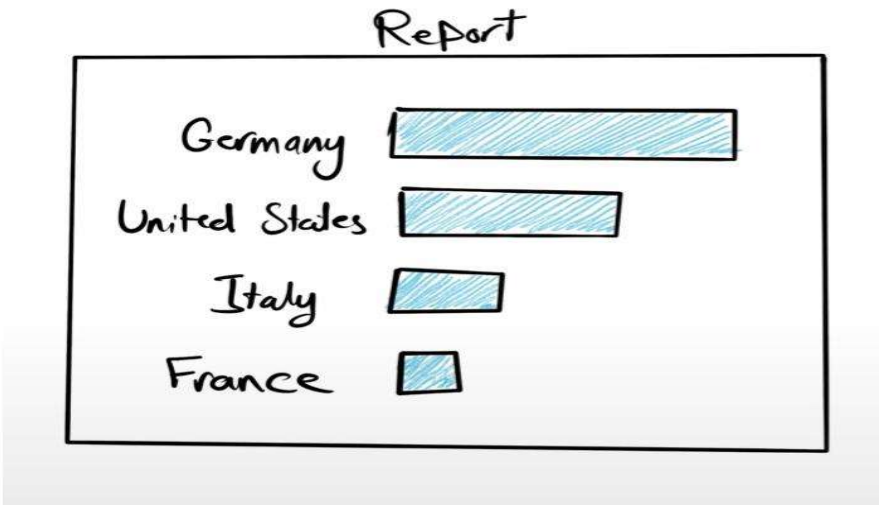
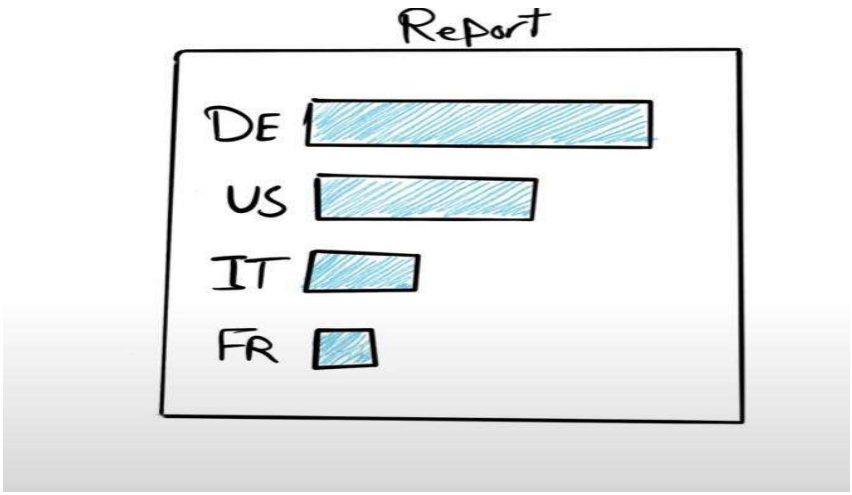
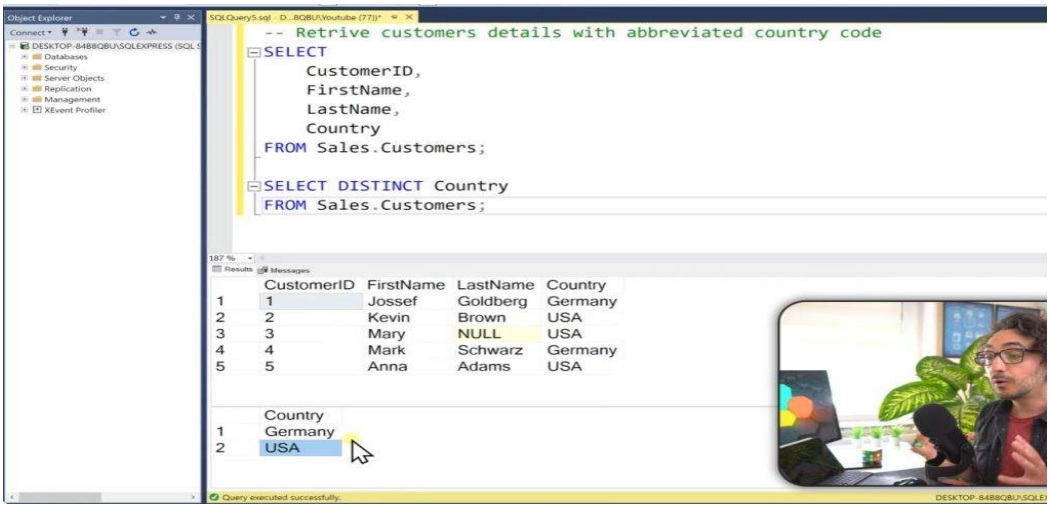


While generating reports we do not have enough space to store full name of values so we need abbrebiations short form of values
So we used case statement to match full value with abbebiated values of abbribiated value with full vale





Another example -

Column Name
to be evaluated (Only One)

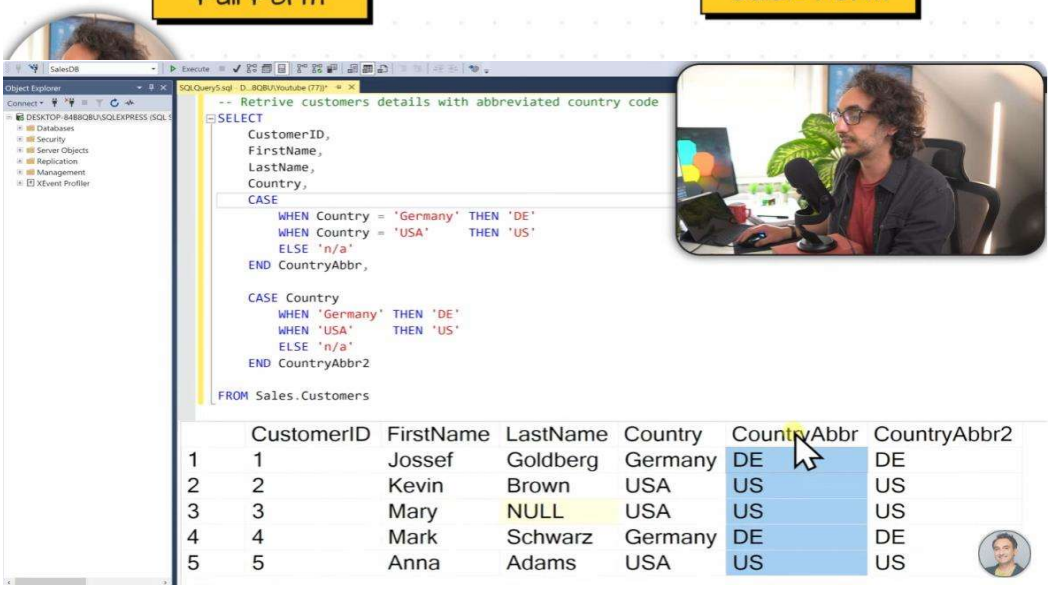
Column Value
To be compared

```
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

```
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

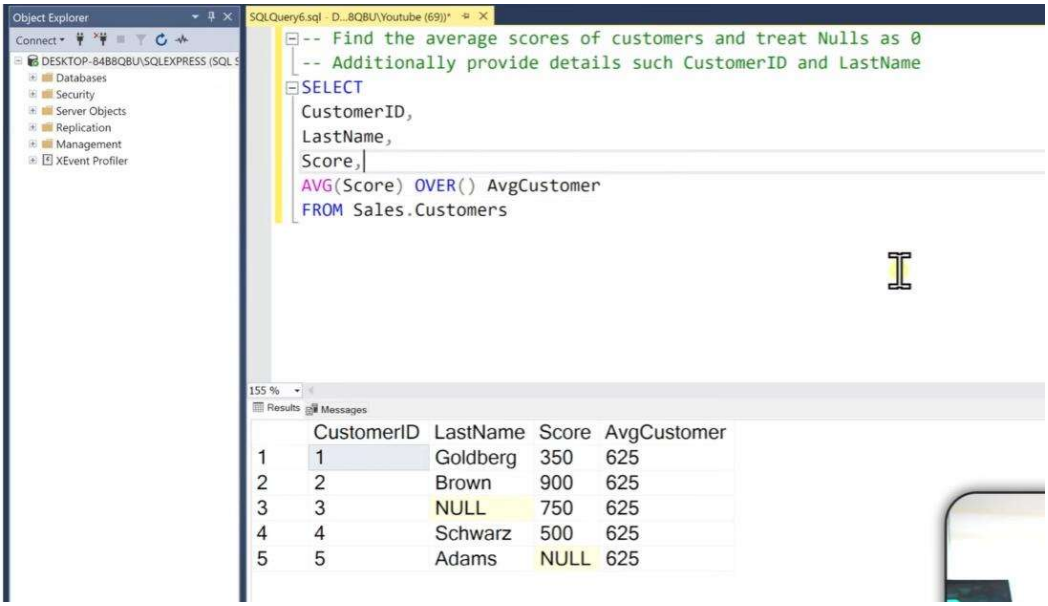


	CustomerID	FirstName	LastName	Country	CountryAbbr	CountryAbbr2
1	1	Jossef	Goldberg	Germany	DE	DE
2	2	Kevin	Brown	USA	US	US
3	3	Mary	NULL	USA	US	US
4	4	Mark	Schwarz	Germany	DE	DE
5	5	Anna	Adams	USA	US	US

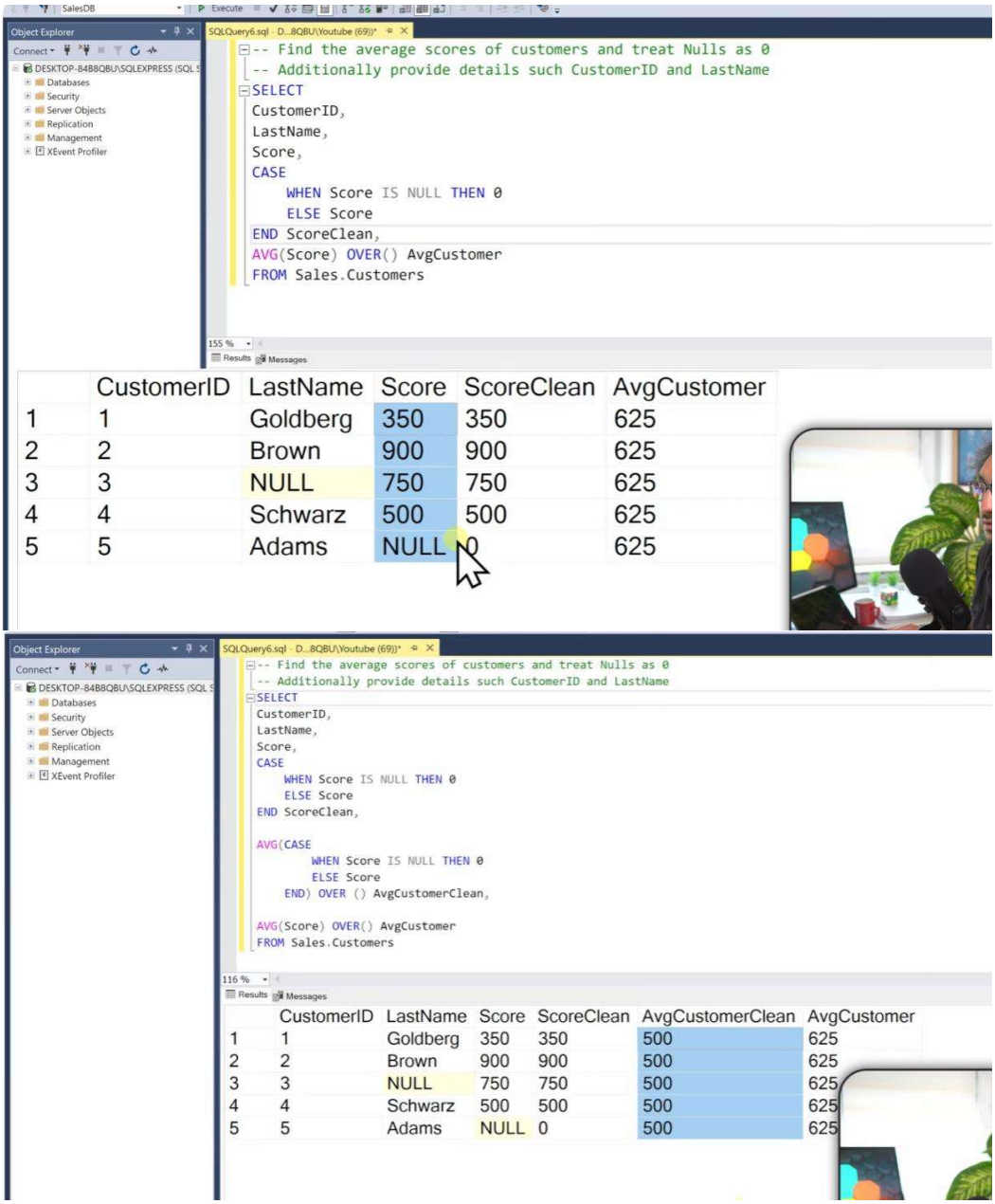
HANDLING NULLS

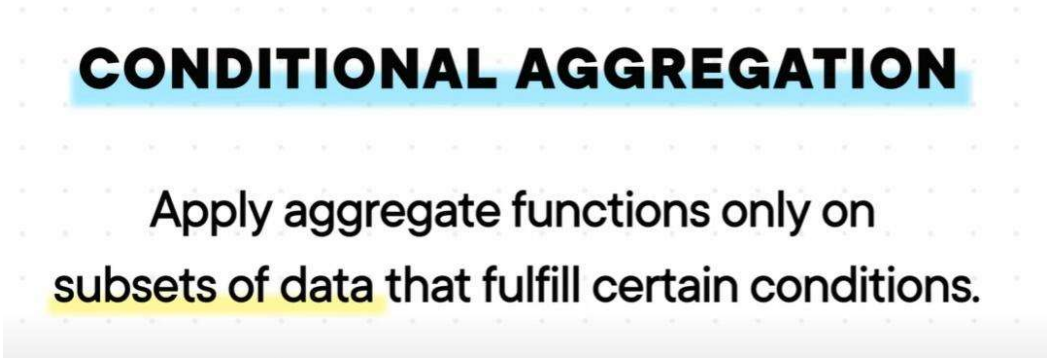
Replace NULLs with a specific value.

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



Null - missing info in above example
Our businees need null - 0 so that it consider 0 not missing info so that the avg will also change





Analyze target on specific subset of data

```sql
-- 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
```

FLAG

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

```sql
-- 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
```

STEP-2

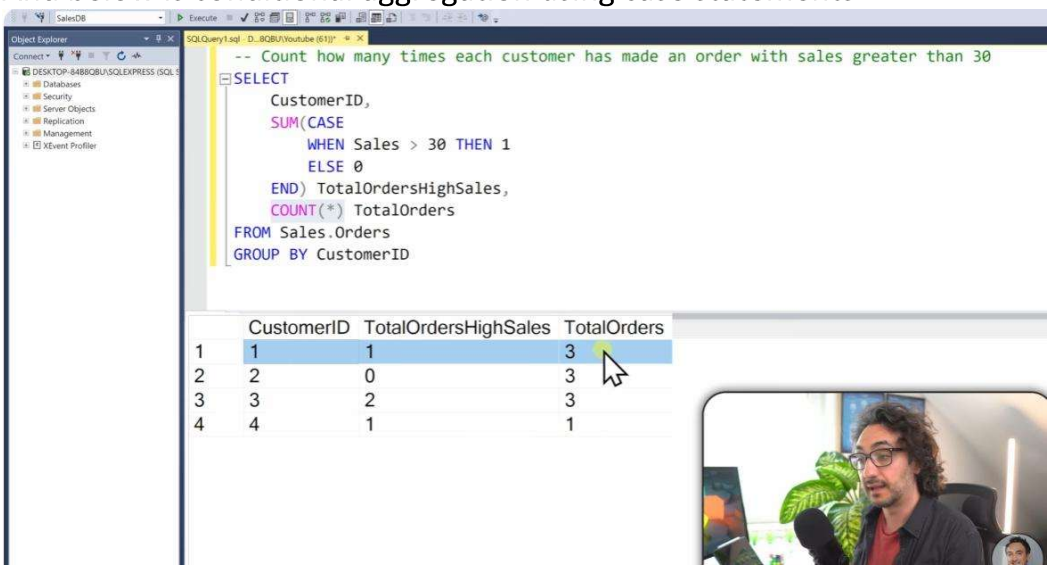
Summarize the binary flag

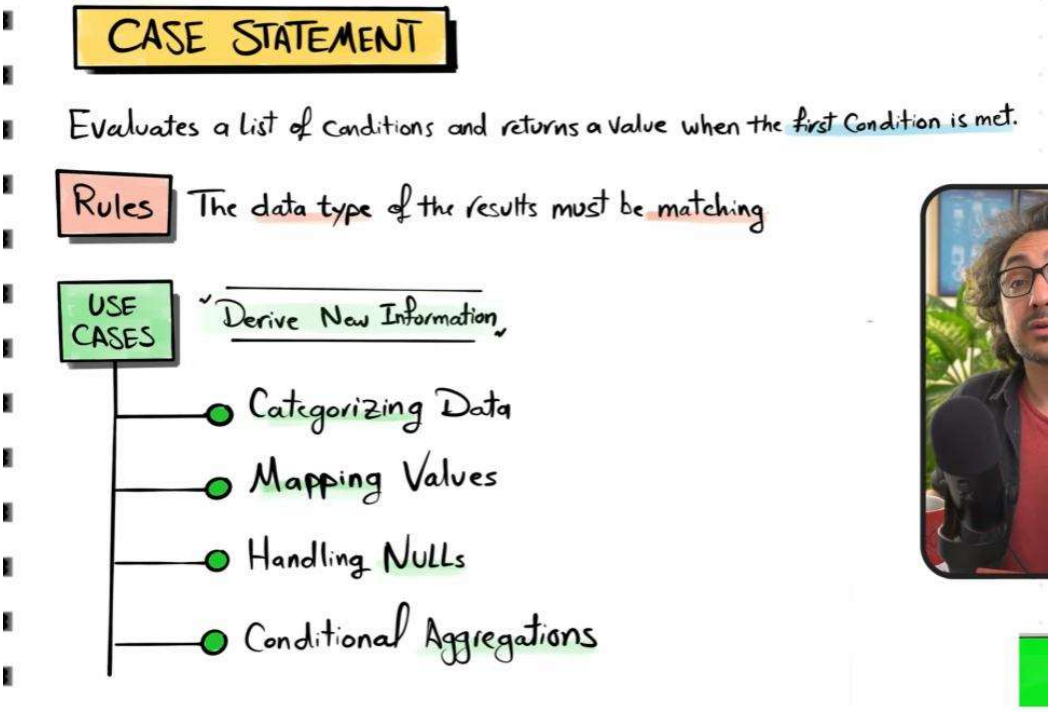
	OrderID	CustomerID	Sales	SalesFlag
			20	0
			60	1
3	7	1	30	0
4	1	2	10	0
5	5	2	25	0
6	9	2	20	0
7	10	3	60	1
8	6	3	50	1
9	2	3	15	0
10	8	4	90	1

```sql
-- 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
```

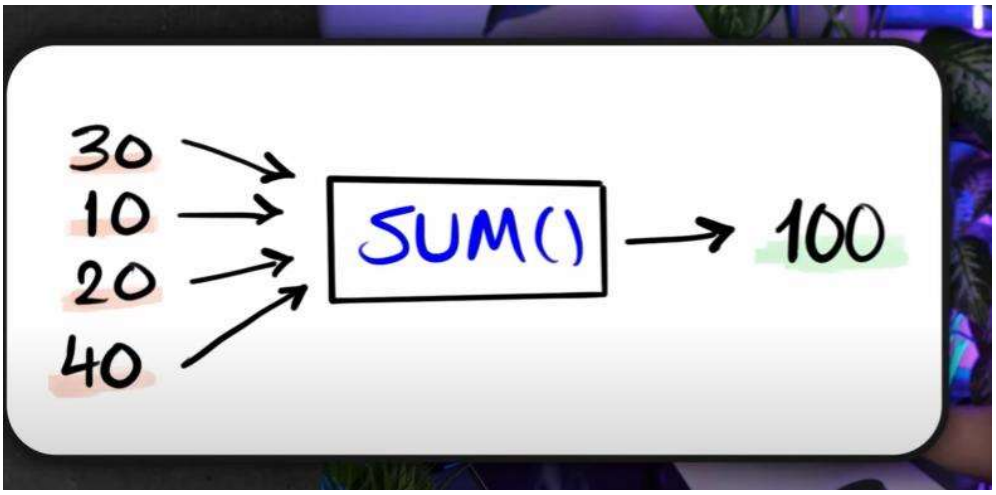
	CustomerID	TotalOrders
1	1	1
2	2	0
3	3	2
4	4	1

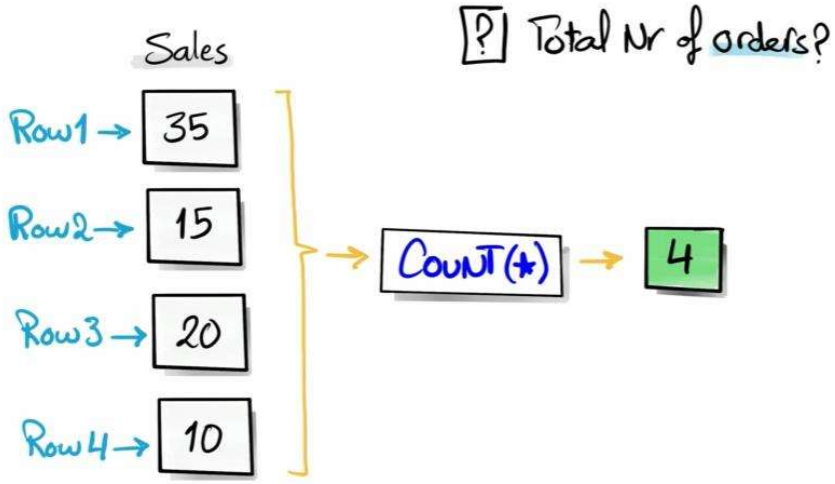
Above is conditional Aggregation
And below is conditional aggregation using case statements

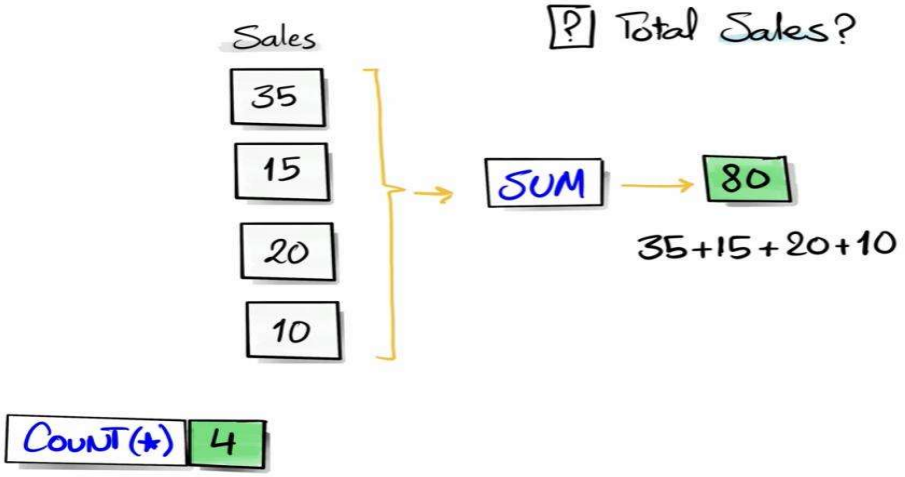


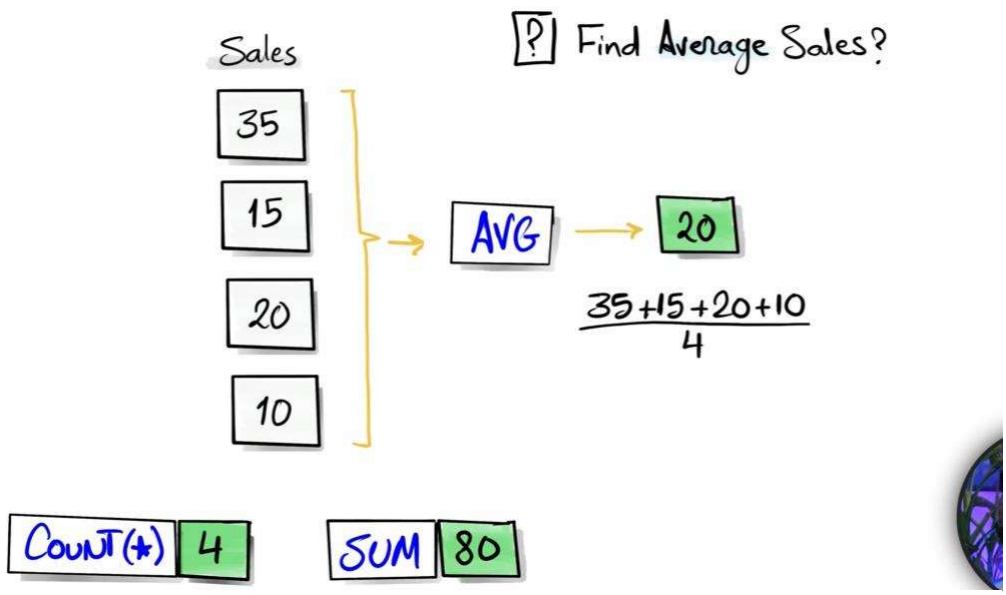


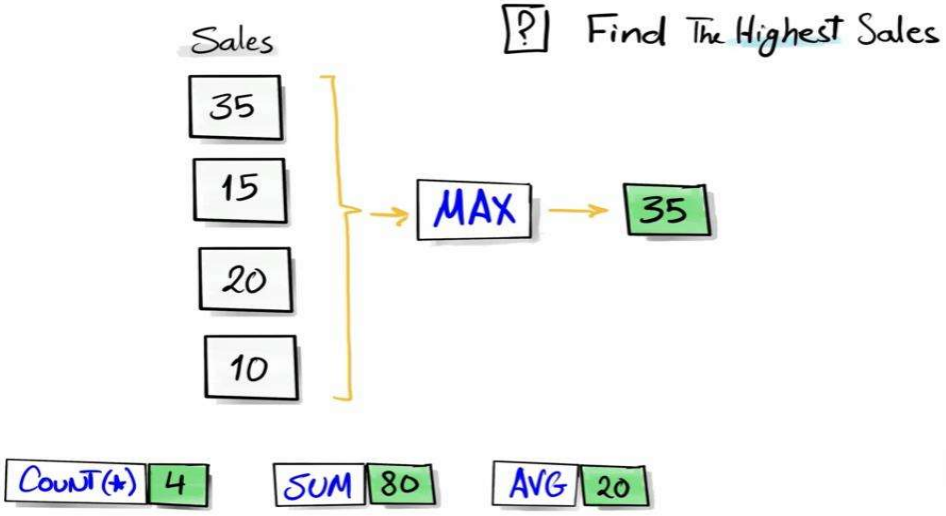
Aggregate Functions: Takes multiple rows value as an input and output gona one single value

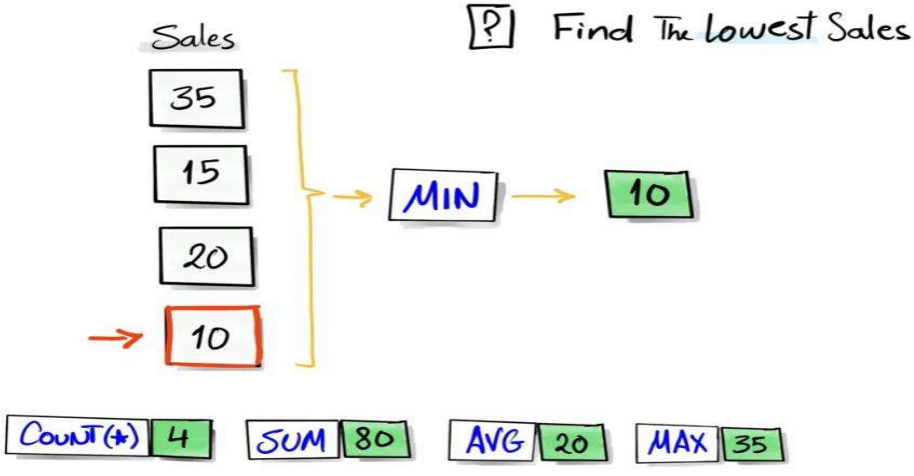


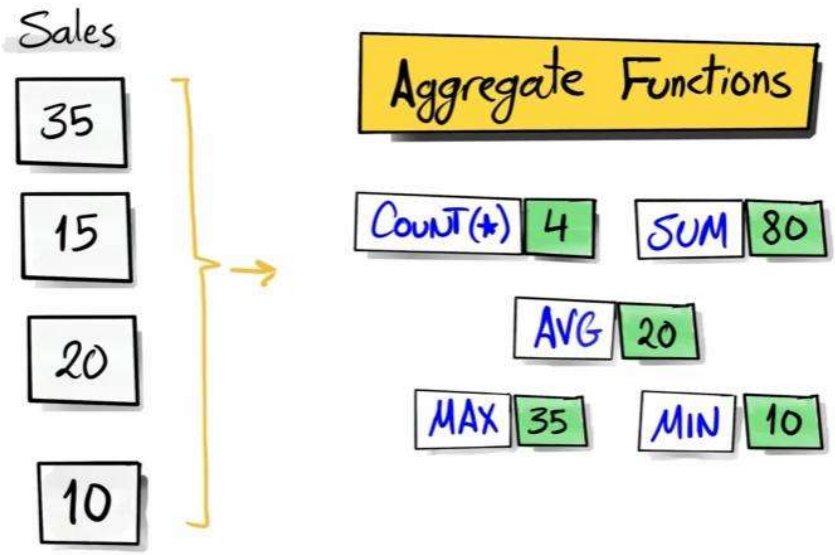



Sales
35
15
20
10
? Total Sales?
SUM
80
35+15+20+10
COUNT(*)
4


Sales
35
15
20
10
AVG
20
35+15+20+10
4
COUNT(*) 4
SUM 80


Find The Highest Sales
Sales
35
15
20
10
MAX
35
COUNT(*) 4
SUM 80
AVG 20


Find The Lowest Sales
Sales
35
15
20
10
MIN
10
COUNT(*) 4
SUM 80
AVG 20
MAX 35




```sql
-- Find the total number of orders
SELECT
COUNT(*) AS total_nr_orders
FROM orders
```

Results Messages

	total_nr_orders
1	4

```sql
-- Find the total sales of all orders
SELECT
COUNT(*) AS total_nr_orders,
SUM(sales) AS total_sales
FROM orders
```

Results Messages

	total_nr_orders	total_sales
1	4	80

```sql
-- 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

```sql
-- 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

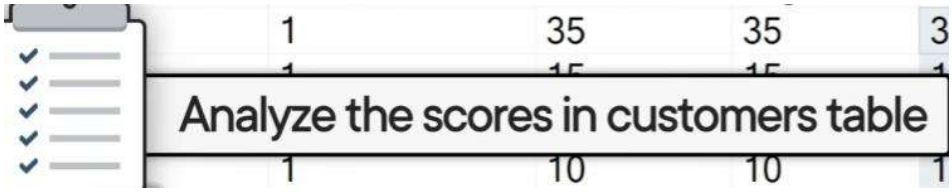
	total_nr_orders	total_sales	avg_sales	highest_sales	lowest_sales
1	4	80	20	35	10

Each customer data using group by

```sql
-- 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
```

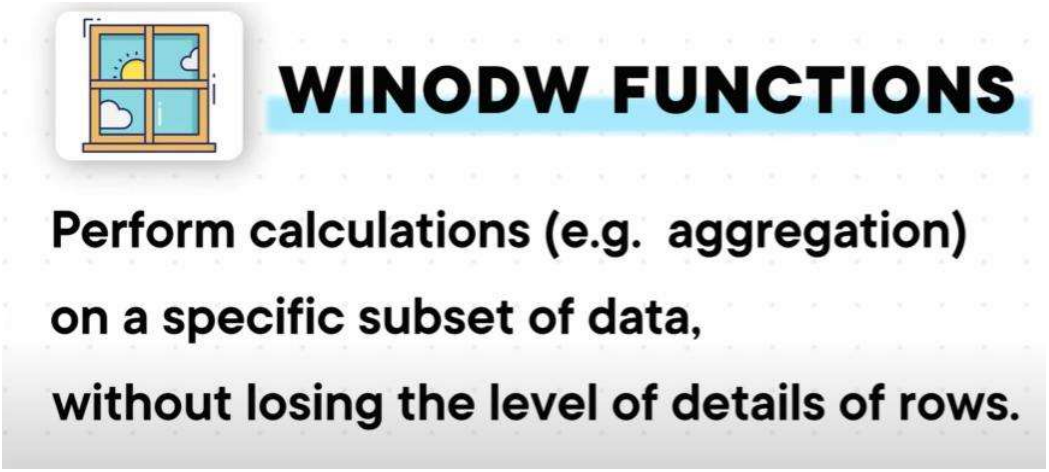
Results Messages

	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

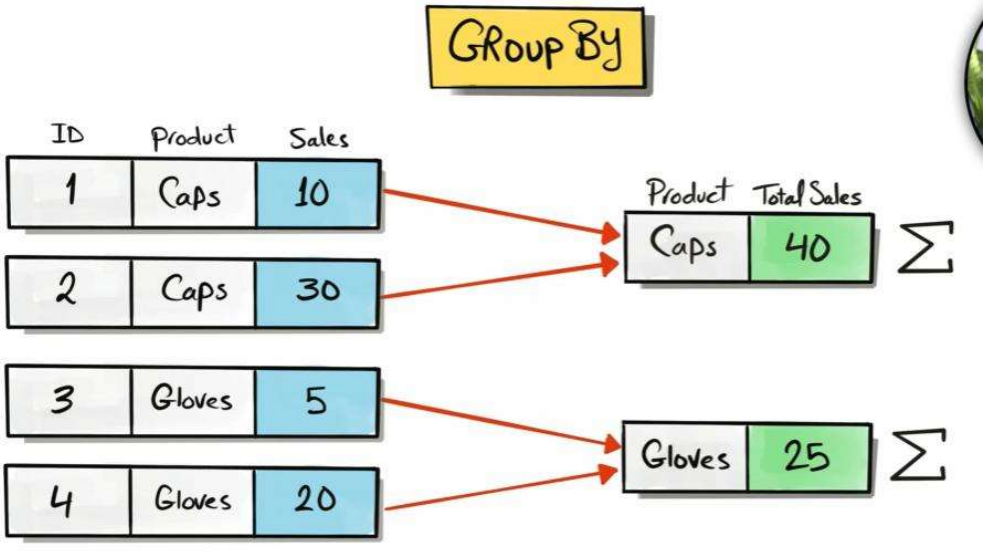
Task 

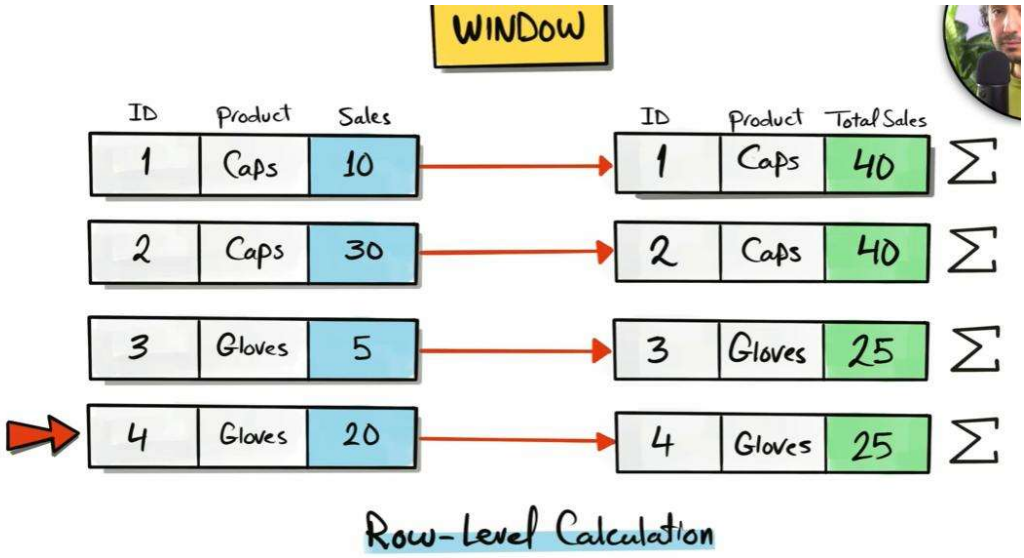
and group by data by country

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

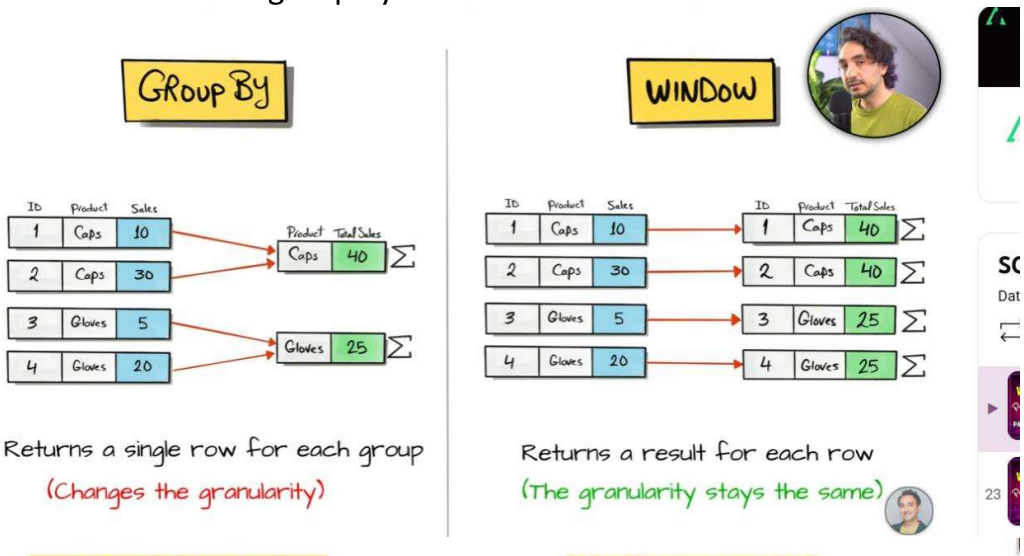


Order ID controlling level of details in input and product is controlling level of details in output





Difference between group by and window functions:



GROUP BY

Simple Data Analysis

WINDOW

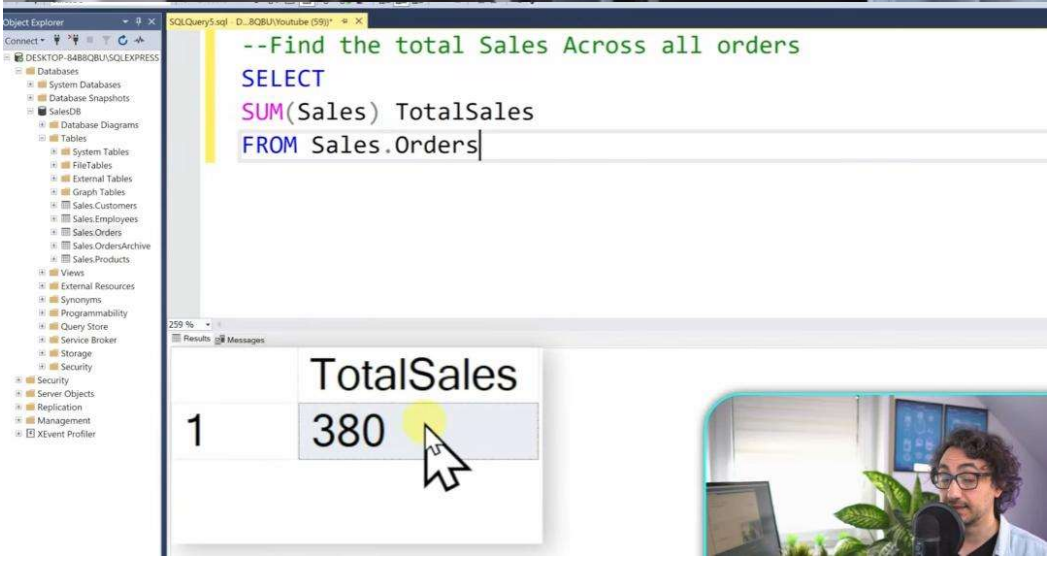
Advanced Data Analysis

GROUP BY Functions

Aggregate Functions	COUNT (*expr*)
	SUM (*expr*)
	AVG (*expr*)
	MIN (*expr*)
	MAX (*expr*)

WINDOW Functions

Aggregate Functions	COUNT (*expr*)
	SUM (*expr*)
	AVG (*expr*)
	MIN (*expr*)
	MAX (*expr*)
Rank Functions	ROW_NUMBER ()
	RANK ()
	DENSE_RANK ()
	CUME_DIST ()
	PERCENT_RANK ()
	NTILE (*n*)
Value (Analytics) Functions	LEAD (*expr,offset,default*)
	LAG (*expr,offset,default*)
	FIRST_VALUE (*expr*)
	FIRST_VALUE (*expr*)



```sql
--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

```sql
/*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
```

GROUP BY RULE

All columns in SELECT must be included in GROUP BY

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

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

```sql
/*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

```sql
/*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
```

GROUP BY LIMITS

Can't do aggregations and provide details at same time

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

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

Result Granularity

Window functions returns a result for each row

	(No column name)
1	380
2	380
3	380
4	380
5	380
6	380
7	380
8	380
9	380
10	380

```sql
/*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	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

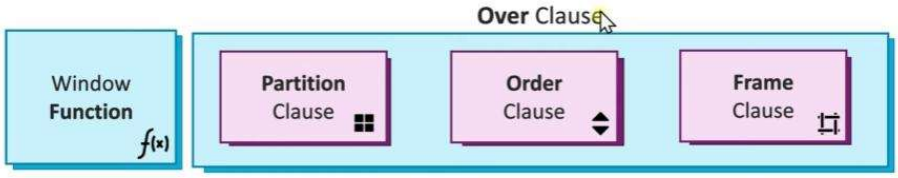
GROUP BY

Simple Data Analysis
(Aggregations)

WINDOW

Advanced Data Analysis
(Aggregations + Details)

</> Window Syntax




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

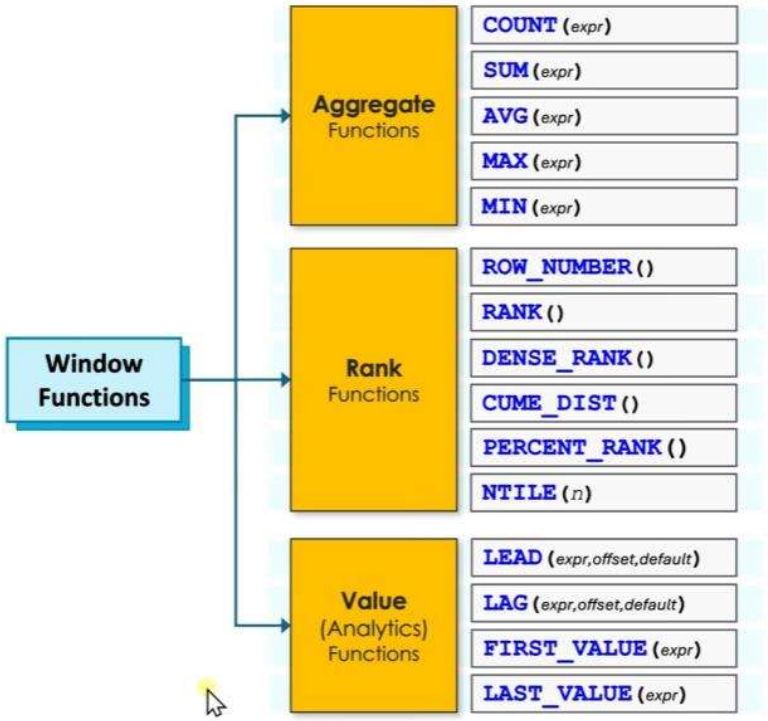
Calculation used on the Window

Window Function

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

WINODW FUNCTIONS

Perform calculations within a window

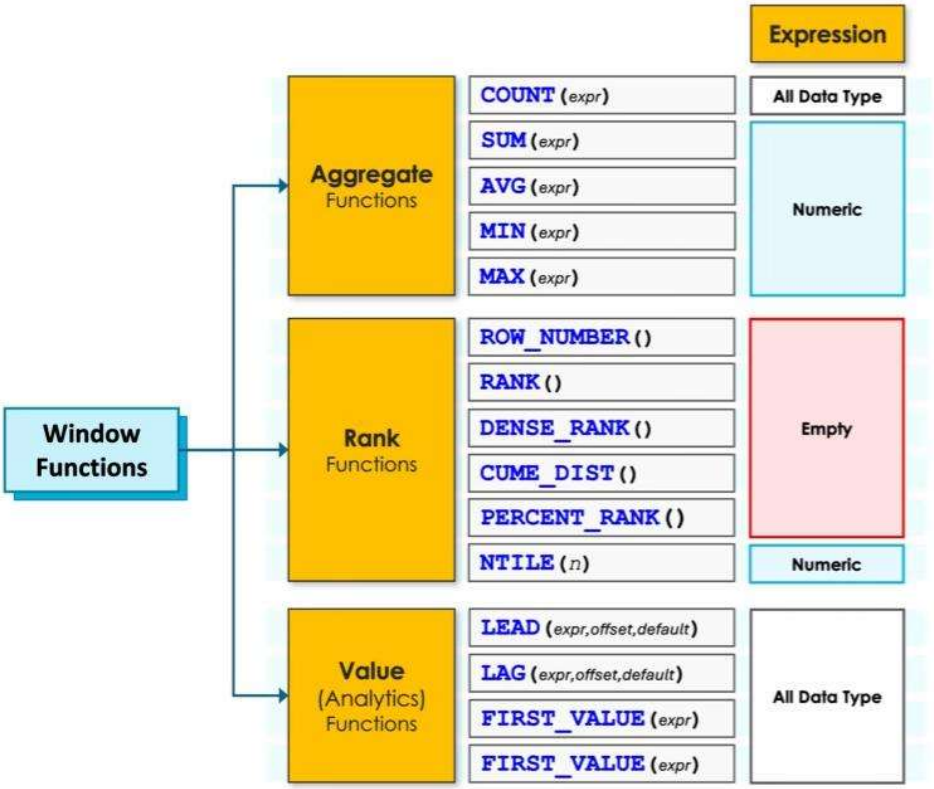


FUNCTION EXPRESSION

Arguments you pass to a function

f(x) Window Expression

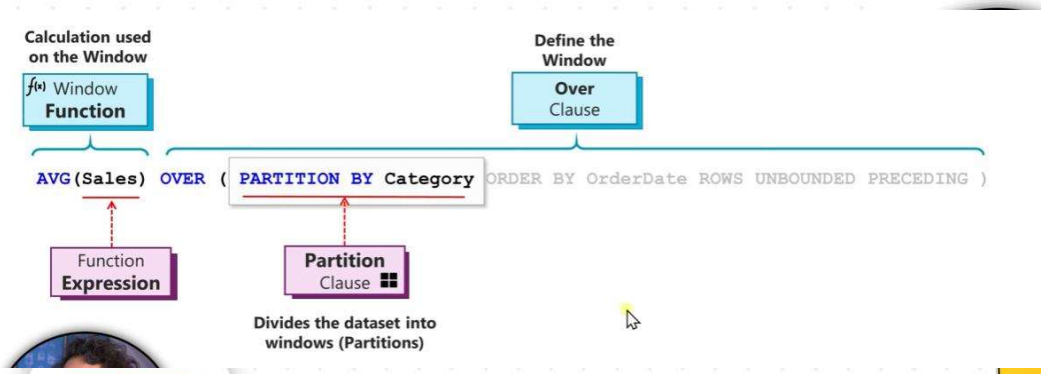
Empty	`RANK() OVER (ORDER BY OrderDate)`
Column	`AVG(Sales) OVER (ORDER BY OrderDate)`
Number	`NTEIL(2) OVER (ORDER BY OrderDate)`
Multiple Arguments	`LEAD(Sales,2,10) OVER (ORDER BY OrderDate)`
Conditional Logic	`SUM(CASE WHEN Sales > 100 THEN 1 ELSE 0 END) OVER (ORDER BY OrderDate)`



OVER CLAUSE

Tells SQL that the function used is a window function

It defines a window or subset of data

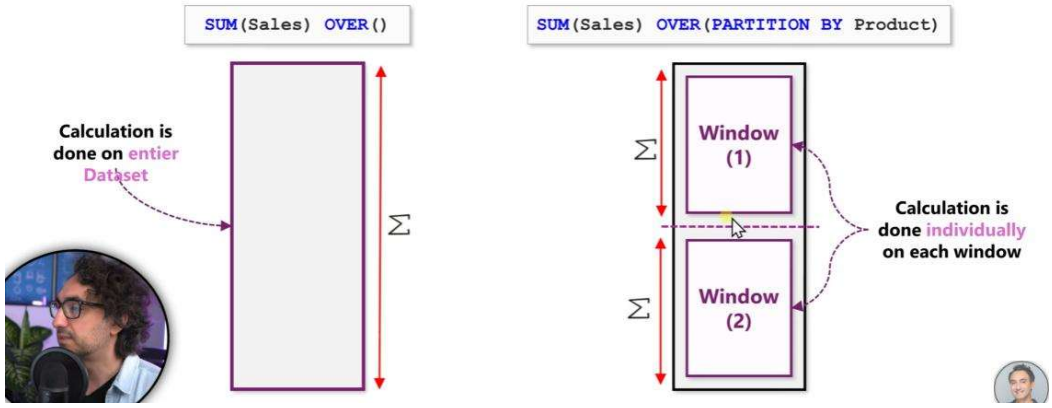


PARTITION BY

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()
```

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

Dataset is divided into 2 window

Without Partition By

Total sales across all rows (Entire Result Set)

```
SUM(Sales) OVER ()
```

Partition By **Single Column**

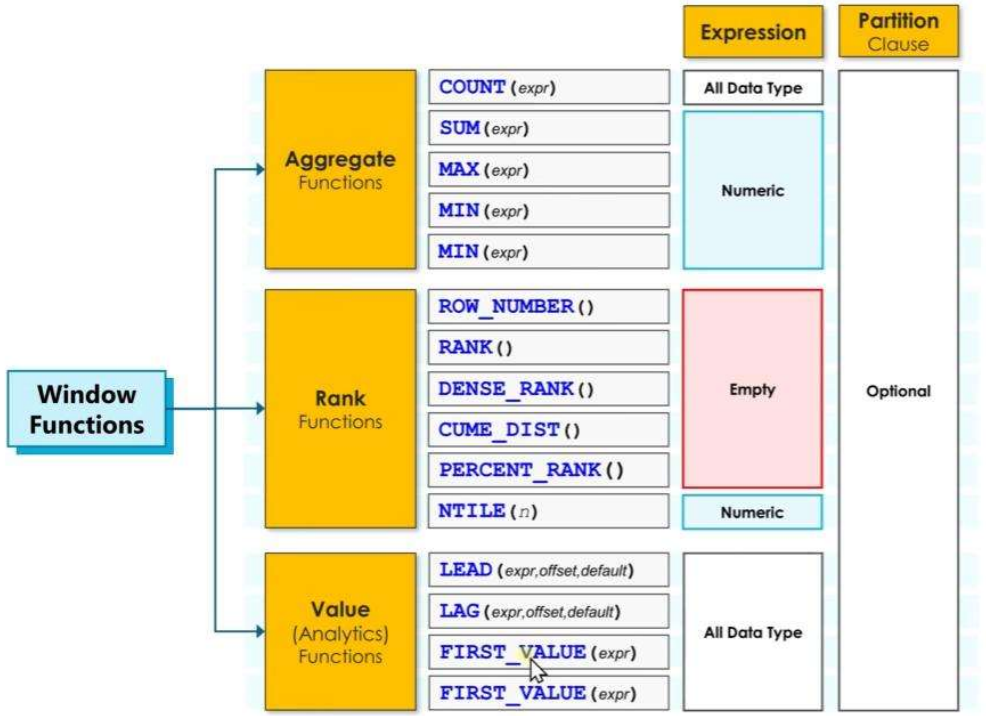
Total sales for each Product

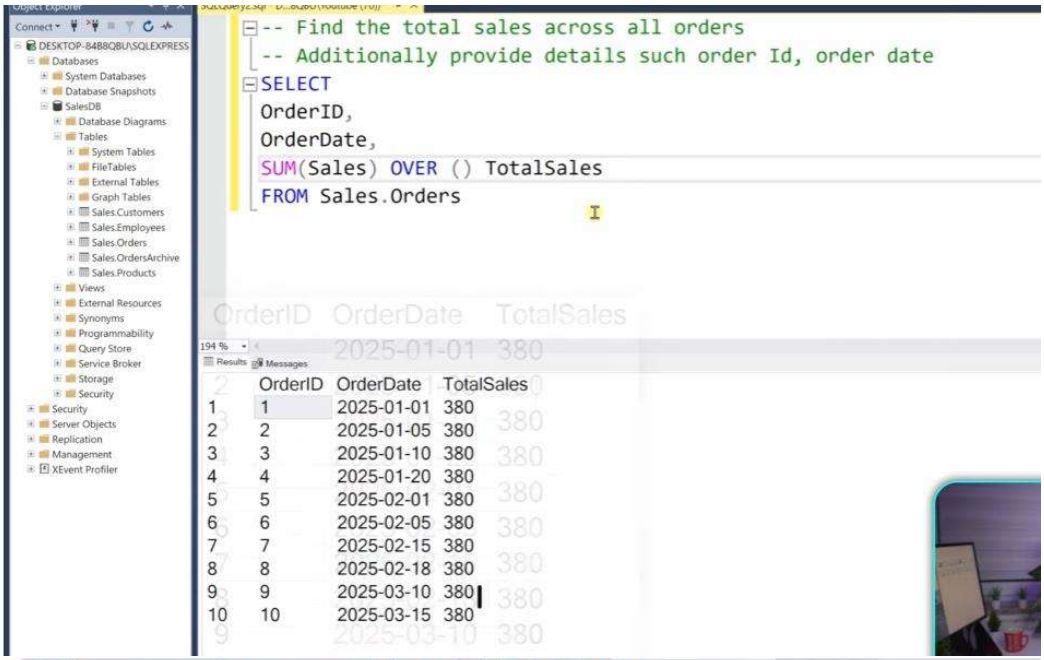
```
SUM(Sales) OVER (PARTITION BY Product)
```

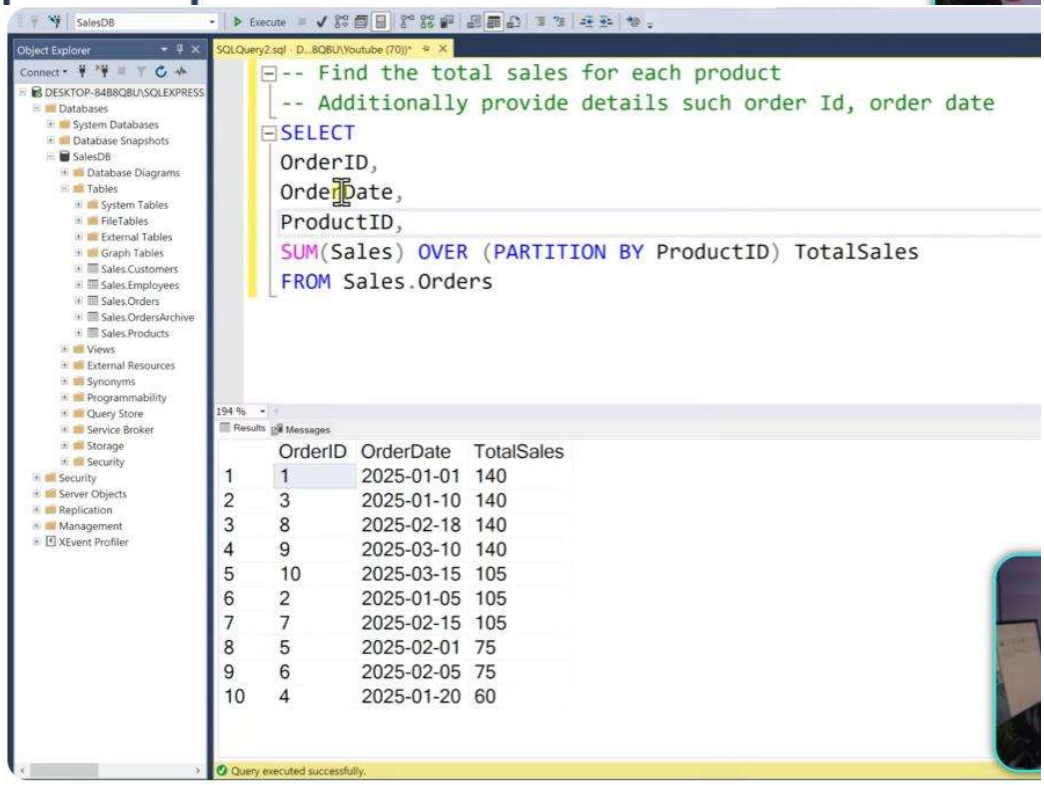
Partition By **Combined-Columns**

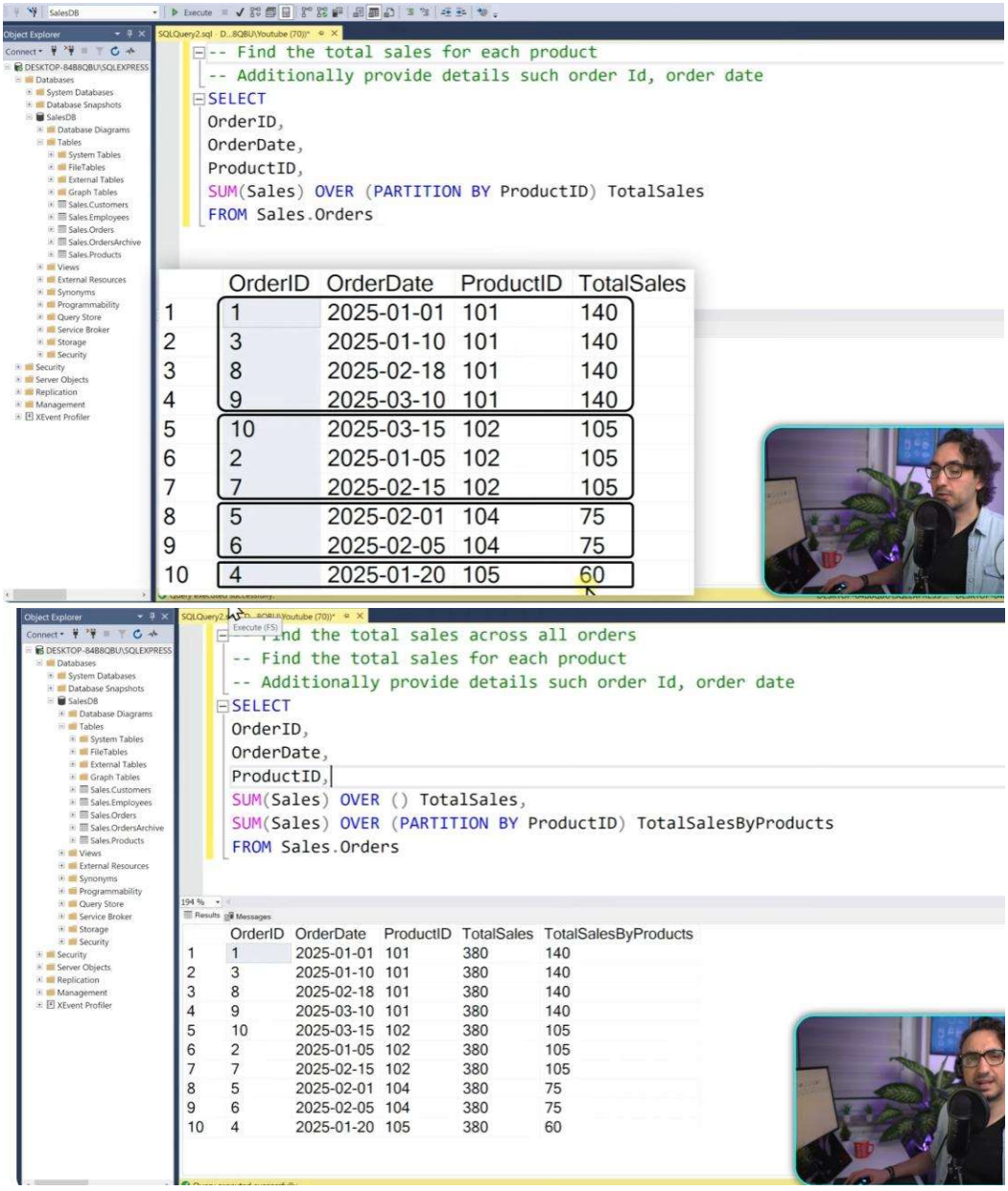
Total sales for each combination of Product and Order Status

```
SUM(Sales) OVER (PARTITION BY Product, OrderStatus)
```

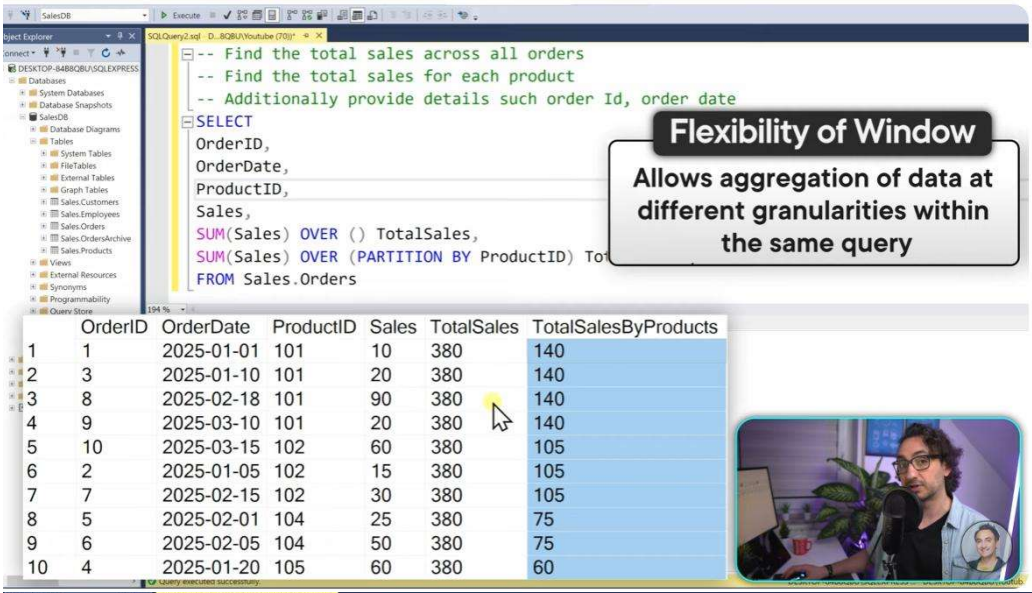



-- Find the total sales across all orders
-- Additionally provide details such order Id, order date
SELECT
OrderID,
OrderDate,
SUM(Sales) OVER () TotalSales
FROM Sales.Orders
OrderID OrderDate TotalSales
1 1 2025-01-01 380
2 2 2025-01-05 380
3 3 2025-01-10 380
4 4 2025-01-20 380
5 5 2025-02-01 380
6 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

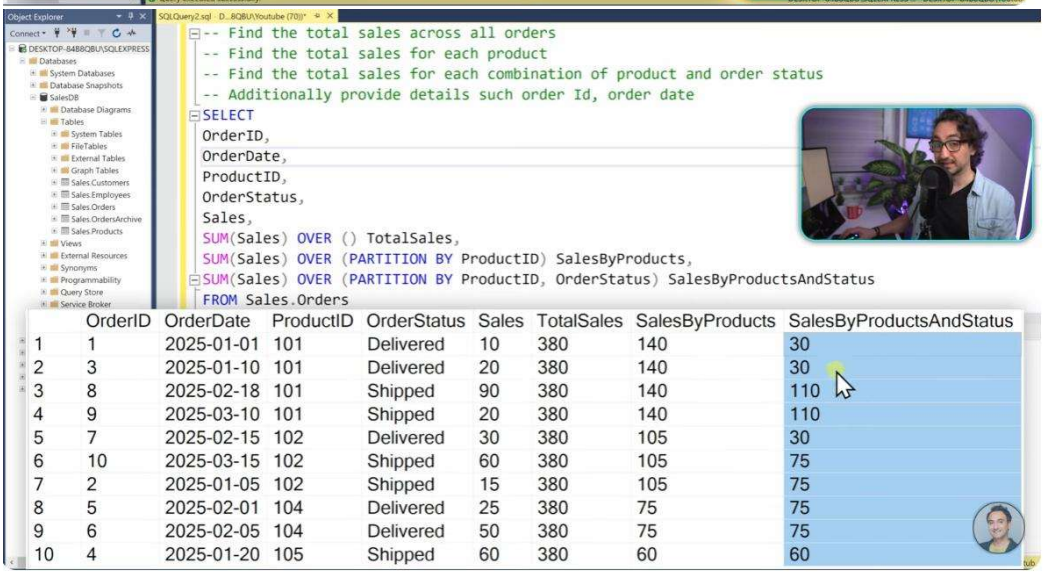

-- Find the total sales for each product
-- Additionally provide details such order Id, order date
SELECT
OrderID,
OrderDate,
ProductID,
SUM(Sales) OVER (PARTITION BY ProductID) TotalSales
FROM Sales.Orders
OrderID OrderDate TotalSales
1 1 2025-01-01 140
2 3 2025-01-10 140
3 8 2025-02-18 140
4 9 2025-03-10 140
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.


-- 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

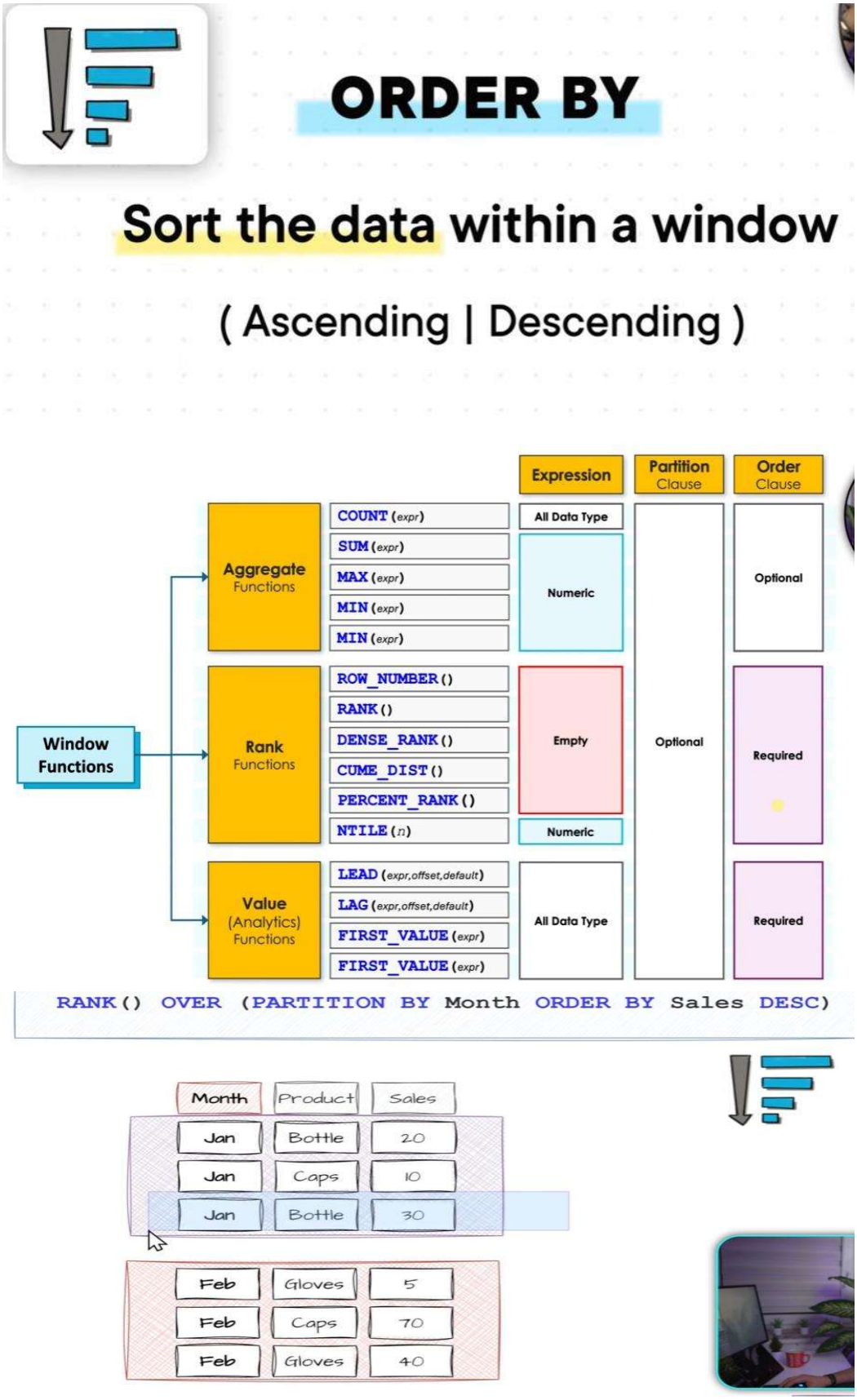
-- Find the total sales across all orders
-- Find the total sales for each product
-- Additionally provide details such order Id, order date
SELECT
OrderID,
OrderDate,
ProductID,
SUM(Sales) OVER () TotalSales,
SUM(Sales) OVER (PARTITION BY ProductID) TotalSalesByProducts
FROM Sales.Orders
OrderID OrderDate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



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



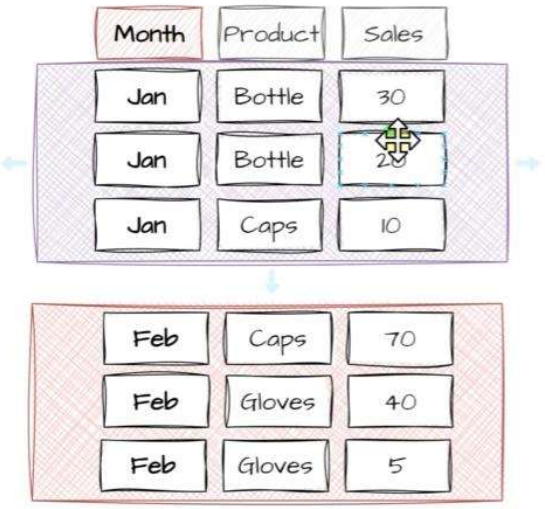
	OrderID	OrderDate	ProductID	OrderStatus	Sales	TotalSales	SalesByProducts	SalesByProductsAndStatus
1	1	2025-01-01	101	Delivered	10	380	140	30
2	3	2025-01-10	101	Delivered	20	380	140	30
3	8	2025-02-18	101	Shipped	90	380	140	110
4	9	2025-03-10	101	Shipped	20	380	140	110
5	7	2025-02-15	102	Delivered	30	380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



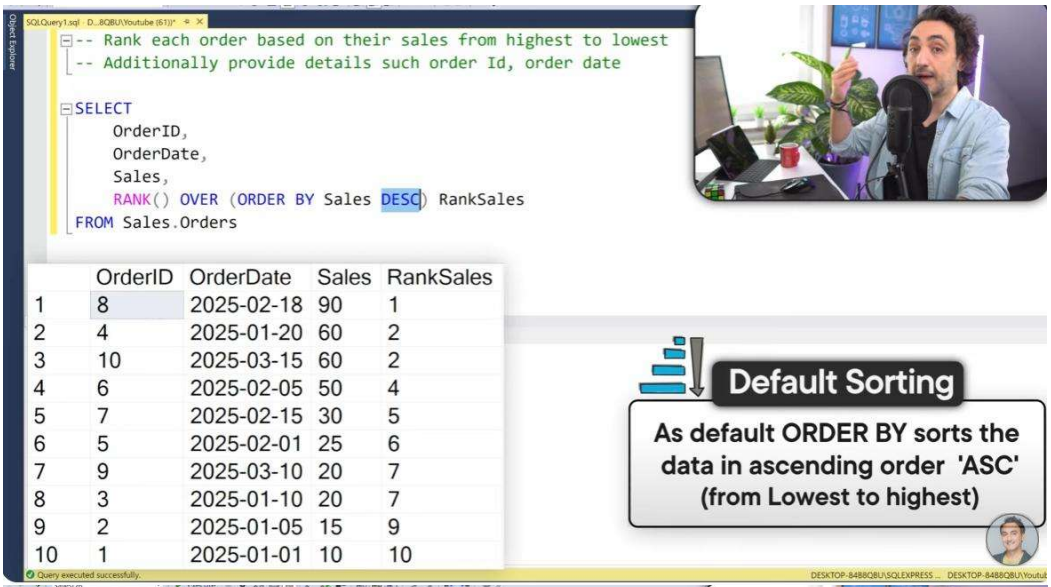
QL Window Functions Basics | Partition By, Order By, Frame | #SQL Co

Spillted by month and sorted by sales

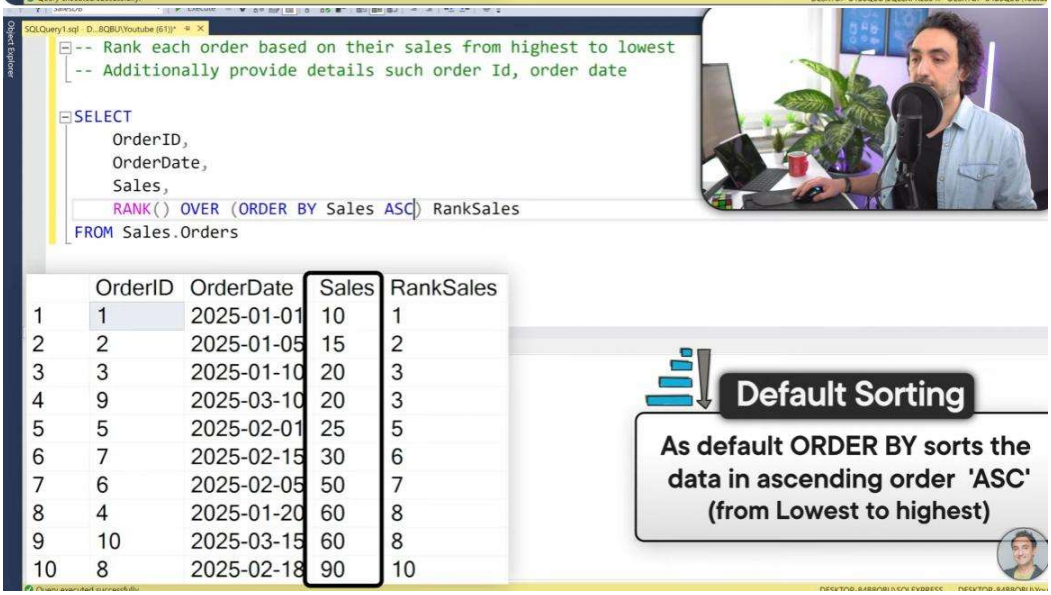
```
RANK() OVER (PARTITION BY Month ORDER BY Sales DESC)
```



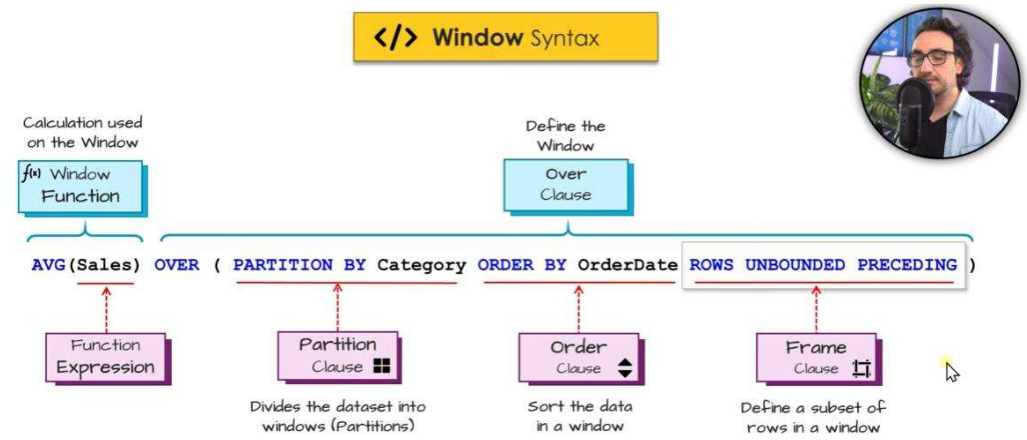
```
RANK() OVER (PARTITION BY Month ORDER BY Sales DESC)
```

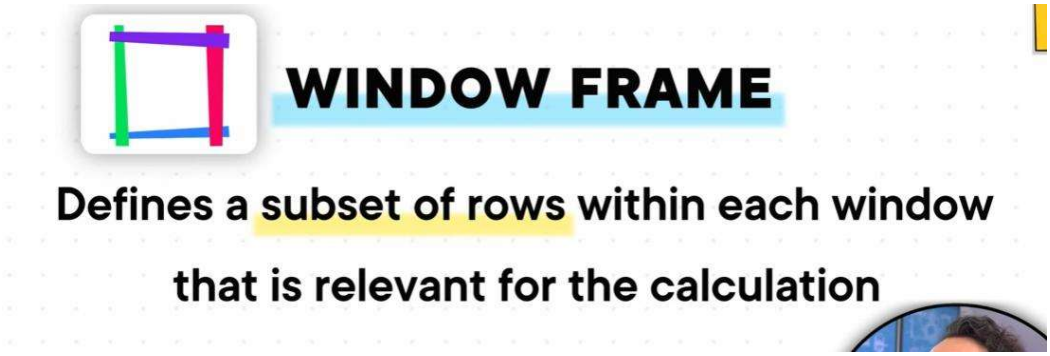


	OrderID	OrderDate	Sales	RankSales
1	8	2025-02-18	90	1
2	4	2025-01-20	60	2
3	10	2025-03-15	60	2
4	6	2025-02-05	50	4
5	7	2025-02-15	30	5
6	5	2025-02-01	25	6
7	9	2025-03-10	20	7
8	3	2025-01-10	20	7
9	2	2025-01-05	15	9
10	1	2025-01-01	10	10

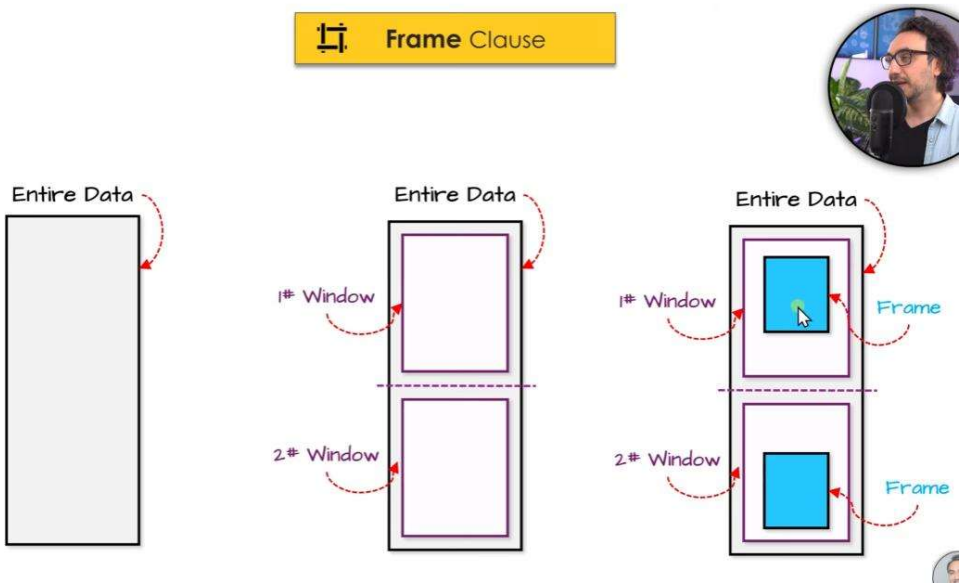


	OrderID	OrderDate	Sales	RankSales
1	1	2025-01-01	10	1
2	2	2025-01-05	15	2
3	3	2025-01-10	20	3
4	9	2025-03-10	20	3
5	5	2025-02-01	25	5
6	7	2025-02-15	30	6
7	6	2025-02-05	50	7
8	4	2025-01-20	60	8
9	10	2025-03-15	60	8
10	8	2025-02-18	90	10





Window inside window
Specific rows



When we foucus subset of data from each window

Frame Clause | Syntax

```
AVG(Sales) OVER (PARTITION BY Category ORDER BY OrderDate
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING)
```

- Frame Types: ROWS, RANGE
- Frame Boundary (Lower Value): CURRENT ROW, N PRECEDING, UNBOUNDED PRECEDING
- Frame Boundary (Higher Value): CURRENT ROW, N FOLLOWING, UNBOUNDED FOLLOWING

Rules

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

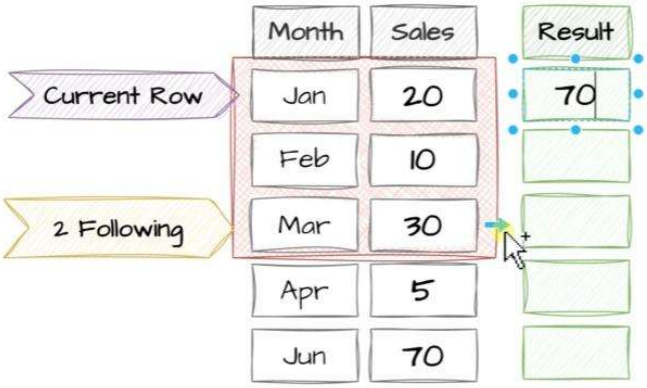
```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)
```

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

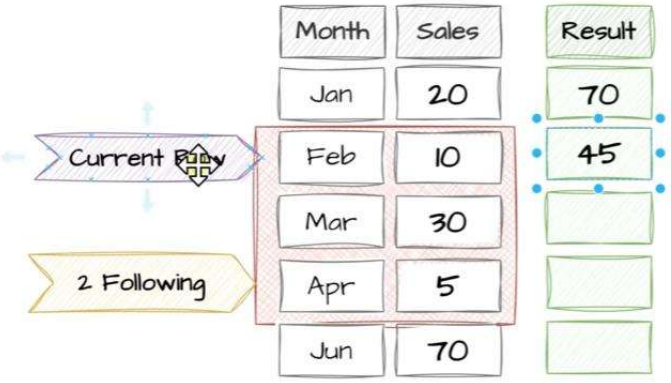
N-FOLLOWING

The n-th row before the current row

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

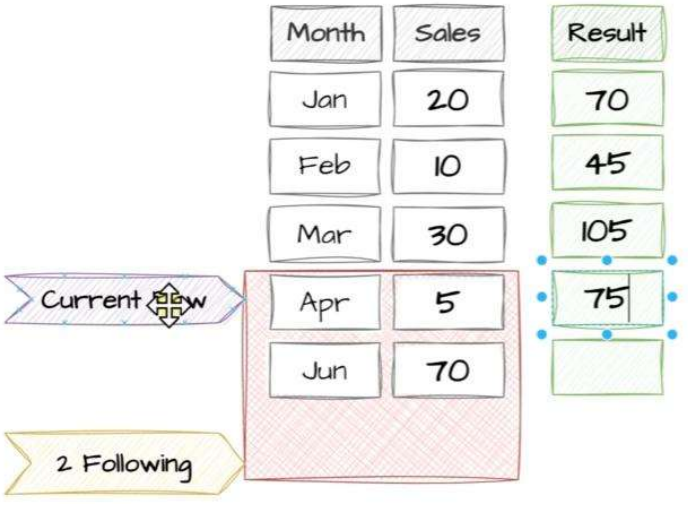


```
SUM(Sales)
OVER(ORDER BY Month
ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)
```



```
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)
```

```
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

Current Row

2 Following

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

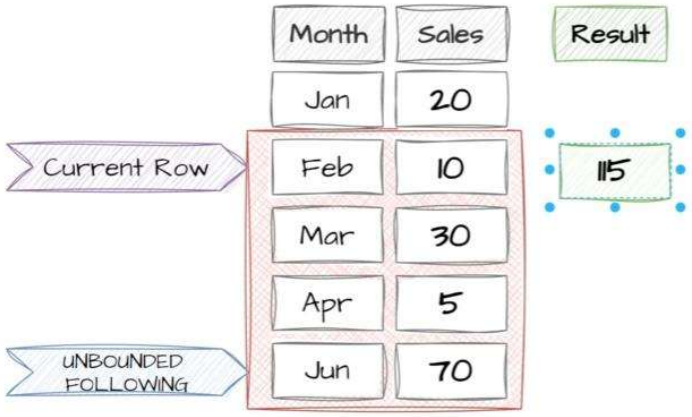
Month	Sales	Result
Jan	20	
Feb	10	
Mar	30	
Apr	5	
Jun	70	

UNBOUNDED FOLLOWING

The last possible row within a window

Ex- we start as a example from feb it goes on for all or samefor 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)
```