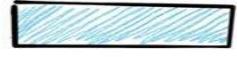


Report

| | |
|---------------|--|
| Germany |  |
| United States |  |
| Italy |  |
| France |  |

While generating reports we do not have enough space to store full name of values so we need abbreviations short form of values
So we used case statement to match full value with abbebiated values of abbrribiated value with full vale

Report

| | |
|----|---|
| DE |  |
| US |  |
| IT |  |
| FR |  |

Object Explorer

```
-- Retrieve customers details with abbreviated country code
SELECT
    CustomerID,
    FirstName,
    LastName,
    Country
FROM Sales.Customers;

SELECT DISTINCT Country
FROM Sales.Customers;
```

Results

| | CustomerID | FirstName | LastName | Country |
|---|------------|-----------|----------|---------|
| 1 | 1 | Jossef | Goldberg | Germany |
| 2 | 2 | Kevin | Brown | USA |
| 3 | 3 | Mary | NULL | USA |
| 4 | 4 | Mark | Schwarz | Germany |
| 5 | 5 | Anna | Adams | USA |

| | Country |
|---|---------|
| 1 | Germany |
| 2 | USA |



DESKTOP-B4B8QBU\SOLEXPRESS

Query executed successfully.

Object Explorer

```
-- Retrieve customers details with abbreviated country code
SELECT
    CustomerID,
    FirstName,
    LastName,
    Country,
    CASE
        WHEN Country = 'Germany' THEN 'DE'
        WHEN Country = 'USA' THEN 'US'
        ELSE 'n/a'
    END CountryAbbr
FROM Sales.Customers
```

Results

| | CustomerID | FirstName | LastName | Country | CountryAbbr |
|---|------------|-----------|----------|---------|-------------|
| 1 | 1 | Jossef | Goldberg | Germany | DE |
| 2 | 2 | Kevin | Brown | USA | USA |
| 3 | 3 | Mary | NULL | USA | US |
| 4 | 4 | Mark | Schwarz | Germany | DE |
| 5 | 5 | Anna | Adams | USA | US |



DESKTOP-B4B8QBU\SOLEXPRESS ... DESKTOP-B4B8QBU\YouTube

Query executed successfully.

Another example -

```

CASE
    WHEN Country = 'Germany' THEN 'DE'
    WHEN Country = 'India' THEN 'IN'
    WHEN Country = 'United States' THEN 'US'
    WHEN Country = 'France' THEN 'FR'
    WHEN Country = 'Italy' THEN 'IT'
    ELSE 'n/a'
END

```

Full Form

Column Name
to be evaluated (only one)

Column Value
To be compared

```

CASE Country
    WHEN 'Germany' THEN 'DE'
    WHEN 'India' THEN 'IN'
    WHEN 'United States' THEN 'US'
    WHEN 'France' THEN 'FR'
    WHEN 'Italy' THEN 'IT'
    ELSE 'n/a'
END

```

Quick Form





HANDLING NULLS

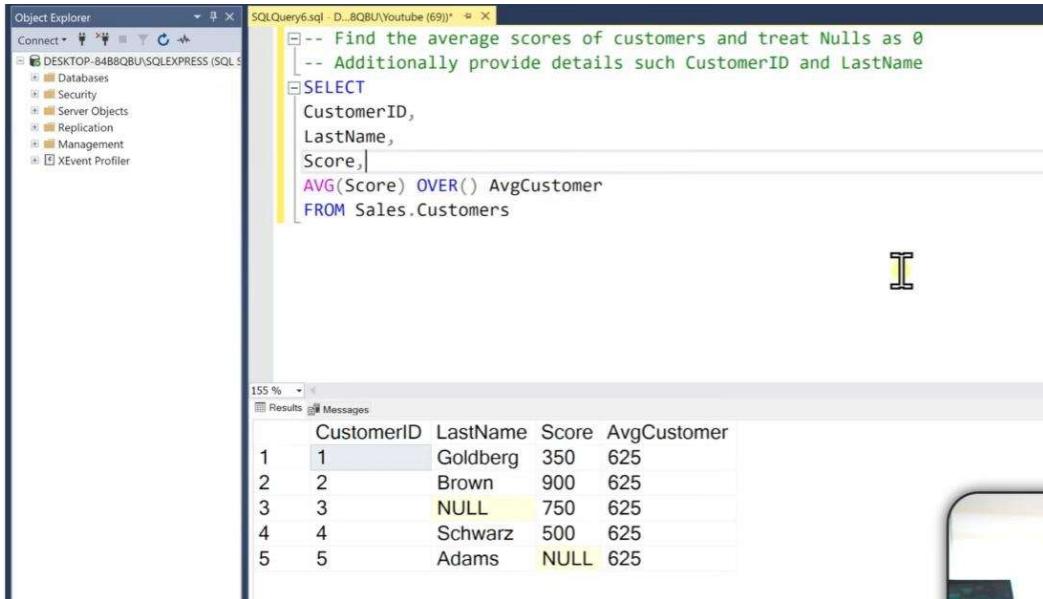
Replace NULLs with a specific value.

NULLs can lead to inaccurate results,
which can lead to wrong decision-making.

SQL TASK

Find the average scores of customers and treat Nulls as 0

And additionally provide details such CustomerID & LastName



The screenshot shows the SQL Server Management Studio interface. On the left is the Object Explorer pane, which is mostly empty except for a connection to 'DESKTOP-B4BBQBU\SQLEXPRESS (SQL Server)'. In the center is a query window titled 'SQLQuery6.sql - D:\QB\BU\Youtube (69)*'. The query is:

```
-- Find the average scores of customers and treat Nulls as 0
-- Additionally provide details such CustomerID and LastName
SELECT
    CustomerID,
    LastName,
    Score,
    AVG(Score) OVER() AvgCustomer
FROM Sales.Customers
```

Below the query window is a results grid. The columns are 'CustomerID', 'LastName', 'Score', and 'AvgCustomer'. The data is as follows:

| | CustomerID | LastName | Score | AvgCustomer |
|---|------------|----------|-------|-------------|
| 1 | 1 | Goldberg | 350 | 625 |
| 2 | 2 | Brown | 900 | 625 |
| 3 | 3 | NULL | 750 | 625 |
| 4 | 4 | Schwarz | 500 | 625 |
| 5 | 5 | Adams | NULL | 625 |

Null - missing info in above example

Our business needs null - 0 so that it considers 0 not missing info so that the avg will also change

SQL Server Management Studio (SSMS) interface showing two queries and their results.

Query 1:

```
-- Find the average scores of customers and treat Nulls as 0
-- Additionally provide details such CustomerID and LastName
SELECT
    CustomerID,
    LastName,
    Score,
    CASE
        WHEN Score IS NULL THEN 0
        ELSE Score
    END ScoreClean,
    AVG(Score) OVER() AvgCustomer
FROM Sales.Customers
```

Results:

| | CustomerID | LastName | Score | ScoreClean | AvgCustomer |
|---|------------|----------|-------|------------|-------------|
| 1 | 1 | Goldberg | 350 | 350 | 625 |
| 2 | 2 | Brown | 900 | 900 | 625 |
| 3 | 3 | NULL | 750 | 750 | 625 |
| 4 | 4 | Schwarz | 500 | 500 | 625 |
| 5 | 5 | Adams | NULL | 0 | 625 |



Query 2:

```
-- Find the average scores of customers and treat Nulls as 0
-- Additionally provide details such CustomerID and LastName
SELECT
    CustomerID,
    LastName,
    Score,
    CASE
        WHEN Score IS NULL THEN 0
        ELSE Score
    END ScoreClean,
    AVG(CASE
        WHEN Score IS NULL THEN 0
        ELSE Score
    END) OVER () AvgCustomerClean,
    AVG(Score) OVER() AvgCustomer
FROM Sales.Customers
```

Results:

| | CustomerID | LastName | Score | ScoreClean | AvgCustomerClean | AvgCustomer |
|---|------------|----------|-------|------------|------------------|-------------|
| 1 | 1 | Goldberg | 350 | 350 | 500 | 625 |
| 2 | 2 | Brown | 900 | 900 | 500 | 625 |
| 3 | 3 | NULL | 750 | 750 | 500 | 625 |
| 4 | 4 | Schwarz | 500 | 500 | 500 | 625 |
| 5 | 5 | Adams | NULL | 0 | 500 | 625 |



CONDITIONAL AGGREGATION

Apply aggregate functions only on
subsets of data that fulfill certain conditions.

Analyze target on specific subset of data

FLAG

```
-- Count how many times each customer has made an order with sales greater than 30
SELECT
    OrderID,
    CustomerID,
    Sales
FROM Sales.Orders
ORDER BY CustomerID
```

Binary indicator (1,0) to be summarized to show how many times the condition is true

| | OrderID | CustomerID | Sales |
|----|---------|------------|-------|
| 1 | 3 | 1 | 20 |
| 2 | 4 | 1 | 60 |
| 3 | 7 | 1 | 30 |
| 4 | 1 | 2 | 10 |
| 5 | 5 | 2 | 25 |
| 6 | 9 | 2 | 20 |
| 7 | 10 | 3 | 60 |
| 8 | 6 | 3 | 50 |
| 9 | 2 | 3 | 15 |
| 10 | 8 | 4 | 90 |



STEP-2

```
-- Count how many times each customer has made an order with sales greater than 30
SELECT
    OrderID,
    CustomerID,
    Sales,
    CASE
        WHEN Sales > 30 THEN 1
        ELSE 0
    END SalesFlag
FROM Sales.Orders
ORDER BY CustomerID
```

| | CustomerID | Sales | SalesFlag | |
|----|------------|-------|-----------|---|
| | | 20 | 0 | |
| | | 60 | 1 | |
| 3 | 7 | 1 | 30 | 0 |
| 4 | 1 | 2 | 10 | 0 |
| 5 | 5 | 2 | 25 | 0 |
| 6 | 9 | 2 | 20 | 0 |
| 7 | 10 | 3 | 60 | 1 |
| 8 | 6 | 3 | 50 | 1 |
| 9 | 2 | 3 | 15 | 0 |
| 10 | 8 | 4 | 90 | 1 |

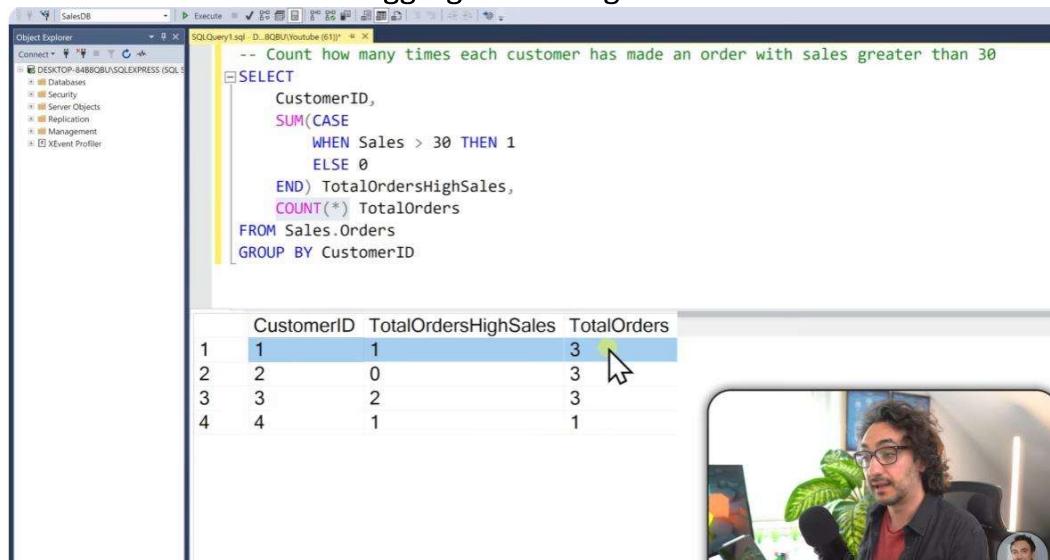


```
-- Count how many times each customer has made an order with sales greater than 30
SELECT
    CustomerID,
    SUM(CASE
        WHEN Sales > 30 THEN 1
        ELSE 0
    END) TotalOrders
FROM Sales.Orders
GROUP BY CustomerID
```

| | CustomerID | TotalOrders |
|---|------------|-------------|
| 1 | 1 | 1 |
| 2 | 2 | 0 |
| 3 | 3 | 2 |
| 4 | 4 | 1 |



Above is conditional Aggregation
And below is conditional aggregation using case statements



```
-- Count how many times each customer has made an order with sales greater than 30
SELECT
    CustomerID,
    SUM(CASE
        WHEN Sales > 30 THEN 1
        ELSE 0
    END) TotalOrdersHighSales,
    COUNT(*) TotalOrders
FROM Sales.Orders
GROUP BY CustomerID
```

| | CustomerID | TotalOrdersHighSales | TotalOrders |
|---|------------|----------------------|-------------|
| 1 | 1 | 1 | 3 |
| 2 | 2 | 0 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 1 | 1 |



CASE STATEMENT

Evaluates a list of conditions and returns a value when the first Condition is met.

Rules The data type of the results must be matching

USE CASES

Derive New Information,

Categorizing Data

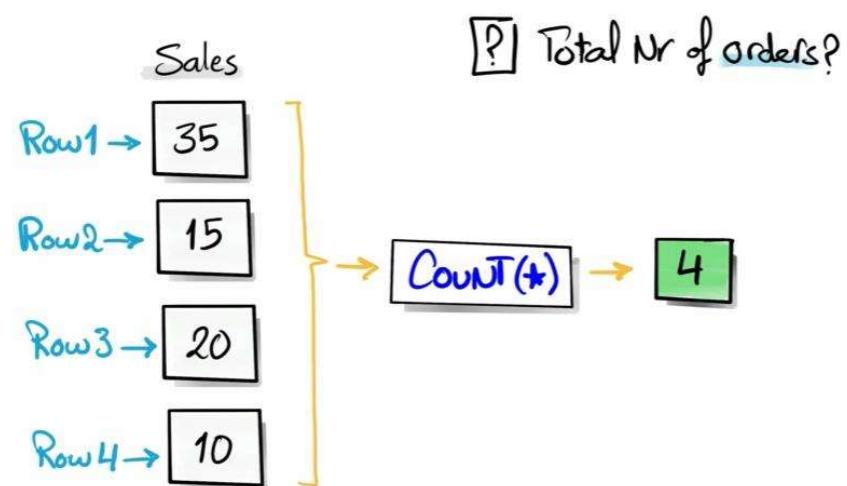
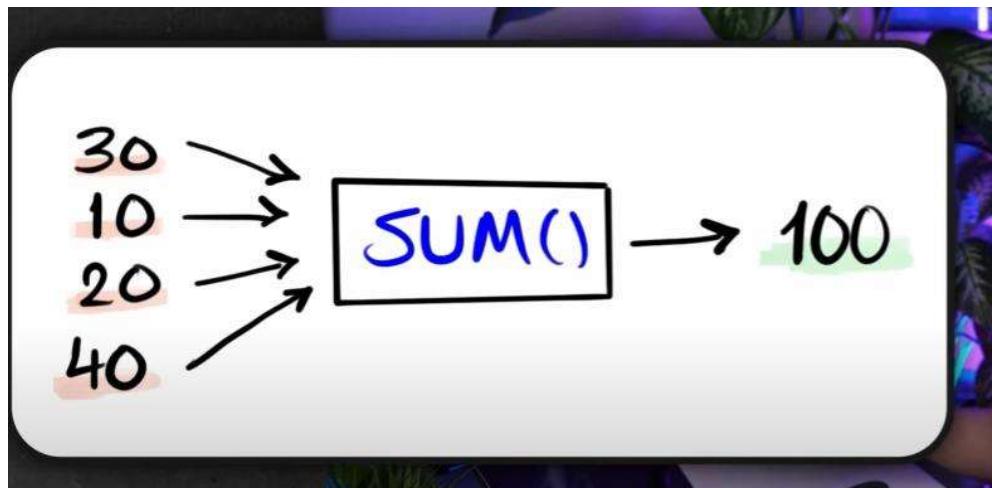
Mapping Values

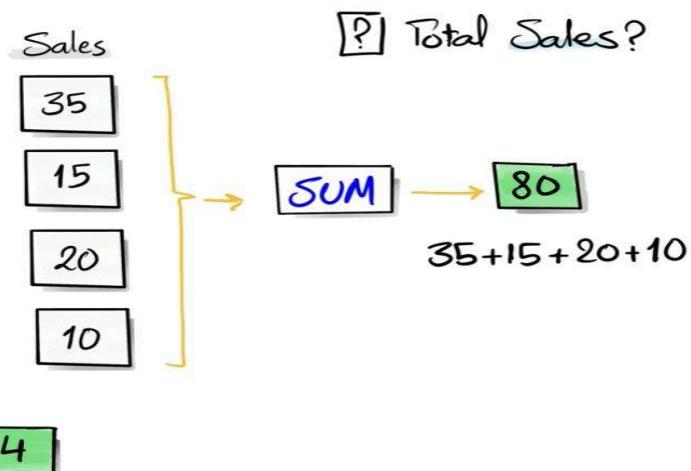
Handling NULLs

Conditional Aggregations



Aggregate Functions: Takes multiple rows value as an input and output gonna one single value





Sales

35
15
20
10

?

Find Average Sales?

AVG → 20

$$\frac{35+15+20+10}{4}$$

Count(*) 4

Sum 80

Sales

35
15
20
10

?

Find The Highest Sales

MAX → 35

Count(*) 4

Sum 80

?

Find The lowest Sales

Sales

35
15
20
→ 10

MIN → 10

Count(*) 4

Sum 80

Avg 20

Max 35

Sales

35

15

20

10

Aggregate Functions

COUNT(*) 4

SUM 80

Avg 20

MAX 35

MIN 10

-- Find the total number of orders

```
SELECT  
COUNT(*) AS total_nr_orders  
FROM orders
```

| Results | |
|---------|----------------------|
| 1 | total_nr_orders 4 |

-- Find the total sales of all orders

```
SELECT  
COUNT(*) AS total_nr_orders,  
SUM(sales) AS total_sales  
FROM orders
```

| Results | |
|---------|----------------------|
| 1 | total_nr_orders 4 |

| | total_nr_orders | total_sales |
|---|-----------------|-------------|
| 1 | 4 | 80 |

```
-- Find the average sales of all orders
SELECT
COUNT(*) AS total_nr_orders,
SUM(sales) AS total_sales,
AVG(sales) AS avg_sales
FROM orders
```

Results Messages

| | total_nr_orders | total_sales | avg_sales |
|---|-----------------|-------------|-----------|
| 1 | 4 | 80 | 20 |

-- Find the lowest sales of all orders

```
SELECT
COUNT(*) AS total_nr_orders,
SUM(sales) AS total_sales,
AVG(sales) AS avg_sales,
MAX(sales) AS highest_sales,
MIN(sales) AS lowest_sales
FROM orders
```

Results Messages

| | total_nr_orders | total_sales | avg_sales | highest_sales | lowest_sales |
|---|-----------------|-------------|-----------|---------------|--------------|
| 1 | 4 | 80 | 20 | 35 | 10 |

Each customer data using group by

```
-- Find the lowest sales of all orders
SELECT
customer_id,
COUNT(*) AS total_nr_orders,
SUM(sales) AS total_sales,
AVG(sales) AS avg_sales,
MAX(sales) AS highest_sales,
MIN(sales) AS lowest_sales
FROM orders
GROUP BY customer_id
```



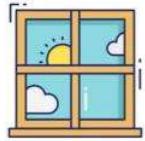
Results Messages

| | customer_id | total_nr_orders | total_sales | avg_sales | highest_sales | lowest_sales |
|---|-------------|-----------------|-------------|-----------|---------------|--------------|
| 1 | 1 | 1 | 35 | 35 | 35 | 35 |
| 2 | 2 | 1 | 15 | 15 | 15 | 15 |
| 3 | 3 | 1 | 20 | 20 | 20 | 20 |
| 4 | 6 | 1 | 10 | 10 | 10 | 10 |

| | | | | |
|--|---|----|----|---|
| | 1 | 35 | 35 | 3 |
| ✓ | 1 | 15 | 15 | 1 |
| Analyze the scores in customers table | | | | |
| ✓ | 1 | 10 | 10 | 1 |
| ✓ | 1 | 1 | 1 | 1 |

Task and group by data by country

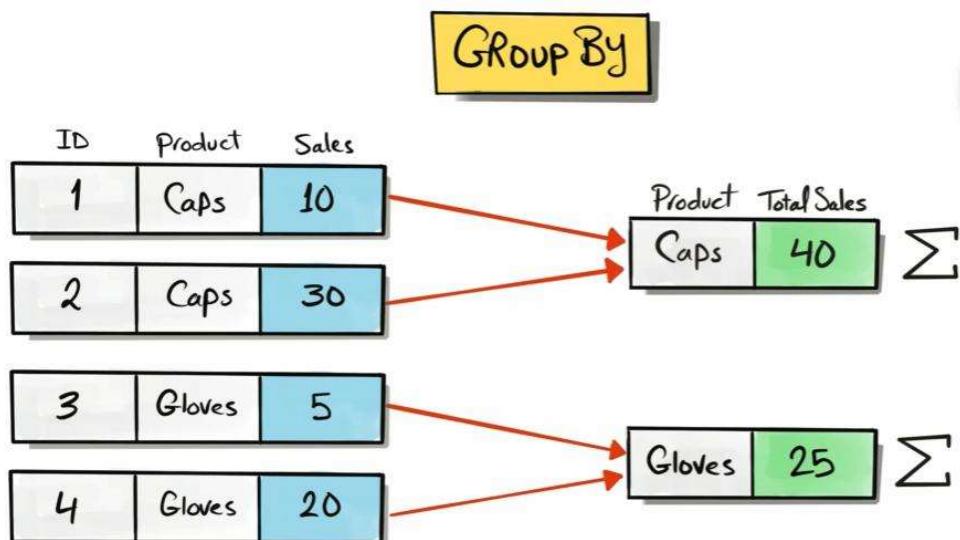
Window Functions: similar to group by but difference without losing level of details of rows



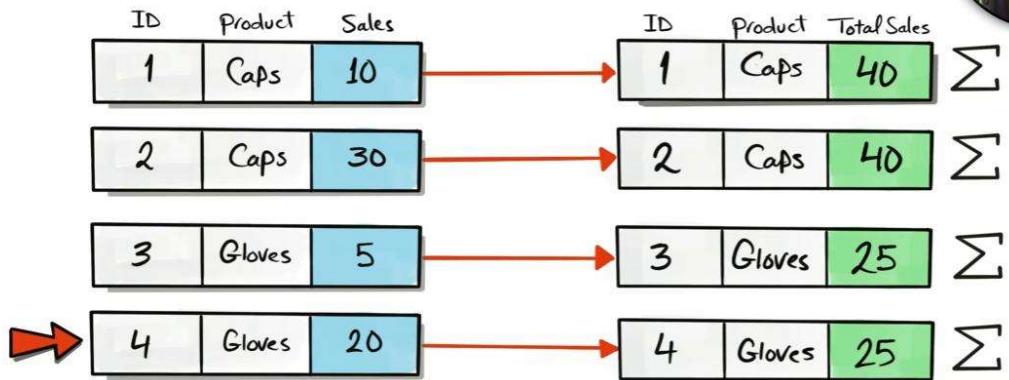
WINODW FUNCTIONS

**Perform calculations (e.g. aggregation)
on a specific subset of data,
without losing the level of details of rows.**

Order ID controlling level of details in input and product is controlling level of details in output



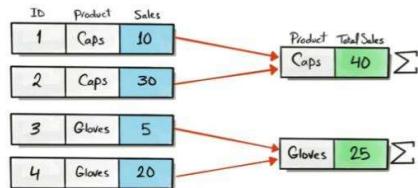
WINDOW



Row-Level Calculation

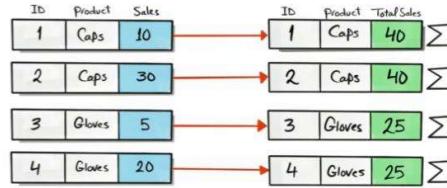
Difference between group by and window functions:

Group By



Returns a single row for each group
(Changes the granularity)

WINDOW



Returns a result for each row
(The granularity stays the same)

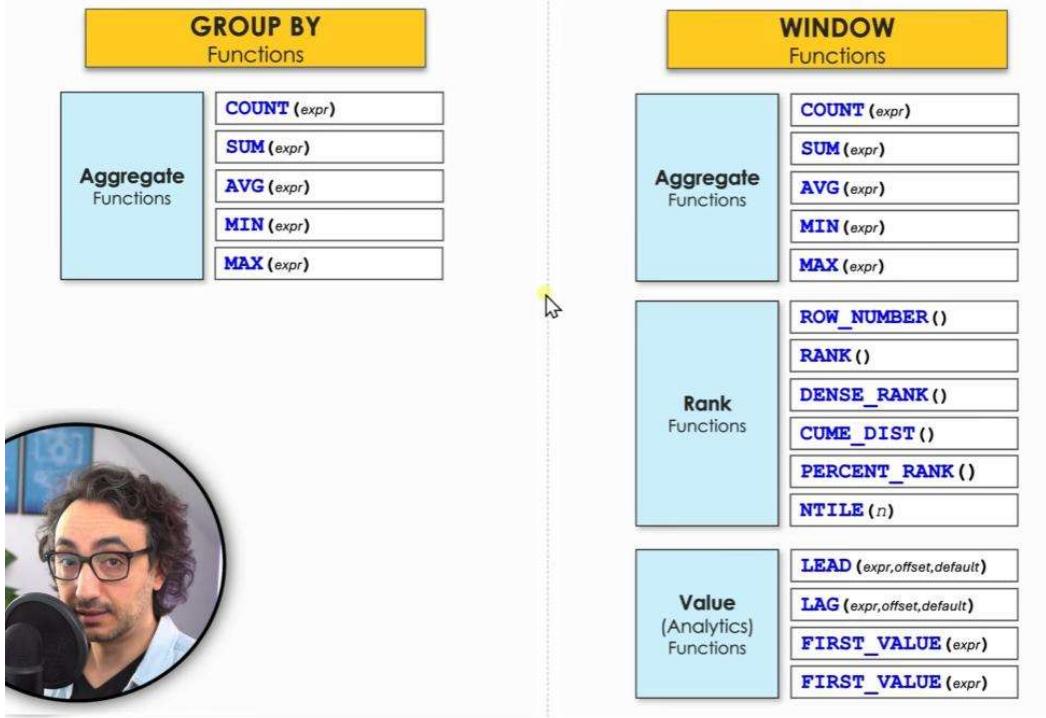
GROUP BY

Simple
Data Analysis

WINDOW

Advanced
Data Analysis





- WHY WE NEED **WINDOW FUNCTIONS?**
 - WHY GROUP BY IS **NOT ENOUGH ?**

A screenshot of the SQL Server Management Studio interface. The Object Explorer shows a database structure with tables like SalesDB, Sales, and SalesOrderArchive. The SQL Query Editor window contains the following code:

```
--Find the total Sales Across all orders
SELECT
    SUM(Sales) TotalSales
FROM Sales.Orders
```

The Results pane displays the output:

| TotalSales |
|------------|
| 1 380 |

A cursor arrow points to the value '380'. A circular portrait of the same man from the previous section is visible in the bottom right corner.

--Find the total sales for each product

```


SELECT
    ProductID,
    SUM(Sales) TotalSales
FROM Sales.Orders
GROUP BY ProductID


```

| | ProductID | TotalSales |
|---|-----------|------------|
| 1 | 101 | 140 |
| 2 | 102 | 105 |
| 3 | 104 | 75 |
| 4 | 105 | 60 |

/*Find the total sales for each product
Additionally provide details such order Id, order date*/

```


SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) TotalSales
FROM Sales.Orders
GROUP BY ProductID


```

Msg 8120, Level 16, State 1, Line 4
Column 'Sales.Orders.OrderID' is invalid.

Completion time: 2024-02-25T15:30:46.0

/*Find the total sales for each product
Additionally provide details such order Id, order date*/

```


SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) TotalSales
FROM Sales.Orders
GROUP BY
    OrderID,
    OrderDate,
    ProductID


```

| | OrderID | OrderDate | ProductID | TotalSales |
|----|------------|-----------|-----------|------------|
| 1 | 2025-01-01 | 101 | 10 | |
| 2 | 2025-01-05 | 102 | 15 | |
| 3 | 2025-01-10 | 101 | 20 | |
| 4 | 2025-01-20 | 105 | 60 | |
| 5 | 2025-02-01 | 104 | 25 | |
| 6 | 2025-02-05 | 104 | 50 | |
| 7 | 2025-02-15 | 102 | 30 | |
| 8 | 2025-02-18 | 101 | 90 | |
| 9 | 2025-03-10 | 101 | 20 | |
| 10 | 2025-03-15 | 102 | 60 | |

GROUP BY LIMITS

Can't do aggregations and provide details at same time

```

/*Find the total sales for each product
Additionally provide details such order Id, order date*/
SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) TotalSales
FROM Sales.Orders
GROUP BY
    OrderID,
    OrderDate,
    ProductID

```

| OrderID | OrderDate | ProductID | TotalSales |
|---------|------------|-----------|------------|
| 1 | 2025-01-01 | 101 | 10 |
| 2 | 2025-01-05 | 102 | 15 |
| 3 | 2025-01-10 | 101 | 20 |
| 4 | 2025-01-20 | 105 | 60 |
| 5 | 2025-02-01 | 104 | 25 |
| 6 | 2025-02-05 | 104 | 50 |
| 7 | 2025-02-15 | 102 | 30 |
| 8 | 2025-02-18 | 101 | 90 |
| 9 | 2025-03-10 | 101 | 20 |
| 10 | 2025-03-15 | 102 | 60 |

Result Granularity

Window functions returns a result for each row

```

/*Find the total sales for each product
Additionally provide details such order Id, order date*/
SELECT
    SUM(Sales) OVER()
FROM Sales.Orders

```

| (No column name) |
|------------------|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

```

/*Find the total sales for each product
Additionally provide details such order Id, order date*/
SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) OVER(PARTITION BY ProductID) TotalSalesByProducts
FROM Sales.Orders

```

| OrderID | OrderDate | ProductID | TotalSalesByProducts |
|---------|------------|-----------|----------------------|
| 1 | 2025-01-01 | 101 | 140 |
| 2 | 2025-01-10 | 101 | 140 |
| 3 | 2025-02-18 | 101 | 140 |
| 4 | 2025-03-10 | 101 | 140 |
| 5 | 2025-03-15 | 102 | 105 |
| 6 | 2025-01-05 | 102 | 105 |
| 7 | 2025-02-15 | 102 | 105 |
| 8 | 2025-02-01 | 104 | 75 |
| 9 | 2025-02-05 | 104 | 75 |
| 10 | 2025-01-20 | 105 | 60 |

GROUP BY

Simple Data Analysis
(Aggregations)

WINDOW

Advanced Data Analysis
(Aggregations + Details)

</> Window Syntax



```
AVG(Sales) OVER ( PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING )
```

Calculation used
on the Window

$f(x)$ Window
Function

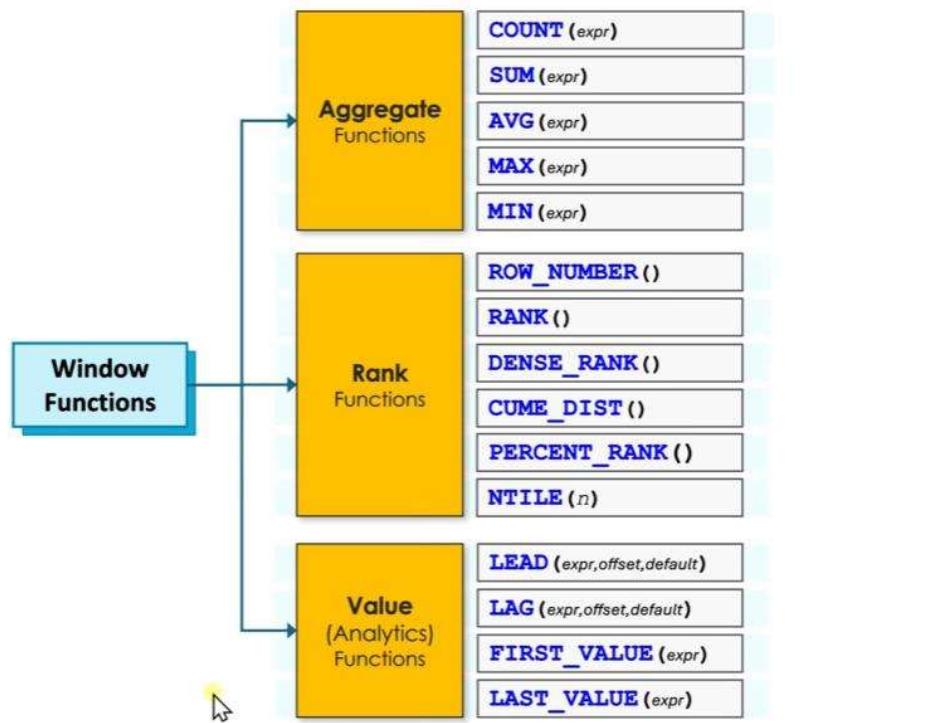
AVG(Sales)

```
OVER ( PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING )
```



WINODW FUNCTIONS

Perform calculations within a window



Calculation used
on the Window

f(\star) Window
Function

AVG (Sales)

OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)

Function
Expression



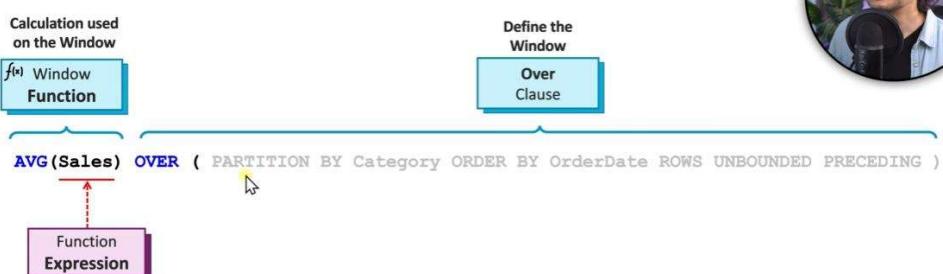
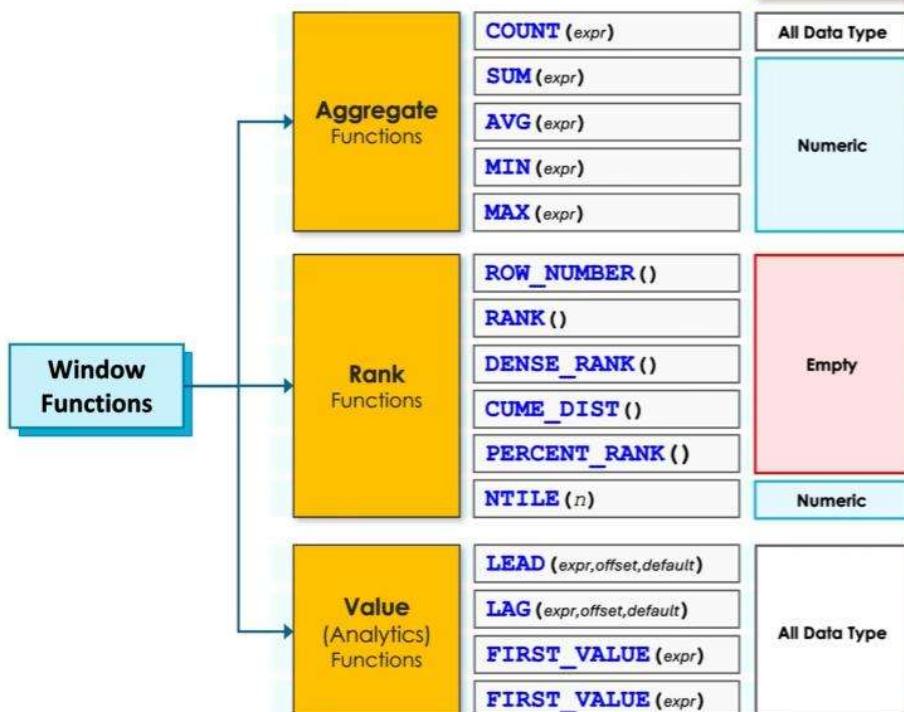
FUNCTION EXPRESSION

Arguments you pass to a function

$f(x)$ Window Expression



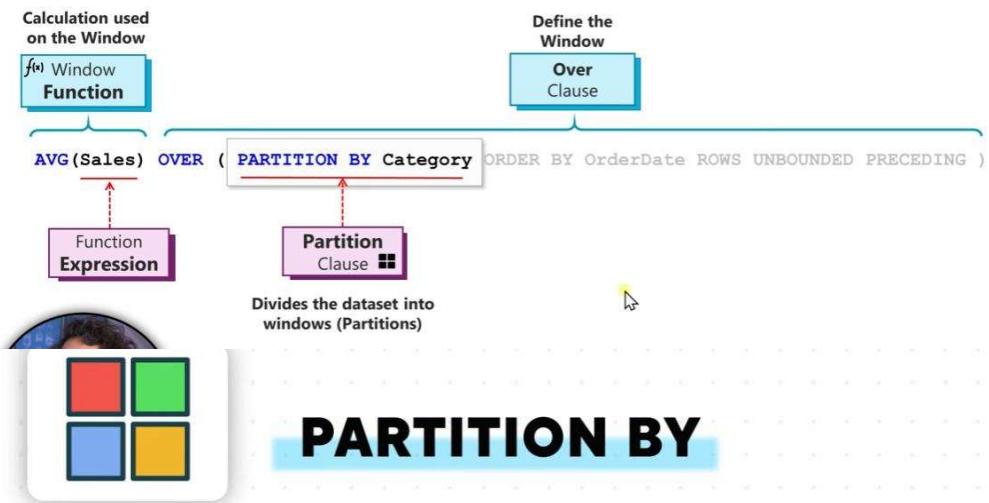
Expression



OVER CLAUSE

Tells SQL that the function used is a window function

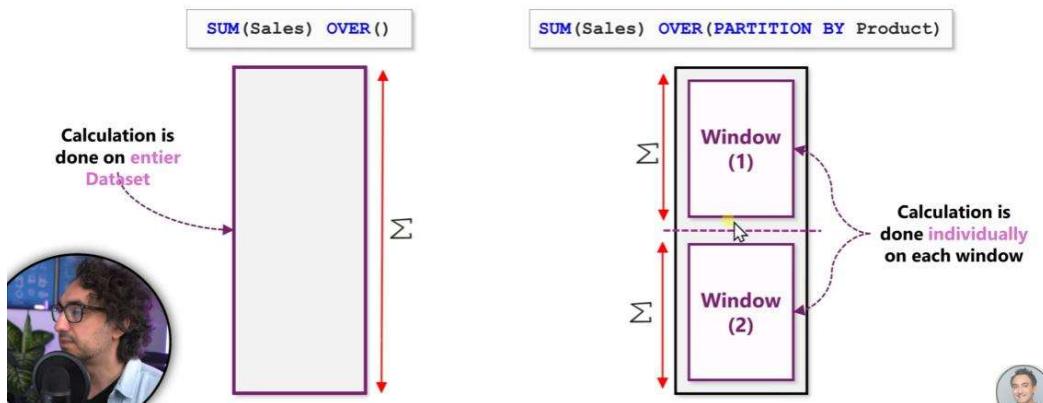
It defines a window or subset of data



Divides the result set into partitions (Windows)

Partition Clause

`PARTITION BY` divides the rows into groups, based on the column/s



SUM(Sales) OVER()

| Month | Product | Sales |
|-------|---------|-------|
| Jan | Bottle | 20 |
| Jan | Caps | 10 |
| Jan | Bottle | 30 |
| Feb | Gloves | 5 |
| Feb | Caps | 70 |
| Feb | Gloves | 40 |

Entire Dataset

**SUM(Sales) OVER(
PARTITION BY Month)**

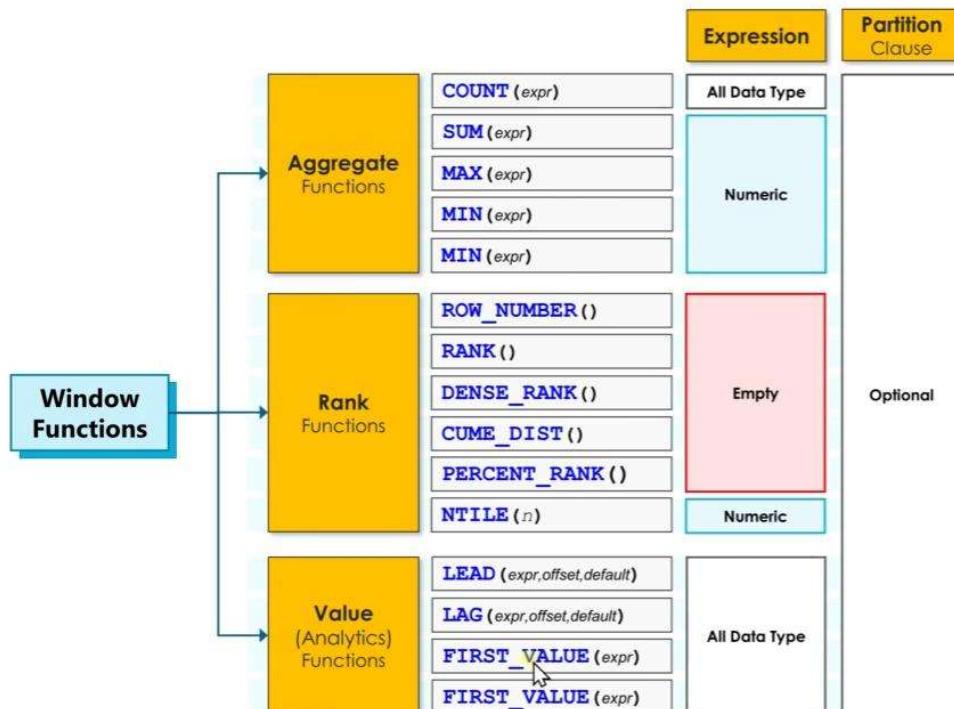
| Month | Product | Sales |
|-------|---------|-------|
| Jan | Bottle | 20 |
| Jan | Caps | 10 |
| Jan | Bottle | 30 |
| Feb | Gloves | 5 |
| Feb | Caps | 70 |
| Feb | Gloves | 40 |

Entire Dataset

Dataset is divided into 2 window

| | |
|--------------------------------------|--|
| Without Partition By | Total sales across all rows (Entire Result Set) <code>SUM(Sales) OVER ()</code> |
| Partition By Single Column | Total sales for each Product <code>SUM(Sales) OVER (PARTITION BY Product)</code> |
| Partition By Combined- Columns | Total sales for each combination of Product and Order Status <code>SUM(Sales) OVER (PARTITION BY Product, OrderStatus)</code> |

1 / 19



Object Explorer

```
-- Find the total sales across all orders
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    SUM(Sales) OVER () TotalSales
FROM Sales.Orders
```

| | OrderID | OrderDate | TotalSales |
|----|---------|------------|------------|
| 1 | 1 | 2025-01-01 | 380 |
| 2 | 2 | 2025-01-05 | 380 |
| 3 | 3 | 2025-01-10 | 380 |
| 4 | 4 | 2025-01-20 | 380 |
| 5 | 5 | 2025-02-01 | 380 |
| 6 | 6 | 2025-02-05 | 380 |
| 7 | 7 | 2025-02-15 | 380 |
| 8 | 8 | 2025-02-18 | 380 |
| 9 | 9 | 2025-03-10 | 380 |
| 10 | 10 | 2025-03-15 | 380 |
| | | 2025-03-10 | 380 |

SalesDB

```
-- Find the total sales for each product
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) OVER (PARTITION BY ProductID) TotalSales
FROM Sales.Orders
```

| | OrderID | OrderDate | TotalSales |
|----|---------|------------|------------|
| 1 | 1 | 2025-01-01 | 140 |
| 2 | 3 | 2025-01-10 | 140 |
| 3 | 8 | 2025-02-18 | 140 |
| 4 | 9 | 2025-03-10 | 140 |
| 5 | 10 | 2025-03-15 | 105 |
| 6 | 2 | 2025-01-05 | 105 |
| 7 | 7 | 2025-02-15 | 105 |
| 8 | 5 | 2025-02-01 | 75 |
| 9 | 6 | 2025-02-05 | 75 |
| 10 | 4 | 2025-01-20 | 60 |

Query executed successfully.

SQLQuery2.sql - D:\QBQ\YouTube (70) - SQL Server Object Explorer

```
-- Find the total sales for each product
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) OVER (PARTITION BY ProductID) TotalSales
FROM Sales.Orders
```

| | OrderID | OrderDate | ProductID | TotalSales |
|----|---------|------------|-----------|------------|
| 1 | 1 | 2025-01-01 | 101 | 140 |
| 2 | 3 | 2025-01-10 | 101 | 140 |
| 3 | 8 | 2025-02-18 | 101 | 140 |
| 4 | 9 | 2025-03-10 | 101 | 140 |
| 5 | 10 | 2025-03-15 | 102 | 105 |
| 6 | 2 | 2025-01-05 | 102 | 105 |
| 7 | 7 | 2025-02-15 | 102 | 105 |
| 8 | 5 | 2025-02-01 | 104 | 75 |
| 9 | 6 | 2025-02-05 | 104 | 75 |
| 10 | 4 | 2025-01-20 | 105 | 60 |



SQLQuery1.sql - D:\QBQ\YouTube (70) - SQL Server Object Explorer

```
-- Find the total sales across all orders
-- Find the total sales for each product
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    SUM(Sales) OVER () TotalSales,
    SUM(Sales) OVER (PARTITION BY ProductID) TotalSalesByProducts
FROM Sales.Orders
```

| | OrderID | OrderDate | ProductID | TotalSales | TotalSalesByProducts |
|----|---------|------------|-----------|------------|----------------------|
| 1 | 1 | 2025-01-01 | 101 | 380 | 140 |
| 2 | 3 | 2025-01-10 | 101 | 380 | 140 |
| 3 | 8 | 2025-02-18 | 101 | 380 | 140 |
| 4 | 9 | 2025-03-10 | 101 | 380 | 140 |
| 5 | 10 | 2025-03-15 | 102 | 380 | 105 |
| 6 | 2 | 2025-01-05 | 102 | 380 | 105 |
| 7 | 7 | 2025-02-15 | 102 | 380 | 105 |
| 8 | 5 | 2025-02-01 | 104 | 380 | 75 |
| 9 | 6 | 2025-02-05 | 104 | 380 | 75 |
| 10 | 4 | 2025-01-20 | 105 | 380 | 60 |



SalesDB

```
-- Find the total sales across all orders
-- Find the total sales for each product
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    Sales,
    SUM(Sales) OVER () TotalSales,
    SUM(Sales) OVER (PARTITION BY ProductID) TotalSalesByProducts
FROM Sales.Orders
```

Flexibility of Window

Allows aggregation of data at different granularities within the same query

| OrderID | OrderDate | ProductID | Sales | TotalSales | TotalSalesByProducts |
|---------|------------|-----------|-------|------------|----------------------|
| 1 | 2025-01-01 | 101 | 10 | 380 | 140 |
| 2 | 2025-01-10 | 101 | 20 | 380 | 140 |
| 3 | 2025-02-18 | 101 | 90 | 380 | 140 |
| 4 | 2025-03-10 | 101 | 20 | 380 | 140 |
| 5 | 2025-03-15 | 102 | 60 | 380 | 105 |
| 6 | 2025-01-05 | 102 | 15 | 380 | 105 |
| 7 | 2025-02-15 | 102 | 30 | 380 | 105 |
| 8 | 2025-02-01 | 104 | 25 | 380 | 75 |
| 9 | 2025-02-05 | 104 | 50 | 380 | 75 |
| 10 | 2025-01-20 | 105 | 60 | 380 | 60 |

SalesDB

```
-- Find the total sales across all orders
-- Find the total sales for each product
-- Find the total sales for each combination of product and order status
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    OrderStatus,
    Sales,
    SUM(Sales) OVER () TotalSales,
    SUM(Sales) OVER (PARTITION BY ProductID) SalesByProducts,
    SUM(Sales) OVER (PARTITION BY ProductID, OrderStatus) SalesByProductsAndStatus
FROM Sales.Orders
```

| OrderID | OrderDate | ProductID | OrderStatus | Sales | TotalSales | SalesByProducts | SalesByProductsAndStatus |
|---------|------------|-----------|-------------|-------|------------|-----------------|--------------------------|
| 1 | 2025-01-01 | 101 | Delivered | 10 | 380 | 140 | 30 |
| 2 | 2025-01-10 | 101 | Delivered | 20 | 380 | 140 | 30 |
| 3 | 2025-02-18 | 101 | Shipped | 90 | 380 | 140 | 110 |
| 4 | 2025-03-10 | 101 | Shipped | 20 | 380 | 140 | 110 |
| 5 | 2025-02-15 | 102 | Delivered | 30 | 380 | 105 | 30 |
| 6 | 2025-03-15 | 102 | Shipped | 60 | 380 | 105 | 75 |
| 7 | 2025-01-05 | 102 | Shipped | 15 | 380 | 105 | 75 |
| 8 | 2025-02-01 | 104 | Delivered | 25 | 380 | 75 | 75 |
| 9 | 2025-02-05 | 104 | Delivered | 50 | 380 | 75 | 75 |
| 10 | 2025-01-20 | 105 | Shipped | 60 | 380 | 60 | 60 |

</> Window Syntax

Calculation used on the Window

fn Window Function

Define the Window

Over Clause

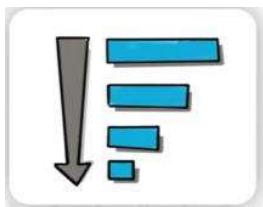
AVG(Sales) OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)

Function Expression

Partition Clause

Divides the dataset into windows (Partitions)

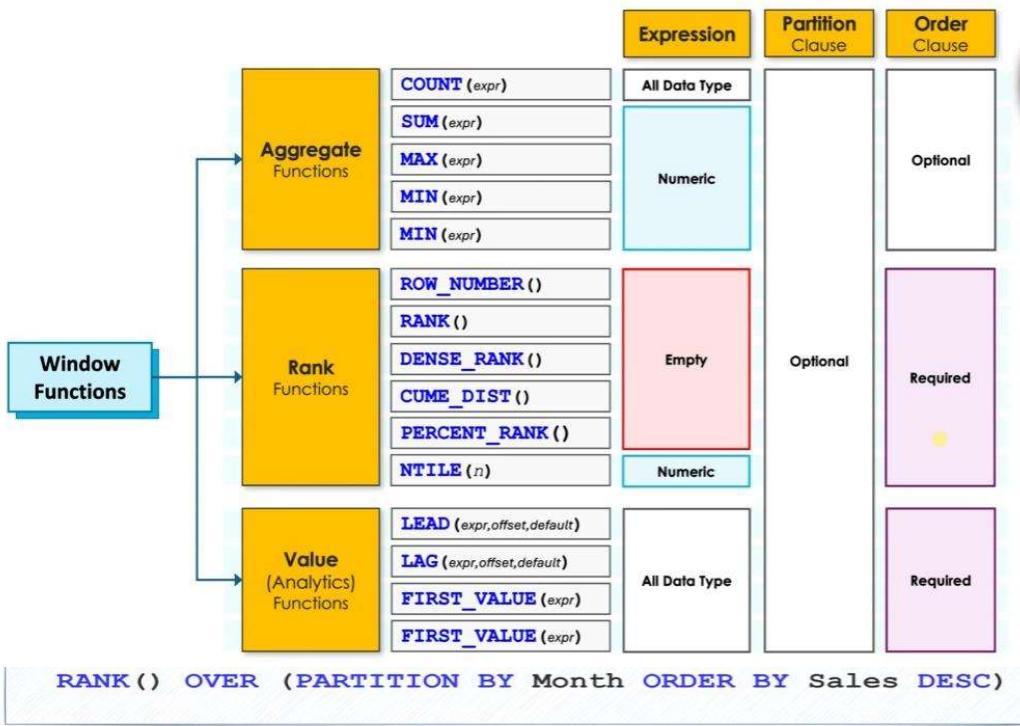




ORDER BY

Sort the data within a window

(Ascending | Descending)



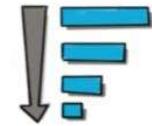
| Month | Product | Sales |
|-------|---------|-------|
| Jan | Bottle | 20 |
| | Caps | 10 |
| | Bottle | 30 |
| Feb | Gloves | 5 |
| | Caps | 70 |
| | Gloves | 40 |



Spiltted by month and sorted by sales

```
RANK() OVER (PARTITION BY Month ORDER BY Sales DESC)
```

| Month | Product | Sales |
|-------|---------|-------|
| Jan | Bottle | 30 |
| Jan | Bottle | 20 |
| Jan | Caps | 10 |



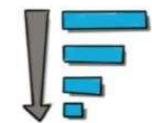
| Month | Product | Sales |
|-------|---------|-------|
| Feb | Caps | 70 |
| Feb | Gloves | 40 |
| Feb | Gloves | 5 |



```
RANK() OVER (PARTITION BY Month ORDER BY Sales DESC)
```

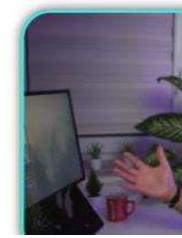
| Month | Product | Sales |
|-------|---------|-------|
| Jan | Bottle | 30 |
| Jan | Bottle | 20 |
| Jan | Caps | 10 |

| Rank |
|------|
| 1 |
| 2 |
| 3 |



| Month | Product | Sales |
|-------|---------|-------|
| Feb | Caps | 70 |
| Feb | Gloves | 40 |
| Feb | Gloves | 5 |

| Rank |
|------|
| 1 |
| 2 |
| 3 |



```

-- Rank each order based on their sales from highest to lowest
-- Additionally provide details such order Id, order date

SELECT
    OrderID,
    OrderDate,
    Sales,
    RANK() OVER (ORDER BY Sales DESC) RankSales
FROM Sales.Orders

```

| | OrderID | OrderDate | Sales | RankSales |
|----|---------|------------|-------|-----------|
| 1 | 8 | 2025-02-18 | 90 | 1 |
| 2 | 4 | 2025-01-20 | 60 | 2 |
| 3 | 10 | 2025-03-15 | 60 | 2 |
| 4 | 6 | 2025-02-05 | 50 | 4 |
| 5 | 7 | 2025-02-15 | 30 | 5 |
| 6 | 5 | 2025-02-01 | 25 | 6 |
| 7 | 9 | 2025-03-10 | 20 | 7 |
| 8 | 3 | 2025-01-10 | 20 | 7 |
| 9 | 2 | 2025-01-05 | 15 | 9 |
| 10 | 1 | 2025-01-01 | 10 | 10 |

Query executed successfully.



Default Sorting

As default ORDER BY sorts the data in ascending order 'ASC' (from Lowest to highest)



```

-- Rank each order based on their sales from highest to lowest
-- Additionally provide details such order Id, order date

SELECT
    OrderID,
    OrderDate,
    Sales,
    RANK() OVER (ORDER BY Sales ASC) RankSales
FROM Sales.Orders

```

| | OrderID | OrderDate | Sales | RankSales |
|----|---------|------------|-------|-----------|
| 1 | 1 | 2025-01-01 | 10 | 1 |
| 2 | 2 | 2025-01-05 | 15 | 2 |
| 3 | 3 | 2025-01-10 | 20 | 3 |
| 4 | 9 | 2025-03-10 | 20 | 3 |
| 5 | 5 | 2025-02-01 | 25 | 5 |
| 6 | 7 | 2025-02-15 | 30 | 6 |
| 7 | 6 | 2025-02-05 | 50 | 7 |
| 8 | 4 | 2025-01-20 | 60 | 8 |
| 9 | 10 | 2025-03-15 | 60 | 8 |
| 10 | 8 | 2025-02-18 | 90 | 10 |

Default Sorting

As default ORDER BY sorts the data in ascending order 'ASC' (from Lowest to highest)



</> Window Syntax



Calculation used on the Window

f(x) Window Function

Define the Window

Over Clause

AVG(Sales) OVER (PARTITION BY Category ORDER BY OrderDate ROWS UNBOUNDED PRECEDING)

Function Expression

Partition Clause

Order Clause

Frame Clause

Divides the dataset into windows (Partitions)

Sort the data in a window

Define a subset of rows in a window



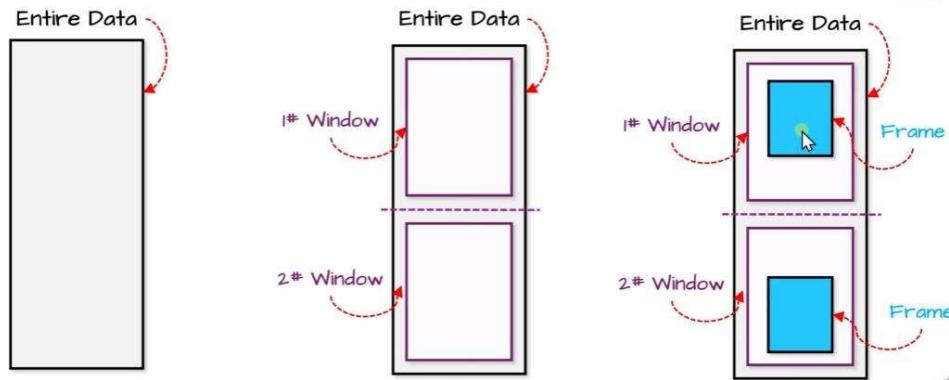
WINDOW FRAME

Defines a subset of rows within each window
that is relevant for the calculation

Window inside window

Specific rows

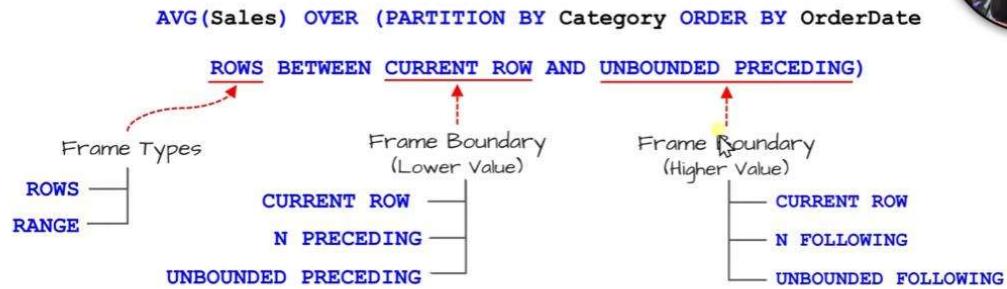
Frame Clause



When we focus subset of data from each window



Frame Clause | Syntax



Rules

- Frame Clause can only be used **together** with order by clause.
- Lower Value must be **BEFORE** the higher Value.

SUM(Sales)
 OVER(ORDER BY Month
 ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)

| Month | Sales | Result |
|-------|-------|--------|
| Jan | 20 | |
| Feb | 10 | |
| Mar | 30 | |
| Apr | 5 | |
| Jun | 70 | |

N-FOLLOWING

The n-th row before the current row



```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)

```

The diagram illustrates a windowed sum calculation for the month of March. A table on the left shows monthly sales data:

| Month | Sales |
|-------|-------|
| Jan | 20 |
| Feb | 10 |
| Mar | 30 |
| Apr | 5 |
| Jun | 70 |

A red box highlights the current row for March (30). A yellow box labeled "2 Following" points to the rows for April (5) and June (70). To the right, a "Result" column shows the cumulative sum: 70 (for March), 45 (for April), and 70 (for June). A cursor is shown clicking on the value 70 in the Result column for June.



```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)

```

The diagram illustrates a windowed sum calculation for the month of February. A table on the left shows monthly sales data:

| Month | Sales |
|-------|-------|
| Jan | 20 |
| Feb | 10 |
| Mar | 30 |
| Apr | 5 |
| Jun | 70 |

A red box highlights the current row for February (10). A yellow box labeled "2 Following" points to the rows for March (30) and June (70). To the right, a "Result" column shows the cumulative sum: 70 (for January), 45 (for February), and 70 (for June).



```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)

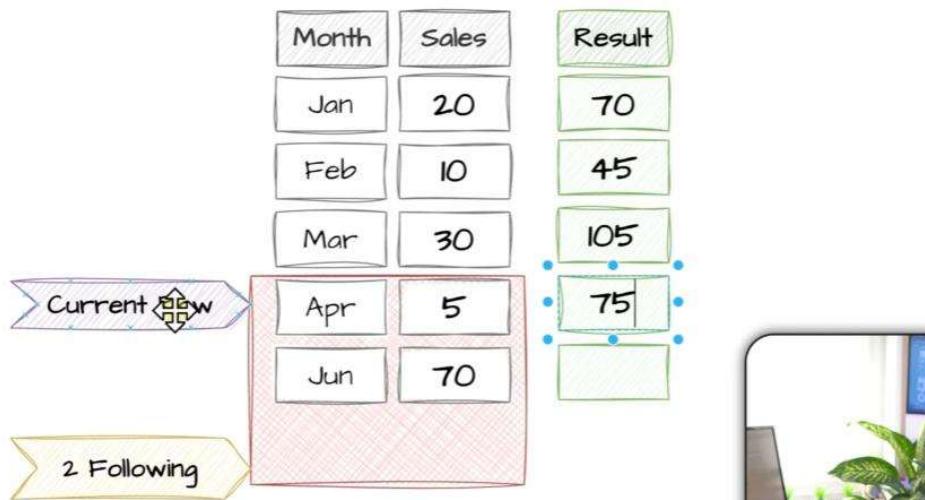
```



```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)

```

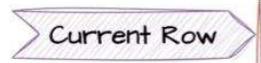


```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)

```

| Month | Sales | Result |
|-------|-------|--------|
| Jan | 20 | 70 |
| Feb | 10 | 45 |
| Mar | 30 | 105 |
| Apr | 5 | 75 |
| Jun | 70 | 70 |






```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)

```

| Month | Sales | Result |
|-------|-------|--------|
| Jan | 20 | |
| Feb | 10 | |
| Mar | 30 | |
| Apr | 5 | |
| Jun | 70 | |

UNBOUNDED FOLLOWING

The last possible row within a window

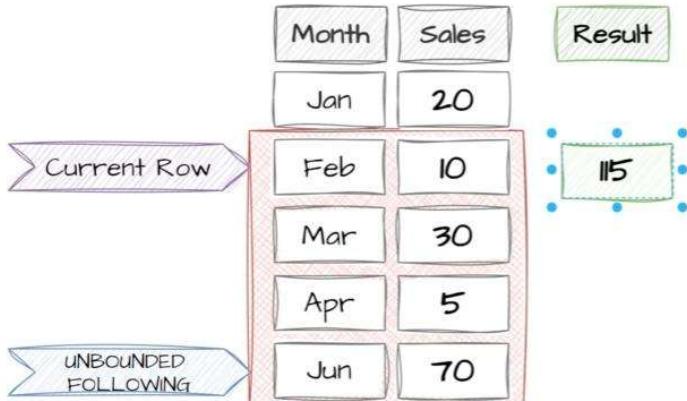


Ex- we start as a example from feb it goes on for all or same for all

```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)

```



```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)

```

