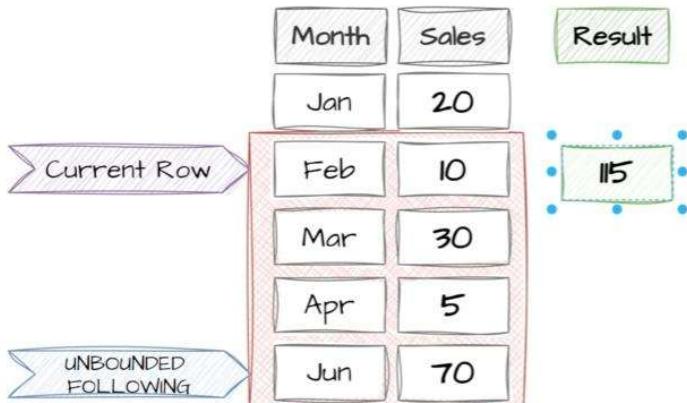


```

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)

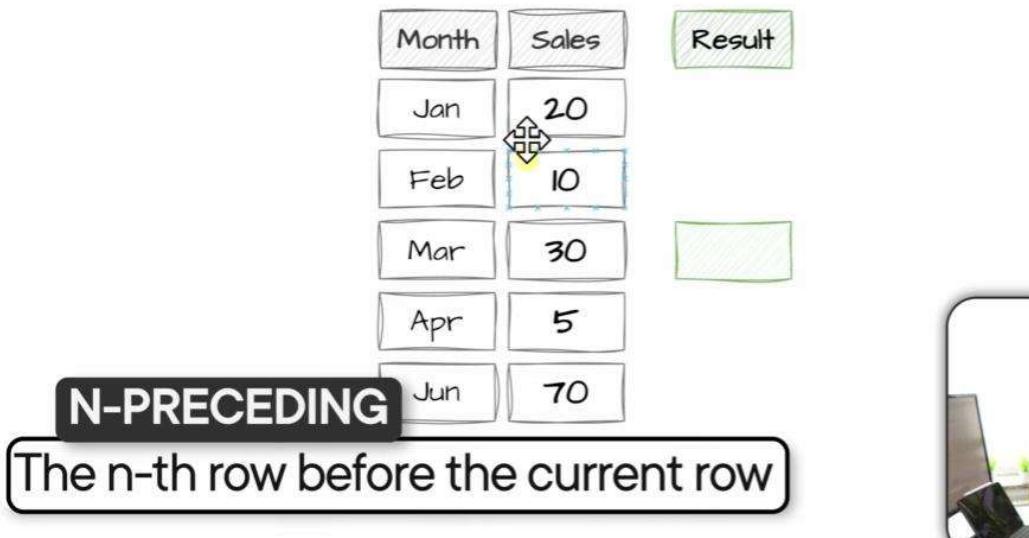
```



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



```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)
```

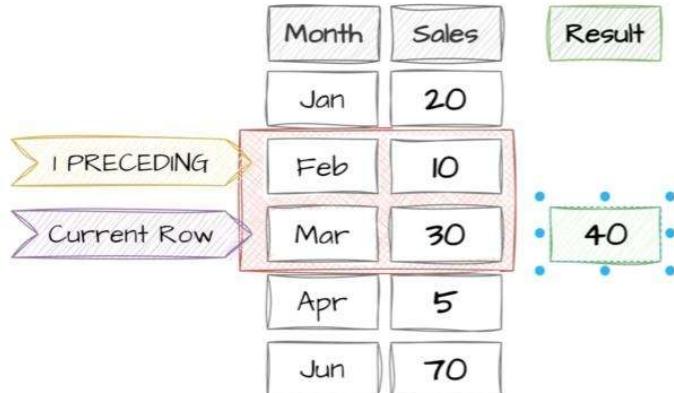


Targetting row before current row

```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)

```



```

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

```



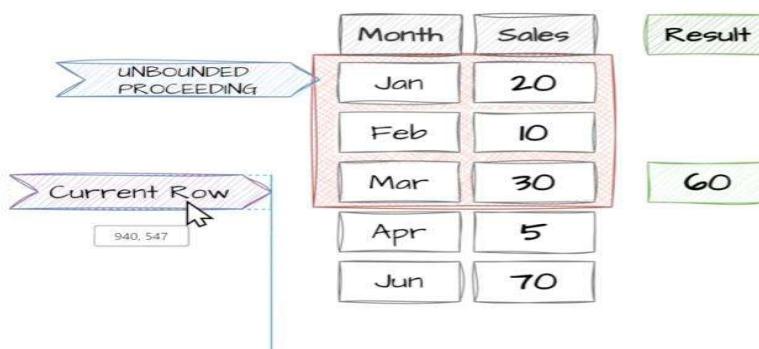
UNBOUNDED PRECEDING

(The first possible row within a window)

```

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

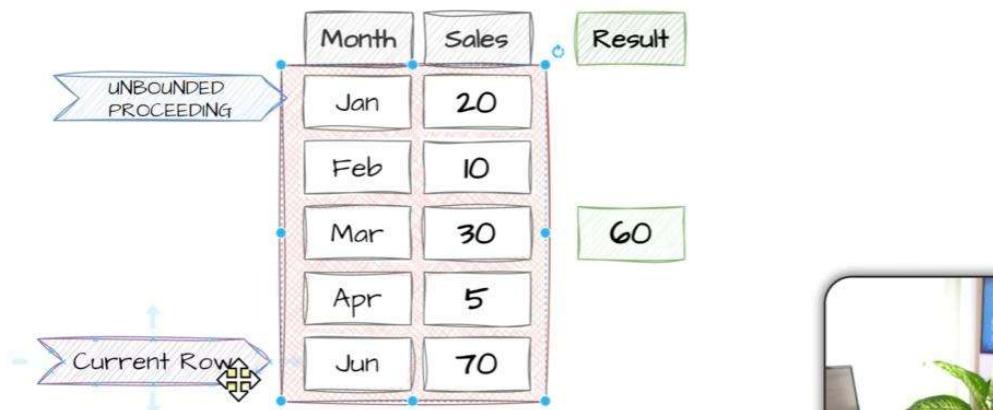
```



```

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

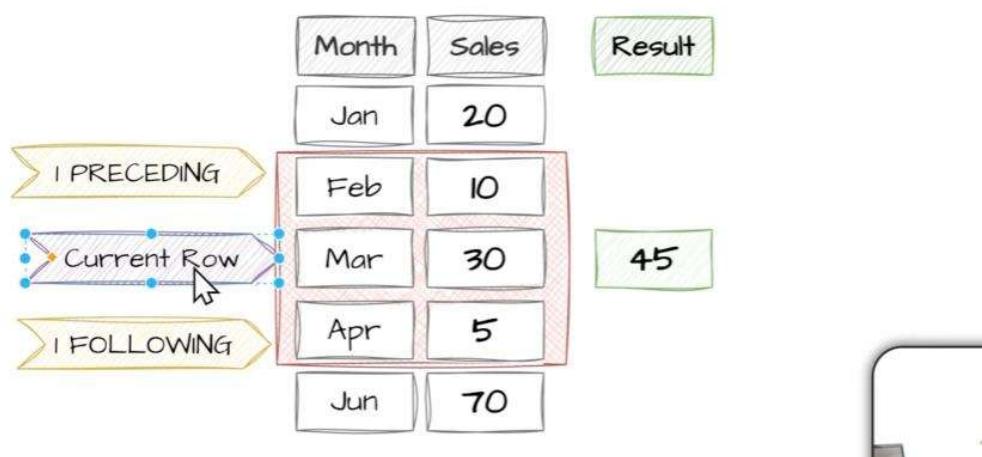
```



```

SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)

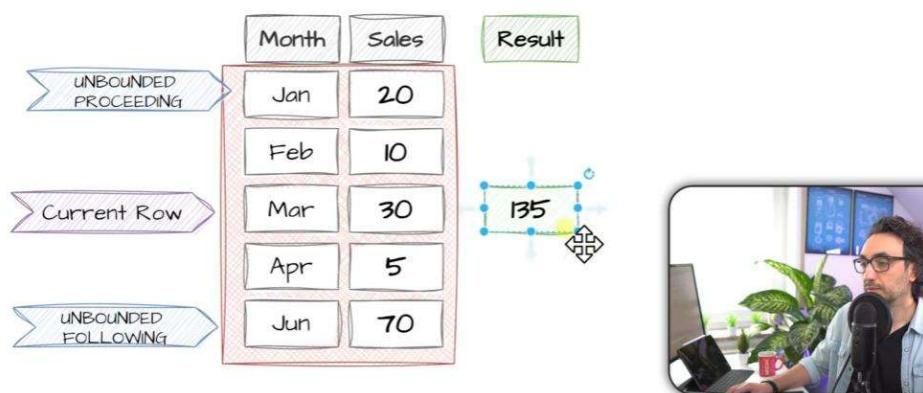
```



```

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

```



Current row can be anywhere but we ans will be same for all rows because it's gonna be a fixed subset.

`SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)`

Month	Sales	Result
Jan	20	135
Feb	10	135
Mar	30	135
Apr	5	135
Jun	70	135

UNBOUNDED PRECEDING
Current Row
UNBOUNDED FOLLOWING

41:44 / 57:18 • Frame Clause >

Object Explorer

```
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING) TotalSales
FROM Sales.Orders
```

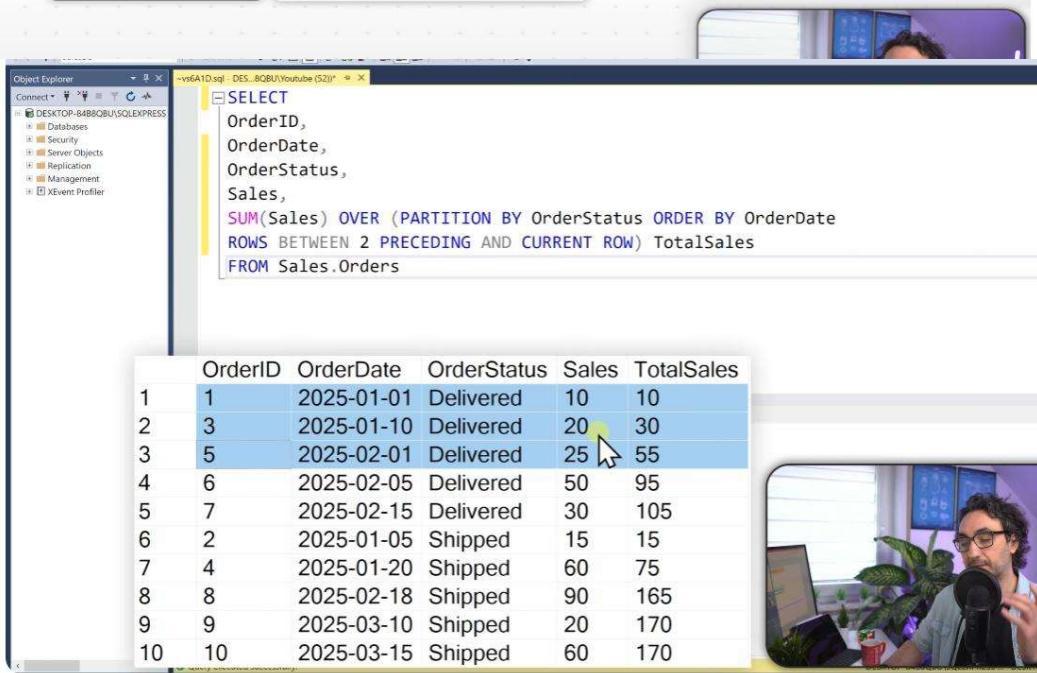
	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	55
2	3	2025-01-10	Delivered	20	95
#1 WINDOW	5	2025-02-01	Delivered	25	105
4	6	2025-02-05	Delivered	50	80
5	7	2025-02-15	Delivered	30	30
6	2	2025-01-05	Shipped	15	165
7	4	2025-01-20	Shipped	60	170
8	8	2025-02-18	Shipped	90	170
9	9	2025-03-10	Shipped	20	80
10	10	2025-03-15	Shipped	60	60

COMPACT FRAME

For only PRECEDING, the CURRENT ROW can be skipped

NORMAL FORM ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING

SHORT FORM ROWS 2 FOLLOWING



The screenshot shows a SQL Server Management Studio interface. On the left is the Object Explorer with a connection to 'DESKTOP-848B08U\SQLExpress'. In the center is the Query Editor window containing the following T-SQL code:

```
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) TotalSales
FROM Sales.Orders
```

Below the Query Editor is a results grid displaying data from the 'Sales.Orders' table. The columns are OrderID, OrderDate, OrderStatus, Sales, and TotalSales. The data is as follows:

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	95
5	7	2025-02-15	Delivered	30	105
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165
9	9	2025-03-10	Shipped	20	170
10	10	2025-03-15	Shipped	60	170

Shortcut work for preceding only else it throws error

Compact frame works only for preceding

SalesDB

```
Object Explorer Execute File View Tools Options Help
Connect To Server Connect To Database
DESKTOP-84B8QBU\SQLEXPRESS
  Databases
  Security
  Server Objects
  Replication
  Management
  XEvent Profiler

~vs6A1D.sql : DES...8QBU\Youtube (52)* | X

SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS 2 PRECEDING) AS TotalSales
FROM Sales.Orders
```

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	95
5	7	2025-02-15	Delivered	30	105
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165
9	9	2025-03-10	Shipped	20	170
10	10	2025-03-15	Shipped	60	170

Object Explorer Execute File View Tools Options Help
Connect To Server Connect To Database
DESKTOP-84B8QBU\SQLEXPRESS
 Databases
 Security
 Server Objects
 Replication
 Management
 XEvent Profiler

~vs6A1D.sql : DES...8QBU\Youtube (52)* | X

SELECT
 OrderID,
 OrderDate,
 OrderStatus,
 Sales,
 SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
 ROWS UNBOUNDED PRECEDING) AS TotalSales
FROM Sales.Orders

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	135
5	7	2025-02-15	Delivered	30	135
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165

```

Object Explorer    execute  ~vs6A1D.sql [DES...8QBU\Youtube (52)] * X
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS UNBOUNDED FOLLOWING) TotalSales
FROM Sales.Orders

```

Msg 102, Level 15, State 1, Line 7
Incorrect syntax near 'FOLLOWING'.

Completion time: 2024-02-27T17:23:15.078641


```

Object Explorer    execute  ~vs6A1D.sql [DES...8QBU\Youtube (52)] * X
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS 1 FOLLOWING) TotalSales
FROM Sales.Orders

```

Msg 4193, Level 16, State 4, Line 1
'BETWEEN ... FOLLOWING AND CURRENT ROW' is r

Completion time: 2024-02-27T17:23:23.951057

DEFAULT FRAME

SQL uses Default Frame, if ORDER BY is used without FRAME

Default frame between the unbounded precedings and current row.
Default frames only working with the order by

There is a hidden frame used while we used order by that is default frame

Object Explorer

```
-vs6A1D.sql - DES...BOBU\Youtube (S2) - >
```

```

SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate) TotalSales
    FROM Sales.Orders
  
```

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	105
5	7	2025-02-15	Delivered	30	135
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165
9	9	2025-03-10	Shipped	20	185
10	10	2025-03-15	Shipped	60	245



Object Explorer

```
-vs6A1D.sql - DES...BOBU\Youtube (S2) - >
```

```

SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) TotalSales
    FROM Sales.Orders
  
```

NOTE

ORDER BY always uses a FRAME

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	105
5	7	2025-02-15	Delivered	30	135
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165
9	9	2025-03-10	Shipped	20	185
10	10	2025-03-15	Shipped	60	245



SQL Window Functions Basics | Partition By, Order By, Frame | #SQL Course 22

Another way

Object Explorer

```
-vs6A1D.sql - DES...BOBU\Youtube (S2) - >
```

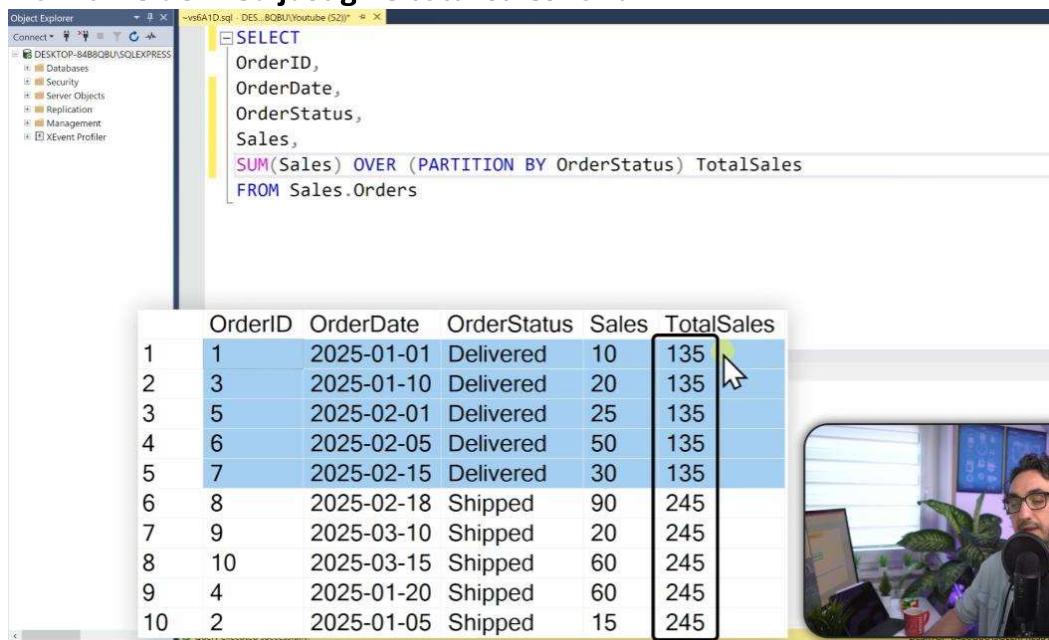
```

SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
    ROWS UNBOUNDED PRECEDING) TotalSales
    FROM Sales.Orders
  
```

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	105
5	7	2025-02-15	Delivered	30	135
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165
9	9	2025-03-10	Shipped	20	185
10	10	2025-03-15	Shipped	60	245



No Frame defined just give total sales for all



The screenshot shows a SQL Server Management Studio window. On the left is the Object Explorer pane, and on the right is a query editor window titled 'vs6A1D.sql' with the database 'DESKTOP-B4B8QBU\SQLEXPRESS'. The query is:

```
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
```

Below the query is a results grid:

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	135
2	3	2025-01-10	Delivered	20	135
3	5	2025-02-01	Delivered	25	135
4	6	2025-02-05	Delivered	50	135
5	7	2025-02-15	Delivered	30	135
6	8	2025-02-18	Shipped	90	245
7	9	2025-03-10	Shipped	20	245
8	10	2025-03-15	Shipped	60	245
9	4	2025-01-20	Shipped	60	245
10	2	2025-01-05	Shipped	15	245



Total sales highest value to lowest value

-- What we can do and not do using window function



#1 RULE

Window functions can be used ONLY

in **SELECT** and **ORDER BY** Clauses



```
vs7ED5.sql - DES...8QBU\Youtube (60)* - X -vs5021.sql - DES...8QBU\Youtube (53))  
SELECT  
    OrderID,  
    OrderDate,  
    OrderStatus,  
    Sales,  
    SUM(Sales) OVER (PARTITION BY OrderStatus) TotalSales  
FROM Sales.Orders  
ORDER BY SUM(Sales) OVER (PARTITION BY OrderStatus) DESC
```

226 %

Results Messages

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	8	2025-02-18	Shipped	90	245
2	9	2025-03-10	Shipped	20	245
3	10	2025-03-15	Shipped	60	245
4	4	2025-01-20	Shipped	60	245
5	2	2025-01-05	Shipped	15	245
6	1	2025-01-01	Delivered	10	135
7	3	2025-01-10	Delivered	20	135
8	5	2025-02-01	Delivered	25	135
9	6	2025-02-05	Delivered	50	135
10	7	2025-02-15	Delivered	30	135

```
vs7ED2.sql - DES...8QBU\Youtube (60)* - X -vs5021.sql - DES...8QBU\Youtube (53))  
SELECT  
    OrderID,  
    OrderDate,  
    OrderStatus,  
    Sales,  
    SUM(Sales) OVER (PARTITION BY OrderStatus) To  
FROM Sales.Orders  
ORDER BY SUM(Sales) OVER (PARTITION BY OrderStatus)
```

NOTE

WINDOW Functions
can't be used to filter Data

26 %

Results Messages

	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	8	2025-02-18	Shipped	90	245
2	9	2025-03-10	Shipped	20	245
3	10	2025-03-15	Shipped	60	245
4	4	2025-01-20	Shipped	60	245
5	2	2025-01-05	Shipped	15	245
6	1	2025-01-01	Delivered	10	135
7	3	2025-01-10	Delivered	20	135
8	5	2025-02-01	Delivered	25	135
9	6	2025-02-05	Delivered	50	135
10	7	2025-02-15	Delivered	30	135



Window function not used to filter data using where clause only used with select and order by clause

```
~vs7ED5.sql - DES...8QBU\Youtube (60)* - X ~vs5021.sql - DES...8QBU\Youtube (53)
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
WHERE SUM(Sales) OVER (PARTITION BY OrderStatus) > 100
```

226 %  Messages

Msg 4108, Level 15, State 1, Line 8
Windowed functions can only appear in the SELECT or ORDER BY clauses.

Completion time: 2024-03-02T13:59:08.4640664+01:00



Window function not used in group by

```
~vs7ED5.sql - DES...8QBU\Youtube (60)* - X ~vs5021.sql - DES...8QBU\Youtube (52)
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
GROUP BY SUM(Sales) OVER (PARTITION BY OrderStatus)
```

226 %  Messages

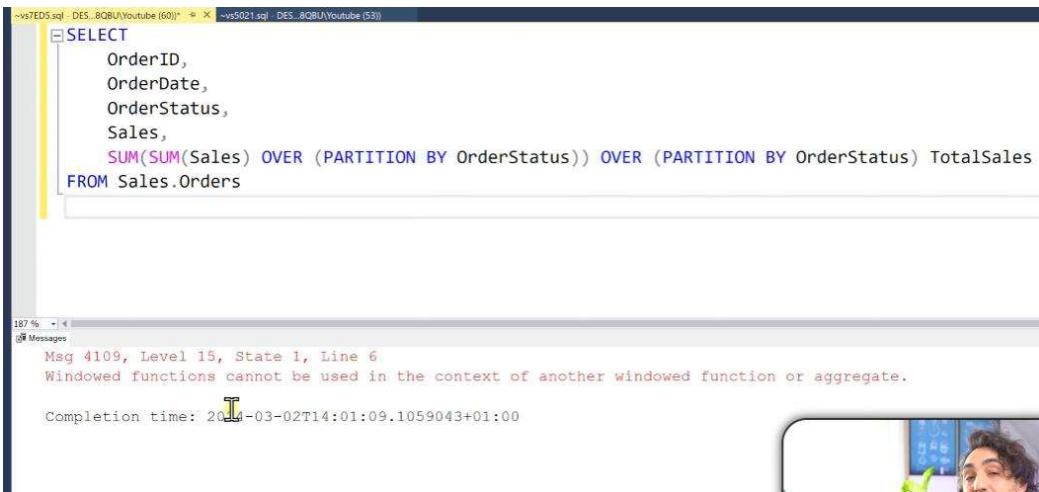
Msg 4108, Level 15, State 1, Line 8
Windowed functions can only appear in the SELECT or ORDER BY clauses.

Completion time: 2024-03-02T13:59:50.9199323+01:00



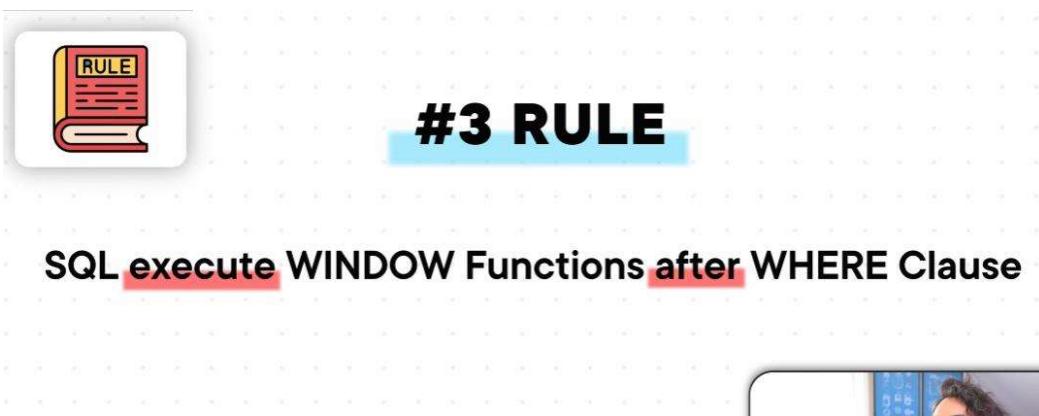
#2 RULE

Nesting Window Functions is not allowed !

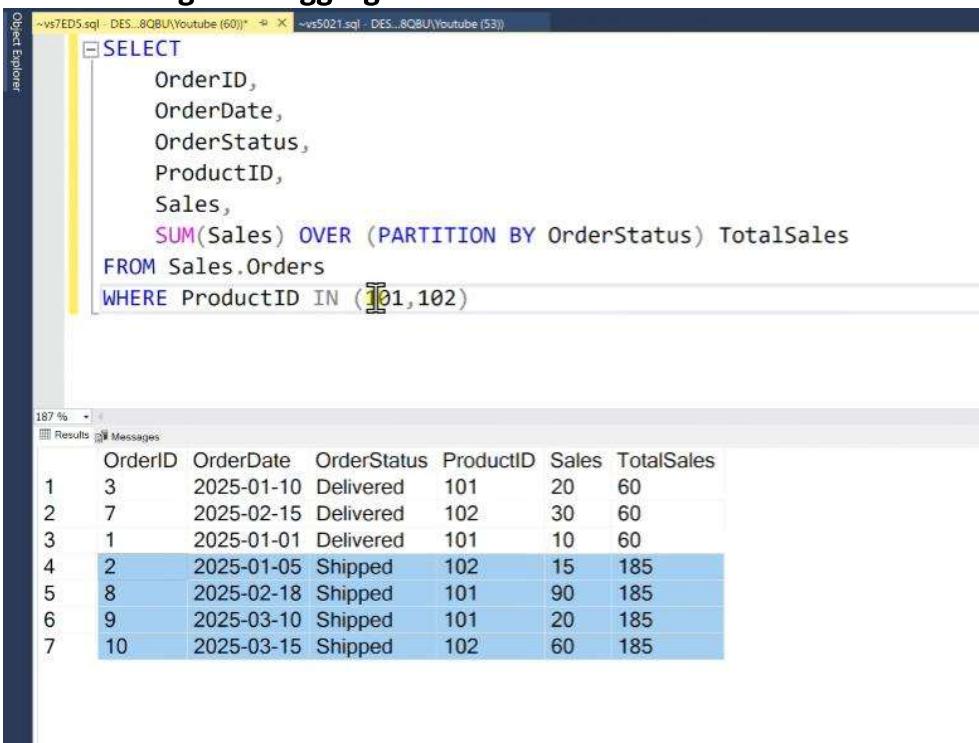


```
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(SUM(Sales)) OVER (PARTITION BY OrderStatus) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
```

Msg 4109, Level 15, State 1, Line 6
Windowed functions cannot be used in the context of another windowed function or aggregate.
Completion time: 2024-03-02T14:01:09.1059043+01:00



First Filtering then Aggregation



```
Object Explorer
~vs7ED5.sql - DES...8QBU(Youtube (60)) * ~vs5021.sql - DES...8QBU(Youtube (53))
```

```
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    ProductID,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
WHERE ProductID IN (101,102)
```

	OrderID	OrderDate	OrderStatus	ProductID	Sales	TotalSales
1	3	2025-01-10	Delivered	101	20	60
2	7	2025-02-15	Delivered	102	30	60
3	1	2025-01-01	Delivered	101	10	60
4	2	2025-01-05	Shipped	102	15	185
5	8	2025-02-18	Shipped	101	90	185
6	9	2025-03-10	Shipped	101	20	185
7	10	2025-03-15	Shipped	102	60	185



#4 RULE

Window Function can be used together with GROUP BY
in the same query, ONLY if the same columns are used

--Rank Customers based on their total sales

```
SELECT
    CustomerID,
    SUM(Sales) TotalSales,
    RANK() OVER(ORDER BY SUM(Sales) DESC) RankCustomers
FROM Sales.Orders
GROUP BY CustomerID
```

The screenshot shows a SQL query in the Object Explorer pane. The results pane displays a table with four columns: CustomerID, TotalSales, and RankCustomers. The data is as follows:

	CustomerID	TotalSales	RankCustomers
1	3	125	1
2	1	110	2
3	4	90	3
4	2	55	4

Anything that you used inside window function should be a part of group by

--Rank Customers based on their total sales

```
SELECT
    CustomerID,
    SUM(Sales) TotalSales,
    RANK() OVER(ORDER BY Sales DESC) RankCustomers
FROM Sales.Orders
GROUP BY CustomerID
```

The screenshot shows a SQL query in the Object Explorer pane. The results pane displays an error message:

Msg 8120, Level 16, State 1, Line 5
Column 'Sales.Orders.Sales' is invalid in the select list because it is not contained in either an aggregate function



Error because sales not part of group by



SQL WINDOW FUNCTIONS

Performs Calculations on Subset of data without losing details

Window v/s Group By

- Window is more powerful & dynamic than Group By
- Data Analysis → Advanced → Window
→ Simple → Group By
- Use Group By + window in same query, only if same column used.

Components

Window Functions + Window Definition **OVER**

Rules

Divide Data → PARTITION BY
Sort Data → ORDER BY
Define Subst → FRAME

Nesting is not allowed!

Window can be used only in **SELECT** and **ORDER BY**

SQL executes window AFTER filtering data using **WHERE**

Aggregated window function:

Month	Sales
Jan	20
Feb	10
Mar	30
Apr	5
Jun	70
Jul	40

One Aggregated Value

Σ

175



</> Aggregate Functions | Syntax

```
AVG(Sales) OVER (PARTITION BY ProductID ORDER BY Sales)
```

Expression
is required
(only Numeric Values)

Partition By
is Optional

Order By
is Optional

</> Aggregate Functions | Syntax

Aggregate Functions	Expression	Partition Clause	Order Clause	Frame Clause
COUNT (expr)	All Data Type			
SUM (expr)	NumericValues			
AVG (expr)	NumericValues	Optional	Optional	Optional
MIN (expr)	NumericValues			
MAX (expr)	NumericValues			

COUNT ()

Returns the number of rows within a window

COUNT (*) OVER(PARTITION BY Product)

Product	Sales
Caps	20
Caps	10
Caps	5

Gloves	Sales
Gloves	30
Gloves	70
Gloves	NULL

COUNT
3
3
3
3
3
3

TASK

Find the total number of Orders
for each product



Sql count null as one row

COUNT (*)

counts all the rows in a table,
regardless of whether any value is NULL

COUNT(Sales) OVER(PARTITION BY Product)

Product	Sales
Caps	20
Caps	10
Caps	5

Gloves	Sales
Gloves	30
Gloves	70
Gloves	NULL

COUNT
3
3
3

COUNT
2
2
2

TASK

Find the total number of Sales
for each product



COUNT (column)

counts the number of non-NULL values in the column

COUNT(1) OVER(PARTITION BY Product)

COUNT(*) OVER(PARTITION BY Product)

Product	Sales
Caps	20
Caps	10
Caps	5

COUNT
3
3
3

COUNT(Sales) OVER(PARTITION BY Product)

Product	Sales
Caps	20
Caps	10
Caps	5

COUNT
3
3
3

COUNT(1) equals to COUNT(*)



COUNT ()

counts the number of values in a column,

regardless of their data types.

```
COUNT(Product) OVER(PARTITION BY Product)
```

Product	Sales
Caps	20
Caps	10
Caps	5

COUNT
3
3
3

NOTE

Counts the total number of rows,
including duplicates,
not the unique values !

Gloves	30
Gloves	70
Gloves	NULL

3
3
3



It helps to understand data

-- Find the total number of Orders

```
SELECT
COUNT(*) TotalOrders
FROM Sales.Orders
```

206 %

TotalOrders
10

-- Find the total number of Orders
-- Additionally provide details such order Id, order date

```
SELECT
OrderID,
OrderDate,
COUNT(?) TotalOrders
FROM Sales.Orders
```

Msg 8120, Level 16, State 1, Line 4
Column 'Sales.Orders.OrderID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.



```
SQLQuery7.sql - D:\8Q8U\Youtube (72)* ✎ ×
Object Explorer
SELECT
    OrderID,
    OrderDate,
    COUNT(*) OVER() TotalOrders
FROM Sales.Orders
```

206 %

Results Messages

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



```
SQLQuery7.sql - D:\8Q8U\Youtube (72)* ✎ ×
Object Explorer
SELECT
    OrderID,
    OrderDate,
    CustomerID,
    COUNT(*) OVER() TotalOrders,
    COUNT(*) OVER(PARTITION BY CustomerID) OrdersByCustomers
FROM Sales.Orders
```

206 %

	OrderID	OrderDate	CustomerID	TotalOrders	OrdersByCustomers
1	3	2025-01-10	1	10	3
2	4	2025-01-20	1	10	3
3	7	2025-02-15	1	10	3
4	1	2025-01-01	2	10	3
5	5	2025-02-01	2	10	3
6	9	2025-03-10	2	10	3
7	10	2025-03-15	3	10	3
8	6	2025-02-05	3	10	3
9	2	2025-01-05	3	10	3
10	8	2025-02-18	4	10	1

Query executed successfully.



Object Explorer

```
-- Find the total number of Orders
-- Find the total number of Orders for each customers
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    CustomerID,
    COUNT(*) OVER() TotalOrders,
    COUNT(*) OVER(PARTITION BY CustomerID) OrdersByCustomers
FROM Sales.Orders
```

#2 USE CASE

TOTAL PER GROUPS

Group-wise analysis, to understand patterns within different categories

206 %

	OrderID	OrderDate	CustomerID	TotalOrders	OrdersByCustomers
1	3	2025-01-10	1	10	3
2	4	2025-01-20	1	10	3
3	7	2025-02-15	1	10	3
4	1	2025-01-01	2	10	3
5	5	2025-02-01	2	10	3
6	9	2025-03-10	2	10	3
7	10	2025-03-15	3	10	3
8	6	2025-02-05	3	10	3
9	2	2025-01-05	3	10	3
10	8	2025-02-18	4	10	1



Object Explorer

```
-- Find the total number of Customers
-- Additionally provide All customers Details

SELECT
    *,
    COUNT(*) OVER () TotalCustomers
FROM Sales.Customers
```



206 %

	CustomerID	FirstName	LastName	Country	Score	TotalCustomers
1	1	Jossef	Goldberg	Germany	350	5
2	2	Kevin	Brown	USA	900	5
3	3	Mary	NULL	USA	750	5
4	4	Mark	Schwarz	Germany	500	5
5	5	Anna	Adams	USA	NULL	5



SQLQuery8.sql D:\8GBU\Youtube (53).sql

```
-- Find the total number of Customers
-- Find the total number of scores for the customers
-- Additionally provide All customers Details

SELECT
*, 
COUNT(*) OVER () TotalCustomers,
COUNT(Score) OVER() TotaScores
FROM Sales.Customers
```

	CustomerID	FirstName	LastName	Country	Score	TotalCustomers	TotaScores
1	1	Jossef	Goldberg	Germany	350	5	4
2	2	Kevin	Brown	USA	900	5	4
3	3	Mary	NULL	USA	750	5	4
4	4	Mark	Schwarz	Germany	500	5	4
5	5	Anna	Adams	USA	NULL	5	4

SQLQuery8.sql D:\8GBU\Youtube (53).sql

```
-- Find the total number of Customers
-- Find the total number of scores for the customers
-- Additionally provide All customers Details

SELECT
*, 
COUNT(*) OVER () TotalCustomers,
COUNT(Score) OVER() TotaScores
FROM Sales.Customers
```

	CustomerID	FirstName	LastName	Country	Score	TotalCustomers	TotaScores
1	1	Jossef	Goldberg	Germany	350	5	4
2	2	Kevin	Brown	USA	900	5	4
3	3	Mary	NULL	USA	750	5	4
4	4	Mark	Schwarz	Germany	500	5	4
5	5	Anna	Adams	USA	NULL	5	4

#3 USE CASE
DATA QUALITY CHECK

Detecting number of NULLS by comparing to total number of rows

SQLQuery8.sql D:\8GBU\Youtube (53).sql

```
-- Find the total number of Customers
-- Find the total number of scores for the customers
-- Additionally provide All customers Details

SELECT
*, 
COUNT(*) OVER () TotalCustomers,
COUNT(Score) OVER() TotalScores,
COUNT(Country) OVER() TotalCountries
FROM Sales.Customers
```

	CustomerID	FirstName	LastName	Country	Score	TotalCustomers	TotalScores	TotalCountries
1	1	Jossef	Goldberg	Germany	350	5	4	5
2	2	Kevin	Brown	USA	900	5	4	5
3	3	Mary	NULL	USA	750	5	4	5
4	4	Mark	Schwarz	Germany	500	5	4	5
5	5	Anna	Adams	USA	NULL	5	4	5

$5 - 5 = 0$

No NULLs in Column Country

```

SQLQuery8.sql D:\BOBUTYouTube (S3) * x
-- Find the total number of Customers
-- Find the total number of scores for the customers
-- Additionally provide All customers Details

SELECT
    *,
    COUNT(*) OVER () TotalCustomersStar,
    COUNT(1) OVER () TotalCustomersOne,
    COUNT(Score) OVER() TotalScores,
    COUNT(Country) OVER() TotalCountries
FROM Sales.Customers

```

	CustomerID	FirstName	LastName	Country	Score	TotalCustomersStar	TotalCustomersOne	TotalScores	TotalCountries
1	1	Jossef	Goldberg	Germany	350	5	5	4	5
2	2	Kevin	Brown	USA	900	5	5	4	5
3	3	Mary	NULL	USA	750	5	5	4	5
4	4	Mark	Schwarz	Germany	500	5	5	4	5
5	5	Anna	Adams	USA	NULL	5	5	4	5

We have to handle data quality issues and we have to clean it otherwise we will get bad result of analysis

DATA QUALITY ISSUE

Duplicates leads to inaccuracies in analysis

We used count function to check data duplicate

```
-- Check whether the table 'orders' contains any duplicate rows
SELECT
    OrderID,
    COUNT(*) OVER (PARTITION BY OrderID)
FROM Sales.Orders
```



Divides the data by the primary Key (OrderID)

EXPECTATION

Maximum number of rows for each window (ID) = 1

```
-- Check whether the table 'orders' contains any duplicate rows
SELECT
    OrderID,
    COUNT(*) OVER (PARTITION BY OrderID) CheckPK
FROM Sales.Orders
```

	OrderID	CheckPK
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1

Values of OrderID are unique

Object Explorer

```
-- Check whether the table 'orders' contains any duplicate rows
SELECT
    OrderID,
    COUNT(*) OVER (PARTITION BY OrderID) CheckPK
FROM Sales.OrdersArchive
```

	OrderID	CheckPK
1	1	1
2	2	1
3	3	1
4	4	2
5	4	2
6	5	1
7	6	3
8	6	3
9	6	3
10	7	1

Duplicates in OrderID

SQL Aggregate Window Functions | COUNT AVG SUM MAX MIN | #SQL Course 23

```
-- Check whether the table 'orders' contains any duplicate rows
SELECT
    *
FROM (
    SELECT
        OrderID,
        COUNT(*) OVER (PARTITION BY OrderID)
    FROM Sales.OrdersArchive
) t WHERE CheckPK > 1
```

	OrderID	CheckPK
1	4	2
2	4	2
3	6	3
4	6	3
5	6	3

#4 USE CASE
IDENTIFY DUPLICATES
Identify duplicate rows to improve data quality

COUNT | USE CASES

#1 Overall Analysis

#2 Category Analysis

#3 Quality Checks: Identify NULLs

#4 Quality Checks: Identify Duplicates

SUM()

Returns the sum of values within a window

SUM(Sales) OVER(PARTITION BY Product)

TASK

Find the total sales
for each product



Product	Sales	SUM
Caps	20	35
Caps	10	35
Caps	5	35
Gloves	30	100
Gloves	70	100
Gloves	NULL	100

$$20 + 10 + 5 = 35$$

$$30 + 70 = 100$$



SUM(Sales) OVER(PARTITION BY Product)

RULE

SUM() Accepts
Only Numbers



Product	Sales	SUM
Caps	20	35
Caps	10	35
Caps	5	35
Gloves	30	100
Gloves	70	100
Gloves	NULL	100

$$20 + 10 + 5 = 35$$

$$30 + 70 = 100$$

SalesDB

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

	OrderID	OrderDate	Sales	TotalSales
1	1	2025-01-01	10	380
2	2	2025-01-05	15	380
3	3	2025-01-10	20	380
4	4	2025-01-20	60	380
5	5	2025-02-01	25	380
6	6	2025-02-05	50	380
7	7	2025-02-15	30	380
8	8	2025-02-18	90	380
9	9	2025-03-10	20	380
10	10	2025-03-15	60	380

Query executed successfully.



SalesDB

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

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



Object Explorer SQLQuery9.sql - D...80BU\Youtube (S8)* SQLQuery8.sql - D...80BU\Youtube (S3)*

```
-- Find the total sales across all orders
-- And the total sales for each product
-- Additionally provide details such order Id, order date, etc.
#2 USE CASE
TOTAL PER GROUPS
Group-wise analysis, to understand
patterns within different categories
```

SELECT
OrderID,
OrderDate,
Sales,
ProductID,
SUM(Sales) OVER () TotalSales,
SUM(Sales) OVER (PARTITION BY ProductID) SalesByProducts
FROM Sales.Orders

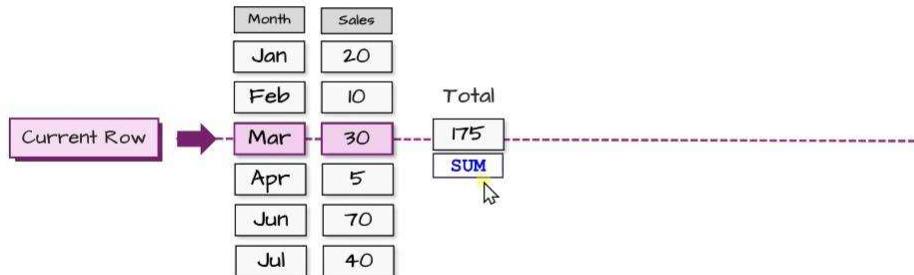
	OrderID	OrderDate	Sales	ProductID	TotalSales	SalesByProducts
1	1	2025-01-01	10	101	380	140
2	3	2025-01-10	20	101	380	140
3	8	2025-02-18	90	101	380	140
4	9	2025-03-10	20	101	380	140
5	10	2025-03-15	60	102	380	105
6	2	2025-01-05	15	102	380	105
7	7	2025-02-15	30	102	380	105
8	5	2025-02-01	25	104	380	75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

Query executed successfully.



Helps to analyzes the product

Compare the current value and aggregated value of window functions



Part-to-Whole Analysis

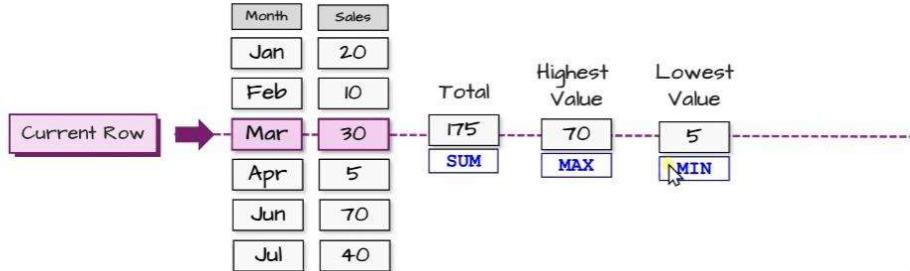
Compare current sales to total sales



How important the sales of this month to the total sales this analysis goes here

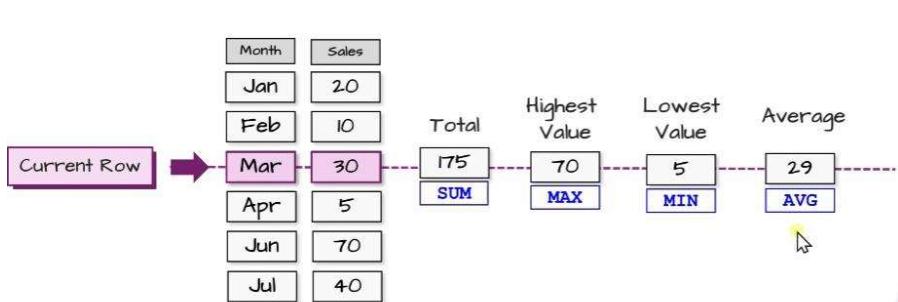
Comparision Use Cases

Compare the **current value** and aggregated value of **window functions**



Compare to Extremes Analysis

Compare current sales to the highest or lowest sales



Compare to Average Analysis

Help to evaluate whether a value is above or below the average

Its important to understand the performance of data

Sales.Employees

Sales.Orders

Columns

- OrderID (PK, int, not null)
- ProductID (int, null)
- CustomerID (int, null)
- SalesPersonID (int, null)
- OrderDate (date, null)
- ShipDate (date, null)
- OrderStatus (varchar(10), null)
- ShipAddress (varchar(60), null)
- BillAddress (varchar(60), null)
- Quantity (int, null)
- Sales (int, null)**
- CreationTime (date, null)

Keys

Constraints

PROBLEM

```
-- Find the percentage contribution of each product's sales to the total sale
SELECT
    OrderID,
    ProductID,
    Sales,
    SUM(Sales) OVER () TotalSales,
    Sales / SUM(Sales) OVER () * 100
FROM Sales.orders
```

$$\text{Percentage Contribution} = \left(\frac{\text{Product's Sales}}{\text{Total Sales}} \right) \times 100$$

	OrderID	ProductID	Sales	TotalSales	(No column name)
1	101	10	380	0	
2	102	15	380	0	
3	101	20	380	0	
4	105	60	380	0	
5	104	25	380	0	
6	104	50	380	0	
7	102	30	380	0	
8	101	90	380	0	



Dividing two integer columns produces an integer, not a decimal!

Sales.OrdersArchive
Sales.Products

Object Explorer

Connect * SQL Server

DESKTOP-848BQBU\SQLEXPRESS (SQL Server)

Databases

Tables

Views

Materialized Views

External Tables

Graph Tables

Customers

Employees

Orders

OrderID (PK, int, not null)

ProductID (int, null)

CustomerID (int, null)

SalesPersonID (int, null)

OrderDate (date, null)

ShipDate (date, null)

OrderStatus (varchar(10), null)

ShipAddress (varchar(60), null)

BillAddress (varchar(60), null)

Quantity (int, null)

Sales (int, null)

CreationTime (date, null)

Keys

Constraints

Triggers

Indexes

Statistics

Sales.Orders

OrderID

ProductID

Sales

SUM(Sales) OVER () TotalSales

ROUND (CAST (Sales AS Float) / SUM(Sales) OVER () * 100, 2) PercentageOfTotal

```
-- Find the percentage contribution of each product's sales to the total sale:
SELECT
    OrderID,
    ProductID,
    Sales,
    SUM(Sales) OVER () TotalSales,
    ROUND (CAST (Sales AS Float) / SUM(Sales) OVER () * 100, 2) PercentageOfTotal
FROM Sales.orders
```

	OrderID	ProductID	Sales	TotalSales	PercentageOfTotal
1	1	101	10	380	2,63
2	2	102	15	380	3,95
3	3	101	20	380	5,26
4	4	105	60	380	15,79
5	5	104	25	380	6,58
6	6	104	50	380	13,16
7	7	102	30	380	7,89
8	8	101	90	380	23,68
9	9	101	20	380	5,26
10	10	102	60	380	15,79



Window function helps to compare performance of current value from the total value it helps to check performance

AVG ()

Returns the average of values within a window

AVG(Sales) OVER(PARTITION BY Product)

Product	Sales
Caps	20
Caps	10
Caps	5

AVG

$$(20 + 10 + 5) / 3 = ||$$

Gloves	30
Gloves	70
Gloves	NULL

50
50
50

$$(30 + 70) / 2 = 50$$

TASK
Find the Average Sales
for each product.



Before finding avg we have to handle null

AVG(Sales) OVER(PARTITION BY Product)

TASK

Find the Average Sales
for each product.

NULL Sales means ZERO.



Product	Sales
Caps	20
Caps	10
Caps	5

Avg

$$(20 + 10 + 5) / 3 = ||$$

Gloves	30
Gloves	70
Gloves	NULL

50
50
50

$$(30 + 70) / 2 = 50$$



AVG(COALESCE(Sales,0)) OVER(PARTITION BY Product)

TASK

Find the Average Sales
for each product.

NULL Sales means ZERO.



Product	Sales
Caps	20
Caps	10
Caps	5

Avg

$$(20 + 10 + 5) / 3 = ||$$

Gloves	30
Gloves	70
Gloves	0

33
33
33

$$(30 + 70 + 0) / 3 = 33$$



Object Explorer

SQLQuery100 - DESKTOP-8488QBU\SQLEXPRESS [AdventureworksDW2022]

```
-- Find the average sales across all orders
-- And Find the average sales for each product
-- Additionally provide details such order Id, #2 USE CASE
SELECT
    OrderID,
    OrderDate,
    Sales,
    ProductID,
    AVG(Sales) OVER () AvgSales,
    AVG(Sales) OVER (PARTITION BY ProductID) AvgSalesByProducts
FROM Sales.Orders
```

TOTAL PER GROUPS

Group-wise analysis, to understand patterns within different categories

OrderID	OrderDate	Sales	ProductID	AvgSales	AvgSalesByProducts
1	2025-01-01	10	101	38	35
2	2025-01-10	20	101	38	35
3	2025-02-18	90	101	38	35
4	2025-03-10	20	101	38	35
5	2025-03-15	60	102	38	35
6	2025-01-05	15	102	38	35
7	2025-02-15	30	102	38	35
8	2025-02-01	25	104	38	37
9	2025-02-05	50	104	38	37
10	2025-01-20	60	105	38	60

DESKTOP-8488QBU\SQLEXPRESS ... DESKTOP-8488QBU\You

DESKTOP-8488QBU\SQLEXPRESS -> DESKTOP-8488QBU\You

DESKTOP-8488QBU\SQLEXPRESS -> DESKTOP-8488QBU\You

SalesDB

```
-- Find the average scores of customers
-- Additionally provide details such CustomerID and LastName

SELECT
    CustomerID,
    LastName,
    Score,
    COALESCE
        COALESC
            BCFCOLLATIONNAME
                BIT_COUNT
                CERTENCODED
                CHANGE_TRACKING_IS_COLUMN_IN_MASK
                COL_LENGTH
                COL_NAME
FROM Sales.Customers
```

	CustomerID	COLLATIONNAME	COLLATIONPROPERTY	COLLATIONPROPERTYFROMID
1	1	Brown	900	625
2	2	NULL	750	625
3	3	Schwartz	500	625
4	COALESCE	Repalces NULLs with specified value		

Object Explorer

```
-- Find the average scores of customers
-- Additionally provide details such CustomerID and LastName

SELECT
    CustomerID,
    LastName,
    Score,
    COALESCE(Score,0) CustomerScore,
    AVG(Score) OVER () AvgScore,
    AVG(COALESCE(Score,0)) OVER () AvgScoreWithoutNull
FROM Sales.Customers
```

	CustomerID	LastName	Score	CustomerScore	AvgScore	AvgScoreWithoutNull
1	1	Goldberg	350	350	625	500
2	2	Brown	900	900	625	500
3	3	NULL	750	750	625	500
4	4	Schwartz	500	500	625	500
5	5	Adams	NULL	0	625	500

We need to find the current sales here

```
--Find all orders where sales are higher than the average sales across all orders
```

```
SELECT
    *
FROM()
SELECT
    OrderID,
    ProductID,
    Sales,
    AVG(Sales) OVER() AvgSales
FROM Sales.Orders
```

	OrderID	ProductID	Sales	AvgSales
1	101	10	38	
2	102	15	38	
3	101	20	38	
4	105	60	38	
5	104	25	38	
6	104	50	38	
7	102	30	38	
8	101	90	38	
9	101	20	38	
0	102	60	38	

WINDOW RULE

Window functions can't be used in the WHERE clause

--Find all orders where sales are higher than the average sales across all orders

```

SELECT
*
FROM(
    SELECT
        OrderID,
        ProductID,
        Sales,
        AVG(Sales) OVER() AvgSales
    FROM Sales.Orders
)t WHERE Sales > AvgSales

```

	OrderID	ProductID	Sales	AvgSales
1	4	105	60	38
2	6	104	50	38
3	8	101	90	38
4	10	102	60	38

#3 USE CASE COMPARE TO AVERAGE

Helps to evaluate whether a value is above or below the average



MIN ()

Returns the lowest value within a window

MAX ()

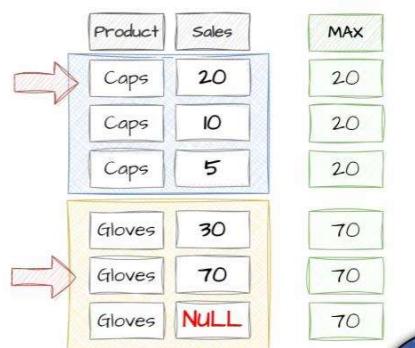
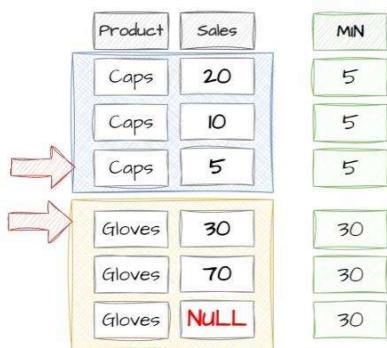
Returns the highest value within a window

Find the highest sales for each product

Find the lowest sales for each product

MIN(Sales) OVER(PARTITION BY Product)

MAX(Sales) OVER(PARTITION BY Product)



Our business scenario

Find the highest sales for each product

```
MIN(Sales) OVER(PARTITION BY Product)
```

Product	Sales
Caps	20
Caps	10
Caps	5

MN
5
5
5

Find the lowest sales for each product

```
MAX(Sales) OVER(PARTITION BY Product)
```

Product	Sales
Caps	20
Caps	10
Caps	5

MAX
20
20
20



#1 USE CASE

OVERALL ANALYSIS

Quick summary or snapshot
of the entire dataset

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



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



