**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

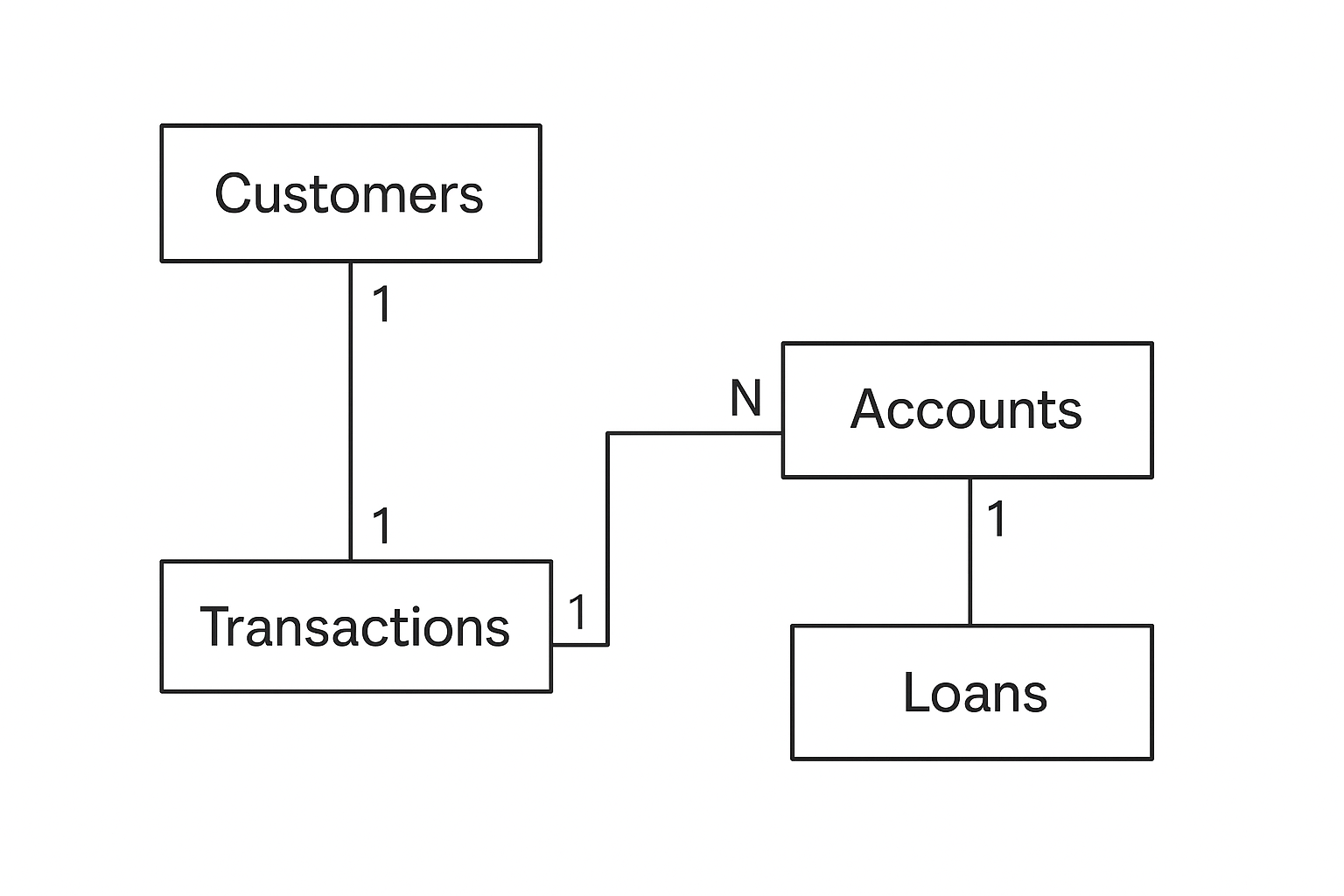
✅ Aggregations (COUNT, SUM, MIN, MAX, AVG)

✅ 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).