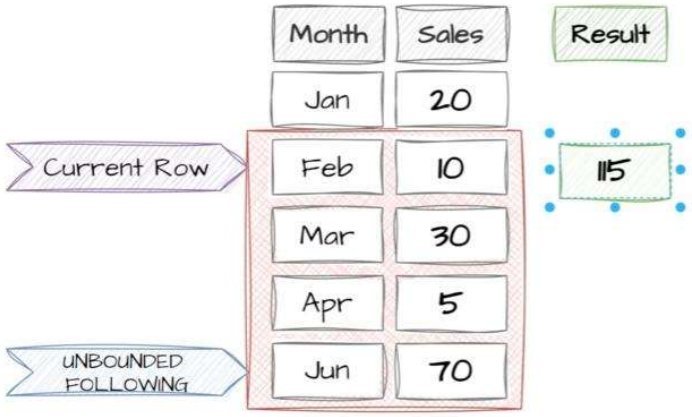
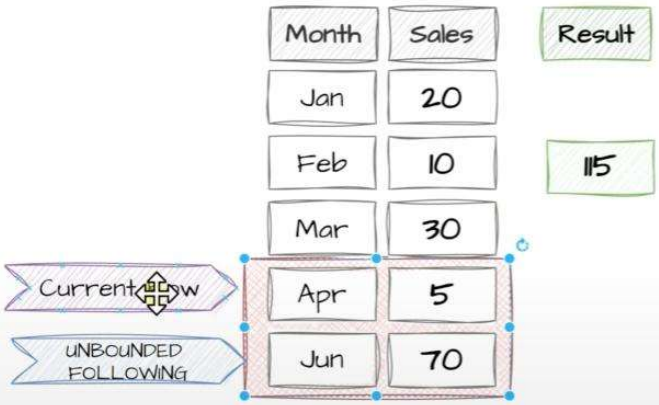
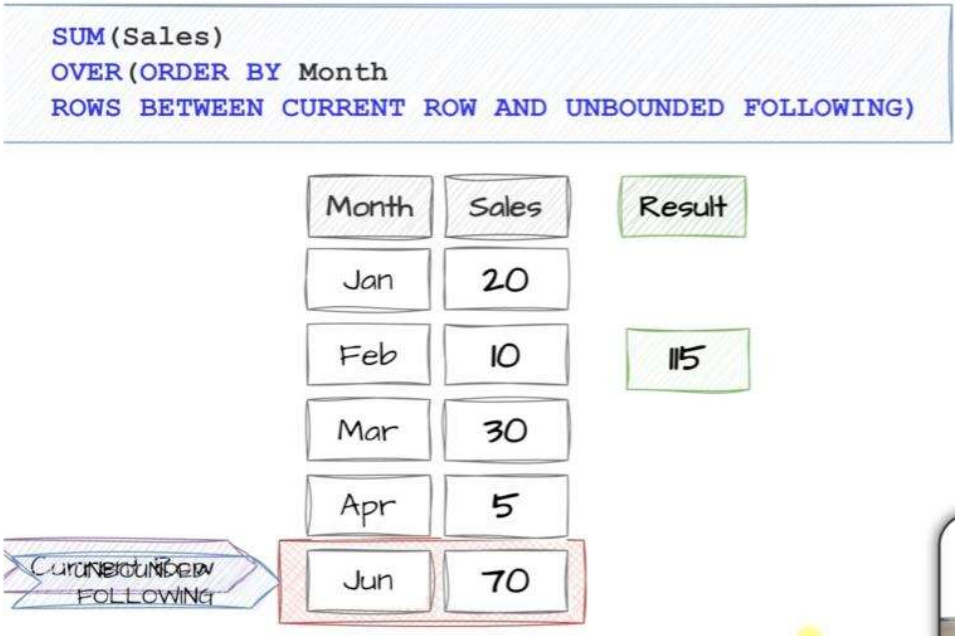
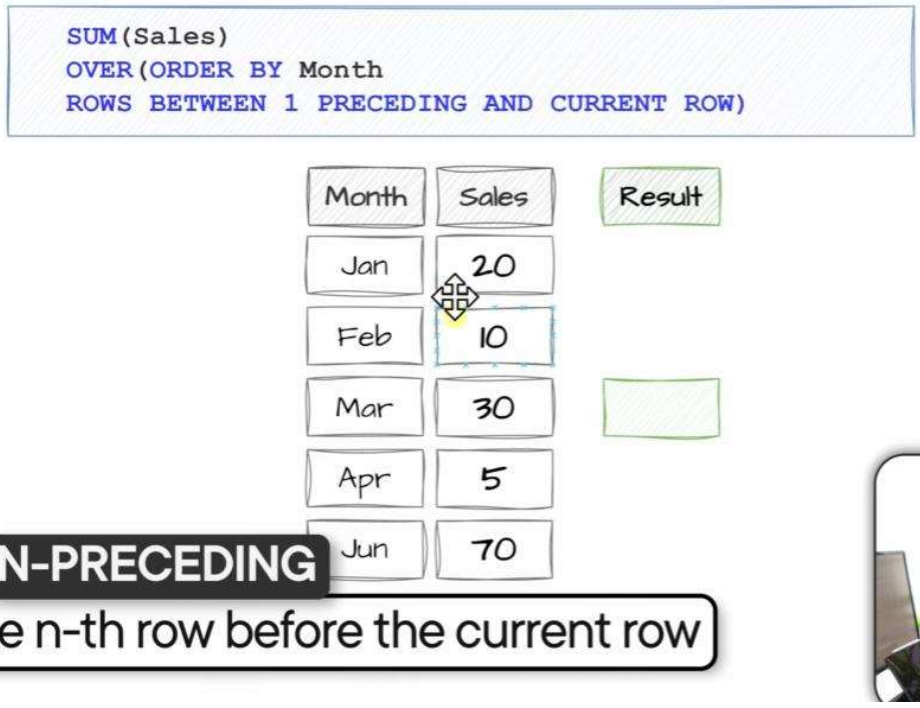
```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
```



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







Targetting row before current row

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

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

1 PRECEDING

Current Row

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

Month	Sales	Result
Jan	20	
Feb	10	
Mar	30	
Apr	5	

UNBOUNDED PRECEDING

The first possible row within a window

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

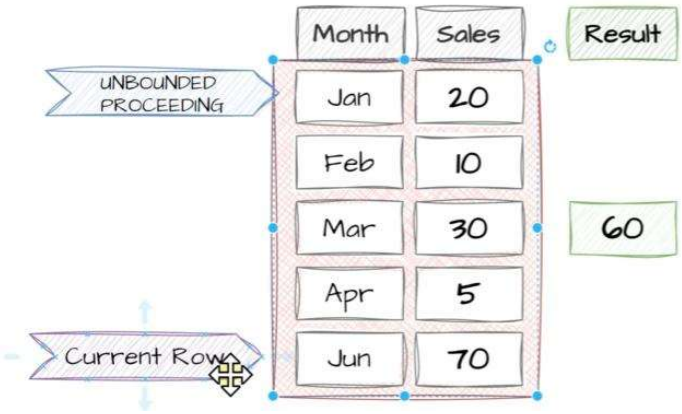
Month	Sales	Result
Jan	20	
Feb	10	
Mar	30	60
Apr	5	
Jun	70	

UNBOUNDED PRECEEDING

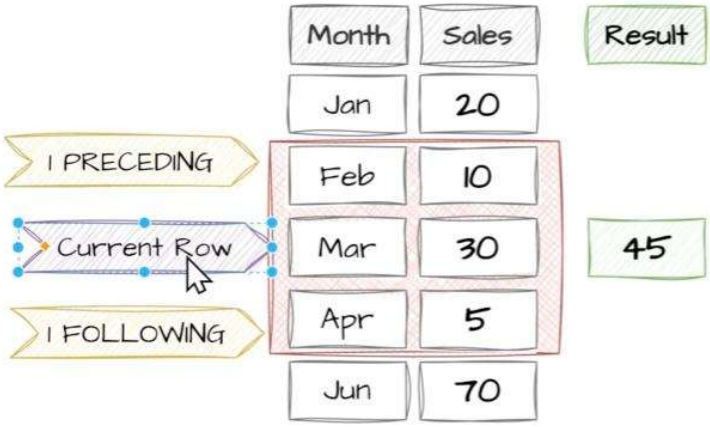
Current Row

940, 547

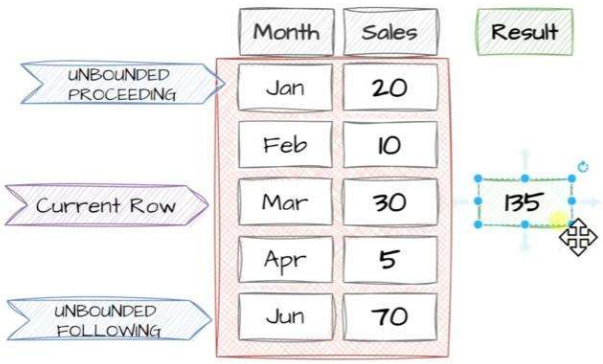
```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
```



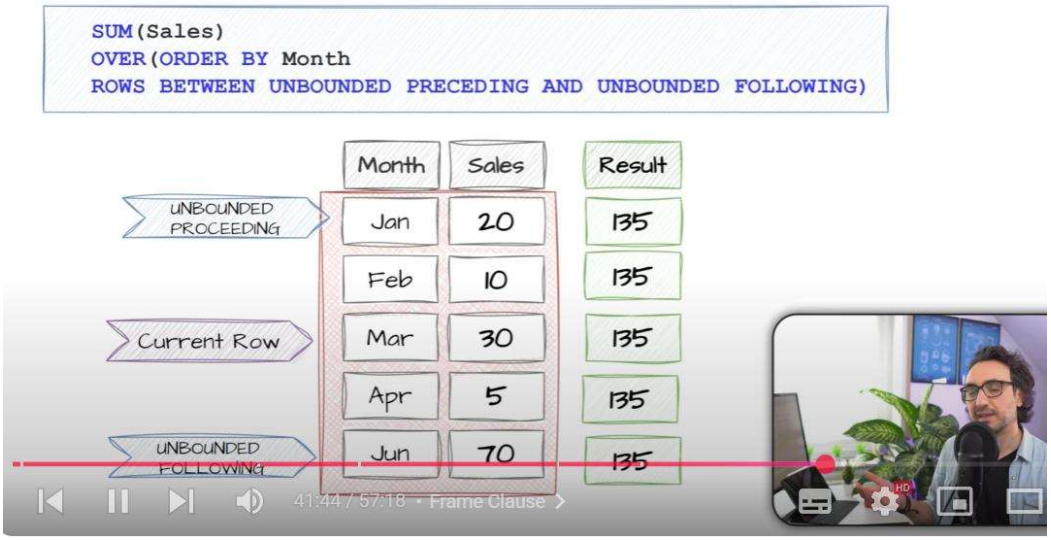
```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
```

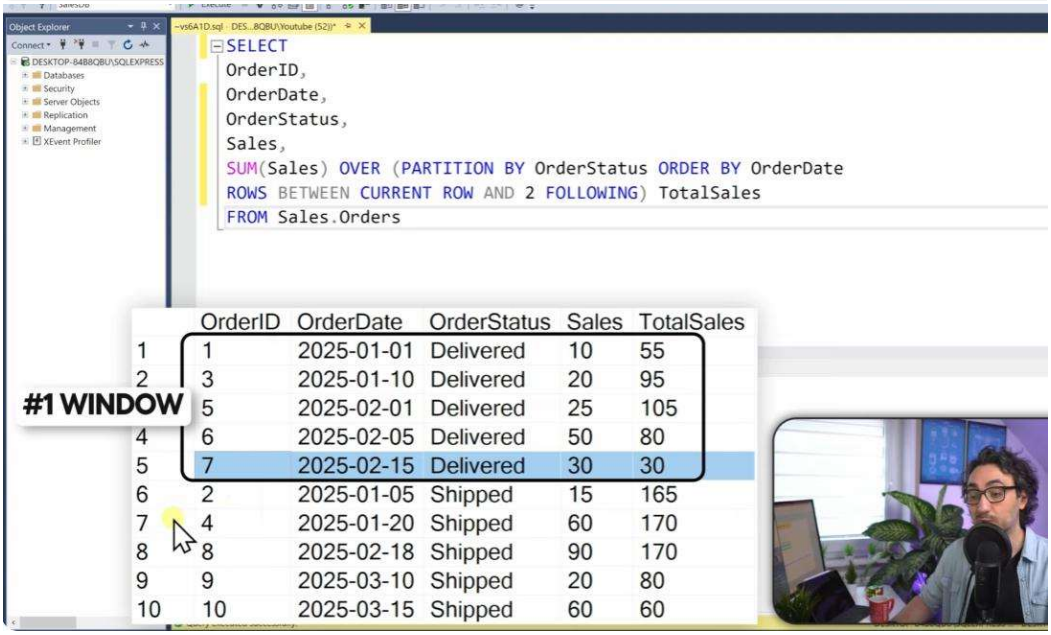


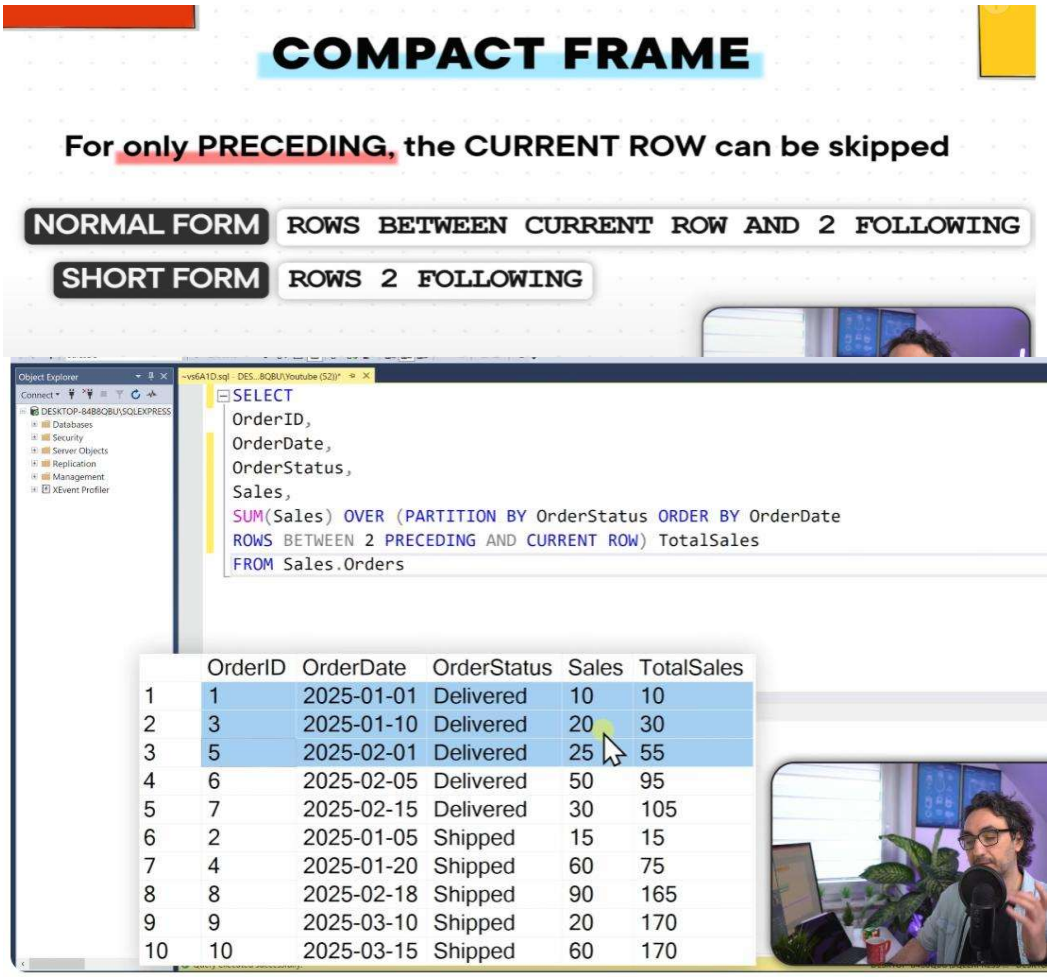
```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
```



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

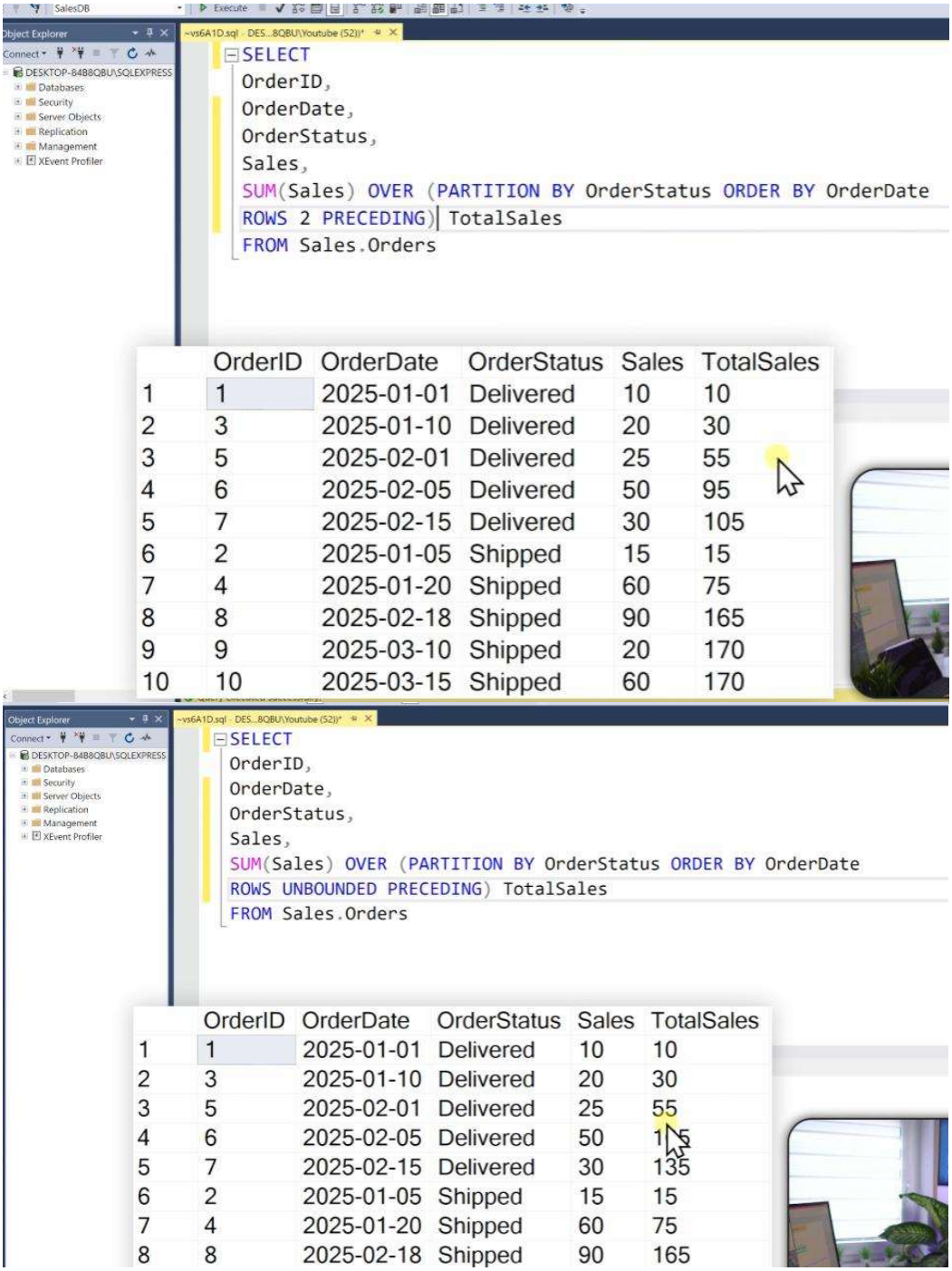


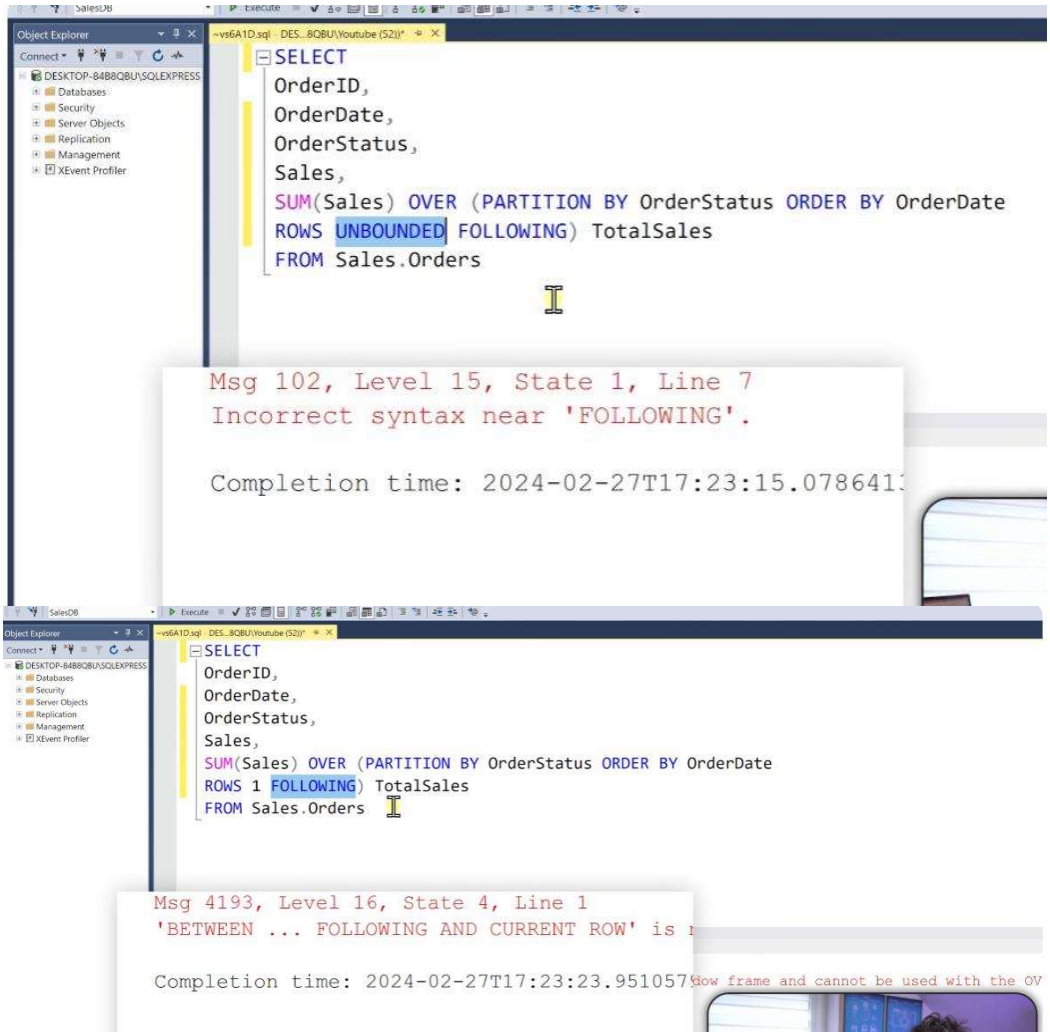




Shortcut work for preceding only else it throws error

Compact frame works only for preceding


SalesDB
Execute
Object Explorer
Connect
DESKTOP-84B8QBU\SQLEXPRESS
Databases
Security
Server Objects
Replication
Management
XEvent Profiler
~vs6A1D.sql - DES...8QBU\Youtube (52))*
SELECT
OrderID,
OrderDate,
OrderStatus,
Sales,
SUM(Sales) OVER (PARTITION BY OrderStatus ORDER BY OrderDate
ROWS 2 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 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
Connect
DESKTOP-84B8QBU\SQLEXPRESS
Databases
Security
Server Objects
Replication
Management
XEvent Profiler
~vs6A1D.sql - DES...8QBU\Youtube (52))*
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 1
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

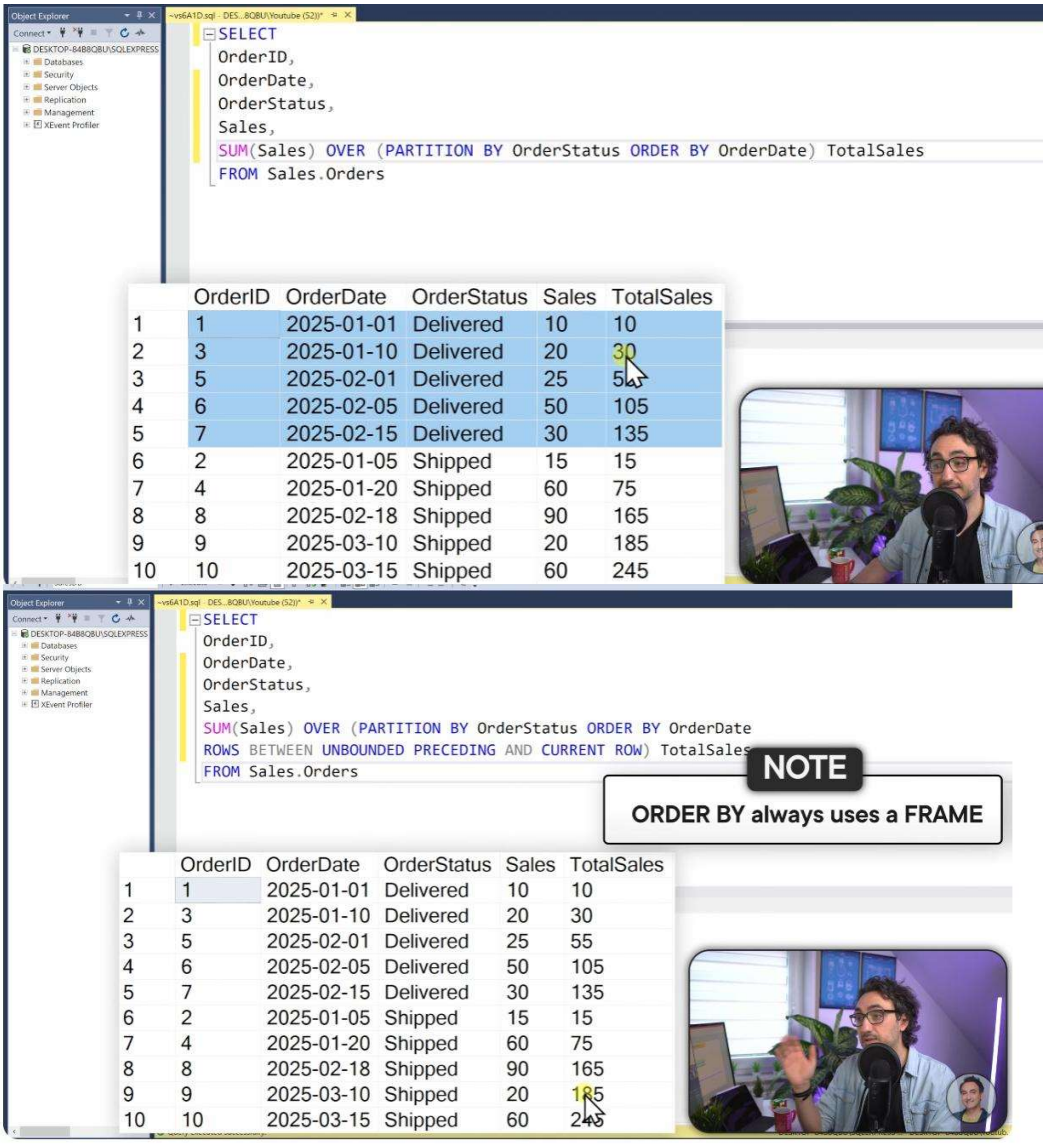


DEFAULT FRAME

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

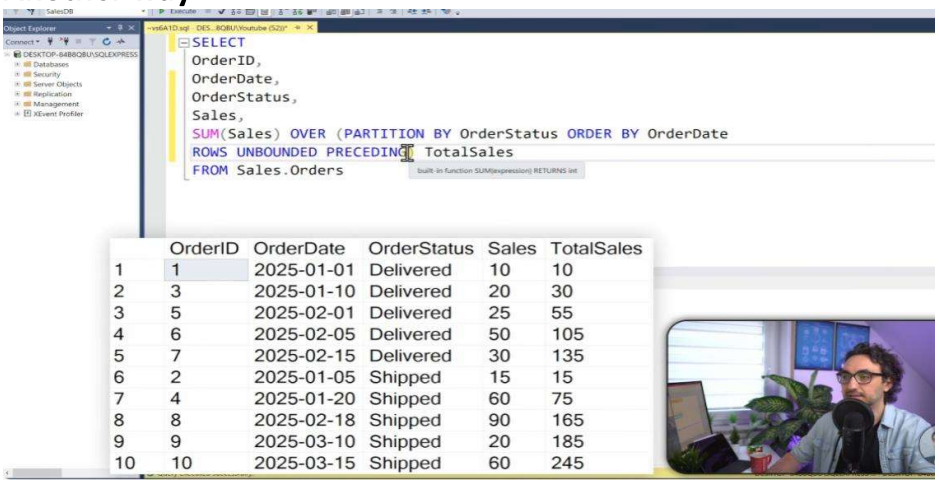
There is a hidden frame used while we used order by that is defauly frame



	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

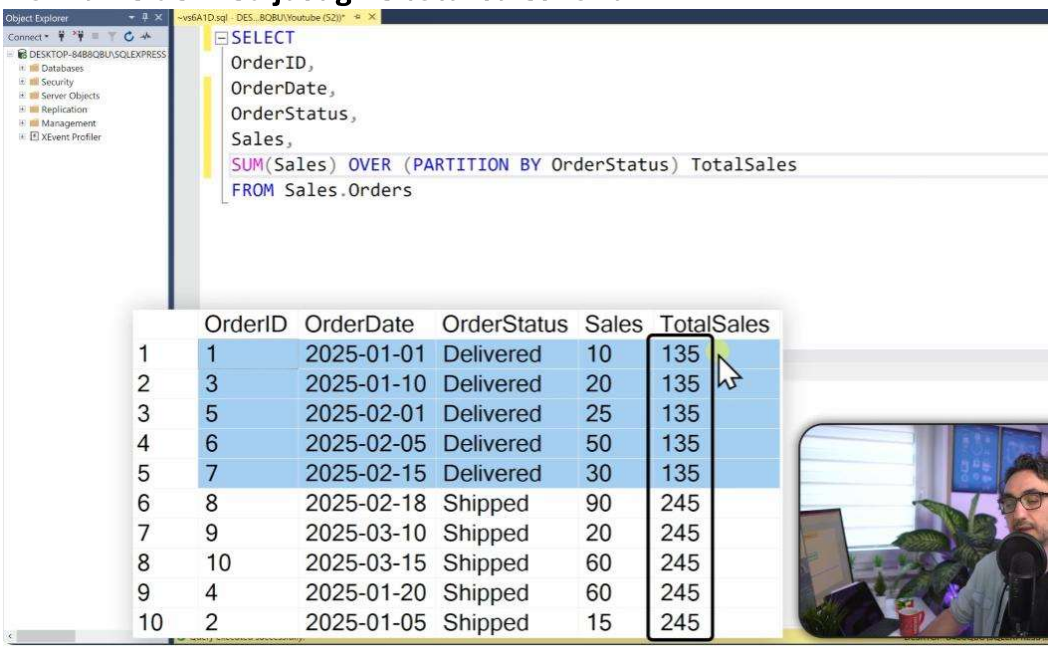
SQL Window Functions Basics | Partition By, Order By, Frame | #SQL Course 22

Another way



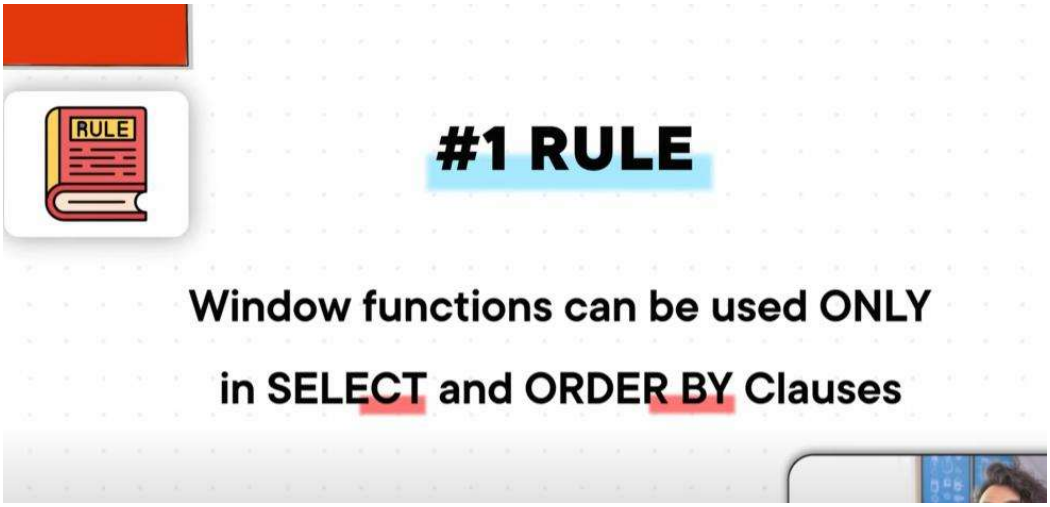
	OrderID	OrderDate	OrderStatus	Sales	TotalSales
1	1	2025-01-01	Delivered	10	10
2	3	2025-01-10	Delivered	20	30
3	5	2025-02-01	Delivered	25	55
4	6	2025-02-05	Delivered	50	105
5	7	2025-02-15	Delivered	30	135
6	2	2025-01-05	Shipped	15	15
7	4	2025-01-20	Shipped	60	75
8	8	2025-02-18	Shipped	90	165
9	9	2025-03-10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

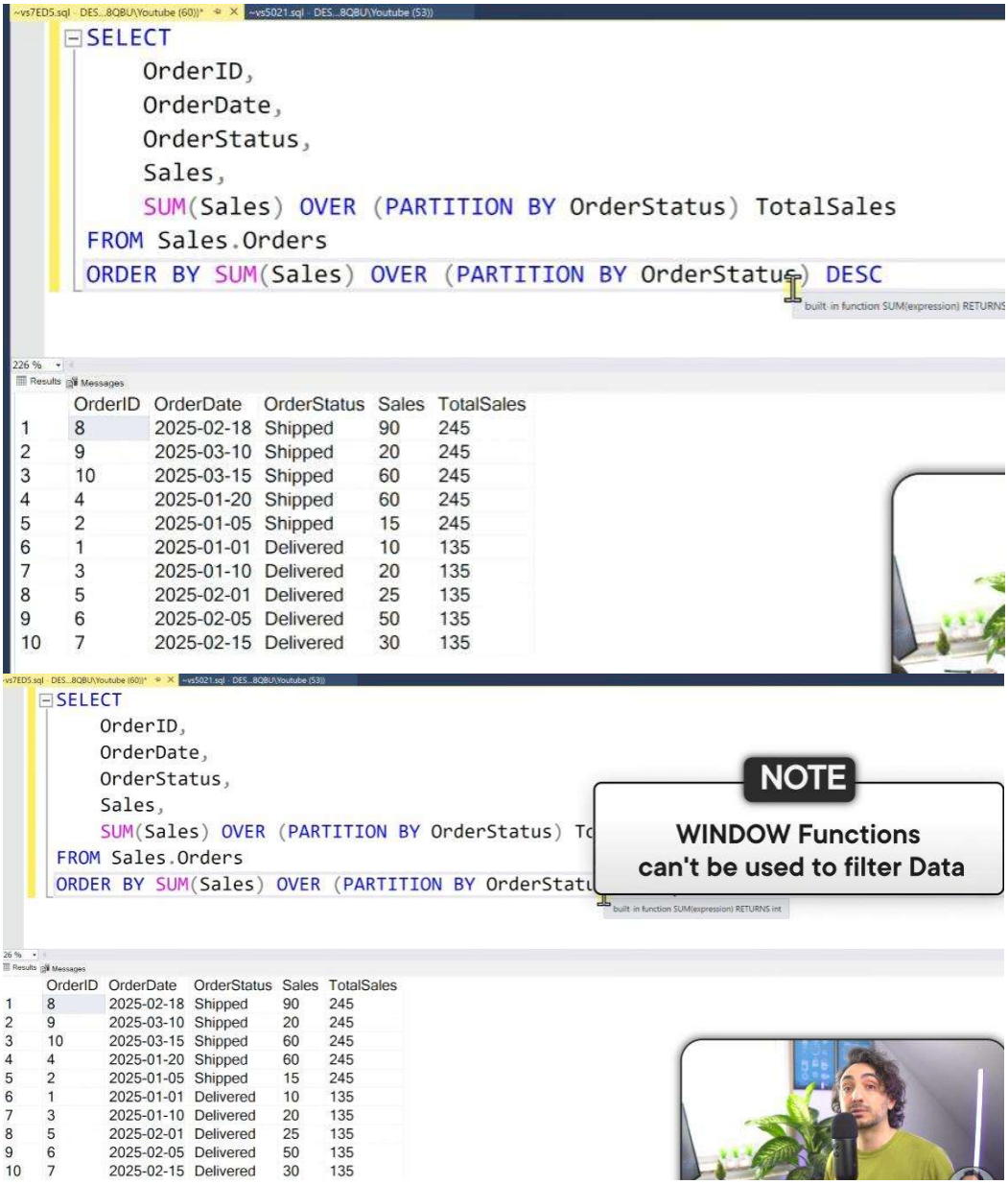
No Frame defined just give total sales for all



Total sales highest value to lowest value

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





	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

```sql
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(Sales) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
WHERE SUM(Sales) OVER (PARTITION BY OrderStatus) > 100
```

```
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

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

```
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
```

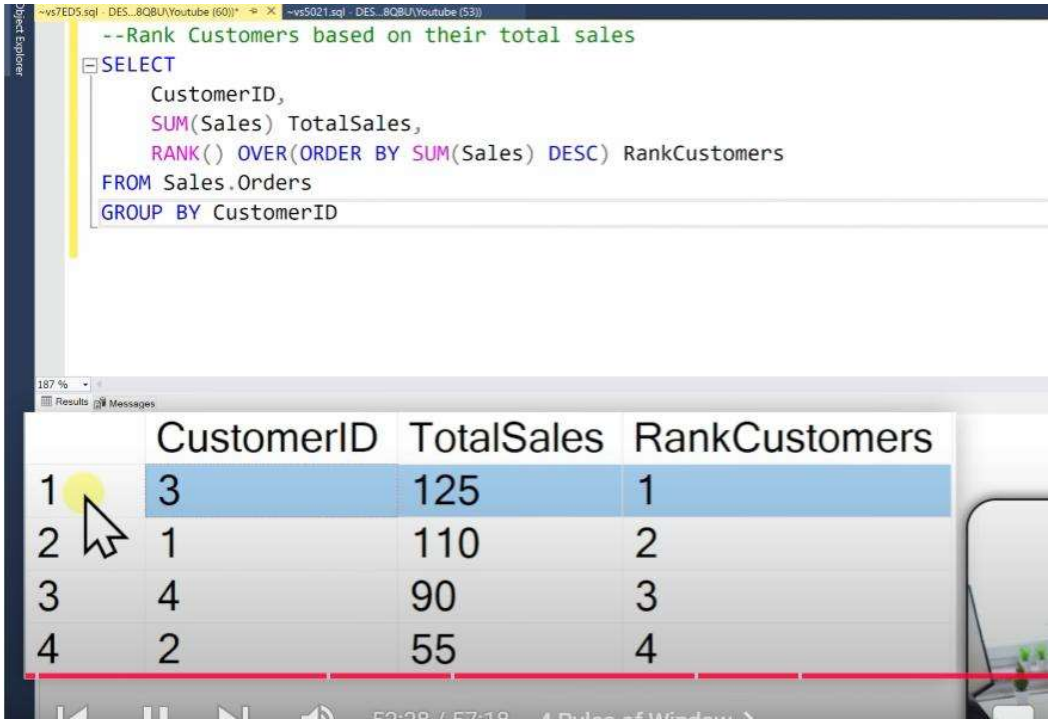
```sql
SELECT
    OrderID,
    OrderDate,
    OrderStatus,
    Sales,
    SUM(SUM(Sales) OVER (PARTITION BY OrderStatus)) OVER (PARTITION BY OrderStatus) TotalSales
FROM Sales.Orders
```

```
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
```

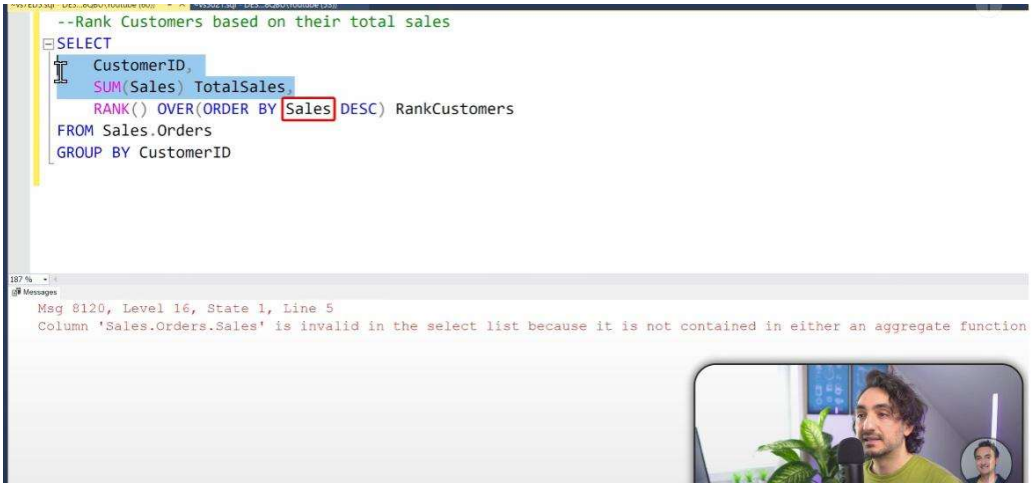
First Filtering then Aggregation

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

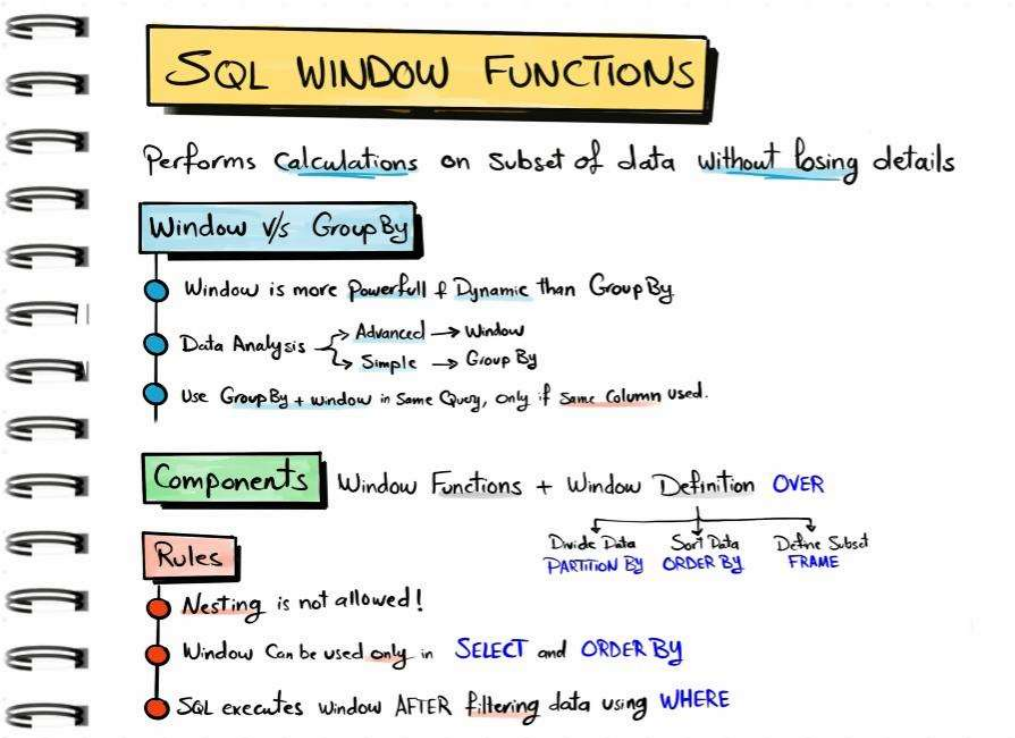
	OrderID	OrderDate	OrderStatus	ProductID	Sales	TotalSales
1	3	2025-01-10	Delivered	101	20	60
2	7	2025-02-15	Delivered	102	30	60
3	1	2025-01-01	Delivered	101	10	60
4	2	2025-01-05	Shipped	102	15	185
5	8	2025-02-18	Shipped	101	90	185
6	9	2025-03-10	Shipped	101	20	185
7	10	2025-03-15	Shipped	102	60	185



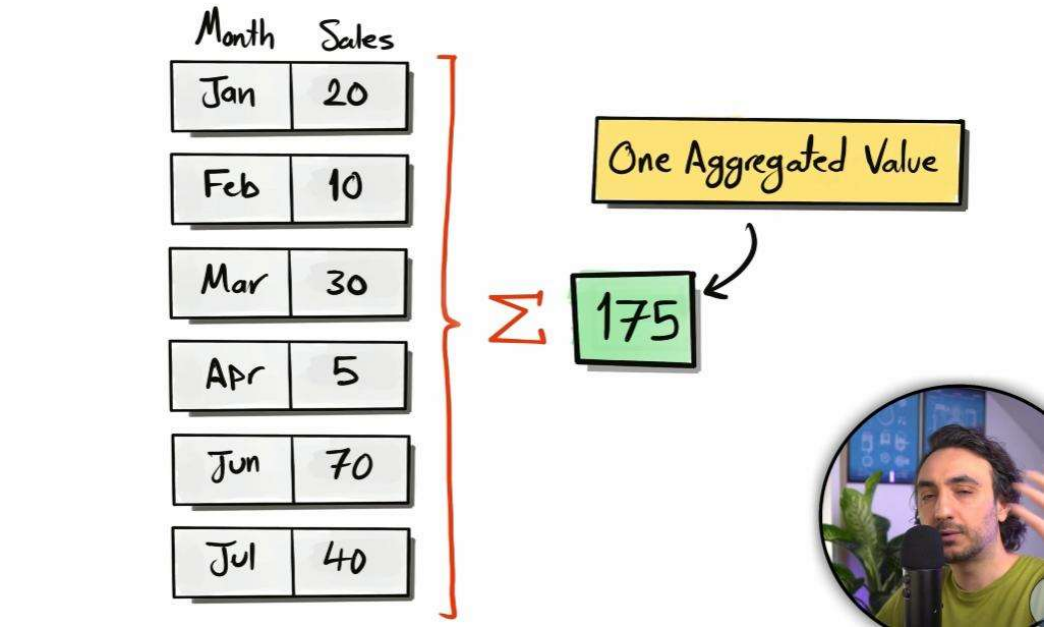
Anlything that you used inside window function should be a part of group by



Error because sales not part of group by



Aggregated window function:



</> Aggregate Functions | Syntax

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

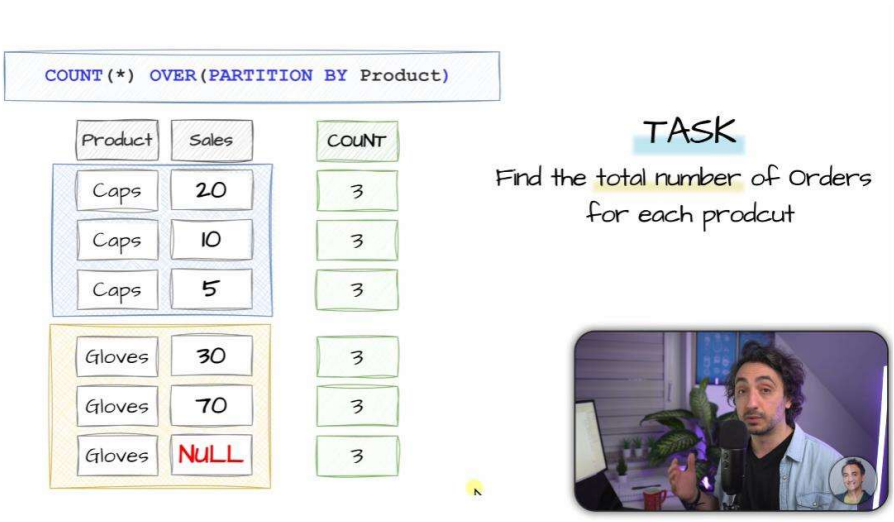
- `AVG(Sales)` — Expression is required (Only Numeric Values)
- `PARTITION BY ProductID` — Partition By is Optional
- `ORDER BY Sales` — Order By is Optional

</> Aggregate Functions | Syntax

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

COUNT ( )

Returns the number of rows within a window



Sql count null as one row

COUNT (*)

counts all the rows in a table,

regardless of whether any value is NULL

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	COUNT
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	3
Gloves	70	3
Gloves	NULL	3

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

Product	Sales	COUNT
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	2
Gloves	70	2
Gloves	NULL	2

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	COUNT
Caps	20	3
Caps	10	3
Caps	5	3
Gloves	30	3
Gloves	70	3
Gloves	NULL	3

NOTE

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

It helps to understand data

```
-- Find the total number of Orders
SELECT
COUNT(*) TotalOrders
FROM Sales.Orders
```

	TotalOrders
1	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 aggre
```

```sql
-- Find the total number of Orders
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    COUNT(*) OVER() TotalOrders
FROM Sales.Orders
```

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

```sql
-- 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
```

	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.

```sql
-- 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

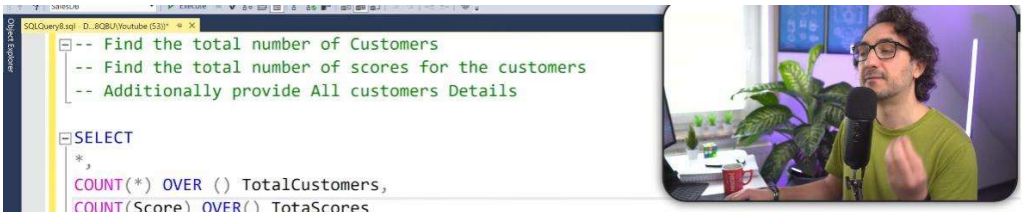
	OrderID	OrderDate	CustomerID	TotalOrders	OrdersByCustomers
1	3	2025-01-10	1	10	3
2	4	2025-01-20	1	10	3
3	7	2025-02-15	1	10	3
4	1	2025-01-01	2	10	3
5	5	2025-02-01	2	10	3
6	9	2025-03-10	2	10	3
7	10	2025-03-15	3	10	3
8	6	2025-02-05	3	10	3
9	2	2025-01-05	3	10	3
10	8	2025-02-18	4	10	1

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

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

	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

```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




```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

5 - 4 = 1

1 NULL in Scores

#3 USE CASE

DATA QUALTIY CHECK

Detecting number of NULLS by comparing to total number of rows

```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

```sql
-- 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

We used count function to check data duplicate

```sql
-- 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)

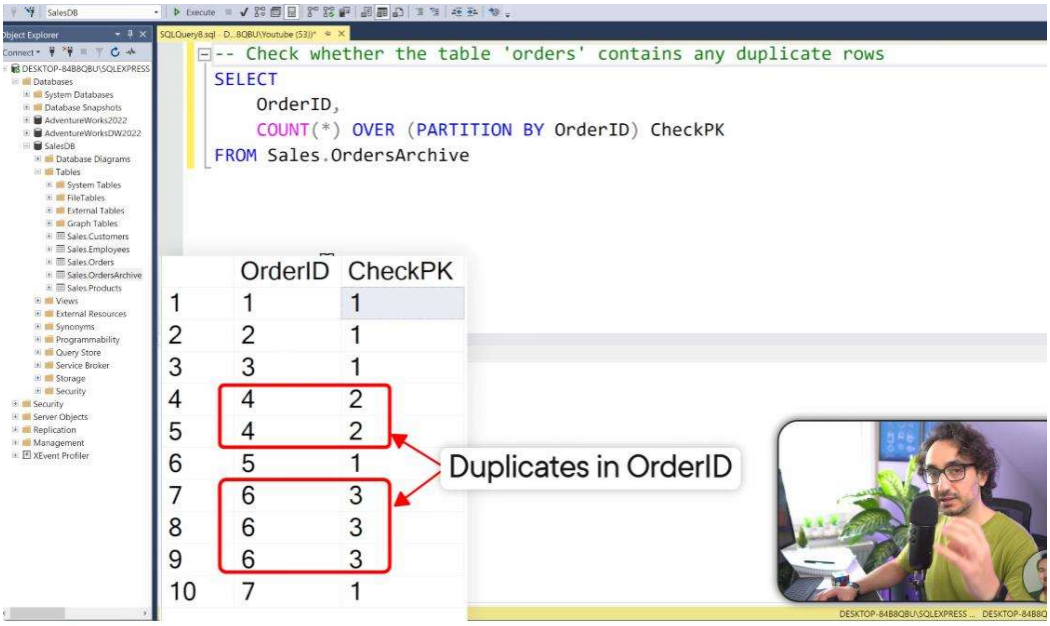
```sql
-- Check whether the table 'orders' contains any duplicate rows
SELECT
    OrderID,
    COUNT(*) OVER (PARTITION BY OrderID) CheckPK
FROM Sales.Orders
```

EXPECTATION

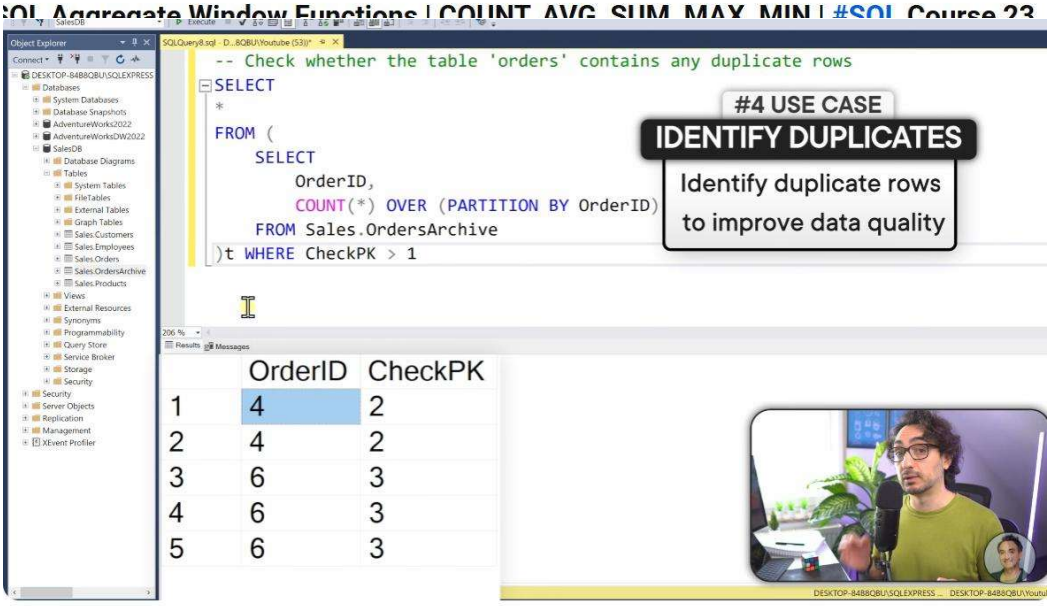
Maximum number of rows for each window (ID) = 1

	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



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



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)
```

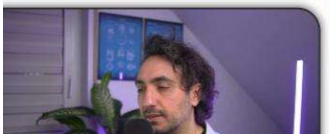
TASK

Find the total sales for each prodcut

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)
```

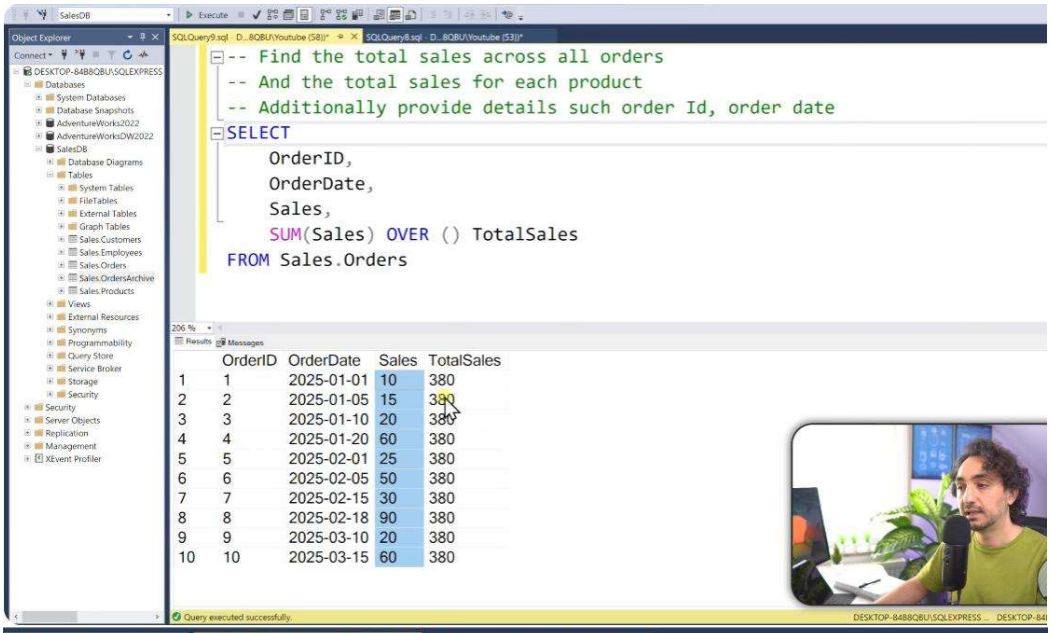

RULE

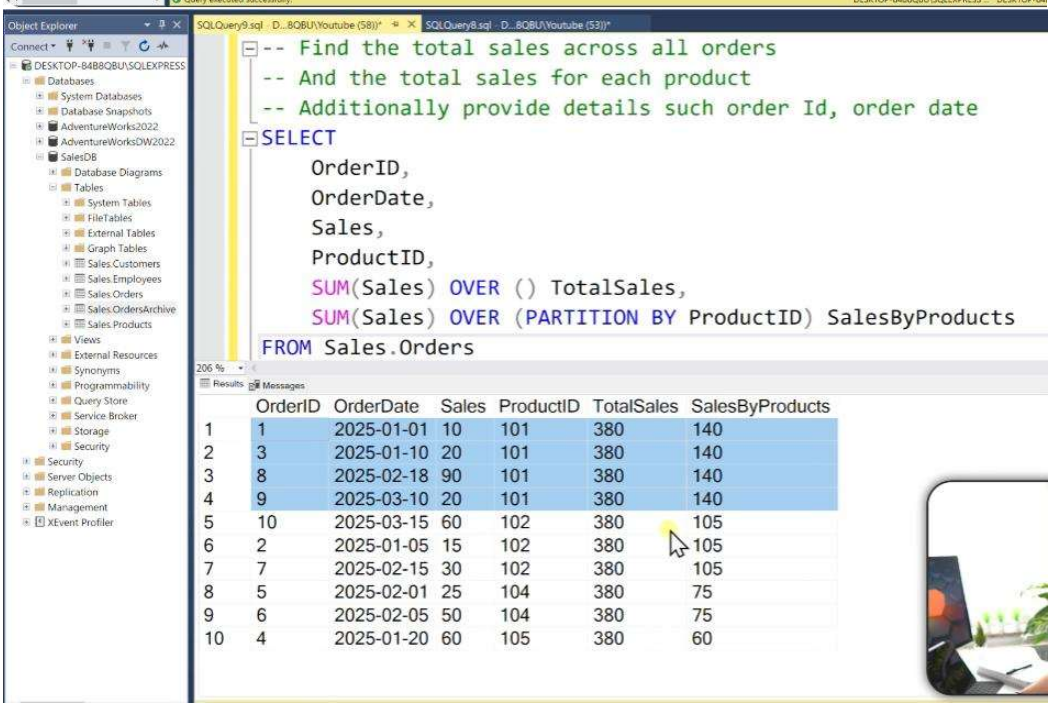
SUM() Accepts Only Numbers

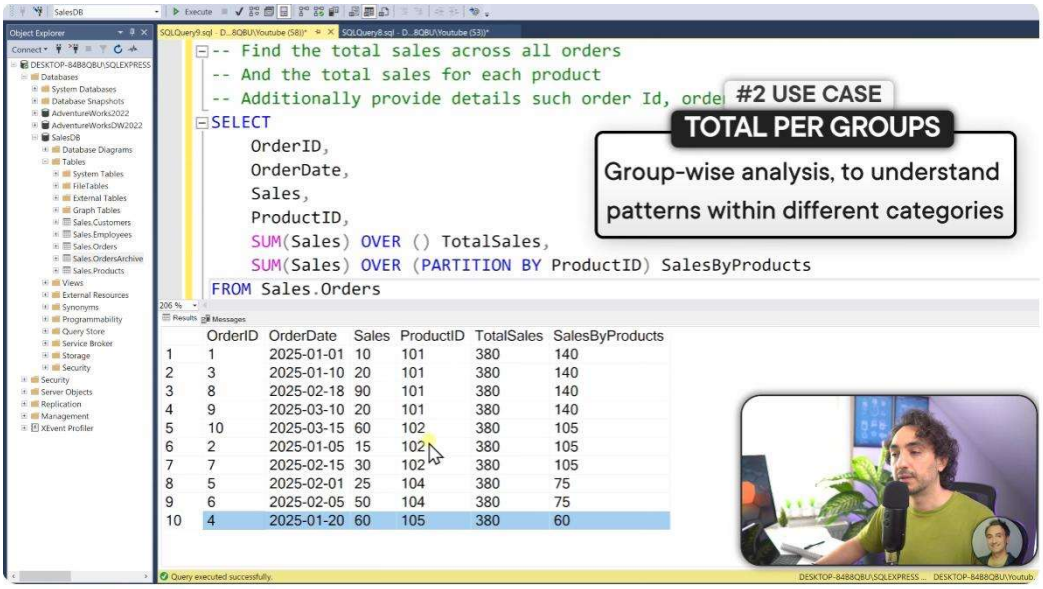
Product	Sales	SUM
Caps	20	35
Caps	10	35
Caps	5	35
Gloves	30	100
Gloves	70	100
Gloves	NULL	100

20 + 10 + 5 = 35

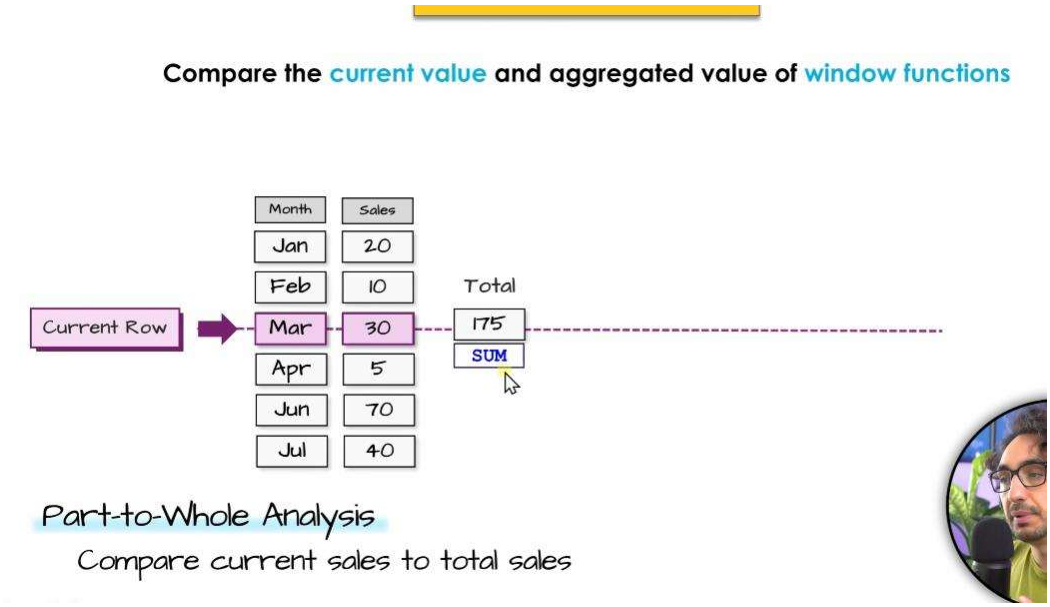
30 + 70 = 100


-- 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


-- 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



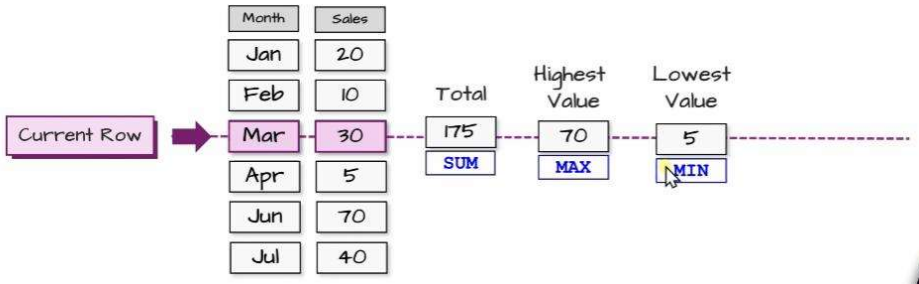
Helps to analyzes the product



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

Comparision Use Cases

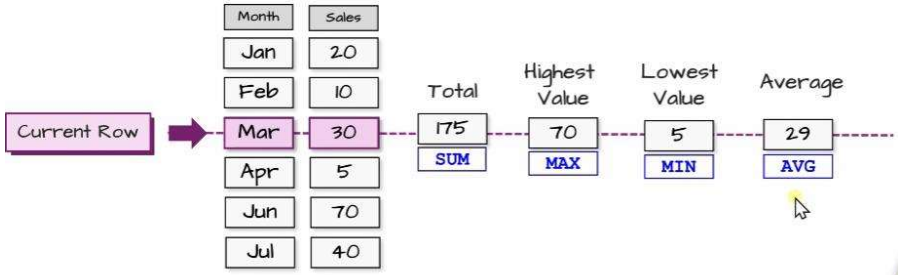
Compare the current value and aggregated value of window functions



Compare to Extremes Analysis

Compare current sales to the highest or lowest sales

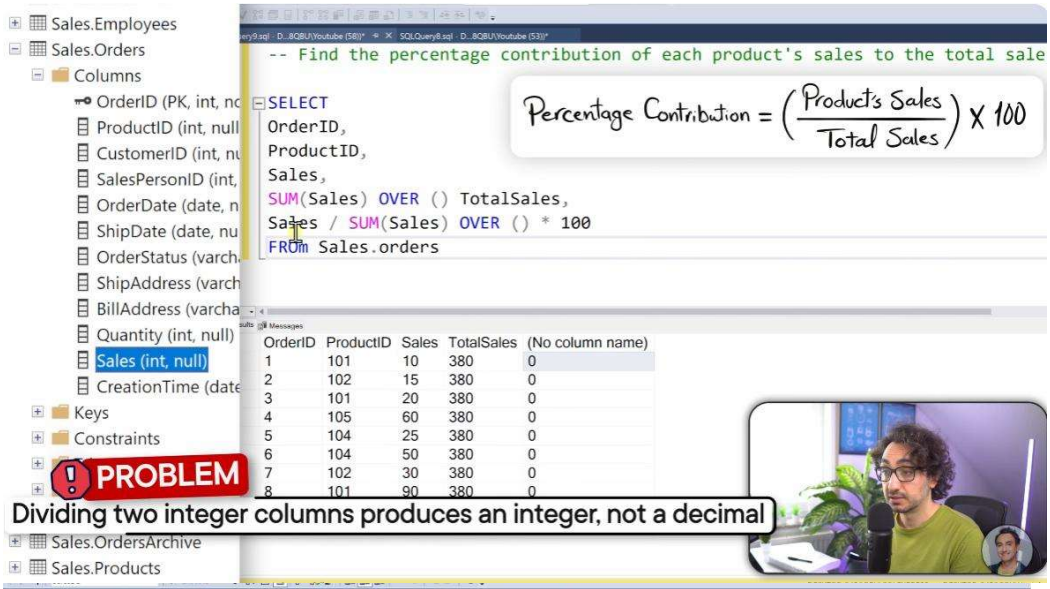
Compare the current value and aggregated value of window functions

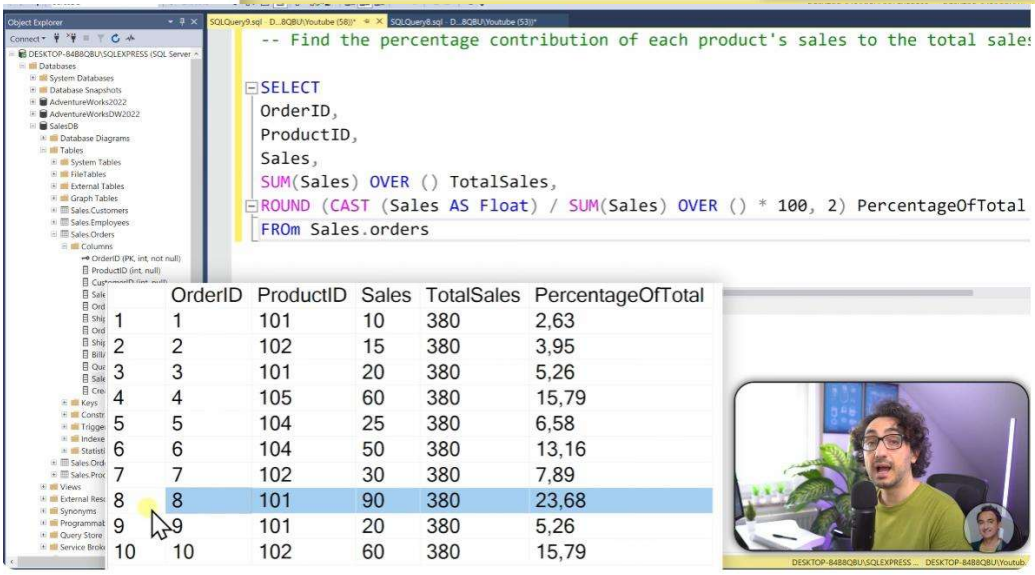


Compare to Average Analysis

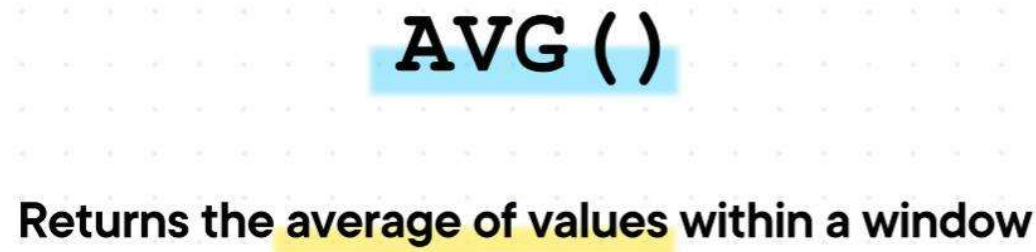
Help to evaluate whether a value is above or below the average

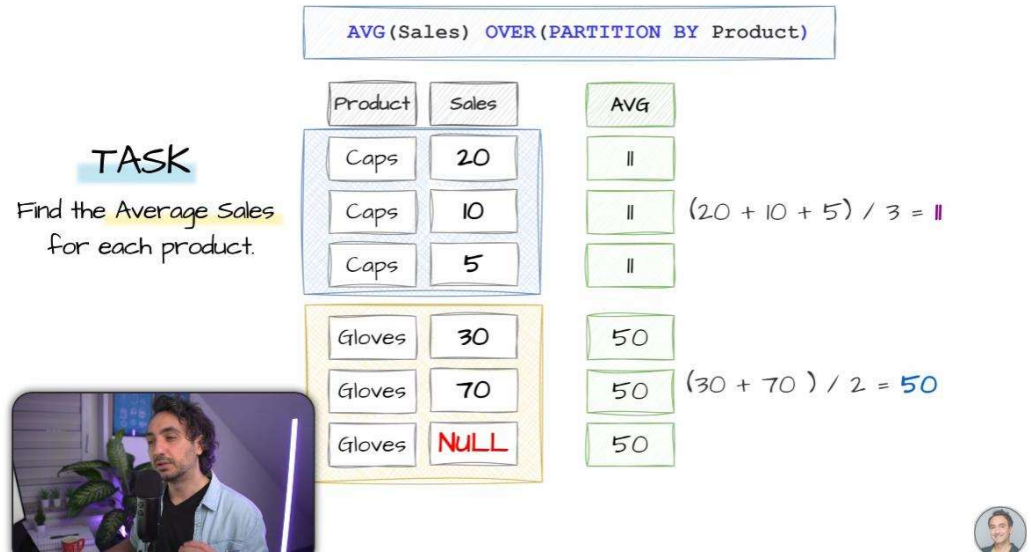
Its important to understand the performance of data





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





Before finding avg we have to handle null

TASK

Find the Average Sales for each product.

NULL Sales means ZERO.

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

Product	Sales	AVG
Caps	20	11
Caps	10	11
Caps	5	11
Gloves	30	50
Gloves	70	50
Gloves	NULL	50

(20 + 10 + 5) / 3 = 11

(30 + 70) / 2 = 50

TASK

Find the Average Sales for each product.

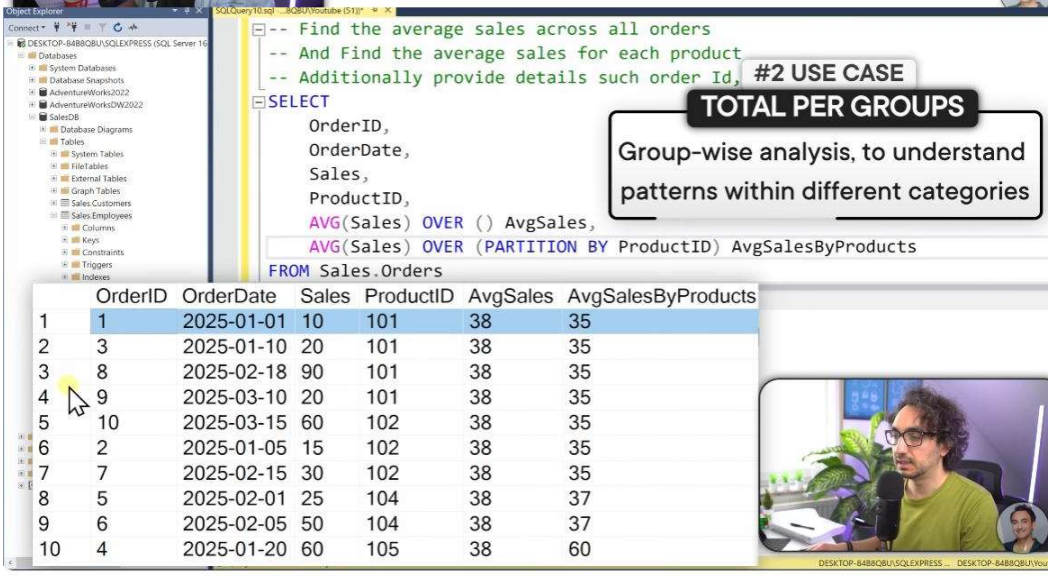
NULL Sales means ZERO.

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

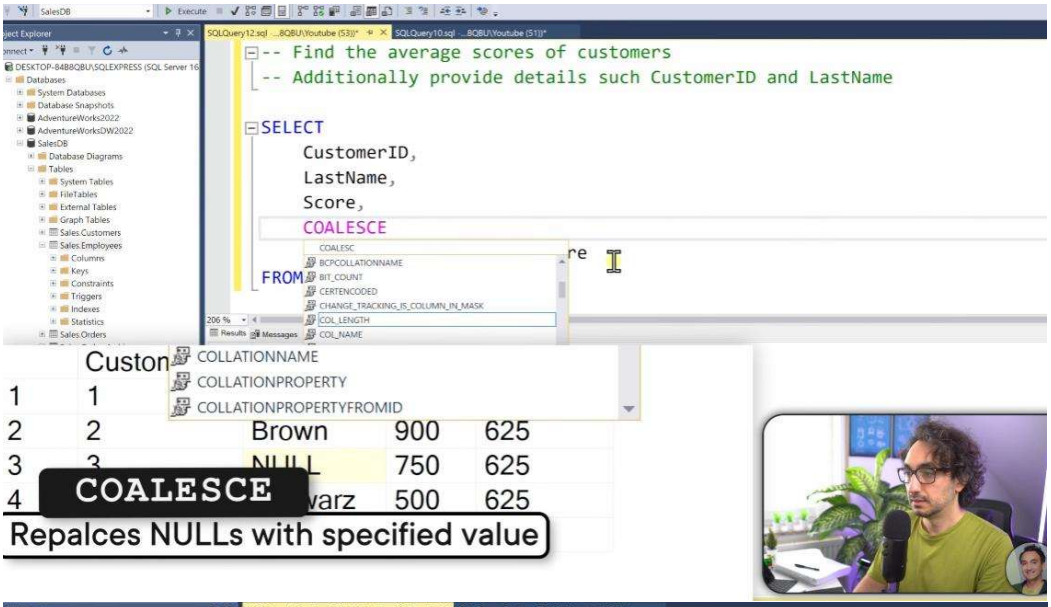
Product	Sales	AVG
Caps	20	11
Caps	10	11
Caps	5	11
Gloves	30	33
Gloves	70	33
Gloves	0	33

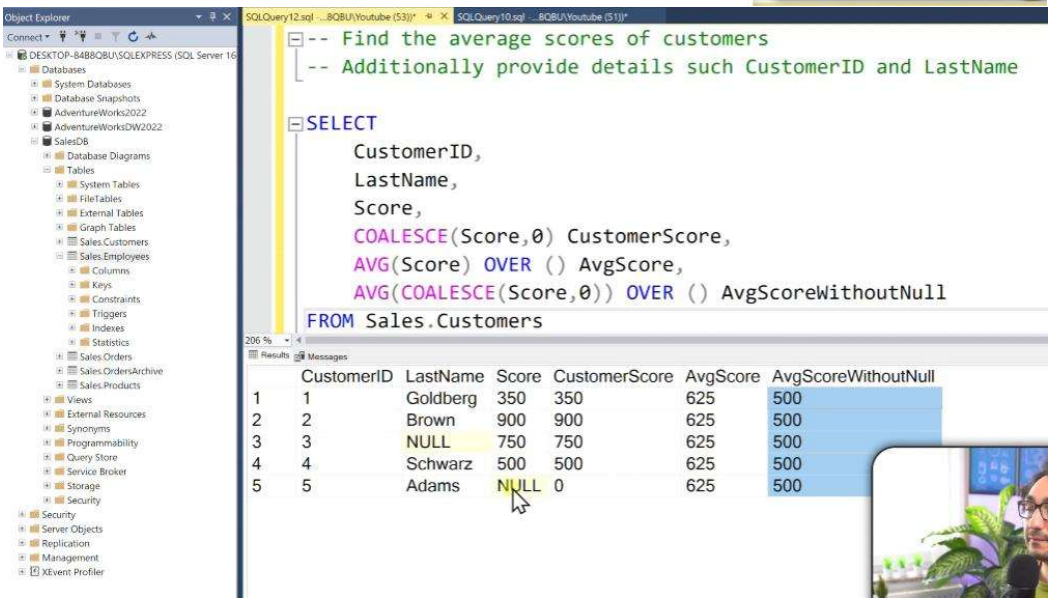
(20 + 10 + 5) / 3 = 11

(30 + 70 +0) / 3 = 33



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





We need to find the current sales here

```sql
--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
```

#3 USE CASE

COMPARE TO AVERAGE

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

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

MIN ()

Returns the lowest value within a window

MAX ()

Returns the highest value within a window

Find the highest sales for each product

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

Product	Sales	MIN
Caps	20	5
Caps	10	5
Caps	5	5
Gloves	30	30
Gloves	70	30
Gloves	NULL	30

Find the lowest sales for each product

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

Product	Sales	MAX
Caps	20	20
Caps	10	20
Caps	5	20
Gloves	30	70
Gloves	70	70
Gloves	NULL	70

Our busineess scenerio

Find the highest sales for each product

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

Product	Sales	MIN
Caps	20	5
Caps	10	5
Caps	5	5
Gloves	30	0
Gloves	70	0
Gloves	0	0

Find the lowest sales for each product

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

Product	Sales	MAX
Caps	20	20
Caps	10	20
Caps	5	20
Gloves	30	70
Gloves	70	70
Gloves	0	70

```sql
-- Find the highest and lowest sales of all orders
-- Find the highest and lowest sales for each product
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    Sales,
    MAX(Sales) OVER() HighestSales,
    MIN(Sales) OVER() LowestSales
FROM Sales.Orders
```

#1 USE CASE

OVERALL ANALYSIS

Quick summary or snapshot of the entire dataset

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

```sql
-- Find the highest and lowest sales of all orders
-- Find the highest and lowest sales for each product
-- Additionally provide details such order Id, order date
SELECT
    OrderID,
    OrderDate,
    ProductID,
    Sales,
    MAX(Sales) OVER() HighestSales,
    MIN(Sales) OVER() LowestSales,
    MAX(Sales) OVER(PARTITION BY ProductID) HighestSalesByProduct,
    MIN(Sales) OVER(PARTITION BY ProductID) LowestSalesByProduct
FROM Sales.Orders
```

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