**SQL Basics Part 5**

* **This PDF about JOINS**

/\*

=============

== 20 JOIN ==

=============

- JOINS are used to combine data from two or more tables based on a related column between them

- they help us answer questions like:

1- get all customers and their orders

2- list employees and departments names

- without JOINs, you'd have to manually fetch data from each table

----------------

-- JOIN Types --

----------------

- INNER JOIN:

- LEFT JOIN:

- RIGHT JOIN:

- FULL JOIN:

\*/

/\*

===================

== 01 INNER Join ==

===================

- the most common Join use between developers

- used to combine rows from two or more tables based on related columns between them

- it only return rows where there's a match in both tables

If a row exists in one table but not in the other → it will be excluded.

It’s the most commonly used join in SQL.

- syntax

SELECT table1.column1, table1.column2, table2.column1, table2.column2

FROM table1

INNER JOIN table2

ON table1.common\_column = table2.common\_column;

\*/

-- list all customers ID, first name, order ID, amount, exclude the customers who have not placed aby order

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

INNER JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY c.customer\_id ASC;

-- we’ll display customer name, country, total score, and order amount

SELECT

c.first\_name,

c.country,

c.score,

o.order\_id,

o.amount

FROM customers AS c

INNER JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY o.amount DESC;

-- show full orders details

SELECT

c.first\_name,

c.country,

o.order\_id,

o.order\_date,

o.amount

FROM customers AS c

INNER JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY o.amount DESC;

-- count orders per customer

SELECT

c.first\_name,

COUNT(o.order\_id) AS total\_orders

FROM customers AS c

INNER JOIN orders AS o

ON c.customer\_id = o.customer\_id

GROUP BY c.first\_name

ORDER BY total\_orders DESC;

/\*

==================

== 21 LEFT JOIN ==

==================

- all rows from the left table matching from the right table

if there's no match, it still show the left table, and write the right table rows as NULL

- syntax

LEFT JOIN = All Left Table Rows + Matching Right Table Rows

If no match → NULL for right table columns

\*/

use MyDatabase;

-- list customer ID, first name, oreder ID, amount. exclude the customers who have no placed any orders

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

LEFT JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY c.customer\_id;

-- Example 2 — Find Customers Without Orders (LEFT JOIN + WHERE)

SELECT

c.customer\_id,

c.first\_name,

c.country

FROM customers AS c

LEFT JOIN orders AS o

ON c.customer\_id = o.customer\_id

WHERE o.order\_id IS NULL;

-- Example 3 — Count Total Orders per Customer (LEFT JOIN + GROUP BY)

SELECT

c.customer\_id,

c.first\_name,

COUNT(o.order\_id) AS total\_orders

FROM customers AS c

LEFT JOIN orders AS o

ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.first\_name

ORDER BY total\_orders DESC;

-- Example 4 — Show Customers and Latest Order Amount (LEFT JOIN + MAX)

SELECT

c.customer\_id,

c.first\_name,

MAX(o.amount) AS latest\_order

FROM customers AS c

LEFT JOIN orders AS o

ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.first\_name

ORDER BY latest\_order DESC;

/\*

===================

== 22 RIGHT JOIN ==

===================

- A RIGHT JOIN returns all rows from the right table (e.g., orders),

and the matching rows from the left table (e.g., customers).

- If there’s no match in the left table, the result will contain NULL for columns from the left table.

\*/

USE MyDatabase;

-- Show me all orders even if there is no matching customer

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

RIGHT JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY o.order\_id;

-- Example 1 — Get All Orders with Customer Names ✅ (Basic RIGHT JOIN)

SELECT

o.order\_id,

o.amount,

c.customer\_id,

c.first\_name

FROM customers AS c

RIGHT JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY o.order\_id;

-- Example 2 — Find Orders Without a Customer (RIGHT JOIN + WHERE)

SELECT

o.order\_id,

o.amount,

o.customer\_id

FROM customers AS c

RIGHT JOIN orders AS o

ON c.customer\_id = o.customer\_id

WHERE c.customer\_id IS NULL;

-- Example 3 — Count Total Orders Per Customer (RIGHT JOIN + GROUP BY)

SELECT

o.customer\_id,

c.first\_name,

COUNT(o.order\_id) AS total\_orders

FROM customers AS c

RIGHT JOIN orders AS o

ON c.customer\_id = o.customer\_id

GROUP BY o.customer\_id, c.first\_name

ORDER BY total\_orders DESC;

/\*

==================

== 23 FULL JOIN ==

==================

- A FULL JOIN (or FULL OUTER JOIN) combines the results of LEFT JOIN and RIGHT JOIN

It returns all rows from both tables.

If there’s a match → it shows data from both tables.

If there’s no match → it shows NULL for the missing values.

\*/

USE MyDatabase;

-- Find Customers Without Orders or Orders Without Customers [MySQL Syntax]

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

LEFT JOIN orders AS o

ON c.customer\_id = o.customer\_id

UNION

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

RIGHT JOIN orders AS o

ON c.customer\_id = o.customer\_id

ORDER BY customer\_id;

-- Find Only Unmatched Rows

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

LEFT JOIN orders AS o

ON c.customer\_id = o.customer\_id

WHERE o.customer\_id IS NULL

UNION

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

RIGHT JOIN orders AS o

ON c.customer\_id = o.customer\_id

WHERE c.customer\_id IS NULL;

-- Find Only Unmatched Rows in SQL Server

SELECT

c.customer\_id,

c.first\_name,

o.order\_id,

o.amount

FROM customers AS c

FULL JOIN orders AS o

ON c.customer\_id = o.customer\_id

WHERE c.customer\_id IS NULL OR o.customer\_id IS NULL;

/\*

==========================

== 24 UNION & UNION ALL ==

==========================

- UNION used to combine the rows from both tables

- used to combines the rows together

-----------

-- UNION --

-----------

- operator used to combine the results of two or more SELECT statements into a single result

- it removes duplicate rows

The number of columns in all SELECT statements must be the same.

The data types of the columns must be compatible.

Column names in the final result are taken from the first SELECT.

- Syntax

SELECT country FROM table1

UNION

SELECT country FROM table2;

---------------

-- UNION ALL --

---------------

- the UNION ALL operator is used to combine results from multiple SELECT without removing duplicates

- Syntax

SELECT country FROM table1

UNION ALL

SELECT country FROM table2;

\*/

USE MyDatabase;

CREATE TABLE students\_2024 (

id INT,

name VARCHAR(50),

country VARCHAR(50)

);

CREATE TABLE students\_2025 (

id INT,

name VARCHAR(50),

country VARCHAR(50)

);

-- Insert data into students\_2024

INSERT INTO students\_2024 (id, name, country) VALUES

(1, 'Ali', 'Egypt'),

(2, 'Sara', 'USA'),

(3, 'John', 'UK'),

(4, 'Mona', 'Canada');

-- Insert data into students\_2025

INSERT INTO students\_2025 (id, name, country) VALUES

(3, 'John', 'UK'),

(4, 'Mona', 'Canada'),

(5, 'Youssef', 'Egypt'),

(6, 'Emma', 'France');

-- UNION removes duplicate records automatically

SELECT name, country FROM students\_2024

UNION

SELECT name, country FROM students\_2025

ORDER BY country ASC;

-- Count Unique Students

SELECT COUNT(\*) AS unique\_students

FROM (

SELECT name FROM students\_2024

UNION

SELECT name FROM students\_2025

) AS unique\_list;

-- UNION ALL keeps duplicates.

SELECT name, country FROM students\_2024

UNION ALL

SELECT name, country FROM students\_2025;

-- Count How Many Students in Both Tables

SELECT COUNT(\*) AS total\_students

FROM (

SELECT name FROM students\_2024

UNION ALL

SELECT name FROM students\_2025

) AS all\_students;