SQL BASICS

INDEX

[1 - BASIC SQL 3](#_Toc95236358)

[1.1 Select, From and Order By 3](#_Toc95236359)

[1.2 Filter with Where, And and Or 3](#_Toc95236360)

[1.3 Filtering with IN and NOT IN, LIKE and NOT LIKE 3](#_Toc95236361)

[1. 4 Filtering with GREATER THAN, LESS THAN, NOT EQUAL, and BETWEEN 3](#_Toc95236362)

[1.5 SELECT DISTINCT, COUNT and GROUP BY 3](#_Toc95236363)

[1.6 Filtering aggregates with HAVING 3](#_Toc95236364)

[1.7 Colum aliases and combining two columns with CONCAT 3](#_Toc95236365)

[1.8 Returning top rows with LIMIT 3](#_Toc95236366)

[1.9 SUM, AVG, MIN, MAX, TRUNC and ROUND 3](#_Toc95236367)

[Customer Analysis task 4](#_Toc95236368)

[Precision and Scale 7](#_Toc95236369)

[2 JOINS SQL 7](#_Toc95236370)

[2.1 DataBase Constraints and Foreign Keys 7](#_Toc95236371)

[2.2 Left Join 7](#_Toc95236372)

[2.3 Inner Join 9](#_Toc95236373)

[Task – Find the best way to write a query that returns all actors and actresses who starred in the film “Snowman Rollercoaster” 10](#_Toc95236374)

[Inner join vs Join 10](#_Toc95236375)

[2.4 Cross join 10](#_Toc95236376)

[2.5 Anti-join 11](#_Toc95236377)

[Qualifying in SQL 11](#_Toc95236378)

[Task: Find all customers who never paid for their film rentals 12](#_Toc95236379)

[2.6 Full Outer Join 12](#_Toc95236380)

[3 SUBQUERIES 14](#_Toc95236381)

[3.1 Subqueries 14](#_Toc95236382)

[3.2 More Subqueries and STRING\_AGG 14](#_Toc95236383)

[3.3 More Subqueries and derived tables 14](#_Toc95236384)

[3.4 EXISTS AND NOT EXISTS 14](#_Toc95236385)

[4 COMBINING RESULTS 14](#_Toc95236386)

[4.1 UNION and UNION ALL 14](#_Toc95236387)

[4.2 INTERSECT and EXCEPT 15](#_Toc95236388)

[5 DATA INSPECTION AND CONVERSION 15](#_Toc95236389)

[5.1 Overview of data types in PostgreSQL 15](#_Toc95236390)

[5.2 Date and time functions 15](#_Toc95236391)

[5.3 Analyzing data with date and time functions 15](#_Toc95236392)

[5.4 String manipulation 15](#_Toc95236393)

[5.5 CASE statements 15](#_Toc95236394)

[5.6 COALESCE 15](#_Toc95236395)

[6 DATA MANIPULATION 15](#_Toc95236396)

[6.1 CREATE TABLE – INSERT – UPDATE – DELETE 15](#_Toc95236397)

[7 ANALYTIV FUNCTIONS 15](#_Toc95236398)

[7.1 OVER and PARTITION BY 15](#_Toc95236399)

[7.2 RANK, DENSE\_RANK, and ROW\_NUMBER 16](#_Toc95236400)

[7.3 LEAD and LAG 16](#_Toc95236401)

[8 VARIABLES AND TEMPORARY SETS 16](#_Toc95236402)

[8.1 Temporary tables 16](#_Toc95236403)

[8.2 Common table expressions 16](#_Toc95236404)

[8.3 Recursive operations using CTEs 16](#_Toc95236405)

[8.4 Pivotinh using the FILTER clause 16](#_Toc95236406)

[9 SAMPLE DATABASE INSTALLATION 16](#_Toc95236407)

[9.1 Installing the AdventureWorks sample database 16](#_Toc95236408)

# 1 - BASIC SQL

## 1.1 Select, From and Order By

<https://youtu.be/3pAAHAgvkEE>

## 1.2 Filter with Where, And and Or

<https://youtu.be/7YgZZPdA7AM>

More info: <https://www.postgresqltutorial.com/postgresql-where/>

## 1.3 Filtering with IN and NOT IN, LIKE and NOT LIKE

<https://youtu.be/l7tC_DlWIf8>

## 1. 4 Filtering with GREATER THAN, LESS THAN, NOT EQUAL, and BETWEEN

https://youtu.be/VhEC\_bsU0Zg

## 1.5 SELECT DISTINCT, COUNT and GROUP BY

<https://youtu.be/zUzcA5Pbeis>

## 1.6 Filtering aggregates with HAVING

<https://youtu.be/97TjQFpjKg4>

## 1.7 Colum aliases and combining two columns with CONCAT

<https://youtu.be/__qy-WwWP9I>

## 1.8 Returning top rows with LIMIT

<https://youtu.be/UxIHgQaFRiA>

## 1.9 SUM, AVG, MIN, MAX, TRUNC and ROUND

<https://youtu.be/sfTfbyKb56A>

Here we will see some functions to aggregate data.

* SUM (allows us to sum a given value across rows)
* AVG (finds the average of a given value)
* MIN (can find the smallest value of a given value across rows whether that’s a number, a date or a letter.
* MAX (can find the highest value od a given value across rows)

### Customer Analysis task

We want to know, not only how many films they have rented from our company, but we also want to know how much money each person has spent. What is the average length of the films each person has rented? When was each person's earliest rental date, and when was each person's most recent rental date?

We will use the functions sum, avg, min and max, in order to solve the problem

We start by preparing the query:

Texto

Descripción generada automáticamente

We first use the **SELECT DISTINCT** in order to get just non-repeated values. Then we use the **INNER JOIN** in order to add more information about the inventory, the rentals and the customer. Finally, we use a **LEFT JOIN** for the payment table. In this case, we use the **LEFT JOIN** because we want to find out how many each customer has rented, and if we had used an **INNER JOIN**, we would have only got how many movies each customer rented and paid for (reminder: **left join** returns us non-matching values as null, while **inner join** returns us matching values -review inner join and left join to clarify).

Después de esto, creamos un **COUNT (\*) AS “Count of Rentals”.** In this part we are counting all rows in the the full table (including joins) with the same customer\_id, first\_name and last\_name. Básicamente, estamos uniendo toda la información de los joins en la misma tabla, entonces, con el COUNT (\*) nos suma las rows de la tabla completa (incluyendo rental etc) en las que aparece ese id, esto es posible gracias al **GROUP BY** (We always need a group by clause when we use the count function) que determina las variables por las que se agrupa el count. Por ejemplo, si hubiéramos grouped by fisrt\_name, nos sumaría las rentals de todos los que se llamaran igual, por eso le ponemos las tres variables, para que agrupe por su conjunto.

Texto

Descripción generada automáticamente

Captura de pantalla de un videojuego

Descripción generada automáticamenteThis query will return us a table showing which client has rented more times.

#### SUM

Now we are going to use **SUM** function in order to know how much money each customer has spent at our business. We will sum the amount column from the payment table (payment.ammount)

Texto

Descripción generada automáticamente

We only need to use the function in the **SELECT** part and give a name to the column.

#### AVG

We now need to find the average length of all the films each customer has rented. By adding it in the Select clause too.

#### MIN

We add another line to get the earliest rental date

#### MAX

And finally another line to get the latest rental date

Texto

Descripción generada automáticamente

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

### Precision and Scale

Precision – Refers to the number of digits contained in a number, regardless of decimal places.

Scale – Refers to the number of digits that occur after the decimal point.

# 2 JOINS SQL

## 2.1 DataBase Constraints and Foreign Keys

https://youtu.be/fnSWaW2R-CU

## 2.2 Left Join

https://www.postgresqltutorial.com/postgresql-left-join/

We use Left Join in order to Join different tables. To do so, we will need to join tables relating equal values from each different column. The difference between **LEFT JOIN** and **INNER JOIN** is that **INNER JOINS** will only return values that match wihle  **LEFT JOIN** will return us those values that match and add “null” when there is not information

Texto

Descripción generada automáticamente

If we add a conditional in the “FROM” part, for example:

Texto

Descripción generada automáticamente

Then, we will get null values for the rest of the rows. En resumen, esto nos devuelve toda la tabla, pero como le hacemos un join solo con la peli Snowman Rollercoaster, la app no encuentra más coincidencias en la tabla y solo nos mostrará la coincidencia y rellanará el resto con null:

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

However, if we add the conditional in the “WHERE” part, we will receive just the row of “snowman rollercoaster” since we are filtering after the data extraction.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

So, if we add a conditional in the FROM part, the conditional will apply to the extraction, and if we do it in the WHERE part, it will be applied after the extraction as a filter

More info: https://youtu.be/S1bEFqvbNnw

## 2.3 Inner Join

https://www.postgresqltutorial.com/postgresql-inner-join/

INNER JOIN only returns us matching results, if there is a null value, we won’t get it in the output

Texto

Descripción generada automáticamente

### Task – Find the best way to write a query that returns all actors and actresses who starred in the film “Snowman Rollercoaster”

Texto

Descripción generada automáticamente

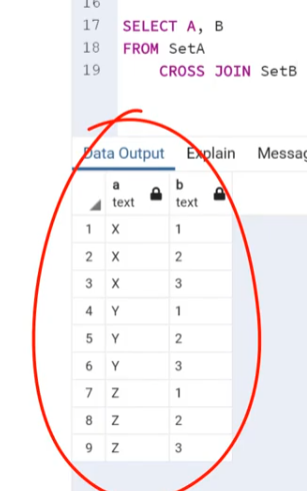
### Inner join vs Join

Although we need to add “LEFT” or “RIGHT” when doing a LEFT JOIN or a RIGHT JOIN, There is no need to add INNER when doing an INNER JOIN query, so we can only write JOIN

More info: <https://youtu.be/ccPxV1QKQeI>

## Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto Descripción generada automáticamente2.4 Cross join

<https://youtu.be/3rSP9vSMKtY>

**CROSS JOINT is used when there is no data to match on between two different sets. In contrast, it takes two sets and generates a Cartesian product.**

**So, if we had two datasets, one with {X,Y,Z} and the other with {1,2,3}, we would get the following output if we used the CROSS JOIN** query:

## ****2.5 Anti-join****

<https://youtu.be/AnnDqx3GvPs>

### Qualifying in SQL

First of all we will know what is **Qualifying** in SQL

Captura de pantalla de un celular

Descripción generada automáticamente

When we **SELECT** customer.first\_name, we a are qualifying since we are selecting an specific column from an specific table. If we just **SELECT** first\_name, then pgAdmin could return us a column same named from another table. Is a way to specify the **SELECT** command. This is commonly used, when joining many different tables, if we do not specify the table from where the column comes, we will get an error.

Anti joins are **a type of filtering join**, since they return the contents of the first table, but with their rows filtered depending upon the match conditions.

### Task: Find all customers who never paid for their film rentals

Texto

Descripción generada automáticamente

The purpose of an anti-join in SQL is to locate records in one table that do not exist in another table. In other words, it is a method to find in this case, rental records that do not have their corresponding payment records.

Anti-joins don’t have their own syntax like other joins. The trick to anti-joins is writing that is null condition in the **WHERE** clause so you only return those records where the data does not exist.

## 2.6 Full Outer Join

<https://youtu.be/CnofJgIHV0U>

This is an uncommonly used join, also known as Full Join. It returns all matching and NON-matching records from table A and table B. It is like a combination of left and right join, it will return all records from both tables, as well as any matching records.

To demonstrate this, we will use the same common table expression setup used to demonstrate the cross join.

Texto

Descripción generada automáticamente

Pantalla con letras y números

Descripción generada automáticamente con confianza media

As we can see here, we got both sets we created a and b together in a table. When we don’t have any matching results, we get a null value.

Texto

Descripción generada automáticamenteLet’s see an example where two values match. We will change from SetA the value “Y” to value “3” and the output will be 5 columns matching equal values.

We can see how the output shows us the matching and non-matching values, so we can use this query to check which values we have in common from different tables.

# 3 SUBQUERIES

## 3.1 Subqueries

<https://youtu.be/f_SFMevQAns>

## 3.2 More Subqueries and STRING\_AGG

<https://youtu.be/04bXK5COKs0>

## 3.3 More Subqueries and derived tables

<https://youtu.be/Lt7jOYx5ggg>

## 3.4 EXISTS AND NOT EXISTS

<https://youtu.be/c-wqoAmELGo>

# 4 COMBINING RESULTS

## 4.1 UNION and UNION ALL

<https://youtu.be/QOm_M36C5So>

## 4.2 INTERSECT and EXCEPT

<https://youtu.be/-b0WtsHnRuk>

# 5 DATA INSPECTION AND CONVERSION

## 5.1 Overview of data types in PostgreSQL

<https://youtu.be/zuKH-Vs4tjw>

## 5.2 Date and time functions

<https://youtu.be/1GfMC0FPRYA>

## 5.3 Analyzing data with date and time functions

<https://youtu.be/_nE79_rnvQ8>

## 5.4 String manipulation

Part 1: <https://youtu.be/ngwBfXr82HQ>

Part 2: <https://youtu.be/dd69yNLIZec>

## 5.5 CASE statements

<https://youtu.be/2_Kkp5cogY8>

## 5.6 COALESCE

<https://youtu.be/rFveAjbXTKE>

# 6 DATA MANIPULATION

## 6.1 CREATE TABLE – INSERT – UPDATE – DELETE

<https://youtu.be/t5ZSY4HWr-k>

Create table part 2: <https://youtu.be/mObtb_l5Wys>

# 7 ANALYTIV FUNCTIONS

## 7.1 OVER and PARTITION BY

<https://youtu.be/bDmqeEOzNtM>

## 7.2 RANK, DENSE\_RANK, and ROW\_NUMBER

<https://youtu.be/TY6Rk1VrB3w>

## 7.3 LEAD and LAG

<https://youtu.be/AJ1IBdCRv_4>

# 8 VARIABLES AND TEMPORARY SETS

## 8.1 Temporary tables

<https://youtu.be/tKxZrWO-SoA>

## 8.2 Common table expressions

<https://youtu.be/Cz21U83MMGs>

## 8.3 Recursive operations using CTEs

<https://youtu.be/pvlR_d8HgM8>

## 8.4 Pivotinh using the FILTER clause

<https://youtu.be/8huntSfDswU>

# 9 SAMPLE DATABASE INSTALLATION

## 9.1 Installing the AdventureWorks sample database

<https://youtu.be/IkYKfprbIl8>

LAB Exercise: https://www.youtube.com/watch?v=QyHfn2N\_e-E