What are Window Functions in SQL?

Window functions perform calculations on a set of rows that are related together. But, unlike the aggregate functions, windowing functions do not collapse the result of the rows into a single value. Instead, all the rows maintain their original identity and the calculated result is returned for every row.

OVER Clause

The OVER clause signifies a window of rows over which a window function is applied. It can be used with aggregate functions, like we have used with the SUM function here, thereby turning it into a window function.

```
select *, sum(cost_to_customer_per_qty) OVER() AS Total_cost
 from customer_purchases
```

Windowing with PARTITION BY

The PARTITION BY clause is used in conjunction with the OVER clause. It breaks up the rows into different partitions. These partitions are then acted upon by the window function.

```
select *,
cost_to_customer_per_qty*quantity,sum(cost_to_customer_per_qty*quantity)
OVER(partition by vendor_id)
AS Total_cost
 from customer_purchases
```

1. Row_Number

Sometimes your dataset might not have a column depicting the sequential order of the rows.

ROW_NUMBER function is a SQL ranking function that **assigns a sequential rank number to each new record in a partition**. When the SQL Server ROW NUMBER function detects two identical values in the same partition, it assigns different rank numbers to both.

```
SELECT
        ROW_NUMBER() OVER (
                ORDER BY product_name
        ) row_num,
   product_name,
   product_size
FROM
        product
ORDER BY
        product_name;
```

Rank:

- The RANK() function is a window function that could be used in SQL Server to calculate a rank for each row within a partition of a result set.

- The same rank is assigned to the rows in a partition which have the same values.
- The rank of the first row is 1.
- The ranks may not be consecutive in the RANK() function as it adds the number of repeated rows to the repeated rank to calculate the rank of the next row.

```
SELECT
        rank() OVER (
                ORDER BY product_name
        ) rank_num,
   product_name,
   product_size
FROM
        product
ORDER BY
        product_name;
```

insert into product values (24,'Carrots','sold by weight',1,'lbs')

https://www.mysqltutorial.org/mysql-window-functions/

DENSE_RANK

- It assigns rank to each row within the partition.
- Just like the rank function, the first row is assigned rank 1 and rows having the same value have the same rank.
- The difference between RANK() and DENSE_RANK() is that in DENSE_RANK(), for the next rank after two same rank, consecutive integers are used, no rank is skipped.

NTILE Functions:

- NTILE() function in SQL Server is a window function that distributes rows of an ordered partition into a pre-defined number of roughly equal groups.
- It assigns each group a number_expression ranging from 1.
- NTILE() function assigns a number_expression for every row in a group, to which the row belongs.
- When number of rows isn't divisible by the number_expression, the NTILE() function results the groups of two sizes with the difference by one

Parameters of syntax in detail :

- **number_expression**

  The number_expression is the integer into which the rows are divided.
- **PARTITION BY clause**

  The PARTITION BY is optional, it differs the rows of a result set into partitions where the NTILE() function is used.
- **ORDER BY clause**

  The ORDER BY clause defines the order of rows in each partition where the NTILE() is used.

```
SELECT
        product_id,
        product_name,
        product_category_id,
        NTILE(5) OVER (ORDER BY product_name DESC) AS price_ntile
FROM farmers_market.product
ORDER BY product_name DESC
```

Inside product_name

```
SELECT
        product_id,
        product_name,
        product_category_id,
        NTILE(2) OVER (ORDER BY product_name DESC) AS price_ntile,
    NTILE(2) OVER (PARTITION BY product_name ORDER BY product_name
DESC) AS product_tile
FROM farmers_market.product
ORDER BY product_name DESC
```

LAG() window function

- At many instances, users would like to access data of the previous row or any row before the previous row from the current row.
- To solve this problem SQL Server's LAG() window function can be used.
- SQL Server provides a LAG() function which is very useful in case the current row values need to be compared with the data/value of the previous record or any record before the previous record.

Syntax :

```
LAG (scalar_expression [, offset] [, default])
OVER ( [ partition_by ] order_by )
```

Where :

1. **scalar_expression –**

   The value to be returned based on the specified offset.

2. **offset –**

   The number of rows back from the current row from which to obtain a value. If not specified, the default is 1.

3. **default –**

   default is the value to be returned if offset goes beyond the scope of the partition. If a default value is not specified, NULL is returned.

4. **over ([partition_by] order_by) –**

   partition_by divides the result set produced by the FROM clause into partitions to which the function is applied. If you omit the PARTITION BY clause, the function treats the whole result set as a single group. By default order_by clause sorts in ascending order.