Chapter 3 - Important SQL commands

In this chapter we will show you some common SQL command used to manage a database. We will describe how you can insert new records in a table, modify a table, update the contents of a field... We will also show you how to query a database using the SELECT statement.

Insert new records

To insert new records in an MySQL table, the **INSERT INTO** command is used. **INSERT INTO** allows you to insert one or more rows into a table you specify. Here is the syntax:

```
INSERT INTO table_name (column1, column2...) VALUES (value1, value2,...);
```

The **table_name** parameter specifies the table you would like to insert a row into. After the table name, a list of comma-separated column names is specified. This tells MySQL that these are the fields into which the data will be inserted. Finally, after the **VALUES** keyword, a comma-separated values of the corresponding columns are specified.

Here is an example. In the previous chapters we've created an empty table called **testtable**. This table contains three columns: **name**, **surname**, and **year**. Let's say that we want to insert a new record. Here is how we would do that:

```
mysql> INSERT INTO testtable (name, surname, year) VALUES ('Amy','Goodridge','19
91');
Query OK, 1 row affected (0.01 sec)
```

To display the content of our table, we can use the **SELECT * FROM testtable** command:

```
mysql> SELECT * FROM testtable;
+------+-----------+------+
| name | surname   | year |
+------+-----------+------+
| Amy  | Goodridge | 1991 |
+------+-----------+------+
1 row in set (0.00 sec)
```

As you can see from the output above, the record was successfully added to the table.

If you specify the value of the corresponding column for all columns in the table, you don't need to specify the column names, only their values. Here is an example:

```
mysql> INSERT INTO testtable VALUES ('Mark','Smith','1955');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM testtable;
+------+-----------+------+
| name | surname   | year |
+------+-----------+------+
| Amy  | Goodridge | 1991 |
| Mark | Smith     | 1955 |
+------+-----------+------+
2 rows in set (0.00 sec)
```

Notice how we didn't specify the column names in the command displayed above.

Modify a table

The ALTER TABLE statement is used to change the existing table structure. It can be used to add or remove columns, change the column type, rename a table, etc.

Rename a table

The **ALTER TABLE** statement can be used to rename a table. The syntax:

```
ALTER TABLE old_name RENAME TO new_name
```

Here is how we can rename our table from **testtable** to **testtb**:

```
mysql> ALTER TABLE testtable RENAME TO testtb;
Query OK, 0 rows affected (0.01 sec)
```

Change the column data type

We can change the data type of a column using the **ALTER TABLE** statement. Here is the syntax:

```
ALTER TABLE table_name MODIFY column_name new_type
```

For example, to change the data type of a column called year from **CHAR** to **INT**, we can use the following command:

```
mysql> ALTER TABLE testtb MODIFY year INT;
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Add a new column

We can use the **ALTER TABLE** statement to add a new column to our table. Here is the syntax:

```
ALTER TABLE table_name  ADD COLUMN column_name TYPE
```

Here is an example. To add a new column called **postcode** of the **INT** type to our table **testtb**, we can use the following command:

```
mysql> ALTER TABLE testtb ADD COLUMN postcode INT;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM testtb;
+------+-----------+------+----------+
| name | surname   | year | postcode |
+------+-----------+------+----------+
| Amy  | Goodridge | 1991 |     NULL |
| Mark | Smith     | 1955 |     NULL |
+------+-----------+------+----------+
2 rows in set (0.00 sec)
```

Remove a column

You can use the following **ALTER TABLE** statement to drop a column from a table:

```
ALTER TABLE table_name DROP COLUMN column_name
```

To remove the column postcode we've created in the previous step, we would use the following command:

```
mysql> ALTER TABLE testtb DROP COLUMN postcode;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM testtb;
+------+-----------+------+
| name | surname   | year |
+------+-----------+------+
| Amy  | Goodridge | 1991 |
| Mark | Smith     | 1955 |
+------+-----------+------+
2 rows in set (0.00 sec)
```

Query a database

We've learned that a table consists of rows and columns. Often, you want to see a subset rows, a subset of columns, or a combination of two. To query data from tables in MySQL, the **SELECT** statement is used. Here is the basic syntax:

```
SELECT column_name,column_name FROM table_name
```

The **SELECT** statement is used to control which columns and rows will be displayed. For example, if we only want to display the first name and the surname of all users in our testtb, we would use the following command:

```
mysql> SELECT name, surname FROM testtb;
+------+-----------+
| name | surname   |
+------+-----------+
| Amy  | Goodridge |
| Mark | Smith     |
+------+-----------+
2 rows in set (0.00 sec)
```

To display every column in the table, we can use the asterix (*) character:

```
mysql> SELECT * FROM testtb;
+------+-----------+------+
| name | surname   | year |
+------+-----------+------+
| Amy  | Goodridge | 1991 |
| Mark | Smith     | 1955 |
+------+-----------+------+
2 rows in set (0.00 sec)
```

Advanced SELECT statements

We can use many different clauses to change the behaviour of the **SELECT** statement. In this chapter we will describe and give examples of some common ones.

Display the number of rows in the table

We can use the **COUNT()** function to return the number of rows that matches a specified criteria. For example, to display the number of all rows in a table, we can use the following command:

```
mysql> SELECT * FROM testtb;
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| Amy  | Goodridge   | 1991 |
| Mark | Smith       | 1955 |
| John | von Neumann | 1921 |
| John | Jones       | 1985 |
+------+-------------+------+
4 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM testtb;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Remove duplicate rows

Sometimes, when querying data from a table, you might get duplicate rows. To remove these duplicate rows you use the **DISTINCT** clause in the **SELECT** statement. For example, let's say that we want a list of all first name in our table. If we select just the name column from a table, we will get duplicate results:

```
mysql> SELECT name FROM testtb;
+------+
| name |
+------+
| Amy  |
| Mark |
| John |
| John |
+------+
4 rows in set (0.00 sec)
```

Notice how the name John appears twice. To weed out multiple entries, we can use the **DISTINCT** clause:

```
mysql> SELECT DISTINCT name FROM testtb;
+------+
| name |
+------+
| Amy  |
| Mark |
| John |
```

```
+------+
3 rows in set (0.01 sec)
```

Filter records

You can narrow down queries by returning only those where a certain expression is true. To do that, the **WHERE** clause is used. Here is the syntax:

```
SELECT column_name FROM table_name WHERE column_name operator value
```

Here is an example. Let's say that we want to select all rows with the value John in the name column:

```
mysql> SELECT * FROM testtb;
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| Amy  | Goodridge   | 1991 |
| Mark | Smith       | 1955 |
| John | von Neumann | 1921 |
| John | Jones       | 1985 |
+------+-------------+------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM testtb WHERE name='John';
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| John | von Neumann | 1921 |
| John | Jones       | 1985 |
+------+-------------+------+
2 rows in set (0.01 sec)
```

NOTE - make sure to use the single quotes around text values when working with strings.

Here is another example. Let's say that we want to select all rows where the year column is greater than 1980:

```
mysql> SELECT * FROM testtb WHERE year>1980;
+------+-----------+------+
| name | surname   | year |
+------+-----------+------+
| Amy  | Goodridge | 1991 |
| John | Jones     | 1985 |
+------+-----------+------+
2 rows in set (0.00 sec)
```

Remove a row

You can use the **DELETE** command to remove a row from a table. The syntax:

```
DELETE FROM table_name WHERE column_name operator value
```

Here is an example. We have our old table **testtb**:

```
mysql> SELECT * FROM testtb;
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| Amy  | Goodridge   | 1991 |
| Mark | Smith       | 1955 |
| John | von Neumann | 1921 |
| John | Jones       | 1985 |
+------+-------------+------+
4 rows in set (0.01 sec)
```

Let's say that we watn to delete the last row. We can do it by specifying the surname **Jones** with the **WHERE** clause:

```
mysql> DELETE FROM testtb WHERE surname='Jones';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM testtb;
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| Amy  | Goodridge   | 1991 |
| Mark | Smith       | 1955 |
| John | von Neumann | 1921 |
+------+-------------+------+
3 rows in set (0.00 sec)
```

NOTE - make sure to include the **WHERE** clause. If you omit it, all records in the table will be deleted!

LIMIT clause

Sometimes tables contain thousands of rows and if you run a **SELECT** statement to display all rows, you will impact performance and possibly even crash the system. However, you can use the **LIMIT** clause to choose how many rows will be returned in a query. Here is an example:

```
mysql> SELECT * FROM employees LIMIT 5;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      | 1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 |
+--------+------------+------------+-----------+--------+------------+
5 rows in set (0.01 sec)
```

The **employees** table mentioned above contains thousands of rows. Using the **LIMIT** clause, we were able to display only the first 5 rows.

You can also specify the starting position. For example, to start at the position 2 and return 7 rows, we can use the following command:

```
mysql> SELECT * FROM employees LIMIT 2,7;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 |
|  10006 | 1953-04-20 | Anneke     | Preusig   | F      | 1989-06-02 |
|  10007 | 1957-05-23 | Tzvetan    | Zielinski | F      | 1989-02-10 |
|  10008 | 1958-02-19 | Saniya     | Kalloufi  | M      | 1994-09-15 |
|  10009 | 1952-04-19 | Sumant     | Peac      | F      | 1985-02-18 |
+--------+------------+------------+-----------+--------+------------+
7 rows in set (0.00 sec)
```

NOTE - notice that the second row in the employees table was not included in the output above. The **LIMIT 2,7** keyword means return seven rows starting from the third row.

Update the contents of a field

One of the most common tasks when working with MySQL databases is the data update. To update records in an MySQL table, the **UPDATE** statement is used. Here is the syntax:

```
UPDATE table_name SET column1=value1,column2=value2,... WHERE column_name operator
value
```

As you can see from the syntax above, you first need to specify the table name. Second, you need to specify which columns will be modified and their new values after the SET clause. Lastly, you need to specify which rows will be updated using the WHERE clause.

Here is an example. We have our old table **testtb**:

```
mysql> SELECT * FROM testtb;
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| Amy  | Goodridge   | 1991 |
| Mark | Smith       | 1955 |
| John | von Neumann | 1921 |
+------+-------------+------+
3 rows in set (0.00 sec)
```

Let's say that **Amy Goodridge** married and changed her surname to **Bryant**. To update the surname field, we can use the following syntax:

```
mysql> UPDATE testtb SET surname='Bryant' WHERE surname='Goodridge';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM testtb;
+------+-------------+------+
| name | surname     | year |
+------+-------------+------+
| Amy  | Bryant      | 1991 |
```

```
| Mark | Smith       | 1955 |
| John | von Neumann | 1921 |
+------+-------------+------+
3 rows in set (0.00 sec)
```

NOTE - make sure to include the **WHERE** clause; otherwise, the **UPDATE** statement will update all rows in the table!

Sort results

You might have noticed that, when you use the **SELECT** statement to query data, the results are not sorted in any orders. To sort returned results by one or more columns in ascending or descending order, you can use the ORDER BY clause. Here is the syntax:

```
SELECT column1, column2,... FROM table_name ORDER BY column1 [ASC|DESC], column2
[ASC|DESC],...
```

Here is a simple example. We will work with the following table:

```
mysql> SELECT * FROM testtb;
+-------+-------------+------+
| name  | surname     | year |
+-------+-------------+------+
| Amy   | Bryant      | 1991 |
| Mark  | Smith       | 1955 |
| John  | von Neumann | 1921 |
| Aaron | Rogers      | 1995 |
| Brian | Cormier     | 1988 |
+-------+-------------+------+
5 rows in set (0.00 sec)
```

To sort the results by name, we would use the following command:

```
mysql> SELECT * FROM testtb ORDER BY name;
+-------+-------------+------+
| name  | surname     | year |
+-------+-------------+------+
| Aaron | Rogers      | 1995 |
| Amy   | Bryant      | 1991 |
| Brian | Cormier     | 1988 |
| John  | von Neumann | 1921 |
| Mark  | Smith       | 1955 |
+-------+-------------+------+
5 rows in set (0.01 sec)
```

To sort in descending order, we would add the **DESC** keyword after the column name:

```
mysql> SELECT * FROM testtb ORDER BY name DESC;
+-------+-------------+------+
| name  | surname     | year |
+-------+-------------+------+
| Mark  | Smith       | 1955 |
| John  | von Neumann | 1921 |
| Brian | Cormier     | 1988 |
```

```
| Amy   | Bryant      | 1991 |
| Aaron | Rogers      | 1995 |
+-------+-------------+------+
5 rows in set (0.00 sec)
```

Use logical operators

You can use the logical operators **AND**, **OR**, and **NOT** to filter records based on more than one condition and narrow down your selection. Here are the meanings of these operators:

- **AND** - displays a record if both the first condition and the second condition are true.
- **OR** - displays a record if either the first condition or the second condition is true.
- **NOT** - displays a record if the opposite of the condition is true

Here are the examples. We will work with our old **testtb** table:

```
mysql> SELECT * FROM testtb;
+-------+-------------+------+
| name  | surname     | year |
+-------+-------------+------+
| Amy   | Bryant      | 1991 |
| Mark  | Smith       | 1955 |
| John  | von Neumann | 1921 |
| Aaron | Rogers      | 1995 |
| Brian | Cormier     | 1988 |
+-------+-------------+------+
5 rows in set (0.00 sec)
```

AND example:

```
mysql> SELECT * FROM testtb WHERE name='Amy' AND surname='Bryant';
+------+---------+------+
| name | surname | year |
+------+---------+------+
| Amy  | Bryant  | 1991 |
+------+---------+------+
1 row in set (0.01 sec)
```