**INNER JOIN (Detailed Process)**

The **INNER JOIN** works like a strict matchmaker between two tables.

1. **Selection:** You tell the query which two tables you want to join (e.g., Customers and Orders).
2. **Matching Rule (The ON Clause):** You provide the rule (ON) for how to connect the rows, which must be a matching column in both tables (e.g., CustomerID).
3. **Result:** SQL checks every row in the first table against every row in the second table. It only keeps and combines the rows where the values in the matching columns are **exactly the same**.

Any customer who has **no orders** is completely dropped. Any order that has **no matching customer** is also completely dropped. The result is a new table containing only the intersection of the two original tables.

**Syntax**

SELECT column1, column2

FROM TableA

INNER JOIN TableB ON TableA.MatchingID = TableB.MatchingID;

**Basic Example Query**

To show the name of every customer and the date of their orders, but only where a matching order actually exists:

SELECT Customers.Name, Orders.OrderDate

FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

**LEFT JOIN**

The **LEFT JOIN** is a type of outer join that is **"left-biased,"** meaning it makes sure to include every single row from the **left table** (the first table you mention in the query).

**How it works:**

1. It starts with all records from the left table.
2. It then finds all matching records from the right table based on your matching condition (ON clause) and combines them.
3. If a row in the left table has **no match** in the right table, the columns from the right table are filled with **NULL**. This is useful for finding records that are missing related data.

**Syntax**

SELECT A.column1, B.column2

FROM TableA AS A

LEFT JOIN TableB AS B ON A.MatchingID = B.MatchingID;

**Basic Example Query**

To list **every customer** in the Customers table, and show their OrderDate only if they have an order. Customers without orders will have NULL in the OrderDate column.

SELECT Customers.Name, Orders.OrderDate

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

**RIGHT JOIN**

The **RIGHT JOIN** is the opposite of the LEFT JOIN; it is **"right-biased,"** meaning it guarantees you get every single row from the **right table** (the second table you mention in the query).

**How it works:**

1. It starts with all records from the right table.
2. It then looks for matching records in the left table and combines them based on your condition (ON clause).
3. If a row in the right table has **no match** in the left table, the columns from the left table are filled with **NULL**. This is useful for finding orphaned records in the right table.

**Syntax**

SELECT A.column1, B.column2

FROM TableA AS A

RIGHT JOIN TableB AS B ON A.MatchingID = B.MatchingID;

**Basic Example Query**

To list **every product** from the Products table, and show the supplier's name only if a matching supplier exists. Products without a supplier will show NULL for the supplier's name.

SELECT Suppliers.Name, Products.ProductName

FROM Suppliers

RIGHT JOIN Products ON Suppliers.SupplierID = Products.SupplierID;

**Full Join in MySQL**

A **FULL JOIN** (or FULL OUTER JOIN) is the most inclusive type of join. It returns **all rows** from **both** the left and the right table, filling in NULLs for any missing matches on either side. It's the opposite of an INNER JOIN.

Important Note for MySQL:

MySQL is one of the few major databases that does not have a native FULL JOIN keyword. To get the same result, you must combine a LEFT JOIN and a RIGHT JOIN using the UNION operator.

**How the workaround works:**

1. **LEFT JOIN** gets all rows from the Left table + matches from the Right table.
2. **UNION** stacks the two result sets on top of each other and removes any duplicate rows (which are the matching rows found by both joins).
3. **RIGHT JOIN** gets all rows from the Right table + matches from the Left table.

**Syntax (The MySQL Workaround)**

SELECT columns\_list

FROM TableA

LEFT JOIN TableB ON TableA.MatchingID = TableB.MatchingID

UNION

SELECT columns\_list

FROM TableA

RIGHT JOIN TableB ON TableA.MatchingID = TableB.MatchingID;

**Basic Example Query**

To list **every customer** and **every order**, regardless of whether they have a match:

SELECT \*

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

UNION

SELECT \*

FROM Customers

RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

**UNION and UNION ALL**

UNION and UNION ALL are used to **stack the result sets** of two or more separate SELECT queries on top of each other into one long result list. This is useful for combining similar data that is stored in different tables.

**Key Requirement:** For both to work, every SELECT query must have the **exact same number of columns**, and those columns must be in the **same order** with compatible data types.

| Operator | What it does | Duplicates? | Performance |
| --- | --- | --- | --- |
| **UNION** | Combines results and **removes duplicate rows**. | Only unique rows are kept. | Slower, because SQL has to check for and remove duplicates. |
| **UNION ALL** | Combines results and **keeps all rows**. | Includes every row, even if it's an exact duplicate. | Faster, because SQL just stacks the results without checking. |

**Syntax (Applies to both)**

SELECT column1, column2 FROM TableA

UNION [ALL]

SELECT column1, column2 FROM TableB;

**Basic Example Query**

To get a complete, de-duplicated list of names from two different employee tables:

SELECT FirstName, LastName FROM SalesTeam

UNION

SELECT FirstName, LastName FROM MarketingTeam

ORDER BY LastName;