Excellent 💪 — let’s go **deep into Advanced Window Functions in SQL**, step by step — from **concept → syntax → advanced use cases → interview questions**.

This topic is very powerful for **analytics, reporting, ranking, and running totals** — and is frequently asked in **SQL interviews**.

**🪟 1. What Are Window Functions?**

**Definition:**  
A **window function** performs a **calculation across a set of rows** that are **related to the current row**, without collapsing the result set (unlike GROUP BY).

Think of it as:

“Aggregate functions but they return **a value for each row**, not one value for the group.”

**🧠 Example**

SELECT

Name,

Salary,

AVG(Salary) OVER() AS Avg\_Salary

FROM Employee;

➡ Every row still appears, but each row shows the **average salary of all employees**.

**⚙️ 2. General Syntax**

function\_name(expression)

OVER (

[PARTITION BY column\_name]

[ORDER BY column\_name]

[ROWS or RANGE clause]

)

| **Clause** | **Purpose** |
| --- | --- |
| PARTITION BY | Divides result into groups (like GROUP BY) |
| ORDER BY | Defines ordering within each partition |
| ROWS / RANGE | Defines the frame (which rows are considered for calculation) |

**🧱 3. Types of Window Functions**

| **Type** | **Examples** |
| --- | --- |
| **Aggregate** | SUM(), AVG(), COUNT(), MIN(), MAX() |
| **Ranking** | ROW\_NUMBER(), RANK(), DENSE\_RANK(), NTILE() |
| **Value** | LAG(), LEAD(), FIRST\_VALUE(), LAST\_VALUE() |
| **Distribution** | PERCENT\_RANK(), CUME\_DIST() |

**🔹 4. Aggregate Window Functions**

**Example 1 — Average salary per department**

SELECT

DeptID,

Name,

Salary,

AVG(Salary) OVER(PARTITION BY DeptID) AS AvgDeptSalary

FROM Employee;

➡ Shows each employee along with **average salary of their department**.

**Example 2 — Running total (cumulative sum)**

SELECT

EmpID,

Salary,

SUM(Salary) OVER(ORDER BY EmpID) AS RunningTotal

FROM Employee;

➡ Keeps a **cumulative total** as rows progress.

**🏆 5. Ranking Window Functions**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| ROW\_NUMBER() | Sequential number per row | 1, 2, 3, ... |
| RANK() | Same rank for ties, skips next number | 1, 1, 3, ... |
| DENSE\_RANK() | Same rank for ties, no gaps | 1, 1, 2, ... |
| NTILE(n) | Divides rows into n groups | Quartiles/percentiles |

**Example — Ranking by Salary**

SELECT

Name,

Salary,

RANK() OVER(ORDER BY Salary DESC) AS RankBySalary,

DENSE\_RANK() OVER(ORDER BY Salary DESC) AS DenseRank,

ROW\_NUMBER() OVER(ORDER BY Salary DESC) AS RowNum

FROM Employee;

| **Name** | **Salary** | **RankBySalary** | **DenseRank** | **RowNum** |
| --- | --- | --- | --- | --- |
| A | 90000 | 1 | 1 | 1 |
| B | 90000 | 1 | 1 | 2 |
| C | 85000 | 3 | 2 | 3 |

**📈 6. Value Functions (Access Other Rows)**

These functions look “around” the current row — perfect for **trend analysis** or **comparison**.

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| LAG(expr, offset) | Returns value from previous row | Compare current vs previous |
| LEAD(expr, offset) | Returns value from next row | Compare current vs next |
| FIRST\_VALUE(expr) | First value in window | Get first salary in dept |
| LAST\_VALUE(expr) | Last value in window | Get last salary in dept |

**Example 1 — LAG & LEAD**

SELECT

EmpID,

Name,

Salary,

LAG(Salary) OVER(ORDER BY EmpID) AS PrevSalary,

LEAD(Salary) OVER(ORDER BY EmpID) AS NextSalary

FROM Employee;

➡ Compare each employee’s salary with their previous and next one.

**Example 2 — FIRST\_VALUE and LAST\_VALUE**

SELECT

DeptID,

Name,

Salary,

FIRST\_VALUE(Salary) OVER(PARTITION BY DeptID ORDER BY Salary DESC) AS HighestSalary,

LAST\_VALUE(Salary) OVER(PARTITION BY DeptID ORDER BY Salary DESC

RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS LowestSalary

FROM Employee;

➡ Shows each employee along with the **highest and lowest salary** in their department.

**🔍 7. ROWS vs RANGE Clause (Frame Specification)**

Defines **which rows** to include in the window for calculation.

| **Frame** | **Description** |
| --- | --- |
| ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW | From start to current |
| ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING | Sliding window |
| RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING | Entire partition |

**Example — Rolling Average**

SELECT

EmpID,

Salary,

AVG(Salary) OVER(ORDER BY EmpID

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAvg

FROM Employee;

➡ Each row shows **average salary of current + previous 2 rows**.

**💡 8. Distribution Functions**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| PERCENT\_RANK() | Relative rank between 0 and 1 | 0 = lowest, 1 = highest |
| CUME\_DIST() | Cumulative distribution up to current row | Shows percentile position |

SELECT

Name,

Salary,

PERCENT\_RANK() OVER(ORDER BY Salary) AS PercentRank,

CUME\_DIST() OVER(ORDER BY Salary) AS CumDist

FROM Employee;

➡ Used in statistical analysis and percentile reporting.

**🧮 9. Combine Multiple Window Functions**

SELECT

DeptID,

Name,

Salary,

SUM(Salary) OVER(PARTITION BY DeptID) AS TotalDeptSalary,

RANK() OVER(PARTITION BY DeptID ORDER BY Salary DESC) AS DeptRank,

AVG(Salary) OVER() AS CompanyAvg

FROM Employee;

✅ Combine aggregates, ranks, and overall calculations in one result.

**⚙️ 10. Performance Tips**

✅ Use PARTITION BY wisely — large partitions may slow queries.  
✅ Index columns used in ORDER BY and PARTITION BY.  
✅ Avoid unnecessary nested window functions.  
✅ Use CTEs or subqueries to simplify logic for readability.

**🧠 11. Real-Life Use Cases**

| **Use Case** | **Function** |
| --- | --- |
| Top 3 employees per department | RANK() or DENSE\_RANK() |
| Running total of sales | SUM() OVER (ORDER BY Date) |
| Month-over-month growth | LAG() |
| Calculate percentile | CUME\_DIST() |
| Compare current vs previous year revenue | LAG() |
| Find first/last transaction | FIRST\_VALUE(), LAST\_VALUE() |

**💬 12. Interview Questions on Window Functions**

| **#** | **Question** | **Answer** |
| --- | --- | --- |
| 1 | What is the difference between GROUP BY and OVER()? | GROUP BY collapses rows, OVER() keeps all rows. |
| 2 | Difference between RANK() and DENSE\_RANK()? | RANK() skips numbers, DENSE\_RANK() doesn’t. |
| 3 | Can we use ORDER BY in both query and window function? | Yes — independent of each other. |
| 4 | What’s the difference between LAG() and LEAD()? | LAG() looks at previous row, LEAD() at next. |
| 5 | Can window functions be used in WHERE clause? | ❌ No — use subquery or CTE. |
| 6 | Default window frame for SUM() with ORDER BY? | RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. |
| 7 | What does PARTITION BY do? | Divides result into separate windows. |
| 8 | How to get top 3 salaries per department? | Use RANK() with a filter in outer query. |
| 9 | What is difference between ROWS and RANGE? | ROWS counts physical rows; RANGE uses value ranges. |
| 10 | Can you nest window functions? | No — but you can use them in subqueries. |

**🧭 13. Summary Mind Map**

Window Functions

│

├── Definition → Calculations across related rows

├── Clauses

│ ├─ PARTITION BY

│ ├─ ORDER BY

│ └─ ROWS / RANGE

├── Categories

│ ├─ Aggregate → SUM, AVG

│ ├─ Ranking → RANK, DENSE\_RANK, ROW\_NUMBER

│ ├─ Value → LAG, LEAD, FIRST\_VALUE

│ ├─ Distribution → PERCENT\_RANK, CUME\_DIST

├── Use Cases

│ ├─ Running totals

│ ├─ Ranking top N

│ ├─ Trends / Comparisons

│ ├─ Percentiles

│ └─ Rolling averages

└── Restrictions

├─ Not in WHERE

├─ Used in SELECT / ORDER BY

└─ Executes after aggregation