Introduction to MySQL

The EasyLearn

Created By :- Ankit M Patel

What is MySQL?

- An Open Source, platform-independent, Enterprise-level, multi-threaded, relational database management system that stores and retrieves data using the SQL(Structured Query Language).
- SQL is 4th generational language in which we have to tell what is to be done instead of how it is to be done
- It is very popular and much cheaper the oracle which is also another RDBMS package.

Why use MySql?

- MySQL server can handle very large databases.
- Offers rich and very useful set of functions.
- Connectivity, speed and security make MySQL very suited for accessing database on a network.
- A lot of contributed software available.
- Multi-threaded request-handling using kernel thread.
- Replication features.
- Very actively developed.
- Memory leak proof.
- Each student can install MySQL locally.
- Easy to use Shell for creating tables, querying tables, etc.

Connecting to MySQL **shell**/**console**.

Basic Queries

- Once logged in, you can try some simple queries.
- For example:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 3.23.49   | 2002-05-26   |
+-----------+--------------+
1 row in set (0.00 sec)
```

- Note that most MySQL commands end with a semicolon (;)
- MySQL returns the total number of rows found, and the total time to execute the query.

Basic Queries

- Keywords may be entered in any lettercase.
- Queries in mysql is not case sensitive. So all below command are valid and will give same result.

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Basic Queries

- You can also do mathametical operations like below:

```
mysql> SELECT (4+1)*5;
        25
```

Basic Queries

- You can get both date & time using now() functions.

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2018-09-06 00:15:33 |
+---------------------+
```

Multi-Line Commands

- mysql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line.
- Here's a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
+--------------------+--------------+
| root@localhost |     2021-12-25   |
+--------------------+--------------+
```

How to get list of all database?

```
mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.01 sec)
```

How to create new database?

- To create a new database, issue the "create database databasename" command:
  - `create database theeasylearn;`
- To the select a database as current database, issue the "use databasename" command:
  - `use theeasylearn;`

How to select database to work

- To work with databases, first we have select database as active database.
- To do that use SQL command
  - USE databasename
  - example
  - USE theeasylearn

How to get list of all tables in current database

- Once you have selected a database, you can view all existing tables in it using command:

```
show tables;
```

How to remove database

- To remove database with all tables and all rows in it, use command
  - drop database databasename
  - drop database theeasylearn

Let us create table

- Let's create a table for storing PERSON data.
- Table: person
  - name: VARCHAR(20)
  - surname: VARCHAR(20)
  - Weight: int
  - gender: char(1)
  - birthdate: DATE

VARCHAR is usually used to store string data.

Creating a Table

- To create a table, use the CREATE TABLE command:

```
CREATE TABLE person (
      id int PRIMARY KEY AUTO_INCREMENT,
      name VARCHAR(20),
      surname VARCHAR(20),
      weight int(3),
      gender CHAR(1),
      birthdate DATE
);

Query OK, 0 rows affected (0.04 sec)
```

Showing Tables

- To verify that the table has been created:

```
mysql> show tables;
+--------------------+
| Tables_in_test     |
+--------------------+
| pet                |
+--------------------+
1 row in set (0.01 sec)
```

Describing Tables (display structure of the table)

- To view a table structure, use the DESCRIBE command:
- `describe person;`
- Refer following output.

```
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(32) | YES  |     | NULL    |                |
| surname   | varchar(32) | YES  |     | NULL    |                |
| weight    | int(3)      | YES  |     | NULL    |                |
| gender    | char(1)     | YES  |     | NULL    |                |
| birthdate | date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Deleting a Table

- To delete an entire table with data, use the DROP TABLE tablename command:

```
drop table person;
Query OK, 0 rows affected (0.02 sec)
```

SQL Select

- The SELECT statement is used to pull information from a table.
- The general format is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

Selecting All Data

- The simplest form of SELECT retrieves everything from a table

```
mysql> select * from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1999-02-04 | NULL       |
| Claws    | Gwen   | cat     | f    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1999-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1998-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    |      | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
8 rows in set (0.00 sec)
```

Selecting Particular Rows

- You can select only particular rows from your table.
- For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1998-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)
```

Selecting Particular Rows

- To find all animals born after 1998

  ```
  SELECT * FROM pet WHERE birth >= "1998-1-1";
  ```

- To find all female dogs, use a logical AND

  ```
  SELECT * FROM pet WHERE species = "dog" AND sex =
      "f";
  ```

- To find all snakes or birds, use a logical OR

  ```
  SELECT * FROM pet WHERE species = "snake"
  OR species = "bird";
  ```

Selecting Particular Columns

- If you don't want to see entire rows from your table, just name the columns in which you are interested, separated by commas.
- For example, if you want to know when your pets were born, select the name and birth columns.
- (see example next slide.)

Selecting Particular Columns

```
mysql> select name, birth from pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1999-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1999-08-27 |
| Bowser   | 1998-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
+----------+------------+
8 rows in set (0.01 sec)
```

Sorting Data

- To sort a result, use an ORDER BY clause.
- For example, to view animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Bowser   | 1998-08-31 |
| Chirpy   | 1998-09-11 |
| Fluffy   | 1999-02-04 |
| Fang     | 1999-08-27 |
+----------+------------+
8 rows in set (0.02 sec)
```

Sorting Data

- To sort in reverse order, add the DESC (descending keyword)

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fang     | 1999-08-27 |
| Fluffy   | 1999-02-04 |
| Chirpy   | 1998-09-11 |
| Bowser   | 1998-08-31 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
+----------+------------+
8 rows in set (0.02 sec)
```

Working with NULLs

- NULL means missing value or unknown value.
- To test for NULL, you cannot use the arithmetic comparison operators, such as =, < or <>.
- Rather, you must use the IS NULL and IS NOT NULL operators instead.

Working with NULLs

- For example, to find all your dead pets (what a morbid example!)

```
mysql> select name from pet where death >IS NOT
    NULL;
+---------+
| name    |
+---------+
| Bowser  |
+---------+
1 row in set (0.01 sec)
```

Pattern Matching

- MySQL provides:
  - standard SQL pattern matching;
  - regular expression pattern matching, similar to those used by Unix utilities such as vi, grep and sed.
- SQL Pattern matching:
  - To perform pattern matching, use the LIKE or NOT LIKE comparison operators
  - By default, patterns are case insensitive.
- Special Characters:
  - _ Used to match any single character.
  - % Used to match an arbitrary number of characters.

Pattern Matching Example

- To find names beginning with ‘b’:

```
mysql> SELECT * FROM pet WHERE name LIKE "b%";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

Pattern Matching Example

- To find names ending with `fy':

```
mysql> SELECT * FROM pet WHERE name LIKE "%fy";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

Pattern Matching Example

- To find names containing a ‘w’:

```
mysql> SELECT * FROM pet WHERE name LIKE "%w%";
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Pattern Matching Example

- To find names containing exactly five characters, use the _ pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE "_____";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

Regular Expression Matching

- The other type of pattern matching provided by MySQL uses extended regular expressions.
- When you test for a match for this type of pattern, use the REGEXP and NOT REGEXP operators (or RLIKE and NOT RLIKE, which are synonyms).

Regular Expressions

- Some characteristics of extended regular expressions are:
  - . matches any single character.
  - A character class [...] matches any character within the brackets. For example, [abc] matches a, b, or c. To name a range of characters, use a dash. [a-z] matches any lowercase letter, whereas [0-9] matches any digit.
  - * matches zero or more instances of the thing preceding it. For example, x* matches any number of x characters, [0-9]* matches any number of digits, and .* matches any number of anything.
  - To anchor a pattern so that it must match the beginning or end of the value being tested, use ^ at the beginning or $ at the end of the pattern.

Reg Ex Example

- To find names beginning with b, use ^ to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP "^b";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

Reg Ex Example

- To find names ending with `fy', use `$' to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP "fy$";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```