

ISNULL

Limited to two values

Fast

SQL Server → ISNULL

Oracle → NVL

MySQL → IFNULL

COALESCE

Unlimited

Slow

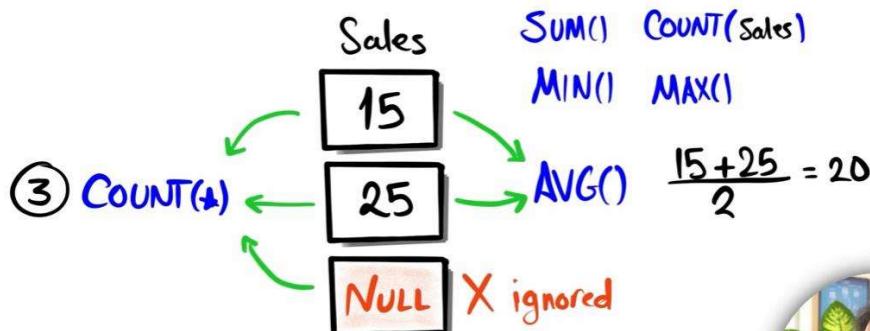
Available in All Databases

Usecase of data handling

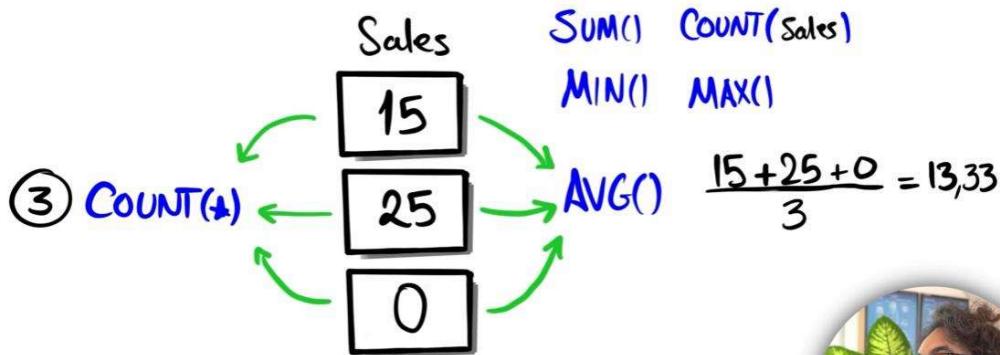
ISNULL | COALESCE

- USE CASE -

Handle the NULL before doing data aggregations



Replace Null to get accurate result



```
SQLQuery1.sql - D:\8Q8BU\Youtube (64) * 
-- Find the average scores of the customers
SELECT
    CustomerID,
    Score,
    COALESCE(Score,0) Score2,
    AVG(Score) OVER () AvgScores,
    AVG(COALESCE(Score,0)) OVER() AvgScores2
FROM Sales.Customers
```

	CustomerID	Score	Score2	AvgScores	AvgScores2
1	1	350	350	625	500
2	2	900	900	625	500
3	3	750	750	625	500
4	4	500	500	625	500
5	5	NULL	0	625	500

ISNULL | COALESCE

- USE CASE -

Handle the NULL before doing mathematical operations

$$1 + 5 \Rightarrow 6$$

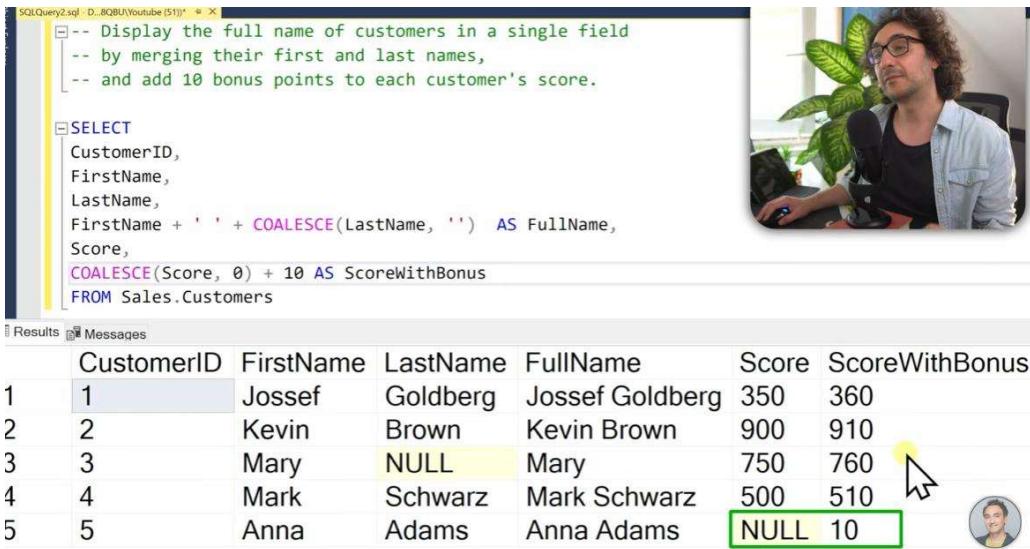
$$'A' + 'B' \Rightarrow 'AB'$$

$$0 + 5 \Rightarrow 5$$

$$'' + 'B' \Rightarrow 'B'$$

$$\text{NULL} + 5 \Rightarrow \text{NULL}$$

$$\text{NULL} + 'B' \Rightarrow \text{NULL}$$



```

SQLQuery2.sql - D:\8QBU\Youtube (51).sql * X
-- 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 FullName,
Score,
COALESCE(Score, 0) + 10 AS ScoreWithBonus
FROM Sales.Customers

```

Results Messages

	CustomerID	FirstName	LastName	FullName	Score	ScoreWithBonus
1	1	Jossef	Goldberg	Jossef Goldberg	350	360
2	2	Kevin	Brown	Kevin Brown	900	910
3	3	Mary	NULL	Mary	750	760
4	4	Mark	Schwarz	Mark Schwarz	500	510
5	5	Anna	Adams	Anna Adams	NULL	10

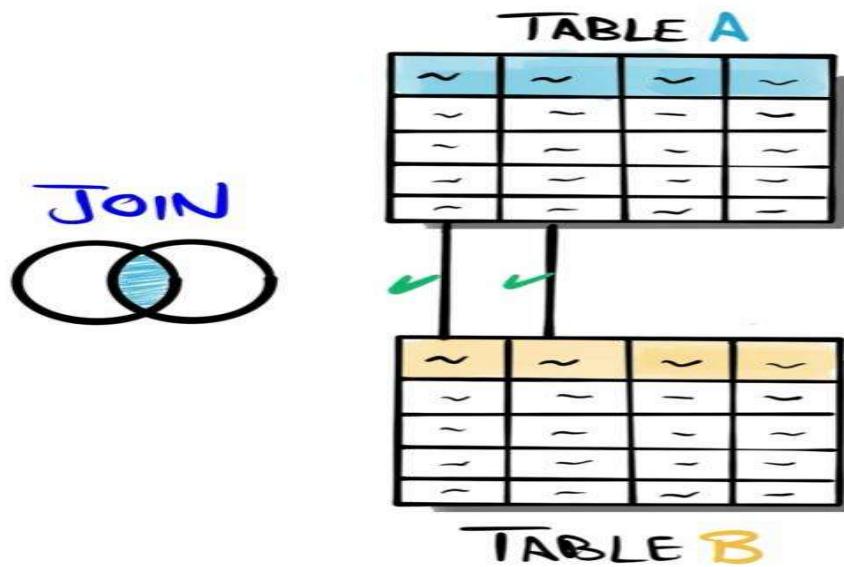
Handling Null Joins

ISNULL | COALESCE

- USE CASE -

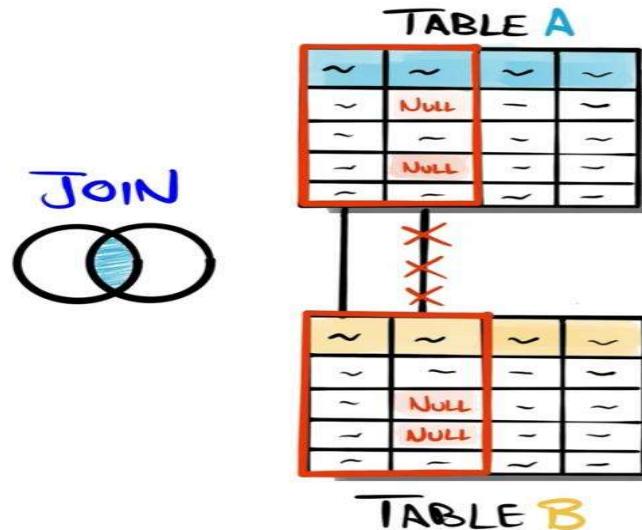
Handle the NULL before JOINING tables

Tbales joins perfectly when no null values in data

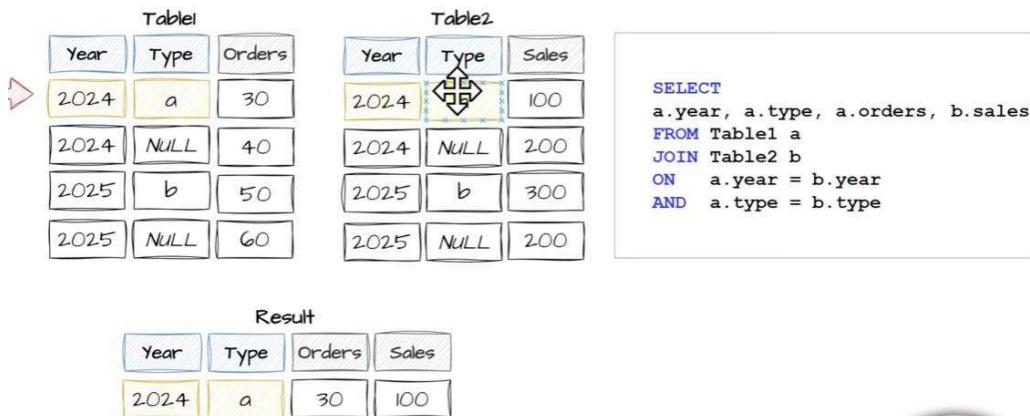


When we have null values in data we get inaccurate result by joining the table

So, Handle Null inside the keys before doing joins

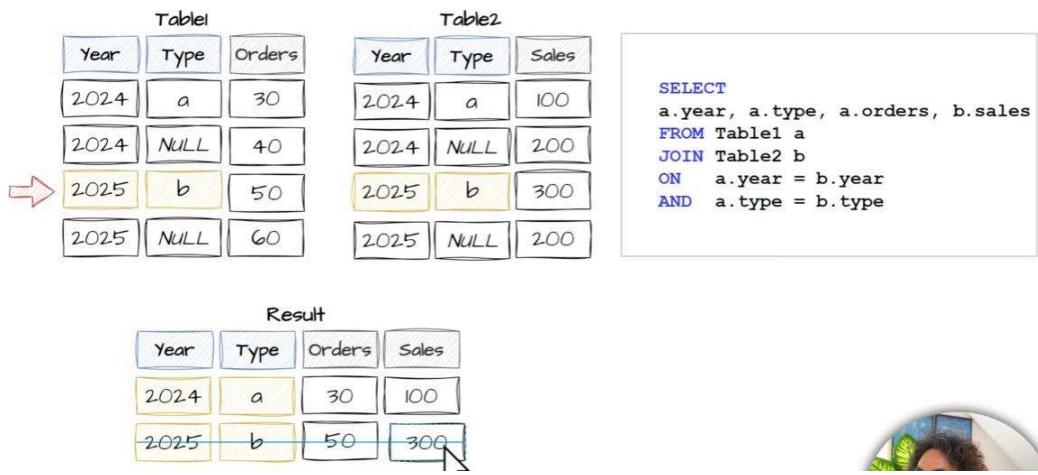
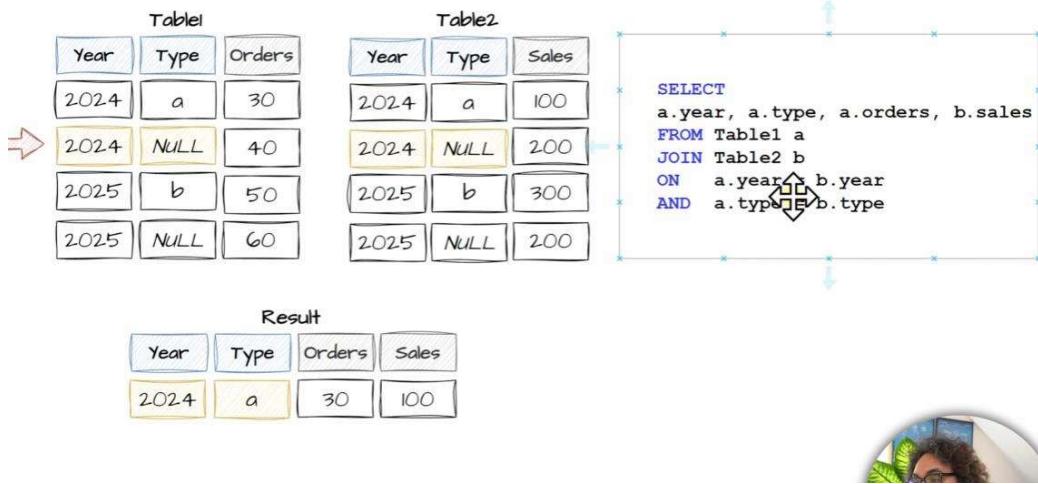


First row matching data add in result inner join add only matched data into the result



Here second example of row everything matching but sql not perform equal operator because of NULL values. SQL not compares Null so for this row combination SQL not found any result.

So as per business this is called missing information and accurate results.



Final Output -



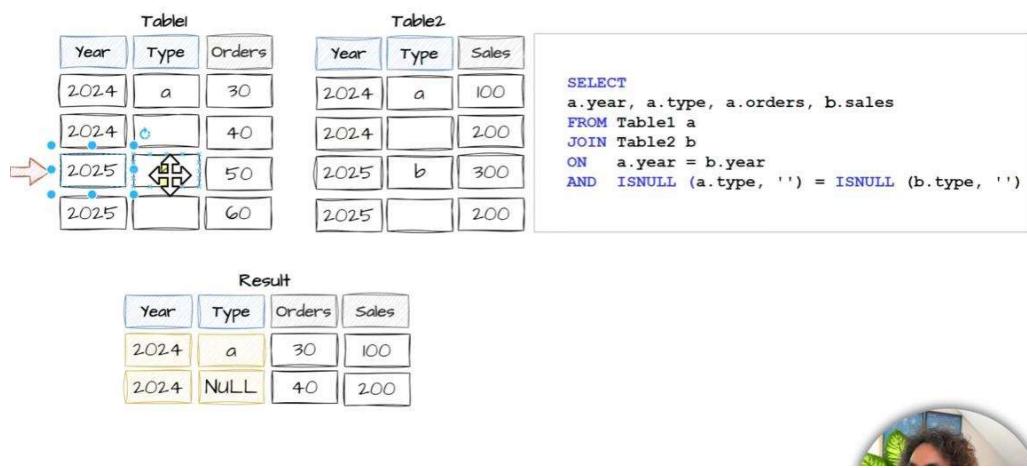
Above two tables were identical as we compare keys so we are losing data in the result and getting inaccurate result.

If we have null inside the data we have to handle those Null values before performing join Operation Otherwise we will got an inaccurate result.

We handle it by replacing the null value by an empty string, blank or any value to replace

Second row output we got Null as an output because we handling null values just to keep records so that we get an accurate data.

We are not changing anything in select or original data so as a result we got null at place of those values those not exist. And having value null in original data .



Accurate Result - we does not miss any value in our data

Table1

Year	Type	Orders
2024	a	30
2024		40
2025	b	50
2025		60

Table2

Year	Type	Sales
2024	a	100
2024		200
2025	b	300
2025		200

Result

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

```



Next Use case - Handle NULL value before sorting it

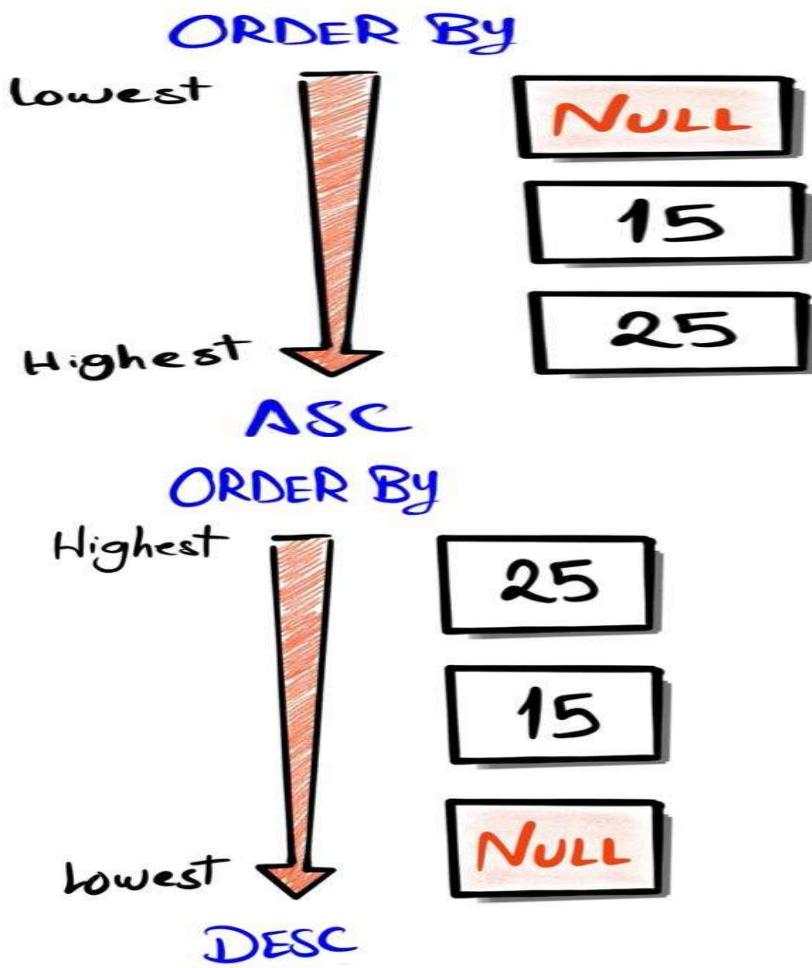
ISNULL | COALESCE

- USE CASE -

Handle the NULL before sorting data

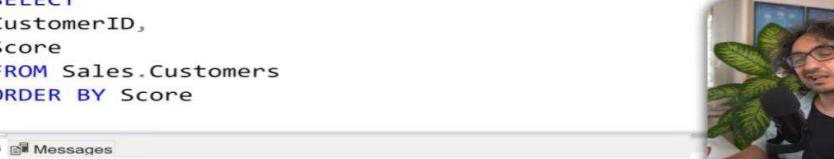


Values like this sql cosidered null as smallest value



-- Sort the customers from lowest to highest scores,
-- with nulls appearing last

```
SELECT  
CustomerID,  
Score  
FROM Sales.Customers  
ORDER BY Score
```



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

Handle Null Before Shorting the data

Object Explorer

```
-- Sort the customers from lowest to highest scores,
-- with nulls appearing last
SELECT
CustomerID,
Score,
COALESCE (Score, 9999999)
FROM Sales.Customers
ORDER BY COALESCE (Score, 9999999)
```

Results Messages

	CustomerID	Score	(No column name)
1	1	350	350
2	4	500	500
3	3	750	750
4	2	900	900
5	5	NULL	9999999

DESKTOP-84B8QBU\SQLEXPRESS...



Object Explorer

```
-- 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 COALESCE (Score, 9999999)
```

Results Messages

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

DESKTOP-84B8QBU\SQLEXPRESS...



```
-- 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
```

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

Coalesce and Is Null used to handle Null values before dealing with joins operations and shorting your data

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

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

NULLIF()

Compares two expressions returns:

- **NULL, if they are equal.**
- **First Value, if they are not equal.**

Syntax

```
NULLIF(value1, Value2)
```



Example

```
NULLIF(Shipping_Address, 'unknown')
```

Example

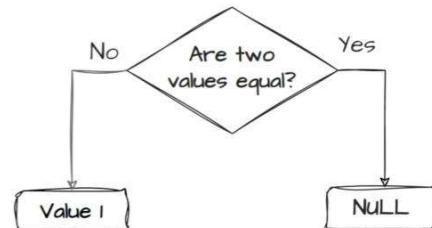
```
NULLIF(Shipping_Address, Billing_Address)
```

SYNTAX

```
NULLIF(value1, value2)
```

```
NULLIF(Price, -1)
```

OrderID	Price	NULLIF
1	90	90
2	-1	



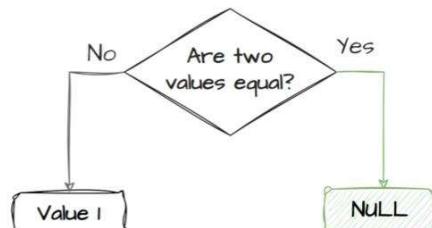
Here we are replacing a real value with null values

SYNTAX

```
NULLIF(value1, value2)
```

```
NULLIF(Price, -1)
```

OrderID	Price	NULLIF
1	90	90
2	-1	NULL



Case if both values are equal in Null IF function:



In this case again we get Null if two prices are same

NULL iF case when division by zero

NULLIF

- USE CASE -

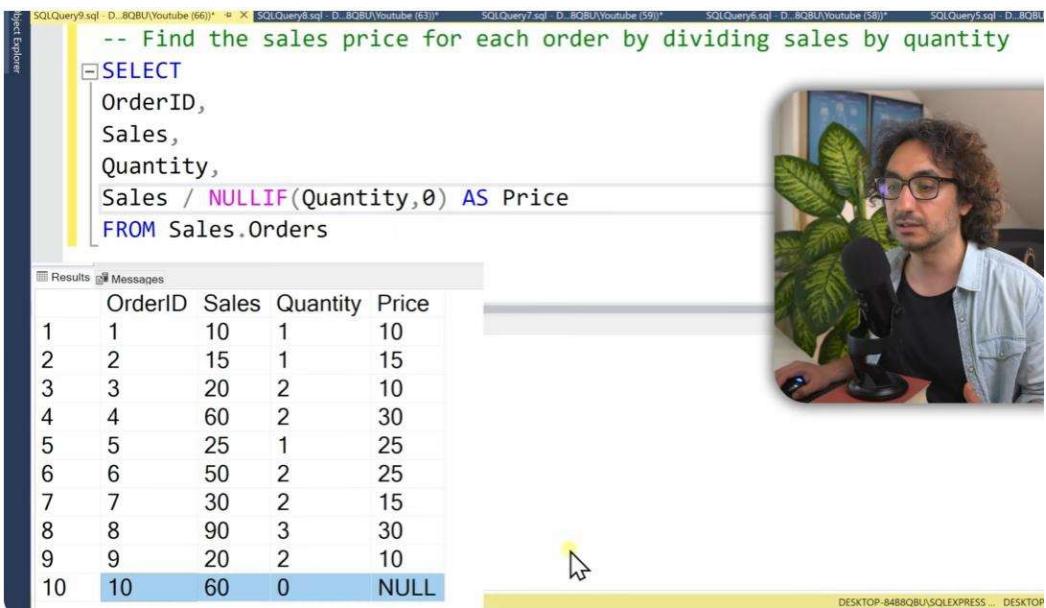
Preventing the error of dividing by zero

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

Msg 8134, Level 16, State 1, Line 2
Divide by zero error encountered.

Completion time: 2024-06-30T19:27:18.6584744+02:00





-- 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
```

	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



IS NULL

Returns **TRUE** if the value **IS NULL**,
otherwise it returns **FALSE**.

IS NOT NULL

Returns **TRUE** if the value **IS NOT NULL**,
otherwise it returns **FALSE**.

Syntax

Value IS NULL

Value IS NOT NULL

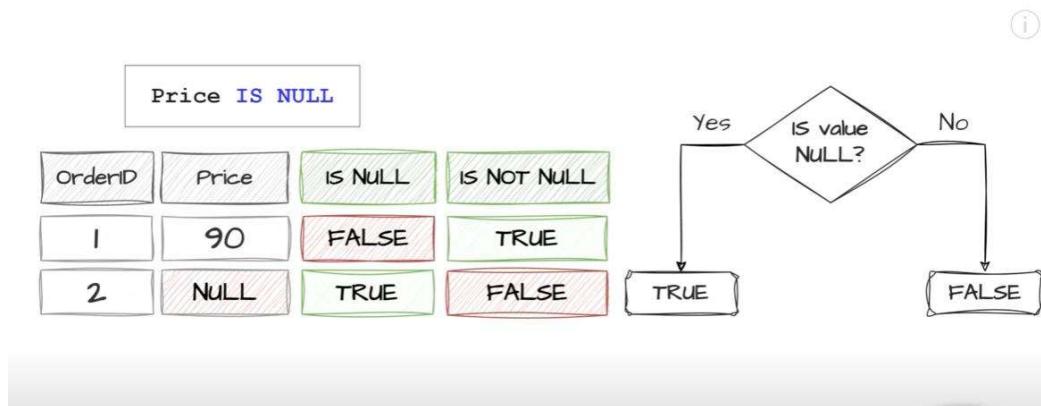
Example

Shipping_Address IS NULL

Example

Shipping_Address IS NOT NULL

How it works



Usecases : search for missing information

IS NULL | IS NOT NULL

- USE CASE -

Searching for **missing** information

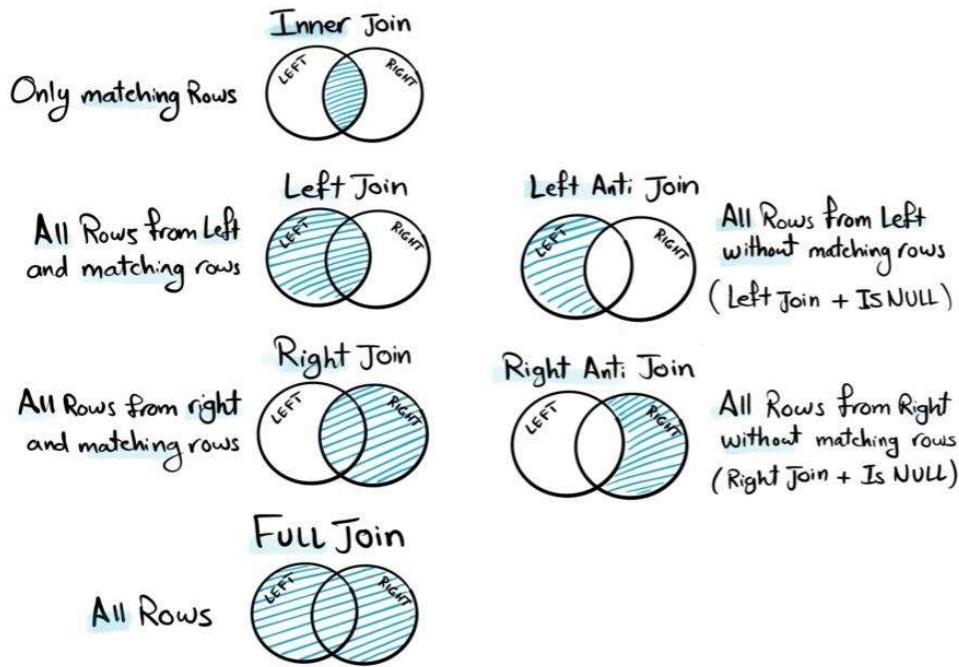
```
-- Identify the customers who have no scores
SELECT
*
FROM Sales.Customers
WHERE Score IS NULL
```

	CustomerID	FirstName	LastName	Country	Score
1	5	Anna	Adams	USA	NULL

```
-- List all customers who have scores
SELECT
*
FROM Sales.Customers
WHERE Score IS NOT NULL
```

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

IS Null Use case -- Anti Join



```
--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
```

CustomerID	FirstName	LastName	Country	Score	OrderID
1	Jossef	Goldberg	Germany	350	4
1	Jossef	Goldberg	Germany	350	7
2	Kevin	Brown	USA	900	1
2	Kevin	Brown	USA	900	5
2	Kevin	Brown	USA	900	9
3	Mary	NULL	USA	750	2
3	Mary	NULL	USA	750	6
3	Mary	NULL	USA	750	10
4	Mark	Schwarz	Germany	500	8
5	Anna	Adams	USA	NULL	NULL



LEFT ANTI JOIN

All rows from the left table without matches in the right table

```
--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
5	Anna	Adams	USA	NULL	NULL



NULL

means nothing, unknown!

EMPTY STRING ''

String value has zero characters

BLANK SPACE ' '

String value has one or more space characters

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

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


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

	Id	Category
1	1	A
2	2	NUL
3	3	
4	4	

```

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	NULL
3	3		0
4	4		1

```

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	NULL
3	3		0
4	4		2



Representation

NULL

Empty String

Blank Space

" "

" "

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

= ''

= ' '





DATA POLICY

Set of rules that defines
how data should be handled.



#1 DATA POLICY

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

The screenshot shows a SQL Server Management Studio (SSMS) interface with two query windows. The left window contains the following T-SQL code:

```
SELECT *  
FROM Orders
```

A large blue box highlights the word **TRIM**. Below it, a callout box states: **remove unwanted leading and trailing spaces from a string**.

The right window shows the results of the query. It has two panes: **Object Explorer** on the left and **Results** on the right. The Results pane displays the following data:

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

The results show that the **Category** column contains 'A', 'NULL', an empty string, and another empty string respectively. The **CategoryLen** column shows the lengths of these strings: 1, NULL, 0, and 2. The **Policy1** column shows the result of the **TRIM** function applied to the **Category** column, resulting in 1, NULL, 0, and 0 respectively.



#2 DATA POLICY

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

```
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
```

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

NULL IF to replace any empty string to Null Values
And black spaces gets trim and converted into empty strings



#3 DATA POLICY

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

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

```

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(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	
4	4			NULL	

SalesDB

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

```

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

First trim data then convert empty string with null value then finally replace all with value unknown

Always try to avoid policy 1

Used policy 2 because it keep less storage and having good performance of query

While doing data preparation in ETL without inserting it into a table then we used policy 2

#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.

Used when we do data preparation before showing report like powerBI and tableau

Last step before show data to user.

Because null inside report is very hard so we used like this - used policy 3

#3 DATA POLICY

- USE CASE -



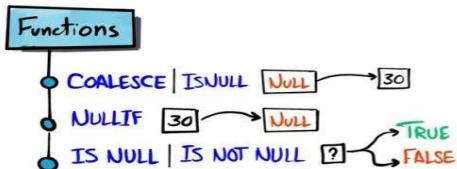
Replacing empty strings, blanks, NULL with default value

during data preparation before using it in reporting

to improve readability and reduce confusion

NULL Functions

- NULLs special markers means missing value.
- Using NULLs can optimize storage and performance.



USE CASES

- Handle NULLs - Data Aggregation
- Handle NULLs - Mathematical operations
- Handle NULLs - Joining Tables
- Handle NULLs - Sorting Data
- Finding unmatched data - Left Anti-Join
- Data Policies
 - Nulls
 - Default Value

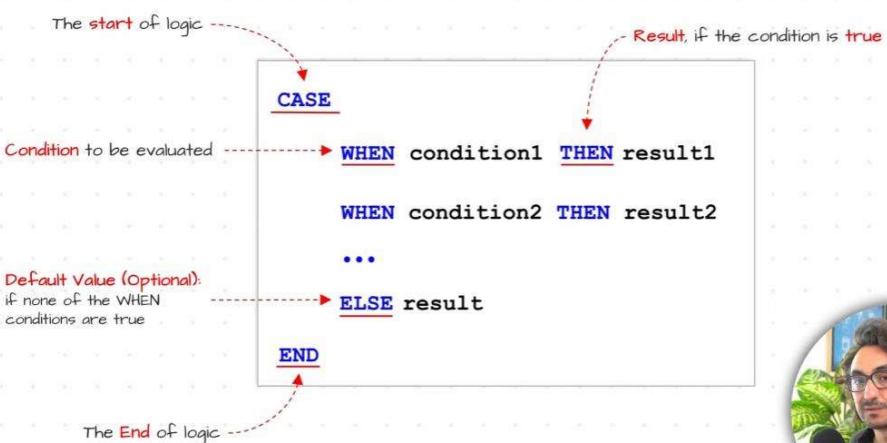


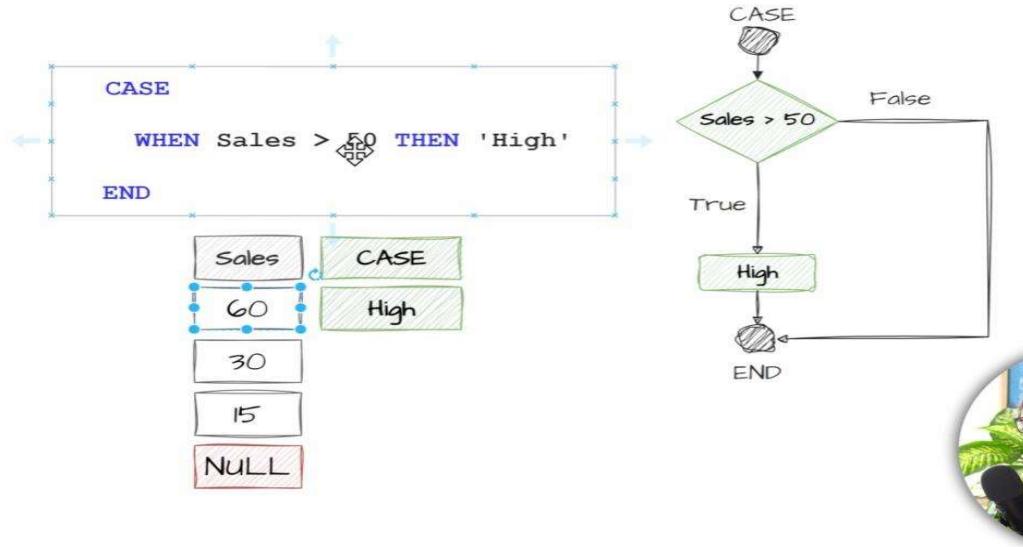
Case - When Statements

CASE STATEMENT

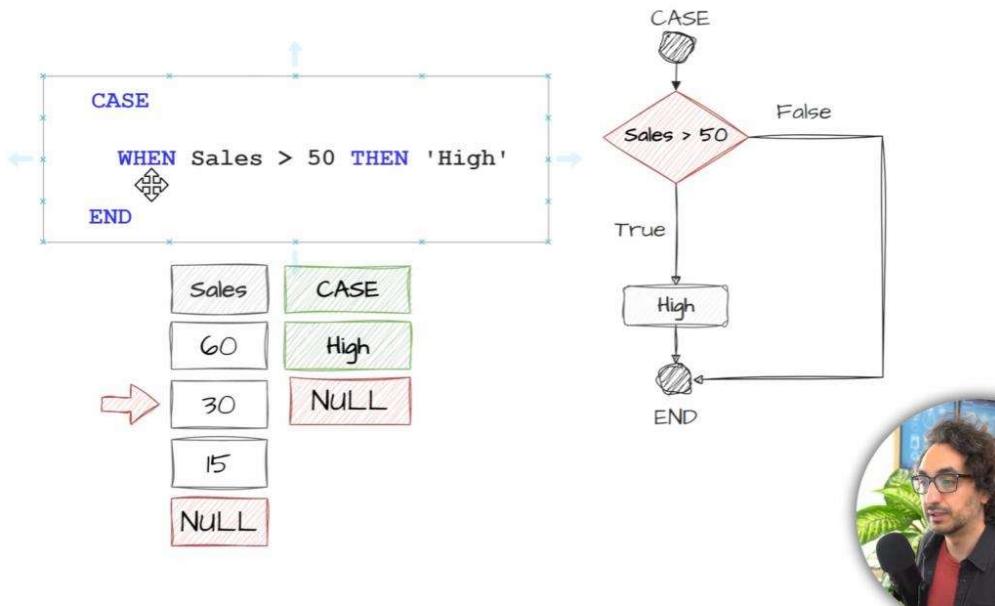
Evaluates a list of conditions and returns a value

when the first condition is met





We get Null as output because we did not used else statement like if when condition not satisfied then what to do

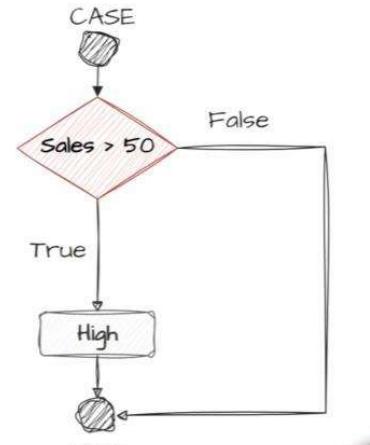


```

CASE
  WHEN Sales > 50 THEN 'High'
END

```

Sales	CASE
60	High
30	NULL
15	NULL
NULL	NULL

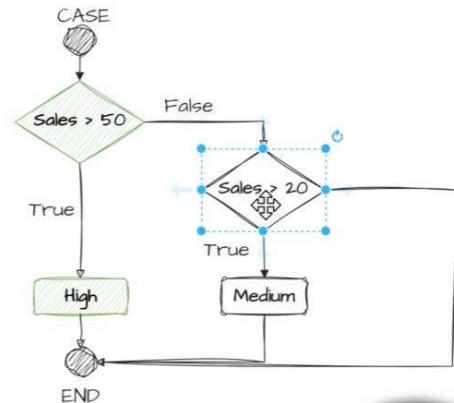


```

CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
END

```

Sales	CASE
60	High
30	
15	

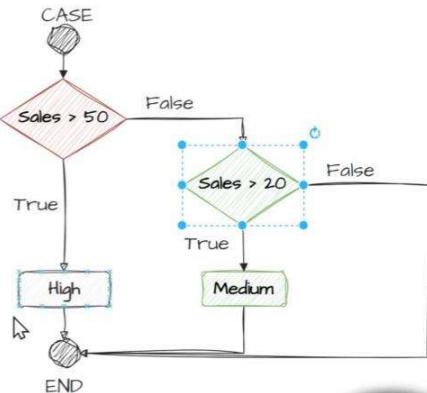
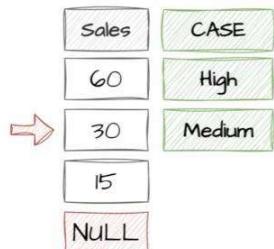


SQL execution stops once the first condition is met

```

CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
END

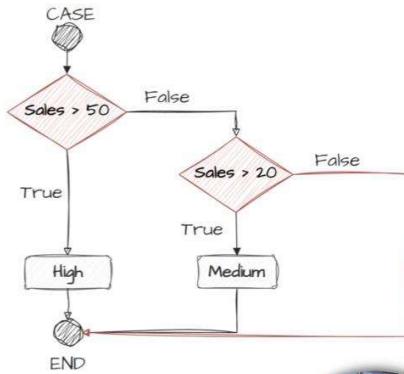
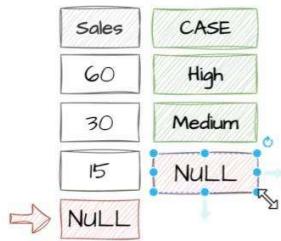
```



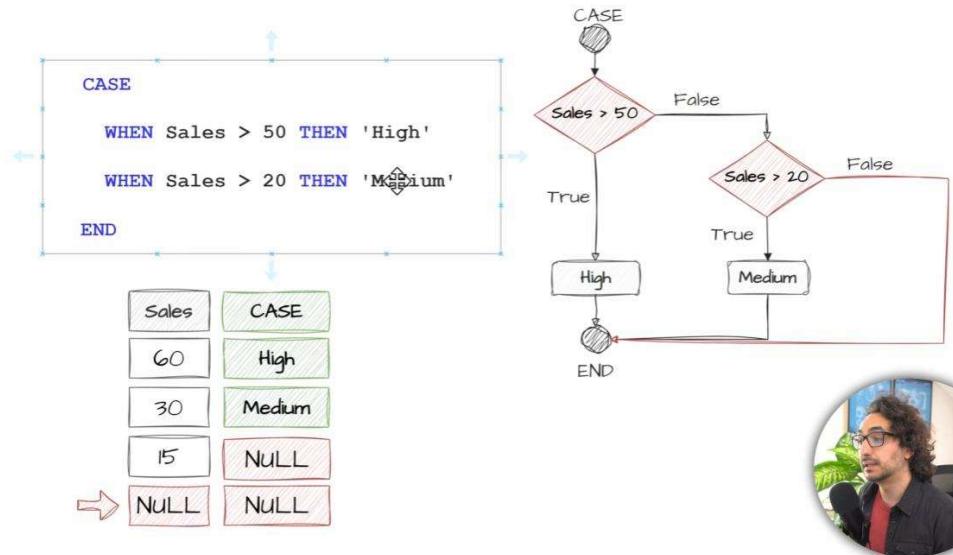
```

CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
END

```



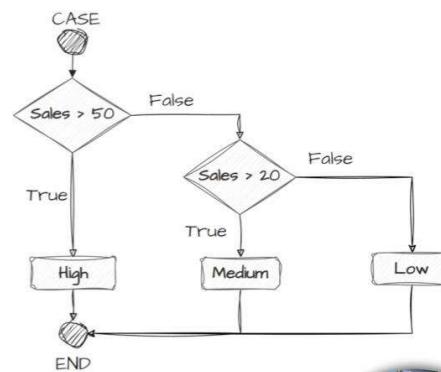
1
2
2
2



```

CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
  ELSE 'Low'
END
  
```

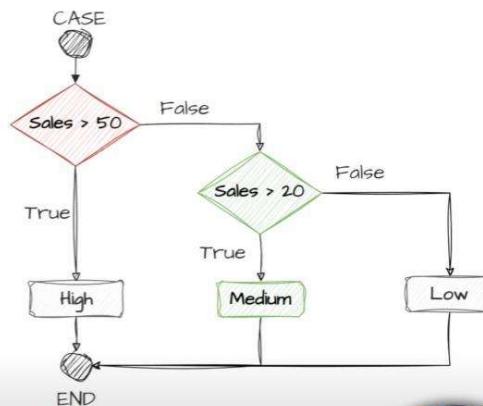
Sales	CASE
60	High
30	
15	
NULL	



```

CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
  ELSE 'Low'
END
  
```

Sales	CASE
60	High
30	Medium
15	
NULL	

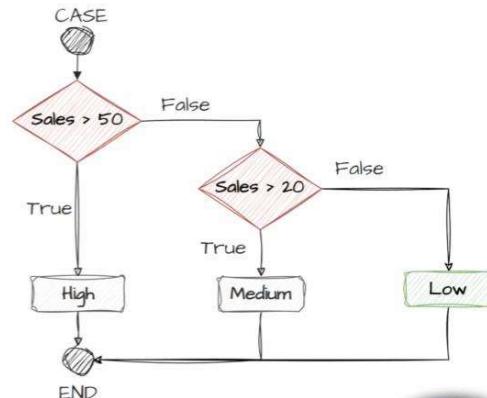


```

CASE
    WHEN Sales > 50 THEN 'High'
    WHEN Sales > 20 THEN 'Medium'
    ELSE 'Low'
END

```

Sales	CASE
60	High
30	Medium
15	Low
NULL	

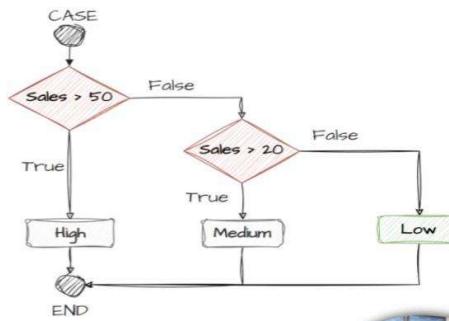


```

CASE
    WHEN Sales > 50 THEN 'High'
    WHEN Sales > 20 THEN 'Medium'
    ELSE 'Low'
END

```

Sales	CASE
60	High
30	Medium
15	Low
NULL	Low



Use cases of Case Statement -

1. Data Transformation

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.

Classifying and grouping data is fundamental in data analysing and reporting because it makes data easier to understand and track

It helps us to aggregating data based on categories

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.

Only case when



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

	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

Query executed successfully.



/* 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
```

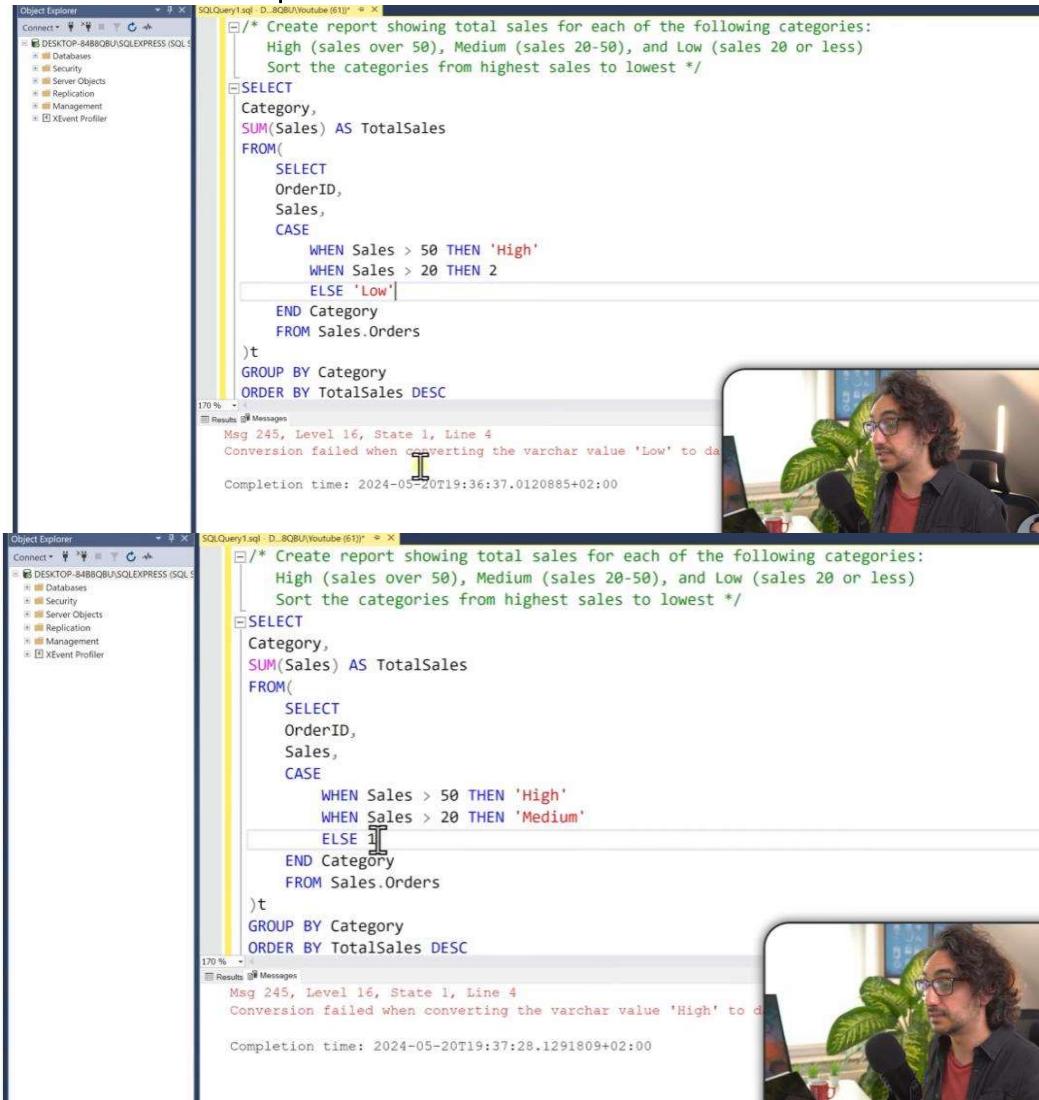
	Category	TotalSales
1	High	210
2	Medium	105
3	Low	65

CASE STATEMENT RULES

Rule 1 :

The data type of the results must be matching

Break rule 1 - example



The screenshot shows two separate SQL queries in the Object Explorer of SQL Server Management Studio.

Query 1:

```
/* 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 2
             ELSE 'Low'
           END Category
      FROM Sales.Orders
  ) t
 GROUP BY Category
 ORDER BY TotalSales DESC
```

Execution results show an error message:

```
Msg 245, Level 16, State 1, Line 4
Conversion failed when converting the varchar value 'Low' to date/time.
```

Query 2:

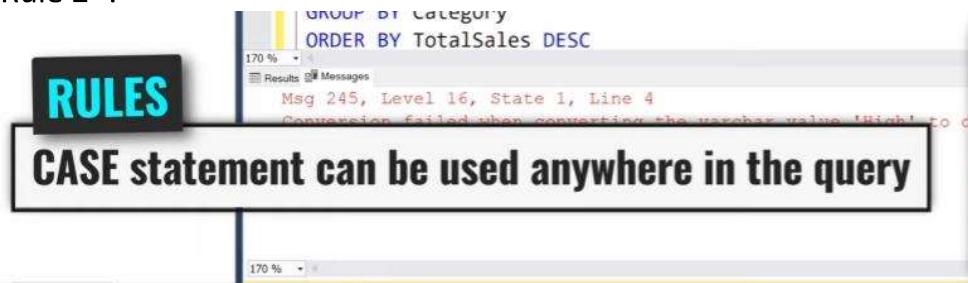
```
/* Create report showing total sales for each of the following categories:
   High (sales over 50), Medium (sales 20-50), and Low (sales 20 or less)
   Sort the categories from highest sales to lowest */

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
```

Execution results show an error message:

```
Msg 245, Level 16, State 1, Line 4
Conversion failed when converting the varchar value 'High' to date/time.
```

Rule 2 :-



Mapping Values : Usecase of case statement

MAPPING

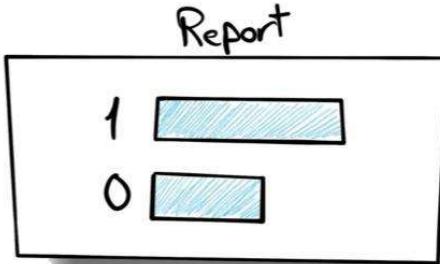
Transform the values from one form to another

Table

~	Status	~	~	~
~		~	~	~
~	0	~	~	~
~	0	~	~	~
—	1	~	~	~

1 → Active

0 → Inactive

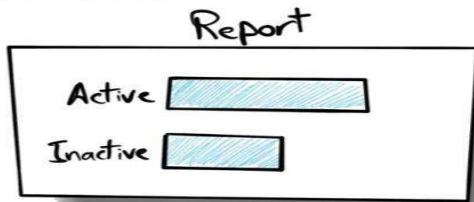


Table

~	Status	~	~	~
~	Active	~	~	~
~	Inactive	~	~	~
~	Inactive	~	~	~
—	Active	~	~	~

1 → Active

0 → Inactive



Object Explorer

SQLQuery2.sql D:\SQLBU\Youtube (77) * x

```
-- Retrieve employee details with gender displayed as full text
SELECT
EmployeeID,
FirstName,
LastName,
Gender
FROM Sales.Employees
```

187 %

Results Messages

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

DESKTOP-84B8QBU\SQLEXPRESS ... DESKTOP-84B8QBU\Y

Used Mapping between the old value and the new value - case when statement

SalesDB

Object Explorer

SQLQuery2.sql D:\SQLBU\Youtube (77) * x

```
-- Retrieve 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 Available'
END GenderFullText
FROM Sales.Employees
```

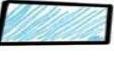
187 %

Results Messages

	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

DESKTOP-84B8QBU\SQLEXPRESS ... DESKTO

Report

Germany	
United States	
Italy	
France	

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

Report

DE	
US	
IT	
FR	