--------LECTURE- 13--------

1️⃣ What are Window Functions?

Window functions perform calculations across a set of rows related to the current row, without collapsing rows like GROUP BY.

👉 You still see **all rows**, but with **extra calculated columns**.

2️⃣ OVER() clause (Heart of Window Functions)

```
function_name() OVER (
  PARTITION BY column
  ORDER BY column
)
```

- **PARTITION BY** → divides data into groups (like GROUP BY, but keeps rows)
- **ORDER BY** → defines order inside each group

3️⃣ ROW_NUMBER()

Assigns a **unique number** to each row.

ROW_NUMBER() OVER (PARTITION BY dept ORDER BY salary DESC)

- Always unique
- Used for **deduplication**, **top N records**

📌 If salaries are same → still different row numbers.

4️⃣ RANK()

Assigns rank with **gaps**.

RANK() OVER (PARTITION BY dept ORDER BY salary DESC)

- Same values → same rank
- Next rank is skipped

Example: 1, 1, 3

5️⃣ DENSE_RANK()

Assigns rank **without gaps**.

DENSE_RANK() OVER (PARTITION BY dept ORDER BY salary DESC)

Example: 1, 1, 2

📌 Used when **continuous ranking** is needed.

6️⃣ COALESCE()

Not a window function, but **commonly used with them**.

COALESCE(column, 0)

- Returns first **non-NULL** value
- Used to handle NULLs in calculations

7️⃣ NTILE(n)

Divides data into **n equal buckets**.

NTILE(4) OVER (ORDER BY salary)

- Used for **quartiles, percentiles**
- Common in analytics & reporting

8️⃣ LAG()

Gets value from **previous row**.

LAG(salary, 1) OVER (ORDER BY date)

Used for:

- Comparing current vs previous
- Growth analysis

9️⃣ LEAD()

Gets value from **next row**.

LEAD(salary, 1) OVER (ORDER BY date)

Used for:

- Future comparison
- Trend analysis

🔟 RUNNING TOTAL

Cumulative sum over rows.

SUM(amount) OVER (ORDER BY date)

- Adds current + previous values
- Common in **financial reports**

🔷 CTE (Common Table Expression)

1️⃣ What is a CTE?

A **temporary named result set** used inside a query.

```
WITH cte_name AS (
  SELECT ...
)
SELECT * FROM cte_name;
```

📌 Improves **readability** and **reusability**.

2️⃣ Simple CTE

Used to break complex logic.

```
WITH sales_cte AS (
  SELECT emp_id, SUM(amount) total_sales
  FROM sales
  GROUP BY emp_id
)
SELECT * FROM sales_cte;
```

3️⃣ Using CTE Multiple Times

Same CTE can be reused in one query.

```
WITH data AS (...)
SELECT ...
FROM data d1
JOIN data d2 ...
```

📌 Avoids repeating subqueries.

---

4 Window Functions + CTE Combo

CTE prepares data → Window function analyzes it.

```
WITH ranked_sales AS (
  SELECT emp_id, salary,
         RANK() OVER (ORDER BY salary DESC) rnk
  FROM employees
)
SELECT * FROM ranked_sales WHERE rnk <= 3;
```

---

5 Recursive CTE

Used for **hierarchical data** (manager → employee).

```
WITH RECURSIVE emp_hierarchy AS (
  SELECT emp_id, manager_id
  FROM employees
  WHERE manager_id IS NULL

  UNION ALL

  SELECT e.emp_id, e.manager_id
  FROM employees e
  JOIN emp_hierarchy h
  ON e.manager_id = h.emp_id
)
SELECT * FROM emp_hierarchy;
```

📌 Used for:

- Org charts
- Tree structures
- Parent-child relationships

✅ **Summary**

- **Window functions** → calculations without collapsing rows
- **OVER()** → defines window
- **ROW_NUMBER / RANK / DENSE_RANK** → ranking logic
- **LAG / LEAD** → row comparison
- **NTILE** → bucket distribution
- **Running total** → cumulative analytics
- **CTE** → readable, reusable queries
- **Recursive CTE** → hierarchical data