📘 Module 30: Window Functions

This module introduces **window functions** in SQL that let you perform calculations across rows while keeping individual row details. These are powerful for advanced analytics like ranking, running totals, comparisons, etc.

🧾 Reference Tables Used in This Module:

📊 Table 1: Sales

```
select * from Sales;
```

100 %

Results | Messages

	SalesID	SalesDate	Amount
1	1	2024-01-01	100
2	2	2024-01-02	200
3	3	2024-01-03	150
4	4	2024-01-04	300
5	5	2024-01-05	250

👨‍💼 Table 2: Employees

```
select * from Employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary
1	1	John	HR	50000.00
2	2	Jane	HR	55000.00
3	3	JayY	HR	50000.00
4	4	Bob	IT	70000.00
5	5	Alice	IT	75000.00
6	6	Charlie	IT	72000.00
7	7	Raghav	IT	72000.00
8	8	Dave	Sales	60000.00
9	9	Eve	Sales	53000.00
10	10	Frank	Sales	62000.00
11	11	Grace	HR	65000.00
12	12	Heidi	IT	68000.00

🌐 1. 💡 What are Window Functions

🔍 **Window functions** perform calculations across a group of rows (a "window") that are related to the current row.

They **do not collapse** the rows like `GROUP BY`. Instead, they **return values for every row**.

🧩 Syntax:

```
FUNCTION_NAME(column)
OVER (
  PARTITION BY column_name
  ORDER BY column_name
  ROWS BETWEEN ... -- optional
)
```

💡 Components:

- 🧭 `PARTITION BY` – Divides the result into groups (like departments).
- 🪜 `ORDER BY` – Sorts rows inside each group (e.g., by salary).
- 🧱 `ROWS` **or** `RANGE` – Controls which rows are included in the window (optional, advanced use).

🔢 2. 🎯 `ROW_NUMBER()`

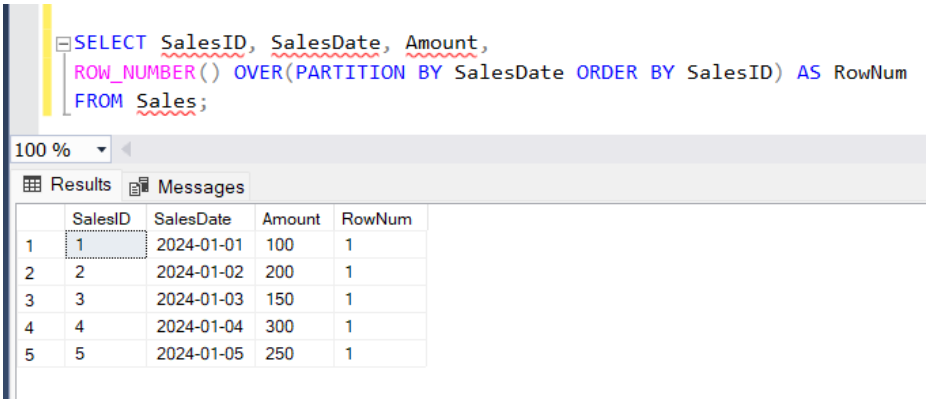
🚦 Gives a unique serial number to each row within a partition.

🔣 Syntax:

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

📌 Example:

```
SELECT SalesID, SalesDate, Amount,
ROW_NUMBER() OVER(PARTITION BY SalesDate ORDER BY SalesID) AS RowNum
FROM Sales;
```



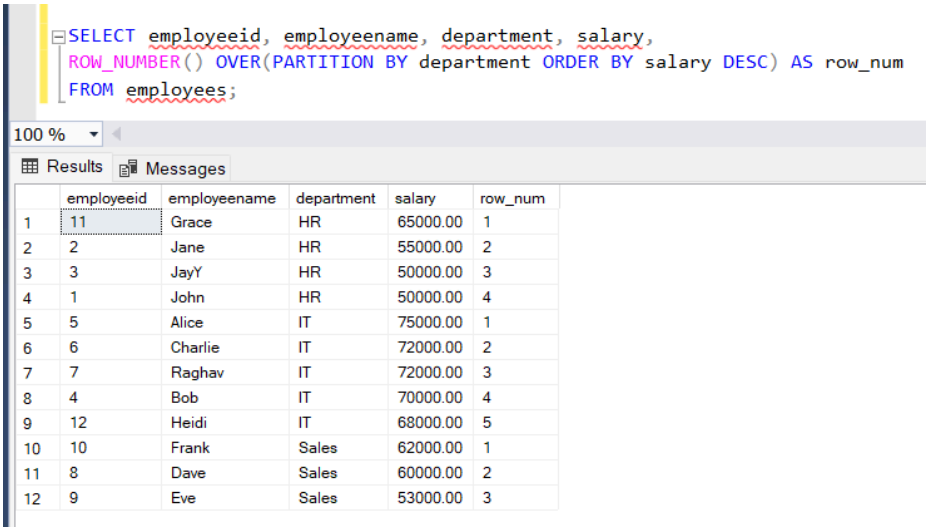
	SalesID	SalesDate	Amount	RowNum
1	1	2024-01-01	100	1
2	2	2024-01-02	200	1
3	3	2024-01-03	150	1
4	4	2024-01-04	300	1
5	5	2024-01-05	250	1

👑 3. 🧱 Row Number Implementation (Use Case)

Use this to get top earners in each department.

📌 Example 1:

SELECT employeeid, employeename, department, salary,
ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS row_num
FROM employees;



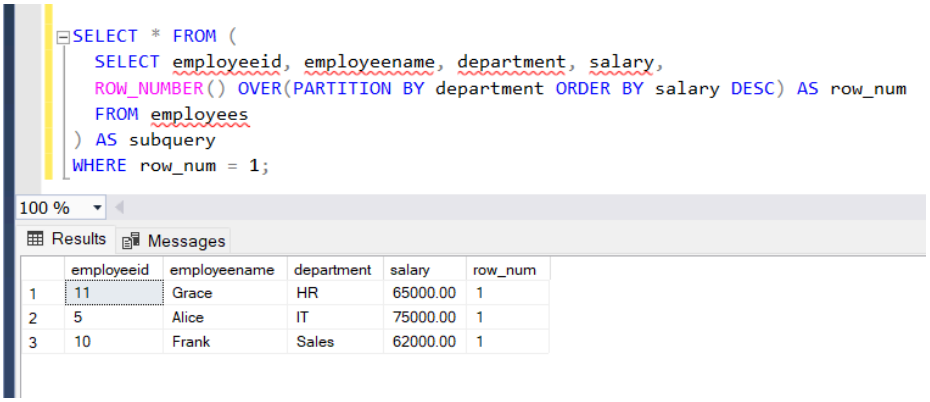
```
SELECT employeeid, employeename, department, salary,
ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS row_num
FROM employees;
```

100 % | Results | Messages

	employeeid	employeename	department	salary	row_num
1	11	Grace	HR	65000.00	1
2	2	Jane	HR	55000.00	2
3	3	JayY	HR	50000.00	3
4	1	John	HR	50000.00	4
5	5	Alice	IT	75000.00	1
6	6	Charlie	IT	72000.00	2
7	7	Raghav	IT	72000.00	3
8	4	Bob	IT	70000.00	4
9	12	Heidi	IT	68000.00	5
10	10	Frank	Sales	62000.00	1
11	8	Dave	Sales	60000.00	2
12	9	Eve	Sales	53000.00	3

📌 Example 2 (Only Top Earners):

SELECT * FROM (
  SELECT employeeid, employeename, department, salary,
  ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS row_num
  FROM employees
) AS subquery
WHERE row_num = 1;



```
SELECT * FROM (
  SELECT employeeid, employeename, department, salary,
  ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS row_num
  FROM employees
) AS subquery
WHERE row_num = 1;
```

100 % | Results | Messages

	employeeid	employeename	department	salary	row_num
1	11	Grace	HR	65000.00	1
2	5	Alice	IT	75000.00	1
3	10	Frank	Sales	62000.00	1

🥇 4. 🎖️ RANK() vs DENSE_RANK()

🗂️ Used to rank rows within a partition.

🔣 Syntax:

RANK() OVER(PARTITION BY column ORDER BY column)
DENSE_RANK() OVER(PARTITION BY column ORDER BY column)

📌 Examples:

SELECT *, RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;

```
SELECT *, RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	rank
1	11	Grace	HR	65000.00	1
2	2	Jane	HR	55000.00	2
3	3	JayY	HR	50000.00	3
4	1	John	HR	50000.00	3
5	5	Alice	IT	75000.00	1
6	6	Charlie	IT	72000.00	2
7	7	Raghav	IT	72000.00	2
8	4	Bob	IT	70000.00	4
9	12	Heidi	IT	68000.00	5
10	10	Frank	Sales	62000.00	1
11	8	Dave	Sales	60000.00	2
12	9	Eve	Sales	53000.00	3

SELECT *, DENSE_RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;

```
SELECT *, DENSE_RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	rank
1	11	Grace	HR	65000.00	1
2	2	Jane	HR	55000.00	2
3	3	JayY	HR	50000.00	3
4	1	John	HR	50000.00	3
5	5	Alice	IT	75000.00	1
6	6	Charlie	IT	72000.00	2
7	7	Raghav	IT	72000.00	2
8	4	Bob	IT	70000.00	3
9	12	Heidi	IT	68000.00	4
10	10	Frank	Sales	62000.00	1
11	8	Dave	Sales	60000.00	2
12	9	Eve	Sales	53000.00	3

🧠 Key Difference:

Function	Handles Ties	Skips Rank
ROW_NUMBER	❌ No	❌ No
RANK	✅ Yes	✅ Yes
DENSE_RANK	✅ Yes	❌ No

employee_id	name	salary	rank	dense_rank	row_num
4	David	70000	1	1	1
2	Bob	60000	2	2	2
5	Eva	60000	2	2	3
1	Alice	50000	4	3	4
3	Charlie	50000	4	3	5

🎯 5. 🪜 NTILE(n)

📊 Divides data into equal parts or buckets (for percentile-based grouping).

🔣 Syntax:

NTILE(n) OVER(ORDER BY column)

📌 Example:

SELECT employeeid, employeename, salary,
NTILE(3) OVER(ORDER BY salary DESC) AS ntile_group
FROM employees;

```
SELECT employeeid, employeename, salary,
NTILE(3) OVER(ORDER BY salary DESC) AS ntile_group
FROM employees;
```

100 %

Results | Messages

	employeeid	employeename	salary	ntile_group
1	5	Alice	75000.00	1
2	6	Charlie	72000.00	1
3	7	Raghav	72000.00	1
4	4	Bob	70000.00	1
5	12	Heidi	68000.00	2
6	11	Grace	65000.00	2
7	10	Frank	62000.00	2
8	8	Dave	60000.00	2
9	2	Jane	55000.00	3
10	9	Eve	53000.00	3
11	3	JayY	50000.00	3
12	1	John	50000.00	3

⚖️ 6. 📉 AVG() (Window Average)

📐 Calculates average for a group but keeps the row.

🔣 Syntax:

AVG(column) OVER(PARTITION BY column)

📌 Examples:

SELECT *, AVG(salary) OVER(PARTITION BY department) AS avg_salary
FROM employees;

```
SELECT *, AVG(salary) OVER(PARTITION BY department) AS avg_salary
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	avg_salary
1	1	John	HR	50000.00	55000.000000
2	2	Jane	HR	55000.00	55000.000000
3	3	JayY	HR	50000.00	55000.000000
4	11	Grace	HR	65000.00	55000.000000
5	12	Heidi	IT	68000.00	71400.000000
6	4	Bob	IT	70000.00	71400.000000
7	5	Alice	IT	75000.00	71400.000000
8	6	Charlie	IT	72000.00	71400.000000
9	7	Raghav	IT	72000.00	71400.000000
10	8	Dave	Sales	60000.00	58333.333333
11	9	Eve	Sales	53000.00	58333.333333
12	10	Frank	Sales	62000.00	58333.333333

-- Rounded version
SELECT *, FLOOR(AVG(salary) OVER(PARTITION BY department)) AS avg_salary
FROM employees;

```
SELECT *, FLOOR(AVG(salary) OVER(PARTITION BY department)) AS avg_salary
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	avg_salary
1	1	John	HR	50000.00	55000
2	2	Jane	HR	55000.00	55000
3	3	JayY	HR	50000.00	55000
4	11	Grace	HR	65000.00	55000
5	12	Heidi	IT	68000.00	71400
6	4	Bob	IT	70000.00	71400
7	5	Alice	IT	75000.00	71400
8	6	Charlie	IT	72000.00	71400
9	7	Raghav	IT	72000.00	71400
10	8	Dave	Sales	60000.00	58333
11	9	Eve	Sales	53000.00	58333
12	10	Frank	Sales	62000.00	58333

🔢🔁 7. COUNT() (Window Count)

📋 Counts how many rows are there in the partition.

🔣 Syntax:

COUNT(column) OVER(PARTITION BY column)

📌 Example:

SELECT *, COUNT(EmployeeID) OVER(PARTITION BY department) AS employee_count
FROM employees;

```
SELECT *, COUNT(EmployeeID) OVER(PARTITION BY department) AS employee_count
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	employee_count
1	1	John	HR	50000.00	4
2	2	Jane	HR	55000.00	4
3	3	JayY	HR	50000.00	4
4	11	Grace	HR	65000.00	4
5	12	Heidi	IT	68000.00	5
6	4	Bob	IT	70000.00	5
7	5	Alice	IT	75000.00	5
8	6	Charlie	IT	72000.00	5
9	7	Raghav	IT	72000.00	5
10	8	Dave	Sales	60000.00	3
11	9	Eve	Sales	53000.00	3
12	10	Frank	Sales	62000.00	3

💰➕ 8. SUM() (Window Total)

💸 Gives total sum of values in a group.

🔣 Syntax:

SUM(column) OVER(PARTITION BY column)

📌 Example:

SELECT *, SUM(Salary) OVER(PARTITION BY department) AS total_salary
FROM employees;

```
SELECT *, SUM(Salary) OVER(PARTITION BY department) AS total_salary
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	total_salary
1	1	John	HR	50000.00	220000.00
2	2	Jane	HR	55000.00	220000.00
3	3	JayY	HR	50000.00	220000.00
4	11	Grace	HR	65000.00	220000.00
5	12	Heidi	IT	68000.00	357000.00
6	4	Bob	IT	70000.00	357000.00
7	5	Alice	IT	75000.00	357000.00
8	6	Charlie	IT	72000.00	357000.00
9	7	Raghav	IT	72000.00	357000.00
10	8	Dave	Sales	60000.00	175000.00
11	9	Eve	Sales	53000.00	175000.00
12	10	Frank	Sales	62000.00	175000.00

📈 9. 🔁 Running Total

📊 Running total is a **cumulative sum** row by row.

🔣 Syntax:

SUM(column) OVER(PARTITION BY column ORDER BY column)

📌 Example:

SELECT *,
SUM(Salary) OVER(PARTITION BY department ORDER BY Salary DESC) AS cumulative_salary
FROM employees
ORDER BY department, salary DESC;

```
SELECT *,
SUM(Salary) OVER(PARTITION BY department ORDER BY Salary DESC) AS cumulative_salary
FROM employees
ORDER BY department, salary DESC;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	cumulative_salary
1	11	Grace	HR	65000.00	65000.00
2	2	Jane	HR	55000.00	120000.00
3	3	JayY	HR	50000.00	220000.00
4	1	John	HR	50000.00	220000.00
5	5	Alice	IT	75000.00	75000.00
6	6	Charlie	IT	72000.00	219000.00
7	7	Raghav	IT	72000.00	219000.00
8	4	Bob	IT	70000.00	289000.00
9	12	Heidi	IT	68000.00	357000.00
10	10	Frank	Sales	62000.00	62000.00
11	8	Dave	Sales	60000.00	122000.00
12	9	Eve	Sales	53000.00	175000.00

🔁 10. ⬅️➡️ `LAG()` and `LEAD()`

⏮️ `LAG()` → Gets value from previous row

⏭️ `LEAD()` → Gets value from next row

🔣 Syntax:

LAG(column) OVER(PARTITION BY col ORDER BY col)
LEAD(column) OVER(PARTITION BY col ORDER BY col)

📌 Example:

SELECT *,
LAG(Salary) OVER(PARTITION BY department ORDER BY salary DESC) AS previous_salary,
LEAD(Salary) OVER(PARTITION BY department ORDER BY salary DESC) AS next_salary
FROM employees;

```sql
SELECT *,
LAG(Salary) OVER(PARTITION BY department ORDER BY salary DESC) AS previous_salary,
LEAD(Salary) OVER(PARTITION BY department ORDER BY salary DESC) AS next_salary
FROM employees;
```

100 %

Results | Messages

	EmployeeID	EmployeeName	Department	Salary	previous_salary	next_salary
1	11	Grace	HR	65000.00	NULL	55000.00
2	2	Jane	HR	55000.00	65000.00	50000.00
3	3	JayY	HR	50000.00	55000.00	50000.00
4	1	John	HR	50000.00	50000.00	NULL
5	5	Alice	IT	75000.00	NULL	72000.00
6	6	Charlie	IT	72000.00	75000.00	72000.00
7	7	Raghav	IT	72000.00	72000.00	70000.00
8	4	Bob	IT	70000.00	72000.00	68000.00
9	12	Heidi	IT	68000.00	70000.00	NULL
10	10	Frank	Sales	62000.00	NULL	60000.00
11	8	Dave	Sales	60000.00	62000.00	53000.00
12	9	Eve	Sales	53000.00	60000.00	NULL

🧠 Key Points to Remember

✅ Window functions **do not group** data like `GROUP BY`.

✅ You can use `PARTITION BY` to group, but rows are still individual.

✅ Always use `ORDER BY` for meaningful results in ranking, cumulative total, etc.

✅ LAG/LEAD are useful for **comparing** rows side by side.

✅ ROWS BETWEEN can be used for **moving average, running max, min** (advanced).

🧠 Additional Tips

🔷 You can use **WHERE row_num = 1** trick with ROW_NUMBER to filter top entries per group

🔷 Can also be used in **CTEs** (Common Table Expressions)

🔷 Great for **reporting**, **pagination**, and **analytical dashboards**
