### **Introduction to Joins**

Joins are used in SQL to combine rows from two or more tables based on related columns between them. It allows you to retrieve data from multiple tables in a single query.

### **Types of Joins**

#### **Inner Join**

An inner join returns only the rows that have matching values in both tables.

SELECT \*

FROM table1

INNER JOIN table2 ON table1.column\_name = table2.column\_name;

#### **Left Join (or Left Outer Join)**

A left join returns all rows from the left table and the matched rows from the right table. If there's no match, NULL values are returned for the right table columns.

SELECT \*

FROM table1

LEFT JOIN table2 ON table1.column\_name = table2.column\_name;

#### **Right Join (or Right Outer Join)**

A right join returns all rows from the right table and the matched rows from the left table. If there's no match, NULL values are returned for the left table columns.

SELECT \*

FROM table1

RIGHT JOIN table2 ON table1.column\_name = table2.column\_name;

#### **Full Join (or Full Outer Join)**

A full join returns all rows when there is a match in either left or right table. If there's no match, NULL values are returned for the unmatched side.

SELECT \*

FROM table1

FULL JOIN table2 ON table1.column\_name = table2.column\_name;

#### **Cross Join**

A cross join returns the Cartesian product of two tables, resulting in a combination of every row from the first table with every row from the second table.

SELECT \*

FROM table1

CROSS JOIN table2;

#### 

#### **Natural Join**

A natural join performs a join based on all columns in the two tables that have the same name. It automatically matches the columns with the same name.

SELECT \*

FROM table1

NATURAL JOIN table2;

### **Union & Union ALL**

The UNION operator is used to combine the result sets of two or more SELECT statements into a single result set. It removes duplicate rows.

SELECT column1 FROM table1

UNION

SELECT column1 FROM table2;

The UNION ALL operator is similar to UNION, but it includes all rows, including duplicates.

SELECT column1 FROM table1

UNION ALL

SELECT column1 FROM table2;

### 

### **Self Join**

A self join is a regular join but the table is joined with itself. It is useful when you want to combine rows from the same table based on a related column.

SELECT e1.employee\_name, e2.employee\_name

FROM employee e1

JOIN employee e2 ON e1.manager\_id = e2.employee\_id;

**-- Joins - inner join , left join , right join and full join**

**-- T1 t2**

**-- stid name age stid tid subject**

**-- 1 abc 12 2 3 maths**

**-- 2 bdc 23 4 3 maths**

**-- 3 ojn 25 5 4 science**

**-- 4 ljn 26**

**-- right join - stid**

**-- stid name age stid tid subject**

**-- 2 bdc 23 2 3 maths**

**-- 4 ljn 26 4 3 maths**

**-- nul nul nul 5 4 science**

**--**

select \* from actor;

select \* from actor\_award;

select \* from actor inner join actor\_award on actor.actor\_id = actor\_award.actor\_id;

### **Example Dataset**

We will use a simplified dataset with two tables: customers and orders.

#### **customers Table**

**CREATE TABLE customers (  
    customer\_id INT PRIMARY KEY,  
    customer\_name VARCHAR(100),  
    city VARCHAR(50)  
);  
  
INSERT INTO customers (customer\_id, customer\_name, city) VALUES  
(1, 'Alice Johnson', 'New York'),  
(2, 'Bob Smith', 'Los Angeles'),  
(3, 'Charlie Brown', 'Chicago');**

#### **orders Table**

**CREATE TABLE orders (  
    order\_id INT PRIMARY KEY,  
    customer\_id INT,  
    order\_date DATE,  
    order\_amount DECIMAL(10, 2),  
    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)  
);  
  
INSERT INTO orders (order\_id, customer\_id, order\_date, order\_amount) VALUES  
(1, 1, '2024-01-01', 100.00),  
(2, 2, '2024-01-02', 150.00),  
(3, 1, '2024-01-03', 200.00),  
(4, 3, '2024-01-04', 50.00);**

### **Use Cases**

#### **1. Inner Join: Show all orders with customer details**

**Question**: How can we list all orders along with the customer names who placed them?

**Use Case**: We want to see which customer placed each order.

**SELECT   
    o.order\_id,  
    o.order\_date,  
    o.order\_amount,  
    c.customer\_name,  
    c.city  
FROM   
    orders o  
INNER JOIN   
    customers c ON o.customer\_id = c.customer\_id;**

**Explanation**: This query joins the orders table with the customers table on the customer\_id field, showing details from both tables.

#### 2. **Left Join**: Show all customers and their orders, including those who haven't placed any orders

**Question**: How can we list all customers and show their orders if they have placed any?

**Use Case**: We want to identify which customers have not placed any orders yet.

**SELECT   
    c.customer\_id,  
    c.customer\_name,  
    o.order\_id,  
    o.order\_date,  
    o.order\_amount  
FROM   
    customers c  
LEFT JOIN   
    orders o ON c.customer\_id = o.customer\_id;**

**Explanation**: This query lists all customers and their corresponding orders. If a customer hasn't placed any orders, the order-related columns will be NULL.

#### 3. **Right Join**: Show all orders and their corresponding customers, including orders without customers (if any)

**Question**: How can we list all orders and include the customer details, even if there are orders without matching customer records?

**Use Case**: Although rare in a well-maintained database, this can help identify any discrepancies between orders and customer records.

**SELECT   
    o.order\_id,  
    o.order\_date,  
    o.order\_amount,  
    c.customer\_id,  
    c.customer\_name,  
    c.city  
FROM   
    orders o  
RIGHT JOIN   
    customers c ON o.customer\_id = c.customer\_id;**

**Explanation**: This query shows all orders and their corresponding customer details. If an order does not have a matching customer record, the customer-related columns will be NULL.

#### 4. **Self Join**: Show customers living in the same city

**Question**: How can we find pairs of customers who live in the same city?

**Use Case**: This helps in finding potential groups or pairs of customers based on their city.

**SELECT   
    c1.customer\_name AS customer1,  
    c2.customer\_name AS customer2,  
    c1.city  
FROM   
    customers c1  
INNER JOIN   
    customers c2 ON c1.city = c2.city AND c1.customer\_id <> c2.customer\_id;**

**Explanation**: This query joins the customers table with itself to find pairs of customers living in the same city. The condition c1.customer\_id <> c2.customer\_id ensures that we don't match a customer with themselves.

CREATE TABLE employees (  
 employee\_id INT PRIMARY KEY,  
 employee\_name VARCHAR(50),  
 manager\_id INT,  
 FOREIGN KEY (manager\_id) REFERENCES employees(employee\_id)  
);  
  
INSERT INTO employees VALUES (1, 'John', NULL);  
INSERT INTO employees VALUES (2, 'Jane', 1);  
INSERT INTO employees VALUES (3, 'Bob', 2);  
INSERT INTO employees VALUES (4, 'Alice', 1);  
INSERT INTO employees VALUES (5, 'Charlie', 3);  
  
select \* from employees;

### Example 1: MySQL SELF JOIN Using INNER JOIN Clause

SELECT e1.employee\_id AS employee\_id,   
 e1.employee\_name AS employee\_name,   
 e2.employee\_name AS manager\_name  
FROM employees e1  
INNER JOIN employees e2 ON e1.manager\_id = e2.employee\_id;

#### **5. Cross Join: Show all possible combinations of customers and orders**

**Question**: How can we generate all possible combinations of customers and orders?

**Use Case**: This might be used to create a comprehensive list for analysis or reporting.

**SELECT   
    c.customer\_name,  
    o.order\_id,  
    o.order\_date,  
    o.order\_amount  
FROM   
    customers c  
CROSS JOIN   
    orders o;**

**Explanation**: This query generates a Cartesian product of the customers and orders tables, resulting in every possible combination of customers and orders.

### **Summary**

* **Inner Join**: Finds matching rows in both tables.
* **Left Join**: Finds all rows from the left table and matching rows from the right table.
* **Right Join**: Finds all rows from the right table and matching rows from the left table.
* **Self Join**: Joins a table with itself to find related rows.
* **Cross Join**: Produces all possible combinations of rows from the joined tables.

how joins work and see practical examples of their use cases in a simple e-commerce context.

**Work on own dataset**

**-- Display all the cities with there postal code in India ?**

select \* from country; **-- country data**

select \* from city; **-- city data**

select \* from address; **-- postal codes**

SELECT

postal\_code, city, country

FROM

address

INNER JOIN

city ON address.city\_id = city.city\_id

INNER JOIN

country ON city.country\_id = country.country\_id

WHERE

country = 'India';

**-- Display the names of actors and the names of the films they have acted in.**

select \* from actor; **-- names of actor**

select \* from film\_actor; **-- common connection**

select \* from film; **-- title**

SELECT

concat(a.first\_name, " ", a.last\_name) as Name, title

FROM

actor a

INNER JOIN

film\_actor ON a.actor\_id = film\_actor.actor\_id

INNER JOIN

film ON film.film\_id = film\_actor.film\_id;

**-- Display all the actors and total number of films they have acted in.**

SELECT

concat(first\_name, " ", last\_name) as Name, count(title) as count\_of\_movies

FROM

actor

left JOIN

film\_actor ON actor.actor\_id = film\_actor.actor\_id

left JOIN

film ON film.film\_id = film\_actor.film\_id

group by Name

order by count\_of\_movies;

