Join types explanation

Perfect — let's go **straight to the point**.
We'll use two sample tables:

🧾 Table 1: `Customers`

customer_id	customer_name
1	Alice
2	Bob
3	Charlie
4	David

📦 Table 2: `Orders`

order_id	customer_id	product
101	1	Laptop
102	1	Phone
103	2	Tablet
104	5	Keyboard

Notice:

- `customer_id = 5` in `Orders` has **no matching customer**.
- `customer_id = 3` and `4` in `Customers` have **no orders**.

🔹 1. INNER JOIN

👉 Returns **only the records that have matching values in both tables.**

```sql
SELECT Customers.customer_name, Orders.product
FROM Customers
INNER JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

✅ Result:

customer_name	product
Alice	Laptop
Alice	Phone
Bob	Tablet

Explanation:

- Only customers who have at least one order appear.

- **Charlie** and **David** are excluded (no order).
- **order_id 104** (Keyboard) is excluded (customer_id 5 doesn't exist in Customers).

🔹 2. LEFT JOIN (a.k.a LEFT OUTER JOIN)

👉 Returns **all customers** (left table) and the **matching orders**, if any.
If no matching order exists, it returns **NULL**.

```sql
SELECT Customers.customer_name, Orders.product
FROM Customers
LEFT JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

✅ **Result:**

customer_name	product
Alice	Laptop
Alice	Phone
Bob	Tablet
Charlie	NULL
David	NULL

Explanation:

- All customers are shown.
- Customers without orders (Charlie, David) get **NULL** for **product**.

🔹 3. RIGHT JOIN (a.k.a RIGHT OUTER JOIN)

👉 Returns **all orders** (right table) and the **matching customers**, if any.
If no matching customer exists, returns **NULL**.

```sql
SELECT Customers.customer_name, Orders.product
FROM Customers
RIGHT JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

✅ **Result:**

customer_name	product
Alice	Laptop
Alice	Phone
Bob	Tablet
NULL	Keyboard

Explanation:

- All orders are shown.
- The order with `customer_id = 5` (Keyboard) has no matching customer, so customer_name = `NULL`.

🔹 4. FULL JOIN (a.k.a FULL OUTER JOIN)

👉 Returns **all records** from both tables.
If there's no match, the missing side will have `NULL`.

sql

```sql
SELECT Customers.customer_name, Orders.product
FROM Customers
FULL OUTER JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

✅ **Result:**

customer_name	product
Alice	Laptop
Alice	Phone
Bob	Tablet
Charlie	NULL
David	NULL
NULL	Keyboard

Explanation:

- Combines results of LEFT and RIGHT JOIN.
- Shows all customers and all orders, even if they don't match.

🧠 Quick Summary Table

Join Type	Includes Unmatched Left?	Includes Unmatched Right?	Example
INNER JOIN	❌	❌	Only customers with orders
LEFT JOIN	✅	❌	All customers, with or without orders
RIGHT JOIN	❌	✅	All orders, even if customer missing
FULL JOIN	✅	✅	All customers + all orders

Would you like me to visualize this with a **Venn diagram style explanation** next (so you can see overlaps clearly)?

Alright — let's make **GROUP BY** and **HAVING** crystal clear.
We'll again use the same two tables 👇

🧾 `Customers`

customer_id	customer_name
1	Alice
2	Bob
3	Charlie

📦 `Orders`

order_id	customer_id	product	amount
101	1	Laptop	80000
102	1	Phone	30000
103	2	Tablet	20000
104	2	Mouse	5000
105	2	Keyboard	7000
106	3	Charger	1000

🔹 1. `GROUP BY` — basic concept

`GROUP BY` is used to **group rows that have the same values** in specific columns and perform **aggregate functions** like `SUM()`, `COUNT()`, `AVG()`, `MAX()`, etc.

🧩 Example 1: Total amount spent by each customer

sql

```sql
SELECT customer_id, SUM(amount) AS total_spent
FROM Orders
GROUP BY customer_id;
```

✅ **Result:**

customer_id	total_spent
1	110000
2	32000
3	1000

Explanation:

- All orders for each `customer_id` are grouped together.
- SUM gives the total purchase per customer.

🧩 Example 2: Number of orders per customer

```sql
SELECT customer_id, COUNT(order_id) AS total_orders
FROM Orders
GROUP BY customer_id;
```

✅ Result:

customer_id	total_orders
1	2
2	3
3	1

🧩 Example 3: Maximum order amount per customer

```sql
SELECT customer_id, MAX(amount) AS max_order
FROM Orders
GROUP BY customer_id;
```

✅ Result:

customer_id	max_order
1	80000
2	20000
3	1000

🔹 2. `HAVING` — filtering after grouping

- `WHERE` filters **rows before grouping**.
- `HAVING` filters **groups after aggregation**.

🧩 Example 4: Customers who spent more than 20000 total

```sql
SELECT customer_id, SUM(amount) AS total_spent
FROM Orders
GROUP BY customer_id
HAVING SUM(amount) > 20000;
```

✅ Result:

customer_id	total_spent
1	110000

customer_id	total_spent
2	32000

Explanation:

- **`Charlie`** (customer_id 3) is excluded because his total = 1000.

🧩 Example 5: Customers with at least 2 orders

```
sql

SELECT customer_id, COUNT(order_id) AS total_orders
FROM Orders
GROUP BY customer_id
HAVING COUNT(order_id) >= 2;
```

✅ **Result:**

customer_id	total_orders
1	2
2	3

🧩 Example 6: Combine `WHERE` and `HAVING`

👉 Suppose we only want customers who bought items **above ₹5000 each**, and their **total spending** is **greater than ₹30000**.

```
sql

SELECT customer_id, SUM(amount) AS total_spent
FROM Orders
WHERE amount > 5000          -- filter before grouping
GROUP BY customer_id
HAVING SUM(amount) > 30000;  -- filter after grouping
```

✅ **Result:**

customer_id	total_spent
1	110000

Explanation:

- **`WHERE amount > 5000`** removes small orders first.
- **`HAVING`** then checks total_spent after grouping.

🧠 Summary Table

Clause	Works On	Used For	Example
WHERE	Rows	Filtering before grouping	**`WHERE amount > 5000`**
GROUP BY	Groups	Grouping rows by a column	**`GROUP BY customer_id`**

Clause	Works On	Used For	Example
HAVING	Groups	Filtering after grouping	`HAVING SUM(amount) > 20000`

💡 Tip:

Always remember the **SQL execution order**:

```pgsql
FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY
```

That’s why `HAVING` comes **after** `GROUP BY` — because it works on the grouped results.

Would you like me to combine `JOIN + GROUP BY + HAVING` next (so you can see how they all work together in one query)?