**Joins in MySQL**

In relational databases, data is often distributed across multiple related tables. **Joins** in MySQL are used to combine rows from two or more tables based on a related column between them.

**Types of Joins in MySQL**

To understand joins better, consider the following two sample tables:

create database demojoin;

use demojoin;

CREATE TABLE departments (

id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(50),

dept\_id INT

);

INSERT INTO departments (id, dept\_name) VALUES

(101, 'HR'),

(102, 'Sales'),

(104, 'Marketing');

INSERT INTO employees (id, name, dept\_id) VALUES

(1, 'Alice', 101),

(2, 'Bob', 102),

(3, 'Charlie', NULL),

(4, 'David', 103);

**Table: employees**

| **id** | **name** | **dept\_id** |
| --- | --- | --- |
| 1 | Alice | 101 |
| 2 | Bob | 102 |
| 3 | Charlie | NULL |
| 4 | David | 103 |

**Table: departments**

| **id** | **dept\_name** |
| --- | --- |
| 101 | HR |
| 102 | Sales |
| 104 | Marketing |

**1. INNER JOIN**

**Definition:**  
Returns rows that have matching values in both tables.

**Syntax:**

SELECT column1, column2

FROM table1

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

**Example:**

SELECT employees.name, departments.dept\_name

FROM employees

INNER JOIN departments ON employees.dept\_id = departments.id;

**Output:**

| **name** | **dept\_name** |
| --- | --- |
| Alice | HR |
| Bob | Sales |

Only employees who belong to an existing department are listed.

**2. LEFT JOIN (LEFT OUTER JOIN)**

**Definition:**  
Returns all rows from the left table, and the matched rows from the right table. If no match exists, NULL is returned for right-side columns.

**Syntax:**

SELECT column1, column2

FROM table1

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

**Example:**

SELECT employees.name, departments.dept\_name

FROM employees

LEFT JOIN departments ON employees.dept\_id = departments.id;

**Output:**

| **name** | **dept\_name** |
| --- | --- |
| Alice | HR |
| Bob | Sales |
| Charlie | NULL |
| David | NULL |

This includes all employees, even those not assigned to a department.

**3. RIGHT JOIN (RIGHT OUTER JOIN)**

**Definition:**  
Returns all rows from the right table, and the matched rows from the left table. If no match exists, NULL is returned for left-side columns.

**Syntax:**

SELECT column1, column2

FROM table1

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

**Example:**

SELECT employees.name, departments.dept\_name

FROM employees

RIGHT JOIN departments ON employees.dept\_id = departments.id;

**Output:**

| **name** | **dept\_name** |
| --- | --- |
| Alice | HR |
| Bob | Sales |
| NULL | Marketing |

This includes all departments, even those not assigned to any employee.

**4. FULL OUTER JOIN *(Emulated in MySQL)***

**Definition:**  
Returns all rows from both tables. If there is no match, NULLs are returned for missing values.

**Note:** MySQL doesn’t support FULL JOIN directly. It can be emulated using UNION.

**Query:**

SELECT employees.name, departments.dept\_name

FROM employees

LEFT JOIN departments ON employees.dept\_id = departments.id

UNION

SELECT employees.name, departments.dept\_name

FROM employees

RIGHT JOIN departments ON employees.dept\_id = departments.id;

**Output:**

| **name** | **dept\_name** |
| --- | --- |
| Alice | HR |
| Bob | Sales |
| Charlie | NULL |
| David | NULL |
| NULL | Marketing |

Shows all employees and departments, including those without a match.

**5. SELF JOIN**

**Definition:**  
A table joined with itself. Useful for hierarchical or related data within the same table.

**Modified employees Table:**

| **id** | **name** | **dept\_id** | **manager\_id** |
| --- | --- | --- | --- |
| 1 | Alice | 101 | NULL |
| 2 | Bob | 102 | 1 |
| 3 | Charlie | NULL | 1 |
| 4 | David | 103 | 2 |

**Query:**

SELECT A.name AS Employee, B.name AS Manager

FROM employees A

JOIN employees B ON A.manager\_id = B.id;

**Output:**

| **Employee** | **Manager** |
| --- | --- |
| Bob | Alice |
| Charlie | Alice |
| David | Bob |

Illustrates reporting relationships within the same table.

**Cross Product (Cartesian Join) in MySQL**

**Definition:**  
A **cross join** returns the Cartesian product of the two tables – each row of the first table is paired with every row of the second.

**Syntax:**

SELECT \* FROM table1

CROSS JOIN table2;

**Or simply:**

SELECT \* FROM table1, table2;

**Example:**

SELECT employees.name, departments.dept\_name

FROM employees

CROSS JOIN departments;

**Output (partial):**

| **name** | **dept\_name** |
| --- | --- |
| Alice | HR |
| Alice | Sales |
| Alice | Marketing |
| Bob | HR |
| Bob | Sales |
| ... | ... |

If employees has 4 rows and departments has 3 rows, the result will have **4 × 3 = 12** rows.

**Summary Table**

| **Join Type** | **Description** |
| --- | --- |
| INNER JOIN | Only rows with matching values in both tables |
| LEFT JOIN | All rows from left + matched rows from right |
| RIGHT JOIN | All rows from right + matched rows from left |
| FULL JOIN | All rows from both sides, with NULLs for unmatched |
| SELF JOIN | Join table with itself |
| CROSS JOIN | Cartesian product (every row with every other row) |