SQL

What is "SQL"?

SQL, for Structured Query Language, is the standard language for working with relational databases.

SQL programming is used to insert, search, update, and delete records in a database.

There are many DBMSs that use SQL: MySQL, Oracle, Ms SQL Server etc ...

Some basic rules

- The star symbol '*' selects all the columns of the specified table.
- String (string) must be surrounded by a simple quote.
- Numbers should not be surrounded by single or double quotes.
- Date data must be surrounded by simple quotes in the 'YYYY-MM-DD' format.

SQL databases

The 'SELECT'

command The SELECT command will allow you to retrieve data from the tables of the database. This is part of the data manipulation language responsible for querying the data in the database.

This is the most used SQL command, here is its syntax:

SELECT [DSTINCT | ALL] {* | [fieldExpression [AS newName]}
FROM tableName [alias]
[WHERE condition]
[GROUP BY fieldName]
[HAVING condition]
[ORDER BY fieldName]

- **SELECT** is the SQL keyword that says you want to retrieve data. Following SELECT, here are the possibilities:
  - **[DISTINCT | ALL]** are optional keywords. DISTINCT avoids redundancies in the results (no duplicates). The default value is ALL.
  - **{* | fieldExpression [AS newName]}**: Star '*' selects all columns in the specified table. 'fieldExpression' performs some calculations on the specified fields, for example adding numbers the sum or other.

- **FROM tableName** is the keyword that allows you to specify the table or tables on which to retrieve data. At least a table must be indicated. The tables must be separated by a comma. It is possible to give an alias to a table.

- **WHERE** is optional. This keyword can be used to specify criteria when searching for data.

- **GROUP BY** is used to join records with the same field values.
- **HAVING** is used for specified conditions when using 'GROUP BY'.
- **ORDER BY** to sort the results.

Examples of using the SELECT command:

For the rest, we will start from an existing database.
This database contains only two tables: a "directors" table and a "movies" table.

Here is an overview of the contents of these two tables.
Producer table:

director_id	name	date_of_birth	gender
1	Steven Spielberg	1946-12-18	Male
2	George Lucas	1944-05-14	Male

Movie table:

movie_id	title	year_released	director_id
1	E.T L'extraterrestre	1982-10-02	1
2	Ready Player One	2018-01-01	1
3	Star Wars 7	2015-12-04	2
4	Les infiltrés	2006-01-01	3

- To retrieve and display only the title and the release date of all my films, I use this command:

```
SELECT title, year_released
FROM movies
```

- To recover all the columns of my 'movies' table:

```
SELECT *
FROM movies
```

- Let's imagine that we want the title of the movie and its release date in a single field. We could do this:

```
SELECT Concat (title, '(', year_released, ')') AS
Concat
FROM movies
```

- Show the filmmakers with their year of birth only:

```
SELECT name, LEFT (`date_of_birth`, 4) AS 'year_of_birth
'
FROM directors
```

The' WHERE 'condition The WHERE

command will restrict some results to a particular condition.

Here is a basic syntax using WHERE:

```
SELECT *
FROM movies
WHERE director_id = 1
```

This command will allow to recover all the films that have been made by the director whose id is 1.

We can combine several conditions thanks to the logical operators.

For example, we only want movies whose director id is 1 and the year of release is greater than 2000:

```
SELECT *
FROM movies
WHERE director_id = 1 AND year_released> '2000'
```

The WHERE clause, when used with the keyword IN, affects only the rows whose values correspond to the list of values provided in the key word IN.

For example:

```
SELECT *
FROM movies
WHERE director_id IN (1,2,5)
```

You can also exclude some results directly with the keyword NOT IN

For example:

```
SELECT *
```

```
FROM movies
WHERE director_id NOT IN (1,2,5)
```

The 'INSERT INTO'

command SQL uses the INSERT command to store data in a table. The INSERT command creates a new row in the table to store the data.

Here is the syntax of the INSERT command:

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

- **INSERT INTO tableName** is the command that will specify in which table to add the new record.
- **(column1, column2, ...)** specifies the columns that will be updated in the new record.
- **(value1, value2, ...)** specifies the values to update in the new record.

Examples of using the 'INSERT' command:

```
INSERT INTO directors (name, date_of_birth, gender)
VALUE ('Martin Scorsese', '1942-11-17', 'Male')
```

The order of the columns has no effect on the INSERT query until the correct values have been mapped to the correct columns.

By default, MySQL inserts NULL values into ignored columns in the INSERT query.

The INSERT command can also be used to insert data into a table from another table. The basic syntax is as follows:

```
INSERT INTO table_1 SELECT * FROM table_2
```

The 'DELETE'

command The SQL command 'DELETE' is used to delete the rows that are no longer required from the tables in the database. It removes the whole line

of the table. The DELETE command can delete multiple rows from a table in a single query.

Once a line has been deleted, it can not be retrieved. It is therefore strongly recommended to make backups of the database before deleting data from it. This can allow you to restore the database and view the data later if necessary.

Here is the syntax for the DELETE command:

```
DELETE FROM tableName
[WHERE Condition]
```

If the WHERE clause is not specified, all rows in the table will be deleted.

I want to remove all films directed by George Lucas:

```
DELETE FROM movies WHERE director_id = 2
```

The 'UPDATE'

Command The SQL 'UPDATE' command is used to update rows in a database table. The 'UPDATE' command can be used to update one or more fields at a time. It can also be used to update a table with values from another table.

Here is the syntax of the UPDATE command:

```
UPDATE tableName
    SET columnName = newValue
[WHERE condition]
```

- **UPDATE tableName** is the command that will specify which table to update the record in.
- **SET columnName = newValue** is the name and value of the field to update.

Example:

```
UPDATE directors
Set name = 'Steven Allan Spielberg'
WHERE name = 'Steven Spielberg'
```

SQL Advanced

Sorting and ORDER BY command

In this section, we will see how to sort the results of our query. Sorting is simply reorganizing our query results in a specified way. Sorting can be done on a single column or multiple columns. This can be done on numbers, character strings as well as date data types.

The 'ORDER BY' clause is used to sort the result sets of the query in ascending or descending order. It is used in conjunction with the SELECT query.

Here is its syntax:

```
SELECT *
FROM tableName
ORDER BY fieldName (s) [ASC | DESC]
```

ASC allows sorting in ascending order.
DESC allows sorting in descending order.
By default, **ASC** is used.

Example:

I want to be able to retrieve the list of my movies sorted by release date (from newest to oldest):

```
SELECT *
FROM movies
ORDER BY year_released DESC
```

The 'GROUP BY' and 'HAVING' commands

a) *GROUP BY*

The GROUP clause BY is an SQL command used to group rows with the same values.

Queries containing the GROUP BY clause are called grouped queries and return only one row for each grouped item.

Here is its syntax:

```
SELECT statement ...
GROUP BY columnName (s)
```

Example:

- Among the directors, I want to know the possible unique values for the attribute 'gender':

```
SELECT gender
FROM movies
GROUP BY gender
```

Result:

gender
Female
Male

- One could also choose to group movies by director AND release date.

```
SELECT director_id, year_released
FROM movies
BY BY director_id, year_released
```

If the director is the same but the release year differs, a line will be added. On the other hand, if the director AND the release date are the same, they will be grouped on a single line.

- It is possible to use the clause 'GROUP BY' to use the functions of aggregations. For example, I want to know the number of director of the masculine and feminine genre.

```
SELECT gender, COUNT (director_id)
FROM directors
GROUP BY gender
```

b) ***HAVING***

In some cases, we will want to restrict the results returned by the GROUP BY clause.

In such cases, we will use the HAVING clause.

Suppose we wanted to know all the movies for director 1:

```
SELECT *
FROM movies
GROUP BY director_id, year_released
HAVING director_id = 1
```

Wildcards

Wildcards are characters that search for data that matches complex criteria. Wildcards are used in conjunction with the LIKE comparison operator or the NOT LIKE comparison operator.

a) ***LIKELIKE***

Theoperator is used in the WHERE clause of SQL queries. This keyword allows to search on a particular model. For example, it is possible to search for records whose value of a column begins with a particular letter.

Here is the syntax:

```
SELECT *
    FROM tableName
WHERE columnName LIKE model
```

b) **NOT LIKE**

Theoperator is **NOT LIKE** used in the same way as **LIKE** but returns the opposite result.

c) Percent%

The percent character is used to specify a pattern of zero (0) or more characters. It has the following basic syntax:

```
SELECT statement
FROM table
WHERE fieldName LIKE 'xxx%'
```

For example:

- Search for all directors whose names begin with 'Steven':

```
SELECT *
FROM directors
WHERE name LIKE 'Steven%'
```

- Find all directors whose name contains the letter 's':

```
SELECT *
FROM directors
WHERE name LIKE'% s % '
```

d) Underscore _

The underscore character is used to match exactly one character.
It has the following basic syntax:

```
SELECT statement
FROM table
WHERE fieldName LIKE 'xxx_'
```

Suppose we want to search for all movies released in 200x's where x is exactly one character that could have any value:

```
SELECT *
FROM movies
WHERE year_released LIKE '200_'
```

e) ESCAPEESCAPE

Thekeyword is used to escape special or wildcard characters such as percentage (%) and underscore (_) if they are part of the data.

For example, we want to look the film 'Ten%', here the request to use:

```
SELECT *
FROM movies
WHERE title LIKE 'Ten%' ESCAPE '%'
```

Functions

The functions of aggregates

Aggregate functions can be produced easily summarized data from our database.

There are several aggregate functions in SQL:

a) ***COUNT***

The COUNT function returns the total number of values in the specified field. This works on both numeric and non-numeric data types. All aggregate functions by default exclude NULL values before working on the data.

COUNT (*) is a special implementation of the COUNT function that returns the number of all rows in a specified table. COUNT (*) also takes into account null and duplicate values.

For example, to count the total number of movies in my database:

```
SELECT COUNT (movie_id)
    FROM movies
```

b) ***MIN***

The MIN function returns the smallest value specified in the table.

For example, we want to know the oldest movie in our database:

```
SELECT MIN (year_released)
    FROM movies
```

c) ***MAX***

The MAX function is the inverse of the MIN function: it returns the largest value specified in the table.

For example, we want to know the most recent movie from our database:

```
SELECT MAX (year_released)
    FROM movies
```

d) ***SUM***

The SUM function returns the sum of all the values of a specified column. SUM only works on numeric fields.

For example, if you want to know the sum of payments made:

```
SELECT SUM (payment)
    FROMPayments
```

e) ***AVG***

The AVG function returns the average value of a specified column.

For example, if you want to know the average age of contacts:

```
SELECT AVG (age)
    FROM contacts
```

Custom functions

In SQL, you can create your own functions.
We will not see this part in this course.