Join in SQL :

What is a JOIN in SQL?

A **JOIN** is an SQL operation used to combine data from two or more tables based on a related column between them. It is an essential part of relational databases because it allows you to retrieve and manipulate data from multiple tables in a single query. By using JOIN, you can extract information from different tables that are logically connected, such as orders, customers, products, etc.

When Do We Need a JOIN?

You typically need to use a JOIN when:

- **There is related data in multiple tables**: For example, customer details might be in one table, and their orders in another.
- **You want to combine information**: For example, combining order amounts with customer names, or product details with sales data.
- **You want to filter or analyze based on data from multiple sources**: For example, you may want to find customers who have ordered a specific product, so you need data from both the customers' and orders' tables.

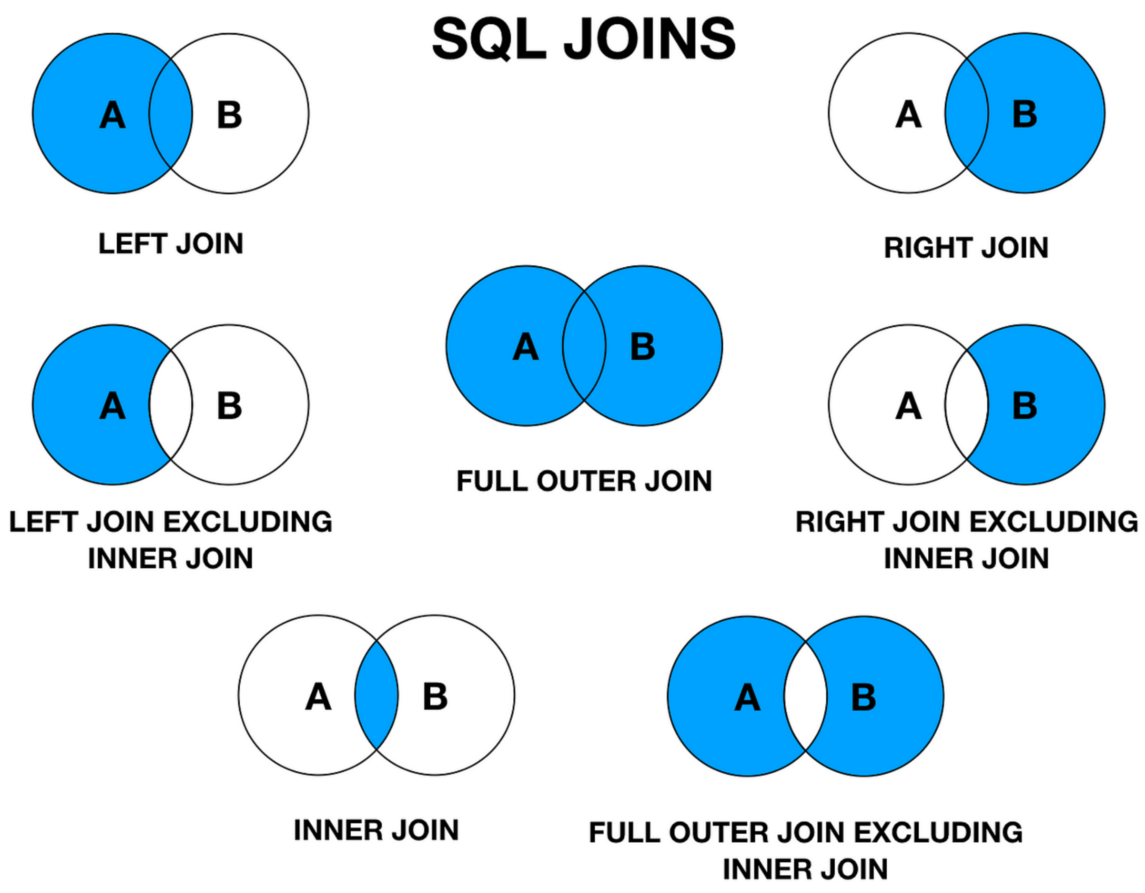
How to Retrieve Data Using JOINS?

To retrieve data using JOINS:

1. **Identify the related columns**: You need to know the column(s) that connect the two tables. For example, `customer_id` in the `orders` table may link to the `id` in the `customers` table.
2. **Use the correct JOIN type**: Depending on your needs (e.g., INNER JOIN for matched data, LEFT JOIN for all left-side data), choose the appropriate type of JOIN.
3. **Write the JOIN query**: Based on the type of JOIN you choose, structure your query to include the relevant tables and columns.

When to Use Each Type of JOIN:

- **INNER JOIN**: Use when you want to return only matching rows from both tables.
- **LEFT JOIN**: Use when you need all rows from the left table and matching rows from the right table (or NULL if there's no match).
- **RIGHT JOIN**: Use when you need all rows from the right table and matching rows from the left table (or NULL if there's no match).
- **FULL OUTER JOIN**: Use when you need to return all rows from both tables, regardless of whether there's a match.
- **CROSS JOIN**: Use when you need the Cartesian product of both tables, meaning every combination of rows from both tables.


SQL JOINS
A
B
LEFT JOIN
RIGHT JOIN
FULL OUTER JOIN
LEFT JOIN EXCLUDING INNER JOIN
RIGHT JOIN EXCLUDING INNER JOIN
INNER JOIN
FULL OUTER JOIN EXCLUDING INNER JOIN

The `INNER JOIN` keyword selects records that have matching values in both tables.

JOIN or INNER JOIN

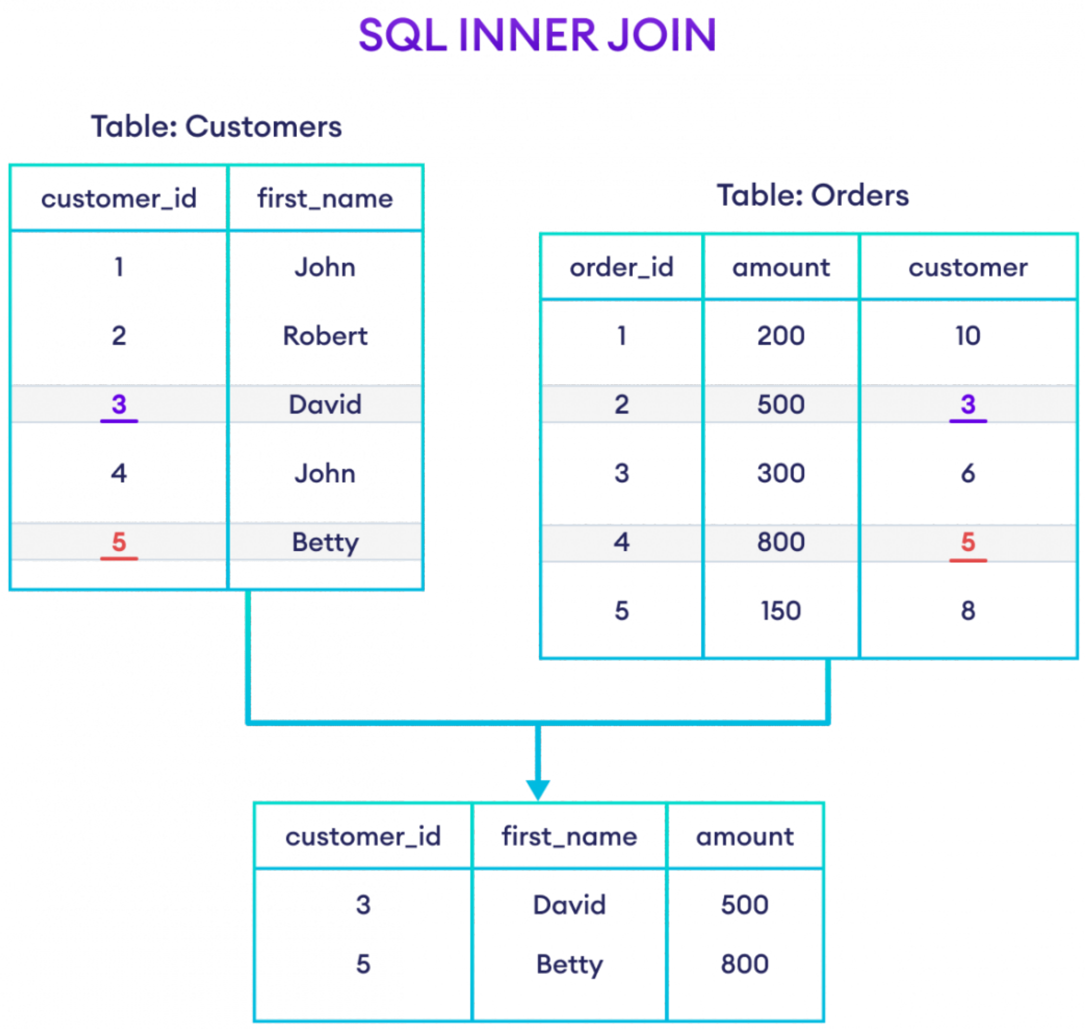


Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
3	David	500
5	Betty	800

`JOIN` and `INNER JOIN` will return the same result.

`INNER` is the default join type for `JOIN`, so when you write `JOIN` the parser actually writes `INNER JOIN`.

Step 1: Create Tables

1. **Create the `customers` table**:

```
sql
Copy code
```

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50)
);
```

1. **Create the `orders` table**:

```
sql
Copy code
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    amount INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_i
d)
);
```

Step 2: Insert Data into Tables

1. **Insert data into the `customers` table**:

```
sql
Copy code
INSERT INTO customers (customer_id, name, city)
VALUES
(1, 'Alice', 'New York'),
(2, 'Bob', 'Los Angeles'),
(3, 'Charlie', 'San Francisco'),
(4, 'David', 'Chicago');
```

1. **Insert data into the `orders` table**:

```
sql
Copy code
INSERT INTO orders (order_id, customer_id, amount)
VALUES
(1001, 1, 500),
(1002, 2, 700),
(1003, 3, 300),
(1004, 4, 600);
```

Step 3: Verify Data in Both Tables

1. Check the data in the `customers` table:

```
sql
Copy code
SELECT * FROM customers;
```

customer_id	name	city
1	Alice	New York
2	Bob	Los Angeles
3	Charlie	San Francisco
4	David	Chicago

1. Check the data in the `orders` table:

```
sql
Copy code
SELECT * FROM orders;
```

order_id	customer_id	amount

1001	1	500
1002	2	700
1003	3	300
1004	4	600

Step 4: Perform an `INNER JOIN` for All Columns

To combine **all columns** from both tables where the `customer_id` matches, use this query:

```
sql
Copy code
SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id;
```

Result of the Query:

customer_id	name	city	order_id	amount
1	Alice	New York	1001	500
2	Bob	Los Angeles	1002	700
3	Charlie	San Francisco	1003	300
4	David	Chicago	1004	600

Explanation:

1. The `INNER JOIN` combines rows from both tables based on the `customer_id` column.
2. The query selects all necessary columns:
    - From `customers`: `customer_id`, `name`, `city`
    - From `orders`: `order_id`, `amount`
3. All records where `customer_id` exists in both tables are displayed.

Step 5: Test Different JOIN Types (Optional)

- **LEFT JOIN**: Includes all rows from the left table (`customers`) and matching rows from the right table (`orders`):

```
sql
Copy code
SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
LEFT JOIN orders
ON customers.customer_id = orders.customer_id;
```

- **RIGHT JOIN**: Includes all rows from the right table (`orders`) and matching rows from the left table (`customers`):

```
sql
Copy code
SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
RIGHT JOIN orders
ON customers.customer_id = orders.customer_id;
```

- **FULL OUTER JOIN**: Combines all rows from both tables (not always supported in all SQL databases):

```
sql
Copy code
SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
FULL OUTER JOIN orders
ON customers.customer_id = orders.customer_id;
```

Types of SQL JOINS

1. **INNER JOIN**
2. **LEFT JOIN (LEFT OUTER JOIN)**
3. **RIGHT JOIN (RIGHT OUTER JOIN)**
4. **FULL OUTER JOIN**
5. **SELF JOIN**
6. **CROSS JOIN**

---

1. INNER JOIN

```
use employees;

CREATE TABLE customers (
customer_id INT PRIMARY KEY,
name VARCHAR(50),
city VARCHAR(50)
);

INSERT INTO customers (customer_id, name, city)
VALUES
(1, 'Alice', 'New York'),
(2, 'Bob', 'Los Angeles'),
(3, 'Charlie', 'San Francisco'),
(4, 'David', 'Chicago');

CREATE TABLE orders (
order_id INT PRIMARY KEY,
customer_id INT,
amount INT,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

select * from orders;

select * from customers;

SELECT
customers.customer_id,

customers.name,
customers.city,
orders.order_id,
orders.amount
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id;

INSERT INTO customers (customer_id, name, city)
VALUES (5, 'Eve', 'Seattle');
```

```
INSERT INTO orders (order_id, customer_id, amount)
VALUES (5, NULL, 500) ;
```

```
INSERT INTO customers (customer_id, name, city)
VALUES (6, 'Eve1', 'Seattle');
```

```
INSERT INTO orders (order_id, customer_id, amount)
VALUES (4, 6, 500) ;
```

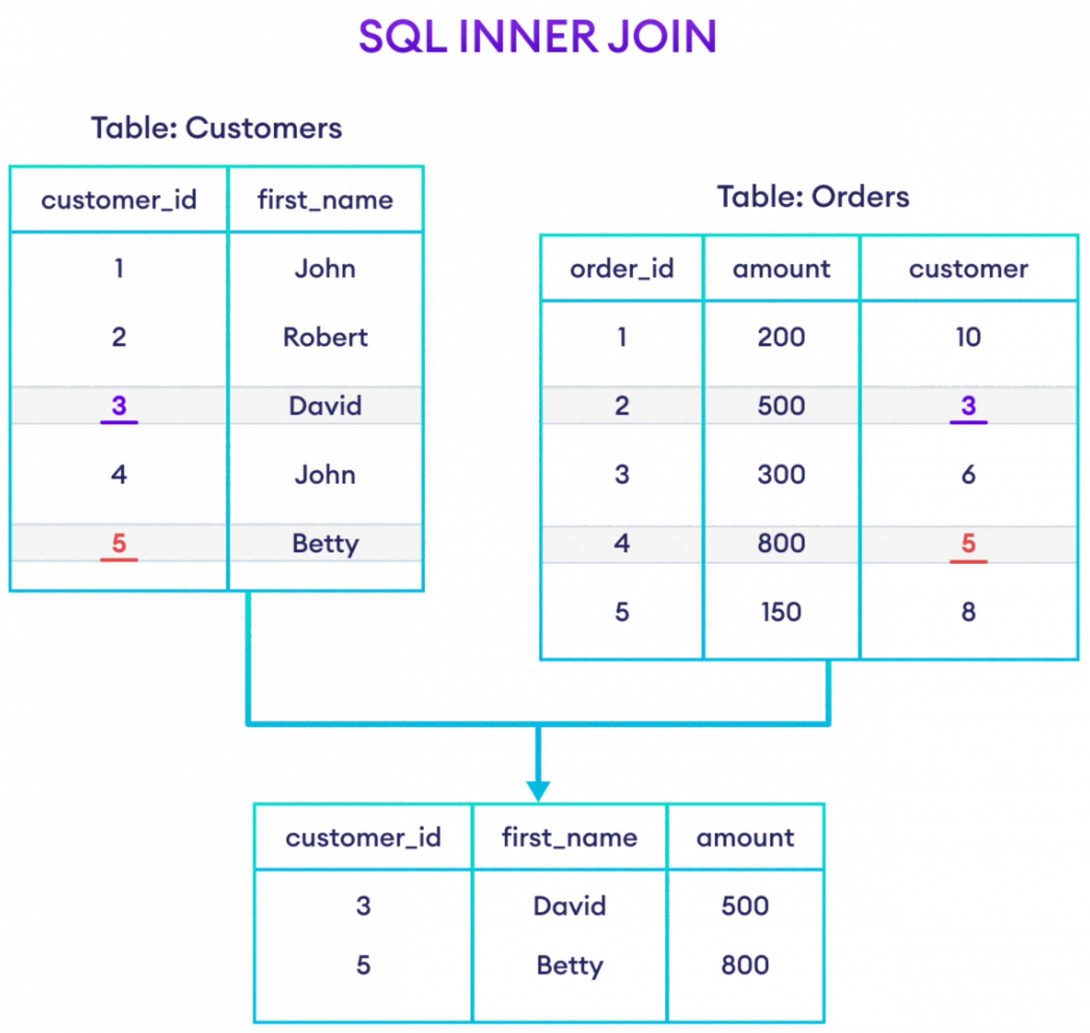


Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
3	David	500
5	Betty	800

Definition

- Combines rows from both tables where there is a match in the specified columns.
- Rows with no matching values are excluded.

Syntax

```
SELECT table1.col1, table2.col2
FROM table1
INNER JOIN table2
ON table1.col = table2.col;
```

Handling NULL Values

- Rows with `NULL` values in the join condition are excluded because `NULL = NULL` evaluates to **false**.

Handling Duplicates

- Duplicate rows are included unless specified using `DISTINCT`.

Features

- Retrieves matching rows only.
- Excludes unmatched rows.

When to Use

- When you only want rows with matching values between the two tables.

---

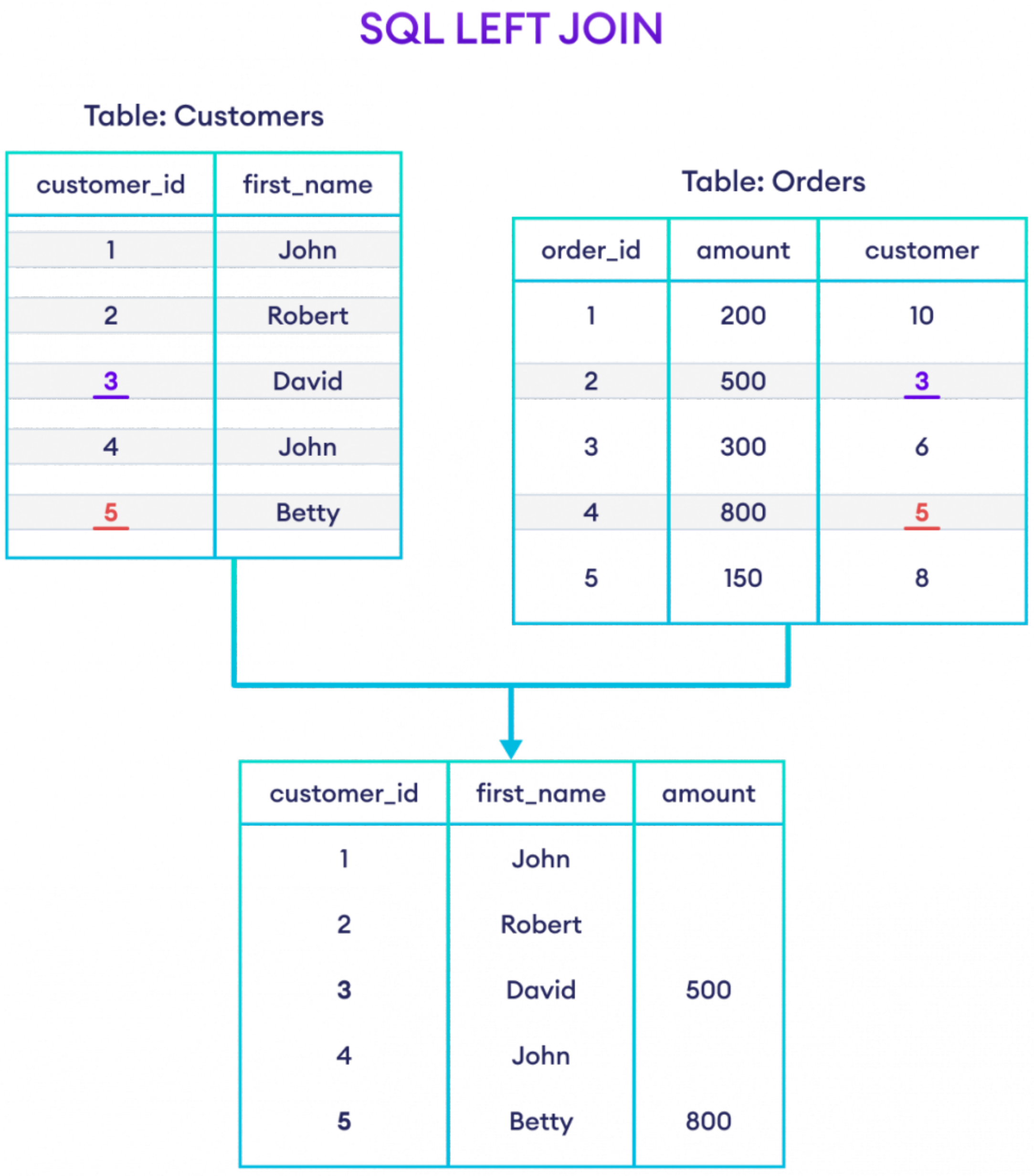
2. LEFT JOIN (LEFT OUTER JOIN)

class code :

```
use employees;

CREATE TABLE customers (
customer_id INT PRIMARY KEY,
name VARCHAR(50),
city VARCHAR(50)
);

INSERT INTO customers (customer_id, name, city)
VALUES
(1, 'Alice', 'New York'),
(2, 'Bob', 'Los Angeles'),
(3, 'Charlie', 'San Francisco'),
(4, 'David', 'Chicago');

CREATE TABLE orders (
order_id INT PRIMARY KEY,
customer_id INT,
amount INT,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

select * from orders;

select * from customers;

SELECT
customers.customer_id,

customers.name,
customers.city,
orders.order_id,
orders.amount
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id;
```

```
INSERT INTO customers (customer_id, name, city)
VALUES (5, 'Eve', 'Seattle');
```

```
INSERT INTO orders (order_id, customer_id, amount)
VALUES (5, NULL, 500) ;
```



SQL LEFT JOIN

Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
1	John	
2	Robert	
3	David	500
4	John	
5	Betty	800

Definition

- Retrieves all rows from the **left table** and matching rows from the right table.
- Rows with no match in the right table will contain `NULL` values.

Syntax

```
sql
Copy code
SELECT table1.col1, table2.col2
FROM table1
LEFT JOIN table2
ON table1.col = table2.col;
```

Handling NULL Values

- Unmatched rows from the **right table** will show `NULL` values in the output.

Handling Duplicates

- Duplicate rows are included unless `DISTINCT` is used.

Features

- Ensures all rows from the left table are included.
- NULL values are shown for unmatched rows in the right table.

When to Use

- When you need all data from the left table, regardless of matching rows in the right table.

---

3. RIGHT JOIN (RIGHT OUTER JOIN)

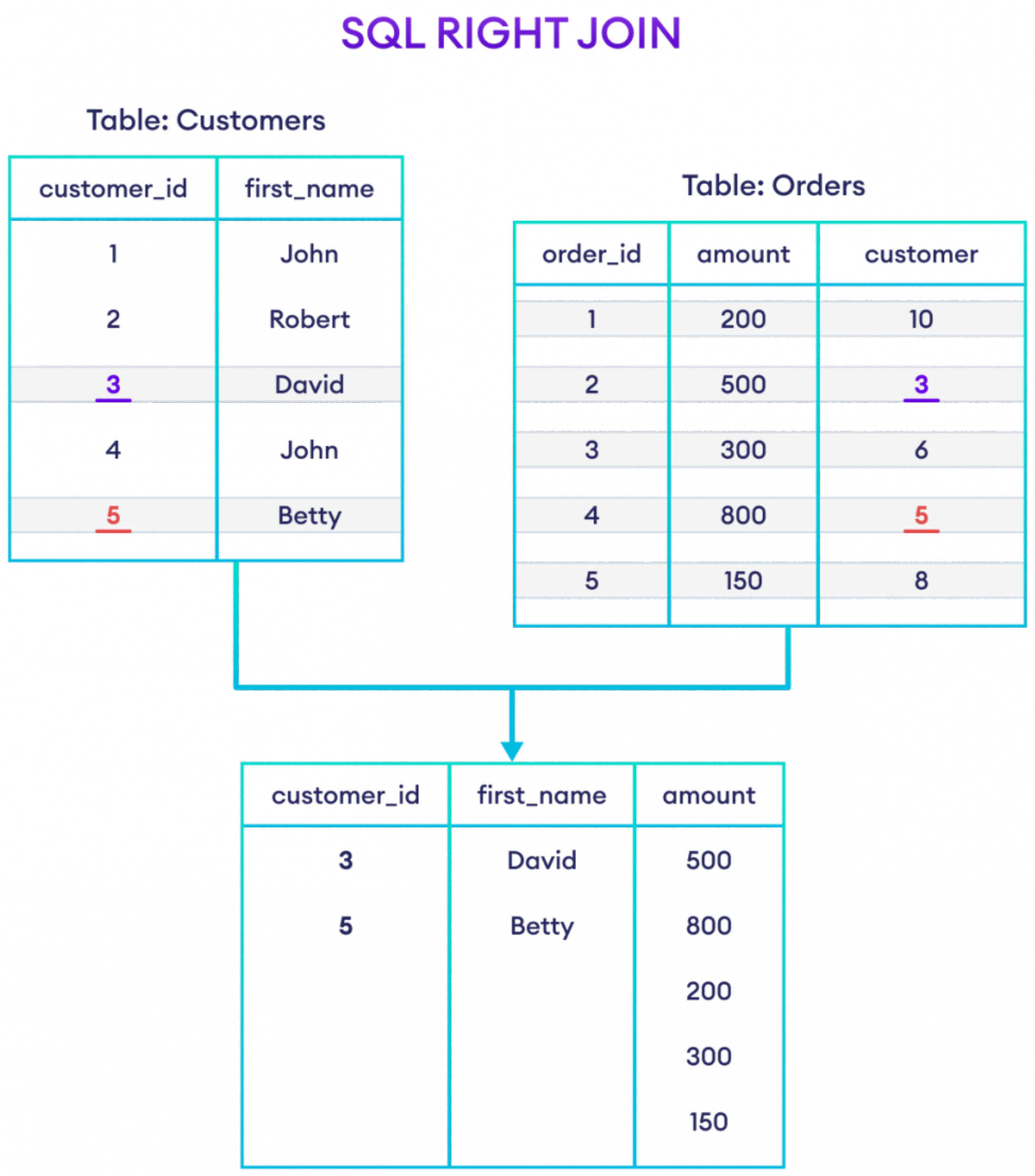


Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
3	David	500
5	Betty	800
		200
		300
		150

Definition

- Retrieves all rows from the **right table** and matching rows from the left table.
- Rows with no match in the left table will contain `NULL` values.

Syntax

```
SELECT table1.col1, table2.col2
FROM table1
RIGHT JOIN table2
ON table1.col = table2.col;
```

Handling NULL Values

- Unmatched rows from the **left table** will show `NULL` values in the output.

Handling Duplicates

- Duplicate rows are included unless `DISTINCT` is used.

Features

- Ensures all rows from the right table are included.
- NULL values are shown for unmatched rows in the left table.

When to Use

- When you need all data from the right table, regardless of matching rows in the left table.

---

4. FULL OUTER JOIN :

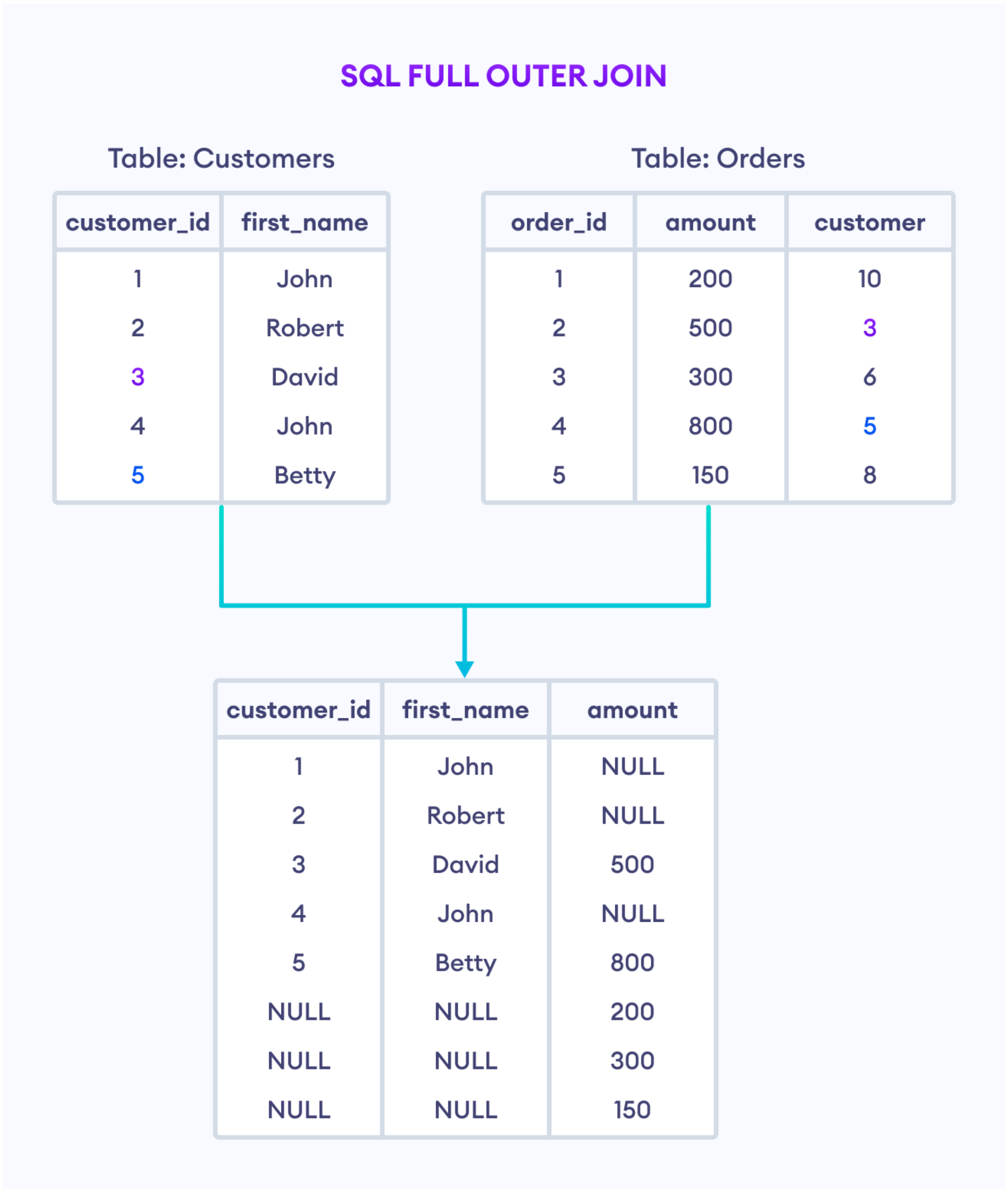


Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
1	John	NULL
2	Robert	NULL
3	David	500
4	John	NULL
5	Betty	800
NULL	NULL	200
NULL	NULL	300
NULL	NULL	150

Definition

- Retrieves all rows from both tables, with `NULL` values for unmatched rows.

Standard Syntax for FULL OUTER JOIN

If your database supports `FULL OUTER JOIN` (e.g., PostgreSQL, SQL Server, Oracle):

```
sql
Copy code
SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
FULL OUTER JOIN orders
ON customers.customer_id = orders.customer_id;
```

How It Works:

1. Matches rows from both `customers` and `orders` based on `customer_id`.
2. Includes unmatched rows from both tables:
    - Unmatched rows from `customers` will have `NULL` for `orders` columns.
    - Unmatched rows from `orders` will have `NULL` for `customers` columns.

---

2. Simulating FULL OUTER JOIN in MySQL

Since MySQL does not support `FULL OUTER JOIN`, you can use `LEFT JOIN` + `RIGHT JOIN` with `UNION`.

Syntax for MySQL:

```
SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
LEFT JOIN orders
ON customers.customer_id = orders.customer_id

UNION

SELECT
    customers.customer_id,
    customers.name,
    customers.city,
    orders.order_id,
    orders.amount
FROM customers
RIGHT JOIN orders
ON customers.customer_id = orders.customer_id;
```

Handling NULL Values

- Rows with no match in either table will have `NULL` values in the output.

Handling Duplicates

- Duplicate rows are included unless `DISTINCT` is used.

Features

- Ensures all rows from both tables are included.
- NULL values are displayed where no match exists.

When to Use

- When you need all data from both tables, regardless of matching rows.

---

5. SELF JOIN

Definition

- A table is joined with itself.
- Useful for finding relationships within the same table.

Syntax

```
sql
Copy code
SELECT A.col1, B.col2
FROM table1 A
JOIN table1 B
ON A.col = B.col;
```

Handling NULL Values

- Same as INNER JOIN.

Handling Duplicates

- Duplicates appear if not controlled using `DISTINCT`.

Features

- A table is treated as two separate tables (aliased).

When to Use

- When comparing rows in the same table.

---

6. CROSS JOIN

Definition

- Produces a Cartesian product of rows from both tables (every row in Table1 is paired with every row in Table2).

Syntax

```
sql
Copy code
SELECT table1.col1, table2.col2
FROM table1
CROSS JOIN table2;
```

Handling NULL Values

- NULL values appear as they are, no special handling.

Handling Duplicates

- Duplicates arise due to the Cartesian product.

Features

- Total rows = `rows in Table1` × `rows in Table2`.

When to Use

- When you need all possible combinations of rows from two tables.

Comparison of JOIN Types

JOIN Type	Includes Rows From	Handles NULLs	Use Case Example
INNER JOIN	Both tables (matches only)	Excludes NULLs in join column	Matching rows only.
LEFT JOIN	Left table (all) + matches	NULL for unmatched right rows	Retain all left table rows.
RIGHT JOIN	Right table (all) + matches	NULL for unmatched left rows	Retain all right table rows.
FULL OUTER JOIN	Both tables (all rows)	NULL for unmatched rows	Retain all rows from both tables.
SELF JOIN	Same table	Depends on condition	Compare rows within the same table.

CROSS JOIN	Cartesian product	No special handling	Generate all combinations of rows.

Handling NULL Values and Duplicates

How to Handle NULL Values

- Use the `COALESCE()` function to replace `NULL` with a default value:

```
sql
Copy code
SELECT name, COALESCE(amount, 0) AS amount
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_i
d;
```

How to Remove Duplicates

- Use the `DISTINCT` keyword:

```
sql
Copy code
SELECT DISTINCT customers.name, orders.amount
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_
id;
```

When to Choose Each JOIN Type

1. **INNER JOIN**:
    - When you only want rows with matches in both tables.
2. **LEFT JOIN**:

- When you need all rows from the left table, even if there are no matches.

3. **RIGHT JOIN**:
    - When you need all rows from the right table, even if there are no matches.
4. **FULL OUTER JOIN**:
    - When you need all rows from both tables, with `NULL` values for non-matches.
5. **SELF JOIN**:
    - When comparing rows within the same table (e.g., employee hierarchy).
6. **CROSS JOIN**:
    - When all possible combinations of rows from two tables are needed.