-- Customers Table

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

age INT,

country VARCHAR(100)

);

-- Orders Table

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

item VARCHAR(100),

amount INT,

customer\_id INT, -- common column with Customers

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

-- Shippings Table

CREATE TABLE Shippings (

shipping\_id INT PRIMARY KEY,

status VARCHAR(50),

customer INT -- common but column name differs from Customers.customer\_id

);

-- Customers

**INSERT INTO Customers (customer\_id, first\_name, last\_name, age, country) VALUES**

(1, 'Aarav', 'Mehta', 28, 'India'),

(2, 'Riya', 'Sharma', 32, 'India'),

(3, 'John', 'Doe', 40, 'USA'),

(4, 'Emma', 'Brown', 29, 'UK');

-- Orders

INSERT INTO Orders (order\_id, item, amount, customer\_id) VALUES

(101, 'Laptop', 60000, 1),

(102, 'Mobile', 20000, 1),

(103, 'Tablet', 30000, 2),

(104, 'Camera', 25000, 3);

-- Shippings

INSERT INTO Shippings (shipping\_id, status, customer) VALUES

(201, 'Delivered', 1),

(202, 'Pending', 2),

(203, 'In Transit', 5); -- Note: 5 doesn't exist in Customers (mismatch case)

**Step 3: Typical Join Scenarios**

**1. INNER JOIN with single common column**

(Only returns rows where there is a match in both tables)

SELECT c.first\_name, c.last\_name, o.item, o.amount

FROM Customers c

INNER JOIN Orders o

ON c.customer\_id = o.customer\_id;

👉 Picks records where a **customer has placed an order**.

**2. INNER JOIN with two common columns**

Suppose we had both customer\_id and country as common keys:

SELECT c.first\_name, c.country, o.item

FROM Customers c

INNER JOIN Orders o

ON c.customer\_id = o.customer\_id

AND c.country = 'India';

👉 Here we filter by both customer\_id and an extra condition (country).  
This is useful when multiple columns define uniqueness.

**3. JOIN when column names differ**

(Customers.customer\_id vs Shippings.customer)

SELECT c.first\_name, s.status

FROM Customers c

INNER JOIN Shippings s

ON c.customer\_id = s.customer;

👉 Even though the column names differ, the values represent the same entity.

**4. LEFT JOIN (keep all from left table, even unmatched)**

SELECT c.first\_name, o.item, o.amount

FROM Customers c

LEFT JOIN Orders o

ON c.customer\_id = o.customer\_id;

👉 Customers with **no orders** will still appear with NULL in order fields.

**5. RIGHT JOIN (keep all from right table, even unmatched)**

SELECT c.first\_name, o.item, o.amount

FROM Customers c

RIGHT JOIN Orders o

ON c.customer\_id = o.customer\_id;

👉 All orders are shown, even if **no matching customer exists**.

**6. FULL OUTER JOIN (all records from both sides)**

(Some DBs like MySQL don’t support directly; need UNION)

SELECT c.first\_name, o.item, o.amount

FROM Customers c

FULL OUTER JOIN Orders o

ON c.customer\_id = o.customer\_id;

Shows **all customers + all orders**, matched or not.

**7. SELF JOIN (joining table to itself)**

E.g., finding customers from the same country:

SELECT c1.first\_name AS Customer1, c2.first\_name AS Customer2, c1.country

FROM Customers c1

INNER JOIN Customers c2

ON c1.country = c2.country

AND c1.customer\_id <> c2.customer\_id;

**8. CROSS JOIN (Cartesian product)**

SELECT c.first\_name, o.item

FROM Customers c

CROSS JOIN Orders o;

Produces **all possible combinations** of customers and orders.

**9. USING keyword (when columns have the same name)**

SELECT first\_name, last\_name, item

FROM Customers

INNER JOIN Orders USING (customer\_id);

**10. JOIN with WHERE vs HAVING**

* WHERE filters before grouping
* HAVING filters after grouping

-- Using WHERE

SELECT c.country, COUNT(o.order\_id) AS total\_orders

FROM Customers c

JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE c.country = 'India'

GROUP BY c.country;

**-- Using HAVING**

SELECT c.country, COUNT(o.order\_id) AS total\_orders

FROM Customers c

JOIN Orders o ON c.customer\_id = o.customer\_id

GROUP BY c.country

HAVING COUNT(o.order\_id) > 1;

**NATURAL JOIN →**

* Automatically joins by all columns with same name.
* SELECT \* FROM Customers NATURAL JOIN Orders;

⚠️ Risky in real projects because it matches all identically named columns (even unintended ones).

* **SEMI JOIN** → Returns rows from left table if a match exists in right, but doesn’t bring right-side columns.  
  Often written with EXISTS instead of a keyword:
* SELECT \*
* FROM Customers c
* WHERE EXISTS (SELECT 1 FROM Orders o WHERE c.customer\_id = o.customer\_id);
* **ANTI JOIN** → Opposite of semi join: rows in left **without matches** in right.
* SELECT \*
* FROM Customers c
* WHERE NOT EXISTS (SELECT 1 FROM Orders o WHERE c.customer\_id = o.customer\_id);

**🔑 So, to summarize:**

* The **main family** of joins = INNER, LEFT, RIGHT, FULL, CROSS, SELF.
* Variants = NATURAL JOIN, USING, multiple/different column names.
* Logical patterns = SEMI JOIN, ANTI JOIN (using EXISTS / NOT EXISTS).