line wraparound:

Try out following example:

```
mysql> SHOW INDEX FROM table_name\G
........
```

The temporary tables could be very useful in some cases to keep temporary data. The most important thing that should be knows for temporary tables is that they will be deleted when the current client session terminates.

Temporary tables where added in MySQL version 3.23. If you use an older version of MySQL than 3.23 you can't use temporary tables, but you can use heap tables.

As stated earlier temporary tables will only last as long as the session is alive. If you run the code in a PHP script, the temporary table will be destroyed automatically when the script finishes executing. If you are connected to the MySQl database server through the MySQL client program, then the temporary table will exist until you close the client or manually destroy the table.

Example

Here is an example showing you usage of temporary table. Same code can be used in PHP scripts using **mysql_query()** function.

```
mysql> CREATE TEMPORARY TABLE SalesSummary (
    -> product_name VARCHAR(50) NOT NULL
    -> , total_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00
    -> , avg_unit_price DECIMAL(7,2) NOT NULL DEFAULT 0.00
    -> , total_units_sold INT UNSIGNED NOT NULL DEFAULT 0
);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SalesSummary
    -> (product_name, total_sales, avg_unit_price, total_units_sold)
    -> VALUES
    -> ('cucumber', 100.25, 90, 2);

mysql> SELECT * FROM SalesSummary;
+--------------+-------------+----------------+------------------+
| product_name | total_sales | avg_unit_price | total_units_sold |
+--------------+-------------+----------------+------------------+
| cucumber     |      100.25 |          90.00 |                2 |
+--------------+-------------+----------------+------------------+
1 row in set (0.00 sec)
```

When you issue a SHOW TABLES command then your temporary table would not be listed out in the list. Now if you will log out of the MySQL session and then you will issue a SELECT command then you will find no data available in the database. Even your temporary table would also not exist.

Dropping Temporary Tables:

By default all the temporary tables are deleted by MySQL when your database connection gets terminated. Still you want to delete them in between then you do so by issuing DROP TABLE command.

Following is the example on dropping a temproary table.

```
mysql> CREATE TEMPORARY TABLE SalesSummary (
    -> product_name VARCHAR(50) NOT NULL
    -> , total_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00
    -> , avg_unit_price DECIMAL(7,2) NOT NULL DEFAULT 0.00
    -> , total_units_sold INT UNSIGNED NOT NULL DEFAULT 0
);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SalesSummary
    -> (product_name, total_sales, avg_unit_price, total_units_sold)
    -> VALUES
    -> ('cucumber', 100.25, 90, 2);
```

```
mysql> SELECT * FROM SalesSummary;
+--------------+-------------+----------------+------------------+
| product_name | total_sales | avg_unit_price | total_units_sold |
+--------------+-------------+----------------+------------------+
| cucumber     |      100.25 |          90.00 |                2 |
+--------------+-------------+----------------+------------------+
1 row in set (0.00 sec)
mysql> DROP TABLE SalesSummary;
mysql>  SELECT * FROM SalesSummary;
ERROR 1146: Table 'TUTORIALS.SalesSummary' doesn't exist
```

There may be a situation when you need an exact copy of a table, and CREATE TABLE ... SELECT doesn't suit your purposes because the copy must include the same indexes, default values, and so forth.

You can handle this situation by following steps.

1. Use SHOW CREATE TABLE to get a CREATE TABLE statement that specifies the source table's structure, indexes and all.
2. Modify the statement to change the table name to that of the clone table and execute the statement. This way you will have exact clone table.
3. Optionally, If you need the table contents copied as well, issue an INSERT INTO ... SELECT statement, too.

Example:

Try out following example to create a clone table for **tutorials_tbl**

Step 1:

Get complete structure about table

```
mysql> SHOW CREATE TABLE tutorials_tbl \G;
*************************** 1. row ***************************
       Table: tutorials_tbl
Create Table: CREATE TABLE `tutorials_tbl` (
  `tutorial_id` int(11) NOT NULL auto_increment,
  `tutorial_title` varchar(100) NOT NULL default '',
  `tutorial_author` varchar(40) NOT NULL default '',
  `submission_date` date default NULL,
  PRIMARY KEY  (`tutorial_id`),
  UNIQUE KEY `AUTHOR_INDEX` (`tutorial_author`)
) TYPE=MyISAM
1 row in set (0.00 sec)

ERROR:
```

```
No query specified
```

Step 2:

Rename this table and create another table

```
mysql> CREATE TABLE `clone_tbl` (
  -> `tutorial_id` int(11) NOT NULL auto_increment,
  -> `tutorial_title` varchar(100) NOT NULL default '',
  -> `tutorial_author` varchar(40) NOT NULL default '',
  -> `submission_date` date default NULL,
  -> PRIMARY KEY  (`tutorial_id`),
  -> UNIQUE KEY `AUTHOR_INDEX` (`tutorial_author`)
-> ) TYPE=MyISAM;
Query OK, 0 rows affected (1.80 sec)
```

Step 3:

After executing step 2 you will a clone table in your database. If you want to copy data from old table then you can do it by using INSERT INTO... SELECT statement.

```
mysql> INSERT INTO clone_tbl (tutorial_id,
    ->                        tutorial_title,
    ->                        tutorial_author,
    ->                        submission_date)
    -> SELECT tutorial_id,tutorial_title,
    ->        tutorial_author,submission_date,
    -> FROM tutorials_tbl;
Query OK, 3 rows affected (0.07 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Finally you will have exact clone table as you wanted to have.

There are three information which you would like to have from MySQL.

- **Information about the result of queries:** This includes number of records effected by any SELECT, UPDATE or DELETE statement.
- **Information about tables and databases:** This includes information pertaining to the structure of tables and databases.
- **Information about the MySQL server:** This includes current status of database server, version number etc.

Its very easy to get all these information at mysql prompt. BUt while using PERL or PHP APIs then we need to call various APIs explicitely to obtain all these information. Following section will show you how to obtain these information.

Obtaining the Number of Rows Affected by a Query:

PERL Example:

In DBI scripts, the affected-rows count is returned by do( ) or by execute( ), depending on how you execute the query:

```
# Method 1
# execute $query using do( )
my $count = $dbh->do ($query);
# report 0 rows if an error occurred
printf "%d rows were affected\n", (defined ($count) ? $count : 0);

# Method 2
# execute query using prepare( ) plus execute( )
my $sth = $dbh->prepare ($query);
my $count = $sth->execute ( );
printf "%d rows were affected\n", (defined ($count) ? $count : 0);
```

PHP Example:

In PHP, invoke the mysql_affected_rows( ) function to find out how many rows a query changed:

```
$result_id = mysql_query ($query, $conn_id);
# report 0 rows if the query failed
$count = ($result_id ? mysql_affected_rows ($conn_id) : 0);
print ("$count rows were affected\n");
```

Listing Tables and Databases:

This is very easy to list down all the databases and tables available with database server. Your result may be null if you don't have sufficient privilege.

Apart from the method I have mentioned below you can use SHOW TABLES or SHOW DATABASES queries to get list of tables or databases either in PHP or in PERL.

PERL Example:

```
# Get all the tables available in current database.
my @tables = $dbh->tables ( );
foreach $table (@tables ){
   print "Table Name $table\n";
}
```

PHP Example:

```
<?php
```

```
$con = mysql_connect("localhost", "userid", "password");
if (!$con)
{
  die('Could not connect: ' . mysql_error());
}

$db_list = mysql_list_dbs($con);

while ($db = mysql_fetch_object($db_list))
{
  echo $db->Database . "<br />";
}
mysql_close($con);
?>
```

Getting Server Metadata:

There are following commands in MySQL which can be executed either are mysql prompt or using any script like PHP to get various important information about database server.

Command	Description
SELECT VERSION( )	Server version string
SELECT DATABASE( )	Current database name (empty if none)
SELECT USER( )	Current username
SHOW STATUS	Server status indicators
SHOW VARIABLES	Server configuration variables

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value, and sequences provide an easy way to generate them. This chapter describes how to use sequences in MySQL.

Using AUTO_INCREMENT column:

The simplest way in MySQL to use Sequences is to define a column as AUTO_INCREMENT and leave rest of the things to MySQL to take care.

Example:

Try out following example. This will create table and after that it will insert few rows in this table where it is not required to give record ID because its auto incremented by MySQL.

```
mysql> CREATE TABLE insect
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> PRIMARY KEY (id),
    -> name VARCHAR(30) NOT NULL, # type of insect
    -> date DATE NOT NULL, # date collected
    -> origin VARCHAR(30) NOT NULL # where collected
);
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO insect (id,name,date,origin) VALUES
    -> (NULL,'housefly','2001-09-10','kitchen'),
    -> (NULL,'millipede','2001-09-10','driveway'),
    -> (NULL,'grasshopper','2001-09-10','front yard');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM insect ORDER BY id;
+----+-------------+------------+------------+
| id | name        | date       | origin     |
+----+-------------+------------+------------+
|  1 | housefly    | 2001-09-10 | kitchen    |
|  2 | millipede   | 2001-09-10 | driveway   |
|  3 | grasshopper | 2001-09-10 | front yard |
+----+-------------+------------+------------+
3 rows in set (0.00 sec)
```

Obtain AUTO_INCREMENT Values:

LAST_INSERT_ID( ) is a SQL function, so you can use it from within any client that understands how to issue SQL statements. otherwise PERL and PHH scripts provide exclusive functions to retrieve auto incremented value of last record.

PERL Example:

Use the mysql_insertid attribute to obtain the AUTO_INCREMENT value generated by a query. This attribute is accessed through either a database handle or a statement handle, depending on how you issue the query. The following example references it through the database handle:

```
$dbh->do ("INSERT INTO insect (name,date,origin)
VALUES('moth','2001-09-14','windowsill')");
my $seq = $dbh->{mysql_insertid};
```

PHP Example:

After issuing a query that generates an AUTO_INCREMENT value, retrieve the value by calling mysql_insert_id( ):

```
mysql_query ("INSERT INTO insect (name,date,origin)
VALUES('moth','2001-09-14','windowsill')", $conn_id);
$seq = mysql_insert_id ($conn_id);
```

Renumbering an Existing Sequence:

There may be a case when you have deleted many records from a table and you want to resequence all the records. This can be done by using a simple trick but you should be very careful to do so if your table is having join with other table.

If you determine that resequencing an AUTO_INCREMENT column is unavoidable, the way to do it is to drop the column from the table, then add it again. The following example shows how to renumber the id values in the insect table using this technique:

```
mysql> ALTER TABLE insect DROP id;
mysql> ALTER TABLE insect
    -> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (id);
```

Starting a Sequence at a Particular Value:

By default MySQL will start sequence from 1 but you can specify any other number as well at the time of table creation. Following is the example where MySQL will start sequence from 100.

```
mysql> CREATE TABLE insect
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT = 100,
    -> PRIMARY KEY (id),
    -> name VARCHAR(30) NOT NULL, # type of insect
    -> date DATE NOT NULL, # date collected
    -> origin VARCHAR(30) NOT NULL # where collected
);
```

Alternatively, you can create the table and then set the initial sequence value with ALTER TABLE.

```
mysql> ALTER TABLE t AUTO_INCREMENT = 10
```

Tables or result sets sometimes contain duplicate records. Sometime it is allowed but sometime it is required to stop duplicate records. Sometime it is required to identify duplicate records and remove them from the table. This chapter will describe how to prevent duplicate records occurring in a table and how to remove already existing duplicate records.

Preventing Duplicates from Occurring in a Table:

You can use a **PRIMARY KEY** or **UNIQUE** Index on a table with appropriate fields to stop duplicate records. Lets take one example, The following table contains no such index or primary key, so it would allow duplicate records for first_name and last_name

```
CREATE TABLE person_tbl
(
    first_name CHAR(20),
    last_name CHAR(20),
    sex CHAR(10)
);
```

To prevent multiple records with the same first and last name values from being created in this table, add a PRIMARY KEY to its definition. When you do this, it's also necessary to declare the indexed columns to be NOT NULL, because a PRIMARY KEY does not allow NULL values:

```
CREATE TABLE person_tbl
(
   first_name CHAR(20) NOT NULL,
   last_name CHAR(20) NOT NULL,
   sex CHAR(10)
   PRIMARY KEY (last_name, first_name)
);
```

The presence of a unique index in a table normally causes an error to occur if you insert a record into the table that duplicates an existing record in the column or columns that define the index.

Use **INSERT IGNORE** rather than **INSERT**. If a record doesn't duplicate an existing record, MySQL inserts it as usual. If the record is a duplicate, the IGNORE keyword tells MySQL to discard it silently without generating an error.

Following example does not error out and same time it will not insert duplicate records.

```
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Jay', 'Thomas');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Jay', 'Thomas');
Query OK, 0 rows affected (0.00 sec)
```

Use **REPLACE** rather than INSERT. If the record is new, it's inserted just as with INSERT. If it's a duplicate, the new record replaces the old one:

```
mysql> REPLACE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Ajay', 'Kumar');
Query OK, 1 row affected (0.00 sec)
mysql> REPLACE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Ajay', 'Kumar');
Query OK, 2 rows affected (0.00 sec)
```

INSERT IGNORE and REPLACE should be chosen according to the duplicate-handling behavior you want to effect. INSERT IGNORE keeps the first of a set of duplicated records and discards the rest. REPLACE keeps the last of a set of duplicates and erase out any earlier ones.

Another way to enforce uniqueness is to add a UNIQUE index rather than a PRIMARY KEY to a table.

```
CREATE TABLE person_tbl
(
   first_name CHAR(20) NOT NULL,
   last_name CHAR(20) NOT NULL,
   sex CHAR(10)
   UNIQUE (last_name, first_name)
);
```

Counting and Identifying Duplicates:

Following is the query to count duplicate records with first_name and last_name in a table.

```
mysql> SELECT COUNT(*) as repetitions, last_name, first_name
    -> FROM person_tbl
    -> GROUP BY last_name, first_name
    -> HAVING repetitions > 1;
```

This query will return a list of all the duplicate records in person_tbl table. In general, to identify sets of values that are duplicated, do the following:

- Determine which columns contain the values that may be duplicated.
- List those columns in the column selection list, along with COUNT(*).
- List the columns in the GROUP BY clause as well.
- Add a HAVING clause that eliminates unique values by requiring group counts to be greater than one.

Eliminating Duplicates from a Query Result:

You can use **DISTINCT** along with SELECT statement to find out unique records available in a table.

```
mysql> SELECT DISTINCT last_name, first_name
    -> FROM person_tbl
    -> ORDER BY last_name;
```

An alternative to DISTINCT is to add a GROUP BY clause that names the columns you're selecting. This has the effect of removing duplicates and selecting only the unique combinations of values in the specified columns:

```
mysql> SELECT last_name, first_name
    -> FROM person_tbl
    -> GROUP BY (last_name, first_name);
```

Removing Duplicates Using Table Replacement:

If you have duplicate records in a table and you want to remove all the duplicate records from that table then here is the procedure.

```
mysql> CREATE TABLE tmp SELECT last_name, first_name, sex
    ->                  FROM person_tbl;
    ->                  GROUP BY (last_name, first_name);
mysql> DROP TABLE person_tbl;
mysql> ALTER TABLE tmp RENAME TO person_tbl;
```

An easy way of removing duplicate records from a table is that add an INDEX or PRIMAY KEY to that table. Even if this table is already available you can use this technique to remove duplicate records and you will be safe in future as well.

```
mysql> ALTER IGNORE TABLE person_tbl
    -> ADD PRIMARY KEY (last_name, first_name)
```

If you take user input through a webpage and insert it into a MySQL database there's a chance that you have left yourself wide open for a security issue known as SQL Injection. This lesson will teach you how to help prevent this from happening and help you secure your scripts and MySQL statements.

Injection usually occurs when you ask a user for input, like their name, and instead of a name they give you a MySQL statement that you will unknowingly run on your database.

Never trust user provided data, process this data only after validation; as a rule, this is done by pattern matching. In the example below, the username is restricted to alphanumerical chars plus underscore and to a length between 8 and 20 chars - modify these rules as needed.

```
if (preg_match("/^\w{8,20}$/", $_GET['username'], $matches))
{
   $result = mysql_query("SELECT * FROM users
                          WHERE username=$matches[0]");
}
 else
{
   echo "username not accepted";
}
```

To demonstrate the problem, consider this excerpt:

```
// supposed input
$name = "Qadir'; DELETE FROM users;";
mysql_query("SELECT * FROM users WHERE name='{$name}'");
```

The function call is supposed to retrieve a record from the users table where the name column matches the name specified by the user. Under normal circumstances, $name would only contain alphanumeric characters and perhaps spaces, such as the string ilia. But here, by appending an entirely new query to $name, the call to the database turns into disaster: the injected DELETE query removes all records from users.

Fortunately, if you use MySQL, the mysql_query() function does not permit query stacking, or executing multiple queries in a single function call. If you try to stack queries, the call fails.

However, other PHP database extensions, such as SQLite and PostgreSQL, happily perform stacked queries, executing all of the queries provided in one string and creating a serious security problem.

Preventing SQL Injection:

You can handle all escape characters smartly in scripting languages like PERL and PHP. The MySQL extension for PHP provides the function mysql_real_escape_string() to escape input characters that are special to MySQL.

```
if (get_magic_quotes_gpc())
{
  $name = stripslashes($name);
}
$name = mysql_real_escape_string($name);
mysql_query("SELECT * FROM users WHERE name='{$name}'");
```

The LIKE Quandary:

To address the LIKE quandary, a custom escaping mechanism must convert user-supplied % and _ characters to literals. Use addcslashes(), a function that let's you specify a character range to escape.

```
$sub = addcslashes(mysql_real_escape_string("%something_"), "%_");
// $sub == \%something\_
mysql_query("SELECT * FROM messages WHERE subject LIKE '{$sub}%'");
```

The simplest way of exporting a table data into a text file is using SELECT...INTO OUTFILE statement that exports a query result directly into a file on the server host.

Exporting Data with the SELECT ... INTO OUTFILE Statement:

The syntax for this statement combines a regular SELECT with INTO OUTFILE *filename* at the end. The default output format is the same as for LOAD DATA, so the following statement exports the tutorials_tbl table into /tmp/tutorials.txt as a tab-delimited, linefeed-terminated file:

```
mysql> SELECT * FROM tutorials_tbl
    -> INTO OUTFILE '/tmp/tutorials.txt';
```

You can change the output format using options to indicate how to quote and delimit columns and records. To export the tutorial_tbl table in CSV format with CRLF-terminated lines, use this statement:

```
mysql> SELECT * FROM passwd INTO OUTFILE '/tmp/tutorials.txt'
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

The **SELECT ... INTO OUTFILE** has the following properties:

- The output file is created directly by the MySQL server, so the filename should indicate where you want the file to be written on the server host. There is no LOCAL version of the statement analogous to the LOCAL version of LOAD DATA.
- You must have the MySQL FILE privilege to execute the SELECT ... INTO statement.
- The output file must not already exist. This prevents MySQL from clobbering files that may be important.
- You should have a login account on the server host or some way to retrieve the file from that host. Otherwise, SELECT ... INTO OUTFILE likely will be of no value to you.

- Under Unix, the file is created world readable and is owned by the MySQL server. This means that although you'll be able to read the file, you may not be able to delete it.

Exporting Tables as Raw Data:

The ***mysqldump*** program is used to copy or back up tables and databases. It can write table output either as a raw datafile, or as a set of INSERT statements that recreate the records in the table.

To dump a table as a datafile, you must specify a --tab option that indicates the directory where you want the MySQL server to write the file.

For example, to dump the tutorials_tbl table from the TUTORIALS database to a file in the /tmp directory, use a command like this:

```
$ mysqldump -u root -p --no-create-info \
             --tab=/tmp TUTORIALS tutorials_tbl
password ******
```

Exporting Table Contents or Definitions in SQL Format:

To export a table in SQL format to a file, use a command like this:

```
$ mysqldump -u root -p TUTORIALS tutorials_tbl > dump.txt
password ******
```

This will create file having content as follows:

```
-- MySQL dump 8.23
--
-- Host: localhost    Database: TUTORIALS
---------------------------------------------------------
-- Server version       3.23.58

--
-- Table structure for table `tutorials_tbl`
--

CREATE TABLE tutorials_tbl (
  tutorial_id int(11) NOT NULL auto_increment,
  tutorial_title varchar(100) NOT NULL default '',
  tutorial_author varchar(40) NOT NULL default '',
  submission_date date default NULL,
  PRIMARY KEY  (tutorial_id),
  UNIQUE KEY AUTHOR_INDEX (tutorial_author)
```

```
) TYPE=MyISAM;

--
-- Dumping data for table `tutorials_tbl`
--

INSERT INTO tutorials_tbl
       VALUES (1,'Learn PHP','John Poul','2007-05-24');
INSERT INTO tutorials_tbl
       VALUES (2,'Learn MySQL','Abdul S','2007-05-24');
INSERT INTO tutorials_tbl
       VALUES (3,'JAVA Tutorial','Sanjay','2007-05-06');
```

To dump multiple tables, name them all following the database name argument. To dump an entire database, don't name any tables after the database as follows:

```
$ mysqldump -u root -p TUTORIALS > database_dump.txt
password ******
```

To backup all the databases available on your host use the following:

```
$ mysqldump -u root -p --all-databases > database_dump.txt
password ******
```

The --all-databases option is available as of MySQL 3.23.12.

These method can be used to implement a database backup stretegy.

Copying Tables or Databases to Another Host:

If you want to copy tables or databases from one MySQL server to another then use mysqldump with database name and table name.

Run the following command at source host. This will dump complete database into dump.txt file:

```
$ mysqldump -u root -p database_name table_name > dump.txt
password *****
```

You can copy complete database without using a particular table name as explained above.

Now ftp dump.txt file on another host and use the following command. Before running this command, make sure you have created database_name on destination server.

```
$ mysql -u root -p database_name < dump.txt
password *****
```

Another way to accomplish this without using an intermediary file is to send the output of mysqldump directly over the network to the remote MySQL server. If you can connect to both servers from the host where the cookbook database resides, use this command:

```
$ mysqldump -u root -p database_name \
            states | mysql -h other-host.com database_name
```

The mysqldump half of the command connects to the local server and writes the dump output to the pipe. The mysql half of the command connects to the remote MySQL server on otherhost.com. It reads the pipe for input and sends each statement to the other-host.com server.

There are two simple ways in MySQL to load data into MySQL database from a previously backed up file.

Importing Data with LOAD DATA:

MySQL provides a LOAD DATA statement that acts as a bulk data loader. Here's an example statement that reads a file dump.txt from your current directory and loads it into the table mytbl in the current database:

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl;
```

- If the LOCAL keyword is not present, MySQL looks for the datafile on the server host using looking into absolute pathname fully specifies the location of the file, beginning from the root of the filesystem. MySQL reads the file from the given location.
- By default, LOAD DATA assumes that datafiles contain lines that are terminated by linefeeds (newlines) and that data values within a line are separated by tabs.
- To specify a file format explicitly, use a FIELDS clause to describe the characteristics of fields within a line, and a LINES clause to specify the line-ending sequence. The following LOAD DATA statement specifies that the datafile contains values separated by colons and lines terminated by carriage returns and new line character:

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl
  -> FIELDS TERMINATED BY ':'
  -> LINES TERMINATED BY '\r\n';
```

- LOAD DATA assumes the columns in the datafile have the same order as the columns in the table. If that's not true, you can specify a list to indicate which table columns the datafile columns should be loaded into. Suppose your table has columns a, b, and c, but successive columns in the datafile correspond to columns b, c, and a. You can load the file like this:

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt'
    -> INTO TABLE mytbl (b, c, a);
```

Importing Data with mysqlimport

MySQL also includes a utility program named ***mysqlimport*** that acts as a wrapper around LOAD DATA so that you can load input files directly from the command line.

To load a data from dump.txt into mytbl use following command at UNIX prompt.

```
$ mysqlimport -u root -p --local database_name dump.txt
password *****
```

If you use mysqlimport, command-line options provide the format specifiers. mysqlimport commands that correspond to the preceding two LOAD DATA statements look like this:

```
$ mysqlimport -u root -p --local --fields-terminated-by=":" \
   --lines-terminated-by="\r\n"  database_name dump.txt
password *****
```

The order in which you specify the options doesn't matter for mysqlimport, except that they should all precede the database name.

The **mysqlimport** statement uses the --columns option to specify the column order:

```
$ mysqlimport -u root -p --local --columns=b,c,a \
    database_name dump.txt
password *****
```

Handling Quotes and Special Characters:

The FIELDS clause can specify other format options besides TERMINATED BY. By default, LOAD DATA assumes that values are unquoted, and interprets the backslash (\) as an escape character for special characters. To indicate the value quoting character explicitly, use ENCLOSED BY; MySQL will strip that character from the ends of data values during input processing. To change the default escape character, use ESCAPED BY.

When you specify ENCLOSED BY to indicate that quote characters should be stripped from data values, it's possible to include the quote character literally within data values by doubling it or by preceding it with the escape character. For example, if the quote and escape characters are " and \, the input value "a""b\"c" will be interpreted as a"b"c.

For mysqlimport, the corresponding command-line options for specifying quote and escape values are --fields-enclosed-by and --fields-escaped-by

MySQL Useful Functions and Clauses

Here is the list of all important MySQL functions. Each function has been explained along with suitable example.

- MySQL Group By Clause - The MySQL GROUP BY statement is used along with the SQL aggregate functions like SUM to provide means of grouping the result dataset by certain database table column(s).
- MySQL IN Clause - This is a clause which can be used alongwith any MySQL query to specify a condition.
- MySQL BETWEEN Clause - This is a clause which can be used alongwith any MySQL query to specify a condition.
- MySQL UNION Keyword - Use a UNION operation to combine multiple result sets into one.
- MySQL COUNT Function - The MySQL COUNT aggregate function is used to count the number of rows in a database table.
- MySQL MAX Function - The MySQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
- MySQL MIN Function - The MySQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- MySQL AVG Function - The MySQL AVG aggregate function selects the average value for certain table column.
- MySQL SUM Function - The MySQL SUM aggregate function allows selecting the total for a numeric column.
- MySQL SQRT Functions - This is used to generate a square root of a given number.
- MySQL RAND Function - This is used to generate a random number using MySQL command.
- MySQL CONCAT Function - This is used to concatenate any string inside any MySQL command.
- MySQL DATE and Time Functions - Complete list of MySQL Date and Time related functions.
- MySQL Numeric Functions - Complete list of MySQL functions required to manipulate numbers in MySQL.
- MySQL String Functions - Complete list of MySQL functions required to manipulate strings in MySQL.

If you want to list down your website, book or any other resource on this page then please contact at webmaster@tutorialspoint.com

- MySQL and PERL - Its a tutorial from Tutorials Point which explains you how to use MySQL along with PERL and DBI module. Here you will learn all the required MySQL operations alongwith examples.
- MySQL Official Website - Here you can download the latest MySQL release, get the MySQL news update. The mailing list is also a great resources for anyone who want to build dynamic websites using MySQL.
- PHP Official Website: - A complete resource for PHP stuff. Starting from latest PHP updates to latest function manual is available at this site.
- MySQL at Wikipedia - A small article on MySQL, worth to go through it.
- MySQL Reference Manual - A complete official reference for MySQL.
- MySQL-sr-lib - MySQL General Purpose Stored Routines Library. This is a A repository of ready to use routines for everyday needs. This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License.
- Understanding SQL - If you are new to SQL then go through this tutorial to understand SQL basics. SQL, the Structured Query Language, is a mature, powerful, and versatile relational query language.

You can use **GROUP BY** to group values from a column, and, if you wish, perform calculations on that column. You can use COUNT, SUM, AVG etc function on the grouped column.

To understand **GROUP BY** clause consider an **employee_tbl** table which is having following records:

```
mysql> 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 we want to count number of days each employee did work.

If we will write a SQL query as follows then we will get following result:

```
mysql> SELECT COUNT(*) FROM employee_tbl;
+--------------------------+
| COUNT(*)                 |
+--------------------------+
| 7                        |
+--------------------------+
```

But this is not serving our purpose, we want to display total number of pages typed by each person separately. This is done by using aggregate functions in conjunction with a **GROUP BY** clause as follows:

```
mysql> SELECT name, COUNT(*)
    -> FROM   employee_tbl
    -> GROUP BY name;
+------+----------+
| name | COUNT(*) |
+------+----------+
| Jack |        2 |
| Jill |        1 |
| John |        1 |
| Ram  |        1 |
| Zara |        2 |
+------+----------+
5 rows in set (0.04 sec)
```

We will see more functionality related to GROUP BY in other functions like SUM, AVG etc.

You can use **IN** clause to replace many **OR** conditions

To understand **IN** clause consider an **employee_tbl** table which is having following records:

```
mysql> 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 display records with daily_typing_pages equal to 250 and 220 and 170. This can be done using **OR** conditions as follows

```
mysql>SELECT * FROM employee_tbl
    ->WHERE daily_typing_pages= 250 OR
    ->daily_typing_pages= 220 OR  daily_typing_pages= 170;
+------+------+------------+--------------------+
| 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 |
|    4 | Jill | 2007-04-06 |                220 |
+------+------+------------+--------------------+
4 rows in set (0.02 sec)
```

Same can be achieved using **IN** clause as follows:

```
mysql> SELECT * FROM employee_tbl
    -> WHERE daily_typing_pages IN ( 250, 220, 170 );
+------+------+------------+--------------------+
| 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 |
|    4 | Jill | 2007-04-06 |                220 |
+------+------+------------+--------------------+
4 rows in set (0.02 sec)
```

You can use **IN** clause to replace a combination of "greather than equal AND less than equal" conditions.

To understand **BETWEEN** clause consider an **employee_tbl** table which is having following records:

```
mysql> 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 records with conditions daily_typing_pages more than 170 and equal and less than 300 and equal. This can be done using >= and <= conditions as follows

```
mysql>SELECT * FROM employee_tbl
    ->WHERE daily_typing_pages >= 170 AND
    ->daily_typing_pages <= 300;
+------+------+------------+--------------------+
| 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 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
+------+------+------------+--------------------+
5 rows in set (0.03 sec)
```

Same can be achieved using **BETWEEN** clause as follows:

```
mysql> SELECT * FROM employee_tbl
    -> WHERE daily_typing_pages BETWEEN 170 AND 300;
+------+------+------------+--------------------+
| 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 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
+------+------+------------+--------------------+
5 rows in set (0.03 sec)
```

You can use **UNION** if you want to select rows one after the other from several tables, or several sets of rows from a single table all as a single result set.

UNION is available as of MySQL 4.0.This section illustrates how to use it.

Suppose you have two tables that list prospective and actual customers, a third that lists vendors from whom you purchase supplies, and you want to create a single mailing list by merging names and addresses from all three tables. UNION provides a way to do this. Assume the three tables have the following contents:

```
mysql> SELECT * FROM prospect;
+---------+-------+-----------------------+
| fname   | lname | addr                  |
+---------+-------+-----------------------+
| Peter   | Jones | 482 Rush St., Apt. 402 |
| Bernice | Smith | 916 Maple Dr.          |
+---------+-------+-----------------------+
mysql> SELECT * FROM customer;
+-----------+------------+--------------------+
| last_name | first_name | address            |
+-----------+------------+--------------------+
| Peterson  | Grace      | 16055 Seminole Ave. |
| Smith     | Bernice    | 916 Maple Dr.       |
| Brown     | Walter     | 8602 1st St.        |
+-----------+------------+--------------------+
mysql> SELECT * FROM vendor;
+-------------------+---------------------+
| company           | street              |
+-------------------+---------------------+
| ReddyParts, Inc.  | 38 Industrial Blvd. |
| Parts-to-go, Ltd. | 213B Commerce Park. |
+-------------------+---------------------+
```

It does not matter if all the three tables have different column names. The following query illustrates how to select names and addresses from the three tables all at once:

```
mysql> SELECT fname, lname, addr FROM prospect
-> UNION
-> SELECT first_name, last_name, address FROM customer
-> UNION
-> SELECT company, '', street FROM vendor;
+-------------------+----------+-----------------------+
| fname             | lname    | addr                  |
+-------------------+----------+-----------------------+
| Peter             | Jones    | 482 Rush St., Apt. 402 |
| Bernice           | Smith    | 916 Maple Dr.          |
| Grace             | Peterson | 16055 Seminole Ave.    |
| Walter            | Brown    | 8602 1st St.           |
| ReddyParts, Inc.  |          | 38 Industrial Blvd.    |
| Parts-to-go, Ltd. |          | 213B Commerce Park.    |
+-------------------+----------+-----------------------+
```

If you want to select all records, including duplicates, follow the first UNION keyword with ALL:

```
mysql> SELECT fname, lname, addr FROM prospect
```

```
-> UNION ALL
-> SELECT first_name, last_name, address FROM customer
-> UNION
-> SELECT company, '', street FROM vendor;
+--------------------+----------+-----------------------+
| fname              | lname    | addr                  |
+--------------------+----------+-----------------------+
| Peter              | Jones    | 482 Rush St., Apt. 402 |
| Bernice            | Smith    | 916 Maple Dr.          |
| Grace              | Peterson | 16055 Seminole Ave.    |
| Bernice            | Smith    | 916 Maple Dr.          |
| Walter             | Brown    | 8602 1st St.           |
| ReddyParts, Inc.   |          | 38 Industrial Blvd.    |
| Parts-to-go, Ltd.  |          | 213B Commerce Park.    |
+--------------------+----------+-------
```

MySQL **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 following records:

```
mysql> 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:

```
mysql>SELECT COUNT(*) FROM employee_tbl ;
+----------+
| COUNT(*) |
+----------+
|        7 |
+----------+
1 row in set (0.01 sec)
```

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

```
mysql>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.

MySQL **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 following records:

```
mysql> 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:

```
mysql> 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:

```
mysql> id, name, work_date, MAX(daily_typing_pages)
```

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

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

```
mysql> 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)
```

MySQL **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 following records:

```
mysql> 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:

```
mysql> 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:

```
mysql> id, name, work_date, MIN(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+------+------------+-------------------------+
| id   | name | work_date  | MIN(daily_typing_pages) |
+------+------+------------+-------------------------+
|    3 | Jack | 2007-05-06 |                     100 |
|    4 | Jill | 2007-04-06 |                     220 |
|    1 | John | 2007-01-24 |                     250 |
|    2 | Ram  | 2007-05-27 |                     220 |
|    5 | Zara | 2007-06-06 |                     300 |
+------+------+------------+-------------------------+
5 rows in set (0.00 sec)
```

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

```
mysql> 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)
```

MySQL **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 following records:

```
mysql> 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:

```
mysql> 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.

```
mysql> 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)
```

MySQL **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 following records:

```
mysql> 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:

```
mysql> 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.

```
mysql> 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)
```

MySQL **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:

```
mysql>  select SQRT(16);
+----------+
| SQRT(16) |
+----------+
```

```
| 4.000000 |
+----------+
1 row in set (0.00 sec)
```

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

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

```
mysql> 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:

```
mysql> 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)
```

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

```
mysql>  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:

```
mysql>  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 following records:

```
mysql> 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:

```
mysql> 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)

mysql> 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)
```

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

```
mysql> 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 following records:

```
mysql> 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 ca do it using following command:

```
mysql> 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)
```

Name	Description
ADDDATE()	Add dates
ADDTIME()	Add time
CONVERT_TZ()	Convert from one timezone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract two dates
DATE()	Extract the date part of a date or datetime expression

Name	Description
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (1-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format date as a UNIX timestamp
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME	MAKETIME()
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)

Name	Description
STR_TO_DATE()	Convert a string to a date
SUBDATE()	When invoked with three arguments a synonym for DATE_SUB()
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIME()	Extract the time portion of the expression passed
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
UNIX_TIMESTAMP()	Return a UNIX timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)

When invoked with the INTERVAL form of the second argument, ADDDATE() is a synonym for DATE_ADD(). The related function SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
+-------------------------------------------------------+
| DATE_ADD('1998-01-02', INTERVAL 31 DAY)               |
+-------------------------------------------------------+
| 1998-02-02                                            |
+-------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
+-------------------------------------------------------+
| ADDDATE('1998-01-02', INTERVAL 31 DAY)                |
+-------------------------------------------------------+
| 1998-02-02                                            |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

When invoked with the days form of the second argument, MySQL treats it as an integer number of days to be added to expr.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
+-------------------------------------------------------+
| DATE_ADD('1998-01-02', INTERVAL 31 DAY)               |
+-------------------------------------------------------+
| 1998-02-02                                            |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

ADDTIME(expr1,expr2)

ADDTIME() adds expr2 to expr1 and returns the result. expr1 is a time or datetime expression, and expr2 is a time expression.

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
+-------------------------------------------------------+
| DATE_ADD('1997-12-31 23:59:59.999999','1 1:1:1.000002') |
+-------------------------------------------------------+
| 1998-01-02 01:01:01.000001                            |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

CONVERT_TZ(dt,from_tz,to_tz)

This converts a datetime value dt from the time zone given by from_tz to the time zone given by to_tz and returns the resulting value. This function returns NULL if the arguments are invalid.

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
+-----------------------------------------------------------+
| CONVERT_TZ('2004-01-01 12:00:00','GMT','MET')             |
+-----------------------------------------------------------+
| 2004-01-01 13:00:00                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
+-----------------------------------------------------------+
| CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00')       |
+-----------------------------------------------------------+
| 2004-01-01 22:00:00                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
+-----------------------------------------------------------+
| CURDATE()                                                 |
+-----------------------------------------------------------+
| 1997-12-15                                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CURDATE() + 0;
+-----------------------------------------------------------+
| CURDATE() + 0                                             |
+-----------------------------------------------------------+
| 19971215                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CURRENT_DATE and CURRENT_DATE()

CURRENT_DATE and CURRENT_DATE() are synonyms for CURDATE()

CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
+-----------------------------------------------------------+
| CURTIME()                                                 |
+-----------------------------------------------------------+
| 23:50:26                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CURTIME() + 0;
+-----------------------------------------------------------+
| CURTIME() + 0                                             |
+-----------------------------------------------------------+
| 235026                                                    |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CURRENT_TIME and CURRENT_TIME()

CURRENT_TIME and CURRENT_TIME() are synonyms for CURTIME().

CURRENT_TIMESTAMP and CURRENT_TIMESTAMP()

CURRENT_TIMESTAMP and CURRENT_TIMESTAMP() are synonyms for NOW().

DATE(expr)

Extracts the date part of the date or datetime expression expr.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
+-----------------------------------------------------------+
| DATE('2003-12-31 01:02:03')                               |
+-----------------------------------------------------------+
|  2003-12-31                                               |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DATEDIFF(expr1,expr2)

DATEDIFF() returns expr1 . expr2 expressed as a value in days from one date to the other. expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
+----------------------------------------------------------+
| DATEDIFF('1997-12-31 23:59:59','1997-12-30')             |
+----------------------------------------------------------+
| 1                                                        |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

DATE_ADD(date,INTERVAL expr unit), DATE_SUB(date,INTERVAL expr unit)

These functions perform date arithmetic. date is a DATETIME or DATE value specifying the starting date. expr is an expression specifying the interval value to be added or subtracted from the starting date. expr is a string; it may start with a .-. for negative intervals. unit is a keyword indicating the units in which the expression should be interpreted.

The INTERVAL keyword and the unit specifier are not case sensitive.

The following table shows the expected form of the expr argument for each unit value;

unit **Value**	**Expected**expr**Format**
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'

HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

The values QUARTER and WEEK are available beginning with MySQL 5.0.0.

```
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    -> INTERVAL '1:1' MINUTE_SECOND);
+-----------------------------------------------------+
| DATE_ADD('1997-12-31 23:59:59', INTERVAL...         |
+-----------------------------------------------------+
| 1998-01-01 00:01:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
+-----------------------------------------------------+
| DATE_ADD('1999-01-01', INTERVAL 1 HOUR)             |
+-----------------------------------------------------+
| 1999-01-01 01:00:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

DATE_FORMAT(date,format)

Formats the date value according to the format string.

The following specifiers may be used in the format string. The .%. character is required before format specifier characters.

Specifier	**Description**
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, .)

%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v

%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal .%. character
%x	x, for any.x. not listed above

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
+-----------------------------------------------------------+
| DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y')            |
+-----------------------------------------------------------+
| Saturday October 1997                                     |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00'
    -> '%H %k %I %r %T %S %w');
+-----------------------------------------------------------+
| DATE_FORMAT('1997-10-04 22:23:00.......                   |
+-----------------------------------------------------------+
|  22 22 10 10:23:00 PM 22:23:00 00 6                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DATE_SUB(date,INTERVAL expr unit)

This is similar to DATE_ADD() function.

DAY(date)

DAY() is a synonym for DAYOFMONTH().

DAYNAME(date)

Returns the name of the weekday for date.

```
mysql> SELECT DAYNAME('1998-02-05');
+-----------------------------------------------------------+
| DAYNAME('1998-02-05')                                     |
+-----------------------------------------------------------+
| Thursday                                                  |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DAYOFMONTH(date)

Returns the day of the month for date, in the range 0 to 31.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
+-----------------------------------------------------------+
| DAYOFMONTH('1998-02-03')                                  |
+-----------------------------------------------------------+
| 3                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DAYOFWEEK(date)

Returns the weekday index for date (1 = Sunday, 2 = Monday, ., 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
+-----------------------------------------------------------+
|DAYOFWEEK('1998-02-03')                                    |
+-----------------------------------------------------------+
| 3                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

DAYOFYEAR(date)

Returns the day of the year for date, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
+-----------------------------------------------------------+
| DAYOFYEAR('1998-02-03')                                   |
+-----------------------------------------------------------+
| 34                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

EXTRACT(unit FROM date)

The EXTRACT() function uses the same kinds of unit specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
+-----------------------------------------------------------+
| EXTRACT(YEAR FROM '1999-07-02')                           |
+-----------------------------------------------------------+
| 1999                                                      |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
+-----------------------------------------------------------+
| EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03')            |
+-----------------------------------------------------------+
| 199907                                                    |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

FROM_DAYS(N)

Given a day number N, returns a DATE value.

```
mysql> SELECT FROM_DAYS(729669);
+-----------------------------------------------------------+
| FROM_DAYS(729669)                                         |
+-----------------------------------------------------------+
| 1997-10-07                                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

Use FROM_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582).

FROM_UNIXTIME(unix_timestamp)

FROM_UNIXTIME(unix_timestamp,format)

Returns a representation of the unix_timestamp argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. unix_timestamp is an internal timestamp value such as is produced by the UNIX_TIMESTAMP() function.

If format is given, the result is formatted according to the format string, which is used the same way as listed in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(875996580);
+-----------------------------------------------------------+
| FROM_UNIXTIME(875996580)                                  |
+-----------------------------------------------------------+
| 1997-10-04 22:23:00                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

HOUR(time)

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
+-----------------------------------------------------------+
| HOUR('10:05:03')                                          |
+-----------------------------------------------------------+
| 10                                                        |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

LAST_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
+-----------------------------------------------------------+
| LAST_DAY('2003-02-05')                                    |
+-----------------------------------------------------------+
| 2003-02-28                                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

LOCALTIME and LOCALTIME()

LOCALTIME and LOCALTIME() are synonyms for NOW().

LOCALTIMESTAMP and LOCALTIMESTAMP()

LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

MAKEDATE(year,dayofyear)

Returns a date, given year and day-of-year values. dayofyear must be greater than 0 or the result is NULL.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
+-----------------------------------------------------------+
| MAKEDATE(2001,31), MAKEDATE(2001,32)                      |
+-----------------------------------------------------------+
| '2001-01-31', '2001-02-01'                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

MAKETIME(hour,minute,second)

Returns a time value calculated from the hour, minute, and second arguments.

```
mysql> SELECT MAKETIME(12,15,30);
+-----------------------------------------------------------+
| MAKETIME(12,15,30)                                        |
+-----------------------------------------------------------+
| '12:15:30'                                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

MICROSECOND(expr)

Returns the microseconds from the time or datetime expression expr as a number in the range from 0 to 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
+-----------------------------------------------------------+
| MICROSECOND('12:00:00.123456')                            |
+-----------------------------------------------------------+
| 123456                                                    |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

MINUTE(time)

Returns the minute for time, in the range 0 to 59.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
+-----------------------------------------------------------+
| MINUTE('98-02-03 10:05:03')                               |
+-----------------------------------------------------------+
| 5                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

MONTH(date)

Returns the month for date, in the range 0 to 12.

```
mysql> SELECT MONTH('1998-02-03')
+-----------------------------------------------------------+
| MONTH('1998-02-03')                                       |
+-----------------------------------------------------------+
| 2                                                         |
```

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

MONTHNAME(date)

Returns the full name of the month for date.

```
mysql> SELECT MONTHNAME('1998-02-05');
+-----------------------------------------------------+
| MONTHNAME('1998-02-05')                             |
+-----------------------------------------------------+
| February                                            |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();
+-----------------------------------------------------+
| NOW()                                               |
+-----------------------------------------------------+
| 1997-12-15 23:50:26                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

PERIOD_ADD(P,N)

Adds N months to period P (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument P is not a date value.

```
mysql> SELECT PERIOD_ADD(9801,2);
+-----------------------------------------------------+
| PERIOD_ADD(9801,2)                                  |
+-----------------------------------------------------+
| 199803                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

PERIOD_DIFF(P1,P2)

Returns the number of months between periods P1 and P2. P1 and P2 should be in the format YYMM or YYYYMM. Note that the period arguments P1 and P2 are not date values.

```
mysql> SELECT PERIOD_DIFF(9802,199703);
+--------------------------------------------------------+
| PERIOD_DIFF(9802,199703)                               |
+--------------------------------------------------------+
| 11                                                     |
+--------------------------------------------------------+
1 row in set (0.00 sec)
```

QUARTER(date)

Returns the quarter of the year for date, in the range 1 to 4.

```
mysql> SELECT QUARTER('98-04-01');
+--------------------------------------------------------+
| QUARTER('98-04-01')                                    |
+--------------------------------------------------------+
| 2                                                      |
+--------------------------------------------------------+
1 row in set (0.00 sec)
```

SECOND(time)

Returns the second for time, in the range 0 to 59.

```
mysql> SELECT SECOND('10:05:03');
+--------------------------------------------------------+
| SECOND('10:05:03')                                     |
+--------------------------------------------------------+
| 3                                                      |
+--------------------------------------------------------+
1 row in set (0.00 sec)
```

SEC_TO_TIME(seconds)

Returns the seconds argument, converted to hours, minutes, and seconds, as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT SEC_TO_TIME(2378);
+--------------------------------------------------------+
| SEC_TO_TIME(2378)                                      |
+--------------------------------------------------------+
| 00:39:38                                               |
+--------------------------------------------------------+
```

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

STR_TO_DATE(str,format)

This is the inverse of the DATE_FORMAT() function. It takes a string str and a format string format. STR_TO_DATE() returns a DATETIME value if the format string contains both date and time parts, or a DATE or TIME value if the string contains only date or time parts.

```
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
+---------------------------------------------------------+
| STR_TO_DATE('04/31/2004', '%m/%d/%Y')                   |
+---------------------------------------------------------+
| 2004-04-31                                              |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

SUBDATE(date,INTERVAL expr unit) and SUBDATE(expr,days)

When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+---------------------------------------------------------+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY)                 |
+---------------------------------------------------------+
| 1997-12-02                                              |
+---------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
+---------------------------------------------------------+
| SUBDATE('1998-01-02', INTERVAL 31 DAY)                  |
+---------------------------------------------------------+
| 1997-12-02                                              |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

SUBTIME(expr1,expr2)

SUBTIME() returns expr1 . expr2 expressed as a value in the same format as expr1. expr1 is a time or datetime expression, and expr2 is a time.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999',
    -> '1 1:1:1.000002');
```

```
+----------------------------------------------------------+
| SUBTIME('1997-12-31 23:59:59.999999'...                  |
+----------------------------------------------------------+
| 1997-12-30 22:58:58.999997                               |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

SYSDATE()

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

```
mysql> SELECT SYSDATE();
+----------------------------------------------------------+
| SYSDATE()                                                |
+----------------------------------------------------------+
| 2006-04-12 13:47:44                                      |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIME(expr)

Extracts the time part of the time or datetime expression expr and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
+----------------------------------------------------------+
| TIME('2003-12-31 01:02:03')                              |
+----------------------------------------------------------+
| 01:02:03                                                 |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMEDIFF(expr1,expr2)

TIMEDIFF() returns expr1 . expr2 expressed as a time value. expr1 and expr2 are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
    -> '1997-12-30 01:01:01.000002');
+----------------------------------------------------------+
| TIMEDIFF('1997-12-31 23:59:59.000001'.....               |
+----------------------------------------------------------+
|  46:58:57.999999                                         |
+----------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMESTAMP(expr), TIMESTAMP(expr1,expr2)

With a single argument, this function returns the date or datetime expression expr as a datetime value. With two arguments, it adds the time expression expr2 to the date or datetime expression expr1 and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
+---------------------------------------------------------+
| TIMESTAMP('2003-12-31')                                 |
+---------------------------------------------------------+
| 2003-12-31 00:00:00                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMESTAMPADD(unit,interval,datetime_expr)

Adds the integer expression interval to the date or datetime expression datetime_expr. The unit for interval is given by the unit argument, which should be one of the following values: FRAC_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

The unit value may be specified using one of keywords as shown, or with a prefix of SQL_TSI_. For example, DAY and SQL_TSI_DAY both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
+---------------------------------------------------------+
| TIMESTAMPADD(MINUTE,1,'2003-01-02')                     |
+---------------------------------------------------------+
| 2003-01-02 00:01:00                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)

Returns the integer difference between the date or datetime expressions datetime_expr1 and datetime_expr2. The unit for the result is given by the unit argument. The legal values for unit are the same as those listed in the description of the TIMESTAMPADD() function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+---------------------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01')          |
+---------------------------------------------------------+
| 3                                                       |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

TIME_FORMAT(time,format)

This is used like the DATE_FORMAT() function, but the format string may contain format specifiers only for hours, minutes, and seconds.

If the time value contains an hour part that is greater than 23, the %H and %k hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
+-------------------------------------------------------+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l')            |
+-------------------------------------------------------+
| 100 100 04 04 4                                       |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

TIME_TO_SEC(time)

Returns the time argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
+-------------------------------------------------------+
| TIME_TO_SEC('22:23:00')                               |
+-------------------------------------------------------+
| 80580                                                 |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

TO_DAYS(date)

Given a date date, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
+-------------------------------------------------------+
| TO_DAYS(950501)                                       |
+-------------------------------------------------------+
| 728779                                                |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date)

If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If UNIX_TIMESTAMP() is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. date may be a

DATE string, a DATETIME string, a TIMESTAMP, or a number in the format YYMMDD or YYYYMMDD.

```
mysql> SELECT UNIX_TIMESTAMP();
+---------------------------------------------------------+
| UNIX_TIMESTAMP()                                        |
+---------------------------------------------------------+
| 882226357                                               |
+---------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
+---------------------------------------------------------+
| UNIX_TIMESTAMP('1997-10-04 22:23:00')                   |
+---------------------------------------------------------+
| 875996580                                               |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

UTC_DATE, UTC_DATE()

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
+---------------------------------------------------------+
| UTC_DATE(), UTC_DATE() + 0                              |
+---------------------------------------------------------+
| 2003-08-14, 20030814                                    |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

UTC_TIME, UTC_TIME()

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
+---------------------------------------------------------+
| UTC_TIME(), UTC_TIME() + 0                              |
+---------------------------------------------------------+
| 18:07:53, 180753                                        |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

UTC_TIMESTAMP, UTC_TIMESTAMP()

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

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
+---------------------------------------------------------+
| UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0                    |
+---------------------------------------------------------+
| 2003-08-14 18:08:04, 20030814180804                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

WEEK(date[,mode])

This function returns the week number for date. The two-argument form of WEEK() allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the mode argument is omitted, the value of the default_week_format system variable is used

Mode	First Day of week	Range	Week 1 is the first week .
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

```
mysql> SELECT WEEK('1998-02-20');
+---------------------------------------------------------+
| WEEK('1998-02-20')                                      |
+---------------------------------------------------------+
| 7                                                       |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

WEEKDAY(date)

Returns the weekday index for date (0 = Monday, 1 = Tuesday, . 6 = Sunday).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
+-----------------------------------------------------------+
| WEEKDAY('1998-02-03 22:23:00')                            |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

WEEKOFYEAR(date)

Returns the calendar week of the date as a number in the range from 1 to 53. WEEKOFYEAR() is a compatibility function that is equivalent to WEEK(date,3).

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
+-----------------------------------------------------------+
| WEEKOFYEAR('1998-02-20')                                  |
+-----------------------------------------------------------+
| 8                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

YEAR(date)

Returns the year for date, in the range 1000 to 9999, or 0 for the .zero. date.

```
mysql> SELECT YEAR('98-02-03');
+-----------------------------------------------------------+
| YEAR('98-02-03')                                          |
+-----------------------------------------------------------+
| 1998                                                      |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

YEARWEEK(date), YEARWEEK(date,mode)

Returns year and week for a date. The mode argument works exactly like the mode argument to WEEK(). The year in the result may be different from the year in the date argument for the first and the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');
+-----------------------------------------------------------+
| YEAR('98-02-03')YEARWEEK('1987-01-01')                    |
+-----------------------------------------------------------+
| 198653                                                    |
+-----------------------------------------------------------+
```

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

Note that the week number is different from what the WEEK() function would return (0) for optional arguments 0 or 1, as WEEK() then returns the week in the context of the given year.

For more information check MySQL Official Website - Date and Time Functions

---

MySQL numeric functions are used primarily for numeric manipulation and/or mathematical calculations. The following table details the numeric functions that are available in the MySQL implementation.

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:

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

mysql> 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:

```
mysql> 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.

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

ATAN(X)

This function returns the arctangent of X.

```
mysql> 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.

```
mysql> 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:

```
mysql> 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:

```
mysql> 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:

```
mysql> 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 function return the smallest integer value that is not smaller than X. Consider the following example:

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

mysql> 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:

```
mysql> 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.

```
mysql>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:

```
mysql>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.

```
mysql>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.

```
mysql>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.

```
mysql>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:

```
mysql>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:

```
mysql>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:

```
mysql>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:

```
mysql>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:

```
mysql>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:

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

mysql>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.

```
mysql>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:

```
mysql>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).

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

PI()

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

```
mysql>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.

```
mysql> 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.

```
mysql>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:

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

mysql>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.

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

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

mysql>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:

```
mysql>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:

```
mysql>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:

```
mysql>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.

```
mysql>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:

```
mysql>SELECT TRUNCATE(7.536432,2);
+-----------------------------------------------------------+
| TRUNCATE(7.536432,2)                                      |
+-----------------------------------------------------------+
```

```
| 7.53                                                  |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string representation of the argument
BIT_LENGTH()	Return length of argument in bits
CHAR_LENGTH()	Return number of characters in argument
CHAR()	Return the character for each integer passed
CHARACTER_LENGTH()	A synonym for CHAR_LENGTH()
CONCAT_WS()	Return concatenate with separator
CONCAT()	Return concatenated string
CONV()	Convert numbers between different number bases
ELT()	Return string at index number
EXPORT_SET()	Return 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()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FORMAT()	Return a number formatted to specified number of decimal places
HEX()	Return a string representation of a hex value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring

LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MID()	Return a substring starting from the specified position
OCT()	Return 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()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds

SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING(), SUBSTR()	Return the substring as specified
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Convert each pair of hexadecimal digits to a character
UPPER()	Convert 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.

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

mysql> 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.

```
mysql> 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.

```
mysql> 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.

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

```
mysql> 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:

```
mysql> SELECT CONCAT('My', 'S', 'QL');
+-------------------------------------------------------+
| CONCAT('My', 'S', 'QL')                               |
+-------------------------------------------------------+
| MySQL                                                 |
+-------------------------------------------------------+
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.

```
mysql> 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 base 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.

```
mysql> 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().

```
mysql> 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).

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

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

mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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 MySQL 5.0.19, the character_set_filesystem system variable controls interpretation of filenames that are given as literal strings.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

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.

```
mysql> SELECT REPEAT('MySQL', 3);
+-------------------------------------------------------+
| REPEAT('MySQL', 3)                                    |
+-------------------------------------------------------+
| MySQLMySQLMySQL                                       |
+-------------------------------------------------------+
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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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.

```
mysql> SELECT SPACE(6);
+-------------------------------------------------------+
| SELECT SPACE(6)                                       |
+-------------------------------------------------------+
| '      '                                              |
+-------------------------------------------------------+
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.

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

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

mysql> 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.

```
mysql> 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.

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

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

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

mysql> 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.

```
mysql> SELECT UNHEX('4D7953514C');
+-------------------------------------------------------+
| UNHEX('4D7953514C')                                   |
+-------------------------------------------------------+
| MySQL                                                 |
+-------------------------------------------------------+
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.

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

DBMS

- It cannot create a relationship between tables
- It does not supports DISTRIBUTE-SYSTEM
- It does not supports CLIENT-SERVER
- It supports less than 7 rules of Dr.E.F. CoDD

RDBMS

- It can create a relationship between tables
- It supports DISTRIBUTE-SYSTEM
- It supports CLIENT-SERVER

- It supports more than 7 rules of Dr.E.F. CoDD

MySql

Data types

- CHAR
- VARCHAR
- INT BIGINT
- DOUBLE Ex: salary double(10,2)
- DATE
- TIMESTAMP

SQL(Structured Query Language)

Components Of SQL

a)DDL(Data Definition Language)

a. CREATE
b. DROP
c. ALTER
d. TRUNCATE

b) DML(Data Manipulation Language)

a. INSERT
b. UPDATE
c. DELETE

c) DCL(Data Control Language)

a. GRANT

b. REVOKE

c. INVOKE

d) TCL(Transaction Control Language)

a. COMMIT

b. ROLLBACK

c. SAVEPOINTS

e)DQL(Data Query Language)

a. SELECT

Creating Table

Syntax:

CREATE TABLE tablename (columnname datatype(size) , columnname datatype(size) , …….);

Examples

mysql> select ename,deptno from emp where deptno=NULL;
Empty set (0.00 sec)

mysql> select ename, deptno from emp ;

```
+--------+--------+
| ename  | deptno |
+--------+--------+
| Raju   |     20 |
| Ashish |   NULL |
| Dinesh |     20 |
| Suresh |     30 |
+--------+--------+
```

4 rows in set (0.00 sec)

mysql> select ename,deptno from emp where deptno=NULL;
Empty set (0.00 sec)

mysql> select ename,deptno from emp where deptno IS NULL;

```
+--------+--------+
| ename  | deptno |
+--------+--------+
| Ashish |   NULL |
+--------+--------+
```

1 row in set (0.36 sec)

mysql> select ename,deptno from emp where deptno IS NOT NULL;

```
+--------+--------+
| ename  | deptno |
+--------+--------+
| Raju   |     20 |
| Dinesh |     20 |
| Suresh |     30 |
+--------+--------+
```

3 rows in set (0.00 sec)

mysql> select ename, salary*12+500 from emp;

```
+--------+--------------+
| ename  | salary*12+500 |
+--------+--------------+
| Raju   |    360501.08 |
| Ashish |         NULL |
| Dinesh |   6003308.00 |
| Suresh |   1186026.68 |
+--------+--------------+
```

4 rows in set (0.00 sec)

```
mysql> select ename, salary*12, salary*12+500 from emp;
+--------+-----------+--------------+
| ename  | salary*12  | salary*12+500 |
+--------+-----------+--------------+
| Raju   |  360001.08 |     360501.08 |
| Ashish |       NULL |          NULL |
| Dinesh | 6002808.00 |    6003308.00 |
| Suresh | 1185526.68 |    1186026.68 |
+--------+-----------+--------------+
4 rows in set (0.00 sec)

mysql> select ename, salary, salary+500 from emp;
+--------+----------+------------+
| ename  | salary    | salary+500 |
+--------+----------+------------+
| Raju   |  30000.09 |   30500.09 |
| Ashish |      NULL |       NULL |
| Dinesh | 500234.00 |  500734.00 |
| Suresh |  98793.89 |   99293.89 |
+--------+----------+------------+
4 rows in set (0.00 sec)

mysql> insert into emp values('E7','Rajan',6700.09,40);
Query OK, 1 row affected (0.24 sec)

mysql> insert into emp values('E8','Tarun',46700.09,40);
Query OK, 1 row affected (0.01 sec)

mysql> select * from emp;
+-------+--------+----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+----------+--------+
| E1    | Raju   |  30000.09 |     20 |
```

```
| E2    | Ashish |      NULL |   NULL |
| E3    | Dinesh | 500234.00 |     20 |
| E4    | Suresh |  98793.89 |     30 |
| E7    | Rajan  |   6700.09 |     40 |
| E8    | Tarun  |  46700.09 |     40 |
+-------+--------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select * from emp where deptno=20 OR deptno=40;
+-------+--------+-----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E1    | Raju   |  30000.09 |     20 |
| E3    | Dinesh | 500234.00 |     20 |
| E7    | Rajan  |   6700.09 |     40 |
| E8    | Tarun  |  46700.09 |     40 |
+-------+--------+-----------+--------+
4 rows in set (0.00 sec)

mysql> select * from emp where deptno IN (20,40);
+-------+--------+-----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E1    | Raju   |  30000.09 |     20 |
| E3    | Dinesh | 500234.00 |     20 |
| E7    | Rajan  |   6700.09 |     40 |
| E8    | Tarun  |  46700.09 |     40 |
+-------+--------+-----------+--------+
4 rows in set (0.00 sec)

mysql> select * from emp where deptno NOT IN (20,40);
+-------+--------+----------+--------+
| empid | ename  | salary   | deptno |
+-------+--------+----------+--------+
```

```
| E4    | Suresh | 98793.89 |     30 |
+-------+--------+----------+--------+
1 row in set (0.00 sec)

mysql> select * from emp where salary >= 3000 AND
salary<=40000;
+-------+-------+----------+--------+
| empid | ename | salary   | deptno |
+-------+-------+----------+--------+
| E1    | Raju  | 30000.09 |     20 |
| E7    | Rajan |  6700.09 |     40 |
+-------+-------+----------+--------+
2 rows in set (0.00 sec)
mysql> select * from emp where salary >= 3000 AND
salary<=40000 OR DEPTNO=30;
+-------+--------+----------+--------+
| empid | ename  | salary   | deptno |
+-------+--------+----------+--------+
| E1    | Raju   | 30000.09 |     20 |
| E4    | Suresh | 98793.89 |     30 |
| E7    | Rajan  |  6700.09 |     40 |
+-------+--------+----------+--------+
3 rows in set (0.00 sec)

mysql> select * from emp;
+-------+--------+-----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E1    | Raju   |  30000.09 |     20 |
| E2    | Ashish |      NULL |   NULL |
| E3    | Dinesh | 500234.00 |     20 |
| E4    | Suresh |  98793.89 |     30 |
| E7    | Rajan  |   6700.09 |     40 |
| E8    | Tarun  |  46700.09 |     40 |
```

```
+-------+--------+----------+--------+
6 rows in set (0.00 sec)

mysql> select * from emp where salary >= 3000 AND
(salary<=40000 OR DEPTNO=30);
+-------+--------+----------+--------+
| empid | ename  | salary   | deptno |
+-------+--------+----------+--------+
| E1    | Raju   | 30000.09 |     20 |
| E4    | Suresh | 98793.89 |     30 |
| E7    | Rajan  |  6700.09 |     40 |
+-------+--------+----------+--------+
3 rows in set (0.00 sec)

mysql> select * from emp where salary >= 40000 AND
(salary<=3000 OR DEPTNO=30);
+-------+--------+----------+--------+
| empid | ename  | salary   | deptno |
+-------+--------+----------+--------+
| E4    | Suresh | 98793.89 |     30 |
+-------+--------+----------+--------+
1 row in set (0.00 sec)
mysql> select deptno from emp ;
+--------+
| deptno |
+--------+
|     20 |
|   NULL |
|     20 |
|     30 |
|     40 |
|     40 |
+--------+
6 rows in set (0.00 sec)
```

```
mysql> select ename from emp ;
+--------+
| ename  |
+--------+
| Raju   |
| Ashish |
| Dinesh |
| Suresh |
| Rajan  |
| Tarun  |
+--------+
6 rows in set (0.00 sec)

mysql> select ename from emp order by ename;
+--------+
| ename  |
+--------+
| Ashish |
| Dinesh |
| Rajan  |
| Raju   |
| Suresh |
| Tarun  |
+--------+
6 rows in set (0.00 sec)

mysql> select * from emp order by ename;
+-------+--------+-----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E2    | Ashish |      NULL |   NULL |
| E3    | Dinesh | 500234.00 |     20 |
| E7    | Rajan  |   6700.09 |     40 |
```

```
| E1    | Raju   |  30000.09 |     20 |
| E4    | Suresh |  98793.89 |     30 |
| E8    | Tarun  |  46700.09 |     40 |
+-------+--------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select * from emp order by ename desc;
+-------+--------+-----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E8    | Tarun  |  46700.09 |     40 |
| E4    | Suresh |  98793.89 |     30 |
| E1    | Raju   |  30000.09 |     20 |
| E7    | Rajan  |   6700.09 |     40 |
| E3    | Dinesh | 500234.00 |     20 |
| E2    | Ashish |      NULL |   NULL |
+-------+--------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select * from emp order by ename asc;
+-------+--------+-----------+--------+
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E2    | Ashish |      NULL |   NULL |
| E3    | Dinesh | 500234.00 |     20 |
| E7    | Rajan  |   6700.09 |     40 |
| E1    | Raju   |  30000.09 |     20 |
| E4    | Suresh |  98793.89 |     30 |
| E8    | Tarun  |  46700.09 |     40 |
+-------+--------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select * from emp order by deptno asc, salary ;
+-------+--------+-----------+--------+
```

```
| empid | ename  | salary    | deptno |
+-------+--------+-----------+--------+
| E2    | Ashish |      NULL |   NULL |
| E1    | Raju   |  30000.09 |     20 |
| E3    | Dinesh | 500234.00 |     20 |
| E4    | Suresh |  98793.89 |     30 |
| E7    | Rajan  |   6700.09 |     40 |
| E8    | Tarun  |  46700.09 |     40 |
+-------+--------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select deptno, salary from emp order by deptno asc,
salary ;
+--------+-----------+
| deptno | salary    |
+--------+-----------+
|   NULL |      NULL |
|     20 |  30000.09 |
|     20 | 500234.00 |
|     30 |  98793.89 |
|     40 |   6700.09 |
|     40 |  46700.09 |
+--------+-----------+
6 rows in set (0.00 sec)

mysql> select deptno, salary from emp order by deptno asc,
salary desc;
+--------+-----------+
| deptno | salary    |
+--------+-----------+
|   NULL |      NULL |
|     20 | 500234.00 |
|     20 |  30000.09 |
|     30 |  98793.89 |
```

| 40 | 46700.09 |
| 40 | 6700.09 |
+--------+-----------+
6 rows in set (0.00 sec)

For more information check MySQL Official Website - String Functions