

NULL FUNCTIONS:

ISNULL ():

Replaces NULL with specified values.

COALESCE ():

Returns the first non-null value from the list.

<u>ISNULL</u>	<u>COALESCE</u>
Limited to two values	Unlimited
Fast	Slow
SQL Server → ISNULL Oracle → NVL MySQL → IFNULL	Available in All Databases

USE CASE – HANDLING NULLS

DATA AGGREGATION:

ISNULL | COALESCE

- USE CASE -

Handle the NULL before doing data aggregations

```

SELECT
CustomerID,
Score,
AVG(Score) OVER() Wrong_Avg_Score,
--AVG function in SQL ignores NULL value, The sum will be divided by 4 instead of 5
AVG(COALESCE(Score,0)) OVER() Correct_Avg_Score
FROM Sales.Customers

```

	CustomerID	Score	Wrong_Avg_Score	Correct_Avg_Score
1	1	350	625	500
2	2	900	625	500
3	3	750	625	500
4	4	500	625	500
5	5	NULL	625	500

MATHEMATICAL OPERATIONS:

ISNULL | COALESCE

- USE CASE -

Handle the NULL before doing mathematical operations

SQL TASK

Display the full name of customers in a single field
by merging their first and last names,
and add 10 bonus points to each customer's score.

```

SELECT
CustomerID,
FirstName,
LastName,
FirstName + ' ' + COALESCE(LastName, '') AS Full_Name,
COALESCE(Score,0) + 10 AS Bonus_Score
FROM Sales.Customers

```

	CustomerID	FirstName	LastName	Full_Name	Bonus_Score
1	1	Jossef	Goldberg	Jossef Goldberg	360
2	2	Kevin	Brown	Kevin Brown	910
3	3	Mary	NULL	Mary	760
4	4	Mark	Schwarz	Mark Schwarz	510
5	5	Anna	Adams	Anna Adams	10

HANDLING NULLS

JOINS:

ISNULL | COALESCE

- USE CASE -

Handle the NULL before JOINING tables

Table1			Table2		
Year	Type	Orders	Year	Type	Sales
2024	a	30	2024	a	100
2024		40	2024		200
2025	b	50	2025	b	300
2025		60	2025		200


```

SELECT
  a.year, a.type, a.orders, b.sales
FROM Table1 a
JOIN Table2 b
  ON   a.year = b.year
  AND  ISNULL (a.type, '') = ISNULL (b.type, '')

```


Result			
Year	Type	Orders	Sales
2024	a	30	100
2024	NULL	40	200
2025	b	50	300
2025	NULL	60	200



HANDLING NULLS:

SORTING DATA

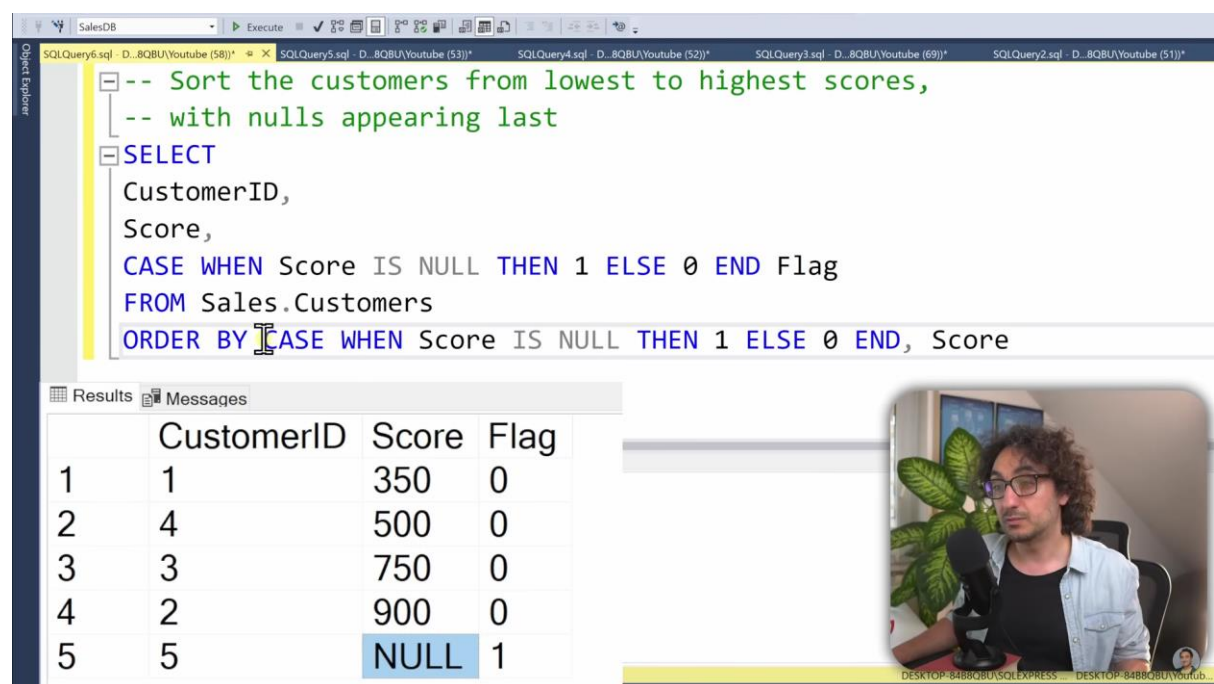
ISNULL | COALESCE

- USE CASE -

Handle the NULL before sorting data

SQL TASK

Sort the customers from lowest to highest scores,
with NULLs appearing last.



The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results pane. The query editor contains the following SQL code:

```
-- Sort the customers from lowest to highest scores,  
-- with nulls appearing last  
SELECT  
    CustomerID,  
    Score,  
    CASE WHEN Score IS NULL THEN 1 ELSE 0 END Flag  
FROM Sales.Customers  
ORDER BY CASE WHEN Score IS NULL THEN 1 ELSE 0 END, Score
```

The results pane displays the following data:

	CustomerID	Score	Flag
1	1	350	0
2	4	500	0
3	3	750	0
4	2	900	0
5	5	NULL	1

In the bottom right corner, there is a small video inset showing a person with curly hair and glasses, wearing a light blue shirt, sitting at a desk with a microphone and a potted plant.

NULLIF ():

Compares two expressions returns:

-NULL if they are equal.

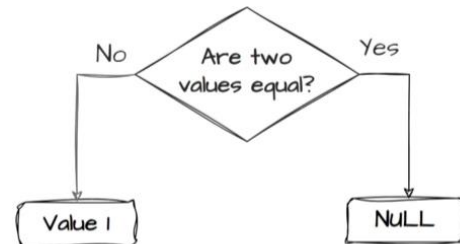
-First Value, if they are not equal.

SYNTAX

NULLIF(value1, value2)

NULLIF(Original_Price, Discount_Price)

OrderID	Original_Price	Discount_Price	NULLIF
1	150	50	150
2	250	250	NULL



NULLIF USE CASES:

DIVISION BY ZERO

SQL TASK

Find the sales price for each order by dividing the sales by the quantity.

The screenshot shows a SQL query in a query editor and its results in a table. The query is:

```
-- Find the sales price for each order by dividing sales by quantity
SELECT
  OrderID,
  Sales,
  Quantity,
  Sales / NULLIF(Quantity, 0) AS Price
FROM Sales.Orders
```

The results table shows the following data:

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

A video inset shows a man with glasses and curly hair speaking into a microphone.

IS NULL AND IS NOT NULL:

IS NULL:

Return true if the value is null. If not null returns false.

IS NOT NULL:

Return true if the value is not null. If null returns false.

Syntax

Value **IS NULL**

Value **IS NOT NULL**

Example

Shipping_Address **IS NULL**

Example

Shipping_Address **IS NOT NULL**

IS NULL USECASE:

FILTERING DATA.

IS NULL | IS NOT NULL

- USE CASE -

Searching for missing information

The screenshot shows the SQL Server Enterprise Manager interface. The 'Object Explorer' on the left shows the 'Sales' database. The 'SQL Query' window displays the following query:

```
-- Identify the customers who have no scores
SELECT
*
FROM Sales.Customers
WHERE Score IS NULL
```

The 'Results' pane at the bottom shows the output of the query:

	CustomerID	FirstName	LastName	Country	Score
1	5	Anna	Adams	USA	NULL

IS NULL USE CASES:

ANTI-JOINS.

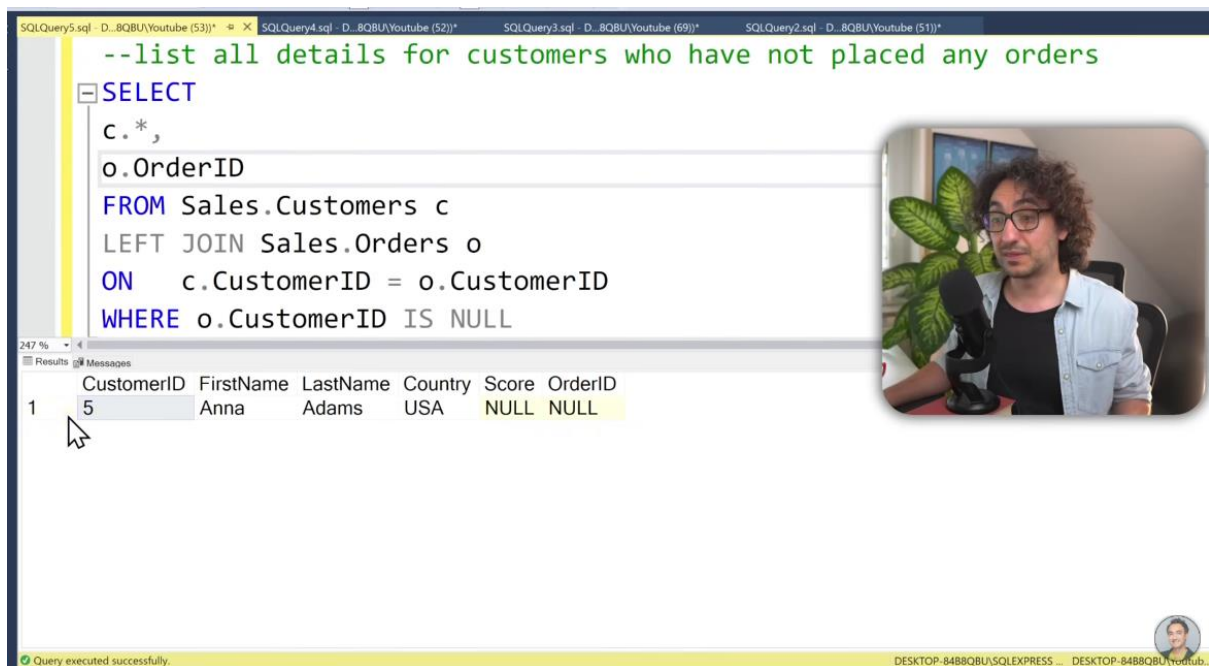
LEFT ANTI JOIN | RIGHT ANTI JOIN

- USE CASE -

Finding the **unmatched rows** between two tables

SQL TASK

List all details for customers who have **not placed** any orders



```
--list all details for customers who have not placed any orders
SELECT
  c.*,
  o.OrderID
FROM Sales.Customers c
LEFT JOIN Sales.Orders o
ON c.CustomerID = o.CustomerID
WHERE o.CustomerID IS NULL
```

	CustomerID	FirstName	LastName	Country	Score	OrderID
1	5	Anna	Adams	USA	NULL	NULL

Query executed successfully.

DESKTOP-84B8QBU\SQLEXPRESS ... DESKTOP-84B8QBU\Youtub...

NULL vs EMPTY vs SPACE.


SQLQuery3.sql - D:\QBUI\Youtube (74))

```

WITH Orders AS (
  SELECT 1 Id, 'A' Category UNION
  SELECT 2, NULL UNION
  SELECT 3, '' UNION
  SELECT 4, ' '
)
SELECT
  *,
  DATALENGTH (Category) CategoryLen
FROM Orders
  
```

Results Messages

	Id	Category	CategoryLen
1	1	A	1
2	2	NULL	0
3	3		0
4	4		1

	<u>NULL</u>	<u>Empty String</u>	<u>Blank Space</u>
Representation	NULL	''	' '
Meaning	Unknown	Known, Empty Value	Known, Space Value
Data Type	Special Marker	String (0)	String (1 or more)
Storage	Very minimal	occupies memory	occupies memory (each space)
Performance	Best	Fast	Slow
Comparison	IS NULL	= ''	= ' '

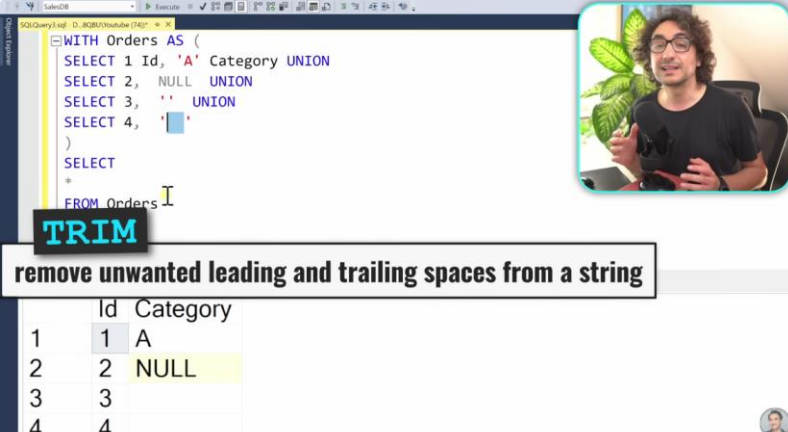
HANDLING NULLS:

DATA POLICIES.



#1 DATA POLICY

Only use **NULLs** and **empty strings**,
but avoid **blank spaces**.



The screenshot shows a SQL query window with the following CTE and query:

```
WITH Orders AS (  
  SELECT 1 Id, 'A' Category UNION  
  SELECT 2, NULL UNION  
  SELECT 3, '' UNION  
  SELECT 4, ' '  
)  
SELECT  
  *  
FROM Orders
```

A red box highlights the word **TRIM**. Below the query window, a text box states: **remove unwanted leading and trailing spaces from a string**.

	Id	Category
1	1	A
2	2	NULL
3	3	
4	4	



The screenshot shows a SQL query window with the following CTE and query:

```
WITH Orders AS (  
  SELECT 1 Id, 'A' Category UNION  
  SELECT 2, NULL UNION  
  SELECT 3, '' UNION  
  SELECT 4, ' '  
)  
SELECT  
  *,  
  DATALENGTH(Category) CategoryLen,  
  DATALENGTH(TRIM(Category)) Policy1  
FROM Orders
```

The results table shows the output of the query:

	Id	Category	CategoryLen	Policy1
1	1	A	1	1
2	2	NULL	NULL	NULL
3	3		0	0
4	4		2	0



#2 DATA POLICY

Only use **NULLS** and
avoid using **empty strings** and **blank spaces**

SQLQuery3.sql - D:\8QBU\Youtube (74))

```

WITH Orders AS (
  SELECT 1 Id, 'A' Category UNION
  SELECT 2, NULL UNION
  SELECT 3, '' UNION
  SELECT 4, ' '
)
SELECT
  *,
  TRIM(Category) Policy1,
  NULLIF(TRIM(Category), '') Policy2
FROM Orders

```

Results Messages

	Id	Category	Policy1	Policy2
1	1	A	A	A
2	2	NULL	NULL	NULL
3	3			NULL
4	4			NULL

Query executed successfully.



#3 DATA POLICY

Use the **default value 'unknown'** and avoid using nulls, empty strings, and blank spaces.

SQLQuery3.sql - D:\8QBU\Youtube (74))

```

WITH Orders AS (
  SELECT 1 Id, 'A' Category UNION
  SELECT 2, NULL UNION
  SELECT 3, '' UNION
  SELECT 4, ' '
)
SELECT
  *,
  TRIM(Category) Policy1,
  NULLIF(TRIM(Category), '') Policy2,
  COALESCE(NULLIF(TRIM(Category), ''), 'unknown') Policy3
FROM Orders

```

Results Messages

	Id	Category	Policy1	Policy2	Policy3
1	1	A	A	A	A
2	2	NULL	NULL	NULL	unknown
3	3			NULL	unknown
4	4			NULL	unknown

#2 DATA POLICY

- USE CASE -



Replacing empty strings and blanks with NULL
during data preparation before inserting into a database
to optimize storage and performance.

#3 DATA POLICY

- USE CASE -

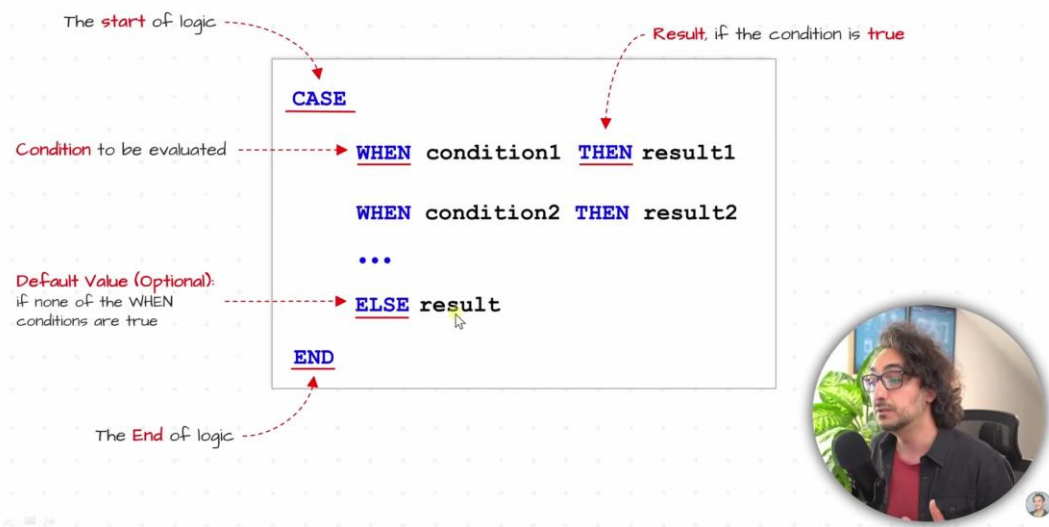


Replacing empty strings, blanks, NULL with default value
during data preparation before using it in reporting
to improve readability and reduce confusion

CASE STATEMENT:

CASE STATEMENT

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



USE CASE:

CATEGORIZING DATA.

Main purpose is Data Transformation

Derive new information

- Create new Columns based on existing data -

CATEGORIZING DATA

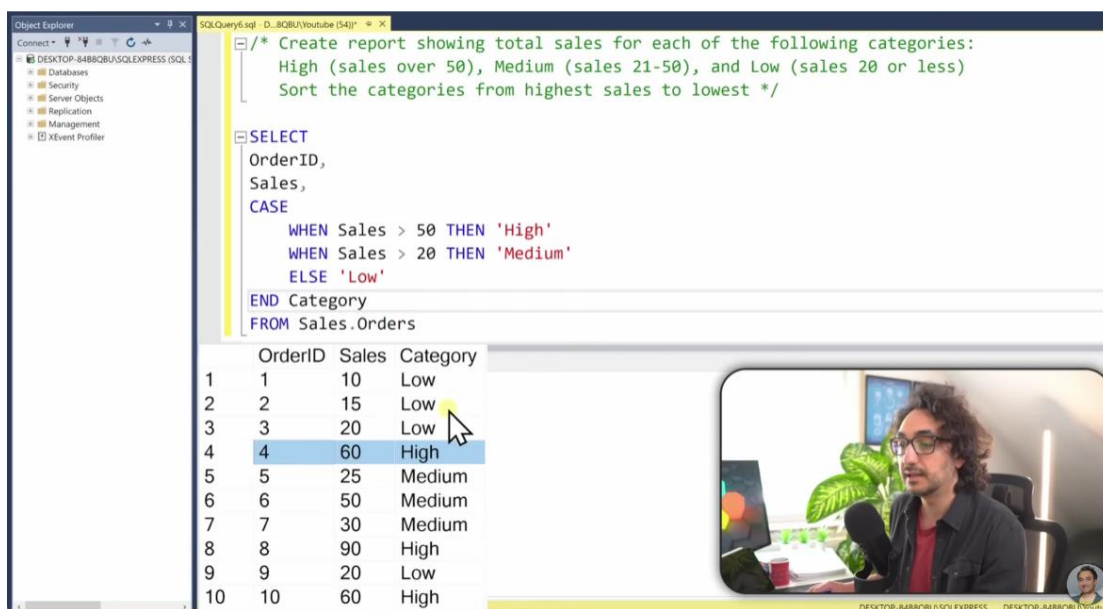
Group the data into different categories based on certain conditions.

SQL TASK

Generate a report showing the total sales for each category:

- High: If the sales higher than 50
- Medium: If the sales between 20 and 50
- Low: If the sales equal or lower than 20

Sort the result from lowest to highest.



The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the Object Explorer with the 'DESKTOP-8488QBU\SQLEXPRESS (SQL)' server selected. The right pane shows a SQL query window with the following text:

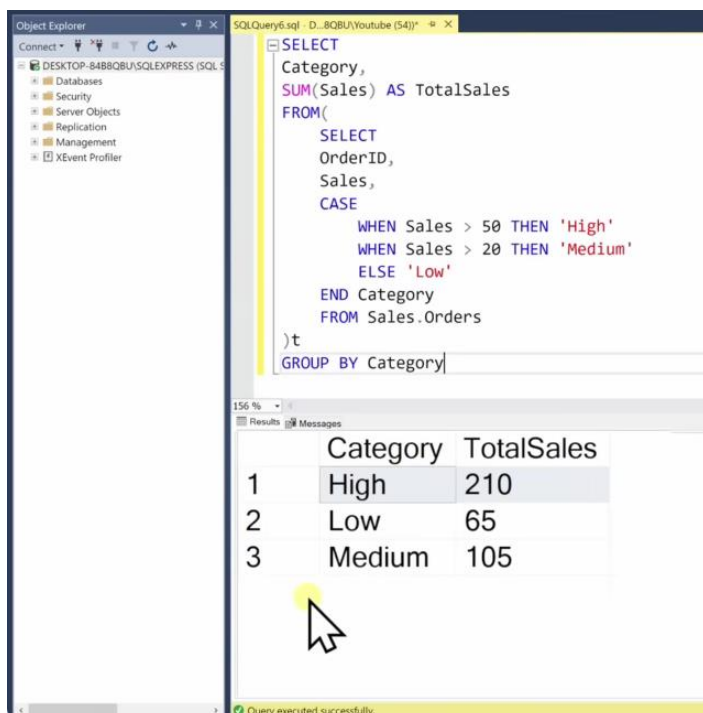
```
/* Create report showing total sales for each of the following categories:
High (sales over 50), Medium (sales 21-50), and Low (sales 20 or less)
Sort the categories from highest sales to lowest */

SELECT
OrderID,
Sales,
CASE
WHEN Sales > 50 THEN 'High'
WHEN Sales > 20 THEN 'Medium'
ELSE 'Low'
END Category
FROM Sales.Orders
```

Below the query, the results are displayed in a table:

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

A small video inset in the bottom right corner shows a person with glasses and a beard, likely the presenter, sitting at a desk with a microphone and a plant.



The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the Object Explorer with the 'DESKTOP-8488QBU\SQLEXPRESS (SQL)' server selected. The right pane shows a SQL query window with the following text:

```
SELECT
Category,
SUM(Sales) AS TotalSales
FROM(
SELECT
OrderID,
Sales,
CASE
WHEN Sales > 50 THEN 'High'
WHEN Sales > 20 THEN 'Medium'
ELSE 'Low'
END Category
FROM Sales.Orders
)t
GROUP BY Category
```

Below the query, the results are displayed in a table:

	Category	TotalSales
1	High	210
2	Low	65
3	Medium	105

A mouse cursor is visible over the 'Low' category row in the results table.


```
SQLQuery6.sql - D:\8QBU\Youtube (54)) * X
/* Create report showing total sales for each of the following categories:
   High (sales over 50), Medium (sales 20-50), and Low (sales 20 or less)
   Sort the categories from highest sales to lowest */
SELECT
  Category,
  SUM(Sales) AS TotalSales
FROM(
  SELECT
    OrderID,
    Sales,
    CASE
      WHEN Sales > 50 THEN 'High'
      WHEN Sales > 20 THEN 'Medium'
      ELSE 'Low'
    END Category
  FROM Sales.Orders
)t
GROUP BY Category
ORDER BY TotalSales DESC
```

MAPPING:

SQL TASK

Retrieve employee details with gender displayed as full text

Object Explorer

Connect +

- DESKTOP-8488QBU\SQLEXPRESS (SQL S
- Databases
- Security
- Server Objects
- Replication
- Management
- XEvent Profiler

SQLQuery2.sql - D:\8QBU\Youtube (77)) * X

```
-- Retrive employee details with gender displayed as full text
SELECT
  EmployeeID,
  FirstName,
  LastName,
  Gender,
  CASE
    WHEN Gender = 'F' THEN 'Female'
    WHEN Gender = 'M' THEN 'Male'
    ELSE 'Not Aavailable'
  END GenderFullText
FROM Sales.Employees
```

Results

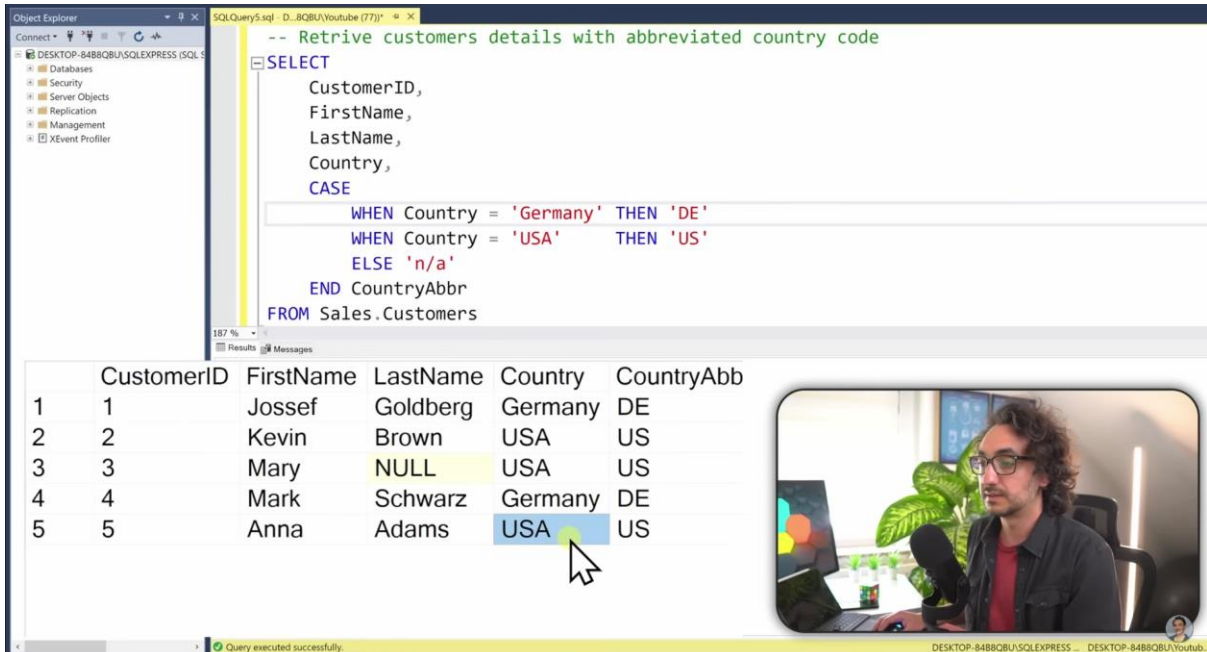
	EmployeeID	FirstName	LastName	Gender	GenderFullText
1	1	Frank	Lee	M	Male
2	2	Kevin	Brown	M	Male
3	3	Mary	NULL	F	Female
4	4	Michael	Ray	M	Male
5	5	Carol	Baker	F	Female

Query executed successfully

DESKTOP-8488QBU\SQLEXPRESS ... DESKTOP-8488QBU\Youtube...

SQL TASK

Retrieve customer details with abbreviated country code



The screenshot shows a SQL Server Enterprise Manager interface. The left pane displays the 'Object Explorer' with a tree view of the server structure. The right pane shows a query window with the following SQL code:

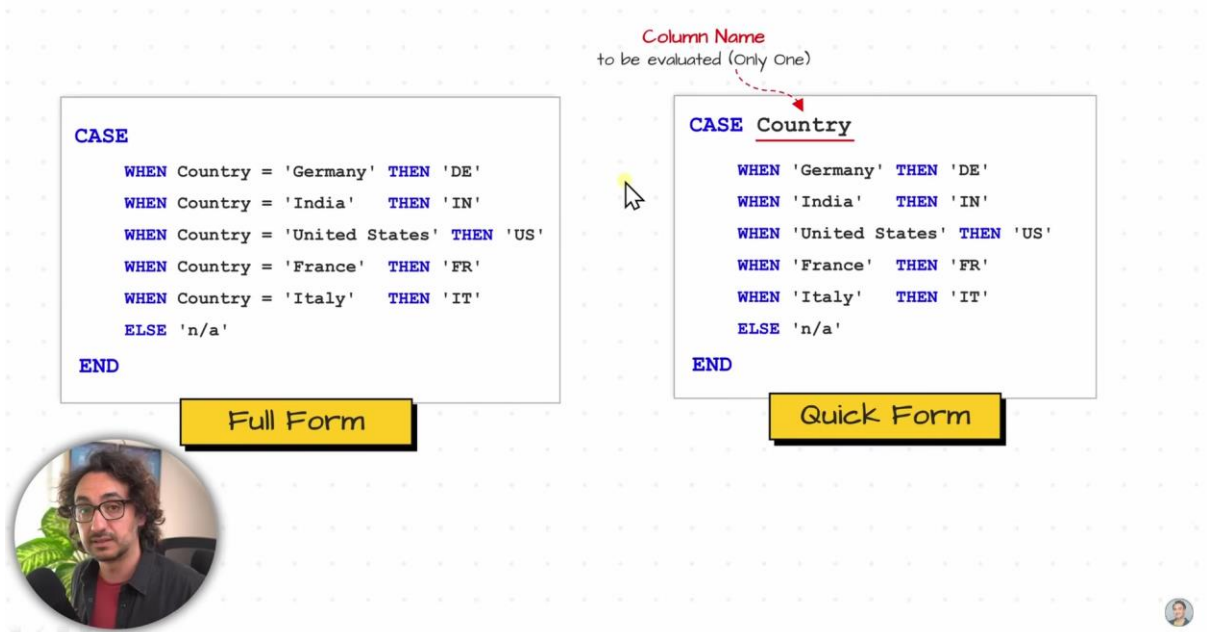
```
-- Retrieve customers details with abbreviated country code
SELECT
    CustomerID,
    FirstName,
    LastName,
    Country,
    CASE
        WHEN Country = 'Germany' THEN 'DE'
        WHEN Country = 'USA' THEN 'US'
        ELSE 'n/a'
    END CountryAbbr
FROM Sales.Customers
```

Below the query window, the results are displayed in a table:

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

A small inset video shows a person with glasses and a beard, wearing a dark shirt, sitting at a desk with a laptop and a microphone.

QUICK FORM:



The diagram compares two forms of a CASE statement. On the left is the 'Full Form', and on the right is the 'Quick Form'.

Full Form:

```
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
```

Quick 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
```

A red arrow points from the text 'Column Name to be evaluated (Only One)' to the 'Country' column name in the Quick Form.

A small inset video shows a person with glasses and a beard, wearing a dark shirt, sitting at a desk with a laptop and a microphone.

HANDLING NULLS:

HANDLING NULLS

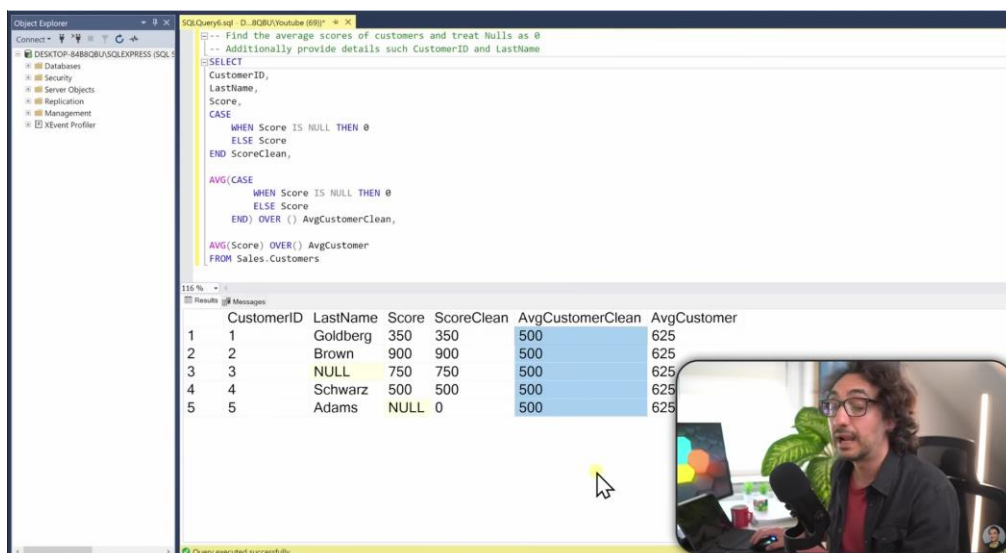
Replace NULLs with a specific value.

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

SQL TASK

Find the average scores of customers and **treat Nulls as 0**

And additionally provide details such CustomerID & LastName



The screenshot shows the SQL Server Enterprise interface. The query editor contains the following SQL code:

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

The results pane displays the following data:

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

A small video inset in the bottom right corner shows a person with glasses and a beard, likely the presenter, speaking into a microphone.

CONDITIONAL AGGRGATION:

CONDITIONAL AGGREGATION

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

SQL TASK

Count how many times each customer has made an order with **sales greater than 30**.

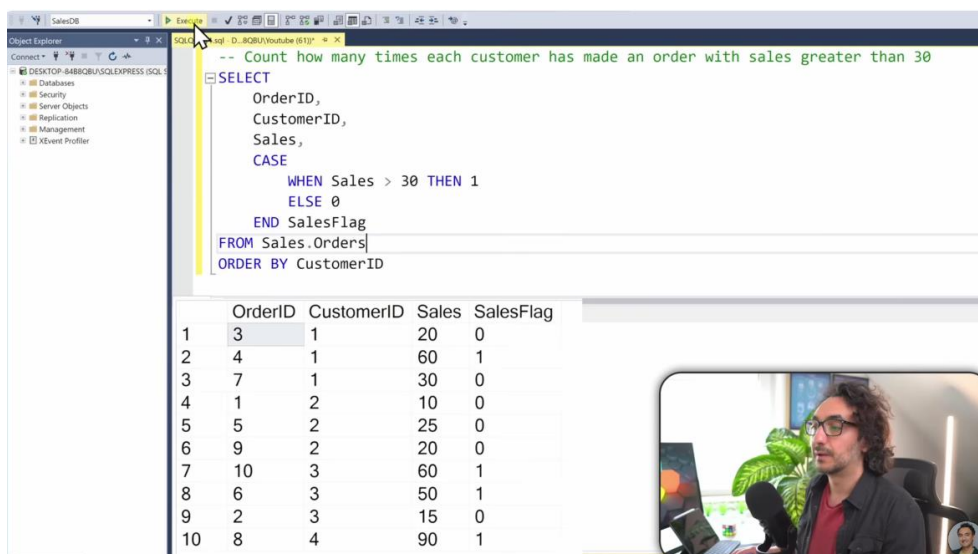
FLAG

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

STEP-1

Create Flag with binary values (0,1) to mark rows that meet certain criteria.

2	4	1	60
3	7	1	30
4	1	2	10



The screenshot shows the SQL Server Enterprise Manager interface. The 'Query' window displays the following SQL query:

```
-- 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
```

The results grid shows the following data:

	OrderID	CustomerID	Sales	SalesFlag
1	3	1	20	0
2	4	1	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

In the bottom right corner, there is a small video inset showing a man with glasses and a beard, wearing a dark jacket, sitting at a desk with a computer monitor and a potted plant.

STEP-2

Summarize the binary flag

ORDER BY CustomerID

SQLQuery1.sql: D:\BQBU\Youtube (611)*

```
-- 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

