📘 Module 23: JOINS & COMBINING QUERIES

🧾 Reference Tables Used:

👤 Table: Customers Orders

```
Select * from Customers;
Select * from Orders;
```

100 %

Results Messages

	CustomerID	CustomerName	ContactNumber
1	1	John Doe	123-456-7890
2	2	Jane Smith	234-567-8901
3	3	Alice Johnson	345-678-9012
4	4	Bob Brown	456-789-0123
5	5	Carol White	567-890-1234

	OrderID	OrderDate	CustomerID	Amount
1	1	2023-06-01	1	150.00
2	2	2023-06-03	2	200.00
3	3	2023-06-05	3	300.00
4	4	2023-06-07	6	250.00
5	5	2023-06-09	7	175.00

👤 Table: Table1 Table2

```
Select * from Table1;
Select * from Table2;
```

100 %

Results Messages

	ID	Name	Value
1	1	Alice	100
2	2	Bob	200
3	3	Charlie	300
4	4	David	400
5	5	Eve	500

	ID	Name	Value
1	3	Charlie	300
2	4	David	400
3	5	Eve	500
4	6	Frank	600
5	7	Grace	700

🔗 1. What are JOINS?

🧩 Definition:

A **JOIN** is used in SQL to **combine data from two or more tables** based on a related column (usually a primary and foreign key).

📌 Why use JOINs?

- To fetch data spread across multiple tables
- For combining customer info, order data, employee details, etc.

📚 Types of Joins:

Type	Description
INNER JOIN	Returns only matching rows from both tables
LEFT JOIN	All rows from the left table + matched rows from the right
RIGHT JOIN	All rows from the right table + matched rows from the left
FULL JOIN	All rows when there is a match in either table
CROSS JOIN	Returns the Cartesian product (every row from one joined with every row from another)

🧱🔁 Common SQL JOIN Syntax

```
SELECT
A.column1, A.column2,
B.column1, B.column2
FROM
TableA AS A
<JOIN_TYPE> JOIN TableB AS B
ON A.common_column = B.common_column;
```

✅ Example for All Joins (with tables: `customers` and `orders`)

JOIN Type	Example
INNER JOIN	`FROM customers INNER JOIN orders ON customers.customerid = orders.customerid`
LEFT JOIN	`FROM customers LEFT JOIN orders ON customers.customerid = orders.customerid`
RIGHT JOIN	`FROM customers RIGHT JOIN orders ON customers.customerid = orders.customerid`
FULL JOIN	`FROM customers FULL OUTER JOIN orders ON customers.customerid = orders.customerid`
CROSS JOIN	`FROM customers CROSS JOIN orders` *(no ON clause required)*

🔄 2. INNER JOIN

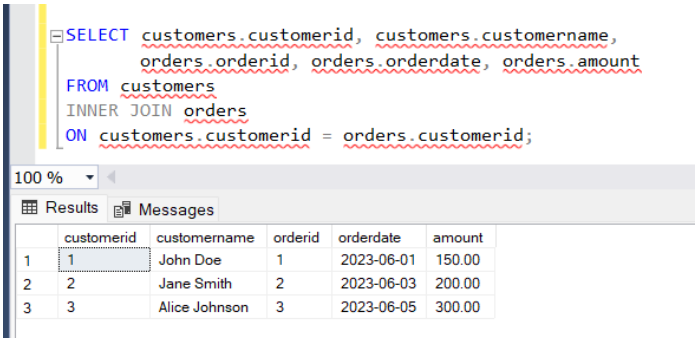
💡 **Definition:**

Returns **only the matching rows** from both tables.

🔣 Syntax | Example:

```
SELECT customers.customerid, customers.customername,
      orders.orderid, orders.orderdate, orders.amount
FROM customers INNER JOIN orders
ON customers.customerid = orders.customerid;
```

🧪 Result: Only customers who placed orders.



```
SELECT customers.customerid, customers.customername,
       orders.orderid, orders.orderdate, orders.amount
FROM customers
INNER JOIN orders
ON customers.customerid = orders.customerid;
```

100 %

Results | Messages

	customerid	customername	orderid	orderdate	amount
1	1	John Doe	1	2023-06-01	150.00
2	2	Jane Smith	2	2023-06-03	200.00
3	3	Alice Johnson	3	2023-06-05	300.00

👈 3. LEFT JOIN (LEFT OUTER JOIN)

💡 **Definition:**

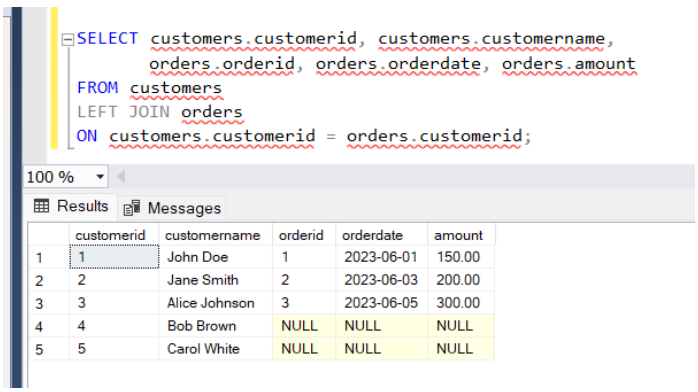
Returns **all rows from the left table (customers)** and **matching rows** from the right table (orders).

If no match, NULLs are shown for right table columns.

🔣 Syntax | Example:

```
SELECT customers.customerid, customers.customername,
      orders.orderid, orders.orderdate, orders.amount
FROM customers LEFT JOIN orders
ON customers.customerid = orders.customerid;
```

🧪 Result: All customers, even those with no orders.



```
SELECT customers.customerid, customers.customername,
       orders.orderid, orders.orderdate, orders.amount
FROM customers
LEFT JOIN orders
ON customers.customerid = orders.customerid;
```

100 %

Results | Messages

	customerid	customername	orderid	orderdate	amount
1	1	John Doe	1	2023-06-01	150.00
2	2	Jane Smith	2	2023-06-03	200.00
3	3	Alice Johnson	3	2023-06-05	300.00
4	4	Bob Brown	NULL	NULL	NULL
5	5	Carol White	NULL	NULL	NULL

👉 4. RIGHT JOIN (RIGHT OUTER JOIN)

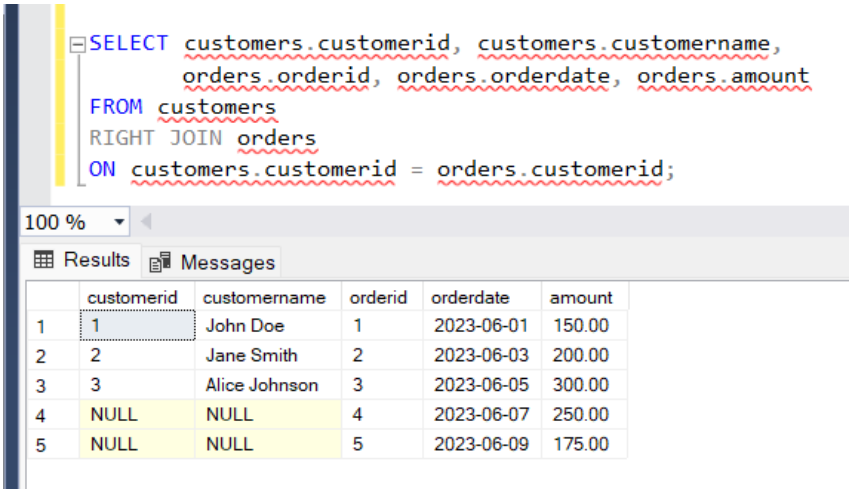
💡 **Definition:**

Returns **all rows from the right table (orders)** and **matching rows** from the left table (customers).

🔣 Syntax | Example:

```
SELECT customers.customerid, customers.customername,
      orders.orderid, orders.orderdate, orders.amount
FROM customers
RIGHT JOIN orders
ON customers.customerid = orders.customerid;
```

🧪 Result: All orders, even if some customers are missing.



	customerid	customername	orderid	orderdate	amount
1	1	John Doe	1	2023-06-01	150.00
2	2	Jane Smith	2	2023-06-03	200.00
3	3	Alice Johnson	3	2023-06-05	300.00
4	NULL	NULL	4	2023-06-07	250.00
5	NULL	NULL	5	2023-06-09	175.00

🔄 5. FULL OUTER JOIN

💡 **Definition:**

Returns **all rows when there is a match in either left or right table**.

If no match, NULLs are filled accordingly.

🔣 Syntax | Example:

```
SELECT customers.customerid, customers.customername,
      orders.orderid, orders.orderdate, orders.amount
FROM customers
FULL OUTER JOIN orders
ON customers.customerid = orders.customerid;
```

🧪 Result: All customers and all orders, matching where possible.

```sql
SELECT customers.customerid, customers.customername,
       orders.orderid, orders.orderdate, orders.amount
FROM customers
FULL OUTER JOIN orders
ON customers.customerid = orders.customerid;
```

100 %

Results | Messages

	customerid	customername	orderid	orderdate	amount
1	1	John Doe	1	2023-06-01	150.00
2	2	Jane Smith	2	2023-06-03	200.00
3	3	Alice Johnson	3	2023-06-05	300.00
4	4	Bob Brown	NULL	NULL	NULL
5	5	Carol White	NULL	NULL	NULL
6	NULL	NULL	4	2023-06-07	250.00
7	NULL	NULL	5	2023-06-09	175.00

🔁 6. CROSS JOIN

💡 **Definition:**

Returns the **Cartesian product** — every row from first table with **every row from second**.

🔣 Syntax | Example:

SELECT customers.customerid AS CustID, customers.customername AS CustName,
orders.orderid AS OrdID, orders.orderdate AS OrdDate, orders.amount AS OrdAmount
FROM customers
CROSS JOIN orders;

🧪 Result: If 3 customers × 4 orders = 12 rows.

⚠️ Use carefully on large data!

```sql
SELECT customers.customerid AS CustID, customers.customername AS CustName,
       orders.orderid AS OrdID, orders.orderdate AS OrdDate, orders.amount AS OrdAmount
FROM customers
CROSS JOIN orders;
```

100 %

Results | Messages

	CustID	CustName	OrdID	OrdDate	OrdAmount
1	1	John Doe	1	2023-06-01	150.00
2	2	Jane Smith	1	2023-06-01	150.00
3	3	Alice Johnson	1	2023-06-01	150.00
4	4	Bob Brown	1	2023-06-01	150.00
5	5	Carol White	1	2023-06-01	150.00
6	1	John Doe	2	2023-06-03	200.00
7	2	Jane Smith	2	2023-06-03	200.00
8	3	Alice Johnson	2	2023-06-03	200.00
9	4	Bob Brown	2	2023-06-03	200.00
10	5	Carol White	2	2023-06-03	200.00
11	1	John Doe	3	2023-06-05	300.00
12	2	Jane Smith	3	2023-06-05	300.00
13	3	Alice Johnson	3	2023-06-05	300.00
14	4	Bob Brown	3	2023-06-05	300.00
15	5	Carol White	3	2023-06-05	300.00
16	1	John Doe	4	2023-06-07	250.00
17	2	Jane Smith	4	2023-06-07	250.00
18	3	Alice Johnson	4	2023-06-07	250.00
19	4	Bob Brown	4	2023-06-07	250.00
20	5	Carol White	4	2023-06-07	250.00
21	1	John Doe	5	2023-06-09	175.00
22	2	Jane Smith	5	2023-06-09	175.00
23	3	Alice Johnson	5	2023-06-09	175.00

Query executed successfully.

🔀 Combining Queries (Set Operators)

〓 7. INTERSECT

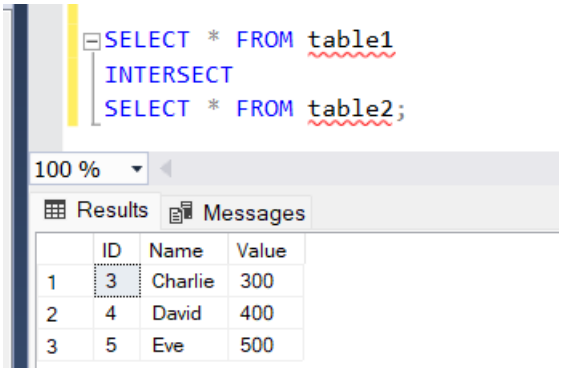
💡 **Definition:**

Returns **common rows** from both queries.

🔣 Syntax | Example:

```
SELECT * FROM table1
INTERSECT
SELECT * FROM table2;
```

🧪 Result: Only rows present in both `table1` and `table2`.



— 8. EXCEPT

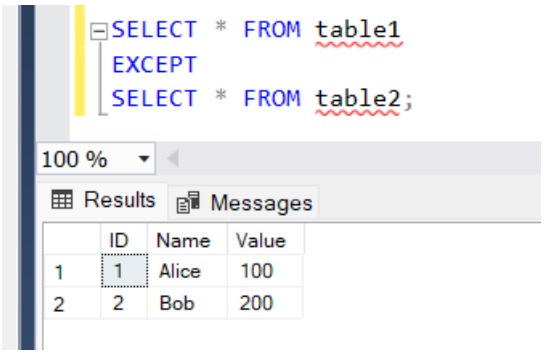
💡 **Definition:**

Returns rows from **first query that are not present** in second.

🔣 Syntax | Example:

```
SELECT * FROM table1
EXCEPT
SELECT * FROM table2;
```

🧪 Result: Data unique to `table1`.



➕ 9. UNION / UNION ALL

💡 **Definition:**

- **UNION** returns **unique** records from both queries.
- **UNION ALL** includes **duplicates**.

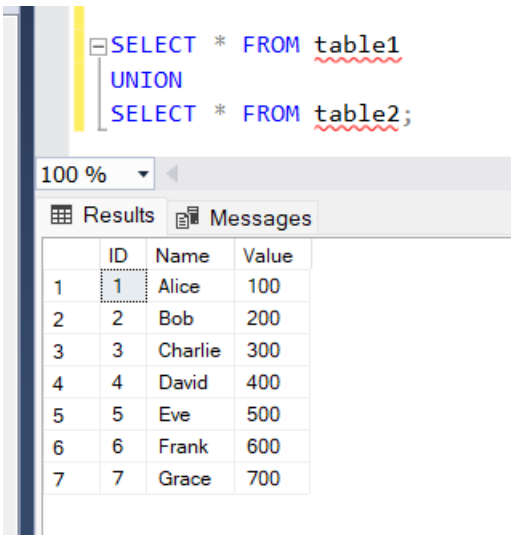
🔣 Syntax | Example:

```
SELECT * FROM table1
UNION
SELECT * FROM table2;

-- OR with duplicates:

SELECT * FROM table1
UNION ALL
SELECT * FROM table2;
```

🧪 Tables must have a **same number of columns** and compatible data types.



	ID	Name	Value
1	1	Alice	100
2	2	Bob	200
3	3	Charlie	300
4	4	David	400
5	5	Eve	500
6	6	Frank	600
7	7	Grace	700

🧠 Key Points to Remember

✅ Use `ON` for join conditions, not just `WHERE`

✅ `LEFT JOIN` brings unmatched right rows as NULL

✅ `FULL JOIN` = all matched + unmatched rows

✅ `INTERSECT` = common only

✅ `EXCEPT` = remove common

✅ `UNION` = combine

✅ `UNION ALL` = combine + duplicates

💡 Additional Tips

🔷 Always use **aliases** for cleaner joins in big queries

🔷 Use **JOIN conditions properly** to avoid unexpected cartesian results

🔷 Prefer **INNER JOIN** if only matching rows are needed — more performance-friendly

🔷 Use **IS NULL** to detect non-matching rows after OUTER JOIN

🔷 Use **ORDER BY** at the end of combined queries if needed
