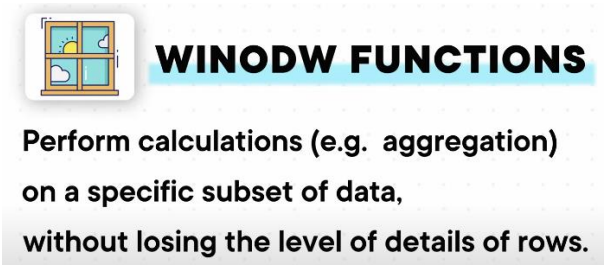
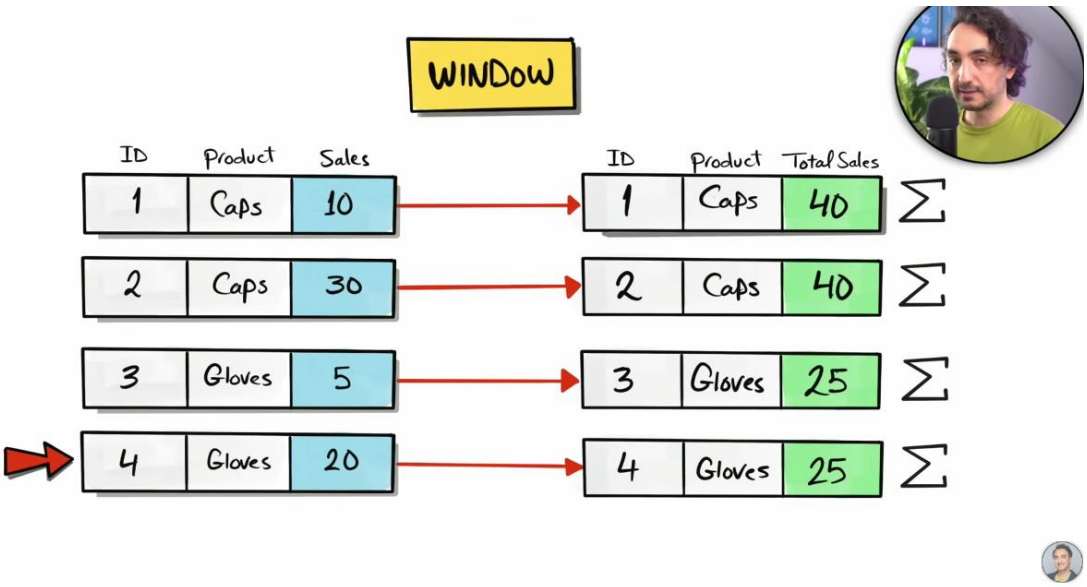
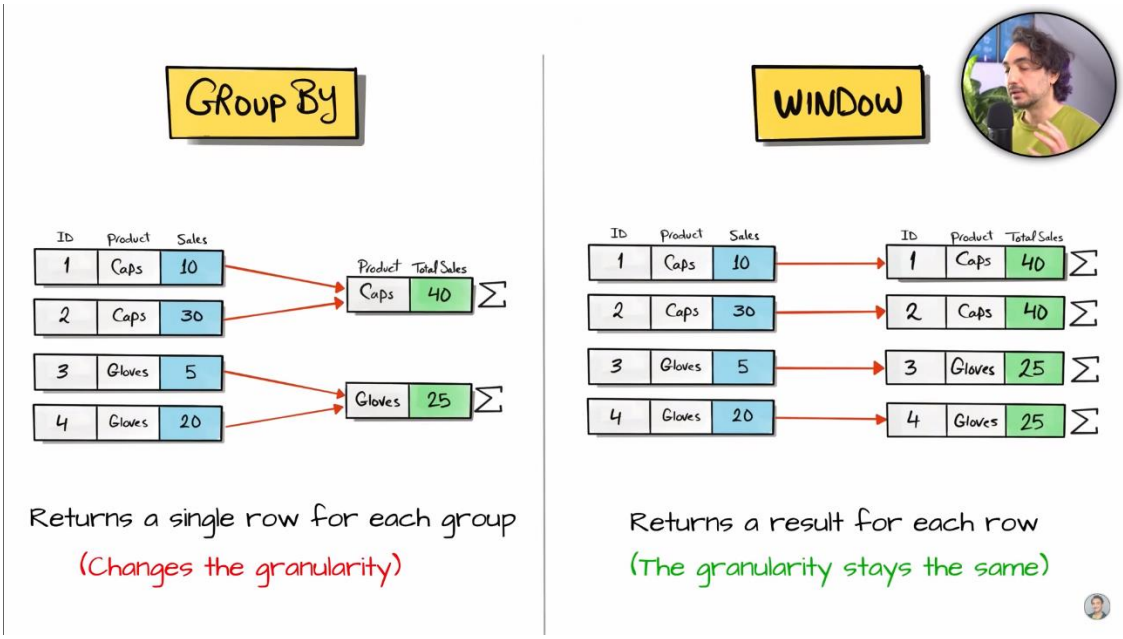
WINDOW FUNCTIONS:







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)

GROUP BY LIMITS

Can't do aggregations and provide details at same time

Result Granularity

Window functions returns a result for each row

QUERY 1:

Find total sales across all orders, additionally provide details such as order id and order date.

```
SELECT
OrderID,
OrderDate,
SUM(sales) over() AS Total_Sales
FROM Sales.Orders
```

	OrderID	OrderDate	Total_Sales
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

QUERY 2:

Find the total sales for each product, additionally provide details such as order id and order date.

```
SELECT
OrderID,
OrderDate,
ProductID,
SUM(sales) over(PARTITION BY ProductID) AS Total_Sales
FROM Sales.Orders
```

	OrderID	OrderDate	ProductID	Total_Sales
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
-- Find the total sales for each combination of product and order status
-- Additionally provide details such order Id, order date

SELECT
OrderID,
OrderDate,
ProductID,
OrderStatus,
SUM(sales) over() AS TotalSales,
SUM(sales) over(PARTITION BY ProductID) AS SalesByProduct,
SUM(sales) over(PARTITION BY ProductID,OrderStatus) AS SalesByProductAndOrderStatus
FROM Sales.Orders
```

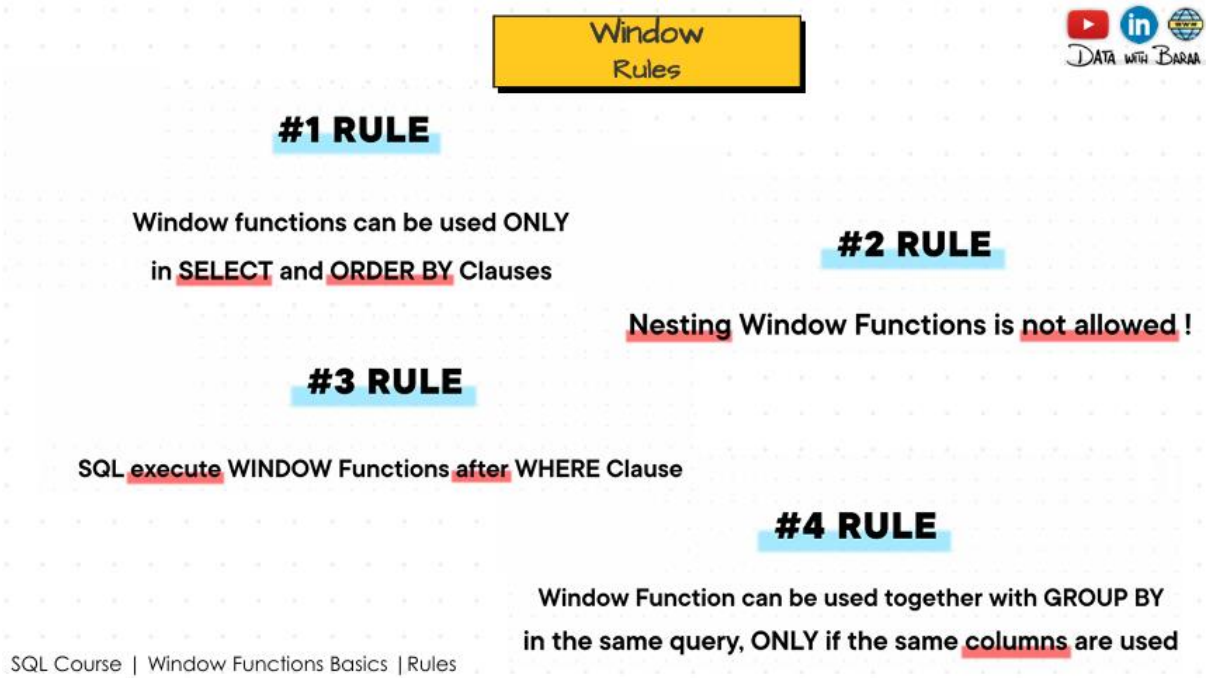
	OrderID	OrderDate	ProductID	OrderStatus	TotalSales	SalesByProduct	SalesByProductAndOrderStatus
1	1	2025-01-01	101	Delivered	380	140	30
2	3	2025-01-10	101	Delivered	380	140	30
3	8	2025-02-18	101	Shipped	380	140	110
4	9	2025-03-10	101	Shipped	380	140	110
5	7	2025-02-15	102	Delivered	380	105	30
6	10	2025-03-15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

QUERY 3:

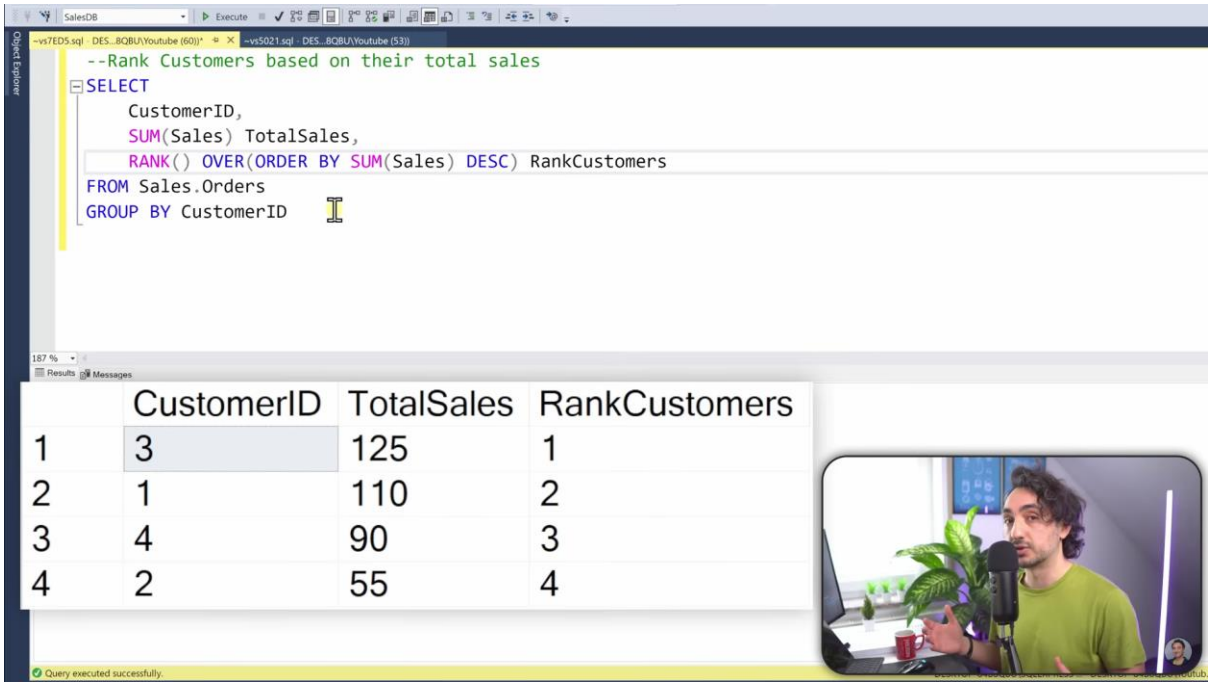
Rank each order base on their sales from highest to lowest, additionally provide details such order id and order date.

```
SELECT
OrderID,
OrderDate,
Sales,
RANK() OVER(ORDER BY Sales DESC) AS Rank_Sales
FROM Sales.Orders
```

	OrderID	OrderDate	Sales	Rank_Sales
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


Window
Rules
DATA WITH BARAA
#1 RULE
Window functions can be used ONLY
in SELECT and ORDER BY Clauses
#2 RULE
Nesting Window Functions is not allowed !
#3 RULE
SQL execute WINDOW Functions after WHERE Clause
#4 RULE
Window Function can be used together with GROUP BY
in the same query, ONLY if the same columns are used
SQL Course | Window Functions Basics | Rules

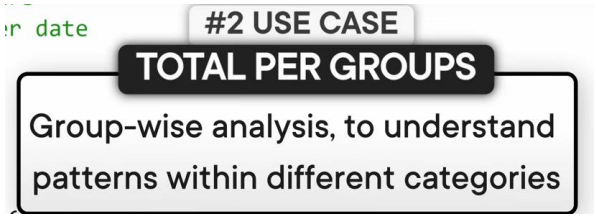
4.


--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
Results
Messages
CustomerID TotalSales RankCustomers
1 3 125 1
2 1 110 2
3 4 90 3
4 2 55 4
Query executed successfully.

WINDOW AGGREGATE FUNCTIONS:

```
-- Find the total number of Orders
-- Find the total number of Orders for each customers
-- Additionally provide details such order Id, order date
```

COUNT:




```
SELECT
OrderID,
OrderDate,
CustomerID,
COUNT(*) over() AS TotalOrders,
COUNT(OrderID) over(PARTITION BY CustomerID) AS TotalOrdersOfCustomers
FROM Sales.Orders
```

	OrderID	OrderDate	CustomerID	TotalOrders	TotalOrdersOfCustomers
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

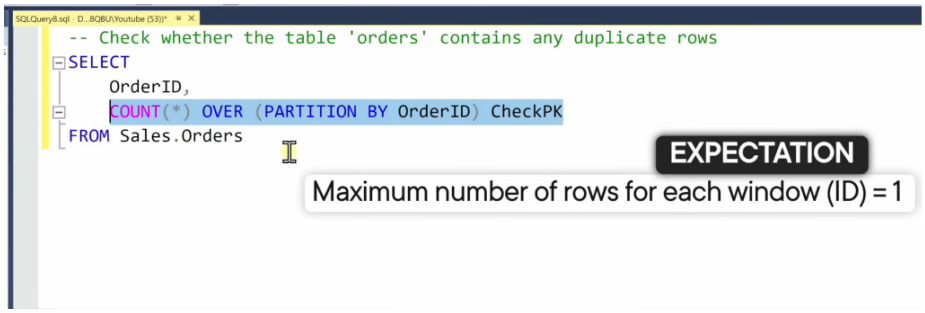
```
-- Find the total number of Customers
-- Additionally provide All customers Details
```

```
SELECT
*,
COUNT(CustomerID) over() AS TotalCustomers
FROM Sales.Customers
```

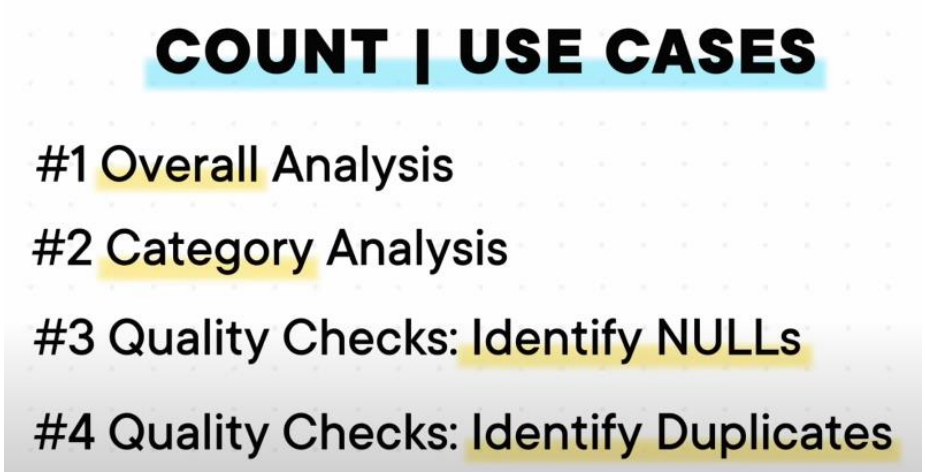
	CustomerID	FirstName	LastName	Country	Score	TotalCustomers
1	1	Jossef	Goldberg	Germany	350	5
2	2	Kevin	Brown	USA	900	5
3	3	Mary	NULL	USA	750	5
4	4	Mark	Schwarz	Germany	500	5
5	5	Anna	Adams	USA	NULL	5

USE CASE OF COUNT():

IDENTIFY DUPLICATES: Identify duplicate rows and to improve data quality.



To check the duplicates in the table orders, we should find the primary key first, and then use a window function to find, if the value greater than 1, then it will be a duplicate.



SUM ():


```
SELECT
OrderID,
OrderDate,
ProductID,
Sales,
SUM(Sales) over() AS TotalSales,
CAST(ROUND(CAST(Sales AS FLOAT)/SUM(Sales) OVER()*100 , 2) AS VARCHAR) + '%' AS
PercentageOfSales
FROM Sales.Orders
```

	OrderID	OrderDate	ProductID	Sales	TotalSales	PercentageOfSales
1	1	2025-01-01	101	10	380	2.63 %
2	2	2025-01-05	102	15	380	3.95 %
3	3	2025-01-10	101	20	380	5.26 %
4	4	2025-01-20	105	60	380	15.79 %
5	5	2025-02-01	104	25	380	6.58 %
6	6	2025-02-05	104	50	380	13.16 %
7	7	2025-02-15	102	30	380	7.89 %
8	8	2025-02-18	101	90	380	23.68 %
9	9	2025-03-10	101	20	380	5.26 %
10	10	2025-03-15	102	60	380	15.79 %

AVG ():

Before using avg window function, do null handling using COALESCE, and then window function.

```
-- Find the average sales across all orders
-- And Find the average sales for each product
-- Additionally provide details such order Id, order date
```

```
-- 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
```

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

SO WE USED SUBQUERY.

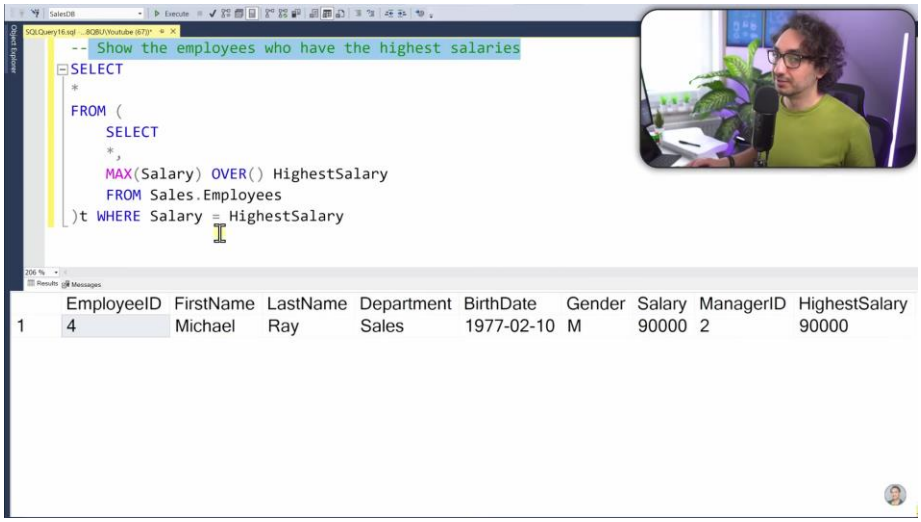
```
SELECT
*
FROM(
    SELECT
        OrderID,
        ProductID,
        Sales,
        AVG(Sales) OVER() AvgSales
    FROM Sales.Orders
)t WHERE Sales > AvgSales
```

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

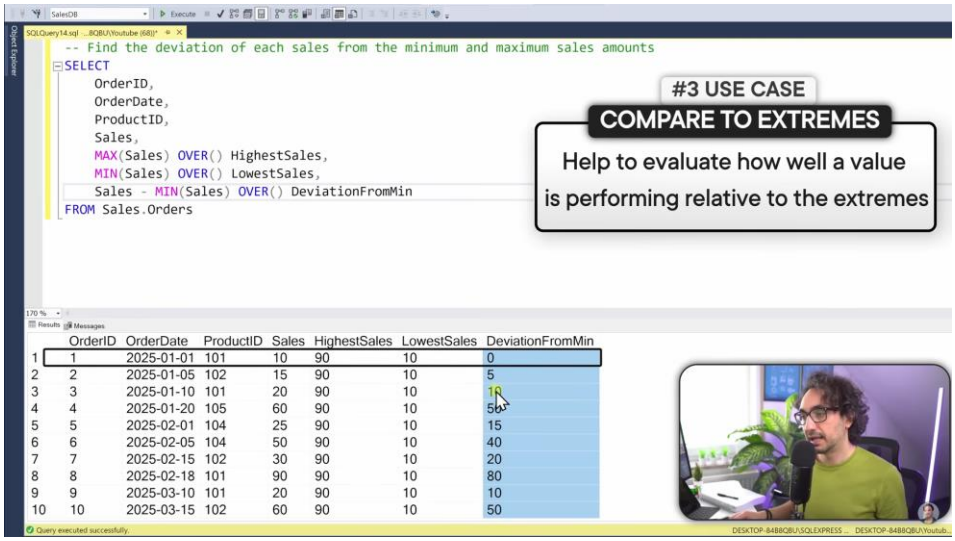
MIN/MAX ():

```
-- 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
```

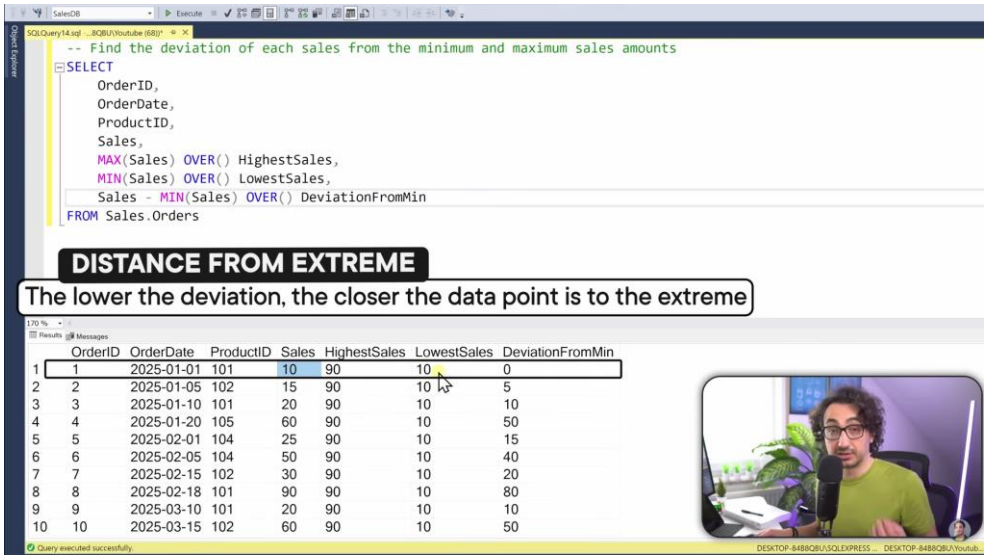
	OrderID	OrderDate	ProductID	Sales	HighestSales	LowestSales	HighestSalesByProduct	LowestSalesByProduct
1	1	2025-01-01	101	10	90	10	90	10
2	3	2025-01-10	101	20	90	10	90	10
3	8	2025-02-18	101	90	90	10	90	10
4	9	2025-03-10	101	20	90	10	90	10
5	10	2025-03-15	102	60	90	10	60	15
6	2	2025-01-05	102	15	90	10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



	EmployeeID	FirstName	LastName	Department	BirthDate	Gender	Salary	ManagerID	HighestSalary
1	4	Michael	Ray	Sales	1977-02-10	M	90000	2	90000



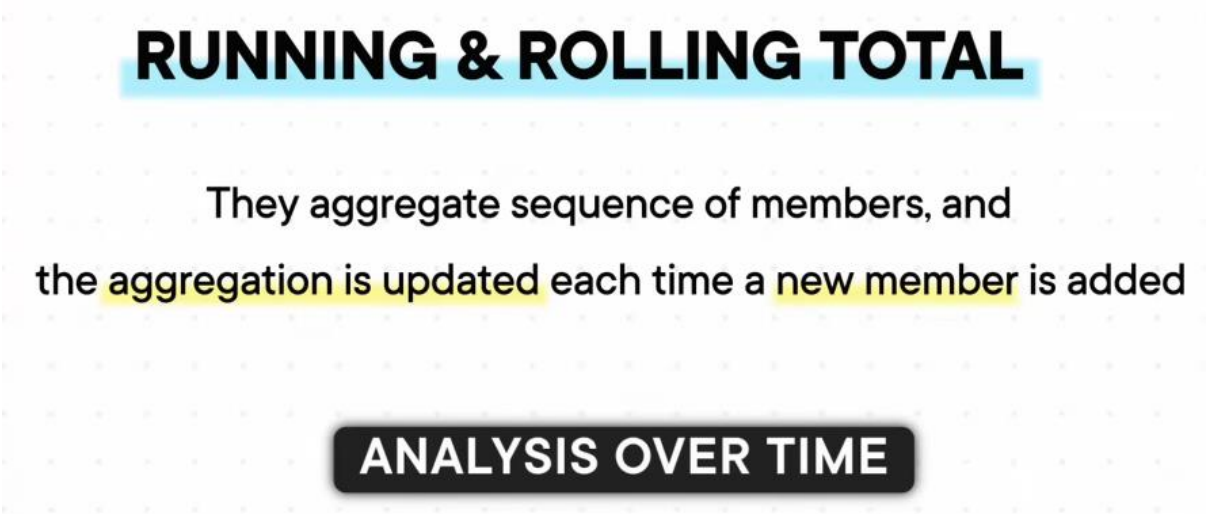
	OrderID	OrderDate	ProductID	Sales	HighestSales	LowestSales	DeviationFromMin
1	1	2025-01-01	101	10	90	10	0
2	2	2025-01-05	102	15	90	10	5
3	3	2025-01-10	101	20	90	10	10
4	4	2025-01-20	105	60	90	10	50
5	5	2025-02-01	104	25	90	10	15
6	6	2025-02-05	104	50	90	10	40
7	7	2025-02-15	102	30	90	10	20
8	8	2025-02-18	101	90	90	10	80
9	9	2025-03-10	101	20	90	10	10
10	10	2025-03-15	102	60	90	10	50



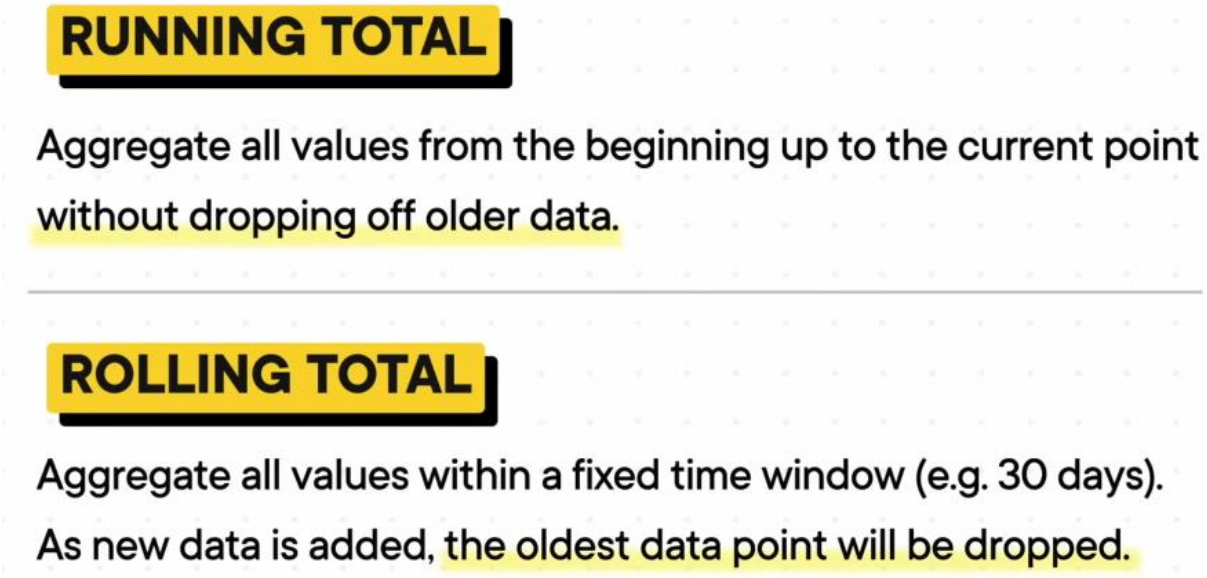
	OrderID	OrderDate	ProductID	Sales	HighestSales	LowestSales	DeviationFromMin
1	1	2025-01-01	101	10	90	10	0
2	2	2025-01-05	102	15	90	10	5
3	3	2025-01-10	101	20	90	10	10
4	4	2025-01-20	105	60	90	10	50
5	5	2025-02-01	104	25	90	10	15
6	6	2025-02-05	104	50	90	10	40
7	7	2025-02-15	102	30	90	10	20
8	8	2025-02-18	101	90	90	10	80
9	9	2025-03-10	101	20	90	10	10
10	10	2025-03-15	102	60	90	10	50

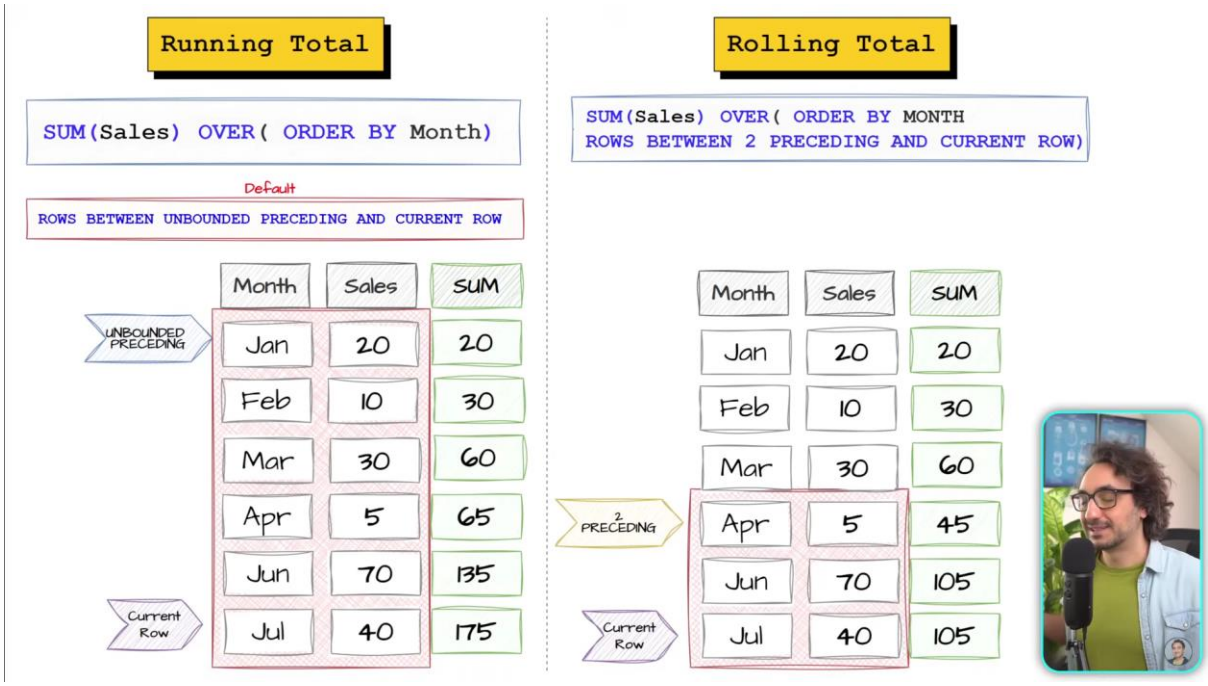
RUNNING TOTAL AND ROLLING TOTAL.

KEY USES:



--






```
-- Calculate moving average of sales for each product over time
```

Over time analysis means sorting dates in ascending order

```
SELECT
    OrderID,
    ProductID,
    OrderDate,
    Sales,
    AVG(Sales) OVER (PARTITION BY ProductID) AvgByProduct,
    AVG(Sales) OVER (PARTITION BY ProductID ORDER BY OrderDate) MovingAvg
FROM Sales.Orders
```

OrderID	ProductID	OrderDate	Sales	AvgByProduct	MovingAvg
1	101	2025-01-01	10	35	10
3	101	2025-01-10	20	35	15
8	101	2025-02-18	90	35	40
9	101	2025-03-10	20	35	35
2	102	2025-01-05	15	35	15
7	102	2025-02-15	30	35	22
10	102	2025-03-15	60	35	35
5	104	2025-02-01	25	37	25
6	104	2025-02-05	50	37	37
4	105	2025-01-20	60	60	60

```sql
-- Calculate moving average of sales for each product over time
-- Calculate moving average of sales for each product over time, including only the next order
SELECT
    OrderID,
    ProductID,
    OrderDate,
    Sales,
    AVG(Sales) OVER (PARTITION BY ProductID) AvgByProduct,
    AVG(Sales) OVER (PARTITION BY ProductID ORDER BY OrderDate) MovingAvg,
    AVG(Sales) OVER (PARTITION BY ProductID ORDER BY OrderDate ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) RollingAvg
FROM Sales.Orders
```

	OrderID	ProductID	OrderDate	Sales	AvgByProduct	MovingAvg	RollingAvg
1	1	101	2025-01-01	10	35	10	15
2	3	101	2025-01-10	20	35	15	55
3	8	101	2025-02-18	90	35	40	55
4	9	101	2025-03-10	20	35	35	20
5	2	102	2025-01-05	15	35	15	22
6	7	102	2025-02-15	30	35	22	45
7	10	102	2025-03-15	60	35	35	60
8	5	104	2025-02-01	25	37	25	37
9	6	104	2025-02-05	50	37	37	50
10	4	105	2025-01-20	60	60	60	60

Overall Total

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

Overview of entire data

Month	Sales	SUM
Caps	20	175
Caps	10	175
Caps	30	175
Gloves	5	175
Gloves	70	175
Gloves	40	175

Total Per Groups

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

Compare Categories

Month	Sales	SUM
Caps	20	60
Caps	10	60
Caps	30	60
Gloves	5	105
Gloves	70	105
Gloves	40	105

Running Total

```sql
SUM(Sales) OVER(
ORDER BY MONTH)
```

progress over time

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	65
Jun	70	135
Jul	40	175

Rolling Total

```sql
SUM(Sales) OVER(
ORDER BY MONTH
ROWS 2 PRECEDING)
```

progress over time in specific fixed window

Month	Sales	SUM
Jan	20	20
Feb	10	30
Mar	30	60
Apr	5	45
Jun	70	105
Jul	40	105

WINDOW AGGREGATE FUNCTIONS

Aggregate set of values and return a single aggregated value

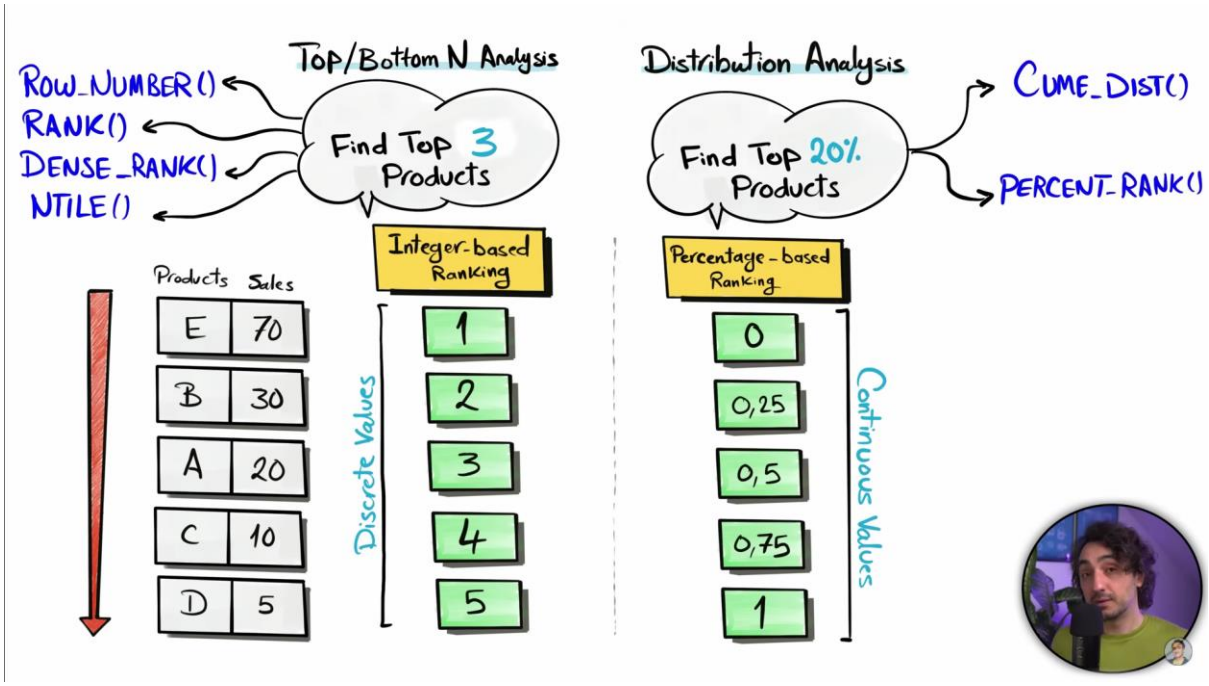
Rules

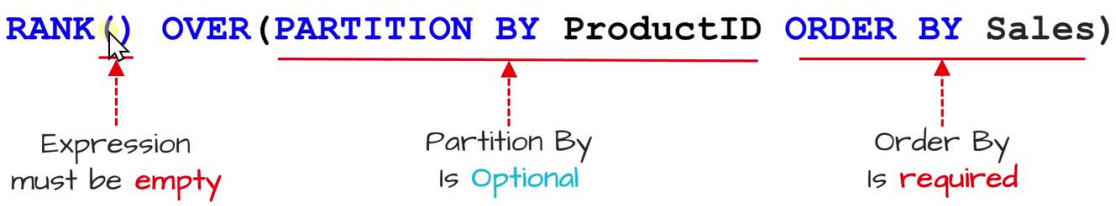
- Expressions
  - Numbers (All Functions)
  - Any Data Type – COUNT()
- All Clauses are Optional

Use Cases

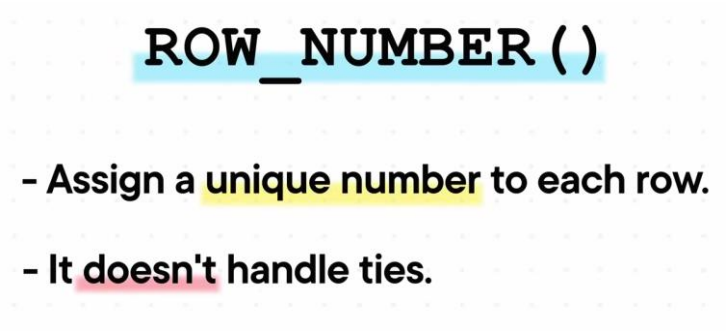
- Overall Analysis
- Total Per Groups Analysis
- Part-to-Whole Analysis
- Comparison Analysis
  - Average
  - Extreme: Highest/Lowest
- Identify Duplicates
- Outlier Detection
- Running Total
- Rolling Total
- Moving Average

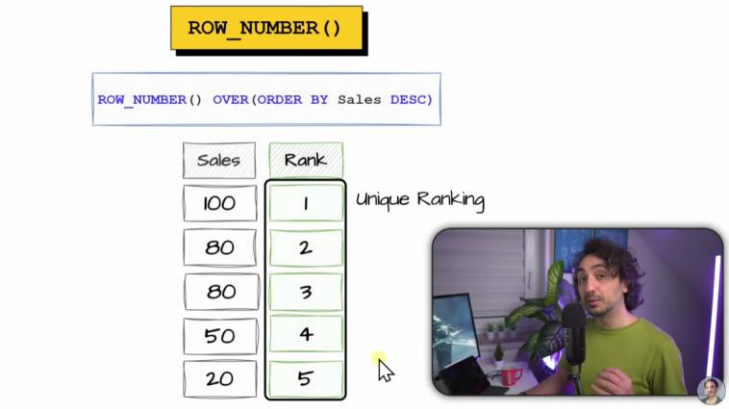
WINDOW RANK FUNCTIONS:

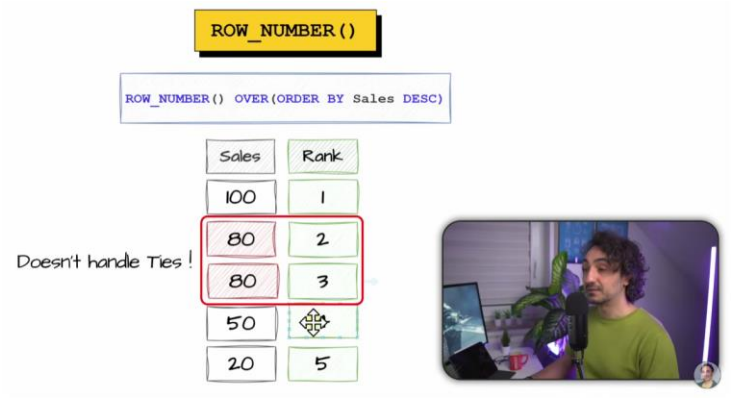


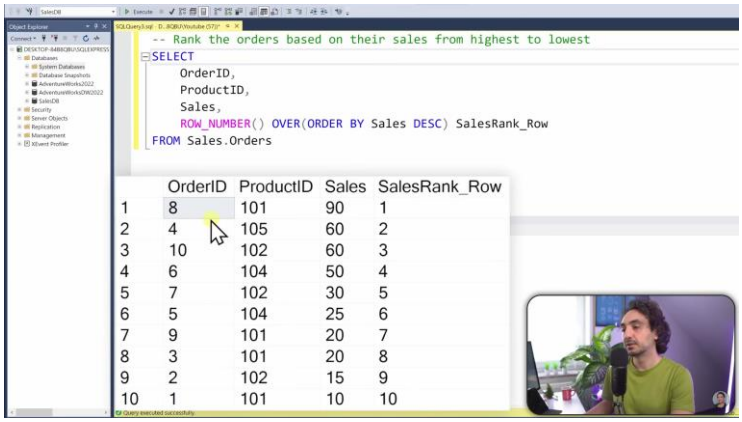


1)



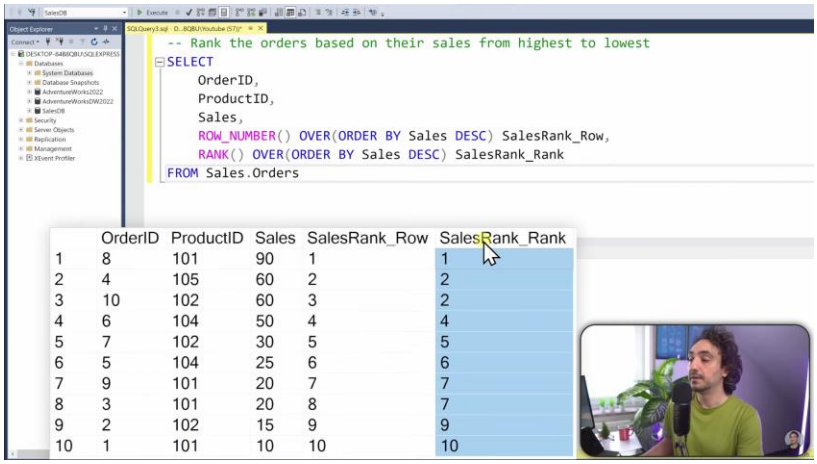



ROW_NUMBER()
ROW_NUMBER() OVER(ORDER BY Sales DESC)
Sales
Rank
100
1
80
2
80
3
50
4
20
5
Doesn't handle Ties !

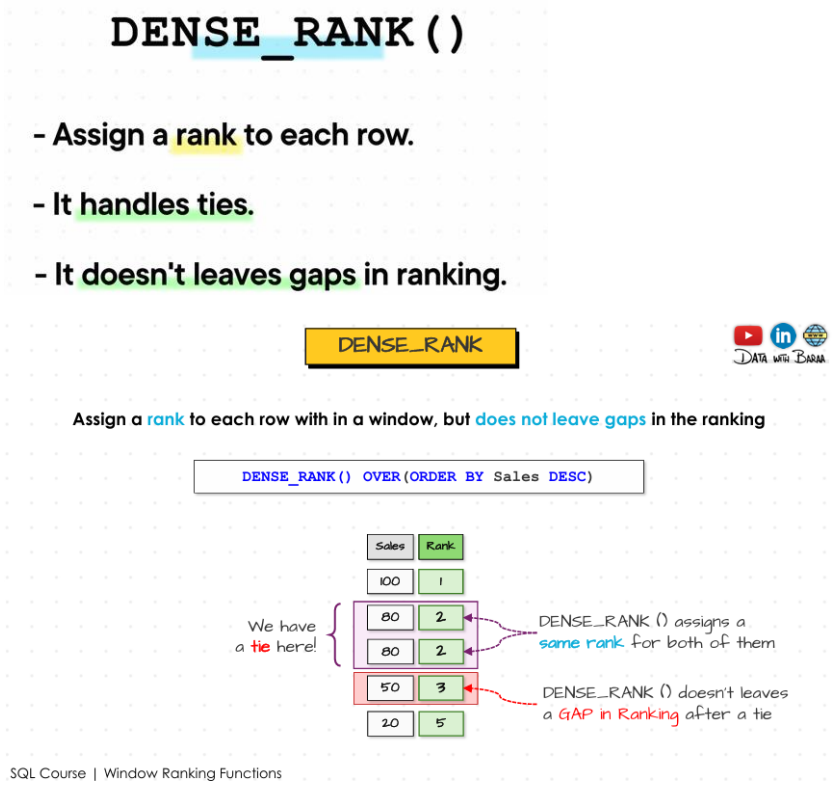

-- Rank the orders based on their sales from highest to lowest
SELECT
OrderID,
ProductID,
Sales,
ROW_NUMBER() OVER(ORDER BY Sales DESC) SalesRank_Row
FROM Sales.Orders
OrderID ProductID Sales SalesRank_Row
1 8 101 90 1
2 4 105 60 2
3 10 102 60 3
4 6 104 50 4
5 7 102 30 5
6 5 104 25 6
7 9 101 20 7
8 3 101 20 8
9 2 102 15 9
10 1 101 10 10

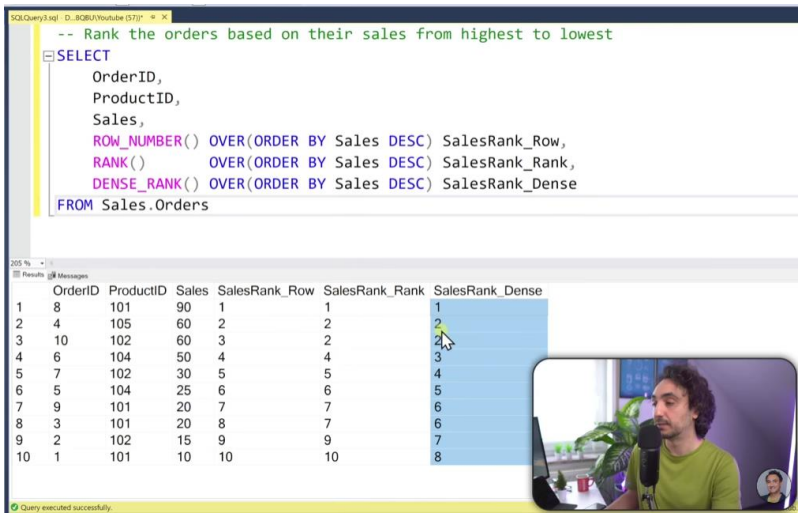
2)

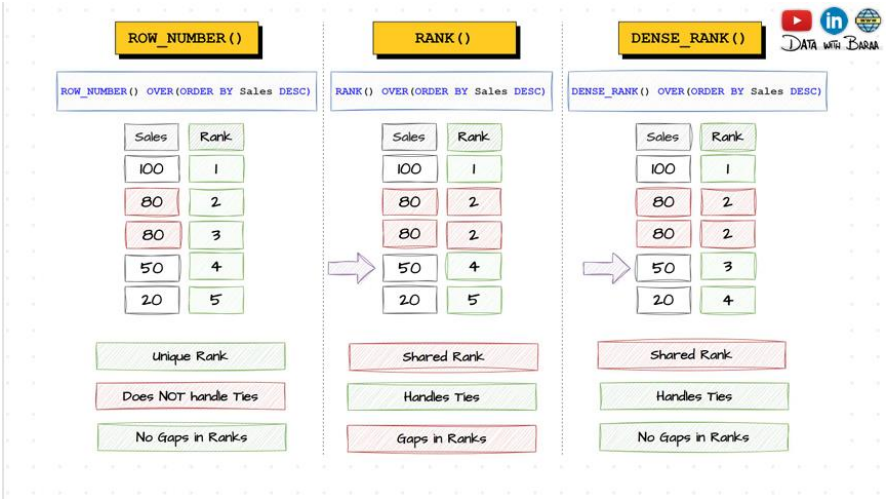

RANK ( )
- Assign a rank to each row.
- It handles ties.
- It leaves gaps in ranking.
RANK
DATA WITH BARAA
Assign a rank to each row with in a window
RANK() OVER(ORDER BY Sales DESC)
Sales
Rank
100
1
80
2
80
2
50
4
20
5
We have a tie here!
RANK () assigns a same rank for both of them
RANK () leaves a GAP in Ranking after a tie
SQL Course | Window Ranking Functions

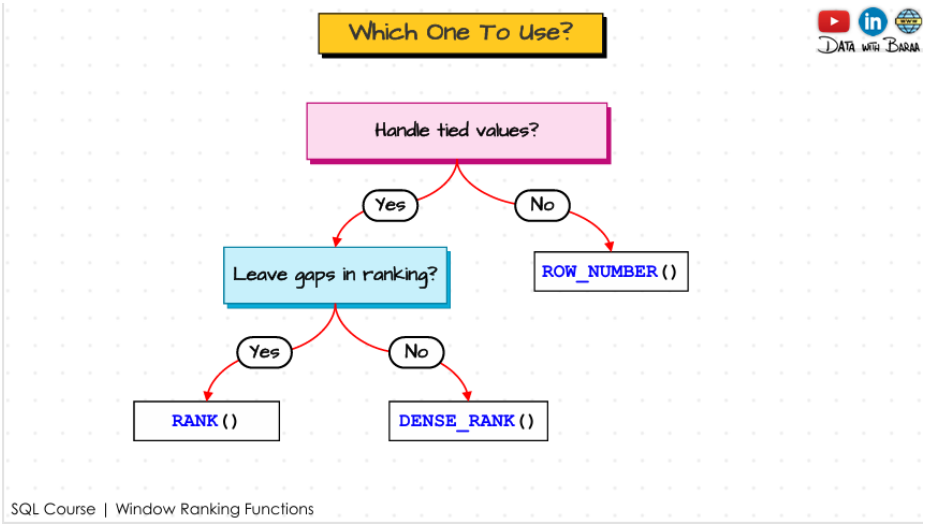

-- Rank the orders based on their sales from highest to lowest
SELECT
OrderID,
ProductID,
Sales,
ROW_NUMBER() OVER(ORDER BY Sales DESC) SalesRank_Row,
RANK() OVER(ORDER BY Sales DESC) SalesRank_Rank
FROM Sales.Orders
OrderID ProductID Sales SalesRank_Row SalesRank_Rank
1 8 101 90 1 1
2 4 105 60 2 2
3 10 102 60 3 2
4 6 104 50 4 4
5 7 102 30 5 5
6 5 104 25 6 6
7 9 101 20 7 7
8 3 101 20 8 7
9 2 102 15 9 9
10 1 101 10 10 10

3)


DENSE_RANK()
- Assign a rank to each row.
- It handles ties.
- It doesn't leaves gaps in ranking.
DENSE_RANK
DATA WITH BARAA
Assign a rank to each row with in a window, but does not leave gaps in the ranking
DENSE_RANK() OVER(ORDER BY Sales DESC)
Sales
Rank
100
1
80
2
80
2
50
3
20
5
We have a tie here!
DENSE_RANK () assigns a same rank for both of them
DENSE_RANK () doesn't leaves a GAP in Ranking after a tie
SQL Course | Window Ranking Functions

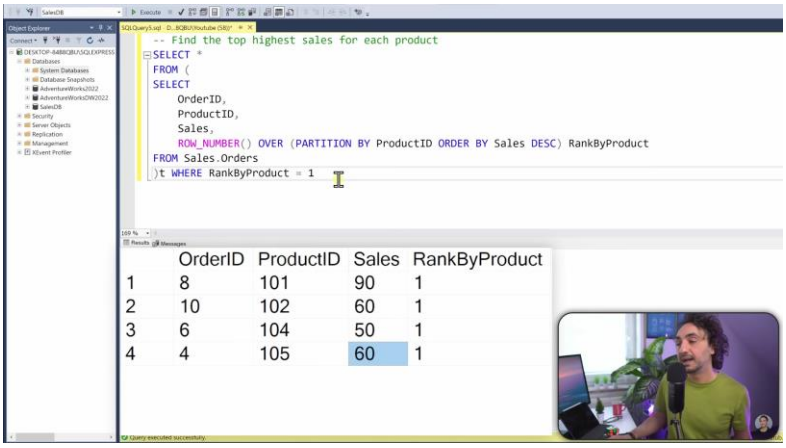

-- Rank the orders based on their sales from highest to lowest
SELECT
OrderID,
ProductID,
Sales,
ROW_NUMBER() OVER(ORDER BY Sales DESC) SalesRank_Row,
RANK() OVER(ORDER BY Sales DESC) SalesRank_Rank,
DENSE_RANK() OVER(ORDER BY Sales DESC) SalesRank_Dense
FROM Sales.Orders
OrderID ProductID Sales SalesRank_Row SalesRank_Rank SalesRank_Dense
1 8 101 90 1 1 1
2 4 105 60 2 2 2
3 10 102 60 3 2 2
4 6 104 50 4 4 3
5 7 102 30 5 5 4
6 5 104 25 6 6 5
7 9 101 20 7 7 6
8 3 101 20 8 7 6
9 2 102 15 9 9 7
10 1 101 10 10 10 8



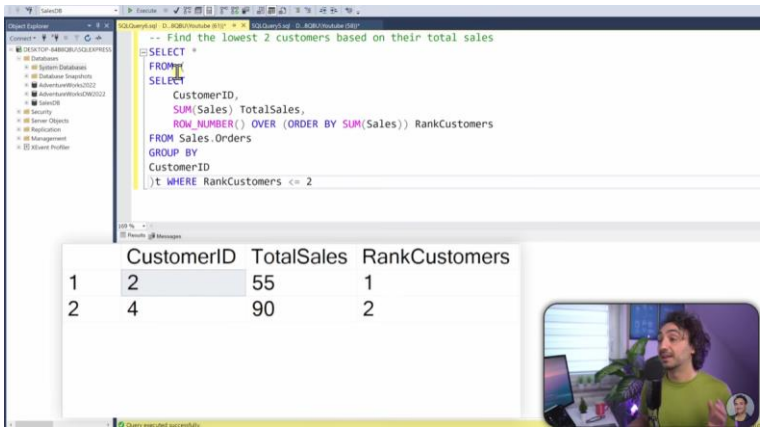


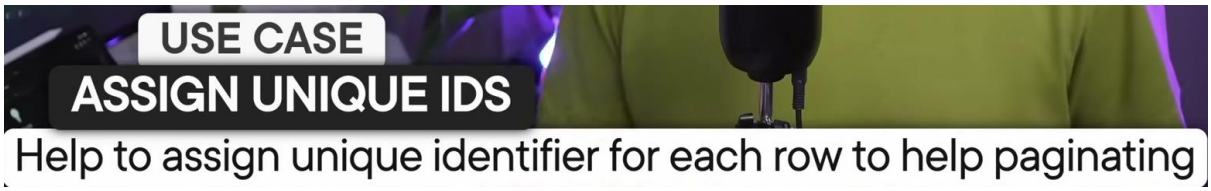
SQL Course | Window Ranking Functions

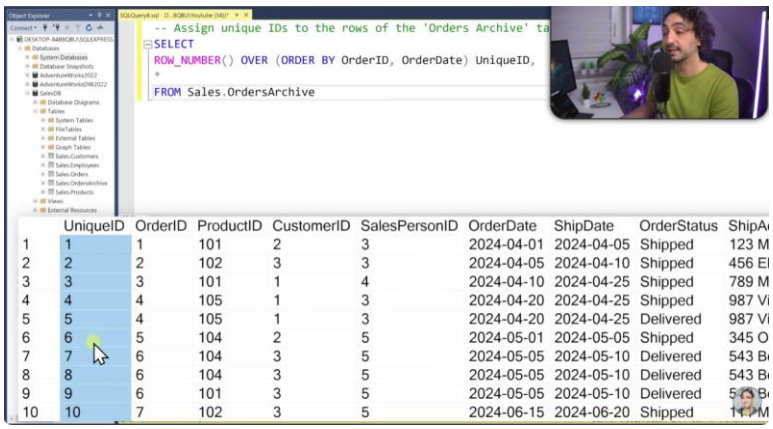

```
-- Find the top highest sales for each product
```




```
-- Find the lowest 2 customers based on their total sales
```



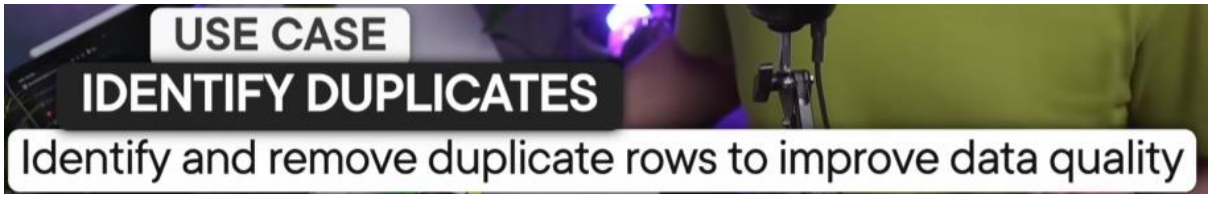


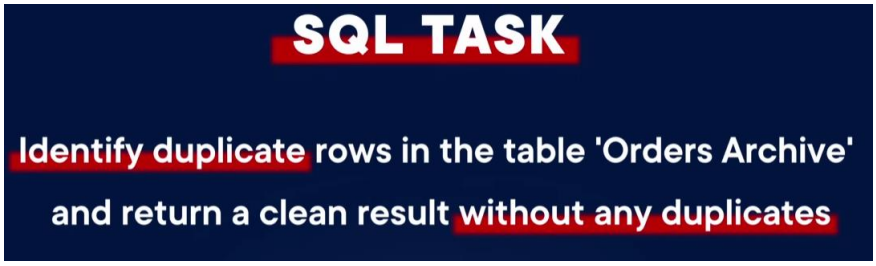


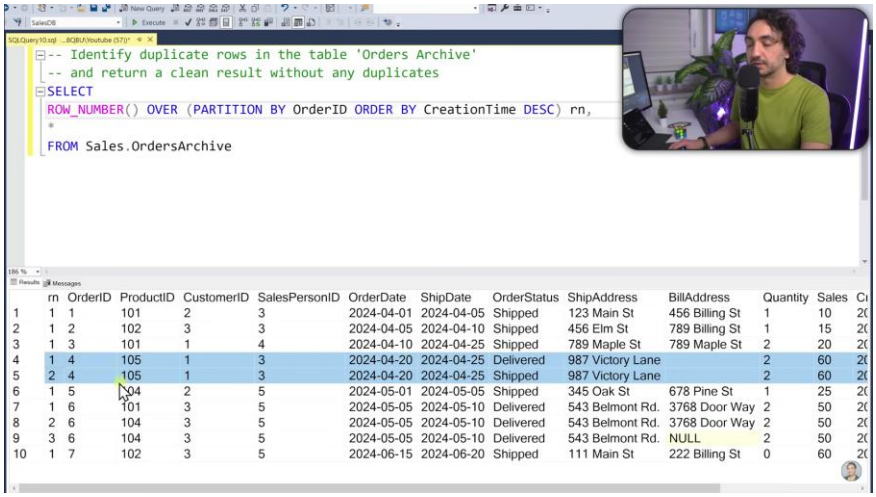
PAGINATING

The process of breaking down a large data into smaller, more manageable chunks

---







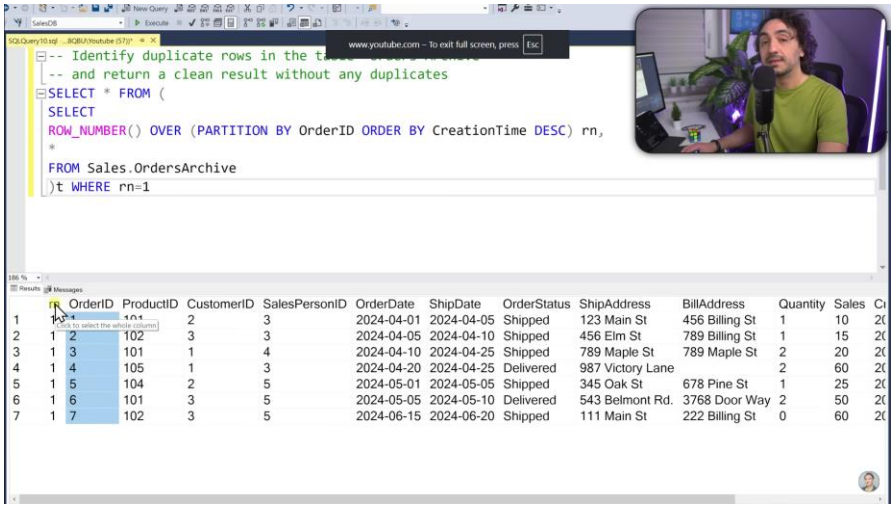
	rn	OrderID	ProductID	CustomerID	SalesPersonID	OrderDate	ShipDate	OrderStatus	ShipAddress	BillAddress	Quantity	Sales	Cu
1	1	1	101	2	3	2024-04-01	2024-04-05	Shipped	123 Main St	456 Billing St	1	10	20
2	1	2	102	3	3	2024-04-05	2024-04-10	Shipped	456 Elm St	789 Billing St	1	15	20
3	1	3	101	1	4	2024-04-10	2024-04-25	Shipped	789 Maple St	789 Maple St	2	20	20
4	1	4	105	1	3	2024-04-20	2024-04-25	Delivered	987 Victory Lane		2	60	20
5	2	4	105	1	3	2024-04-20	2024-04-25	Shipped	987 Victory Lane		2	60	20
6	1	5	104	2	5	2024-05-01	2024-05-05	Shipped	345 Oak St	678 Pine St	1	25	20
7	1	6	101	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	20
8	2	6	104	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	20
9	3	6	104	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	NULL	2	50	20
10	1	7	102	3	5	2024-06-15	2024-06-20	Shipped	111 Main St	222 Billing St	0	60	20

Partition By is used to the primary key of the table.

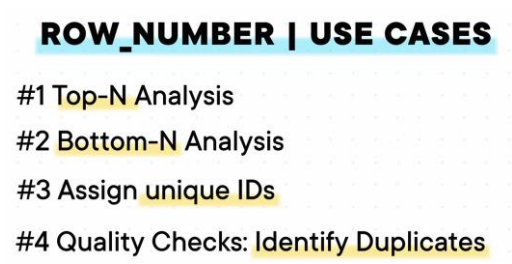
And Creation is to select the valid entry or latest entry.

Put whole query in subquery and filter with WHERE rn=1.

The duplicates will be removed.



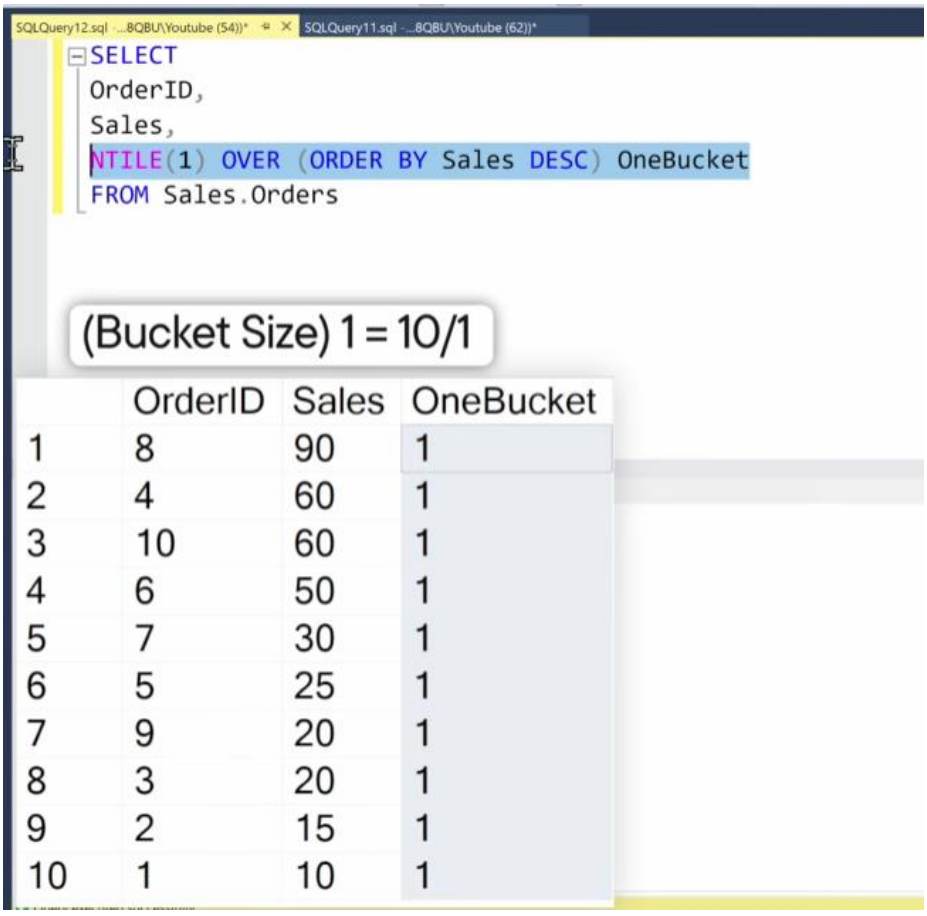
	rn	OrderID	ProductID	CustomerID	SalesPersonID	OrderDate	ShipDate	OrderStatus	ShipAddress	BillAddress	Quantity	Sales	Cu
1	1	1	101	2	3	2024-04-01	2024-04-05	Shipped	123 Main St	456 Billing St	1	10	20
2	1	2	102	3	3	2024-04-05	2024-04-10	Shipped	456 Elm St	789 Billing St	1	15	20
3	1	3	101	1	4	2024-04-10	2024-04-25	Shipped	789 Maple St	789 Maple St	2	20	20
4	1	4	105	1	3	2024-04-20	2024-04-25	Delivered	987 Victory Lane		2	60	20
5	1	5	104	2	5	2024-05-01	2024-05-05	Shipped	345 Oak St	678 Pine St	1	25	20
6	1	6	101	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	20
7	1	7	102	3	5	2024-06-15	2024-06-20	Shipped	111 Main St	222 Billing St	0	60	20



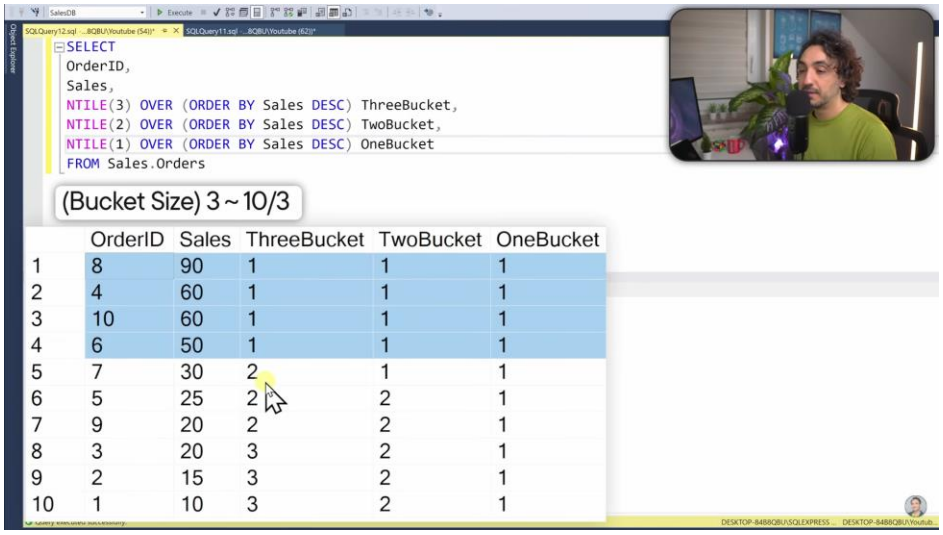
4)

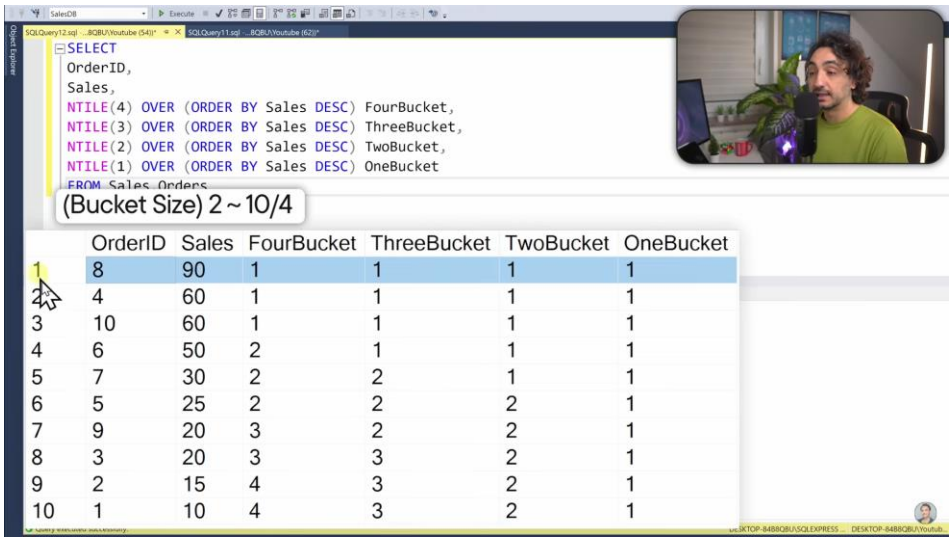
NTILE ( )

Divides the rows into a specified number of approximately equal groups (Buckets)



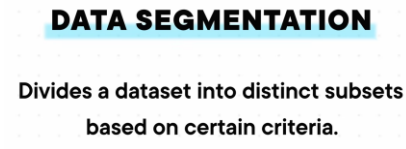
	OrderID	Sales	OneBucket
1	8	90	1
2	4	60	1
3	10	60	1
4	6	50	1
5	7	30	1
6	5	25	1
7	9	20	1
8	3	20	1
9	2	15	1
10	1	10	1

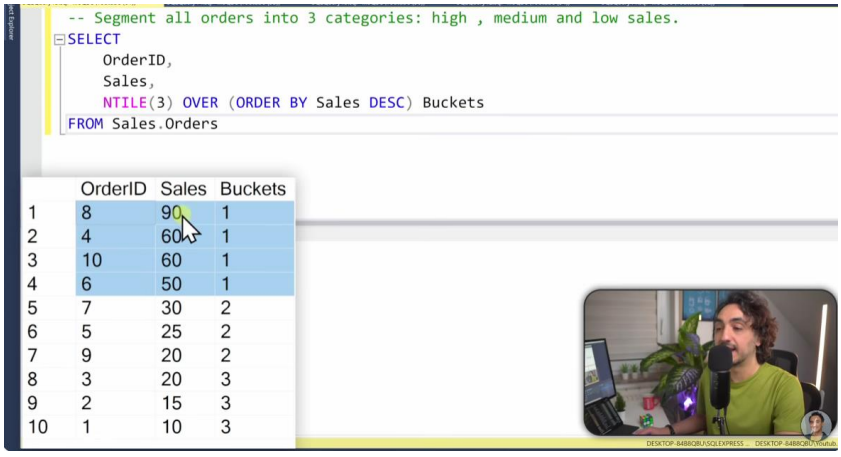




NTILE USE CASE:

DATA SEGMENTATION.





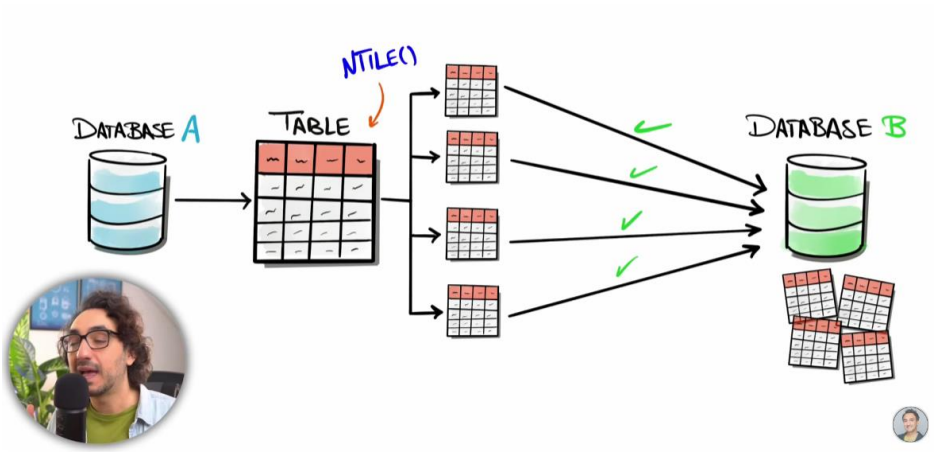
```sql
-- Segment all orders into 3 categories: high , medium and low sales.
SELECT
*,
CASE WHEN Buckets = 1 THEN 'High'
     WHEN Buckets = 2 THEN 'Medium'
     WHEN Buckets = 3 THEN 'Low'
END SalesSegmentations
FROM (
    SELECT
        OrderID,
        Sales,
        NTILE(3) OVER (ORDER BY Sales DESC) Buckets
    FROM Sales.Orders
)t
```

	OrderID	Sales	Buckets	SalesSegmentations
1	8	90	1	High
2	4	60	1	High
3	10	60	1	High
4	6	50	1	High
5	7	30	2	Medium
6	5	25	2	Medium
7	9	20	2	Medium
8	3	20	3	Low
9	2	15	3	Low
10	1	10	3	Low

Query executed successfully.

USE CASE:

EQUALISING LOAD:




```sql
-- In order to export the data, divide the orders into 2 groups.
SELECT
NTILE(2) OVER ( ORDER BY OrderID) Buckets,
*
FROM Sales.Orders
```

	Buckets	OrderID	ProductID	CustomerID	SalesPersonID	OrderDate	ShipDate	OrderStatus	ShipAddress	BillAddress	Quantity	Sa
1	1	1	101	2	3	2025-01-01	2025-01-05	Delivered	9833 Mt. Dias Blv.	1226 Shoe St.	1	10
2	1	2	102	3	3	2025-01-05	2025-01-10	Shipp				5
3	1	3	101	1	5	2025-01-10	2025-01-25	Delive				0
4	1	4	105	1	3	2025-01-20	2025-01-25	Shipp				0
5	1	5	104	2	5	2025-02-01	2025-02-05	Delive				5
6	2	6	104	3	5	2025-02-05	2025-02-10	Delive				0
7	2	7	102	1	1	2025-02-15	2025-02-27	Delive				0
8	2	8	101	4	3	2025-02-18	2025-02-27	Shipp				0
9	2	9	101	2	3	2025-03-10	2025-03-15	Shipp				0
10	2	10	102	3	5	2025-03-15	2025-03-20	Shipp				0

Query executed successfully.

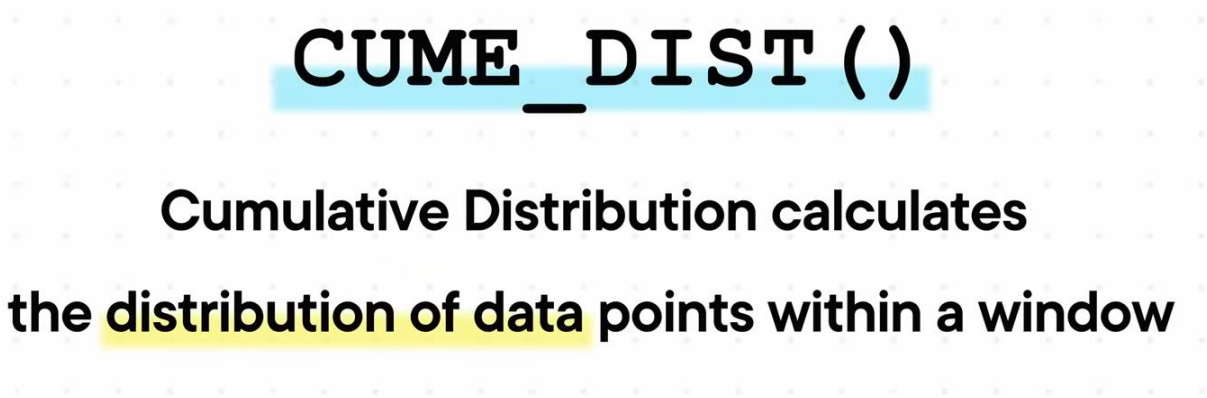
RANK WINDOW FUNCTION:

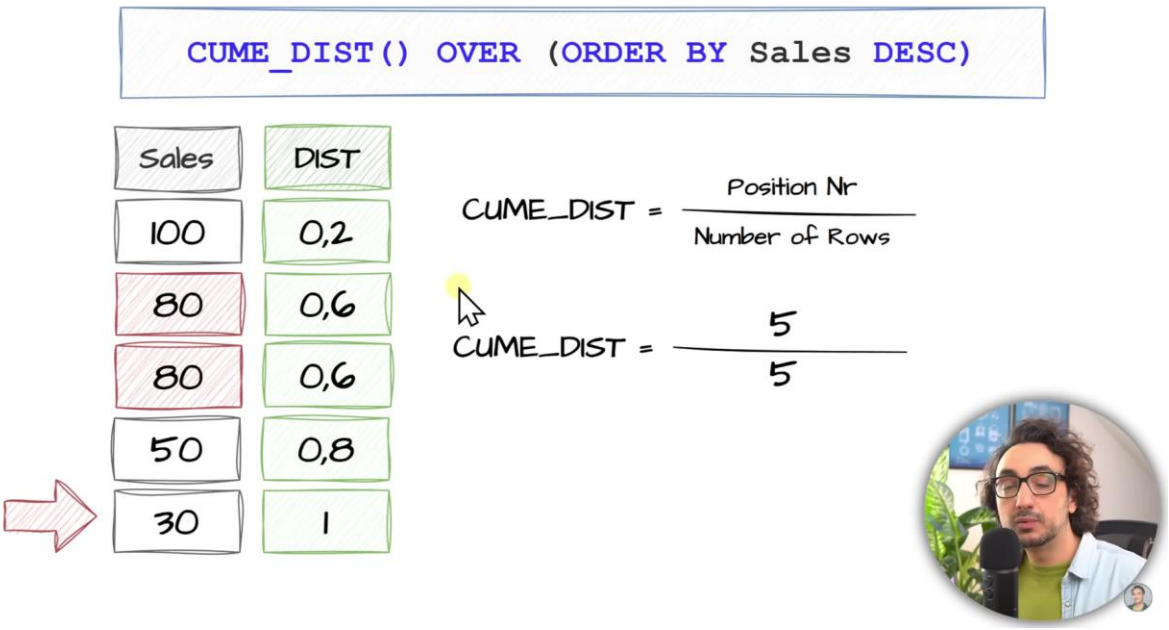
PERCENTAGE – BASED RANKING:

CUME_DIST

PERCENT_RANK

CUME_DIST:





PERCENT_RANK()

Calculates the relative position of each row

```
PERCENT_RANK() OVER (ORDER BY Sales DESC)
```

Sales	Per
100	0
80	0,25
80	0,25
50	0,75
30	1

$$\text{Percent\_Rank} = \frac{\text{Position Nr} - 1}{\text{Number of Rows} - 1}$$

$$\text{Percent\_Rank} = \frac{4}{4}$$

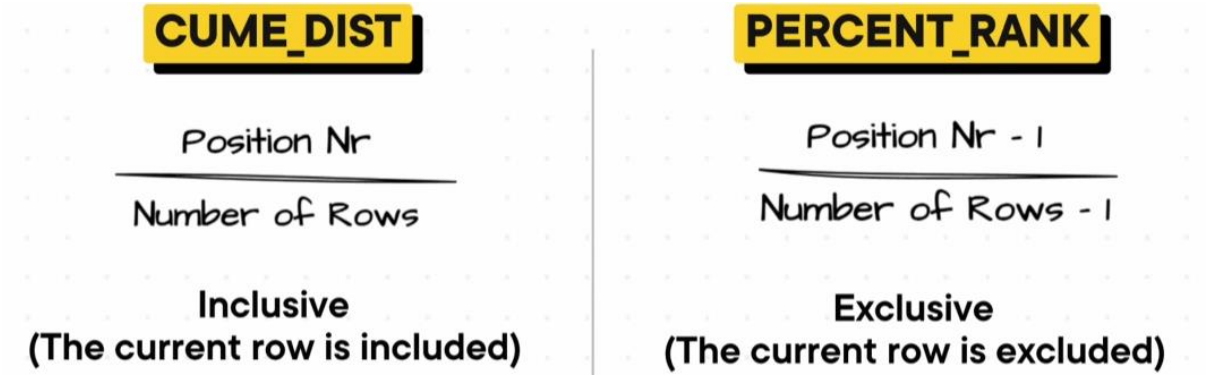
```
CUME_DIST() OVER (ORDER BY Sales DESC)
```

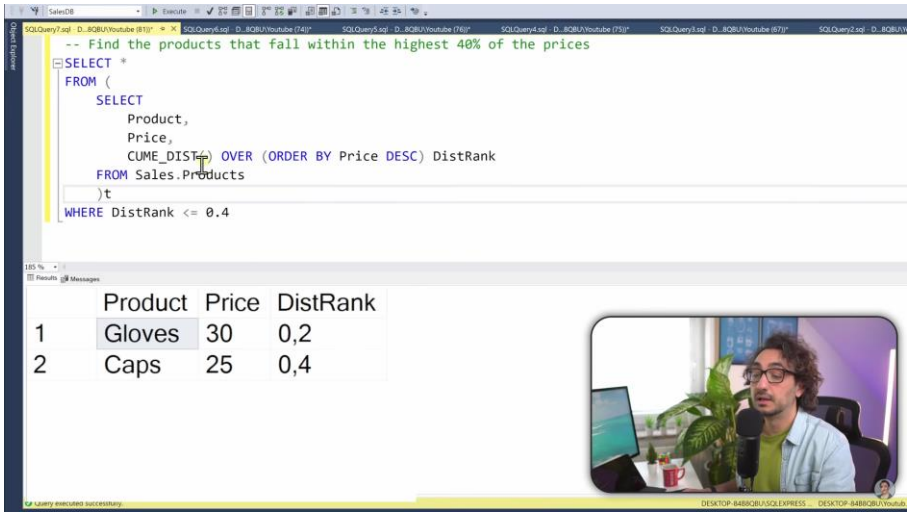
```
PERCENT_RANK() OVER (ORDER BY Sales DESC)
```

Sales	DIST	Per
100	0,2	0
80	0,6	0,25
80	0,6	0,25
50	0,8	0,75
30	1	1

$$\text{CUME\_DIST} = \frac{\text{Position Nr}}{\text{Number of Rows}}$$

$$\text{Percent\_Rank} = \frac{\text{Position Nr} - 1}{\text{Number of Rows} - 1}$$

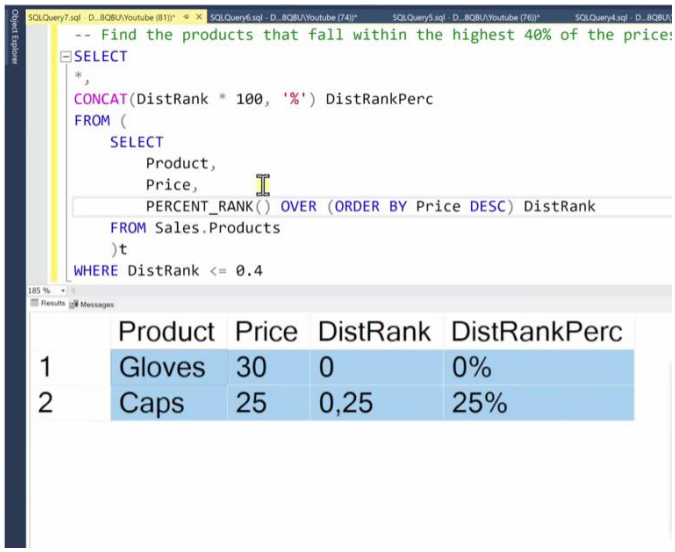




```
-- Find the products that fall within the highest 40% of the prices
SELECT *
FROM (
    SELECT
        Product,
        Price,
        CUME_DIST() OVER (ORDER BY Price DESC) DistRank
    FROM Sales.Products
    )t
WHERE DistRank <= 0.4
```

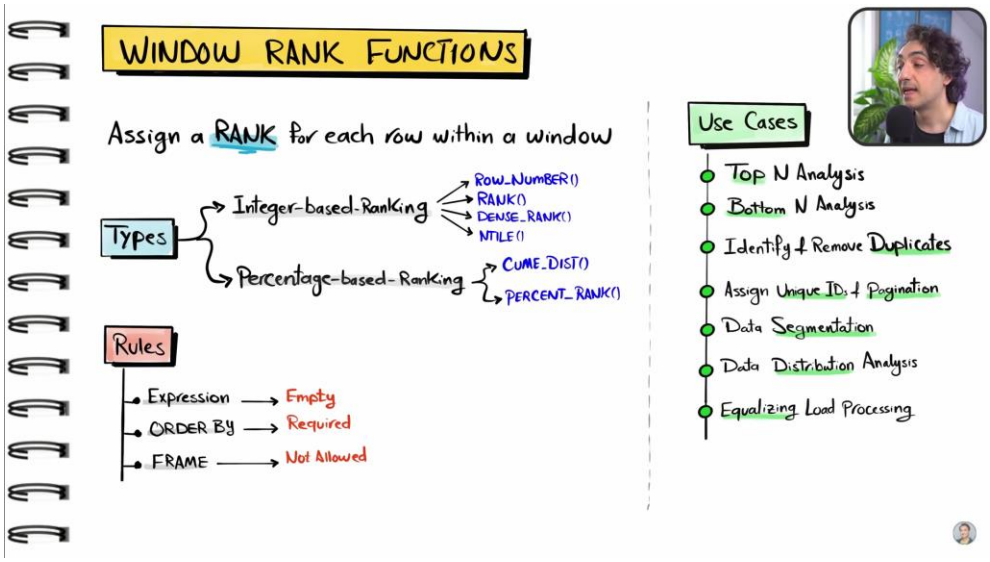
	Product	Price	DistRank
1	Gloves	30	0,2
2	Caps	25	0,4

BOTH ARE SAME ONE IS USING CUME_DIST AND OTHER IS USING PERCENT_RANK.

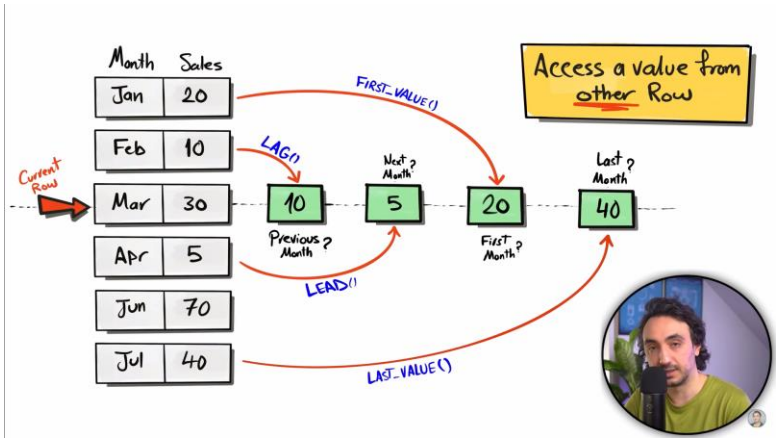


```
-- Find the products that fall within the highest 40% of the prices
SELECT
*,
CONCAT(DistRank * 100, '%') DistRankPerc
FROM (
    SELECT
        Product,
        Price,
        PERCENT_RANK() OVER (ORDER BY Price DESC) DistRank
    FROM Sales.Products
    )t
WHERE DistRank <= 0.4
```

	Product	Price	DistRank	DistRankPerc
1	Gloves	30	0	0%
2	Caps	25	0,25	25%



VALUE WINDOW FUNCTIONS:



</> Value Functions | Syntax

		Expression	Partition Clause	Order Clause	Frame Clause
Value (Analytics) Functions	**LEAD** (*expr,offset,default*)	All Data Type	Optional	Required	Not allowed
	LAG (*expr,offset,default*)				Optional
	FIRST_VALUE (*expr*)				Should be used
	LAST_VALUE (*expr*)				

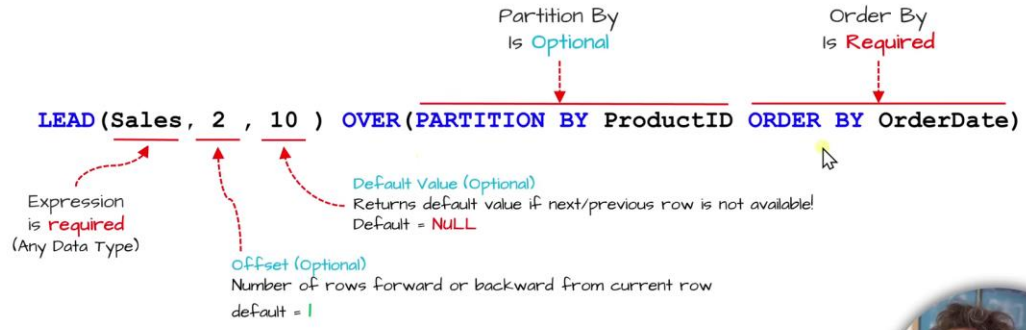
LEAD ( )

Access a value from the next row within a window.

LAG ( )

Access a value from the previous row within a window.

</> LEAD & LAG | Syntax



LEAD(Sales) OVER (ORDER BY Month)

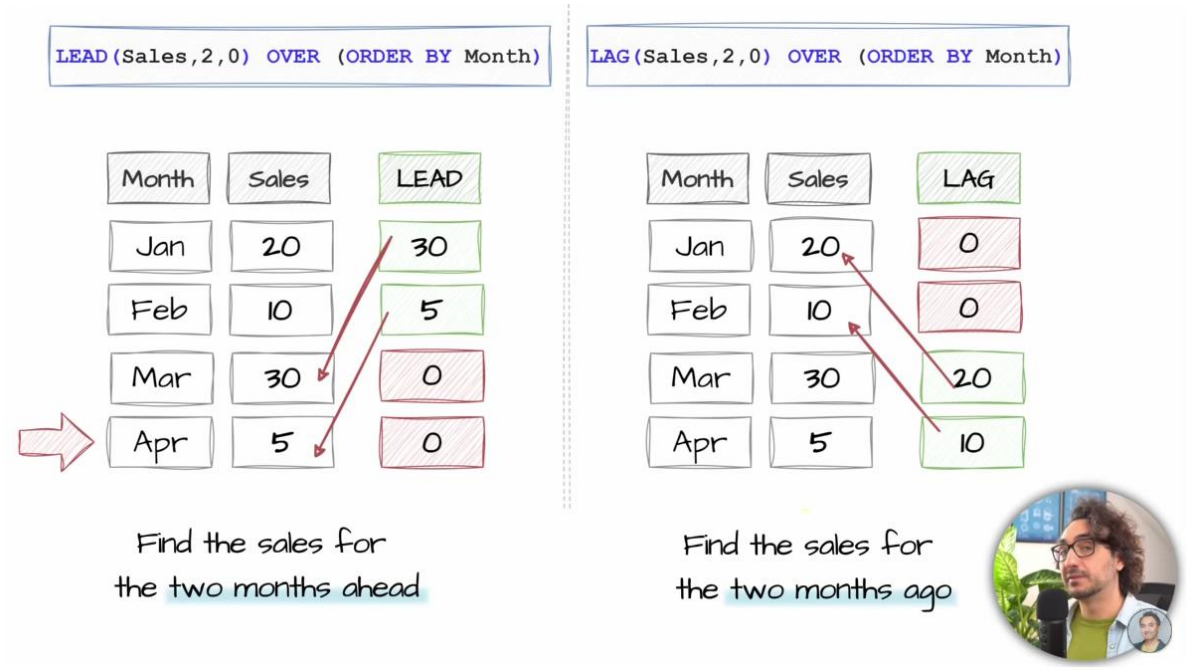
Month	Sales	LEAD
Jan	20	10
Feb	10	30
Mar	30	5
Apr	5	NULL

Find Sales of the next month

LAG(Sales) OVER (ORDER BY Month)

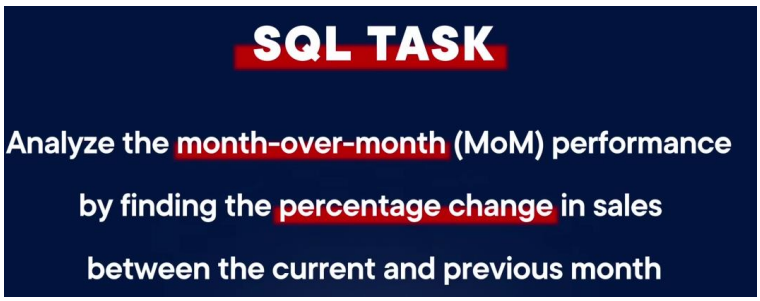
Month	Sales	LAG
Jan	20	NULL
Feb	10	20
Mar	30	10
Apr	5	30

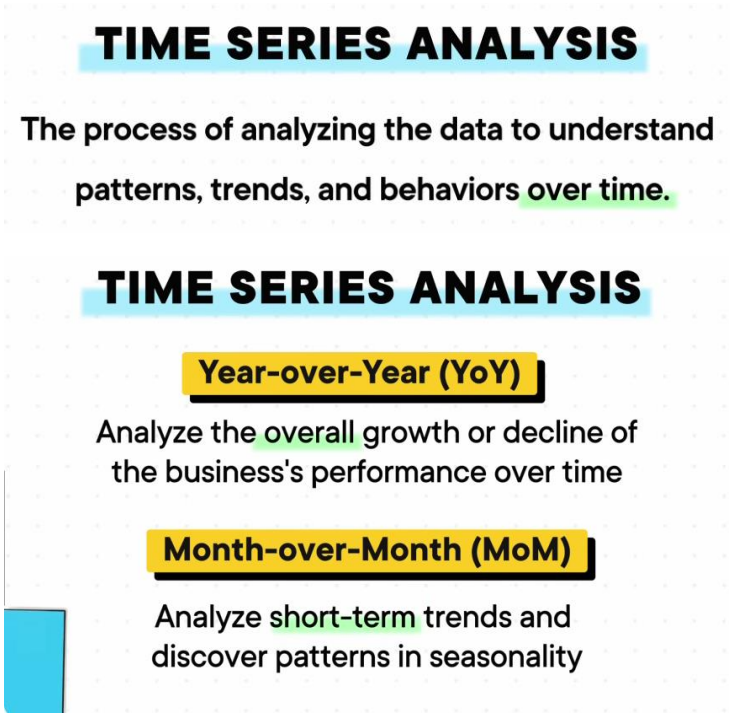
Find Sales of the previous month



LEAD/LAG USECASE:

MoM (Month – over – Month Analysis):



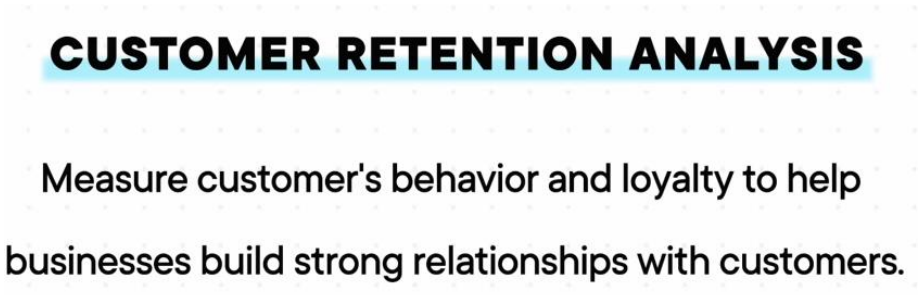


```sql
-- Analyze the month-over-month performance by finding the percentage change
-- in sales between the current and previous months
SELECT
*,
CurrentMonthSales - PreviousMonthSales AS MoM_Change,
ROUND(CAST((CurrentMonthSales - PreviousMonthSales) AS FLOAT)/PreviousMonthSales *100 , 1) AS MoM_Perc
FROM (
SELECT
    MONTH(OrderDate) OrderMonth,
    SUM(Sales) CurrentMonthSales,
    LAG(SUM(Sales)) OVER (ORDER BY MONTH(OrderDate)) PreviousMonthSales
FROM Sales.Orders
GROUP BY
    MONTH(OrderDate)
)t
```

	OrderMonth	CurrentMonthSales	PreviousMonthSales	MoM_Change	MoM_Perc
1	1	105	NULL	NULL	NULL
2	2	195	105	90	85,7
3	3	80	195	-115	-59

LEAD/LAG USE CASE:

CUSTOMER RETENTION ANALYSIS:




```sql
-- In order to analyze customer loyalty,
-- rank customers based on the average days between their orders
SELECT
CustomerID,
AVG(DaysUntilNextOrder) AvgDays,
RANK() OVER (ORDER BY COALESCE(AVG(DaysUntilNextOrder), 999999)) RankAvg
FROM (
    SELECT
    OrderID,
    CustomerID,
    OrderDate CurrentOrder,
    LEAD(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate) NextOrder,
    DATEDIFF(day,OrderDate, LEAD(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate)) DaysUntilNextOrder
    FROM Sales.Orders
)t
GROUP BY
   CustomerID
```

CustomerID	AvgDays	RankAvg
1	18	1
2	34	2
3	34	2
4	NULL	4

FIRST_VALUE()

Access a value from the first row within a window.

LAST_VALUE()

Access a value from the last row within a window.

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

Month	Sales	Last
Jan	20	5
Feb	10	5
Mar	30	5
Apr	5	5

Current Row — UNBOUNDED FOLLOWING

```
FIRST_VALUE(Sales) OVER (ORDER BY Month)
```

Month	Sales	First
Jan	20	20
Feb	10	20
Mar	30	20
Apr	5	20

UNBOUNDED PRECEDING — Current Row

```
LAST_VALUE(Sales) OVER (ORDER BY Month)
```

Month	Sales	Last
Jan	20	20
Feb	10	10
Mar	30	30
Apr	5	5

UNBOUNDED PRECEDING — Current Row

Default

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

SQL TASK

Find the lowest and highest sales for each product

```sql
-- Find the lowest and highest sales for each product
SELECT
    OrderID,
    ProductID,
    Sales,
    FIRST_VALUE(Sales) OVER (PARTITION BY ProductID ORDER BY S
    LAST_VALUE(Sales)  OVER (PARTITION BY ProductID ORDER BY S
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) HighestS
    FIRST_VALUE(Sales) OVER (PARTITION BY ProductID ORDER BY S
    MIN(Sales) OVER (PARTITION BY ProductID) LowestSales2,
    MAX(Sales) OVER (PARTITION BY ProductID) HighestSales3
FROM Sales.Orders
```

USE CASE

Compare to Extermes

How well a value is performing relative to the extremes

	OrderID	ProductID	Sales	LowestSales	HighestSales	HighestSales2	LowestSales2	HighestSales3
1	8	101	90	10	90	90	10	90
2	3	101	20	10	90	90	10	90
3	9	101	20	10	90	90	10	90
4	1	101	10	10	90	90	10	90
5	10	102	60	15	60	60	15	60
6	7	102	30	15	60	60	15	60
7	2	102	15	15	60	60	15	60
8	6	104	50	25	50	50	25	50
9	5	104	25	25	50	50	25	50
10	4	105	60	60	60	60	60	60

WINDOW VALUE (ANALYTICAL) FUNCTIONS

Allow Access specific Value from another Row

Types
- Previous Value — LAG()
- Next Value — LEAD()
- First Value — FIRST_VALUE()
- Last Value — LAST_VALUE()

Rules
- Expression — Any Data Type
- ORDER BY — Required
- FRAME — Optional

Use Cases
- Time Series Analysis: MoM & YoY
- Time Gaps Analysis: Customer Retention
- Comparision Analysis: Extreme → Highest / Lowest