Database Using SQL Server & Raw Data available over my GitHub Profile

MySQL

PostgreSQL

15 Most Commonly Used Window Function Queries in the Industry

AnshLibrary

What is a Window Function

A **window function** performs a calculation **across a window (a group of rows)** defined by the **OVER()** clause.

Syntax of Window Functions :

```
<window_function>() OVER (
    [PARTITION BY column]
    [ORDER BY column]
    [ROWS/RANGE clause]
)
```

Types of Window Functions in SQL Server

1. Aggregate Window Functions

EX:- **SUM( ) , AVG( ), COUNT( ), MAX ( ), Mini ( )**

2. Ranking Window Functions

EX:- **ROW_NUMBER( ), RANK( ), DENSE_RANK( ), NTILE(n)**

3. **Value Window Functions**

EX:- FIRST_VALUE( ), LAST_VALUE( ), LEAD( ), LAG( )

4. Statistical/Offset & Navigation Functions (Advanced Use)

EX:- CUME_DIST( ) , PERCENT_RANK ( )

Common Clauses Used With Window Functions

1. PARTITION BY
Divides the result set into **groups (like GROUP BY)** but **retains all rows**.

2. ORDER BY
Specifies the **order** of rows in each partition for the window function.

```
RANK( ) OVER (PARTITION BY Department ORDER BY Salary DESC)
```

3. ROWS BETWEEN / RANGE BETWEEN
Used to define a **frame** of rows relative to the current row.

```
<function>( ) OVER (
  [PARTITION BY col]
  [ORDER BY col]
  [ROWS | RANGE] BETWEEN <frame_start> AND <frame_end>
)
```

Frame Modes :

1. ROWS Frame

→ **Operates by physical position of rows**
→ **Frame includes specific number of rows before or after the current row.**

2. RANGE Frame

→ Operates **by logical value** in ORDER BY column
→ All rows with the same value as the current row in the ORDER BY column are **treated as one**.

3. Frame Boundaries (Frame Types)

These are used inside ROWS BETWEEN ... AND ... or RANGE BETWEEN ... AND ....

Frame Type / Boundary	Meaning
UNBOUNDED PRECEDING	From the first row in the partition
n PRECEDING	From n rows before the current row
CURRENT ROW	From or to the current row itself
n FOLLOWING	From n rows after the current row
UNBOUNDED FOLLOWING	Until the last row in the partition

01

Track Customer Purchase Journey

```
SELECT

    CustomerID,
    OrderID,
    OrderDate,
    ROW_NUMBER()
OVER(PARTITION BY CustomerID ORDER BY
OrderDate) AS OrderSequence

FROM Orders;
```

Use Case : Marketing team wants to personalize messages based on customer's order sequence.

AnshLibrary

02

Identify Top 5 Customers by Lifetime Spend

```
SELECT TOP 5

    CustomerID,
    SUM(Sales) AS LifetimeSpend,
    RANK() OVER(ORDER BY SUM(Sales) DESC) AS
SpendingRank

FROM Orders
GROUP BY CustomerID;
```

Use case: Sales team wants to prioritize high-value clients.

AnshLibrary

03

Compare Sales Staff Performance by Order Count

```
SELECT

    SalesPersonID,
    COUNT(*) AS TotalOrders,
    DENSE_RANK() OVER (ORDER BY
COUNT(*) DESC) AS PerformanceRank

FROM Orders
GROUP BY SalesPersonID;
```

***Use case*:** HR needs quarterly appraisals based on order count.

AnshLibrary

04

Customer Segmentation Based on Purchase Value

```
SELECT

    CustomerID,
    SUM(Sales) AS  TotalSales,
    NTILE(3) OVER(ORDER BY
SUM(Sales) DESC) AS
CustomerSegment

FROM Orders
GROUP BY CustomerID;
```

Use case: Segment customers into top/mid/low tiers for targeting.

AnshLibrary

05

Previous Order Value – LTV Trends

```
SELECT
    CustomerID,
    OrderID,
    Sales,
    LAG(Sales) OVER(PARTITION BY CustomerID
ORDER BY OrderDate) AS PrevOrderValue
FROM Orders;
```

***Use case*: **Understand if a customer is increasing or decreasing spending.

AnshLibrary

06

Identify Future Order Trends per Customer

```
SELECT

    CustomerID,
    OrderID,
    Sales,
    LEAD(Sales) OVER(PARTITION BY CustomerID
ORDER BY OrderDate) AS NextOrderValue

FROM Orders;
```

***Use case*:** Predict future sales pattern.

AnshLibrary

07

First Product Ever Bought by Each Customer

```
SELECT

    CustomerID,
    FIRST_VALUE(ProductID) OVER(PARTITION
BY CustomerID ORDER BY OrderDate) AS
FirstPurchasedProduct

FROM Orders;
```

Use case: Loyalty program insights.

AnshLibrary

08

Most Recent Product Purchased

```
SELECT

    CustomerID,
    LAST_VALUE(ProductID) OVER(
        PARTITION BY CustomerID ORDER BY
                      OrderDate
        ROWS BETWEEN
       UNBOUNDED PRECEDING  AND
                      UNBOUNDED FOLLOWING
    ) AS LastPurchasedProduct

FROM Orders;
```

Use case: Re-marketing or showing “You last bought…” recommendations.

AnshLibrary

09

Running Total of Customer Spending Over Time

```
SELECT

    CustomerID,
    OrderID,
    OrderDate,
    Sales,
    SUM(Sales) OVER(PARTITION BY CustomerID
ORDER BY OrderDate) AS CumulativeSpend

FROM Orders;
```

***Use case*:** Revenue progression tracking in dashboards.

AnshLibrary

10

Moving Average of Sales Over Last 3 Orders

```
SELECT

    CustomerID,
    OrderID,
    Sales,
    AVG(Sales) OVER(PARTITION BY CustomerID
ORDER BY OrderDate ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) AS
MovingAvgSales

FROM Orders;
```

Use case: Smoothen sales trend and detect spikes/dips.

AnshLibrary

11

Compare Customer Sales from Orders vs OrdersArchive (Past + Present)

```
WITH CombinedOrders AS (
    SELECT CustomerID, Sales, OrderDate
FROM Orders
    UNION ALL
    SELECT CustomerID, Sales, OrderDate
FROM OrdersArchive
)
SELECT

    CustomerID,
    OrderDate,
    Sales,
    SUM(Sales) OVER(PARTITION BY
CustomerID ORDER BY OrderDate) AS
CumulativeSales

FROM CombinedOrders;
```

Use case: Unified revenue tracking.

AnshLibrary

12
Employee Distribution by Orders Handled

```
SELECT

    SalesPersonID,
    COUNT(OrderID) AS TotalOrders,
    CUME_DIST() OVER(ORDER BY
COUNT(OrderID)) AS OrderLoadPercentile

FROM Orders
GROUP BY SalesPersonID;
```

***Use case*:** Fair resource planning for delivery/sales teams.

AnshLibrary