Concept of Joins in MySQL

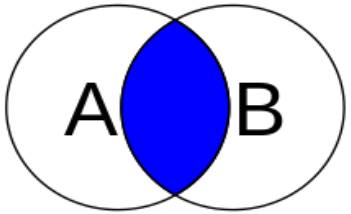
A **Join** in MySQL is used to combine rows from two or more tables based on a related column between them. It helps retrieve data spread across multiple tables.

Types of Joins:

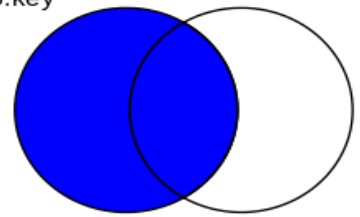
1. **INNER JOIN**
2. **LEFT JOIN (LEFT OUTER JOIN)**
3. **RIGHT JOIN (RIGHT OUTER JOIN)**
4. **FULL JOIN (FULL OUTER JOIN)**
5. **CROSS JOIN**
6. **SELF JOIN**
7. **NATURAL JOIN**

SQL JOINS

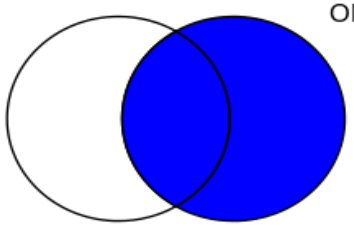
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key



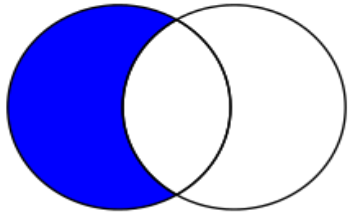
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key

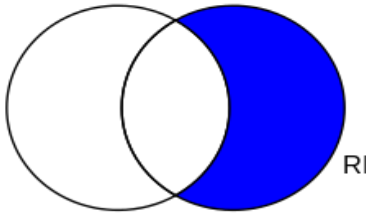


SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key

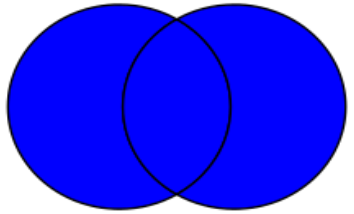


SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL

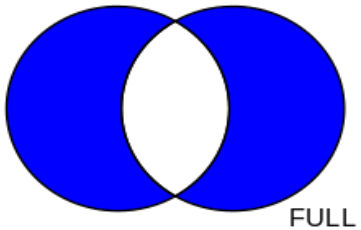




SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.kev = B.kev



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL

Example Tables and Data

1. Employees Table

EmployeeID	Name	DepartmentID	Salary
1	John Doe	101	60000
2	Jane Smith	102	65000
3	Alice Lee	103	70000
4	Bob Brown	NULL	50000

2. Departments Table

DepartmentID	DepartmentName	Location
101	IT	New York
102	HR	Chicago
103	Finance	Los Angeles

3. Projects Table

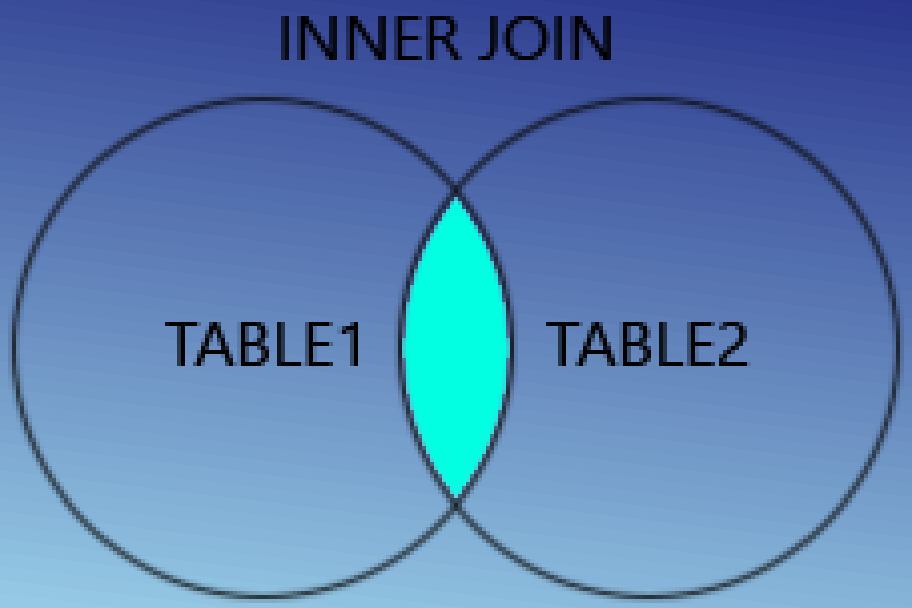
ProjectID	EmployeeID	ProjectName
1	1	AI Development
2	2	Recruitment Portal
3	1	Data Analysis

Inner Join

An **INNER JOIN** is a type of join in SQL that combines rows from two or more tables based on a related column between them. It returns only those rows where there is a match in both tables.

Syntax:

```
SELECT column1, column2, ... FROM table1
 INNER JOIN table2
 ON table1.common_column = table2.common_column;
```



1. Basic Inner Join

•**Scenario**: Combining data from two tables where there is a direct relationship.
•**Example**: **Tables**:
  •Employees: Contains employee details.
  •Departments: Contains department details.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

2. Inner Join with Multiple Tables

•**Scenario**: Joining more than two tables.

•**Example**: **Tables**: Orders, Customers, Products.

```
SELECT Orders.OrderID, Customers.CustomerName, Products.ProductName
FROM Orders INNER JOIN Customers
ON Orders.CustomerID = Customers.CustomerID
INNER JOIN Products ON Orders.ProductID = Products.ProductID;
```

3. Inner Join with Aliases

•**Scenario**: Using table aliases to make the query more readable.

•**Example**:

```
SELECT e.EmployeeID, e.Name, d.DepartmentName FROM Employees e INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

4. Inner Join with Filtering

•**Scenario**: Adding a WHERE clause to filter results.

•**Example**:
sql

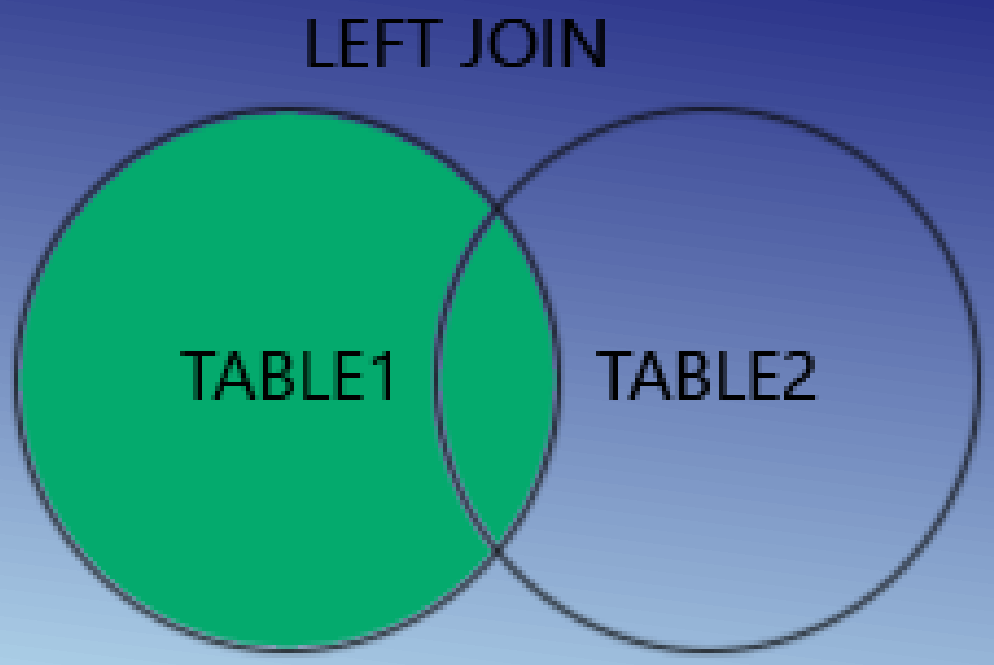
```
SELECT o.OrderID, c.CustomerName, p.ProductName FROM Orders o INNER JOIN Customers c ON o.CustomerID = c.CustomerID INNER JOIN Products p ON o.ProductID = p.ProductID WHERE p.Price > 100;
```

LEFT JOIN

A **LEFT JOIN** in SQL returns all records from the left table (Table A) and the matched records from the right table (Table B). If there is no match, the result is NULL on the side of Table B.

General Syntax

SELECT columns FROM TableA LEFT JOIN TableB ON TableA.common_column = TableB.common_column;



1. Basic Example:

Scenario: A company has two tables - Employees and Departments. Some employees are not assigned to any department yet.

```
SELECT e.Name, d.DepartmentName FROM Employees e LEFT JOIN Departments d
ON e.DepartmentID = d.DepartmentID;
```

2. Filtering Unmatched Records:

Scenario: Find all employees who are not assigned to any department.
Query:

```
SELECT e.Name FROM Employees e LEFT JOIN Departments d ON e.DepartmentID =
d.DepartmentID WHERE d.DepartmentID IS NULL;
```

3. Combining Data from Multiple Tables:

Scenario: A store has Orders and Customers. We want to list all customers, even those who haven't placed an order yet.

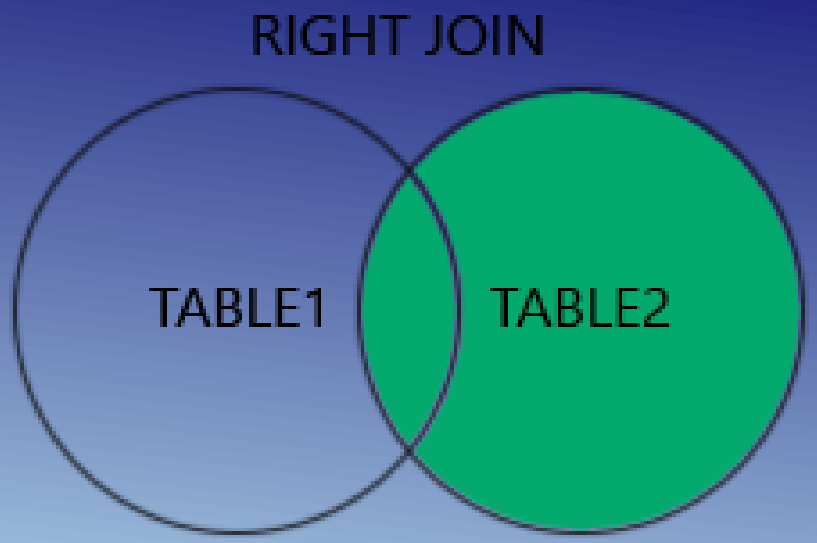
```
SELECT c.CustomerName, o.Amount FROM Customers c LEFT JOIN Orders o ON
c.CustomerID = o.CustomerID;
```

Right Join in SQL

A **Right Join** (or **Right Outer Join**) is a type of SQL join that returns all records from the **right table** and the matched records from the **left table**. If no match is found, NULL values are included for the unmatched records in the **left table**.

Syntax

```
SELECT columns FROM table1 RIGHT JOIN table2 ON table1.common_column = table2.common_column;
```



Retrieve all data from the right table even if there is no match in the left table

Example: List all employees from a department, even if some employees are not assigned any projects.

```
SELECT employees.name, projects.project_name FROM projects RIGHT JOIN employees ON
projects.employee_id = employees.id;
```

Combining sales data with a product catalog

Example: Get a list of all products, including those with no sales.

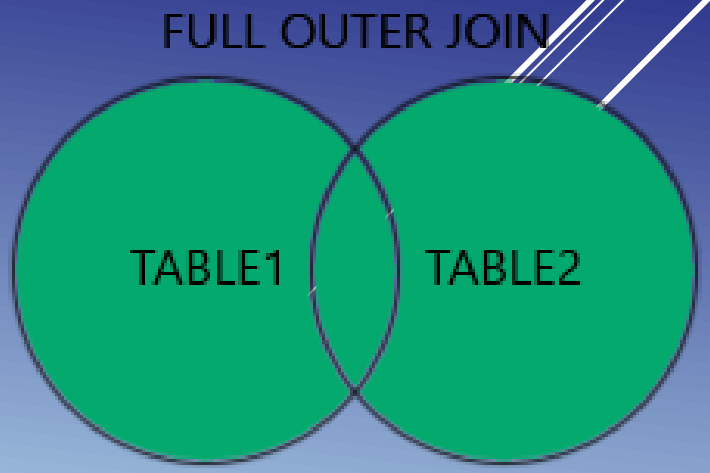
```
SELECT products.product_name, sales.quantity FROM sales RIGHT JOIN products ON
sales.product_id = products.id;
```

Full Outer Join

A **Full Outer Join** is a type of join in SQL that combines the results of both **LEFT JOIN** and **RIGHT JOIN**. It returns all records when there is a match in either the left table or the right table. If there is no match, the result contains NULL for every column of the table that lacks a matching row.

Syntax

```
SELECT *FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name
UNION
SELECT *
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```



Scenario 1: Retrieve All Employees and Projects (with unmatched data)

•**Tables**:

- •Employees (list of employees)
- •Projects (list of projects assigned)

•**Query**:

```
SELECT * FROM Employees Left JOIN Projects ON Employees.EmployeeID = Projects.EmployeeID
Union
SELECT * FROM Employees Ritht  JOIN Projects ON Employees.EmployeeID = Projects.EmployeeID ;
```

•**Result**:All employees, whether they have a project assigned or not.

•All projects, even if no employee is assigned.

Scenario 2: Combining Two Customer Databases

•Merging customer data from two different regions.

•**Query**:

```
SELECT Region1.CustomerID, Region1.Name, Region2.Name AS Region2Name FROM Region1 left JOIN
Region2 ON Region1.CustomerID = Region2.CustomerID
union
SELECT Region1.CustomerID, Region1.Name, Region2.Name AS Region2Name FROM Region1 LEFT JOIN
Region2 ON Region1.CustomerID = Region2.CustomerID;
```

Result:

•A list of all customers from both regions, showing NULL for missing information.

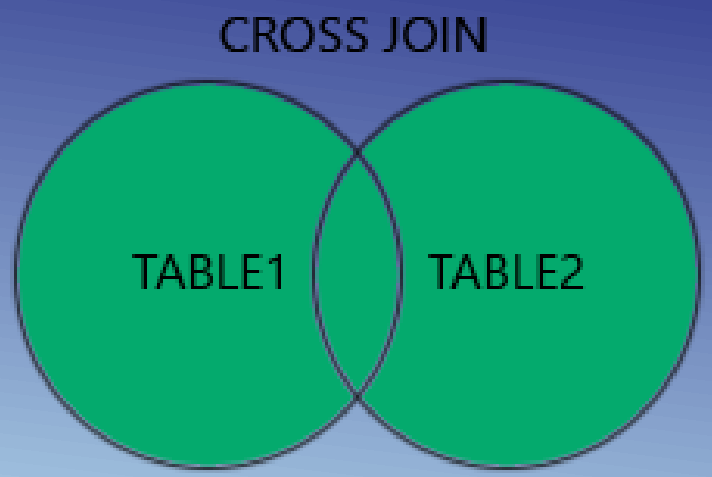
CROSS JOIN

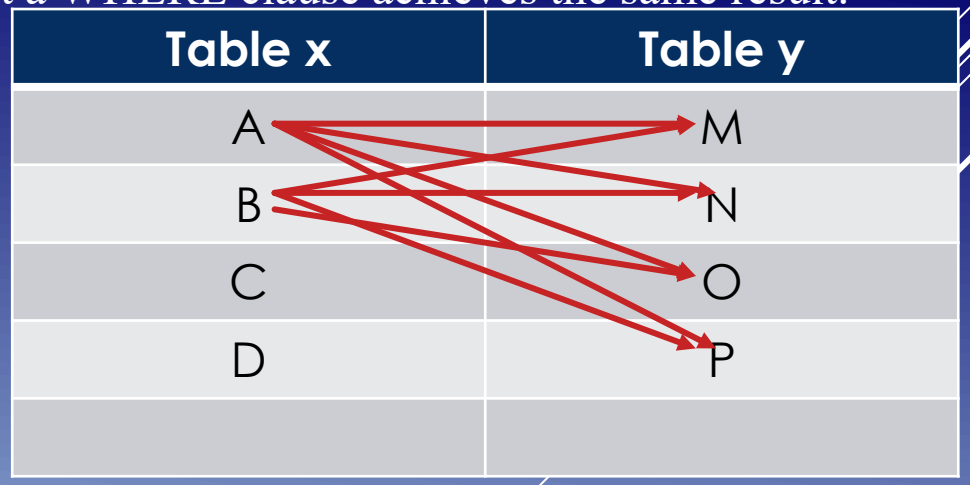
A **CROSS JOIN** in SQL is a type of join that returns the Cartesian product of two tables. It pairs each row from the first table with every row from the second table, resulting in a combination of all possible row pairs.

Syntax :

```
SELECT * FROM table1 CROSS JOIN table2;
```

If CROSS JOIN is omitted, using a simple FROM table1, table2 without a WHERE clause achieves the same result.





When to Use CROSS JOIN

1.Generating Test Data

You can create combinations of datasets for testing scenarios or generating lookup tables.

2.Combination Scenarios

Use CROSS JOIN to find all possible combinations of two data sets.

3.Dynamic Pivoting or Aggregation

CROSS JOIN helps create grid-like data structures for reports.

4.Comparison Across Groups

Compare elements of one set with another, such as sales regions against product categories.

Scenario 1: Generating All Possible Combinations

Imagine you have two tables: Products and Colors.

- **Products Table**
- **Colors Table**

ProductID	ProductName
1	Shirt
2	Pants

ColorID	ColorName
1	Red
2	Blue

Query:

```
SELECT Products.ProductName, Colors.ColorName FROM Products CROSS JOIN Colors;
```

Result:

ProductName	ColorName
Shirt	Red
Shirt	Blue
Pants	Red
Pants	Blue

P_ID	P_NAME
1	SHIRT
2	JINS
3	SUIT

C_ID	C_NAME
101	RED
102	BLUE
103	BLACK

Scenario 2: Creating a Calendar Table

If you have a table of months and another table of years, you can use a CROSS JOIN to create a full calendar.

- **Months Table**
- **Years Table**

Month	Year
Jan	2024
Feb	2025

Query:

```
SELECT Months.Month, Years.Year FROM Months CROSS JOIN Years;
```

Result:

Month	Year
Jan	2024
Jan	2025
Feb	2024
Feb	2025

Self Join

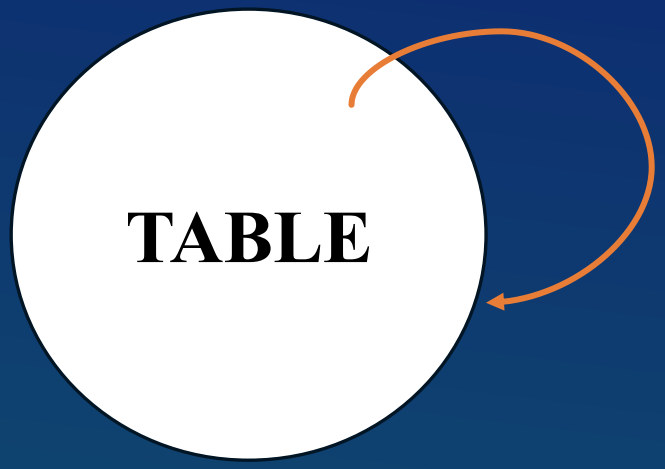
A **Self Join** is a join in which a table is joined with itself. It is used to compare rows within the same table. In a self join, the same table is used twice, but we treat one instance as the "left table" and the other as the "right table" by using table aliases.

Use Cases of Self Join

1.Finding relationships within a hierarchical structure (e.g., managers and employees).

2.Comparing rows within the same table (e.g., employees with the same department or salary).

3.Identifying duplicates or related data in the table.



Employee
A -B
B - D
C- A
D - A

Example
Table: Employees

EmployeeID	Name	ManagerID
1	John Doe	~~NULL~~
2	Jane Smith	1
3	Alice Lee	1
4	Bob Brown	2
5	Charlie Green	2

Self Join Query: Find Employee-Manager Relationships

```
SELECT e1.Name AS Employee, e2.Name AS Manager
FROM Employees e1 LEFT JOIN Employees e2
ON e1.ManagerID = e2.EmployeeID;
```

Output:

Employee	Manager
John Doe	NULL
Jane Smith	John Doe
Alice Lee	John Doe
Bob Brown	Jane Smith
Charlie Green	Jane Smith

Explanation

1.Employees e1: Represents the "employee" instance of the table.

2.Employees e2: Represents the "manager" instance of the table.

3.ON e1.ManagerID = e2.EmployeeID: Matches the ManagerID in the "employee" instance with the EmployeeID in the "manager" instance.

ProductID	ProductName
1	Shirt
2 3	Pants jins

ColorID	ColorName	P_id
1	Red	1
2 3	Blue black	2 4

Select * from product ,color where product.product_id = color.P_id

ProductID	ProductName	ColorID	ColorName	P_id
1	Shirt	1	red	1
2	pants	2	blue	2
3				
				4

1. UNION

- **Purpose**: Combines the results of two or more queries and removes duplicate rows from the final result.
- **Key Property**: Ensures the result set contains only distinct rows.
- **Syntax**:

```
SELECT column1, column2 FROM table1 UNION SELECT column1, column2 FROM table2;
```

Example of UNION
Tables:

Table 1: Employees_US

EmployeeID	Name
2	Jane Smith
3	Alice Lee

Table 2: Employees_EU

EmployeeID	Name
1	John Doe
2	Jane Smith

```
SELECT Name FROM Employees_US
UNION
SELECT Name FROM Employees_EU;
```

2. UNION ALL

•**Purpose**: Combines the results of two or more queries, including duplicate rows.

•**Key Property**: Does not remove duplicate rows.

•**Syntax**:

sql

```
SELECT column1, column2 FROM table1 UNION ALL SELECT column1, column2 FROM table2;
```

Example of UNION ALL

Tables:

Table 1: Employees_US

EmployeeID	Name
1	John Doe
2	Jane Smith

Table 2: Employees_EU

EmployeeID	Name
2	Jane Smith
3	Alice Lee

Query:

```
SELECT Name FROM Employees_US UNION ALL SELECT Name FROM Employees_EU;
```

Key Differences Between UNION and UNION ALL

Feature	UNION	UNION ALL
Duplicates Handling	Removes duplicates	Includes duplicates
Performance	Slower (removes duplicates)	Faster (no duplicate check)
Use Case	When unique records are needed	When all records (including duplicates) are needed

Practical Use Cases

Using UNION:

- To merge datasets from two sources while removing duplicate entries.
- Example: Combine customer data from two regions and remove duplicates.

Using UNION ALL:

- To merge datasets while retaining all entries, including duplicates.
- Example: Combine all transaction records for audit purposes.

---The End---