**SQL - 02-09-2025**

Learn Joins → CTEs → Subqueries → Window Functions (that’s a good progression for data engineering SQL skills).

Got it 👍 You’ve made great progress! Let’s carefully plan the remaining SQL + Data Engineering basics so you’ll be interview-ready in the next 10 days, aiming for your ₹15 LPA goal.

You’ve already covered:

✅ SELECT, WHERE, ORDER BY

✅ Operators (AND, OR, NOT, LIKE, IN, BETWEEN, IS NULL, !=, <>)

✅ DISTINCT, LIMIT, OFFSET, AS

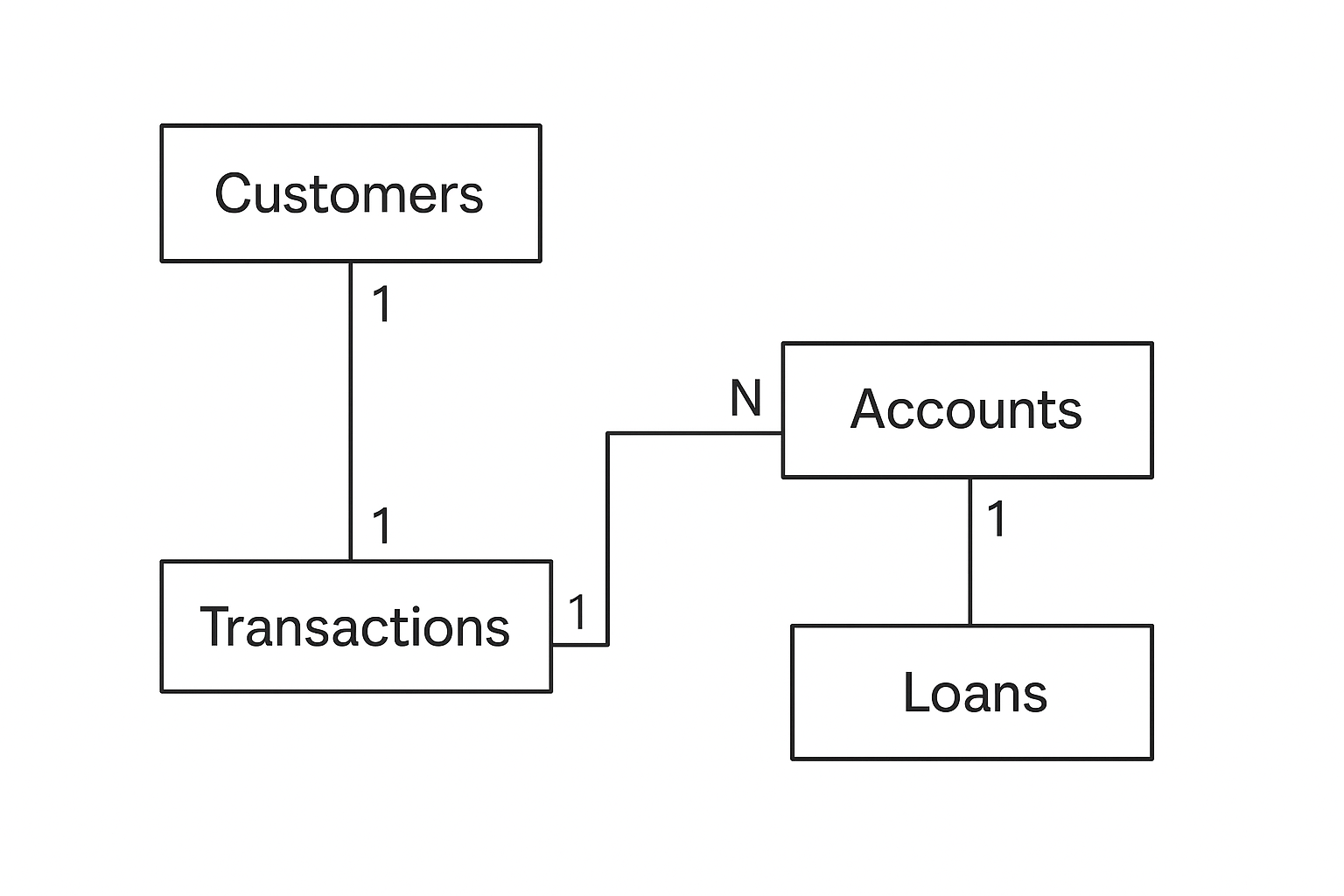
✅ Aggregations (COUNT, SUM, MIN, MAX, AVG)

✅ GROUP BY + HAVING

**📅 10-Day Data Engineering Plan (From Today)**

* Day 6 (Today): Joins (all types) + practice 20 questions.
* Day 7: Multi-joins + Joins with Aggregations.
* Day 8: Subqueries + CTEs (10 questions each).
* Day 9: Window Functions + Set Operators + String/Date functions.
* Day 10: Indexes + Query Performance Basics.
* Day 11–12: Data Modeling + ETL basics (interview focus).
* Day 13–14: Mock Interview Prep – mix SQL + data modeling Qs.
* Day 15: Final Revision + Apply to jobs aggressively.

**Day 6 (Today): Joins (all types) + practice 20 questions.**



**🎯 Roadmap to Learn SQL Joins (From Start to Pro)**

**✅ Step 1: Understand the Problem Joins Solve**

* SQL **Joins** combine data from **multiple tables** using a common column (key).
* Example in banking:
  + Customers are in one table.
  + Accounts are in another.
  + If you want **customer name + account balance**, you need a **JOIN**.

👉 Without joins, you’d have to manually combine tables — impossible in real data (millions of rows).

**✅ Step 2: Start with the Most Common Joins**

**1. INNER JOIN (most used)**

* Returns only **matching rows** between tables.
* Example: Customers who **have accounts**.

Perfect 🎯 Let’s learn **INNER JOIN** step by step — as if you have **never coded before**.  
I’ll go **slow** and explain:

* **What to write**
* **Why we write it**
* **What result it gives**
* **What to add next**

**🟢 Step 1: Start with One Table (Customers)**

First, let’s just see **customers**.

✅ Here we only see **customer data**.  
But we don’t know their account info (savings, current, FD, etc.).

**🟢 Step 2: Look at Second Table (Accounts)**

**✅ This table shows accounts, but we don’t know the names of the customers — only customer\_id.**

🟢 Step 3: Combine Both (INNER JOIN)

We want → Customer Name + Account Type + Balance.

👉 Here comes INNER JOIN:

SELECT c.customer\_name, a.account\_type, a.balance

FROM Customers c

INNER JOIN Accounts a

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

* Customers c → means we are using Customers table (short name = c).
* Accounts a → means we are using Accounts table (short name = a).
* ON c.customer\_id = a.customer\_id → tells SQL how to connect the two tables (the link is customer\_id).

🟢 Step 5: Understand What INNER JOIN Does

* INNER JOIN returns only rows that exist in both tables.
* Example: Rahul Mehta (customer 5) does not appear in result → because he has no account.

**So INNER JOIN = Common part only.**

🟢 Step 6: Practice Questions

Try to write queries (I’ll check your answers later 😊):

1. Show all customers and their account types.
2. Show customer names with balances more than 50,000.
3. Show city, account\_id, and balance of all customers with accounts.
4. Show customer names with their account type sorted by balance (highest first).
5. Find customers who have more than one account.

**Note: Important information**

🟢 SQL Practice Set with Info (Banking Dataset)

Q1. Show all customers and their account types.

* Focus: INNER JOIN
* Note: Very basic, interviewer checks if you know how to link PK & FK.

SELECT c.customer\_name, a.account\_type

FROM Customers c

INNER JOIN Accounts a

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

Q2. Show distinct cities where customers have accounts.

* Focus: DISTINCT
* Note: DISTINCT applies to all columns in SELECT. Use only city.

SELECT DISTINCT c.city

FROM Customers c

INNER JOIN Accounts a

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

Q3. Show customer names and balances for accounts with balance greater than 50,000.

* Focus: WHERE filter
* Note: >= includes equal, > excludes.

SELECT c.customer\_name, a.balance

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

WHERE a.balance > 50000;

Q4. Show customer names with their account type and balance, ordered by balance (highest first) and skip the first 2 results.

* Focus: ORDER BY + OFFSET
* MySQL: must use LIMIT … OFFSET.

-- MySQL

SELECT c.customer\_name, a.account\_type, a.balance

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

ORDER BY a.balance DESC

LIMIT 999999 OFFSET 2;

* Postgres: can use OFFSET 2 directly.
* SQL Server: OFFSET 2 ROWS syntax.

Q5. Show total balance of all accounts for each customer.

* Focus: Aggregation + GROUP BY

SELECT c.customer\_name, SUM(a.balance) AS total\_balance

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

GROUP BY c.customer\_name;

Q6. Find customers who have more than one account.

* Focus: GROUP BY + HAVING
* Tip: Use COUNT(account\_id), not COUNT(account\_type).

SELECT c.customer\_name, COUNT(a.account\_id) AS total\_accounts

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

GROUP BY c.customer\_name

HAVING COUNT(a.account\_id) > 1;

Q7. Show average balance by account type.

* Focus: Aggregation

SELECT a.account\_type, AVG(a.balance) AS avg\_balance

FROM Accounts a

GROUP BY a.account\_type;

Q8. Show customers with total balance above 100,000.

* Focus: HAVING after aggregation

SELECT c.customer\_name, SUM(a.balance) AS total\_balance

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

GROUP BY c.customer\_name

HAVING SUM(a.balance) > 100000;

Q9. Show the maximum balance for each city.

* Focus: GROUP BY city + aggregate MAX

SELECT c.city, MAX(a.balance) AS max\_balance

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

GROUP BY c.city;

Q10. Show top 3 customers by highest balance (pagination style).

* Focus: ORDER BY + LIMIT

-- MySQL

SELECT c.customer\_name, a.balance

FROM Customers c

INNER JOIN Accounts a

ON c.customer\_id = a.customer\_id

ORDER BY a.balance DESC

LIMIT 3;

-- SQL Server

ORDER BY a.balance DESC

OFFSET 0 ROWS FETCH NEXT 3 ROWS ONLY;

🧾 Cheat Notes for You

* DISTINCT → applies to all columns in SELECT. If you only want unique city names, only select city.
* GROUP BY → all non-aggregated columns in SELECT must appear in GROUP BY.
* HAVING → works like WHERE but for aggregated values.
* OFFSET →
  + PostgreSQL → works alone (OFFSET 2)
  + MySQL → must pair with LIMIT (LIMIT 9999 OFFSET 2)
  + SQL Server → needs OFFSET … ROWS FETCH NEXT …
* Aggregations → COUNT, SUM, AVG, MIN, MAX. Always think: row-level vs group-level.

**⚡ Key Rule**

* Without GROUP BY → you can sort by raw column values (a.balance).
* With GROUP BY → you must sort by aggregates (AVG(a.balance), SUM(a.balance), etc.), or by the grouped column (c.city).

**Excellent Paresh 🎉👏 That’s a big milestone! You’ve now built 30 INNER JOIN queries covering:**

* **✅ Basic INNER JOIN**
* **✅ WHERE conditions**
* **✅ DISTINCT**
* **✅ GROUP BY + HAVING**
* **✅ Aggregations (COUNT, SUM, AVG, MAX, MIN)**
* **✅ ORDER BY**
* **✅ LIMIT + OFFSET**

**This means you’ve covered 90% of real interview-style INNER JOIN questions. 🚀**

**📘 Step 1: LEFT JOIN Definition**

We’ll now master LEFT JOIN the same way you did with INNER JOIN — but with 50 progressively harder questions covering all topics:

* ✅ Definition (clear understanding)
* ✅ WHERE conditions
* ✅ DISTINCT
* ✅ GROUP BY + HAVING
* ✅ Aggregations (COUNT, SUM, AVG, MAX, MIN)
* ✅ ORDER BY
* ✅ LIMIT + OFFSET

👉 **LEFT JOIN** returns **all rows from the left table** (first table), and only the matching rows from the right table.  
If no match exists, you still get the left row, but the right side will have **NULLs**.

Example:

SELECT c.customer\_name, a.account\_type

FROM customers c

LEFT JOIN accounts a

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

Shows all customers, including those with no accounts.

If a customer doesn’t have an account → account\_type = NULL.

**📘 Step 2: Practice Questions (50 in Total)**

**🔹 Basic LEFT JOIN (10 Q)**

1. Show all customers with their accounts (even if no account exists).
2. Show all accounts with their customer names (flip roles using RIGHT JOIN or LEFT JOIN with accounts first).
3. Show all customers and replace missing account types with “No Account”.
4. Find customers who don’t have any account (use WHERE account\_type IS NULL).
5. List all customers with their balances, showing NULL if no account exists.
6. Show distinct cities of all customers (even without accounts).
7. Show all customer names and their account types sorted by customer\_name.
8. Show all customers and accounts, ordered by balance DESC (NULL balances last).
9. Show only customers without accounts (filter where account\_type IS NULL).
10. Show customers and accounts but limit the output to 5 rows.

**🔹 WHERE + DISTINCT (10 Q)**

1. Show all customers, but only accounts of type “Savings” (NULL still shows for others).
2. Show distinct account types per customer (including customers with no accounts).
3. Show all customers in Delhi, with accounts if available.
4. Show all customers whose balance is above 50,000 (or NULL if no account).
5. Show distinct cities where customers live, even if no accounts exist.
6. Show all customers and accounts, filtering customers with name starting with “A”.
7. Show all customers, but exclude those who don’t have an account (reverse logic).
8. Show all customers, but accounts only if type is “FD” or “Current”.
9. Show all customers who don’t belong to “Mumbai” and their accounts.
10. Show distinct customer names for all who have or don’t have accounts.

**🔹 GROUP BY + HAVING (10 Q)**

1. Show total accounts per customer (0 if no account).
2. Show average balance per customer (NULL if no accounts).
3. Show total balance per city (include cities with no accounts).
4. Show number of customers per account type (include NULL as “No Account”).
5. Show customers who have more than 2 accounts.
6. Show cities where average balance > 50,000.
7. Show customers whose total balance < 10,000 or no account (treat as 0).
8. Show max balance per customer.
9. Show min balance per city (include cities without accounts).
10. Show customers grouped by account type (NULL = No Account).

**🔹 Aggregations (10 Q)**

1. Show total sum of balances across all customers (NULL = 0).
2. Show count of accounts per customer (include customers without accounts).
3. Show average balance across all accounts.
4. Show highest balance per account type (NULL as “No Account”).
5. Show lowest balance across all accounts.
6. Show customers who hold multiple account types.
7. Show customers with the maximum balance in each city.
8. Show total balances grouped by city and account type.
9. Show customers whose balance is NULL (no accounts).
10. Show total accounts (including customers with 0).

**🔹 ORDER BY + LIMIT + OFFSET (10 Q)**

1. Show all customers with their accounts, ordered by balance DESC.
2. Show top 3 customers by total balance (include NULL = 0).
3. Show customers with no accounts, sorted alphabetically.
4. Show accounts ordered by balance, but include customers without accounts.
5. Show all customers with balances, limit to 5 rows.
6. Show next 5 customers after skipping first 3 (OFFSET + LIMIT).
7. Show customers ordered by city ASC, then balance DESC.
8. Show all accounts with customers, but NULL balances last.
9. Show customers ordered by whether they have an account or not.
10. Show customers ordered by number of accounts (0 first).

 DISTINCT **always applies to the entire row** of the SELECT list.

 So if you do: