Ranking window functions

In a table where we have product and sales data, when assigned a RANK the SQL will arrange the data in descending order. After this the 1st process of ranking is assigned – Integer Based Ranking. Giving the highest number rank 1 and so on.

2nd is Percentage Based Ranking; this assigns percentage respect to the row. Scaled between 0 to 1.

When to use which method?

When finding top 20% products it is better to use continuous method. But when asked to find top 3 products better to use discrete values.

Syntax –

RANK() OVER(PARTITION BY ProductID ORDER BY Sales)

Here the expression is empty, nothing inside RANK. PARTITION BY is optional but ORDER BY is mandatory.

ROW\_NUMBER() –

Assign a unique number to each row, it doesn’t handles ties.

Example –

SELECT pizza\_id, total\_price,

ROW\_NUMBER() OVER(ORDER BY total\_price DESC) SalesRAG

FROM pizza\_sales

RANK() –

Assigns a unique number to each row, does handles ties.

Example –

SELECT pizza\_id, total\_price,

RANK() OVER(ORDER BY total\_price DESC) SalesRAG1

FROM pizza\_sales

DENSE\_RANK() –

Assigns rank to each row, handles ties and doesn’t skips rank.

Example –

SELECT pizza\_id, total\_price,

DENSE\_RANK() OVER(ORDER BY total\_price DESC) SalesRAG2

FROM pizza\_sales

Top-N analysis –

SELECT \* FROM (

SELECT pizza\_id, total\_price, order\_id,

ROW\_NUMBER() OVER(PARTITION BY order\_id ORDER BY total\_price DESC) ByProduct

FROM pizza\_sales

)t

WHERE ByProduct = 1

Bottom-N Analysis –

SELECT \* FROM (

SELECT order\_id, SUM(total\_price) Total,

ROW\_NUMBER() OVER(ORDER BY SUM(total\_price) ASC) CustRank

FROM pizza\_sales

GROUP BY order\_id

)t

WHERE CustRank <= 2

Genrate unique IDs –

SELECT

ROW\_NUMBER() OVER(ORDER BY order\_id, order\_date) UNIQUEID,

\* FROM pizza\_sales

Identify duplicates –

SELECT \* FROM (

SELECT

ROW\_NUMBER() OVER(PARTITION BY order\_id ORDER BY order\_time DESC) rn,

\* FROM pizza\_sales

)t

WHERE rn = 1

NTILE() –

Divides the rows into specific number of approx. equal groups.

Example –

SELECT pizza\_id, total\_price,

NTILE(1) OVER(ORDER BY total\_price DESC) Bucket1,

NTILE(2) OVER(ORDER BY total\_price DESC) Bucket2,

NTILE(3) OVER(ORDER BY total\_price DESC) Bucket3,

NTILE(4) OVER(ORDER BY total\_price DESC) Bucket4

FROM pizza\_sales

Use cases are data segmentation and load balancing.

SELECT \*,

CASE WHEN Bucket = 1 THEN 'High'

WHEN Bucket = 2 THEN 'Medium'

WHEN Bucket = 3 THEN 'Low'

END

FROM (

SELECT pizza\_id, total\_price,

NTILE(3) OVER(ORDER BY total\_price DESC) Bucket

FROM pizza\_sales

)t

Percentage Based Ranking –

CUME\_DIST() –

Cumulative distribution calculates distribution of data points within a window.

Example –

SELECT pizza\_id, order\_id, total\_price,

CUME\_DIST() OVER(PARTITION BY order\_id ORDER BY total\_price DESC)

FROM pizza\_sales

PERCENT\_RANK() –

Calculates the relative position of each row.

Example –

SELECT pizza\_id, order\_id, total\_price,

PERCENT\_RANK() OVER(PARTITION BY order\_id ORDER BY total\_price DESC) PerRank

FROM pizza\_sales