SQL

You

February 18, 2025

Abstract

Your abstract.

1 Basics of SQL

1.1 what is it

Let's imagine that we have a huge library in our home and we would like to organize them by different labels for example author, year of publication etc... . Essentially what we have is a data base and a programing laguage to consult or query such data base. This**Strucuture Query Language** is called**SQL**

1.2 How to start the different features

1.2.1 XAMPP and PhPmyAdmin

We will work with this and for start running it we should just type in our terminal **sudo /opt/lampp/lampp start** and **sudo /opt/lampp/lampp stop**

1.2.2 mysql-workbench

This feature can not be running at the same time as the previous one so we should follow certain steps:

- If we have initialized the xampp before we should stop it
- then star mysql alone **sudo systemctl start mysql**
- now type **mysql-workbench** and it should run with no errors. It should open the localhost automatically.
- we can check the system doing the following **sudo systemctl status mysql**

1.3

1.4 Data base

A data base is essentially information gathered in a document or group of documents. For instance an EXCEL document could be a data base. There are 2 types of data bases :

- **Relacionales**- They are set in tables and are related to each other. For example a data base could be formed by a table of customers and a table of products, where each product correspond to one or more customers
- **No Relacionales** - No SQL.

1.4.1 Components of a data base

- **Tables**-
- **Rows and columns of the tables**
- **Primary Key** it is the identification of each row or column in our table. For example, the index could be a Primary Key. With a primary key we can separate two row with the same data inside, for example two customers who are called the same

1.5 Data base Managment system-DBMS

It is a software meant to allow us to work with our data base. In this course we will use MySQL anf MySQL Workbench, which is a visual tool. For running MySQL Workbench we have to type in our terminal **mysql-workbench**.

1.5.1 Create a DB in workbench

Once we open the program, we will enter the scheme label and with right click we create a new schema which will consist of three tables. Once our new schema is created, we create a new table called usuarios:

- usuariosID we select this table to be a PK(Primary Key, and to have not null values) PK means that each value in this column will be unique. In addition we will select Autoincrement, which means that each time we add a new row this will create a new ID number which is the previous one plus 1.
- nombre
- email
- fecha registro

Now we will create two more tables and link them by a **foreign key**. A foreign key is essentially a column or batch of columns in a table whose values correspond to the values of the primary key in the other column.
To do this properly we select **alter table**

2 Data consult or query

2.1 Select

This sentence enables us to grab certain data from our DB. The first word will always or almost every time **select**

- ***** this symbol will grab all the columns of a certain table. For selecting a certain column of the table, just type the column name, and if u require more than one, just type the names separating them by **,**
- **from**

So the full sentence will be **select * from** and the table name. Furthermore, we can export the selecting by clicking **export/import**.

2.2 Distinct

This sentence grabs a certain and UNIQUE value of a certain column. In order to define it more precisely **select** is a sentence and **distinct** is a clause (cláusula in Spanish). Both are SQL commands, but slightly different. It is like when in pandas we use the function **.unique()**. The position **distinct** takes will always be right after **select**.
This will grab the unique values in that column. So the whhole sentence would be **select distinct** genero **series**.

2.3 Order by

This sentence enables us to sort a certain column by a certain value. We will use the clause in a certain position. It is important to know which position it takes. **order by** will be at the very end of the query.

A full sentence will be **select** titulo, duracion **from** episodios **order by** duracion

select titulo, duracion **from** episodios **order by** duracion **desc**

2.4 Limit

This allow us to grab a certain number of results. Essentially like when in pandas we type **df.head(10)**. This clause will be typed right after **order by**, so at the end of the sentence.

select titulo, duracion **from** episodios **order by** duracion **desc** **limit** 10

2.5 Where

This clause will apply a filter to our table. So we will just grab the results according to a certain condition. For example:

select * **from** series **where** gender = 'drama'

2.5.1 Comparison operators

- = igualdad
- <> desigualdad
- < *menorque* > *mayorque*
- <= *menoroigualque* >= *mayoroigualque*

2.5.2 logic operators

This will be an additional clause to **where**

- and - enables to add up more conditions
- or - enables to select one condition or another
- not - enables to select a condition that we want it not to be in our query

2.6 In, not in

This a clause is and adding to the **where** clause. Essentially this will include (IN) or exclude (NOT IN) the columns that meet the condition.

2.7 Like

This clause is and adding to the **where** clause. It will allow us to search a certain condition in our table. For example if we want to search all the row that contain the word 'The'.

- 'The' - will grab any row that contains this word
- '%The' - will grab any row that end with this word
- 'The%' - will grab any row that begins with this word

2.8 Adding functions - funciones de agregado

This functions allow us to make some reckon to a bunch of values and get a certain value. For example the clause **sum(column name)**. We can also add a name to the result that is obtained with the clause **as** name, the first after our function.

- sum() as :
- count():
- max(), min() :
- avg() :

2.9 Group by

This will grab rows that meet that same value, for example all the episodes of a certain series.

select serie_id, **avg**(duracion) **as** avg_duracion, **sum**(duracion) **as** sum_duracion **from** series

where serie_id **IN** (1,2)

group by serie_id

This code will select serie_id and make the avg of duracion and the sum and show us the result in two new cols such as avg_duracion and sum_duracion. With **where in** it will show us just the ones corresponding to number 1 and 2 of serie_id. and then it will group the result by if the belong to number 1 or 2.
For example, if we want to count how many episodes we for for each series:

select serie_id, count(episodio_id) as count_episodios **from** Episodios

group by serie_id

2.10 Having

This clause requires to be apply after the **group by**. This enables us to filter grouped data that meet a certain condition. It is the same idea as **where** but once the data are grouped. For example, if we want to find all the series that have more than 10 episodes in total.

select serie_id, count(episodio_id) as count_episodios **from** Episodios

group by serie_id

having count(episodio_id) > 10

3 Complex querys

3.1 Join

This enables us to combine tables and grab information of both tables. In the Figure 1 1 we can observe how this works.

3.1.1 Inner join

This allows us to merge two tables which have a common column. Then we will obtain a new table where the rows are fit according to this key column as in Figure 2.

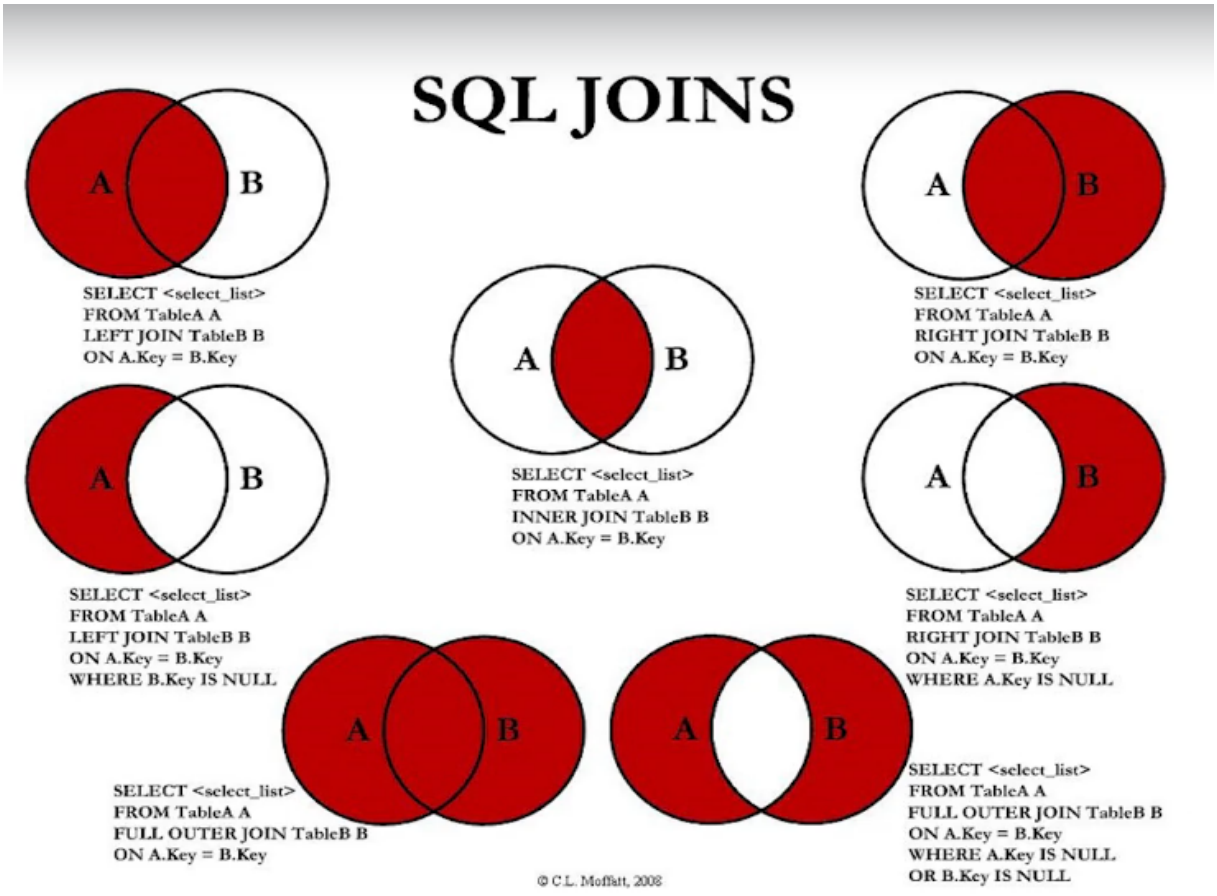


Figure 1: Tipos de JOIN .

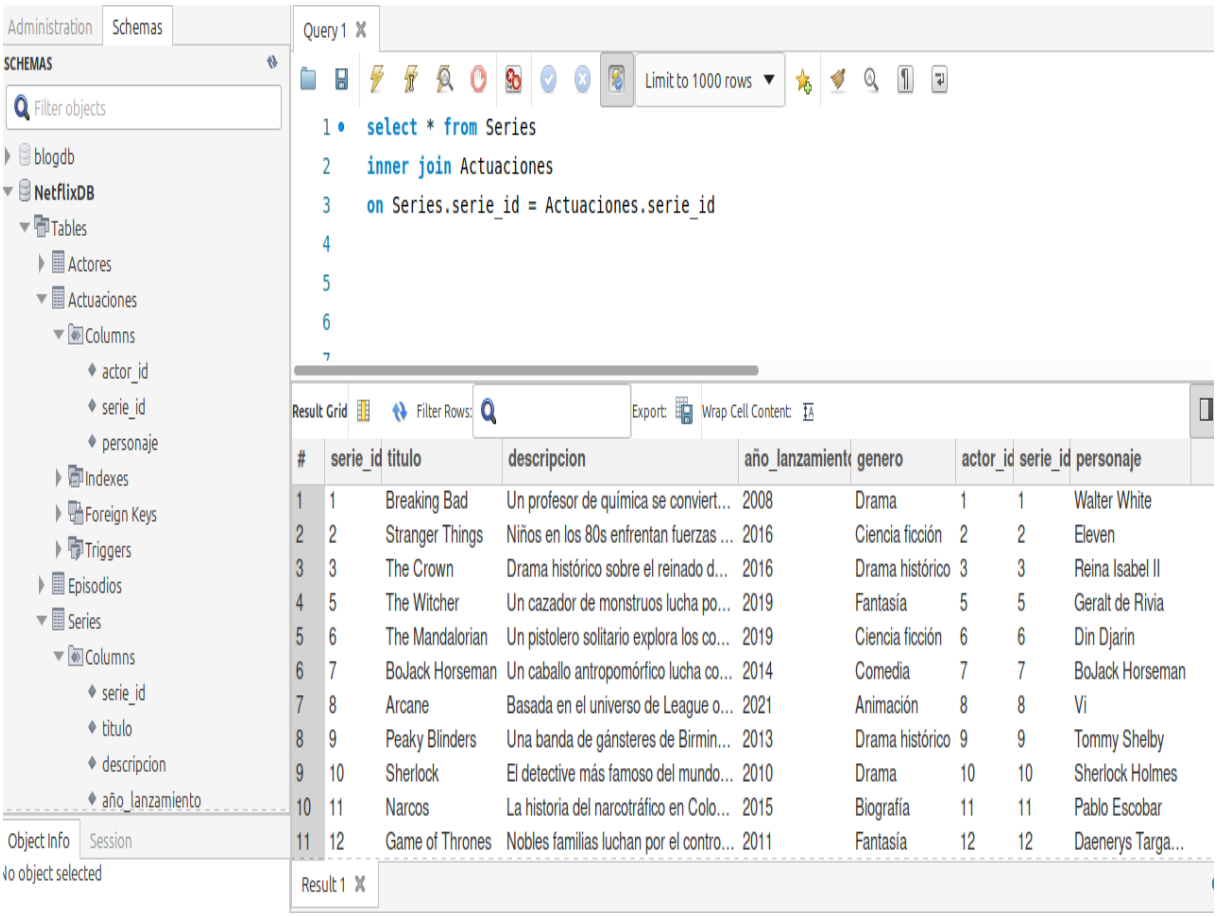


Figure 2: In the code we select all the columns in the Series table (*), and we make **inner join** to the table Actuaciones. The we type the key column with **on** and the columns of each tables

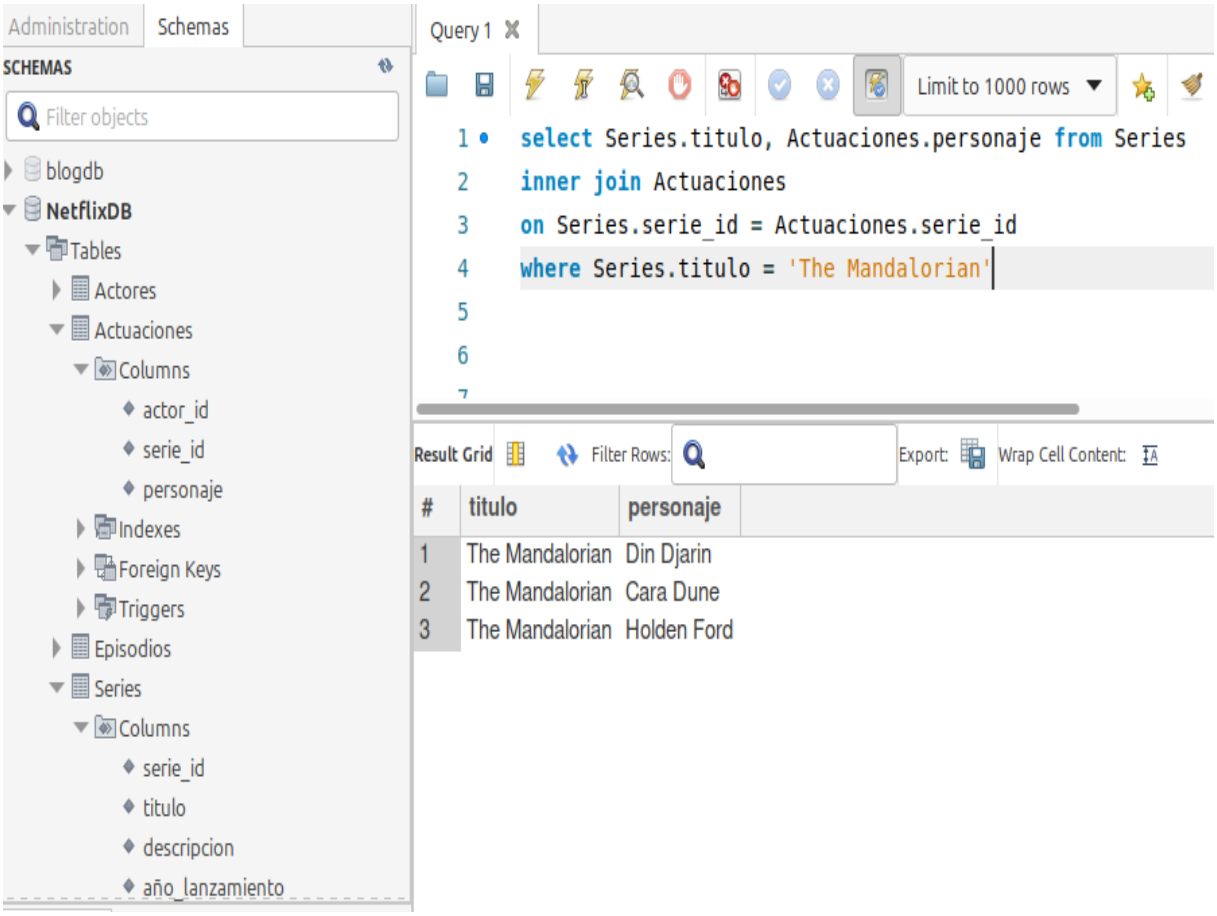


Figure 3: Example of an inner join

We can also select certain columns or rows. Recall previous clauses such as **where**. We can add alias to the tables, but I think it is not so profitable. For example a code where we want to select a column of the table Actuaciones and a column of the table Series, and then just fetch the rows corresponding to the series Mandalorian would be like that. We run the code in the Figure 3

select Series.titulo, Actuaciones.personaje **from** Series

inner join Actuaciones **on** Series.serie_id = Actuaciones. serie_id

where Series.titulo = 'The Mandalorian'

3.1.2 Left join Right join

This clause enables to grab the columns of one table and just the required columns of the other table. Example in the Figure 4

3.2 Union and Union all

This clause enables to merge two or more querys. It is important to know that the querys to merge must have the same number of columns. The main difference between **union** and **union all** is that the first one fecth just unique values, essentially like when we use **distinct** and the second one permit duplicate results.

select * **from** Series

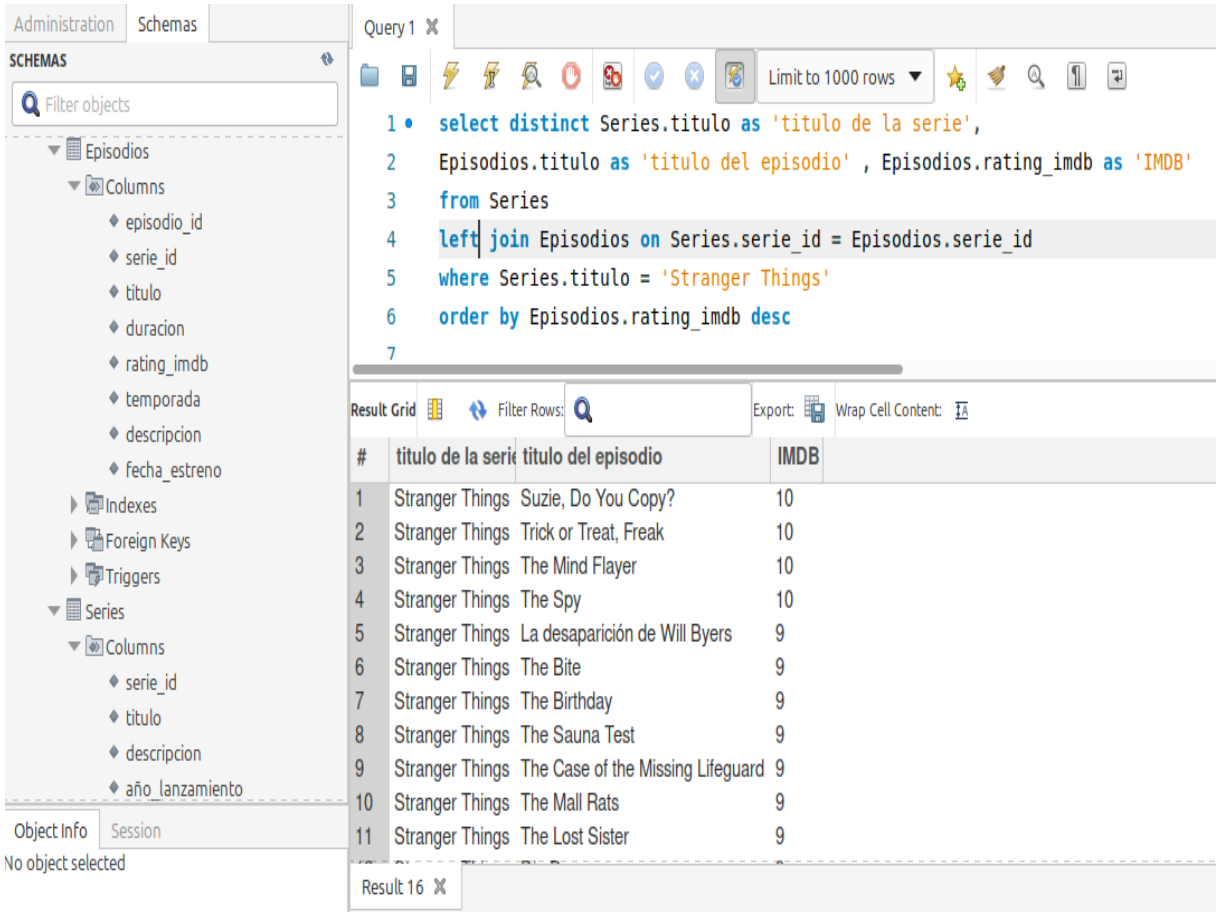


Figure 4: Example of an left join

where genero= 'Ciencia Ficcion'

union

select * **from** Series

where genero= 'Drama'

3.3 Example with NETFLIXDB

3.3.1 Query 1

Which movie gender are the most common?

select Series.genero, count(Series.serie_id) **as** cuenta

group by Series.genero

3.3.2 Query 2

Which are the series with a higher rate in IMDB and how many episodes they have?.

select Series.titulo, count(episodio_id) **as** cuenta, **avg**(rating_imdb) **as** rate **from** Series

left join Episodios **on** Series.serie_id = Episodios.serie_id

group by Series.titulo

order by rate

3.3.3 Query 3

How long is the series Stranger Things?

select Series.titulo, sum(duracion) **as** duracion **from** Series

left join Episodios **on** Series.serie_id = Episodios.serie_id

where Series.titulo = 'Stranger Things'

group by Series.titulo

3.3.4 Full join

3.3.5 Cross join

4 Intermediate Querys

4.1 Subquery

This is query inside a previous query. For example, If we want to grab all the episodes of a certain series we could run two different codes:
1st query:
select * **from** Episodios

where serie_id = a number we dont know

2nd query:

select serie_id **from** Series

where titulo = 'Arcane'

Then we will have our answer, bit with a subquery we can do it faster:

select * **from** Episodios

where serie_id =(**select** serie_id **from** Series **where** titulo = 'Arcane')

We can use **join** with a sub query too. For example, if we want to know which are the series with a higher rating in imdb we can do as follows:

select Series.titulo, Episodios.media **from** Series

join (**select** serie_id, **avg** (rating_imdb) **as** media **from** Episodios

group by serie_id

having media > 8)

Episodios **on** Series.serie_id = Episodios.serie_id

order by media **desc**

4.2 Conditional function IF

Essentially This clause permits to run a part of our code or another according to certain conditions. This clause goes always inside the **select** command, and it goes as follows:

select titulo, rating_imdb, **if** (our_condition, result for yes, result for no) **as** name **from** Episodios

For example:

select titulo, rating_imdb, **if** (rating_imdb > 8, 'Alto', 'Bajo') **as** 'categoria del episodio' **from** Episodios

4.3 Conditional function Case-Else

This will permit to add more than one condition to our results. It is like an upgraded **if** clause. For example if we wanto clasify our series of netflixDB by year we could use this clause.
This clause position is inside the **select** clause.

select titulo, año_lanzamiento,

case

when año_lanzamiento >= 2016 **then** 'mierda woke'

when año_lanzamiento **between** 1990 **and** 2015 **then** 'good'

else 'antigua'

end as 'categoria'

from Series

4.4 Transformation function Cast

This function enables to change the data type of our columns. If we want o to see which data type has our table, just need to open a new query and type **describe** Table name. For example, we can transform a number (integer) into a date(date-time data type)

4.5 Date-Time

4.5.1 year(), month(), day()

select fecha_estreno, **year**(fecha_estreno) **from** Episodios

4.5.2 Dateadd()

This adds a period of time to our date, and returns the final date. We can also deduct time by add a − before the number

select fecha_estreno, **date_add**(fecha_estreno, **interval** 30 **day**

from Episodios

4.5.3 Datediff

It reckons the difference between two dates. For example, using curdate (current day)

select fecha_estreno, **datediff**(fecha_estreno, **curdate()**)

from Episodios

4.6 Text chain management

This enables to fetch or manage certain information from large chains of text, for example a book.

4.6.1 Upper, lower

This will transform every letter into capital letter or reverse

select upper as titulo_mayusculas **from** Series

4.6.2 Concat

This enables to put words together

select concat (titulo, '(',año_lanzamiento,')') as 'Titulo y año' **from**

4.6.3 Substring, length

Enables to extract certain letters from a text, or the lenght.

select substring(titulo, 1,5) **as** extracto **from** Episodios

select titulo **length**(titulo) **as** longitud **from** Series

4.6.4 Left and right

This gets the letters we need starting from the left or rigth.

```
select titulo

left(titulo, 3) as inicio_titulo

right(titulo, 3) as fin_titulo

from Series
```

4.7 Math functions

4.7.1 Round, ceiling, floor

Allows to round a number. Ceiling rounds upwards and floor downwards

```
select titulo, duracion/60.0 as horas, round(duracion/60.0, 0) as horas_redondeo

from Episodios
```

4.8 Example with NETFLIXDB day 5

4.8.1 Query 1

We need to identify the 3 most popular movie labels based on the amount of series. Then for each label show the title, year of release and avg rating.

```
select Series.genero, Series.titulo, Series.año_lanzamiento, avg(Episodios.rating_imdb) as media
from Series

join Episodios on Series.serie_id = Episodios.serie_id

where genero in(select genero from

(genero, count(serie_id) as total_series) from Series

group by serie_id

limit  3) as top_3)
```

5 Advanced querys

5.1 CTE - Common Table Expression

This kind of query work similarly to the subquerys. Essentially they are temporary querys that are used inside an SQL query. This help us to simplify complex querys. We can create our CTE at the very beginning our our code and right after we need to create our main query. I have 3 examples in the mysql folder, however in the Figure 5

5.2 Row_number () Función ventana

This function allows to carry out calculations over rows which share a common point. One of these functions is **row_number**, that simple provides a number according to a certain clause or condition. For exmaple, if we want to use **order by** this function will provide a number to each row according to the established order. I provide you an example in Figure

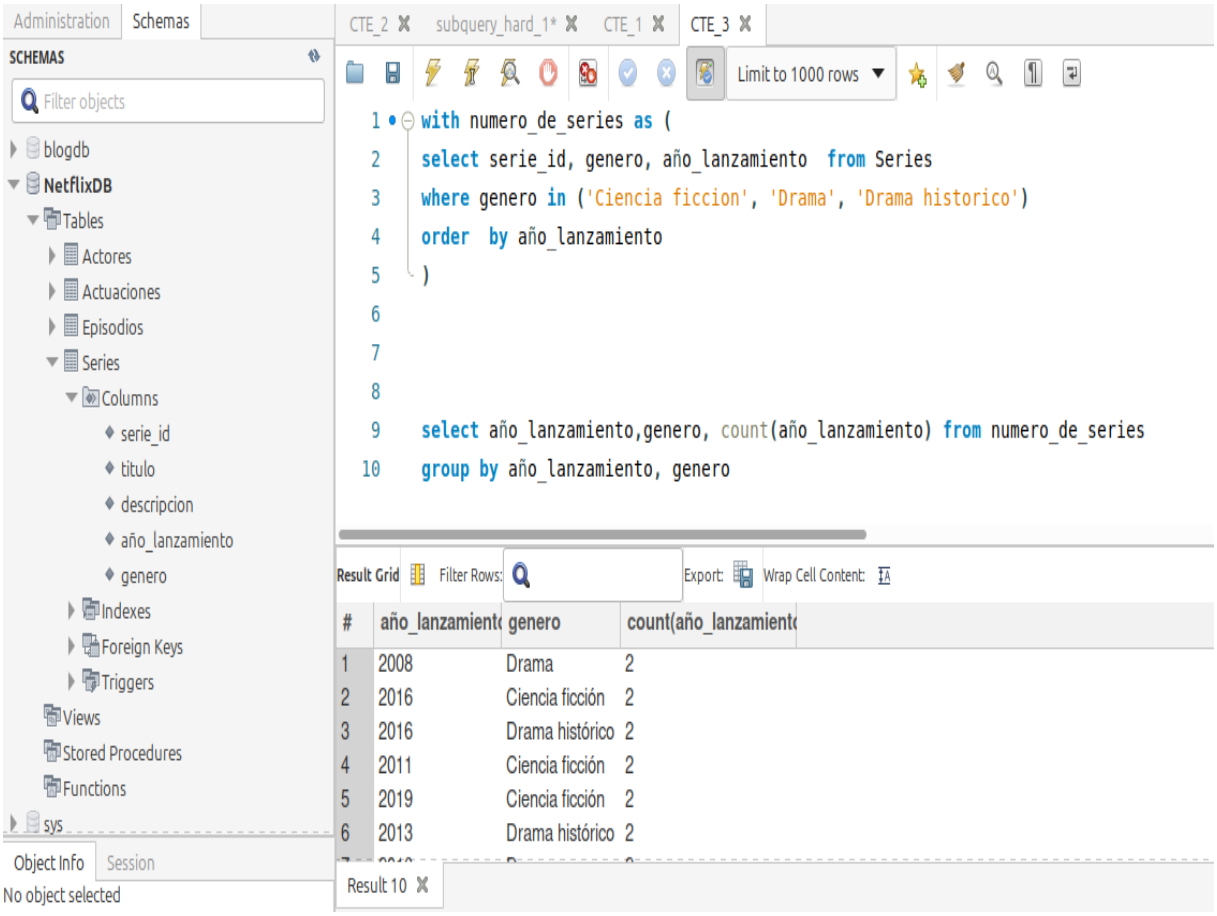


Figure 5: Example of a CTE

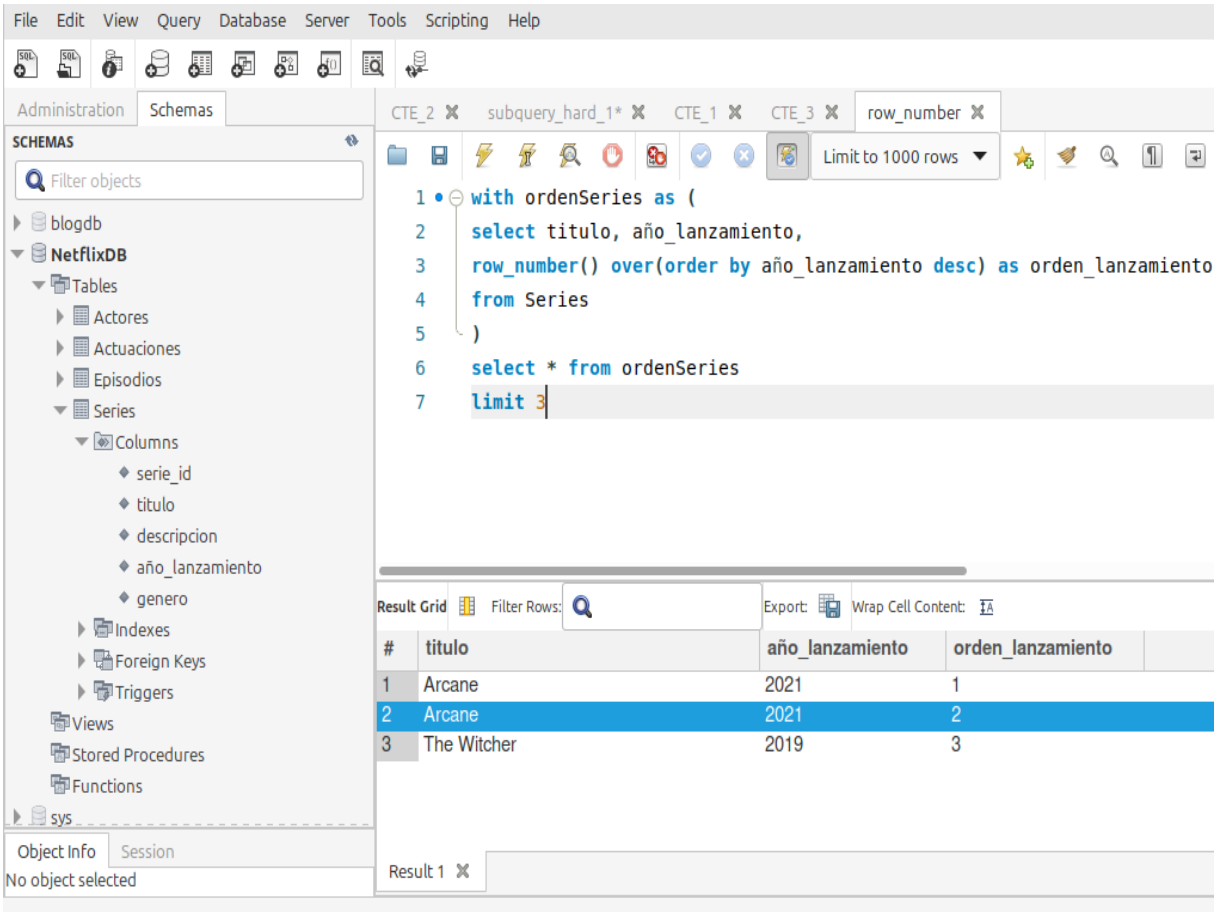


Figure 6: Example of the function rowNumber

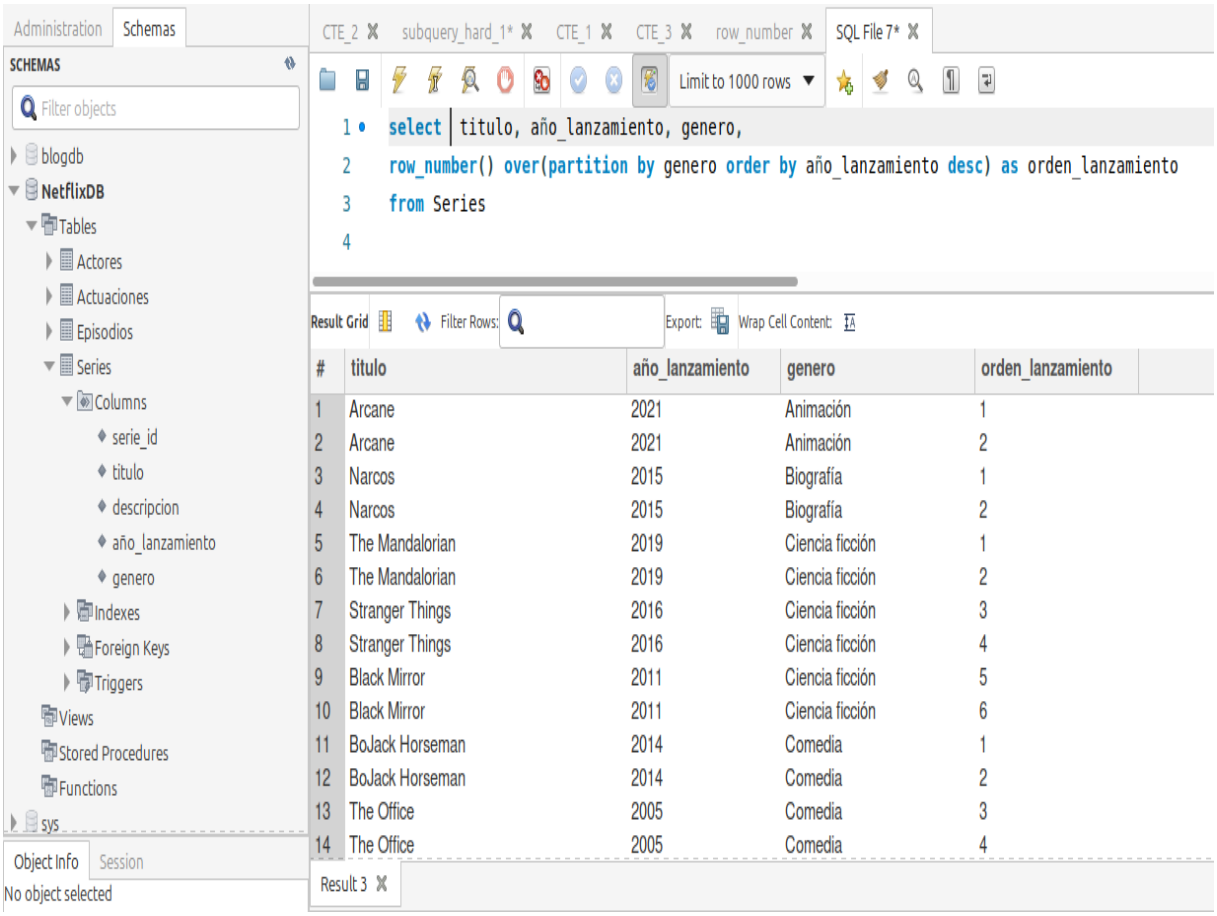


Figure 7: Example of the function partition_by

5.3 Partition by () Función ventana

This function goes inside the **row_number** function and acts similarly to **group by**. You may see it more clearly in the Figure 7

5.4 Rank () Función ventana

It works similarly to **row_number**, but with slightly differences. For example, if we use

select titulo,rating_imdb,

rank () **over**(**order by** rating_imdb) **as** ranking

from Episodios

And two or more episodios have the same value in rating_imdb, then they will be assigned the same rank. Please check Figure 8

5.5 Dense_Rank () Función ventana

It works exactly the same as **rank**() but it leaves no spaces between the ranks. You will see it clearly in th Figure 9

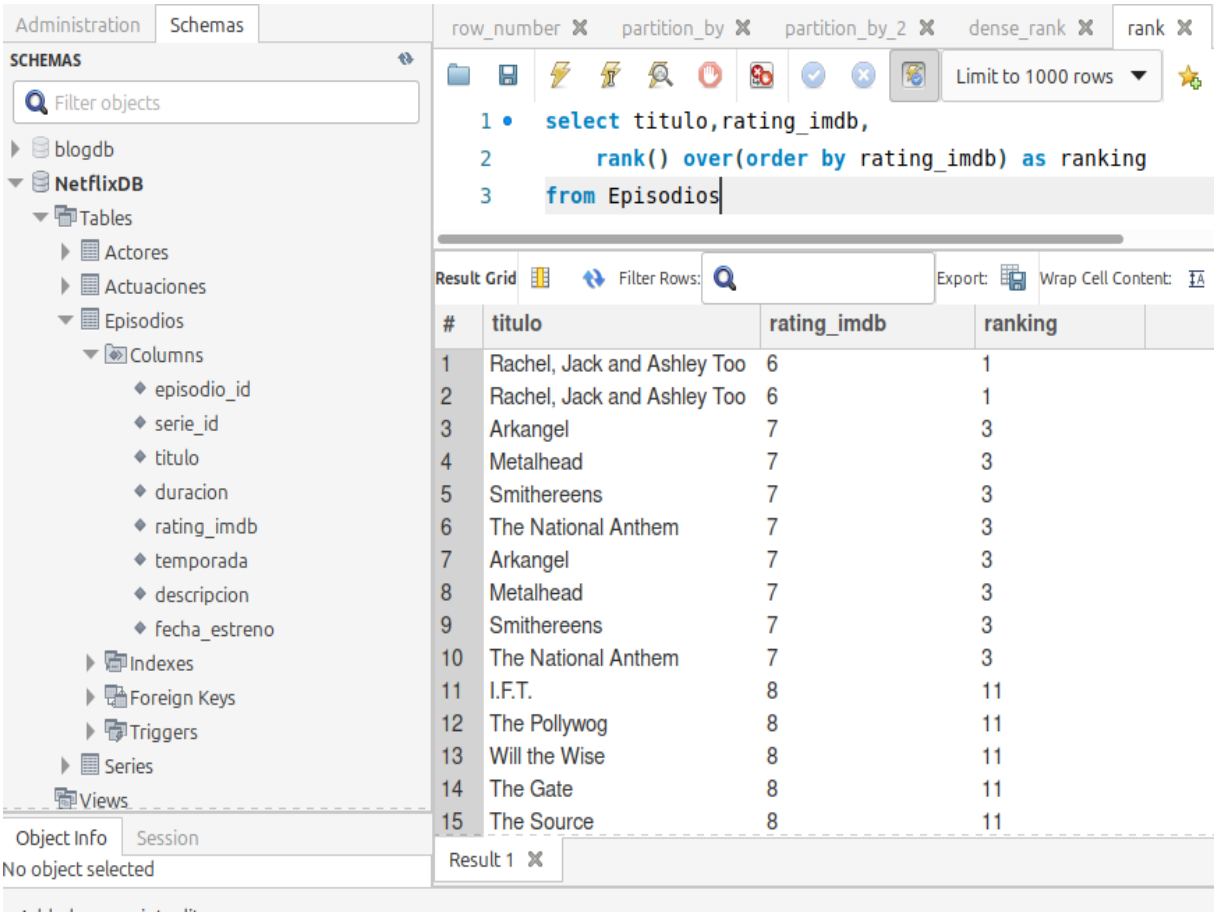


Figure 8: Example of the function rank

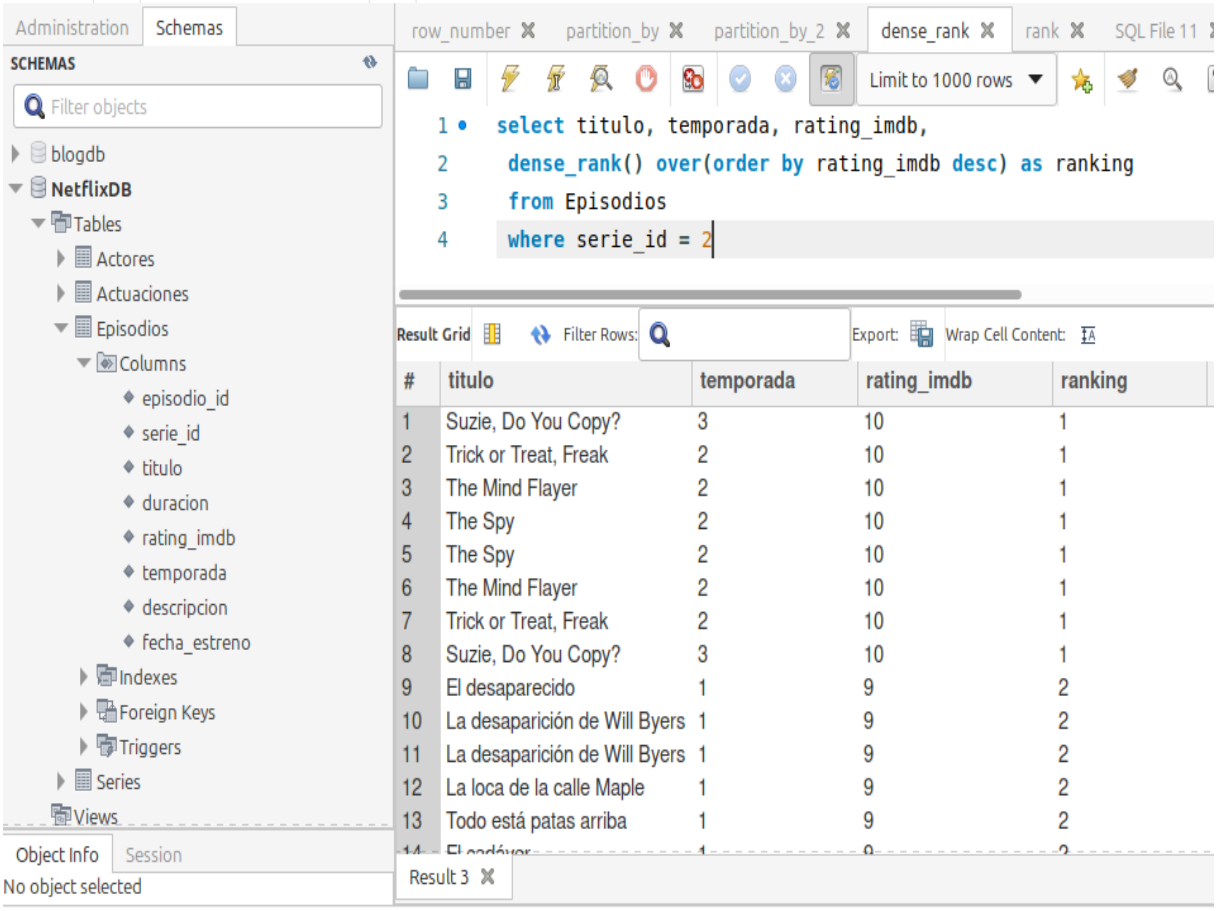


Figure 9: Example of the function denseRank

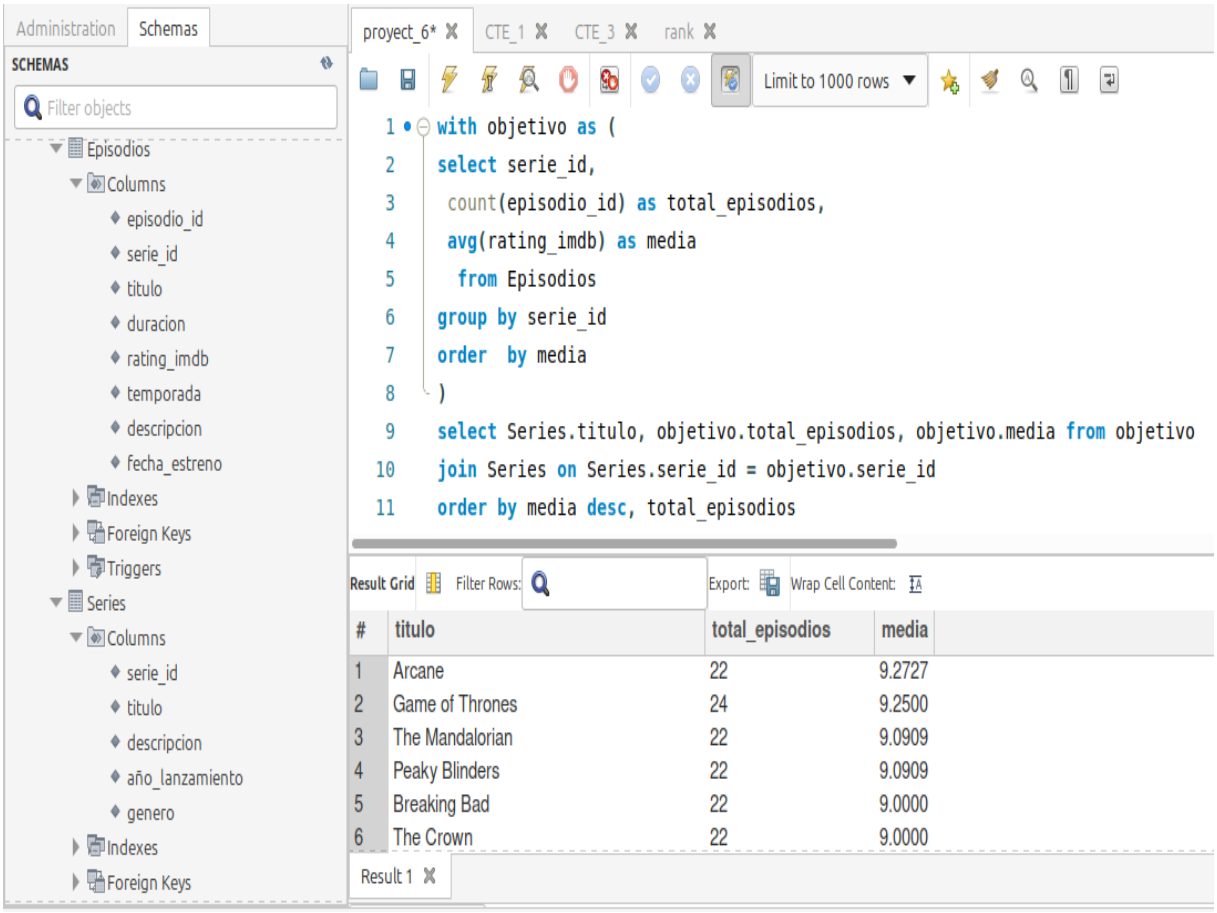


Figure 10: Project of the 6th day

5.6 RegExp Regular Expressions

This function allows us to find a certain combination of letters in a text. For example if I want to fetch all the series that in their description have the word 'woke'. Essentially, we can add this to the clause **where**.

select titulo, descripcion **from** Series

where descripcion **regex** '(?i)woke'

This will return every series that in their description is contained the word 'woke'. The regular Expression is (?i) which will find this word regardless if it is in capital letters, included in a larger word o whatever

5.7 Example with NETFLIXDB day 6

We are required to identify the most popular series in NetflixDB. This can be achived by counting the number of episodies and their imdb rating. The expected result is a table with 3 columns, such as Titulo de la serie, cantidad de episodios, and rating promedio de imdb de la serie. Please check it out in the Figure 10

6 DB Modelling

For modelling a data base we need to know certain concepts.

- Diagramas de entidad relacion - these Diagrams show the DB stricture graphically.
- Foreign keys and Primary keys - These are essentially the columns which permit the relation between the tables. A primary key of a table is a column of unique values in the table, so such values can not appear twice in this table. A foreign key of the table B is the primary key of the table A that works as link between bith tables.
- Cardinalidad - This essentially describe the kind of relation between tables. For example, if the relation is one-one, one-many, many-one, zero-one and more.

There is an additional concept that is important to clarify. Such concept is the Normalization. This essentially avoid data redundancy. So with normalization we simplify our DB. There are 3 NORMAL FORMS.

- 1NF. If a table is in 1NF it must meet the following cirteria:

  Each 'celda' must contain just one value. For example if we have a column called 'name and surname', such column will have two value in it.

  The table must have a PK

  The table can not contain duplicated rows or columns

  each column must have just one value for each row in the table

- 2NF. If a table is in 2NF it must meet the following cirteria:

  It must be in 1NF already.

  It can not have partial dependencies. It means that all the NO KEY attributes must depend on the PK. This only occurs when the PK is a composite PK. For example if the composite PK is composed by 2 columns and our NO KEY attribute depends only on one of the columns, then it has a partial dependency.

- 3NF. If a table is in 3NF it must meet the following cirteria:

  The table must be already in 2NF.

  Must not have transitive dependencies. It means that a NON KEY attribute can not depend on other NON KEY attribute. It means that our NON KEY attribute must depend directly on the PK and not through other NON KEY attributes.

7 Create, modify and manage DB and tables

7.1 DDL - Data Definition Language

TO create a DB we can do it with the graphic tools we have done during the course, or by code. By code we must type **create database if not exists** empresaDB.

Then we tell sql to use such DB by typing **use** empresaDB. Now we start to create our table. Essentially, we will add the columns we need and the features of such columns. The whole code for the creation of a database would be as follows:

```
create database if not exists  empresaDB;

use empresaDB;

create table if not exists departamentos (
depto_id int auto_increment primary key,
nombre varchar(255) not null ,
ubicacion varchar(255) notnull );

create table if not exists empleados (
empleado_id int auto_increment primary key,
nombre varchar(255) not null ,
email varchar(255) notnull,
depto_id int,
foreign key (depto_id),
references departamentos (depto_id),
ON DELETE SET NULL );
```

7.1.1 Other commands

```
alter table departamentos add column email_jefe varchar(255)

drop table if exists asignacionesDeProyectos
```

7.2 DML - Data Manipulation Language

7.2.1 insert into

This commands enables to insert values into our tables. The very first command we need to know is **insert into**:

```
insert into departamentos (nombre, ubicacion) values
('recursos humanos', 'edificio b'),
('marketing', 'edificio central')
```

7.2.2 update

```
update departamentos
set ubicacion='edificio central' where nombre='marketing'
```

7.2.3 delete from

```
delete from departamentos where nombre = 'marketing'
```

7.2.4 User creation and permits

We need to create users and give them certain permits in order to be able to manage our DB. This can be done through the 'User and privileges' in the Adiminstration part in mysql-workbench

7.3 DQL - Data Query Language

7.4 DCL - Data Control Language

7.5 TCL - Transaction Data Language

8 Store Procedures and Transactions

8.1 Store Procedures

These are, essentially, blocks of code that we can use when we need them. This can be quite useful to insert in or update our DB. To create a **procedure** first of all we have to delimiter it somehow and for this we use **delimiter //**. Then if we want to insert data we have to select the table we want, and then the columns of such table. As we can observe, the table empleados has 5 cols, but one of them is the PK and it is Auto-increment, so we just need to fill up the others. At the very beginning we tell sql to use (**//** as a delimiter instead of (**;**, then at the end of the code we tell sql to use the (**;** as a delimiter of procedures

delimiter //

create procedure agregar_empleado(**in** _nombre **varchar(255)**, **in** _apellido **varchar(255)**, **in** _email **varchar(255)**, **in** _depto_id **int**)

begin

insert into empleados(nombre, apellidos, email, depto_id) **values** (_nombre, _apellido, _email, _depto_id);

end//

delimiter ;

Now if we want to use it, we just need to open a new query and call it.

(**call** agregarEmpleado('elena','torres','elena.torres@gmail.com','3')

8.2 Transactions

A transaction is a one unit of work but it can consists in many. For example, it can be composed by modifications of the data in our DB. If a transaction is successful then all the provided changes are confirmed. However, if it is unsuccessful then all the provided changes are reverted. The main commands of a transaction are:

- **begin transaction**
- **commit**
- **rollback**

9 Sakila Proyect Explained

9.1 Table Actor

```
44 • CREATE TABLE actor (
45     actor_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
46     first_name VARCHAR(45) NOT NULL,
47     last_name VARCHAR(45) NOT NULL,
48     last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
49     PRIMARY KEY  (actor_id),
50     KEY idx_actor_last_name (last_name)
51 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
52
53 --
```

Figure 11: actor table. It is an easy table, just a PK, and an index. Additionally it includes **engine** = innoDB which enables transactions, and other things. Also it includes **default charset** which allows many languages and emojis

9.2 Table Address

```
57 • CREATE TABLE address (
58     address_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
59     address VARCHAR(50) NOT NULL,
60     address2 VARCHAR(50) DEFAULT NULL,
61     district VARCHAR(20) NOT NULL,
62     city_id SMALLINT UNSIGNED NOT NULL,
63     postal_code VARCHAR(10) DEFAULT NULL,
64     phone VARCHAR(20) NOT NULL,
65     -- Add GEOMETRY column for MySQL 5.7.5 and higher
66     -- Also include SRID attribute for MySQL 8.0.3 and higher
67     /*!50705 location GEOMETRY */ /*!80003 SRID 0 */ /*!50705 NOT NULL,*/
68     last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
69     PRIMARY KEY  (address_id),
70     KEY idx_fk_city_id (city_id),
71     /*!50705 SPATIAL KEY `idx_location` (location),*/
72     CONSTRAINT `fk_address_city` FOREIGN KEY (city_id) REFERENCES city (city_id) ON DELETE RESTRICT ON UPDATE CASCADE
73 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Figure 12: This table is a regular one but it includes geometry and other commands

Let's check out line by line from it gets complicated

- /*!50705 location GEOMETRY */ /*!80003 SRID 0 */ /*!50705 NOT NULL,*/

  This line holds 3 conditional comments. While the regula comment in sql is /*! something */ the conditional comment works only if the condition is met, in this case it will creates a new column called location which type is geometry, just if the sql version i 5.07.07 or higher.

  The line /*!80003 SRID 0 */ will establish the GRID SYSTEM which is SRID (Spatial Reference System Identifier), and 0 is the default value for SRID.

  The last line /*!50705 NOT NULL,*/ will make the column location not null

- last_update **TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP** These are the specification for the column :

  TIMESTAMP NOT NULL- data type is timestamp and not null

  DEFAULT CURRENT_TIMESTAMP - this inserts automatically the the current time when creates a new register.

ON UPDATE CURRENT_TIMESTAMP - it updates automatically

- KEY idx_fk_city_id (city_id) - this creates an index in the column city_id to have a better performance with the table city.

- /*!50705 SPATIAL KEY 'idx_location' (location),*/ this creates an esapcial index in the location column, that enhances the performance.

- **constraint** 'fk_address_city' **foreign key** city_id **references** city(city_id)**ON DELETE RESTRICT ON UPDATE CASCADE**

  ON DELETE RESTRICT denys to eliminate a city if it has now address.

  ON UPDATE CASCADE - if city_id in the table city changes, then it automatically changes in this table

9.3 Tables category and city

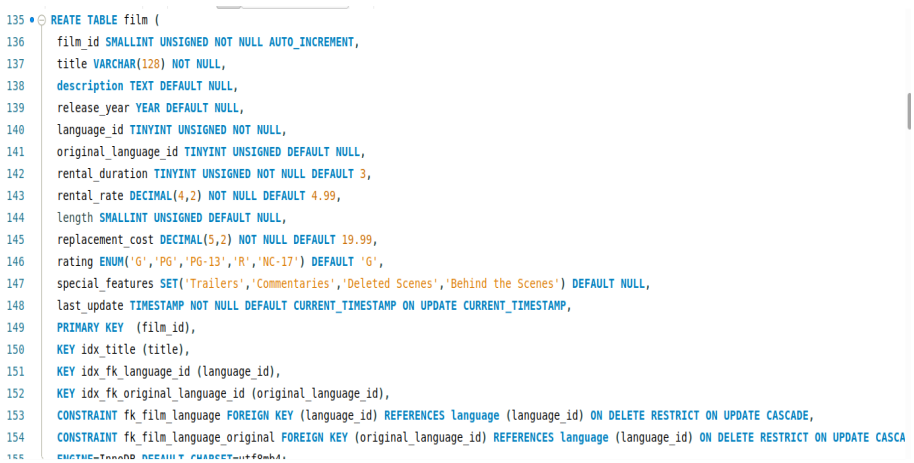
```
79  CREATE TABLE category (
80    category_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
81    name VARCHAR(25) NOT NULL,
82    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
83    PRIMARY KEY  (category_id)
84  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
85  --
86  -- Table structure for table `city`
87  --
88  CREATE TABLE city (
89    city_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
90    city VARCHAR(50) NOT NULL,
91    country_id SMALLINT UNSIGNED NOT NULL,
92    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
93    PRIMARY KEY  (city_id),
94    KEY idx_fk_country_id (country_id),
95    CONSTRAINT `fk_city_country` FOREIGN KEY (country_id) REFERENCES country (country_id) ON DELETE RESTRICT ON UPDATE CASCADE
96  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
97
```

Figure 13: These tables are regular ones, nothing especial. The table country which is not included in the figure is quite similar, so we didn´t included here

9.4 Customer table

```
113 CREATE TABLE customer (
114   customer_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
115   store_id TINYINT UNSIGNED NOT NULL,
116   first_name VARCHAR(45) NOT NULL,
117   last_name VARCHAR(45) NOT NULL,
118   email VARCHAR(50) DEFAULT NULL,
119   address_id SMALLINT UNSIGNED NOT NULL,
120   active BOOLEAN NOT NULL DEFAULT TRUE,
121   create_date DATETIME NOT NULL,
122   last_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
123   PRIMARY KEY  (customer_id),
124   KEY idx_fk_store_id (store_id),
125   KEY idx_fk_address_id (address_id),
126   KEY idx_last_name (last_name),
127   CONSTRAINT fk_customer_address FOREIGN KEY (address_id) REFERENCES address (address_id) ON DELETE RESTRICT ON UPDATE CASCADE,
128   CONSTRAINT fk_customer_store FOREIGN KEY (store_id) REFERENCES store (store_id) ON DELETE RESTRICT ON UPDATE CASCADE
129 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Figure 14: This table is a regular one



```
REATE TABLE film (
  film_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  title VARCHAR(128) NOT NULL,
  description TEXT DEFAULT NULL,
  release_year YEAR DEFAULT NULL,
  language_id TINYINT UNSIGNED NOT NULL,
  original_language_id TINYINT UNSIGNED DEFAULT NULL,
  rental_duration TINYINT UNSIGNED NOT NULL DEFAULT 3,
  rental_rate DECIMAL(4,2) NOT NULL DEFAULT 4.99,
  length SMALLINT UNSIGNED DEFAULT NULL,
  replacement_cost DECIMAL(5,2) NOT NULL DEFAULT 19.99,
  rating ENUM('G','PG','PG-13','R','NC-17') DEFAULT 'G',
  special_features SET('Trailers','Commentaries','Deleted Scenes','Behind the Scenes') DEFAULT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY  (film_id),
  KEY idx_title (title),
  KEY idx_fk_language_id (language_id),
  KEY idx_fk_original_language_id (original_language_id),
  CONSTRAINT fk_film_language FOREIGN KEY (language_id) REFERENCES language (language_id) ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT fk_film_language_original FOREIGN KEY (original_language_id) REFERENCES language (language_id) ON DELETE RESTRICT ON UPDATE CASCA
```

Figure 15: This table is a regular one

9.5 Film table

This table has some columns that are new for us, so let us describe them. Figure 15:

- rental_rate **decimal**(4,2)**not null default** 4.99 - this is a column called rental_rate with data type decimal and contains numbers of maximum 4 digits, and 2 of them must be after the , so the maximum value would be 99.99

- special_features **set** ('Trailers','Commentaries','Deleted Scenes','Behind the Scenes') **default null** - This column can have 4 different types of values inside. A single case can have none of them, one or more. The command **enum** works similarly to **set**, but the first just allow one value per column

10 SET and full_text

Due to the command **full text** doesnt run in sql versions prior 50610 we need to set a new sql store engine, just in case the one who run the code has a previous version. We will us a conditional comment as we did before. In the Figure 16 , we check how it is done.

```
-- InnoDB added FULLTEXT support in 5.6.10. If you use an
-- earlier version, then consider upgrading (recommended) or
-- changing InnoDB to MyISAM as the film_text engine
--

-- Use InnoDB for film_text as of 5.6.10, MyISAM prior to 5.6.10.
SET @old_default_storage_engine = @@default_storage_engine;
SET @@default_storage_engine = 'MyISAM';
/*!50610 SET @@default_storage_engine = 'InnoDB'*/;

CREATE TABLE film_text (
  film_id SMALLINT UNSIGNED NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  PRIMARY KEY  (film_id),
  FULLTEXT KEY idx_title_description (title,description)
) DEFAULT CHARSET=utf8mb4;

SET @@default_storage_engine = @old_default_storage_engine;
```

Figure 16: Here we **set** the new engine according to the condition. Then we use **full text** to create and index of the whole text in the columns title and description. Then we change the engine again back to the default value

.

11 Triggers

We create some triggers

11.1 Good luck!

We hope you find Overleaf useful, and do take a look at our help library for more tutorials and user guides! Please also let us know if you have any feedback using the Contact Us link at the bottom of the Overleaf menu — or use the contact form at `https://www.overleaf.com/contact`.

References