MySQL Guide

Meher Krishna Patel

Created on : Octorber, 2017

Last updated : May, 2020

Table of contents

Chapter 1

MySQL overview

Databases is a collection of data, which can be created and updated using Structured Query Language (SQL) commands. The SQL commands are known as queries. There are various databases which support SQL e.g. MS-SQL, MySQL and PostgreSQL. MySQL is the freely available relational database management system . We can write SQL queries in MySQL to manage data. Further, there are non-relational base database e.g. Redis and Hbase etc. These databases are known as NoSQL databases. Also, there are databases which do not support SQL e.g. MongoDB. This page shows various SQL queries in MySQL with examples.

Note:

- MySQL commands and stored-data are case-insensitive in MySQL.
- But Database and table names are case-sensitive.
- # is used for comments.

1.1 Basic commands

This section includes the basic commands for creating a database.

1.1.1 Login and logout

Execute following commands to enter into mysql server. Replace 'root' with correct username. Use 'exit' to quit from the mysql server.

```
$ mysql -u root -p            # replace 'root' with username
Enter password:
Welcome to the MySQL monitor.
[...]

mysql> quit       # exit the database
Bye
$
```

1.1.2 Display databases

'show databases()' is used to see the complete list of databases. Also, each query is ended with a semicolon (;). In the following outputs, four databases are shown which are created by 'MySQL' itself,

```
$ mysql -u root -p

mysql> show databases;  # show the list of databases

+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

1.1.3 Create and delete database

"CREATE DATABASE" and "DROP DATABASE" commands create and delete the database respectively. "testdb" is the name of the database defined by user. Further, each query (i.e. mysql statement) is ended with semicolon (;).

```
mysql> CREATE DATABASE testdb;     # create new database
Query OK, 1 row affected (0.04 sec)


mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| [...]              |
| testdb             |
+--------------------+
5 rows in set (0.08 sec)


mysql> DROP DATABASE testdb;
```

Note: MySQL is case insensitive language. As convention, uppercase letters are used for MySQL keywords e.g. 'CREATE', 'DATABASE' and 'DROP' are written in uppercase as these are the MySQL keywords.

1.1.4 Selecting database

- 'USE' command is used to select a database for operation.

```
mysql> CREATE DATABASE writerdb;   # create new database
Query OK, 1 row affected (0.00 sec)

mysql> USE writerdb     # select database for operation
Database changed
mysql>
```

- To see the selected database, 'SELECT' command can be used,

```
mysql> SELECT DATABASE();   # display the selected dtatabase

+------------+
| DATABASE() |
```

(continues on next page)

(continued from previous page)

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

- For convenience, '->' and 'mysql>' is removed from the queries, which does not generate any output; but used with queries which generates outputs, to make clear distinction between queries and outputs, hence above code will be shown as below throughout the tutorial,

```
# no '->' and 'mysql' is used for queries which do not generate outputs
DROP DATABASE IF EXISTS writerdb;    # delete database 'writerdb' if exists
CREATE DATABASE writerdb;
USE writerdb;

# 'mysql>' and '->' is used for queries which generate outputs
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| writerdb   |
+------------+
1 row in set (0.00 sec)

mysql>
```

1.1.5 Create Table

Various commands are used in below code to create a table "writer" to store information whhic are explained below,

- **USE writerdb:** It selects the writerdb database, to perform further operations. We need not to execute this command if database is already selected.
- **DROP TABLE IF EXISTS writer**: Deletes table if already exists (optional). This command is not required if table does not exist or we do not want to delete the existed table.
- **id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY**: This query contains various parts and some of these are optional parameters as shown below.
  - **id**: the name of the column.
  - **INT UNSIGNED**: defines the type of values.
  - **NOT NULL(optional)**: This column can not be empty
  - **AUTO_INCREMENT (optional)**: User need not to enter the value for this column. Value will be automatically added by MySQL.
  - **PRIMARY KEY (optional)**: One table can have only one primary key. It uniquely defines the record in the table, therefore it can not be same for two columns and can not be a null value.
  - **name VARCHAR(30) NOT NULL UNIQUE**: VARCHAR(30) is variable length string which can contain 30 characters. Since there can be only one primary key in a table, therefore to make other columns unique, we can use UNIQUE command. There is no limits on numbers of UNIQUE command in a table.
- While adding data to the table, all the elements are compulsory except 'age' as we do not use "not null" for this.

```
DROP TABLE IF EXISTS writer;   # optional: delete the existed table
CREATE TABLE writer
(
  id    INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name  VARCHAR(30) NOT NULL UNIQUE,
  age   int
);

mysql> SHOW TABLES;
```

(continues on next page)

(continued from previous page)

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

mysql>
```

- Some more options for creating table are shown in Table *Various fields in 'CREATE TABLE'*,

Table 1.1: Various fields in 'CREATE TABLE'

ITEM	DETAILS
VARCHAR	character variable
NULL	field can be empty
NOT NULL	must provided some value
ENUM SET	user-defined variable with fixed number of values i.e. only either 'M' or 'F' can be used in above example set is same as enum except multiple values can be assigned at one time
DATE TIME DATETIME YEAR TIMESTAMP	YYYY-MM-DD HH:MM:SS YYYY-MM-DD HH:MM:SS YYYY YYYYMMDDHHMMDD
AUTO_INCREMENT	increment the number, when new student is added
PRIMARY KEY	uniquely identify the record, can not be same for two rows. can not be null, and original value can not be changed.
TINYINT SMALLINT MEDIUMINT INT	range -128 to 127 range -32768 to -32767 range -8,388,608 to 8,388,607 range -2^31 to 2^31 - 1
CHAR VARCHAR	character of fixed length character of variable length

1.1.6 Insert data

Insert statement is used to insert the data into a table. Three methods are shown in below code to save the data in the table, which are explained in comments,

```
#method 1: Fill in correct sequnces
# Since 'id' is AUTO_INCREMENT, therefore NULL is set for this.
INSERT INTO writer VALUES
  (NULL, 'Rabindranath Tagore', 80),
  (NULL, 'Leo Tolstoy', 82);

#Method 2:  skip age as it it optional
INSERT INTO writer (name) VALUES ('Pearl Buck');

# Method 3: fill with keywords args i.e (age, name)
# Since 'id' is PRIMARY KEY AUTO_INCREMENT,
# therefore it is not passed in INSERT statement.
INSERT INTO writer (age, name) VALUES
  (30, 'Meher Krishna Patel');
```

Following are the saved data in the table, which can be seen using 'SELECT' command. Here, '*' is used to show all data in the table. We can also specify the specific column as show below,

```
mysql> SELECT * FROM writer;
+----+---------------------+------+
| id | name                | age  |
+----+---------------------+------+
|  1 | Rabindranath Tagore |   80 |
|  2 | Leo Tolstoy         |   82 |
|  3 | Pearl Buck          | NULL |
|  4 | Meher Krishna Patel |   30 |
+----+---------------------+------+
4 rows in set (0.00 sec)


mysql> SELECT name FROM writer;  # show only name
+---------------------+
| name                |
+---------------------+
| Leo Tolstoy         |
| Meher Krishna Patel |
| Pearl Buck          |
| Rabindranath Tagore |
+---------------------+
4 rows in set (0.00 sec)

mysql>
```

In the above query, three methods are shown to insert data into table.

- **Method 1**: To insert the data, all the column and corresponding values are defined in the table. Note that, NULL is used for primary key as it it auto-incremented.
- **Method 2**: Since 'age' column is not defined as 'NOT NULL', therefore it is not used for inserting the data.
- **Method 3**: Here positional parameters are used i.e. (age, name) and then corresponding values are assigned.
- **SELECT * FROM writer**: In SQL, * indicates all. Therefore this statement says that select everythign from table 'writer'. 'Select' statement is discuss in next section.
- **SELECT name FROM writer**: Display only name column.

Note:

- Primary key defines the clustered-index which decides the physical arrangement of the table.
- To increase the speed of the search, we can create non-clustered index for other columns.

1.2 Select statement

In previous section, SELECT query which display the complete table. In general, SELECT statement is used with various other clauses e.g. WHERE, LIKE and ORDER BY etc. to specify the selection criteria. In this section, various such clauses are shown.

1.2.1 Basic SELECT Queries

- SELECT colName1 AS 'NewName1', colName2 AS 'NewName2', ... FROM tableName ;

```
mysql> SELECT name, age FROM writer;  # Select name and age only
+---------------------+------+
| name                | age  |
+---------------------+------+
| Rabindranath Tagore |   80 |
| Leo Tolstoy         |   82 |
```

(continues on next page)

(continued from previous page)

```
| Pearl Buck          | NULL |
| Meher Krishna Patel |   30 |
+---------------------+------+
4 rows in set (0.00 sec)

mysql> SELECT name as 'Author', age FROM writer; # Display 'name' as 'Author'.
+---------------------+------+
| Author              | age  |
+---------------------+------+
| Rabindranath Tagore |   80 |
| Leo Tolstoy         |   82 |
| Pearl Buck          | NULL |
| Meher Krishna Patel |   30 |
+---------------------+------+
4 rows in set (0.04 sec)

mysql>
```

1.2.2 Where

- SELECT colName1, colName2, ... FROM tableName WHERE conditions ;

```
# Select Leo Tolstoy only
mysql> SELECT * FROM writer WHERE name = 'Leo Tolstoy';
+----+-------------+------+
| id | name        | age  |
+----+-------------+------+
|  2 | Leo Tolstoy |   82 |
+----+-------------+------+
1 row in set (0.00 sec)

# Selects id between 1 and 4 i.e 2 and 3
mysql> SELECT * FROM writer WHERE (id >1 and id < 4);
+----+-------------+------+
| id | name        | age  |
+----+-------------+------+
|  2 | Leo Tolstoy |   82 |
|  3 | Pearl Buck  | NULL |
+----+-------------+------+
2 rows in set (0.06 sec)
```

Note: We can not use 1< id < 4 in SQL. We have to use 'and/or' for multiple conditions.

1.2.3 Inserting data with SELECT-WHERE

- Before moving further, lets create another table "book", which stores the names of the books by the authors in table 'writer'.
- In the below table, SELECT-WHERE statements are used to insert the writer_id into 'book' from 'writer' table.

```
DROP TABLE IF EXISTS book;
CREATE TABLE book # creating table 'book'
(
 writer_id  INT UNSIGNED NOT NULL,
 book_id  INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

(continues on next page)

(continued from previous page)

```
 title VARCHAR(100) NOT NULL,
 price INT UNSIGNED
);

# Insert data in the table
INSERT INTO book (writer_id,title,price)
  SELECT id, 'The Good Earth', 200         # select id
  FROM writer WHERE name = 'Pearl Buck';   # where name = 'Pearl Buck'

INSERT INTO book (writer_id,title,price)
  SELECT id, 'The Home and The World',  250
  FROM writer WHERE name = 'Rabindranath Tagore';

INSERT INTO book (writer_id,title,price)
  SELECT id, 'Gitanjali', 100
  FROM writer WHERE name = 'Rabindranath Tagore';

INSERT INTO book (writer_id,title,price)
    SELECT id, 'War and Peace', 200
    FROM writer WHERE name = 'Leo Tolstoy';

INSERT INTO book (writer_id,title,price)
    SELECT id, 'Anna Karenina', 100
    FROM writer WHERE name = 'Leo Tolstoy';
```

- Following is the data stored in 'book' table,

```
mysql> SELECT * FROM book;
+-----------+---------+------------------------+-------+
| writer_id | book_id | title                  | price |
+-----------+---------+------------------------+-------+
|         3 |       1 | The Good Earth         |   200 |
|         1 |       2 | The Home and The World |   250 |
|         1 |       3 | Gitanjali              |   100 |
|         2 |       4 | War and Peace          |   200 |
|         2 |       5 | Anna Karenina          |   100 |
+-----------+---------+------------------------+-------+
5 rows in set (0.00 sec)

mysql>
```

1.2.4 ORDER BY

- Syntax:

```
SELECT colName1, colName2, ... FROM tableName
    ORDER BY 'colName1' ASC, colName2 DESC, ... ;
```

- Example:

```
# ORDER BY: ASC by default
mysql> SELECT title, price FROM book ORDER BY title;
 +------------------------+-------+
 | title                  | price |
 +------------------------+-------+
 | Anna Karenina          |   100 |
 | Gitanjali              |   100 |
 | The Good Earth         |   200 |
 | The Home and The World |   250 |
```

(continues on next page)

(continued from previous page)

```
| War and Peace           |   200 |
+-------------------------+-------+
5 rows in set (0.06 sec)

# descending order
mysql> SELECT title, price FROM book ORDER BY title DESC;
+-------------------------+-------+
| title                   | price |
+-------------------------+-------+
| War and Peace           |   200 |
| The Home and The World  |   250 |
| The Good Earth          |   200 |
| Gitanjali               |   100 |
| Anna Karenina           |   100 |
+-------------------------+-------+
5 rows in set (0.00 sec)


# First arrange by price and then by title.
# 'The Good Earth' and 'War and Peace' have same prices.
# Since 'title ASC' is used, therefore 'The Good Earth' is place above the 'War and Peace'.

mysql> SELECT title, price FROM book ORDER BY title DESC, price ASC;
+-------------------------+-------+
| title                   | price |
+-------------------------+-------+
| War and Peace           |   200 |
| The Home and The World  |   250 |
| The Good Earth          |   200 |
| Gitanjali               |   100 |
| Anna Karenina           |   100 |
+-------------------------+-------+
5 rows in set (0.00 sec)

mysql>
```

1.2.5 LIKE

LIKE is used for pattern matching with percentage (%) and underscore (_) signs.

```
# %th: find titles which start with 'th'.
mysql> SELECT title, price FROM book WHERE title LIKE 'th%';
+-------------------------+-------+
| title                   | price |
+-------------------------+-------+
| The Good Earth          |   200 |
| The Home and The World  |   250 |
+-------------------------+-------+
2 rows in set (0.00 sec)

# %an%: find titles which contain 'an'.
mysql> SELECT title, price FROM book WHERE  title LIKE '%an%';
+-------------------------+-------+
| title                   | price |
+-------------------------+-------+
| The Home and The World  |   250 |
| Gitanjali               |   100 |
| War and Peace           |   200 |
| Anna Karenina           |   100 |
```

(continues on next page)

(continued from previous page)

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

# th%: find titles which end with 'th'.
mysql> SELECT title, price FROM book WHERE title LIKE '%th';
+----------------+-------+
| title          | price |
+----------------+-------+
| The Good Earth |   200 |
+----------------+-------+
1 row in set (0.00 sec)

# %_an%: find titles which contain
# atleast one word before 'an'.
mysql> SELECT title, price FROM book WHERE title LIKE '%_an%';
+------------------------+-------+
| title                  | price |
+------------------------+-------+
| The Home and The World |   250 |
| Gitanjali              |   100 |
| War and Peace          |   200 |
+------------------------+-------+
3 rows in set (0.00 sec)
```

Note: % : it looks for zero or more characters to fill it's place _ : It looks for exactly one character to fill it's place. For two characters, use two underscore and so on.

- Try these commands as well,

```
#Try for these commands also.
SELECT * FROM book WHERE title LIKE '_rt_';

SELECT * FROM book WHERE title LIKE '%rt_';

#two underscore arter 'rt'
SELECT * FROM book WHERE title LIKE '%rt__';

#three underscore after 'rt'
SELECT * FROM book WHERE title LIKE '%r___';

SELECT * FROM book WHERE title LIKE '%r_%';
```

1.3 JOIN

- Syntax:

```
SELECT colName1, colName2, ... FROM tableName1
JOIN_TYPE tableName2
ON tableName1.colName = tableName2.colName ;
```

Data are often stores in different tables or databases. In our example, Author details are stored in 'writer' table, whereas books' details are stored in 'book' table. Further, 'book' table does not contain the name of the authors; but it contains the 'writer_id' which can be used to find the name of the author from 'writer' table. JOIN queries are used for such operations.

Next, we will see the examples of following joins,

- INNER JOIN

- LEFT OUTER JOIN
- RIGHT OUTER JOIN

Note: Before we start, remember that we don't have any record of writer 'Meher Krishna Patel' in 'book' table.

1.3.1 INNER JOIN

INNER JOIN combines the data which are common in both the tables; e.g. 'Meher Krishna Patel' is not available in 'book' table, therefore it will not be displayed in INNER JOIN's results. Further, order of the tables does not matter in INNER JOIN as shown in example.

In following example, writer.id = book.writer_id is used because in writer's 'id' is stored as 'writer_id' in book.

```
# 'writer' is used before 'book'
mysql> SELECT id, name, title, price FROM writer
-> INNER JOIN book
-> ON writer.id  = book.writer_id ORDER BY id;
+----+--------------------+------------------------+-------+
| id | name               | title                  | price |
+----+--------------------+------------------------+-------+
|  1 | Rabindranath Tagore | Gitanjali              |   100 |
|  1 | Rabindranath Tagore | The Home and The World |   250 |
|  2 | Leo Tolstoy        | Anna Karenina          |   100 |
|  2 | Leo Tolstoy        | War and Peace          |   200 |
|  3 | Pearl Buck         | The Good Earth         |   200 |
+----+--------------------+------------------------+-------+
5 rows in set (0.03 sec)


# 'writer' is used after 'book'
mysql> SELECT id, name, title, price FROM book
    -> INNER JOIN writer
    -> ON writer.id  = book.writer_id ORDER BY id;
+----+--------------------+------------------------+-------+
| id | name               | title                  | price |
+----+--------------------+------------------------+-------+
|  1 | Rabindranath Tagore | Gitanjali              |   100 |
|  1 | Rabindranath Tagore | The Home and The World |   250 |
|  2 | Leo Tolstoy        | Anna Karenina          |   100 |
|  2 | Leo Tolstoy        | War and Peace          |   200 |
|  3 | Pearl Buck         | The Good Earth         |   200 |
+----+--------------------+------------------------+-------+
5 rows in set (0.00 sec)
```

1.3.2 LEFT OUTER JOIN

It is same as inner join except it takes all the rows of left-table, and then joins the right-table row to it.

```
#'writer' is left table.
mysql> SELECT id, name, title, price FROM writer
    -> LEFT OUTER JOIN book
    -> ON writer.id  = book.writer_id ORDER BY id;
+----+--------------------+------------------------+-------+
| id | name               | title                  | price |
+----+--------------------+------------------------+-------+
|  1 | Rabindranath Tagore | Gitanjali              |   100 |
|  1 | Rabindranath Tagore | The Home and The World |   250 |
|  2 | Leo Tolstoy        | Anna Karenina          |   100 |
```

(continues on next page)

(continued from previous page)

```
|  2 | Leo Tolstoy         | War and Peace          |   200 |
|  3 | Pearl Buck          | The Good Earth         |   200 |
|  4 | Meher Krishna Patel | NULL                   |  NULL |
+----+---------------------+------------------------+-------+
6 rows in set (0.00 sec)
```

1.3.3 RIGHT OUTER JOIN

It takes all the rows of right-table, and then joins the left-table row to it.

```
# 'book' is right table.
# Meher Krishna Patel is not added here, because
# it takes all data from 'book' table and then add 'writer' to it
mysql> SELECT id, name, title, price FROM writer
    -> RIGHT OUTER JOIN book
    -> ON writer.id  = book.writer_id ORDER BY id;
+------+---------------------+------------------------+-------+
| id   | name                | title                  | price |
+------+---------------------+------------------------+-------+
|    1 | Rabindranath Tagore | Gitanjali              |   100 |
|    1 | Rabindranath Tagore | The Home and The World |   250 |
|    2 | Leo Tolstoy         | Anna Karenina          |   100 |
|    2 | Leo Tolstoy         | War and Peace          |   200 |
|    3 | Pearl Buck          | The Good Earth         |   200 |
+------+---------------------+------------------------+-------+
5 rows in set (0.02 sec)
```

1.4 Group by

Let's create another table with more data, to understand GROUP BY clause.

```
# use database writerdb
USE writerdb;

# create table
DROP TABLE IF EXISTS student;
CREATE TABLE student
(
  id     INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name   VARCHAR(30) NOT NULL UNIQUE,
  gender NVARCHAR(5),
  age    int,
  city   NVARCHAR(30),
  marks int
);

# insert data
INSERT INTO student (name, gender, age, city, marks) VALUES
  ('Tom', 'M', 20, 'NY', 30),
  ('Kim', 'F', 18, 'NY', 40),
  ('Sam', 'M', 18, 'NY', 60),
  ('Kris', 'M', 20, 'DEL', 80),
  ('Harry', 'M', 19, 'DEL', 70),
  ('Eliza', 'F', 19, 'DEL', 50),
  ('Kate', 'F', 15, 'ACK', 20),
  ('Peter', 'M', 21, 'ACK', 80),
  ('Ryan', 'M', 20, 'ACK', 60);
```

Note: GROUP BY command is always used with aggregate functions e.g. sum, min, max and avg etc.

```
# Find the average marks of students from same city.
# Also, display the total number of students in same city.
mysql> SELECT city, AVG(marks) as 'AVG Marks', count(id) as 'Total Students'
    -> FROM student GROUP BY city;
+------+-----------+----------------+
| city | AVG Marks | Total Students |
+------+-----------+----------------+
| ACK  |   53.3333 |              3 |
| DEL  |   66.6667 |              3 |
| NY   |   43.3333 |              3 |
+------+-----------+----------------+
3 rows in set (0.11 sec)



# Find the sum of marks of students from same city and same gender.
# Also, display the total number of students in this case.
mysql> SELECT city, gender, SUM(marks) as 'Total Marks', COUNT(city) 'Total Students' FROM student
    -> GROUP BY city, gender
    -> ORDER BY city;
+------+--------+-------------+----------------+
| city | gender | Total Marks | Total Students |
+------+--------+-------------+----------------+
| ACK  | F      |          20 |              1 |
| ACK  | M      |         140 |              2 |
| DEL  | F      |          50 |              1 |
| DEL  | M      |         150 |              2 |
| NY   | F      |          40 |              1 |
| NY   | M      |          90 |              2 |
+------+--------+-------------+----------------+
6 rows in set (0.01 sec)
```

1.5 Having vs Where

Note:

- WHERE clause filters rows before aggregation operation.
- HAVING clause filters rows after aggregation operation

```
# writing age once is compulsory, as HAVING operation is performed on age.
# replace both ages from below query, and it will give blank result.
mysql> SELECT age, SUM(marks), COUNT(id) FROM student
    -> GROUP BY age HAVING age < 20;
+------+------------+-----------+
| age  | SUM(marks) | COUNT(id) |
+------+------------+-----------+
|   15 |         20 |         1 |
|   18 |        100 |         2 |
|   19 |        120 |         2 |
+------+------------+-----------+
3 rows in set (0.00 sec)

#writing age is is not compulsory in WHERE.
mysql> SELECT SUM(marks), COUNT(id) FROM student
```

(continues on next page)

(continued from previous page)

```
    -> WHERE age < 20 GROUP BY age;
+------------+-----------+
| SUM(marks) | COUNT(id) |
+------------+-----------+
|         20 |         1 |
|        100 |         2 |
|        120 |         2 |
+------------+-----------+
3 rows in set (0.00 sec)
```

Note: Aggregate function can not be used with WHERE, i.e. [SELECT * from student WHERE sum(marks)>100] is a invalid query; whereas [SELECT city, COUNT(id) FROM student GROUP BY city HAVING SUM(marks)< 180] is valid query.

1.6 Store Procedure

If certain queries are used frequently, then we can write store procedures for it. We can call store procedures using 'CALL' command as shown in next examples. Three examples are shown here. In first example no parameter is passed. In second example only one input parameter is passed. Then in third example, one input and one output parameter is passed. We can have any number of input and output parameters in store procedures.

1.6.1 Simple store procedure

```
# simple store procedure
# use database
USE writerdb;

# creating store procedure
DELIMITER //
CREATE PROCEDURE cpGetWriter()
 BEGIN
 SELECT * FROM writer;
 END //
DELIMITER ;
```

- 'CALL' is used to invoke the store procedure,

```
# results will be same as SELECT * FROM writer;
mysql> CALL cpGetWriter();
+----+---------------------+------+
| id | name                | age  |
+----+---------------------+------+
|  1 | Rabindranath Tagore |   80 |
|  2 | Leo Tolstoy         |   82 |
|  3 | Pearl Buck          | NULL |
|  4 | Meher Krishna Patel |   30 |
+----+---------------------+------+
4 rows in set (0.00 sec)
```

1.6.2 Store procedure with input

```
# Store procedure with input
USE writerdb; # use database

#create store procedure
DELIMITER //
CREATE PROCEDURE cpBookDetails(IN writerID INT)
BEGIN
SELECT id, name, title, price FROM writer
INNER JOIN book
ON writer.id  = book.writer_id WHERE id=writerID;
END //
DELIMITER ;
```

- Following are the outputs for the above store procedure,

```
mysql> CALL cpBookDetails(2);
+----+-------------+---------------+-------+
| id | name        | title         | price |
+----+-------------+---------------+-------+
|  2 | Leo Tolstoy | War and Peace |   200 |
|  2 | Leo Tolstoy | Anna Karenina |   100 |
+----+-------------+---------------+-------+
2 rows in set (0.00 sec)
```

1.6.3 Store procedure with input and output

```
# use database
USE writerdb;

DELIMITER $$
CREATE PROCEDURE cpTotalBooks(
 IN writerID INT,
 OUT total int)
BEGIN
 SELECT COUNT(book_id) INTO total
 FROM book WHERE writer_id = writerID;
END$$
DELIMITER ;
```

1.7 Save and Execute query from '.sql' file

```
#call stored procedure
# input parameter = 1,
# output parameter = @totalBooks. @ is compulsory
mysql> CALL cpTotalBooks(1, @totalBooks);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @totalBooks; #display value
+-------------+
| @totalBooks |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

Note:

1. Delimiter: Create procedure needs two delimiter i.e. to end the create command and store procedure itself. Therefore "DELIMITER $$" or "DELIMITER //" are used. At the end of the store procedure, DELIMITER should be set to ' ; ' again.
2. For multiple inputs and outputs use following format:

```
DELIMITER $$
   CREATE PROCEDURE cpTotalBooks(
     IN writerID INT, IN bookID INT,
     OUT totalWriter int, OUT totalBook int)
   BEGIN
    ---query here---
   END$$
DELIMITER ;
```

1.8 View

VIEWs are nothing but virtual tables. Suppose we do not want to display writer_id, to the clients from 'book' table. Then instead of creating a new table, we can create a view of 'book' table, which does not display the writer_id. All the quires, which are applicable to tables can be used with VIEW.

```
# VIEW Example 1:
USE writerdb;

#create VIEW: display title and price of books.
CREATE VIEW BookPrice AS
    SELECT title, price FROM book;

mysql> select * from BookPrice;
+-------------------------+-------+
| title                   | price |
+-------------------------+-------+
| The Good Earth          |   200 |
| The Home and The World  |   250 |
| Gitanjali               |   100 |
| War and Peace           |   200 |
| Anna Karenina           |   100 |
+-------------------------+-------+
5 rows in set (0.00 sec)



# VIEW Example 2:
# create VIEW: Display writers, their books and prices
CREATE VIEW writerBookPrice AS
    SELECT name, title, price FROM writer
    INNER JOIN book
    ON writer.id  = book.writer_id ORDER BY id;

# using VIEW as table
mysql> select * from writerBookPrice;
+---------------------+-------------------------+-------+
| name                | title                   | price |
+---------------------+-------------------------+-------+
| Rabindranath Tagore | The Home and The World  |   250 |
| Rabindranath Tagore | Gitanjali               |   100 |
| Leo Tolstoy         | War and Peace           |   200 |
| Leo Tolstoy         | Anna Karenina           |   100 |
| Pearl Buck          | The Good Earth          |   200 |
```

(continues on next page)

(continued from previous page)

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

1.9 Constraint

- Syntax

```
ALTER TABLE tableName1
ADD CONSTRAINT constraint_name
FOREIGN KEY foreignKeyName (colName1)
REFERENCES tableName2 (colName2)
ON UPDATE CASCADE
ON DELETE RESTRICT;
```

Currently, table 'book' is independent of table 'writer' i.e. we can any writer_id to 'book' e.g. INSERT INTO book (writer_id,title,price) VALUES (8, 'Unknown ID', 200); will insert data into book. But, there is no such writer with id = 8. To avoid such entries, we can use CONSTRAINT as follows.

```
# ADD CONSTRAINT to existing table:
ALTER TABLE book
  ADD CONSTRAINT fk_Writer
  FOREIGN KEY key_writer(writer_id)
  REFERENCES writer(id)
  ON UPDATE CASCADE
  ON DELETE CASCADE;
# CONSTRAINT will not add, if there is already some writer_id in
# 'book', which is not present in 'writer' table.
```

Note: ON DELETE / UPDATE CASCADE: If any author is deleted from 'writer' table, then CASCADE option will delete the corresponding columns in 'book' table.

We can replace 'CASCADE' with 'RESTRICT' or 'NO ACTION' (these two do not allow delete operation) or "SET NULL" (it will set writer_id to NULL). Same rules apply for ON UPDATE operations.

- We can also define constraint, while creating a new table as follows,

```
# Add constraint while creating table
DROP TABLE IF EXISTS book;
CREATE TABLE book
(
  writer_id  INT UNSIGNED NOT NULL,
  book_id  INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(100) NOT NULL,
  price INT UNSIGNED,
  CONSTRAINT fk_Writer
  FOREIGN KEY key_writer(writer_id)
   REFERENCES writer(id)
   ON UPDATE CASCADE
   ON DELETE RESTRICT
);
# ADD keyword is not used here
```

1.10 Command Summary

Table 1.2: Command Summary

Command	Description
mysql -u root -p	Login (replace 'root' with correct username)
quit	exit from mysql
show databases	display all the databases
CREATE DATABASE dbname	create database 'dbname'
DROP DATABASE dbname	delete database 'dbname'
DROP DATABASE IF EXISTS dbname	delete database 'dbname if exists
USE dbname	select database for working
SELECT DATABASE()	shows the selected database
CREATE TABLE tblname(...);	create table 'tblname'
SHOW TABLES	display the list of tables
DESCRIBE tblname	shows the details of table 'tblname'
INSERT INTO tblname VALUES(...)	insert values to table
SELECT * FROM tblname	show all contents of tblname
ALTER TABLE tblname ADD ...	add row to existed table i.e. alter table

Chapter 2

MySQL with Python

In this chapter, we will see the methods by which we can connect Python to MySQL database,

2.1 Initial setup

For connecting Python to the MySQL, we need to install a 'connector' and create a 'database'. After this, we can read or write data to the database,

- First install a connector which allows Python to connect with the database. Install any one of the following connector,

```
$ pip install mysql-connector

or

$ pip install mysqlclient
```

- These the connectors can be imported in python code as below. Rest of the codes will be same for both the connectors.

```
(mysql-connector)
import mysql.connector as mc

or

(mysqlclient)
import MySQLdb as mc
```

Note: The 'mysql-connector' is not supported by Django-framework. The good option is 'mysqlclient' which is supported by Django as well.

- Next, we need to create a database in MySQL. Let's create a new database with name 'pythonSQL',

```
$ mysql -u root -p
Enter password:

mysql> CREATE DATABASE pythonSQL;
```

2.2 Connect and load data

Following code can be used to connect and load the data to database. Note that, the commands in the c.execute(...) statements are exactly same as the commands in the previous chapters.

```
# create_fill_database.py

import mysql.connector as mc

# connect to database
conn= mc.connect(host='localhost',user='root',password='d',db='pythonSQL')
c = conn.cursor()  # cursor to perform operations

def create_table():
    """ Create table in the database """

    # optional: drop table if exists
    c.execute('DROP TABLE IF EXISTS writer')
    c.execute('CREATE TABLE writer \
            (                   \
              id    INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, \
              name  VARCHAR(30) NOT NULL UNIQUE,  \
              age   int \
            )'
    )


def insert_data():
    """ Insert data to the table """

    c.execute("INSERT INTO writer (name) VALUES ('Pearl Buck')")

    c.execute(" INSERT INTO writer VALUES \
        (NULL, 'Rabindranath Tagore', 80), \
        (NULL, 'Leo Tolstoy', 82)" \
    )

    c.execute(" INSERT INTO writer (age, name) VALUES \
        (30, 'Meher Krishna Patel')" \
    )

def commit_close():
    """ commit changes to database and close connection """

    conn.commit()
    c.close()
    conn.close()

def main():
    """ execute create and insert commands """

    create_table()
    insert_data()
    commit_close()  # required for save the changes


# standard boilerplate to call main function
if __name__ == '__main__':
    main()
```

- Next, run the above file to save the data in the database,

```
$ python create_fill_database.py
```

2.3 Read data from table

Following code can be used to read data from the table,

```
# read_database.py

import mysql.connector as mc

conn= mc.connect(host='localhost',user='root',password='d',db='pythonSQL')
c = conn.cursor()

def read_data():
    c.execute('SELECT * FROM writer')
    writers = c.fetchall() # data is read in the form of list
    for writer in writers: # print individual item in the list
        print(writer)      # data at each row is saved as tuple


def main():
    read_data()

if __name__ == '__main__':
    main()
```

- To see the output, execute the code,

```
$ python read_database.py
(1, 'Pearl Buck', None)
(2, 'Rabindranath Tagore', 80)
(3, 'Leo Tolstoy', 82)
(4, 'Meher Krishna Patel', 30)
```

- In this way, we can get the data from the table and perform various operations on the data.
- Also, we can use all those queries with python, as queries in the execute statements are same as queries in previous chapter.

2.4 Connection in try-except block

We can use following code to put the connection string in the try except block, so that we can get proper message for not connecting with the database,

```
# connect_try.py

import mysql.connector as mq
from mysql.connector import errorcode

try:
    conn = mq.connect(host='localhost', user='root', password='d', db='pythonSQL')
    print("Connected")
except mq.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Something is wrong with your user name or password")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("Database does not exist")
    else:
```

(continues on next page)

(continued from previous page)

```
        print(err)
else:
    print("Connection closed")
    conn.close()
```

```
$ python connect_try.py
Connected
Connection closed
```

- Put the incorrect username or password in connect_try.py file and execute the file again,

```
$ python connect_try.py
Something is wrong with your user name or password
```

- For wrong database name, we will get following error,

```
$ python connect_try.py
Database does not exist
```

Chapter 3

Record selection

Various useful MySQL commands are shown in Chapter *MySQL overview*. Then, Chapter *MySQL with Python* shows the way in which MySQL queries can be used in Python. In this chapter, we will discuss some more record selection features of MySQL, which can be quite useful in processing the data efficiently.

3.1 Using MySQL client

Note: Please see Section *Save connection parameter*, before jumping to sections after that.

3.1.1 Connect and terminate database

To connect with database, we need to provide the hostname, username and password as shown below,

```
$ mysql -h hostname -u username -p
```

- By default, hostname(-h) is set to 'localhost' and username (-u) is set to 'your login name'. To terminate the database connection, 'QUIT' command is used as shown in Chapter *MySQL overview*.

3.1.2 Autocomplete

Use 'tab' button to autocomplete the query. Note that autocomplete work for uppercase keywords i.e. if we press tab after writing 'SEL', it will complete it to 'SELECT' or show the possible options; whereas it will not work for lowercase keywords 'sel'.

3.1.3 Display output of query on terminal

Following code can be used to display the output of a query on the terminal. Note that, query is run through the terminal,

```
$ echo 'SELECT * FROM writer' | mysql -h localhost -u root -p writerdb
Enter password:
id  name    age
1   Rabindranath Tagore     80
2   Leo Tolstoy     82
3   Pearl Buck      NULL
4   Meher Krishna Patel     30
```

3.1.4 Save connection parameter

In the above command, we need to specify the connection parameters along with the queries; which can be quite annoying for repetitive queries. The connection parameters can be saved to a file '.my.cnf' in the home directory as shown below,

```
$ cd
$ cat >> .my.cnf
# .my.cnf
[client]
host=localhost
user=root
password=xyz  # press ctrl-c to exit
```

- Now, command in previous section can be executed without connection parameters, as shown below,

```
$ echo 'SELECT * FROM writer' | mysql writerdb
id  name    age
1   Rabindranath Tagore     80
2   Leo Tolstoy     82
3   Pearl Buck      NULL
4   Meher Krishna Patel     30
```

- Further, protect the '.my.cnf' file from others by changing it's permission as follows,

```
$ chmod 600 .my.cnf
```

3.1.5 Queries from command line

In previous two sections, the unix pipe (|) is used to feed the output of echo command to mysql. But there is a nicer way to execute the query using command prompt, i.e. using -e (execute) as shown below. Note that, name of the database is at the end of the query.

```
$ # note that multiple queries are executed in below code,
$ mysql -e  'SELECT * FROM writer; SELECT NOW()' writerdb
+----+---------------------+------+
| id | name                | age  |
+----+---------------------+------+
|  1 | Rabindranath Tagore |   80 |
|  2 | Leo Tolstoy         |   82 |
|  3 | Pearl Buck          | NULL |
|  4 | Meher Krishna Patel |   30 |
+----+---------------------+------+
+---------------------+
| NOW()               |
+---------------------+
| 2017-03-01 18:09:57 |
+---------------------+
```

3.1.6 Terminating partial query

'\c' is used to terminate the query i.e. when we do not want to execute the query, we can cancel it by '\c',

```
mysql> SELECT * FROM \c
```

3.1.7 SQL variables

We can assign variable names while executing the SELECT statements; and then those variables can be used with other statements as shown below,

```
mysql> USE writerdb
Database changed

# defining variable 'n'
mysql> SELECT @n := name FROM writer WHERE name = 'Leo Tolstoy';
+-------------+
| @n := name  |
+-------------+
| Leo Tolstoy |
+-------------+
1 row in set (0.00 sec)

# using variable 'n'
mysql> SELECT * FROM writer WHERE name = @n;
+----+-------------+------+
| id | name        | age  |
+----+-------------+------+
|  2 | Leo Tolstoy |   82 |
+----+-------------+------+
1 row in set (0.01 sec)
```

- SELECT can be used see the value of the variable,

```
mysql> select @n;
+-------------+
| @n          |
+-------------+
| Leo Tolstoy |
+-------------+
1 row in set (0.00 sec)
```

- If a statement returns multiple values that last value will be stored in the variable; whereas it statement returns no value, then NULL will be stored.
- To store a fix value to a variable, 'SET' command is used,

```
mysql> SET @s = 3 + 9;
Query OK, 0 rows affected (0.00 sec)

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

Note:

In above code, SELECT statement is performing the mathematical operations (not the SET statement). Some more examples are shown below,

```
mysql> SELECT (3+2)/2;
+---------+
| (3+2)/2 |
+---------+
|  2.5000 |
```

(continues on next page)

(continued from previous page)

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

mysql> SELECT 1/0;
+------+
| 1/0  |
+------+
| NULL |
+------+
1 row in set, 1 warning (0.00 sec)

mysql> SELECT 'MEHER' = 'KRISHNA';
+---------------------+
| 'MEHER' = 'KRISHNA' |
+---------------------+
|                   0 |
+---------------------+
1 row in set (0.00 sec)

mysql> SELECT 'MEHER' = 'meher';
+-------------------+
| 'MEHER' = 'meher' |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)
```

3.1.8 Reading query from file

- Let's save a simple query in the 'writers.sql' file, as shown below,

```
$ cat > writers.sql
USE writerdb;
SELECT * FROM writer;
```

Then this query can be run from the terminal as shown below,

```
$ mysql < writers.sql
id  name    age
1   Rabindranath Tagore     80
2   Leo Tolstoy     82
3   Pearl Buck      NULL
4   Meher Krishna Patel     30
```

- Now, remove the first line from the writers.db file i.e USE writerdb,

```
$ cat > writers.sql
SELECT * FROM writer;
```

To execute this code, we need to provide the 'table name' explicitly,

```
$ mysql writerdb < writers.sql
id  name    age
1   Rabindranath Tagore     80
2   Leo Tolstoy     82
3   Pearl Buck      NULL
4   Meher Krishna Patel     30
```

- 'SOURCE' is used to execute query from 'mysql' prompt,

```
mysql> USE writerdb;
Database changed
mysql> SOURCE writers.sql
+----+---------------------+------+
| id | name                | age  |
+----+---------------------+------+
|  1 | Rabindranath Tagore |   80 |
|  2 | Leo Tolstoy         |   82 |
|  3 | Pearl Buck          | NULL |
|  4 | Meher Krishna Patel |   30 |
+----+---------------------+------+
4 rows in set (0.00 sec)
```

3.1.9 Saving output to file

We can send the output of the query to some file using unix commands as follows,

```
$ mysql -e  'SELECT * FROM writer; SELECT NOW()' writerdb  >  writers.txt

$ cat writers.txt
id  name    age
1   Rabindranath Tagore     80
2   Leo Tolstoy     82
3   Pearl Buck      NULL
4   Meher Krishna Patel     30
NOW()
2017-03-01 18:16:35
```

3.1.10 Saving output as CSV file

Unix 'sed' command along with 'regular expression' can be used to convert the output into csv format. In the below code, tab (\t) is converted into comma (,) using 'sed' command,

```
$ mysql -e  "SELECT * FROM writer" writerdb | sed -e "s/\t/,/g" > writers.txt

$ cat writers.txt
id,name,age
1,Rabindranath Tagore,80
2,Leo Tolstoy,82
3,Pearl Buck,NULL
4,Meher Krishna Patel,30
```

- Same can be achieved by reading the file as well as shown below,

```
$ mysql writerdb < writers.sql | sed -e "s/\t/,/g" > writers.txt

$ cat writers.txt
id,name,age
1,Rabindranath Tagore,80
2,Leo Tolstoy,82
3,Pearl Buck,NULL
4,Meher Krishna Patel,30
```

3.1.11 HTML and XML outputs

HTML and XML outputs can be created using -H and -X options respectively,

```
$ # HTML
$ mysql -H -e  'SELECT * FROM writer; SELECT NOW()' writerdb

$ # XML
$ mysql -X -e  'SELECT * FROM writer; SELECT NOW()' writerdb
```

3.1.12 Skipping Header from the output

'-N' can be used to skip the header row, as shown below,

```
$ mysql -N -e  "SELECT * FROM writer" writerdb
+---+--------------------+------+
| 1 | Rabindranath Tagore |   80 |
| 2 |         Leo Tolstoy |   82 |
| 3 |          Pearl Buck | NULL |
| 4 | Meher Krishna Patel |   30 |
+---+--------------------+------+
```

- Following unix commands also generate the same result as above,

```
$ mysql -e  "SELECT * FROM writer" writerdb | tail -n +2

$ mysql -e  "SELECT * FROM writer" writerdb | tail --lines=+2
```

3.1.13 Displaying long results

To display the long results properly, '\G' option can be used,

```
mysql> USE writerdb;
Database changed
mysql> SELECT * FROM writer\G;
*************************** 1. row ***************************
  id: 1
name: Rabindranath Tagore
 age: 80
*************************** 2. row ***************************
  id: 2
name: Leo Tolstoy
 age: 82
*************************** 3. row ***************************
  id: 3
name: Pearl Buck
 age: NULL
*************************** 4. row ***************************
  id: 4
name: Meher Krishna Patel
 age: 30
4 rows in set (0.00 sec)
```

- To see complete details of columns, use following 'SHOW FULL COLUMNS' as below,

```
mysql> SHOW FULL COLUMNS FROM writer\G;
*************************** 1. row ***************************
     Field: id
      Type: int(10) unsigned
 Collation: NULL
      Null: NO
       Key: PRI
```

(continues on next page)

(continued from previous page)

```
   Default: NULL
     Extra: auto_increment
Privileges: select,insert,update,references
   Comment:
*************************** 2. row ***************************
     Field: name
      Type: varchar(30)
 Collation: latin1_swedish_ci
      Null: NO
       Key: UNI
   Default: NULL
     Extra:
Privileges: select,insert,update,references
   Comment:
*************************** 3. row ***************************
     Field: age
      Type: int(11)
 Collation: NULL
      Null: YES
       Key:
   Default: NULL
     Extra:
Privileges: select,insert,update,references
   Comment:
3 rows in set (0.00 sec)
```

3.2 More on SELECT statement

In this section, we will focus on SELECT statement retrieve information from the database. First create a new table using 'mail.sql' files whose contents are shown below,

```
# mail.sql

DROP TABLE IF EXISTS mail;
#@ _CREATE_TABLE_
CREATE TABLE mail
(
  t       DATETIME,    # when message was sent
  srcuser VARCHAR(8),  # sender (source user and host)
  srchost VARCHAR(20),
  dstuser VARCHAR(8),  # recipient (destination user and host)
  dsthost VARCHAR(20),
  size    BIGINT,      # message size in bytes
  INDEX (t)
);
#@ _CREATE_TABLE_

INSERT INTO mail (t,srchost,srcuser,dsthost,dstuser,size)
  VALUES
    ('2014-05-11 10:15:08','saturn','barb','mars','tricia',58274),
    ('2014-05-12 12:48:13','mars','tricia','venus','gene',194925),
    ('2014-05-12 15:02:49','mars','phil','saturn','phil',1048),
    ('2014-05-12 18:59:18','saturn','barb','venus','tricia',271),
    ('2014-05-14 09:31:37','venus','gene','mars','barb',2291),
    ('2014-05-14 11:52:17','mars','phil','saturn','tricia',5781),
    ('2014-05-14 14:42:21','venus','barb','venus','barb',98151),
    ('2014-05-14 17:03:01','saturn','tricia','venus','phil',2394482),
    ('2014-05-15 07:17:48','mars','gene','saturn','gene',3824),
```

(continues on next page)

(continued from previous page)

```
    ('2014-05-15 08:50:57','venus','phil','venus','phil',978),
    ('2014-05-15 10:25:52','mars','gene','saturn','tricia',998532),
    ('2014-05-15 17:35:31','saturn','gene','mars','gene',3856),
    ('2014-05-16 09:00:28','venus','gene','mars','barb',613),
    ('2014-05-16 23:04:19','venus','phil','venus','barb',10294),
    ('2014-05-19 12:49:23','mars','phil','saturn','tricia',873),
    ('2014-05-19 22:21:51','saturn','gene','venus','gene',23992)
;
```

- Use following commands to create and load the content of 'mail.sql' file to database,

```
$ mysql -u root -p
Enter password:


# create and use database 'mysqldb'
mysql> CREATE DATABASE mysqldb;
mysql> USE mysqldb;
Database changed

# load data from 'mail.sql'
mysql> SOURCE mail.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.40 sec)

Query OK, 16 rows affected (0.15 sec)
Records: 16  Duplicates: 0  Warnings: 0

# check data in the new table 'mail'
mysql> SELECT * FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2014-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2014-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2014-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2014-05-12 18:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2014-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2014-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
| 2014-05-14 14:42:21 | barb    | venus   | barb    | venus   |   98151 |
| 2014-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2014-05-15 07:17:48 | gene    | mars    | gene    | saturn  |    3824 |
| 2014-05-15 08:50:57 | phil    | venus   | phil    | venus   |     978 |
| 2014-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2014-05-15 17:35:31 | gene    | saturn  | gene    | mars    |    3856 |
| 2014-05-16 09:00:28 | gene    | venus   | barb    | mars    |     613 |
| 2014-05-16 23:04:19 | phil    | venus   | barb    | venus   |   10294 |
| 2014-05-19 12:49:23 | phil    | mars    | tricia  | saturn  |     873 |
| 2014-05-19 22:21:51 | gene    | saturn  | gene    | venus   |   23992 |
+---------------------+---------+---------+---------+---------+---------+
16 rows in set (0.00 sec)
```

3.2.1 Give name to columns

Currently, the column names (i.e. srcuser, dstuser adn dsthost etc.) do not provide useful information. We can change these names while selecting the data as follows,

```
mysql> SELECT
    -> srcuser AS 'Message Sender',
    -> srchost AS 'Source Host',
    -> dstuser AS 'Message Receiver',
    -> dsthost AS 'Destination Host'
    -> FROM mail;
+----------------+-------------+------------------+------------------+
| Message Sender | Source Host | Message Receiver | Destination Host |
+----------------+-------------+------------------+------------------+
| barb           | saturn      | tricia           | mars             |
| tricia         | mars        | gene             | venus            |
| [...]                                                              |
| gene           | saturn      | gene             | venus            |
+----------------+-------------+------------------+------------------+
16 rows in set (0.00 sec)
```

3.2.2 Combining columns

Suppose, we want to combine the columns e.g. 'Source host' and 'Message Sender', then it can be done using 'CONCAT' keyword, as shown below. Note that, we can use any or no symbol in CONCAT e.g. '-' and '@' is used in below code,

```
mysql> SELECT
    -> CONCAT(srcuser, '-', srchost) AS sender, # e.g. barb-saturn
    -> CONCAT(dstuser, '@', dsthost) AS receiver # e.g. gene@veus
    -> FROM mail;
+---------------+---------------+
| sender        | receiver      |
+---------------+---------------+
| barb-saturn   | tricia@mars   |
| tricia-mars   | gene@venus    |
| phil-mars     | phil@saturn   |
| barb-saturn   | tricia@venus  |
| [...]                         |
| gene-saturn   | gene@venus    |
+---------------+---------------+
16 rows in set (0.00 sec)
```

3.2.3 WHERE (specify selection)

WHERE is used to select specific rows from the table.

- Select rows with dsthost='venus'

```
mysql> SELECT * FROM mail WHERE dsthost='venus';
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2014-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2014-05-12 18:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2014-05-14 14:42:21 | barb    | venus   | barb    | venus   |   98151 |
| [...]                                                              |
| 2014-05-19 22:21:51 | gene    | saturn  | gene    | venus   |   23992 |
+---------------------+---------+---------+---------+---------+---------+
7 rows in set (0.00 sec)
```

- Select rows where srchost starts with 'g',

```
mysql> SELECT * FROM mail WHERE srchost LIKE 's%';
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2014-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2014-05-12 18:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2014-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2014-05-15 17:35:31 | gene    | saturn  | gene    | mars    |    3856 |
| 2014-05-19 22:21:51 | gene    | saturn  | gene    | venus   |   23992 |
+---------------------+---------+---------+---------+---------+---------+
5 rows in set (0.00 sec)
```

- Select rows with srcuser = 'barb' and dstuser = 'tricia',

```
mysql> SELECT * FROM mail WHERE srcuser='barb' AND dstuser = 'tricia';
+---------------------+---------+---------+---------+---------+-------+
| t                   | srcuser | srchost | dstuser | dsthost | size  |
+---------------------+---------+---------+---------+---------+-------+
| 2014-05-11 10:15:08 | barb    | saturn  | tricia  | mars    | 58274 |
| 2014-05-12 18:59:18 | barb    | saturn  | tricia  | venus   |   271 |
+---------------------+---------+---------+---------+---------+-------+
2 rows in set (0.00 sec)
```

- Select rows where srcuser=dstuser and srchost=dsthost,

```
mysql> SELECT * FROM mail WHERE srcuser=dstuser AND srchost=dsthost;
+---------------------+---------+---------+---------+---------+-------+
| t                   | srcuser | srchost | dstuser | dsthost | size  |
+---------------------+---------+---------+---------+---------+-------+
| 2014-05-14 14:42:21 | barb    | venus   | barb    | venus   | 98151 |
| 2014-05-15 08:50:57 | phil    | venus   | phil    | venus   |   978 |
+---------------------+---------+---------+---------+---------+-------+
2 rows in set (0.00 sec)
```

3.2.4 IN clause (for OR operation)

IN clause can be used for performing the OR operation,

```
mysql> SELECT * FROM mail WHERE srcuser IN ('barb', 'gene');
+---------------------+---------+---------+---------+---------+--------+
| t                   | srcuser | srchost | dstuser | dsthost | size   |
+---------------------+---------+---------+---------+---------+--------+
| 2014-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |  58274 |
| 2014-05-12 18:59:18 | barb    | saturn  | tricia  | venus   |    271 |
| 2014-05-14 09:31:37 | gene    | venus   | barb    | mars    |   2291 |
| [... ]                                                            |
| 2014-05-19 22:21:51 | gene    | saturn  | gene    | venus   |  23992 |
+---------------------+---------+---------+---------+---------+--------+
9 rows in set (0.00 sec)
```

3.2.5 NULL values

Let's, add some more entries to table 'mail' as follows,

```
mysql> INSERT INTO mail (t,srcuser,dstuser,size)
->   VALUES
->     ('2014-05-11 10:15:08','barb','tricia',58274),
->     ('2014-05-12 12:48:13','tricia','gene',194925),
```

(continues on next page)

(continued from previous page)

```
->     ('2014-05-12 15:02:49','phil','saturn',1048),
->     ('2014-05-12 18:59:18','barb','tricia',271)
-> ;
Query OK, 4 rows affected (0.11 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

3.2.5.1 Search the NULL values

- In above insertion, 'srchost' and 'dsthost' columns are neglected, therefore these values will be filled as 'NULL'
- Now, we can see some NULL entries as follows. Note that, 'srchost=NULL' does not return the correct answers; hence for NULL values use 'srchost IS NULL' as shown below,
- Similarly, use 'srchost IS NOT NULL (instead of srchost != NULL).

```
mysql> SELECT * FROM mail where srchost=NULL;
Empty set (0.00 sec)

mysql> SELECT * FROM mail where srchost IS NULL;
+---------------------+---------+---------+---------+---------+--------+
| t                   | srcuser | srchost | dstuser | dsthost | size   |
+---------------------+---------+---------+---------+---------+--------+
| 2014-05-11 10:15:08 | barb    | NULL    | tricia  | NULL    |  58274 |
| 2014-05-12 12:48:13 | tricia  | NULL    | gene    | NULL    | 194925 |
| 2014-05-12 15:02:49 | phil    | NULL    | saturn  | NULL    |   1048 |
| 2014-05-12 18:59:18 | barb    | NULL    | tricia  | NULL    |    271 |
+---------------------+---------+---------+---------+---------+--------+
4 rows in set (0.00 sec)
```

3.2.5.2 NULL <=> NULL

NULL <=> NULL compares the NULL values and return '1' if values are NULL; whereas NULL=NULL returns NULL if values are NULL, as shown below,

```
mysql> SELECT NULL=NULL, NULL<=>NULL;
+-----------+-------------+
| NULL=NULL | NULL<=>NULL |
+-----------+-------------+
|      NULL |           1 |
+-----------+-------------+
```

3.2.5.3 Filling NULL with other values

```
mysql> SELECT srcuser, IF(srchost IS NULL, 'Unknown', srchost) FROM mail;
+---------+-----------------------------------------+
| srcuser | IF(srchost IS NULL, 'Unknown', srchost) |
+---------+-----------------------------------------+
| barb    | saturn                                  |
| [...]                                             |
| barb    | Unknown                                 |
| tricia  | Unknown                                 |
| phil    | Unknown                                 |
| barb    | Unknown                                 |
+---------+-----------------------------------------+
20 rows in set (0.00 sec)
```

- Above method will will with any type of column-values, as shown below,

```
mysql> SELECT srcuser, IF(srchost='mars', 'MARS', srchost) FROM mail;
+---------+-------------------------------------+
| srcuser | IF(srchost='mars', 'MARS', srchost) |
+---------+-------------------------------------+
| barb    | saturn                              |
| tricia  | MARS                                |
| phil    | MARS                                |
| [...]                                         |
| barb    | NULL                                |
+---------+-------------------------------------+
20 rows in set (0.00 sec)
```

- 'IFNULL' keyword can be used to fill the NULL value with some other values,

```
mysql> SELECT srcuser, IFNULL(srchost, 'Unknown') FROM mail;
+---------+-----------------------------+
| srcuser | IFNULL(srchost, 'Unknown') |
+---------+-----------------------------+
| barb    | saturn                      |
| tricia  | mars                        |
| [...]                                 |
| barb    | Unknown                     |
| tricia  | Unknown                     |
| phil    | Unknown                     |
| barb    | Unknown                     |
+---------+-----------------------------+
20 rows in set (0.00 sec)o
```

3.2.6 Return part of result with LIMIL

- We can see first few lines of results using LIMIT keyword as follow,

```
mysql> SELECT * FROM mail LIMIT 3;
+---------------------+---------+---------+---------+---------+--------+
| t                   | srcuser | srchost | dstuser | dsthost | size   |
+---------------------+---------+---------+---------+---------+--------+
| 2014-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |  58274 |
| 2014-05-12 12:48:13 | tricia  | mars    | gene    | venus   | 194925 |
| 2014-05-12 15:02:49 | phil    | mars    | phil    | saturn  |   1048 |
+---------------------+---------+---------+---------+---------+--------+
3 rows in set (0.00 sec)
```

- Skip first 2 rows and show next 3 rows (i.e. rows 3-5),

```
mysql> SELECT * FROM mail LIMIT 2,3;
+---------------------+---------+---------+---------+---------+------+
| t                   | srcuser | srchost | dstuser | dsthost | size |
+---------------------+---------+---------+---------+---------+------+
| 2014-05-12 15:02:49 | phil    | mars    | phil    | saturn  | 1048 |
| 2014-05-12 18:59:18 | barb    | saturn  | tricia  | venus   |  271 |
| 2014-05-14 09:31:37 | gene    | venus   | barb    | mars    | 2291 |
+---------------------+---------+---------+---------+---------+------+
3 rows in set (0.00 sec)
```

Chapter 4

Date and time

MySQL supports various formats for dates and time. In this chapter, these different date-time formats are discussed along with the conversion and arithmetic operations. We will use the same table which is created in Section *More on SELECT statement*.

4.1 Current date and time

Current date and time functions can use useful to fill the fields automatically while inserting or updating the data. NOW(), CURDATE() and CURTIME() functions can be used for this purpose as shown below,

```
mysql> SELECT NOW(), CURDATE(), CURTIME();
+---------------------+------------+-----------+
| NOW()               | CURDATE()  | CURTIME() |
+---------------------+------------+-----------+
| 2017-03-02 13:59:27 | 2017-03-02 | 13:59:27  |
+---------------------+------------+-----------+
1 row in set (0.00 sec)
```

4.2 Decomposing dates and times

In this section, we will see various ways to extract the date and times from a date-time field.

4.2.1 Using formatting functions

DATE_FORMAT and TIME_FORMAT functions can be used to extract certain part from the date-time filed. In the following code, DATE_FORMAT is used. Note that, TIME_FORMAT can be used only if the field contains the time as well; whereas DATE_FORMAT can be used in all the cases.

```
mysql> SELECT
    -> t,
    -> DATE_FORMAT(t, '%M %d, %Y') as date, # change format of t
    -> srcuser
    -> FROM mail LIMIT 3;

+---------------------+--------------+---------+
| t                   | date         | srcuser |
+---------------------+--------------+---------+
| 2014-05-11 10:15:08 | May 11, 2014 | barb    |
| 2014-05-12 12:48:13 | May 12, 2014 | tricia  |
```

(continues on next page)

(continued from previous page)

```
| 2014-05-12 15:02:49 | May 12, 2014 | phil    |
+---------------------+--------------+---------+
3 rows in set (0.00 sec)


mysql> SELECT t, TIME_FORMAT(t, '%H') as date,  srcuser FROM mail LIMIT 3;
+---------------------+------+---------+
| t                   | date | srcuser |
+---------------------+------+---------+
| 2014-05-11 10:15:08 | 10   | barb    |
| 2014-05-12 12:48:13 | 12   | tricia  |
| 2014-05-12 15:02:49 | 15   | phil    |
+---------------------+------+---------+
```

- Commonly used formats are shown in below table,

Format	Description
%Y	YYYY
%y	YY
%M	complete name of month e.g. July
%m	month in two digit format i.e. 01
%c	month of year i.e. 1, 2 etc.
%d	two digit date i.e. 01
%e	day of month i.e. 1, 2
%r	12 Hr time with AM/PM
%T	24 Hr time
%H	two digit hour
%i	two digit minute
%s	two digit second

4.2.2 Using component extraction function

There are various functions available in MySQL, which can be used to extract the fields e.g. YEAR() and MONTH() etc.

```
mysql> SELECT t,
    -> YEAR(t) as year,    # year ot t
    -> SECOND(t) as sec    # second of t
    -> from mail LIMIT 3;
+---------------------+------+------+
| t                   | year | sec  |
+---------------------+------+------+
| 2014-05-11 10:15:08 | 2014 |    8 |
| 2014-05-11 10:15:08 | 2014 |    8 |
| 2014-05-12 12:48:13 | 2014 |   13 |
+---------------------+------+------+
```

- The complete list is shown in Table *Component extraction function*.

Table 4.1: Component extraction function

Function	Description
YEAR()	year
MONTH()	month number 1,2
MONTHNAME()	name of month
DAYOFMONTH()	day of the month 1,2
DAYNAME()	sunday, monday
DAYOFWEEK()	1..7 for Sun..Sat
WEEKDAY()	0..6 for Mon..Sun
DAYOFYEAR()	1..366
HOUR()	0..23
MINUTEi()	0..59
SECOND()	0..59

- Saving Sunday as 'Sun' etc.

```
mysql> SELECT t,
    -> LEFT(DAYNAME(t), 3)
    -> FROM mail LIMIT 3;
+---------------------+----------------------+
| t                   | LEFT(DAYNAME(t), 3) |
+---------------------+----------------------+
| 2014-05-11 10:15:08 | Sun                  |
| 2014-05-11 10:15:08 | Sun                  |
| 2014-05-12 12:48:13 | Mon                  |
+---------------------+----------------------+
```

4.3 Calculating intervals

4.3.1 Interval between times

'TIME_TO_SEC' and 'SEC_TO_TIME' functions are used to convert the time into seconds and second into time respectively. With the help of these functions, we can calculate the time interval as shown below.

- Note that the difference is between time only (dates are not considered in the difference).

```
mysql> SELECT
    -> TIME_TO_SEC(NOW())-TIME_TO_SEC(t) as 'interval in sec'
    -> FROM mail LIMIT 3;
+-----------------+
| interval in sec |
+-----------------+
|           16901 |
|           16901 |
|            7716 |
+-----------------+
```

- To convert the above result into HOUR, MIN and SEC, we can use 'extraction functions' as shown below,

```
mysql> SELECT
    -> @sec := TIME_TO_SEC(NOW())-TIME_TO_SEC(t) as 'interval in sec',
    -> HOUR(SEC_TO_TIME(@sec)) AS 'hour',
    -> MINUTE(SEC_TO_TIME(@sec)) AS 'minute',
    -> SECOND(SEC_TO_TIME(@sec)) AS 'second'
    -> FROM mail LIMIT 3;
+-----------------+------+--------+--------+
| interval in sec | hour | minute | second |
```

(continues on next page)

(continued from previous page)

```
+-----------------+------+--------+--------+
|           17298 |    4 |     48 |     18 | # 4*3600 + 48*60 + 18 = 17298
|           17298 |    4 |     48 |     18 |
|            8113 |    2 |     15 |     13 |
+-----------------+------+--------+--------+
```

4.3.2 Interval between days

TO_DAYS can be used to calculate total interval in terms of dates as shown below,

```
mysql> SELECT
    -> @day := TO_DAYS(NOW())-TO_DAYS(t) as 'Total days'
    -> FROM mail LIMIT 3;
+------------+
| Total days |
+------------+
|       1026 |
|       1026 |
|       1025 |
+------------+
3 rows in set (0.00 sec)
```

- Days can be conveted into Year and months etc. as below,

```
mysql> SELECT @day := TO_DAYS(NOW())-TO_DAYS(t) as 'Total days',
    -> MONTH(@day) AS 'Month',
    -> DAYOFYEAR(@day) AS 'Days'
    -> FROM mail LIMIT 3;
+------------+-------+------+
| Total days | Month | Days |
+------------+-------+------+
|       1026 |    10 |  300 |
|       1026 |    10 |  300 |
|       1025 |    10 |  299 |
+------------+-------+------+
```

4.3.3 Adding time to dates

DATE_ADD can be used to shift the dates by a fixed amount,

```
mysql> SELECT t,
    -> DATE_ADD(t, INTERVAL 7 DAY) AS '1 week',
    -> DATE_ADD(t, INTERVAL 2 MONTH) AS '2 month'
    -> FROM mail LIMIT 3;
+---------------------+---------------------+---------------------+
| t                   | 1 week              | 2 month             |
+---------------------+---------------------+---------------------+
| 2014-05-11 10:15:08 | 2014-05-18 10:15:08 | 2014-07-11 10:15:08 |
| 2014-05-11 10:15:08 | 2014-05-18 10:15:08 | 2014-07-11 10:15:08 |
| 2014-05-12 12:48:13 | 2014-05-19 12:48:13 | 2014-07-12 12:48:13 |
+---------------------+---------------------+---------------------+
```