Sai Kumar Bysani

Guide to SQL Window Functions

@saibysani18

Swipe

@saibysani18

1. Syntax Overview

Swipe

- **function_name:** The aggregate, ranking, or analytic function.
- **OVER:** Specifies that a window function is being used.
- **PARTITION BY:** Divides the result set into partitions, and the function is applied to each partition separately.
- **ORDER BY:** Orders the rows within each partition.

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

Swipe

@saibysani18

2. Common WindowFunctions

Swipe

ROW_NUMBER()

Assigns a unique sequential integer to rows within a partition, starting at 1 for the first row in each partition.

This will assign a row number to each employee in the same department based on their salary.

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

Swipe

@saibysani18

RANK()

Ranks rows within a partition, with gaps in the ranking for ties. If two values are the same, they will receive the same rank, and the next rank will skip a number.

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

Swipe

@saibysani18

DENSE_RANK()

Similar to RANK(), but without gaps in the ranking. Tied values will share the same rank, and the next rank will continue sequentially.

```sql
SELECT
  name,
  department,
  salary,
  DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS
dense_rank
FROM employees;
```

Swipe

NTILE()

Divides the result set into a specified number of buckets and assigns a bucket number to each row.

This example divides employees into 4 salary buckets within each department.

```sql
SELECT
  name,
  department,
  salary,
  NTILE(4) OVER (PARTITION BY department ORDER BY salary DESC) AS bucket
FROM employees;
```

Swipe

@saibysani18

LAG()

Provides access to a row at a specified physical offset before the current row in the result set.
This will return the salary from the previous row for each employee.

```sql
SELECT
  name,
  department,
  salary,
  LAG(salary, 1) OVER (PARTITION BY department ORDER BY salary) AS
previous_salary
FROM employees;
```

Swipe

LEAD()

Similar to LAG(), but it accesses the next row's data.

```sql
SELECT
  name,
  department,
  salary,
  LEAD(salary, 1) OVER (PARTITION BY department ORDER BY salary) AS
next_salary
FROM employees;
```

Swipe

@saibysani18

3. Aggregate Functions with Window Functions

Swipe

@saibysani18

SUM()

Calculates the running total or sum of values within a partition.

This calculates the running total salary for employees within each department.

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

Swipe

AVG()

Calculates the average of a set of values.

This calculates the average salary of employees in each department.

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

Swipe

MIN() and MAX()

Finds the minimum or maximum value in a partition.

```sql
SELECT
  name,
  department,
  salary,
  MIN(salary) OVER (PARTITION BY department) AS min_salary,
  MAX(salary) OVER (PARTITION BY department) AS max_salary
FROM employees;
```

Swipe

@saibysani18

4. The OVER Clause: Advanced Options

Swipe

@saibysani18

ROWS/RANGE

Specifies the window frame to which the function is applied.

- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW: This window frame includes all rows from the start of the partition up to the current row.
- RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING: This includes rows within a range of values relative to the current row.

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

Swipe

5. Performance Considerations

Window functions can be computationally expensive, especially when combined with large datasets or complex partitions. Use them wisely, keeping in mind performance-tuning strategies such as:

- Indexing on PARTITION BY or ORDER BY columns
- Minimizing the number of window functions used simultaneously
- Using the appropriate window frame (ROWS vs. RANGE)

Swipe

loved this?