# Programming in SQL
# Notes

Alejandro Florido Reyes

Academic Year 2019-2020

## Contents

**1 Queries** **1**
1.1 Basic Syntax to make SQL queries . . . . . 1
1.2 SELECT . . . . . 2
1.3 WHERE . . . . . 3

## 1 Queries

Queries are the method that we will use to obtain information from a database. A query is a set of lines written in SQL code.

### 1.1 Basic Syntax to make SQL queries

Firstly, I will write the essential sentence we have to keep in mind while we are writing a query:

SELECT AGGREGATE_FUNCTION(column_name1), ...
FROM (register_name1)
JOIN (register_name2) ON (column1_register1=column3_register2, ...), ...
WHERE restriction_1 AND/OR restriction_2 ... ORDER BY column_name1, ...
HAVING restriction_aggregate1 AND/OR ... GROUP BY column_name1, ...

Note. Notice that we are separating the different parts of the query to make it easier to read and understand. In practice, do this is NOT a problem, because the software will ignore weird whitespaces and separate lines like these. Don't be afraid of use it.

Like we can see in the previous sentence, a SQL query is similar to an order that we pronounce with a very specific words. First of all, you SELECT what information you want to see when you RUN this QUERY, then you specify the registry (place, file) FROM where you will obtain that information (if there are several registries, you will have to JOIN those registries in the right way, counting ON possible identical columns among those).

Once you have chosen the registries and the columns appropriate for your task, frequently, it will be necessary to apply some RESTRICTIONS to receive only the information you want (WHERE) and, if you want to be clearer, could be useful ORDER the results BY some specific COLUMNS.

Finally, if you used some AGGREGATE_FUNCTION at the beginning, you can add specific restrictions for them after type HAVING and, overall, you will have to GROUP the resulting columns BY some of the original columns.

Note. You could type any command, name or column in uppercase or lowercase, it won't change the meaning of the query.

## 1.2 SELECT

SELECT can do far more than simply retrieve and display data, but that is the basic function of this statement.

If you want to pull data from a table and then display the results you should write:

SELECT * FROM table_name

The symbol * means "All columns" but if you want to put a certain columns, you only have to write the names of those columns separated by comma (enumerate it). Example:

SELECT column1_name, column2_name, column3_name FROM ...

Note. One important detail is that it is useful to get used to write the semicolon (;) after a COMPLETE SQL statement (for example, the last two examples), because it's needed if you want to run several statements in the same script (for example, in a script to add data one to one in a table).

- **Calculations, another functionality of SELECT statement.**

  You can do calculations on the columns and include them in your query result, for example:

  SELECT column_name * 1.5 AS new_column_name, ... FROM...

  These calculated columns are not stored in the database, but rather calculated and displayed to us every time we run this query.

- **Put a name to your new column (or replace an old one).**

  Notice that here we used AS to choose a name for the new column we have just created.You can use this statement to replace the name of a column in the results.

  Note. One last thing: if you will write a name with several words, use underscore(_) instead of "space" ( ) to avoid errors.

- **Some functions you could use while you are calculating.**

  – round(x,y)

  This function consist of 2 variables, x, which is the number we want to round and y, which is the number of decimal places to round to.

  –

- **Basic mathematical operators.**

  They are essentially the same you use in MATLAB, but there is a new one. % divides two numbers, but **returns the remainder**.
  Not equal could be typed like != or ¡¿, in general. Microsoft Access and IBM DB2 only allow the use of ¡¿.

- **Text concatenation**

  You can use it to join different pieces of text, creating an unique sentence as consequence. You can add numbers (they will be transformed in text when the operator finds them) and can add text manually using ' '. A example would be:

  SELECT street_address || ' ' || city || ', ' || state || ' ' || zip AS Ship_address FROM customer

  This query would return a column named "Ship address" whose rows (each of them) would give to the user the street, city, state and zip (postal) code of one of the customers.

  Note. Here we have used —— to concatenate but MySQL and others require using a CONCAT() function.

## 1.3 WHERE

The statement where acts like a filter, we add conditions after the clause WHERE and then those conditions reduce the number of dates we receive from the query.

- **Using WHERE on numbers**
  - **BETWEEN**
    This function impose that results have to be in an inclusive range of values. The syntax is:

    $$\text{WHERE column\_name BETWEEN x AND y } /x, y \in \Re$$

    In fact, the command BETWEEN is the result of a combination of conditions (sth ¿= x AND sth ¡= y) and combine math operators and logic operators can give us every modification of BETWEEN that we could need.
  - **Logic operators: AND and OR**
  - **IN**
    Instead of using several OR sentences, we could use the command IN to sum up those sentences. For example, if we type:

    $$\begin{gathered}\text{WHERE column\_name} = 3\\ \text{OR column\_name} = 6\\ \text{OR column\_name} = 9\\ \text{OR column\_name} = 12\end{gathered}$$

    Whose result would be the rows where column_name = 3, 6, 9 or 12; we could sum up the previous code using IN in this way:

    $$\text{WHERE column\_name IN (3,6,9,12)}$$

    Likewise, you could use NOT IN, to obtain the rows where column_name $\neq$ 3, 6, 9 and 12. In fact, we could add NOT in any condition to reverse its meaning.
  - **An interesting application of %**
    We'll explain this with an example. If you had a table which has one column with the months of the year and you wanted to select only the rows with even months, how would you do it? One way to solve this problem is to use the operator % in the next way:

    $$\text{WHERE column\_name\_months \% } 2 = 0$$

    Note. In Oravle the operator % is not used, instead of it, you can use the function MOD().

- **Using WHERE on Text**

  When you type text in your code, you will have to type it between simple quotes because this is the way in which the SQL software difference between simple text (part of the data) and column names. This rule applies to every text operation.

  If you forget this and type the text without simple quotes, the programm will send you an error because he won't find a column with that name (unless you, casually, got a column with the same name of the value, which is not usual and, even if it was, the code wouldn't do what you want, so it will be even a bigger problem).

  - **The length() function**
    In a text between single quotes (hereinafter "string"), we can use this function to calculate the number of characters and get it back like the result. In this way, if we apply LENGTH(column_name) in a column with strings, we will receive a column in which each row is the number of characters of one of them.
  - **Wildcards and LIKE**
    In a string, % is a wildcard which represents any number of characters of any type and _ is a wildcard which represents any character (but only one character), they are very used with the function LIKE, which let you look for any word which satisfies a certain condition. For example, let's look for every string which starts with A, its third character is z and whose last letter is s; the sentence we would have to type in our script would be:

WHERE column_name_strings LIKE 'A_z%s'

There are other handy functions like INSTR, SUBSTR and REPLACE that would be good to know, but we'll explain them later to do this piece of lesson shorter.

To conclude this part, it's important to say that sometimes research on regular expressions related to strings could be useful to solve problems, so feel free to research on your own like a complementary activity to grow like a programmer.

- **Using WHERE on booleans**

  In general, you can use true/false or 1/0 to indicate the result of a boolean operation. There is an exception: SQLite don't allow the use of true/false, so for your own sake it would be a good idea to get used to the 1/0 manner.

  Another important thing to know is that you are not forced to write =1 or =true in a sentence if you don't want to, since, behind the scenes, every WHERE condition boils down to a boolean expression. For example, typing

  WHERE column_name=1 ...

  is equivalent to type:

  WHERE column_name ...

  Otherwise, you must explicitly type =0 or =false to type false conditions, or so it could look... In fact, you can add a NOT particle before the name of the column and it will be the same as type =0 or =false (because you are reversing the simple way to type a true condition). Continuing with the previous example:

  WHERE column_name=0 ...

  is equivalent to type:

  WHERE NOT column_name ...

- **Handling NULL**