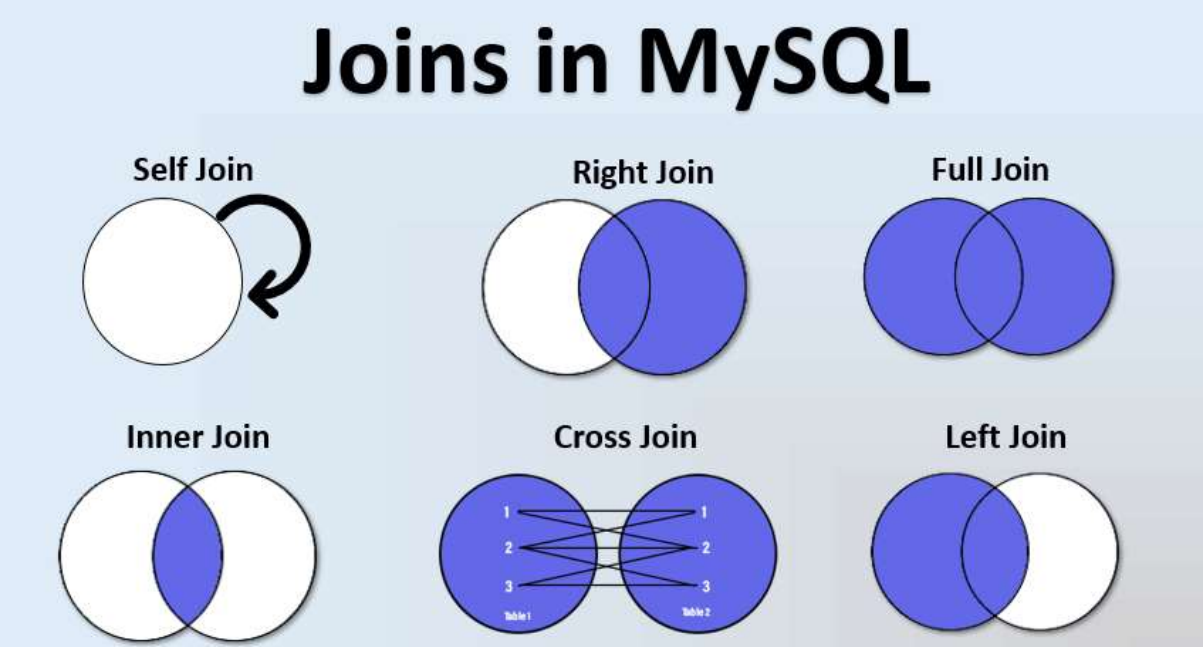
Query Multiple Tables	• Introduction to joins • Types of joins • Inner Join • Left Outer Join • Right Outer Join • Full outer Join • ANSI Join Syntax • Self-Join • Equi and non-equi Join • Set Operations

Introduction to Joins

In SQL, a join is used to combine rows from two or more tables, based on a related column between them. The main purpose of joins is to retrieve data from multiple tables in a single query.

Types of Joins

1. **Inner Join**
2. **Left Outer Join**
3. **Right Outer Join**
4. **Full Outer Join**
5. **Self-Join**
6. **Equi and Non-equi Join**
7. **Set Operations**



Inner Join

An **Inner Join** returns only the rows that have matching values in both tables. If there are rows in one table that do not have matches in the other table, those rows will not be included in the result set.

Syntax:

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

Departments Table

dept_id	dept_name
1	IT
2	Project Management
3	Human Resources
4	Sales
5	Marketing

Employees Table

emp_id	emp_name	job_title	department_id	salary
1	Rahul Sharma	Software Engineer	1	60000.00
2	Priya Verma	Software Engineer	1	65000.00
3	Amit Patel	Project Manager	2	75000.00
4	Sunita Singh	HR Manager	3	55000.00
5	Neha Gupta	Software Engineer	1	62000.00
6	Ravi Kumar	Project Manager	2	77000.00
7	Meera Nair	HR Manager	3	57000.00
8	Arjun Reddy	Sales Executive	4	50000.00
9	Sneha Iyer	Sales Executive	4	51000.00
10	Anjali Desai	HR Specialist	3	52000.00

Example:

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.department_id;
```

Left Outer Join

A **Left Outer Join** (or simply Left Join) returns all rows from the left table, and the matched rows from the right table. If no match is found, NULL values are returned for columns from the right table.

In the context of SQL joins, the "left table" is the table that appears first in the FROM clause, and the "right table" is the table that appears after the JOIN keyword.

Syntax:

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

In this syntax:

- table1 is the left table.
- table2 is the right table.

Example:

```
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.department_id;
```

In this example:

- employees is the left table.
- departments is the right table.

Right Outer Join

A **Right Outer Join** (or simply Right Join) returns all rows from the right table, and the matched rows from the left table. If no match is found, NULL values are returned for columns from the left table.

In the context of SQL joins, the "right table" is the table that appears after the JOIN keyword. For a RIGHT JOIN, it means that all rows from the right table will be included in the result set, along with the matched rows from the left table. If there is no match, the result will include NULLs for columns from the left table.

Syntax:

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

In this syntax:

- table1 is the left table.
- table2 is the right table.

Example:

```
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.department_id;
```

In this example:
employees is the left table.
departments is the right table.

Full Outer Join

A **Full Outer Join** returns all rows when there is a match in either left or right table. This means it returns all rows from both tables, with NULLs in place where the join condition is not met.

Syntax: MySQL does not directly support Full Outer Join. However, it can be simulated using a combination of Left Join and Right Join with UNION.

Example:

```
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments ON employees.department_id = departments.department_id
```

```
UNION
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments ON employees.department_id = departments.department_id;
```

ANSI Join Syntax

ANSI Join Syntax is a standardized SQL join syntax. It is used to ensure compatibility across different SQL database systems.

Syntax:

```
SELECT columns
FROM table1
JOIN_TYPE table2
ON table1.common_column = table2.common_column;
```

Example:

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id = departments.department_id;
```

Self-Join

A **Self-Join** is a regular join but the table is joined with itself.

Syntax:

```
SELECT a.columns, b.columns
FROM table_name a, table_name b
WHERE condition;
```

Example:

```
SELECT e1.emp_id AS emp1_id, e1.emp_name AS emp1_name, e2.emp_id AS emp2_id, e2.emp_name AS emp2_name, e1.department_id
```

```
FROM employees e1

JOIN employees e2 ON e1.department_id = e2.department_id

WHERE e1.emp_id != e2.emp_id;
```

Equi and Non-equi Join

Equi Join uses the equality operator (=) to match rows between tables. It can be an Inner Join, Left Join, etc.

Syntax:

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

Example:

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id = departments.department_id;
```

Non-equi Join uses other comparison operators like >, <, >=, <=, etc.

Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.common_column > table2.common_column;
```

Example:

```
SELECT employees.name, salaries.salary
FROM employees
```

INNER JOIN salaries ON employees.employee_id > salaries.employee_id;

Set Operations

Set Operations are used to combine the results of two or more queries. The main set operations are UNION, UNION ALL, INTERSECT, and EXCEPT.

1. **UNION**: Combines the results of two queries and removes duplicates.
2. **UNION ALL**: Combines the results of two queries without removing duplicates.
3. **INTERSECT**: Returns only the rows that are present in both queries.
4. **EXCEPT**: Returns the rows from the first query that are not present in the second query.

Syntax:

```
SELECT columns FROM table1
UNION
SELECT columns FROM table2;

SELECT columns FROM table1
UNION ALL
SELECT columns FROM table2;

SELECT columns FROM table1
INTERSECT
SELECT columns FROM table2;

SELECT columns FROM table1
EXCEPT
SELECT columns FROM table2;
```

Examples:

UNION:

```
SELECT name FROM employees
```

```
UNION
SELECT name FROM managers;
```

UNION ALL:

```
SELECT name FROM employees
UNION ALL
SELECT name FROM managers;
```

INTERSECT and EXCEPT are not directly supported in MySQL, but can be simulated using JOINs and subqueries.

Conclusion

Joins and set operations are powerful tools in SQL for retrieving and combining data from multiple tables. Understanding these concepts is essential for performing complex queries and extracting meaningful information from a relational database