ADVANCE SQL TOPICS - WINDOW FUNCTIONS

by Akarshan Kapoor

---

1. MAX() - window function returns the maximum value of a column over a specified window of rows.

```
SELECT e.*,
max(salary) over(partition by dept_name) as max_salary
FROM employee e;
```

---

2. ROW_NUMBER() - window function assigns a unique sequential number to each row within a partition, starting from 1, based on the order defined in the ORDER BY clause.

```
SELECT e.*,
row_number() over(partition by dept_name order by emp_id) as rn
FROM employees e;
```

---

3. RANK() - window function assigns a ranking to each row within a partition, with the same rank for tied values. However, it *skips ranks* after ties, leaving gaps in the ranking sequence.

```
SELECT e.*,
rank() over(partition by dept_name order by salary desc) as rnk
FROM employees e;
```

---

4. DENSE_RANK() - window function assigns a ranking to each row within a partition, like RANK(), but without gaps in the ranking sequence when there are ties.

```
SELECT e.*,
dense_rank() over(partition by dept_name order by salary desc) as rnk
FROM employees e;
```

---

5. LEAD() & LAG() - window functions

LEAD(): Retrieves data from the next row within the same partition, allowing you to access a subsequent row's value.

LAG(): Retrieves data from the previous row within the same partition, allowing you to access a prior row's value.

```
SELECT e.*,
lag(salary) over(partition by dept_name order by emp_id) as prev_emp_salary
lead(salary) over(partition by dept_name order by emp_id) as next_emp_salary
FROM employees e;
```

---
important

The FRAME clause in SQL is used within window functions to define a specific range of rows to consider for the computation, relative to the current row. It specifies the subset of rows in the partition that the window function should operate on.

Components of the Frame Clause:

ROWS: Specifies a range of physical rows.

For example,
'ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW'

includes all rows from the start of the partition to the current row.

RANGE: Specifies a range of logical values (useful for numeric or date columns).

For example,
'RANGE BETWEEN 10 PRECEDING AND CURRENT ROW'

includes all rows with values within a certain range relative to the current row's value.

---

6. FIRST_VALUE and LAST_VALUE - window function

typically, column names in a database table, representing an individual's first and last name.

```
SELECT *
first_value(product_name) over(partition by product_category order by price
desc) as most expensive product
FROM product;


SELECT *
last_value(product_name) over(partition by product_category order by price
desc range between unbounded preceding and unbounded following) as least
expensive product
FROM product;
```

---

7. NTH_VALUE()- window function

Returns the nth value (based on a specified position) from the window of rows. For example, NTH_VALUE(column, 2) returns the 2nd value from the column in each partition.

```
SELECT *
nth_value(product_name,2) over(partition by product_category order by price
desc) as second most expensive product
FROM product;
```

8. NTILE() - window function (segregating data among few different groups)

Divides rows within a partition into a specified number of roughly equal-sized groups (or "buckets") and assigns a bucket number to each row. For example, NTILE(4) divides the rows into 4 buckets.

```
SELECT product_name,
CASE WHEN x.buckets = 1 then 'Expensive phone'
CASE WHEN x.buckets = 2 then 'Mid range phone'
CASE WHEN x.buckets = 3 then 'Cheaper phone' END phone_category
       ( SELECT *
         ntile(3) over(order by price desc) as buckets
         FROM product
         where product_category = 'phone';) x;
```

---

9. PERCENT_RANK() - window function

```
Calculates the relative rank of a row as a percentage of the total rows,
ranging from 0 to 1. The first row always gets 0, and the highest-ranked row
gets 1.

SELECT *,
ROUND(percent_rank() over(order by price):: numeric*100,2) as percentage_rank
FROM product;
```

---

10. CUME_DIST() - window function

Stands for cumulative distribution. It calculates the proportion of rows with values less than or equal to the current row's value, ranging between 0 and 1.

```
SELECT *,
round(cume_dist() over(order by price desc):: numeric * 100,2) as
cume_distribution percentage
FROM product;
```
