Complete SQL Bootcamp

SECTION V: JOINS

「Video 39」 Introduction to JOINS

- In this section of the course we are going to discuss the various types of JOINS in pg SQL.

JOINS → Allows us to Combine information from multiple tables!

- Section Overview
  - Creating an alias with the AS clause.
  - Understanding different kinds of JOINS
    - INNER JOIN
    - FULL JOIN
    - OUTER JOIN
    - UNION

} challenge tasks!

「Video 40」 AS Statement

AS → Allows us to create an "alias" or alternative name for a column or result.

- Example syntax 1

```
SELECT column AS new_name
FROM table
```

↳ Allows us to rename a column selected from a table. Data output shows the renamed column.

- Example syntax 2

```
SELECT SUM(column) AS new_name
FROM table
```

↳ We can also rename the results of an aggregate function on a column. Data output shows the new alias of the column.

↳ This in turn, will facilitate the understanding of our SQL queries to other analysts.

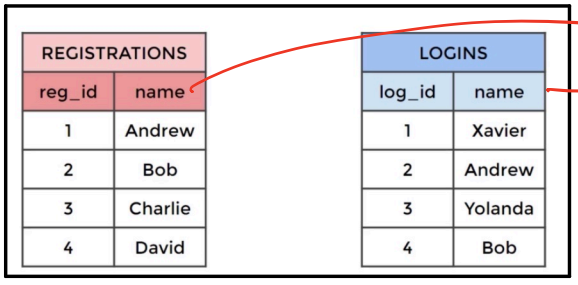
* Readability

- The AS operator gets executed at the very end of a query. -> we CAN'T use the ALIAS inside a WHERE operator.

[Video 41] INNER JOIN

- In this lecture we will go through the simplest JOIN type -> INNER JOIN
- What is a JOIN operation?
  - JOIN allows us to combine multiple tables together.
  - The main reason for the different JOIN types is to decide how to deal with information only present in one of the tables.

- Let's imagine a simple example.
- Our company is holding a conference for people in the movie rental industry.
- We'll have people register online beforehand and then login the day of the conference.
- After the conference we have these tables.



REGISTRATIONS

reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

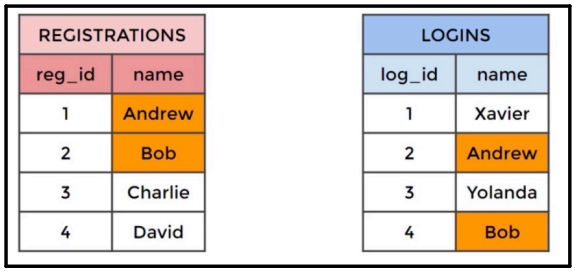
LOGINS

log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

The respective id columns indicate what order they registered or logged in on site.

- For simplicity sake, we will assume the names are unique. (ABCD - Registration id -> Helps you keep track.)
- So, what is an INNER JOIN?

> An INNER JOIN will result with the set of records that match in both tables.



REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

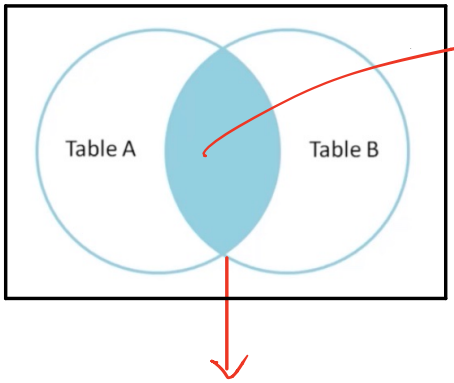
↳ An INNER JOIN will be able to join these two tables on records that match in both tables.

Example syntax

```
SELECT * FROM TABLE A
INNER JOIN TABLE B
ON TABLE A.col_match = TABLE B.col_match
```

↳ ON -> Matches columns that appear on both tables.

- We can use a Venn Diagram to illustrate the above syntax example.



INNER JOIN will only grab the records that happen to be on TABLE A and TABLE B.

- This tables must be joined on a column that appears in both tables. (e.g. name column)

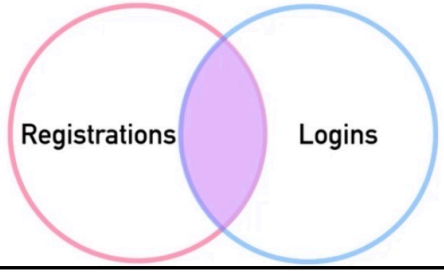
INNER JOIN is Symmetrical!

↳ Syntax can be "switched" with no effect, i.e.:

```
SELECT * FROM TABLE B
INNER JOIN TABLE A
ON TABLE A.col_match = TABLE B.col_match
```

- Let's go back to our example:

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David



LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

↳ INNER JOIN syntax for our example:

```
SELECT * FROM Registrations
INNER JOIN Logins
ON Registrations.name = Logins.name
```

↳ This query will output the following result:

RESULTS			
reg_id	name	log_id	name
1	Andrew	2	Andrew
2	Bob	4	Bob

→ Result of the INNER JOIN. Notice that the name column is duplicated given the SELECT * statement.

↳ If we want to avoid this, we must specify the columns we want in our output.

```
SELECT reg_id, Logins.name, log_id
FROM Registrations
INNER JOIN Logins
ON Registrations.name = Logins.name
```

↳ Because the column "name" exists in both tables, we must fully qualify the table name Logins.name to select the name column.

- Remember that the table order won't matter in an INNER JOIN.
- Also, if you see just JOIN without the INNER, PostgreSQL will treat it as an INNER JOIN.
  - Always recommended to use INNER JOIN.

[Video 42] FULL OUTER JOIN

- There are several types of JOINS → One subset of JOINS is the OUTER JOIN
- There are a few different types of OUTER JOINS.
- OUTER JOIN
  - Will allow us to specify how to deal with values only present in one of the tables being joined
    - They are more complex than INNER JOIN, which is the default JOIN type.
- In these lectures we will explain:
  - FULL OUTER JOIN
    - Clarifying WHERE null
  - LEFT OUTER JOIN
    - Clarifying WHERE null
  - RIGHT OUTER JOIN
    - Clarifying WHERE null

Let's us specify further the records we want.

- Recall from our previous example, we had names that only appear in one table!

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

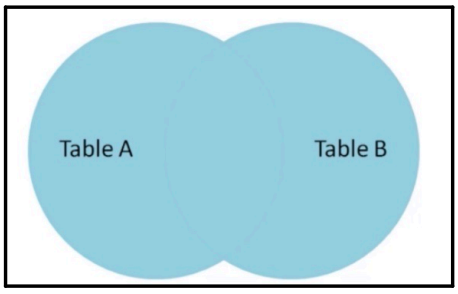
LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

↳ Let's see how the different OUTER JOINS deal with this discrepancy.

- We will first take a look at the simplest, which is the FULL OUTER JOIN

↳ Example Syntax and Venn Diagram:

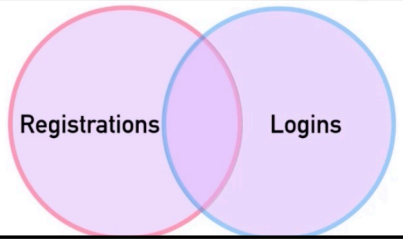
```
SELECT * FROM TABLE A
FULL OUTER JOIN TABLE B
ON TABLE A. col_match = TABLE B. col_match
```



↳ Again, this operation is Symmetrical. Table order is not relevant. FULL OUTER JOIN will grab all columns from both tables regardless if there is a match or not.

Back to our example...

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David



LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

↳ Equivalent syntax:

```
SELECT * FROM Registrations
FULL OUTER JOIN Logins
ON Registrations.name = Logins.name
```

Data output result:

RESULTS			
reg_id	name	log_id	name
1	Andrew	2	Andrew
2	Bob	4	Bob
3	Charlie	null	null
4	David	null	null
null	null	1	Xavier
null	null	3	Yolanda

No name match on Registrations table

No name match on Logins table.

↳ FULL OUTER JOIN grabs everything and fills in with null values where there is no record match between tables.

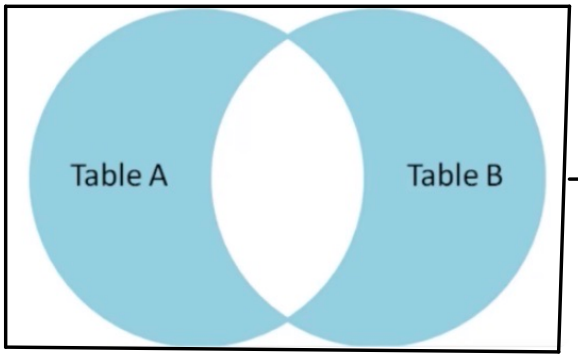
- Now, what if we want only the records that are unique in each table? → where are the null values?

↳ This is the exact opposite of an INNER JOIN. To do so, we qualify a FULL OUTER JOIN with a where statement.

- Consider the following syntax:

```
SELECT * FROM TABLE A
FULL OUTER JOIN TABLE B
ON TABLE A.col_match = TABLE B.col_match
WHERE TABLE A.id IS null
OR TABLE B.id IS null
```

} Could've used name column too. → unique values are found here.



→ Again, this operation is symmetrical. (Table order/JOIN is NOT RELEVANT)

Back to our example...

Equivalent syntax:

```
SELECT * FROM Registrations
FULL OUTER JOIN Logins
ON Registrations.col_match = Logins.col_match
WHERE Registrations.id IS null
OR Logins.id IS null
```

Data output results

RESULTS			
reg_id	name	log_id	name
3	Charlie	null	null
4	David	null	null
null	null	1	Xavier
null	null	3	Yolanda

→ The results show the rows that CAN NOT be found on both tables.

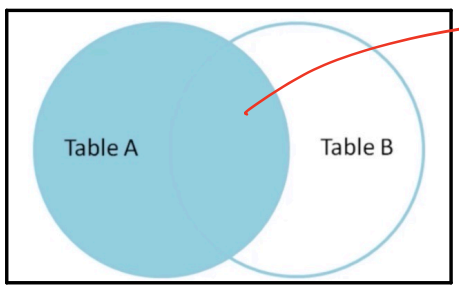
「Video 43」 LEFT OUTER JOIN

- A LEFT OUTER JOIN results in the set of records that are in the left table, if there is no match with the right table, the result is null.

- Later on, we will learn how to modify a LEFT OUTER JOIN by adding a WHERE statement.

- Example syntax

```
SELECT * FROM TABLE A -- Left table
LEFT OUTER JOIN TABLE B
ON TABLE A.col_match = TABLE B.col_match
```



→ Notice that in a LEFT OUTER JOIN the equivalent Venn Diagram is no longer symmetrical.

↓

TABLE ORDER MATTERS!

- Thus, in a LEFT OUTER JOIN we are just grabbing information from TABLE A (Exclusive info) and data that matches in TABLE B.
- If something is only found in TABLE B, it is NOT going to be returned in the query.
- In SQL syntax LEFT OUTER JOIN <=> LEFT JOIN

Equivalent!

- Let's consider our example from previous lessons:

```
SELECT * FROM Registrations
LEFT OUTER JOIN Logins
ON Registrations.name = Logins.name
```

↳ This query returns the following results:

RESULTS			
reg_id	name	log_id	name
1	Andrew	2	Andrew
2	Bob	4	Bob
3	Charlie	null	null
4	David	null	null

Registrations match with Logins table.

No Match in Logins table

Everything from LEFT table is returned.

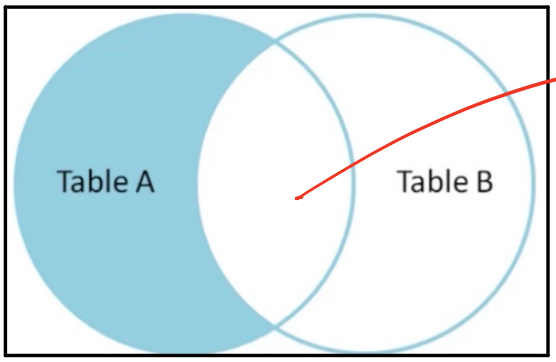
Data exclusive to Registrations table

LEFT OUTER JOIN with WHERE -> Gets rows unique to LEFT TABLE

```
SELECT * FROM TABLE A -- Left table
LEFT OUTER JOIN TABLE B
ON TABLE A.col_match = TABLE B.col_match
WHERE TABLE B.id IS null -> Avoid matching records
```

↳ This query returns the records found in TABLE A AND NOT FOUND in TABLE B.

-> The equivalent Venn Diagram is shown below



Again, a non-symmetrical join.

The equivalent syntax in our example is as follows:

```
SELECT * FROM Registrations
LEFT OUTER JOIN Logins
ON Registrations.name = Logins.name
WHERE Logins.log_id IS null
```

↳ The above query will return the following results:

RESULTS			
reg_id	name	log_id	name
3	Charlie	null	null
4	David	null	null

Because these records are null in the Logins table, the records "Charlie" & "David" are exclusive to the Registrations table.

「Video 44」 RIGHT JOIN

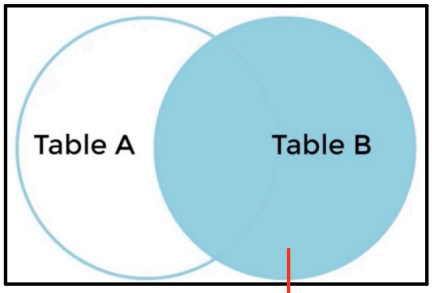
- A RIGHT JOIN is the same as a LEFT JOIN, except the tables are switched.
- This would be the same as switching the table order in a LEFT OUTER JOIN.

Whatever is your need, switching the table order and a LEFT keyword with a RIGHT accomplishes this.

↳ It all the depends in the order of the tables you set "in your mind"

- Consider the following syntax and Venn Diagrams:

```
SELECT * FROM TABLE A -- Left table
LEFT OUTER JOIN TABLE B
ON TABLE A.col_match = TABLE B.col_match
```



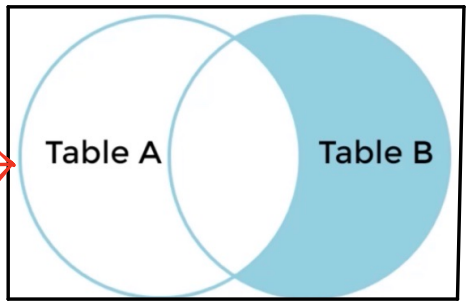
↳ This query specifies to return those records that match in TABLE B and TABLE A, or those that are found exclusively on TABLE B.

Again, this operation is non-symmetrical.

As in the previous lecture, we can add a clarifying WHERE statement on TABLE A:

```
SELECT * FROM TABLE B -- Right table
RIGHT OUTER JOIN TABLE A
ON TABLE A.col_match = TABLE B.col_match
WHERE TABLE A.id IS null -> Avoid matching records
```

↳ This query can be illustrated by the following Venn diagram →



- It is up to you and how you have the tables organized "in your mind" when it comes to choosing a LEFT JOIN vs. RIGHT JOIN.
  - It all depends on the table order you specify in the JOIN, you can perform duplicate JOINS with either method.

「Video 45」 UNION

- The UNION operator is used to combine the result-set of two or more SELECT statements.
  - Serves to directly concatenate two results together, essentially "pasting" them together.

Example syntax

```
SELECT column_name(s) FROM table_1
UNION
SELECT column_name(s) FROM table_1
```

- Columns to be selected and "stacked" together should "match-up" in a way.

↳ Let's explore a UNION with two example tables:

Sales2021_Q1	
name	amount
David	100
Claire	50

Sales2021_Q2	
name	amount
David	200
Claire	100

-> These tables organize the sales per year per quarter.

↳ To get the overall sales results for half a year we can't use JOIN. Instead we concatenate our results (stack them) using the UNION operator:

```
SELECT * FROM Sales2021_Q1
UNION
SELECT * FROM Sales2021_Q2
ORDER BY name
```

-> we can always use another UNION operator should we need to concatenate more records.

↳ This query outputs the following results:

name	amount
David	100
David	200
Claire	50
Claire	100