Sravya Madipalli ✅

SQL
WINDOW
FUNCTIONS

- Key to unlock advanced analytics

SQL window functions allow you to perform calculations across rows of data that are related to each other.

Today, we will look into...

- Understanding the Window Function Syntax

- Types of Window Functions

- How to use AVG() to calculate running totals.

- How to apply RANK() to rank rows within partitions.

- How to use LAG() to compare values from previous rows.

- Real-world examples, step-by-step explanations, and interview-style questions to test your knowledge.

Understanding the Window Function Syntax

1. OVER() Clause

The OVER() clause defines the window of rows that the function operates on. It allows you to specify the partitioning, ordering, and range of rows for the window function.

Example:

```sql
SELECT
    employee_id,
    salary,
    SUM(salary) OVER () AS total_salary
FROM
    employees;
```

In this example, the OVER() clause operates over the entire result set because no partitioning or ordering is specified.

Understanding the Window Function Syntax

2. PARTITION BY Clause

The PARTITION BY clause divides the result set into partitions, similar to how GROUP BY works in aggregate queries. Each partition is then treated independently by the window function.

Example:

```sql
SELECT
    employee_id,
    department_id,
    salary,
    SUM(salary) OVER (PARTITION BY department_id) AS dept_total_salary
FROM
    employees;
```

Here, the result set is partitioned by department_id, so the SUM(salary) is calculated separately for each department.

Understanding the Window Function Syntax

3. ORDER BY Clause

The ORDER BY clause within a window function defines the order of rows in each partition before the window function is applied. This is crucial for functions that depend on row order, such as ranking functions.

Example:

```sql
                        SQL
SELECT
    employee_id,
    department_id,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC)
AS rank_salary
FROM
    employees;
```

In this case, the ORDER BY salary DESC orders employees by salary within each department, allowing the RANK() function to assign ranks based on this order.

Understanding the Window Function Syntax

4. ROWS BETWEEN Clause

The ROWS BETWEEN clause specifies a range of rows relative to the current row. This is particularly useful for cumulative calculations, such as running totals or averages.

Example:

```sql
SELECT
    employee_id,
    salary,
    SUM(salary) OVER (ORDER BY employee_id ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS running_total
FROM
    employees;
```

This example calculates a running total of salaries by summing all rows from the beginning of the partition (UNBOUNDED PRECEDING) to the current row (CURRENT ROW).

Understanding the Window Function Syntax

5. UNBOUNDED PRECEDING

The UNBOUNDED PRECEDING keyword refers to the first row of the partition. It's often used in cumulative calculations to include all rows from the start up to a certain point.

6. CURRENT ROW

The CURRENT ROW refers to the row currently being evaluated by the window function. It acts as a reference point within the window.

Understanding the Window Function Syntax

7. FOLLOWING

The FOLLOWING keyword is used to refer to rows that come after the current row. This can be useful in scenarios where you need to perform calculations involving subsequent rows.

Example:

```sql
SELECT
    employee_id,
    salary,
    SUM(salary) OVER (ORDER BY employee_id ROWS BETWEEN CURRENT
ROW AND 1 FOLLOWING) AS next_two_salaries
FROM
    employees;
```
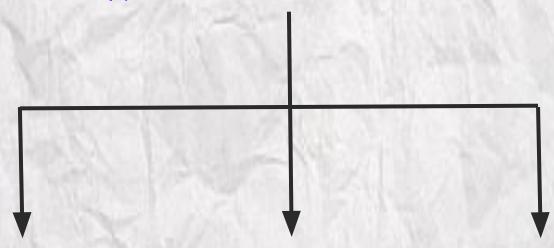
Types of Window Functions

Aggregate
- AVG()
- MAX()
- MIN()
- SUM()
- COUNT()

Ranking
- ROW_NUMBER()
- RANK()
- DENSE_RANK()
- PERCENT_RANK()
- NTILE()

Aggregate
- LAG()
- LEAD()
- FIRST_VALUE()
- LAST_VALUE()
- NTH_VALUE()

Aggregate Window Functions

Functions: AVG(), SUM(), COUNT(), MIN(), MAX()

- Example Function: AVG()
- Explanation:
  - Aggregate window functions compute values over a specified window of rows. Unlike standard aggregate functions that collapse rows into a single result, window functions retain the individual row details while applying the aggregate calculation.
- <u>Sample Code:</u>

```sql
SELECT
    employee_id,
    department_id,
    salary,
    AVG(salary) OVER (PARTITION BY department_id ORDER BY employee_id) AS avg_salary_by_dept
FROM
    employees;
```

Aggregate Window Functions

<u>Step-by-Step Explanation:</u>

- Select Relevant Columns:
  - employee_id, department_id, and salary are selected from the employees table to display each employee's information along with their department and salary.
- Apply the AVG() Function:
  - The AVG(salary) function calculates the average salary over a specific window of rows. Unlike the regular AVG() function that would collapse the rows into a single result, the window function version retains each row while applying the calculation.
- Define the Window with OVER():
  - The OVER() clause defines the window of rows over which the AVG() function is applied.
- Partition the Data with PARTITION BY:
  - PARTITION BY department_id divides the data into partitions based on the department. This means the average salary is calculated separately for each department.
- Order the Rows with ORDER BY:
  - ORDER BY employee_id orders the rows within each partition by employee_id. While this doesn't directly affect the average calculation, it determines the order in which the rows are processed within each department.
- Alias the Result:
  - AS avg_salary_by_dept assigns an alias to the calculated average salary, making it easier to reference in the result set.

Result: This query calculates the average salary for each department and displays it alongside each employee's details.

Aggregate Window Functions

- ## <u>Sample Output:</u>

employee_id	department_id	salary	avg_salary_by_dept
1	10	70000	70000
2	10	75000	72500
3	10	80000	75000
4	20	60000	60000
5	20	62000	61000

- ## <u>Interview Question:</u>

  - *How would you calculate a running average of employee salaries within each department?*

- ## <u>ChatGPT Prompt:</u>

  - *"Explain how to calculate a cumulative average for a specific group using SQL window functions."*

Ranking Window Functions

Functions: ROW_NUMBER(), RANK(), DENSE_RANK()

- Example Function: RANK()
- Explanation:
  - Ranking window functions assign a unique rank to each row based on the order specified in the window. These functions are helpful for generating row numbers, rankings, and handling ties in your data.
- Sample Code:

```sql
SELECT
    employee_id,
    department_id,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC)
AS salary_rank
FROM
    employees;
```

Ranking Window Functions

Step-by-Step Explanation:

1. Select Relevant Columns:
   - employee_id, department_id, and salary are selected to display each employee's basic information and their salary.
2. Apply the RANK() Function:
   - The RANK() function assigns a rank to each row within the window defined by OVER(). The rank is based on the order specified in the ORDER BY clause.
3. Define the Window with OVER():
   - The OVER() clause specifies the window of rows over which the RANK() function operates.
4. Partition the Data with PARTITION BY:
   - PARTITION BY department_id divides the data into partitions by department, so the ranking is done separately within each department.
5. Order the Rows with ORDER BY:
   - ORDER BY salary DESC orders the rows within each partition by salary in descending order. The highest salary in each department gets the rank of 1.
6. Handle Ties:
   - The RANK() function handles ties by assigning the same rank to rows with equal values and skipping the next rank(s).
7. Alias the Result:
   - AS salary_rank assigns an alias to the rank, making it easy to reference in the result set.

Result: This query ranks employees within each department based on their salary. Employees with the same salary will receive the same rank, and the next rank will be skipped.

Ranking Window Functions

- Sample Output

employee_id	department_id	salary	salary_rank
1	10	80000	1
2	10	75000	2
3	10	75000	2
4	20	90000	1
5	20	60000	2

- <u>Interview Question:</u>

  - *What is the difference between RANK(), ROW_NUMBER() and DENSE_RANK() functions? In what scenarios would you use each?*

- <u>ChatGPT Prompt:</u>

  - Describe a scenario where using ROW_NUMBER(), DENSE_RANK() in SQL would be more appropriate than using RANK().

Value Window Functions

Functions: LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE()

- Example Function: LAG()
- Explanation:
  - Value window functions allow you to access data from preceding or following rows within your window. These functions are perfect for comparing current row values with previous or next rows.
- <u>Sample Code:</u>

```sql
SELECT
    employee_id,
    department_id,
    salary,
    LAG(salary, 1) OVER (PARTITION BY department_id ORDER BY
employee_id) AS prev_salary
FROM
    employees;
```

Value Window Functions

<u>Step-by-Step Explanation:</u>

1. Select Relevant Columns:
   - employee_id, department_id, and salary are selected to display the basic details of each employee.
2. Apply the LAG() Function:
   - The LAG(salary, 1) function retrieves the salary value from the previous row within the same window. The 1 specifies that the function should look one row back.
3. Define the Window with OVER():
   - The OVER() clause defines the window of rows the function operates over.
4. Partition the Data with PARTITION BY:
   - PARTITION BY department_id divides the data into partitions by department. This means the LAG() function looks for the previous salary within the same department.
5. Order the Rows with ORDER BY:
   - ORDER BY employee_id orders the rows within each partition by employee_id, which dictates the sequence in which the LAG() function retrieves the previous salary.
6. Alias the Result:
   - AS prev_salary assigns an alias to the retrieved value, making it easy to reference in the result set.

<u>Result:</u> <u>This query compares each employee's current salary with their previous salary in the department, helping identify salary changes.</u>

Ranking Window Functions

- Sample Output

employee_id	department_id	salary	prev_salary
1	10	70000	NULL
2	10	75000	70000
3	10	80000	75000
4	20	60000	NULL
5	20	62000	60000

- Interview Question:
  - How would you use the LAG() function to identify salary changes in employees?

- ChatGPT Prompt:
  - Explain how to use the LAG() function in SQL to compare the current row with the previous row in a partition.

Sravya Madipalli ✅

Was this Helpful?

🔖 Save it

➕ Follow Me

♻️ Repost and Share it
with your friends