Ranking Functions

1- ROW_NUMBER()

- Gives a unique number to every row, even if values are the same → numbers are different

```
SELECT
    OrderID,
    Sales,
    ROW_NUMBER() OVER(ORDER BY Sales DESC) AS row_num
FROM Sales.Orders;
```

2- RANK()

- Rows with the same value get the same rank, but the next RANK skipped

```
SELECT
    OrderID,
    Sales,
    RANK() OVER(ORDER BY Sales DESC) AS rank_num
FROM Sales.Orders;
```

3- DENSE_RANK()

- Same as RANK, but no skipped numbers
- Numbers stay continuous

```
SELECT
    OrderID,
    Sales,
    DENSE_RANK() OVER(ORDER BY Sales DESC) AS dense_rank_num
FROM Sales.Orders;
```

4- NTILE()

- Divides groups into equal groups
- Not really ranking – it makes buckets
- It splits rows evenly

```
SELECT
    OrderID,
    Sales,
    NTILE(2) OVER(ORDER BY Sales DESC) AS group_num
FROM Sales.Orders;
```

5- PERCENT_RANK()

- It shows the relative rank of row as percentage between 0 and 1
- It tells you “where does this row and compared to others”
- The formula like this
  - **(rank - 1) / (total_rows - 1)**
- So:
  - Lowest value → 0
  - Highest value → 1
  - Others → between 0 and 1

```
SELECT
    OrderID,
    Sales,
    PERCENT_RANK() OVER(ORDER BY Sales) AS percent_rank
FROM Sales.Orders;
```

6- LAG()

- Used to get the previous row value
- It returns a value from a previous value
- Think: give me the value before this row
- The first row get **NULL** value

```
SELECT
    OrderID,
    Sales,
    LAG(Sales) OVER(ORDER BY OrderID) AS PreviousSales
FROM Sales.Orders;
```

7- LEAD()

- It looks forward
- It returns value from next row
- Think: give me the value after this row
- The last row get **NULL** value

```
SELECT
    OrderID,
    Sales,
    LEAD(Sales) OVER(ORDER BY OrderID) AS NextSales
FROM Sales.Orders;
```

8- FIRST_VALUE()

- Returns the first value in the window
- Think: give me the first value in this group

```sql
SELECT
    OrderID,
    Sales,
    FIRST_VALUE(Sales) OVER(ORDER BY Sales DESC) AS HighestSales
FROM Sales.Orders;
```

```sql
-- Select customer, order date, and sales amount
SELECT
    CustomerID,
    OrderDate,
    Sales,

    -- LAG() gets the previous row value
    -- PARTITION BY CustomerID means:
    -- Reset the calculation for each customer separately
    -- ORDER BY OrderDate means:
    -- Look at the previous order based on date
    LAG(Sales) OVER(
        PARTITION BY CustomerID
        ORDER BY OrderDate
    ) AS PrevSale,

    -- FIRST_VALUE() returns the first value in the window
    -- For each customer, this gives their FIRST order's sales
    FIRST_VALUE(Sales) OVER(
        PARTITION BY CustomerID
        ORDER BY OrderDate
    ) AS FirstSale

-- Data is coming from the Orders table
FROM Sales.Orders;



-- Select the sales value
SELECT
    Sales,

    -- LAG(Sales) gets the previous row's sales
    -- ORDER BY OrderDate ensures we compare in time order
    -- Then we subtract:
    -- Current Sales - Previous Sales
    -- This shows growth or decline between orders
    Sales - LAG(Sales) OVER(
        ORDER BY OrderDate
    ) AS Difference

FROM Sales.Orders;
```