MySQL Notes

Iñaki Lakunza

June 9, 2024

Contents

1 Introduction

A **Database** is a collection of data stored in a format that can easily be accessed.

In order to access the database we use a software application named **Database Management System (DBMS)**. We connect tot the DBMS and give it instructions for querying or modifying data. The DBMS will execute our isntructions and send the results back.

There are two main DBMS categories: **Relational** and **NonRelational (NoSQL)**.

In **Relational** Databases we store data in tables that are linked to each other using relationships. Each table stores data about a specific type of object.

SQL (or Sequel) is the language we use to work with these relational DBMS.

In **NonRelational** databases we do not have tables or relationships.

2 The Select Statement

The first step to query data from a database is to select a database.

USE is a keyword to select the database we will be working. We can also select it by double clicking it on the workbench.

With the **SELECT** keyword we will select which columns we want to retrieve. We can then specify which columns we want, or we can select all using *.

Then we have to choose from which table we want to retrieve the columns, using the **FROM** keyword.

Whenever we have multiple SQL statements we have to terminate eacy statement using a semicolumn

```
USE sql_store;

SELECT *
FROM customers
```

And, if we want to run the script we can go to query and press to *execute*. Or, in Windows, we can press *ctrl+mayus+enter*.

If we want to pick rows which have some specific attributes we can use the **WHERE** clause.

For instance, if we execute the following code we will just retrieve the row with *customer_id* equal to 1.

```
USE sql_store;

SELECT *
FROM customers
WHERE customer_id = 1
```

We can also sort the data using **ORDER BY**. We have to specify the column which will be used to sort the data.

By default it will be sorted by **ascending order**, but we can do it in descending order using **DESC**.

(**For clarity, we should always specify how we want to sort it, using ASC or DESC**).

We can use – for writting comments, and if we want to write comments which are larger than one row we can use * and */.

```
USE sql_store;

SELECT *
FROM customers
-- WHERE customer_id = 1
ORDER BY first_name DESC
```

3 The SELECT Clause

Apart from the original columns, we can retrieve different combinations of them.

We can combine the tables between them or we can use different values for that.

For instance, with the following code we will retrieve the column points and another column with the name *points + 10*, which will have the values of the table *points* summed by 10:

```
SELECT points, points + 10
FROM customers
```

We have addition, substraction, multiplication (*), division (/), modulo (%).

If our line is getting to long, we can divide it in different rows. And, we have to take into account that if not specified, the name of the column will be the line used to get it, but we can change it using **AS**:

```
SELECT
    last_name,
    first_name,
    points,
    (points + 10) * 100 AS 'discount␣factor'
FROM customers
```

We may have repeated elements. If we want to get just UNIQUE elements, we can use the **DISTINCT** keyword:

```
SELECT DISTINCT state
FROM customers
```

4 The WHERE clause

We can use the **WHERE** clause to filter data.

These are the operators that we have:

$>, >=, <, <=, =, <>$ $(or !=)$.

For text, we have to enclose it in single quotes (') or double quotes (").

When working with DATES, even though they are not text, we have to use quotes to compare them:

```
SELECT *
FROM customers

WHERE birth_date > '1990-01-01'
```

5 The AND, OR and NOT Operators

For combining conditions:

```
SELECT *
FROM customers

WHERE birth_date > '1990-01-01' AND points > 1000
```

And the **OR** clause works similarly.

We have to remember that the AND operator is always evaluated faster than the OR operator (similarly to + and *) But we can always use parenthesis to better define the wanted order.

The NOT operator is used to negate a condition.

6 The IN Operator

If we want to use a conditions with multiple ORs, we could write it the following way:

```
SELECT *
FROM Customers
WHERE state = 'VA' OR state = 'GA' OR state = 'FL'
```

But this code is not clean and is quite tedious.

Instead, we can use the **IN** operator:

```
SELECT *
FROM Customers
WHERE state IN ('VA', 'FL', 'GA')
```

And we can also use the NOT operator, using **NOT IN**.

7 The BETWEEN Operator

Instead of:

```
WHERE points >= 1000 AND points <= 3000
```

We can write:

```
WHERE points BETWEEN 1000 AND 3000
```

8 The LIKE Operator

We can use the **LIKE** operator to retrieve elements which follow a concrete pattern.

For that purpose we use the % character, it will define a pattern, it does not matter if it is uppercase of lowercase.

The % symbol does not have to be at the end of the text, it can be placed before or/and after.

On the other hand, we can use _ for just character, in that position there will just be able to place a single character (we can use multiple _).

```
WHERE last_name LIKE '\%a\%'
WHERE last_name LIKE '___'
```

9 The REGEXP Operator

The following 2 lines are identical:

```
WHERE last_name LIKE '\%field\%'
WHERE last_name REGEXP 'field'
```

If we want for instance to start with a chosen word, we can use the ^ symbol, so **we represent the beginning of a string**.

And the same way, we can use a dollar sign $ to represent the end of a string:

```
WHERE last_name REGEXP '^field' -- The last name must begin with 'field'
WHERE last_name REGEXP 'field$' -- The last name must end with 'field '
$
```

And if we want to use different patterns we can use |:

```
WHERE last_name REGEXP '^field|mac|rose'
```

So, it should start with 'field', or have 'mac' or 'rose'.

On the other hand, we can do different options, for instance, if we want to find last names which contain 'ge' or 'ie' or 'me':

```
WHERE last_name REGEXP '[gim]e'
WHERE last_name REGEXP '[a-h]e' -- Instead of [abc...h]
```

10 The IS NULL Operator

We will have some missing elements, these will be 'filled' with a 'NULL'.
NULL means the absence of a value.

For instance, if we want to get the customers which do not have a phone number associated:

```
SELECT *
FROM customers
WHERE phone IS NULL
```

And obviously we can use **IS NOT NULL** the same way.

11 The ORDER BY Operator