Window Functions-IV

🎯 Session Goals

- ✅ Understand what window functions are and when to use them.
- ✅ Break down and apply the syntax of common window functions like ROW_NUMBER(), SUM(), AVG(), etc.
- ✅ Differentiate window functions from regular aggregate functions.
- ✅ Use analytical window functions to extract advanced insights.
- ✅ Compare rows without self-joins.
- ✅ Partition and rank data meaningfully.
- ✅ Detect trends, group data into tiles, and calculate change over time.

First_Value() → Earliest

Syntax:

```
First_Value(Column) OVER(Partition BY.... ORDER BY .....)
```

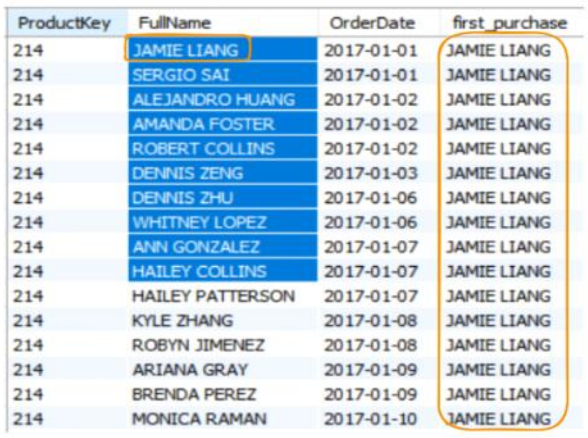
Syntax:

```
SELECT
  window_function(...) OVER (
    PARTITION BY column_name
    ORDER BY column_name
    ROWS/RANGE ...
  ) AS result_column
FROM table_name;
```

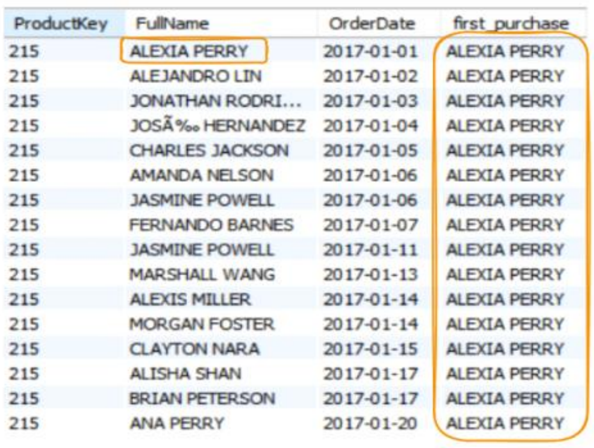
→ Unbounded

1. Find the customers who purchased each products based on the earliest date.

```
SELECT
    s.ProductKey,
    c.FullName,
    s.OrderDate,
    FIRST_VALUE(c.FullName) OVER (PARTITION BY s.ProductKey ORDER BY s.OrderDate)
    AS first_purchase
FROM `sales-2017` s
JOIN Customers c
ON c.CustomerKey = s.CustomerKey
ORDER BY s.ProductKey;
```



ProductKey	FullName	OrderDate	first_purchase
214	JAMIE LIANG	2017-01-01	JAMIE LIANG
214	SERGIO SAI	2017-01-01	JAMIE LIANG
214	ALEJANDRO HUANG	2017-01-02	JAMIE LIANG
214	AMANDA FOSTER	2017-01-02	JAMIE LIANG
214	ROBERT COLLINS	2017-01-02	JAMIE LIANG
214	DENNIS ZENG	2017-01-03	JAMIE LIANG
214	DENNIS ZHU	2017-01-06	JAMIE LIANG
214	WHITNEY LOPEZ	2017-01-06	JAMIE LIANG
214	ANN GONZALEZ	2017-01-07	JAMIE LIANG
214	HAILEY COLLINS	2017-01-07	JAMIE LIANG
214	HAILEY PATTERSON	2017-01-07	JAMIE LIANG
214	KYLE ZHANG	2017-01-08	JAMIE LIANG
214	ROBYN JIMENEZ	2017-01-08	JAMIE LIANG
214	ARIANA GRAY	2017-01-09	JAMIE LIANG
214	BRENDA PEREZ	2017-01-09	JAMIE LIANG
214	MONICA RAMAN	2017-01-10	JAMIE LIANG

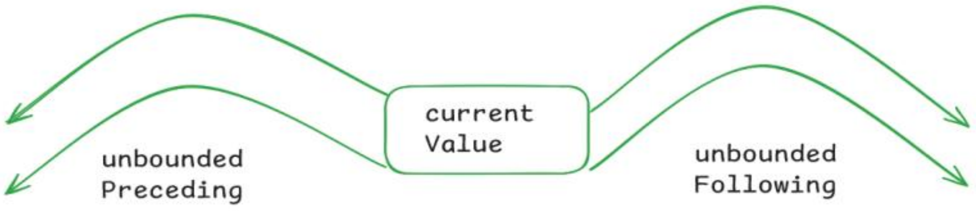


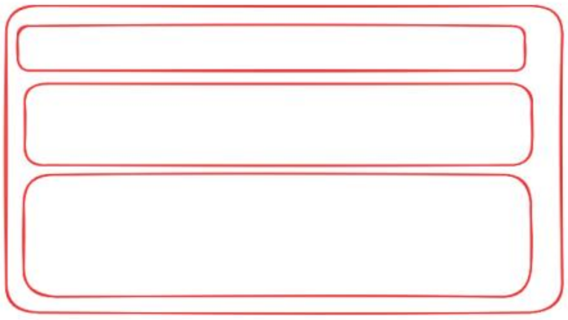
ProductKey	FullName	OrderDate	first_purchase
215	ALEXIA PERRY	2017-01-01	ALEXIA PERRY
215	ALEJANDRO LIN	2017-01-02	ALEXIA PERRY
215	JONATHAN RODRI...	2017-01-03	ALEXIA PERRY
215	JOSÃ‰ HERNANDEZ	2017-01-04	ALEXIA PERRY
215	CHARLES JACKSON	2017-01-05	ALEXIA PERRY
215	AMANDA NELSON	2017-01-06	ALEXIA PERRY
215	JASMINE POWELL	2017-01-06	ALEXIA PERRY
215	FERNANDO BARNES	2017-01-07	ALEXIA PERRY
215	JASMINE POWELL	2017-01-11	ALEXIA PERRY
215	MARSHALL WANG	2017-01-13	ALEXIA PERRY
215	ALEXIS MILLER	2017-01-14	ALEXIA PERRY
215	MORGAN FOSTER	2017-01-14	ALEXIA PERRY
215	CLAYTON NARA	2017-01-15	ALEXIA PERRY
215	ALISHA SHAN	2017-01-17	ALEXIA PERRY
215	BRIAN PETERSON	2017-01-17	ALEXIA PERRY
215	ANA PERRY	2017-01-20	ALEXIA PERRY

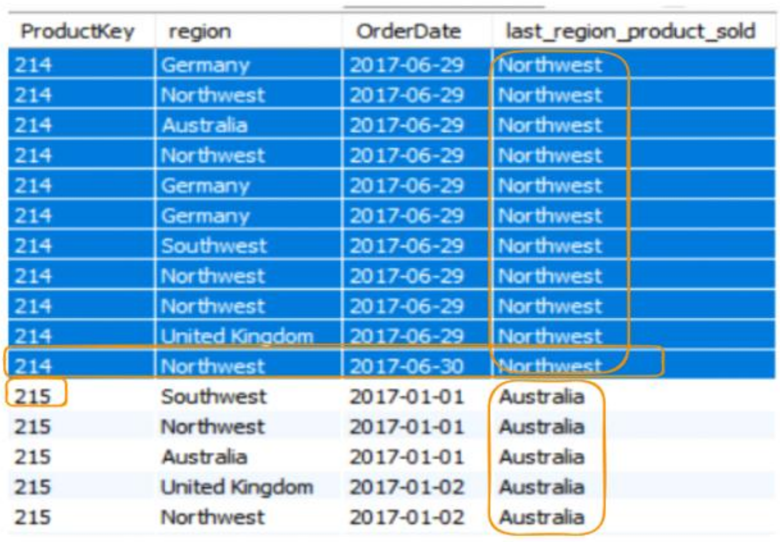
LAST_VALUE() → Latest Record

UNBOUNDED PRECEDING &
UNBOUNDED FOLLOWING

Find the last region where each product was sold based on the latest order date?







ProductKey	region	OrderDate	last_region_product_sold
214	Germany	2017-06-29	Northwest
214	Northwest	2017-06-29	Northwest
214	Australia	2017-06-29	Northwest
214	Northwest	2017-06-29	Northwest
214	Germany	2017-06-29	Northwest
214	Germany	2017-06-29	Northwest
214	Southwest	2017-06-29	Northwest
214	Northwest	2017-06-29	Northwest
214	Northwest	2017-06-29	Northwest
214	United Kingdom	2017-06-29	Northwest
214	Northwest	2017-06-30	Northwest
215	Southwest	2017-01-01	Australia
215	Northwest	2017-01-01	Australia
215	Australia	2017-01-01	Australia
215	United Kingdom	2017-01-02	Australia
215	Northwest	2017-01-02	Australia

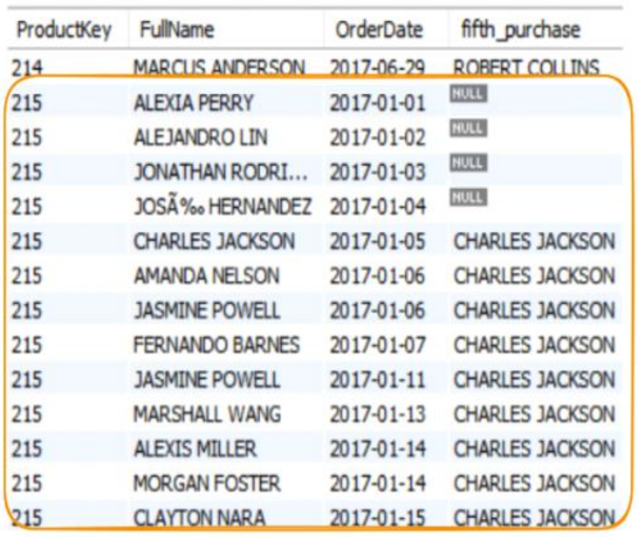
```sql
SELECT
    s.ProductKey,
    t.region,
    s.OrderDate,
    LAST_VALUE(t.region) OVER(PARTITION BY s.ProductKey ORDER BY s.OrderDate
    RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
    ) AS last_region_product_sold
FROM territories t
JOIN `sales-2017` s
ON t.SalesTerritoryKey = s.TerritoryKey
ORDER BY s.ProductKey;
```

NTH-VALUE()

```
NTH-VALUE(Column,Nth) OVER(PARTITION BY ..... ORDER BY ..... RANGES .....)
```

From the Sales Record, Retrieve the 5th Customer from every product using Nth_value.

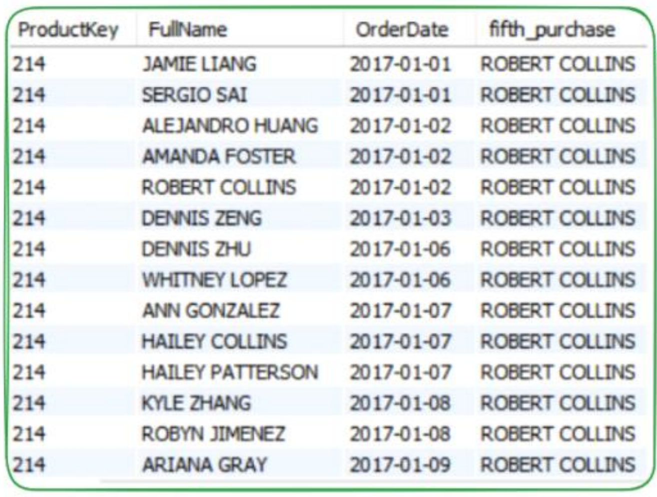
```sql
SELECT
    s.ProductKey,
    c.FullName,
    s.OrderDate,
    NTH_VALUE(c.FullName, 5) OVER
    (PARTITION BY  s.ProductKey ORDER BY s.OrderDate) AS fifth_purchase
FROM `sales-2017` s
JOIN Customers c
ON c.CustomerKey = s.CustomerKey
ORDER BY s.ProductKey;
```



ProductKey	FullName	OrderDate	fifth_purchase
214	MARCUS ANDERSON	2017-06-29	ROBERT COLLINS
215	ALEXIA PERRY	2017-01-01	NULL
215	ALEJANDRO LIN	2017-01-02	NULL
215	JONATHAN RODRI...	2017-01-03	NULL
215	JOSÃ‰ HERNANDEZ	2017-01-04	NULL
215	CHARLES JACKSON	2017-01-05	CHARLES JACKSON
215	AMANDA NELSON	2017-01-06	CHARLES JACKSON
215	JASMINE POWELL	2017-01-06	CHARLES JACKSON
215	FERNANDO BARNES	2017-01-07	CHARLES JACKSON
215	JASMINE POWELL	2017-01-11	CHARLES JACKSON
215	MARSHALL WANG	2017-01-13	CHARLES JACKSON
215	ALEXIS MILLER	2017-01-14	CHARLES JACKSON
215	MORGAN FOSTER	2017-01-14	CHARLES JACKSON
215	CLAYTON NARA	2017-01-15	CHARLES JACKSON

Without unbounded preceding and unbounded following, you can't insert the previous null values with the exact customer Name, who purchased the particular products at 5th number.

```
SELECT
    s.ProductKey,
    c.FullName,
    s.OrderDate,
    NTH_VALUE(c.FullName, 5) OVER (PARTITION BY  s.ProductKey ORDER BY s.OrderDate
    RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS fifth_purchase
FROM `sales-2017` s
JOIN Customers c
ON c.CustomerKey = s.CustomerKey
ORDER BY s.ProductKey;
```



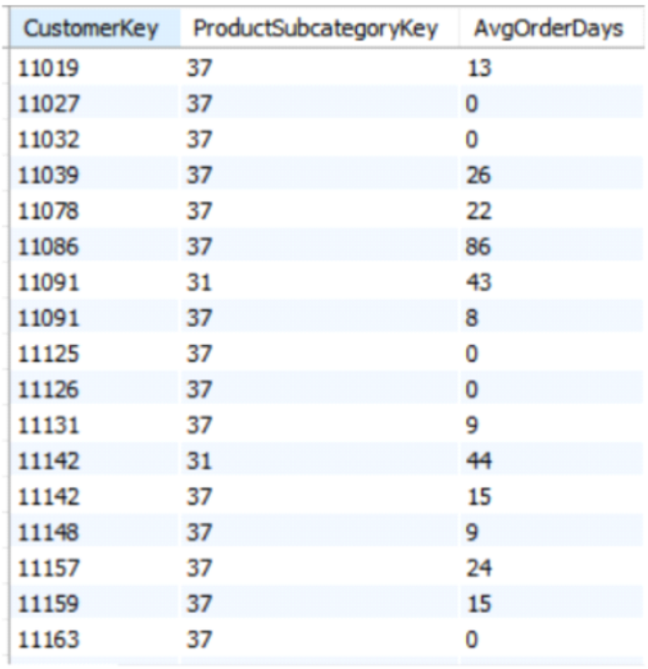
ProductKey	FullName	OrderDate	fifth_purchase
214	JAMIE LIANG	2017-01-01	ROBERT COLLINS
214	SERGIO SAI	2017-01-01	ROBERT COLLINS
214	ALEJANDRO HUANG	2017-01-02	ROBERT COLLINS
214	AMANDA FOSTER	2017-01-02	ROBERT COLLINS
214	ROBERT COLLINS	2017-01-02	ROBERT COLLINS
214	DENNIS ZENG	2017-01-03	ROBERT COLLINS
214	DENNIS ZHU	2017-01-06	ROBERT COLLINS
214	WHITNEY LOPEZ	2017-01-06	ROBERT COLLINS
214	ANN GONZALEZ	2017-01-07	ROBERT COLLINS
214	HAILEY COLLINS	2017-01-07	ROBERT COLLINS
214	HAILEY PATTERSON	2017-01-07	ROBERT COLLINS
214	KYLE ZHANG	2017-01-08	ROBERT COLLINS
214	ROBYN JIMENEZ	2017-01-08	ROBERT COLLINS
214	ARIANA GRAY	2017-01-09	ROBERT COLLINS

🔶 Question 1: Avg Days Between Orders for Same Subcategory

🧩 Problem Statement :

For each customer and product subcategory, calculate the average number of days between consecutive orders.

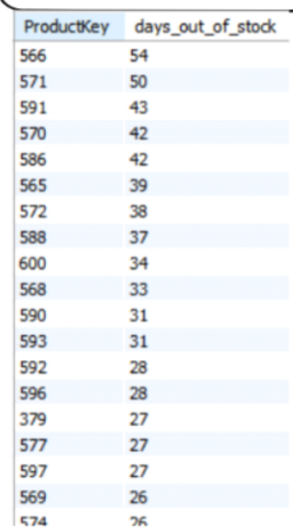
```
WITH subcategory_orders AS(
    SELECT
        s.CustomerKey,
        p.ProductSubcategoryKey,
        s.OrderDate,
        LAG(s.OrderDate) OVER(
            PARTITION BY s.CustomerKey, p.ProductSubcategoryKey ORDER BY s.OrderDate
    ) AS prevOrderDate
    FROM products p
    JOIN `sales-2017` s
    ON p.ProductKey = s.ProductKey
)
SELECT
    CustomerKey,
    ProductSubcategoryKey,
    ROUND(AVG(DATEDIFF(OrderDate,prevOrderDate)),0) AS AvgOrderDays
FROM subcategory_orders
WHERE prevOrderDate IS NOT NULL
GROUP BY CustomerKey, ProductSubcategoryKey
HAVING COUNT(*) > 1;
```



CustomerKey	ProductSubcategoryKey	AvgOrderDays
11019	37	13
11027	37	0
11032	37	0
11039	37	26
11078	37	22
11086	37	86
11091	31	43
11091	37	8
11125	37	0
11126	37	0
11131	37	9
11142	31	44
11142	37	15
11148	37	9
11157	37	24
11159	37	15
11163	37	0

🔶 Question 2: Longest Out-of-Stock Duration
🧩 Problem Statement :
Find the product that remained out of stock for the longest time and calculate how many days it was unavailable.

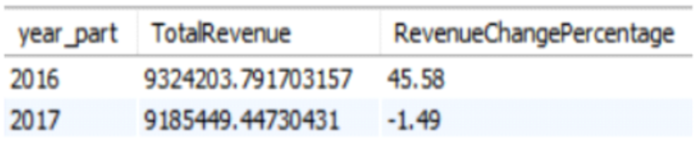
```
WITH stock_changes AS (
    SELECT
        ProductKey,
        StockDate,
        LEAD(StockDate) OVER (PARTITION BY ProductKey ORDER BY StockDate)
        AS NextStockDate
    FROM `sales-2017`
)
SELECT
    ProductKey,
    MAX(DATEDIFF(NextStockDate,StockDate)) AS days_out_of_stock
FROM stock_changes
WHERE NextStockDate IS NOT NULL
GROUP BY ProductKey
ORDER BY days_out_of_stock DESC;
```



ProductKey	days_out_of_stock
566	54
571	50
591	43
570	42
586	42
565	39
572	38
588	37
600	34
568	33
590	31
593	31
592	28
596	28
379	27
577	27
597	27
569	26
574	26

🔶 Question 3: Yearly Sales Trend with % Change
🧩 Problem Statement :
Show total sales per year and calculate the percentage growth or decline from the previous year.

```
WITH Yearly_sales AS (
    SELECT
        YEAR(s.OrderDate) As year_part,
        SUM(s.OrderQuantity * p.ProductPrice) AS TotalRevenue
    FROM (
        SELECT * FROM `sales-2015`
        UNION ALL
        SELECT * FROM `sales-2016`
        UNION ALL
        SELECT * FROM `sales-2017`
    ) AS s
    JOIN Products p
    ON p.ProductKey = s.ProductKey
    GROUP BY YEAR(OrderDate)
),
SalesTrend AS (
        SELECT
            year_part,
            TotalRevenue,
            LAG(TotalRevenue) OVER (ORDER BY year_part) AS PrevRevenue
        FROM Yearly_sales
)
SELECT
    year_part,
    TotalRevenue,
    ROUND((TotalRevenue - PrevRevenue)/PrevRevenue * 100.0, 2) AS RevenueChangePercentage
FROM SalesTrend
WHERE PrevRevenue IS NOT NULL;
```



year_part	TotalRevenue	RevenueChangePercentage
2016	9324203.791703157	45.58
2017	9185449.44730431	-1.49

🔶 Question 4: Row-wise Value Insights
🧩 Problem Statement :
For each sale, show the first, last, and third sale amount within its product's transaction history.