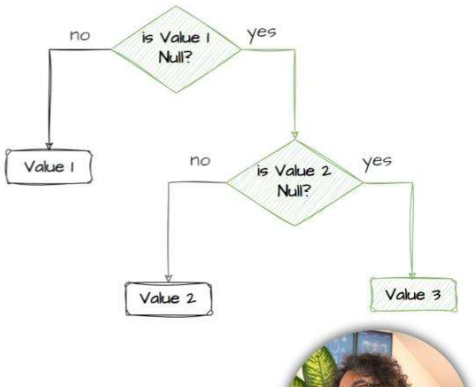
SYNTAX

```
COALESCE(value1, value2, value3)
```

```
COALESCE(ShippingAddress, BillingAddress, 'N/A')
```

OrderID	Shipment Address	Billing Address	COALESCE
1	A	B	A
2	NULL	C	C
3	NULL	NULL	N/A



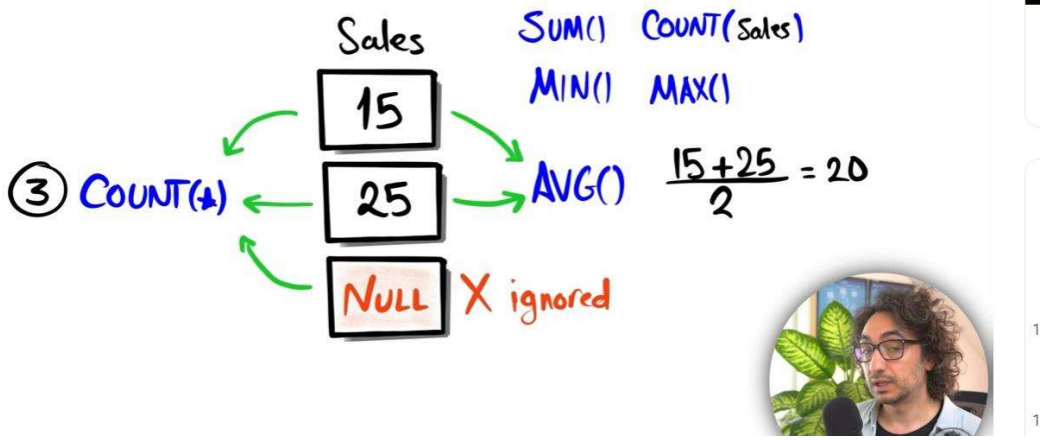
ISNULL	COALESCE
Limited to two values	Unlimited
Fast	Slow
SQL Server → ISNULL Oracle → NVL MySQL → IFNULL	Available in All Databases

Usecase of data handling

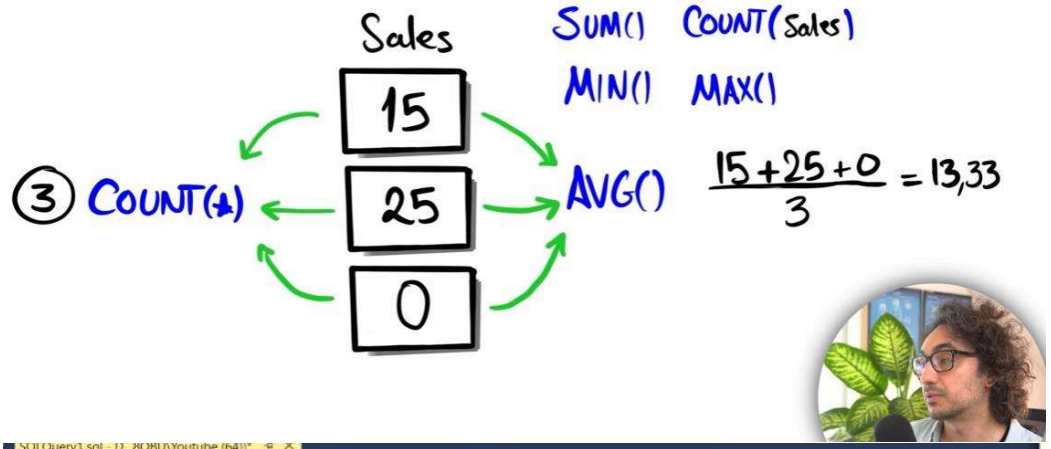
ISNULL | COALESCE

- USE CASE -

Handle the NULL before doing data aggregations



Replace Null to get accurate result




```sql
-- 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
```

185 %

Results Messages

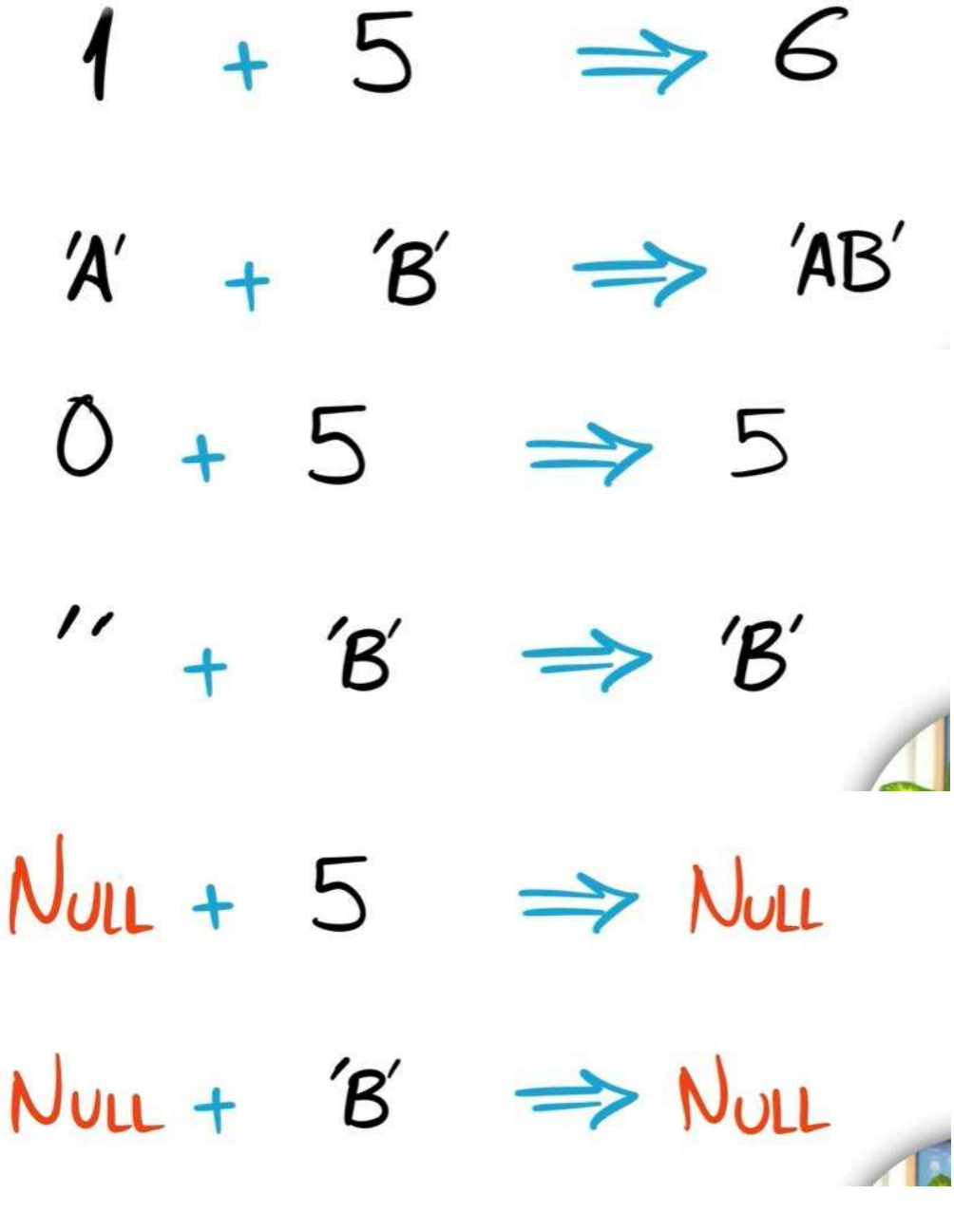
	CustomerID	Score	Score2	AvgScores	AvgScores2
1	1	350	350	625	500
2	2	900	900	625	500
3	3	750	750	625	500
4	4	500	500	625	500
5	5	NULL	0	625	500

Click to select the whole column

ISNULL | COALESCE

- USE CASE -

Handle the NULL before doing mathematical operations



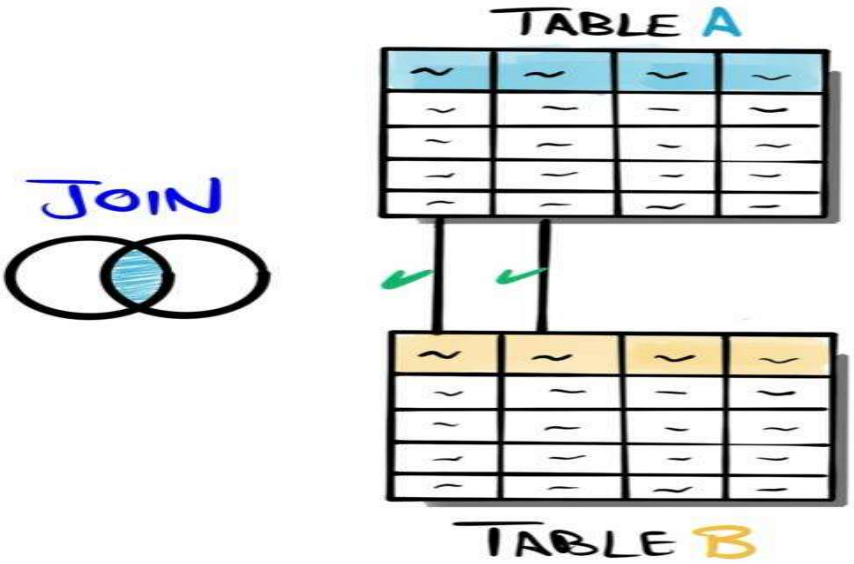
```sql
-- Display the full name of customers in a single field
-- by merging their first and last names,
-- and add 10 bonus points to each customer's score.

SELECT
CustomerID,
FirstName,
LastName,
FirstName + ' ' + COALESCE(LastName, '')  AS FullName,
Score,
COALESCE(Score, 0) + 10 AS ScoreWithBonus
FROM Sales.Customers
```

Results | Messages

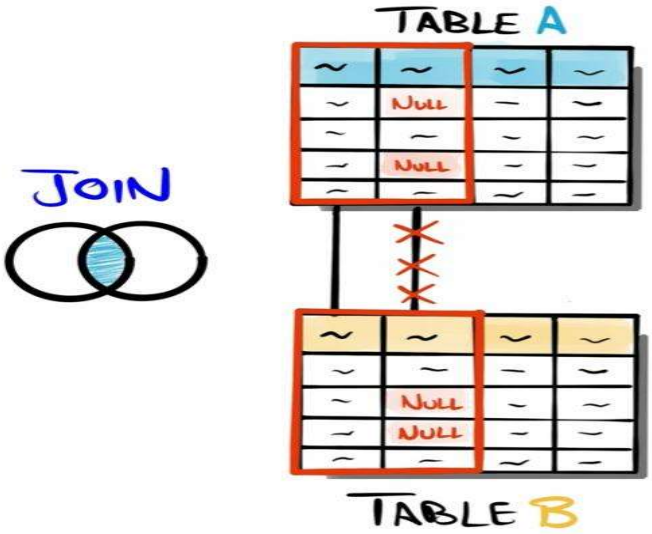
	CustomerID	FirstName	LastName	FullName	Score	ScoreWithBonus
1	1	Jossef	Goldberg	Jossef Goldberg	350	360
2	2	Kevin	Brown	Kevin Brown	900	910
3	3	Mary	NULL	Mary	750	760
4	4	Mark	Schwarz	Mark Schwarz	500	510
5	5	Anna	Adams	Anna Adams	NULL	10

Handling Null Joins

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

Table1

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

Table2

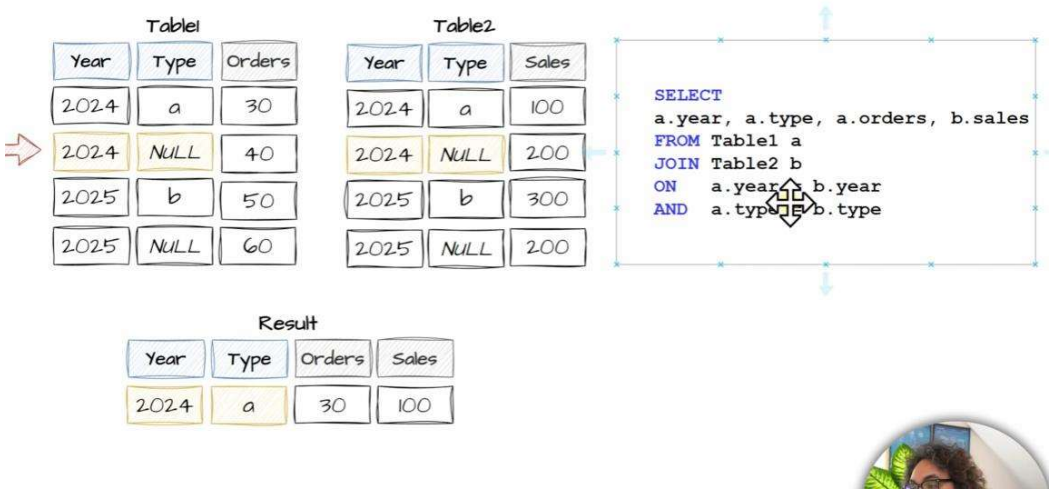
Year	Type	Sales
2024	a	100
2024	NULL	200
2025	b	300
2025	NULL	200

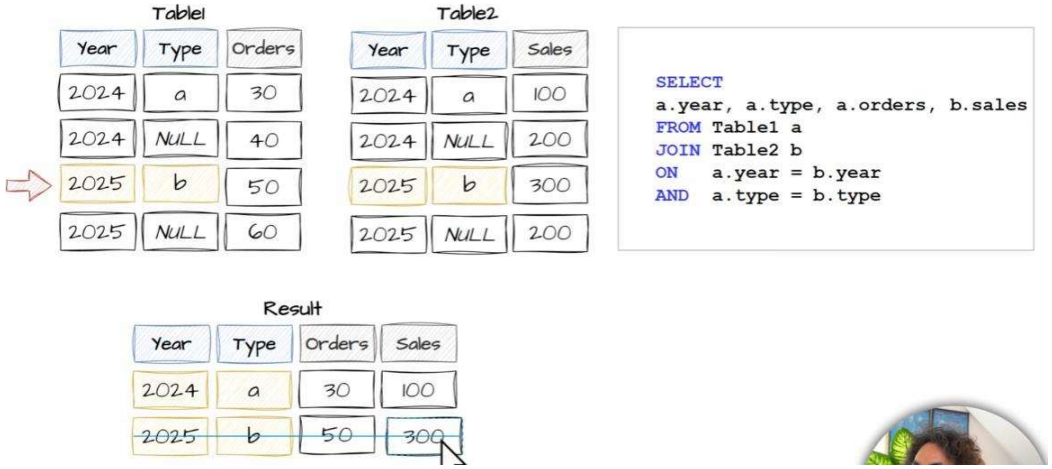
```
SELECT
a.year, a.type, a.orders, b.sales
FROM Table1 a
JOIN Table2 b
ON   a.year = b.year
AND  a.type = b.type
```

Result

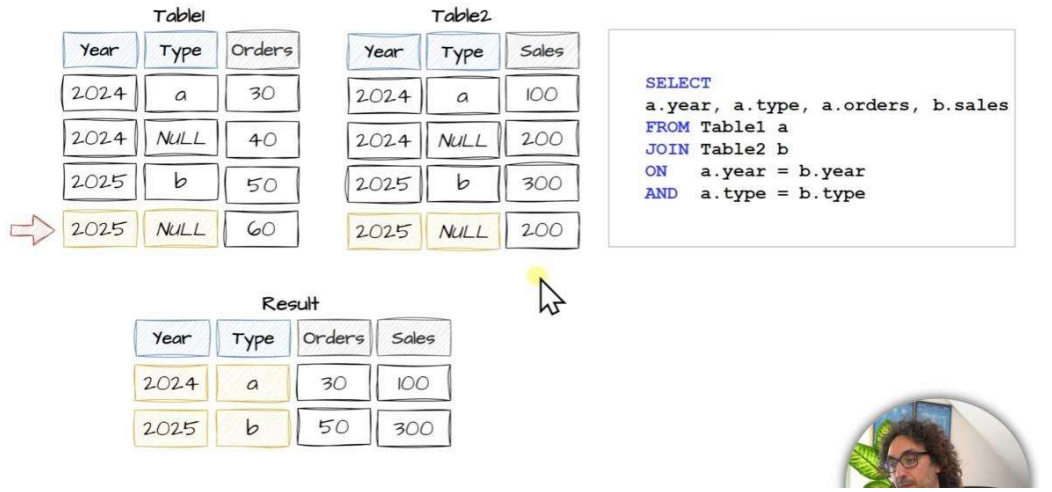
Year	Type	Orders	Sales
2024	a	30	100

Here second example of row everything matching but sql not perfrom 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 -



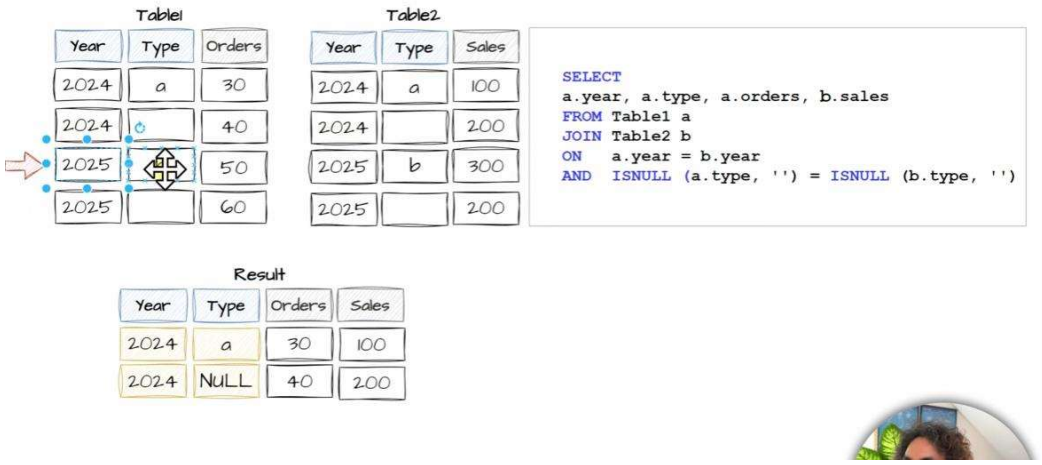
Above two tables were identical as we compare keys so we are losing data in the result and getting inaccurate result.

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

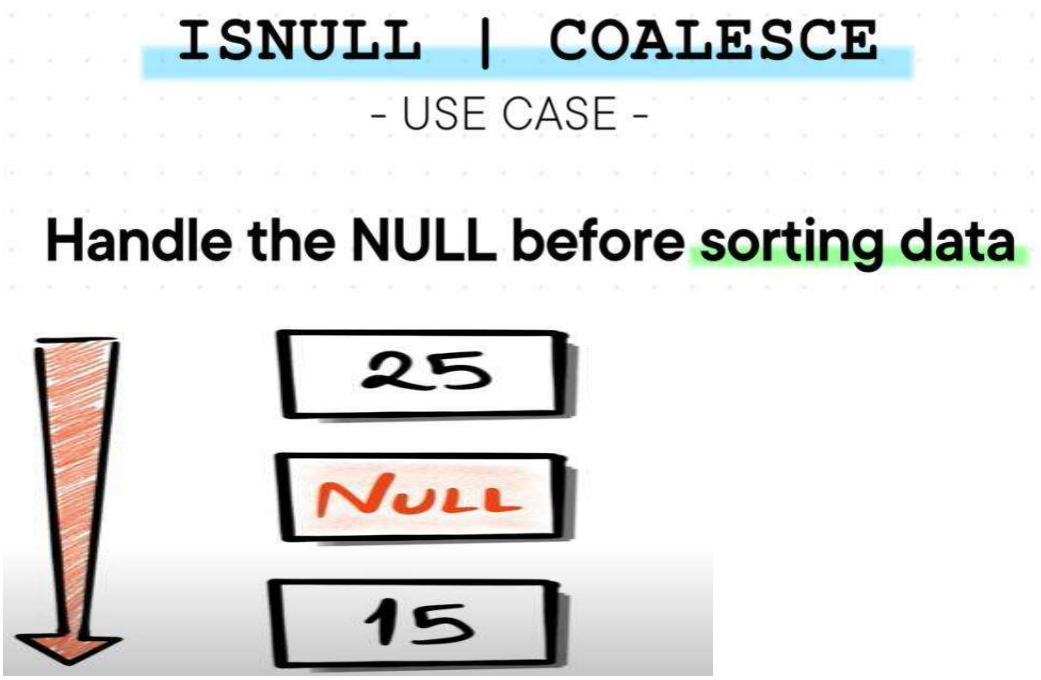
Year	Type	Sales
2024	a	100
2024		200
2025	b	300
2025		200

```
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, '')
```

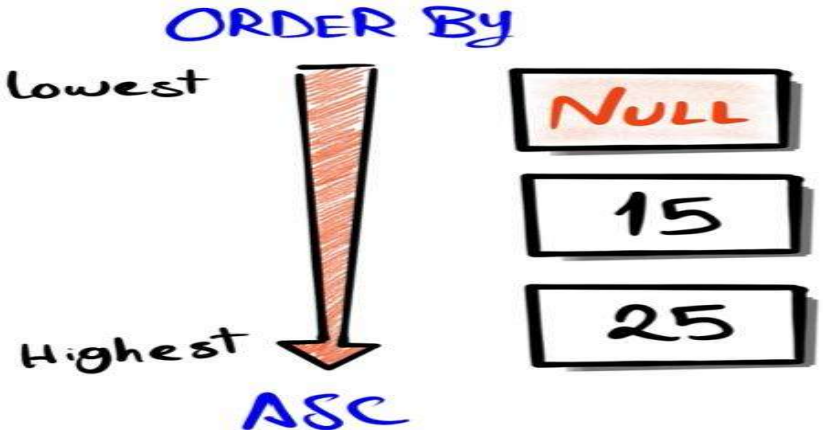
Result

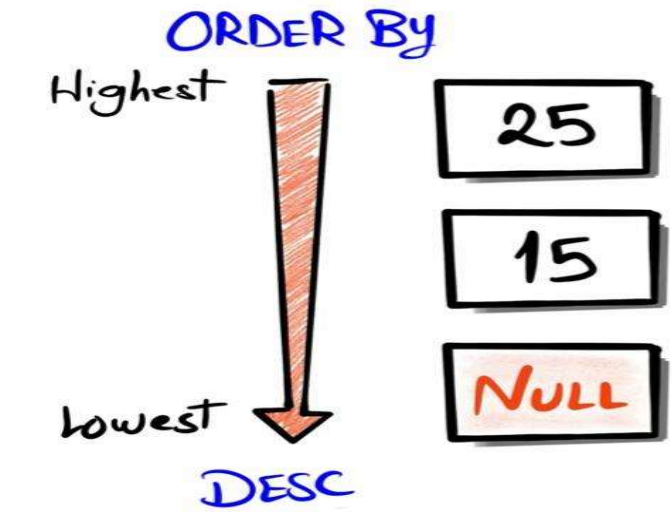
Year	Type	Orders	Sales
2024	a	30	100
2024	NULL	40	200
2025	b	50	300
2025	NULL	60	200

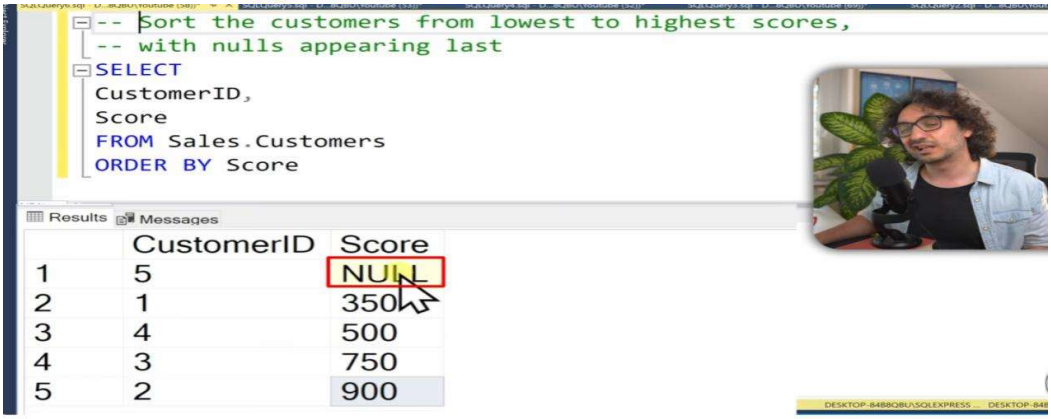
Next Use case - Handle NUll value before sorting it



Values like this sql cosidered null as smallest value







Handle Null Before Shorting the data

```sql
-- 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

SQL NULL Functions | COALESCE, ISNULL, NULLIF, IS (NOT) NULL | #SQL Cou

```sql
-- 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)
```

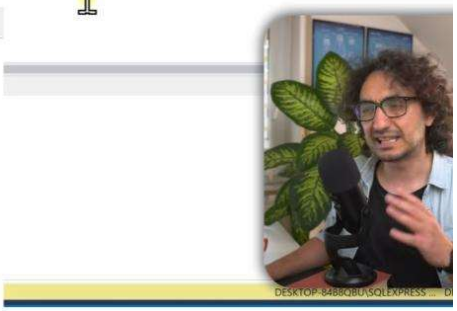
Results | Messages

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

```sql
-- 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
```

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



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

```sql
-- 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
```

Results Messages

	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

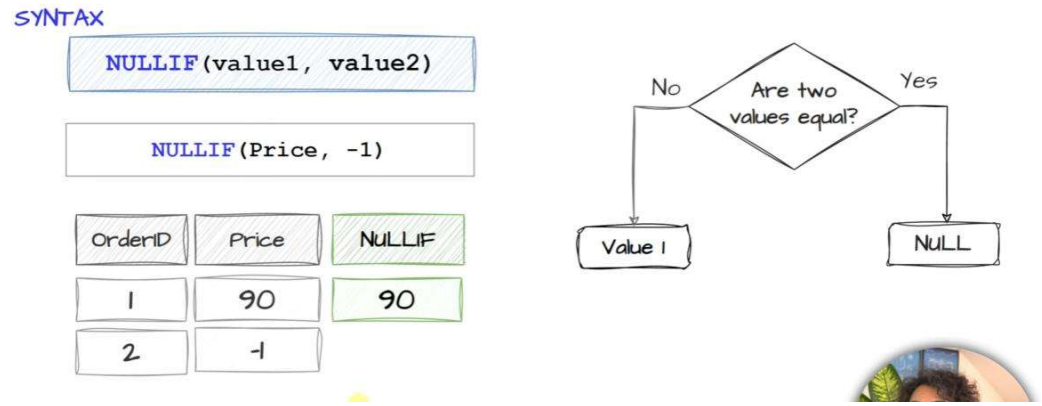
```
NULLIF(value1, Value2)
```

Example

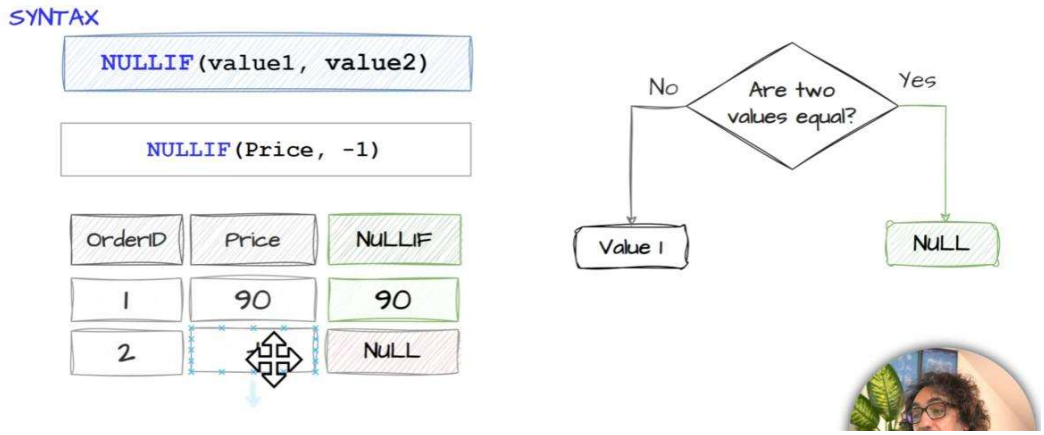
```
NULLIF(Shipping_Address, 'unknown')
```

Example

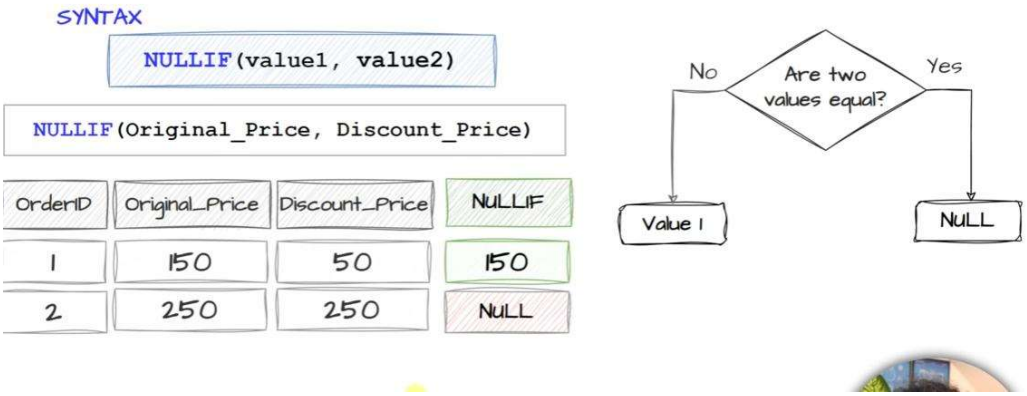
```
NULLIF(Shipping_Address, Billing_Address)
```



Here we are replacing a real value with null values

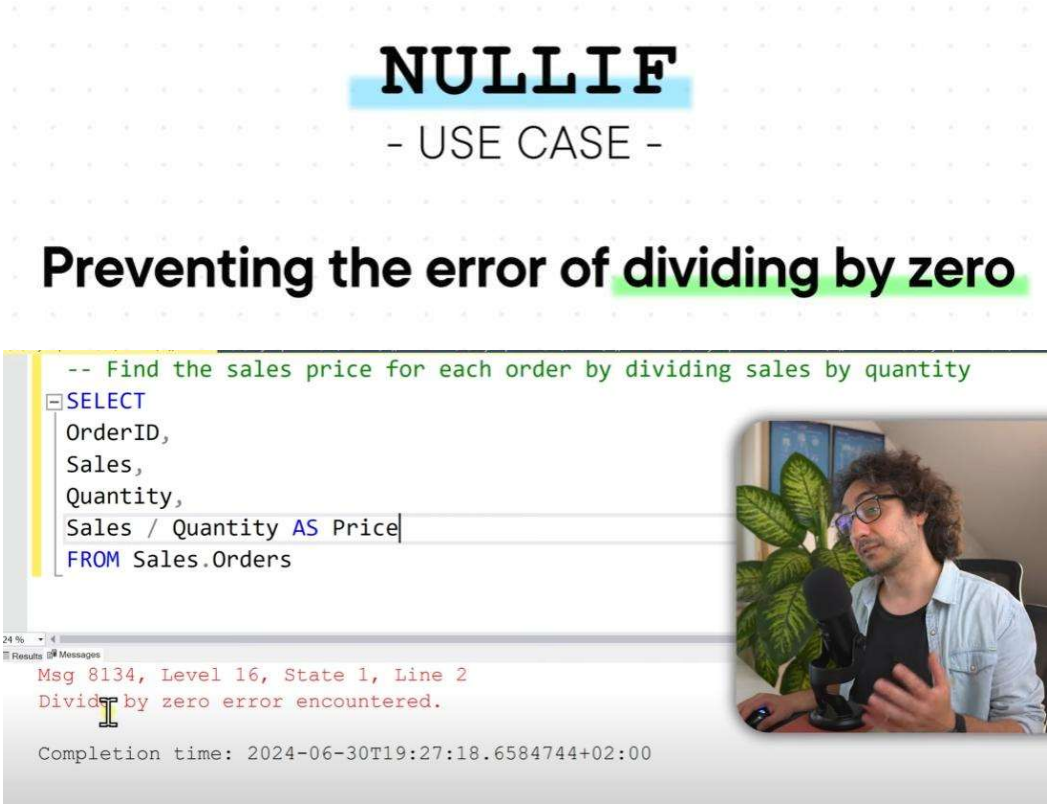


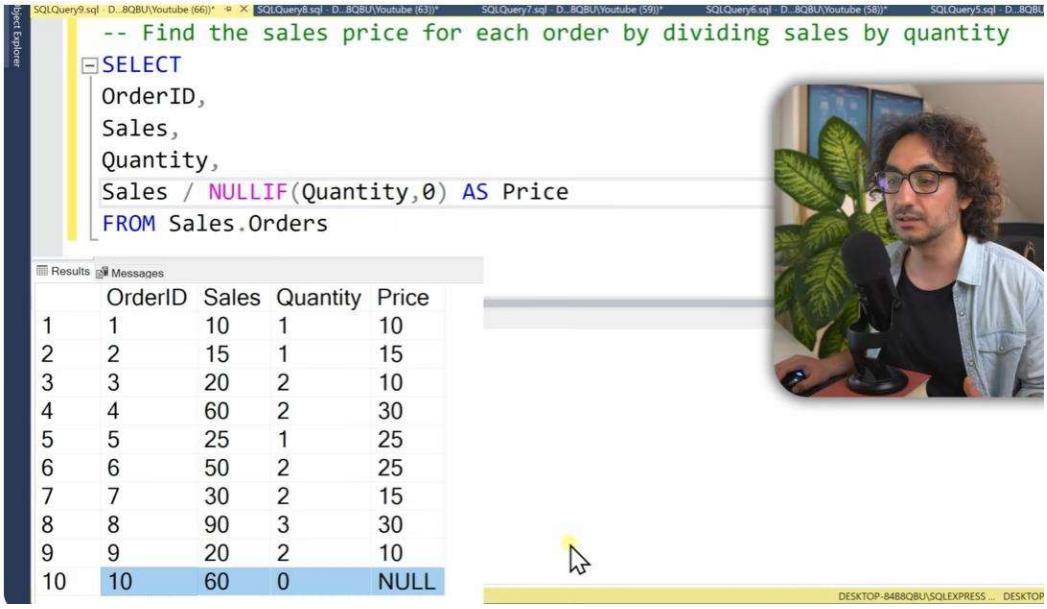
Case if both values are equal in Null IF function:



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

NUll iF case when division by zero





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
```

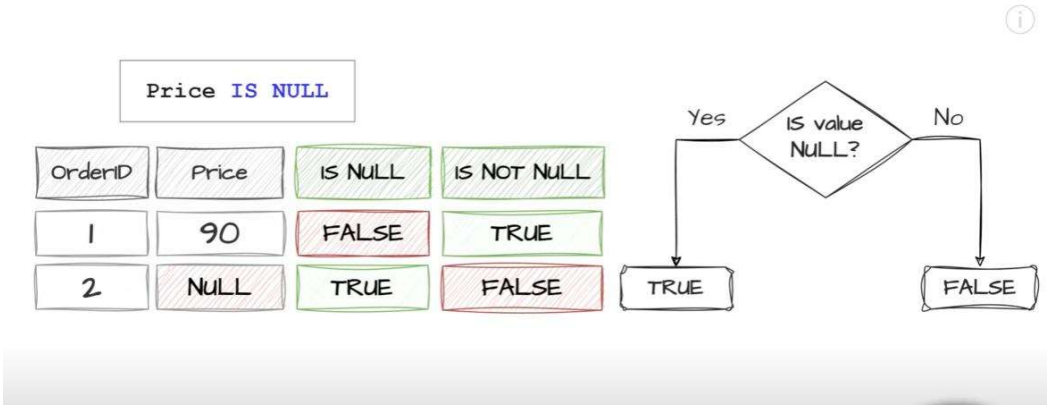
Example

```
Shipping_Address IS NULL
```

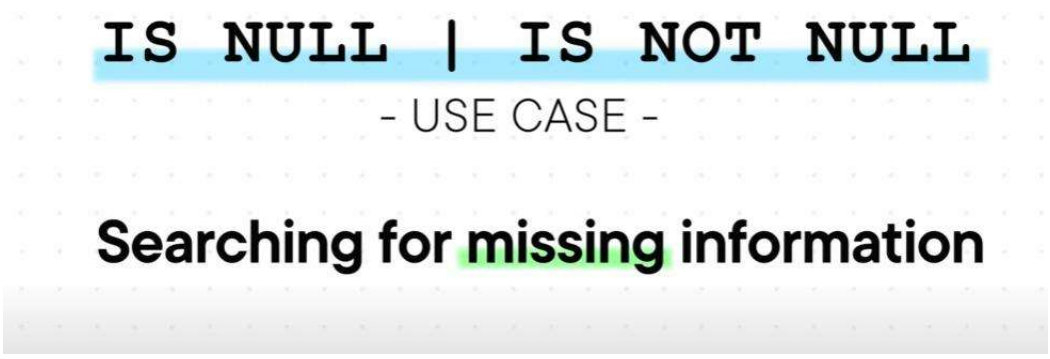
Example

```
Shipping_Address IS NOT NULL
```

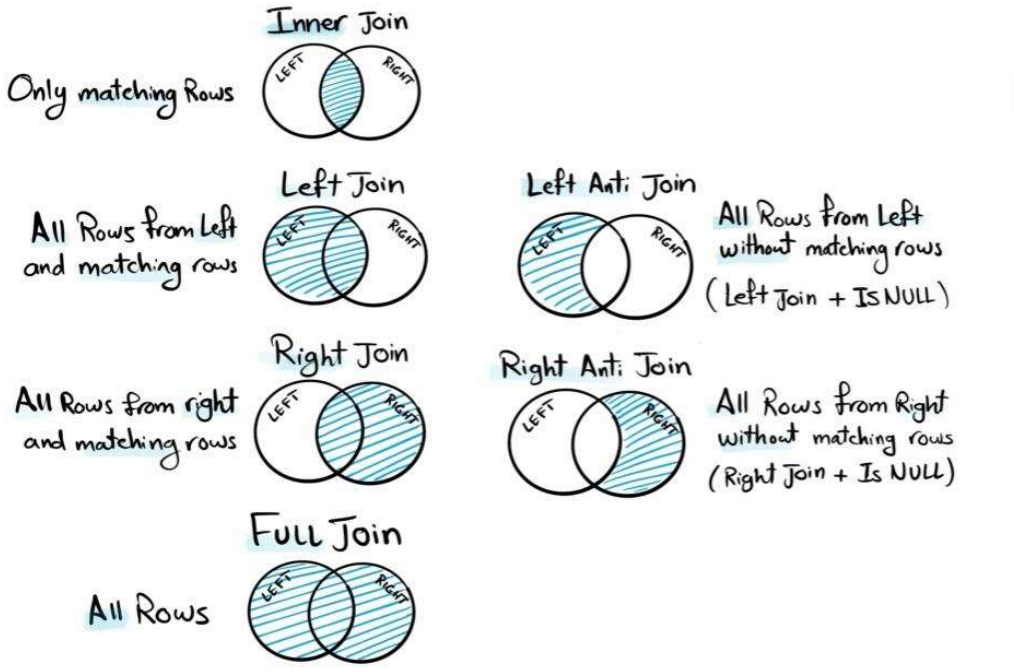
How it works

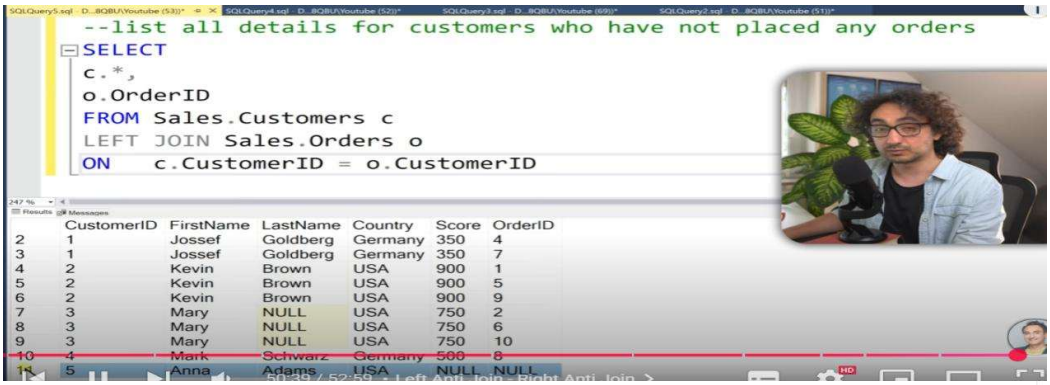


Usecases : search for missing information



IS Null Use case -- Anti Join



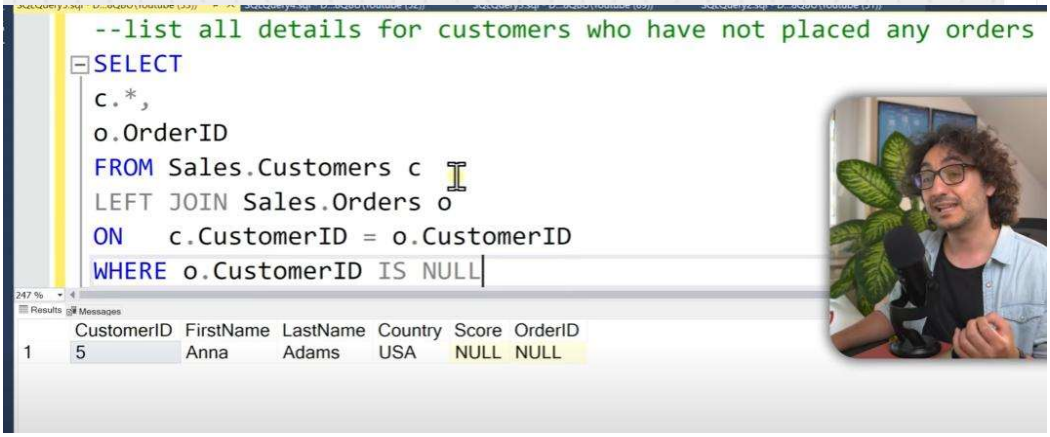


```
--list all details for customers who have not placed any orders
SELECT
c.*,
o.OrderID
FROM Sales.Customers c
LEFT JOIN Sales.Orders o
ON    c.CustomerID = o.CustomerID
```

	CustomerID	FirstName	LastName	Country	Score	OrderID
2	1	Jossef	Goldberg	Germany	350	4
3	1	Jossef	Goldberg	Germany	350	7
4	2	Kevin	Brown	USA	900	1
5	2	Kevin	Brown	USA	900	5
6	2	Kevin	Brown	USA	900	9
7	3	Mary	NULL	USA	750	2
8	3	Mary	NULL	USA	750	6
9	3	Mary	NULL	USA	750	10
10	4	Mark	Schwarz	Germany	500	8
11	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
1	5	Anna	Adams	USA	NULL	NULL

NULL

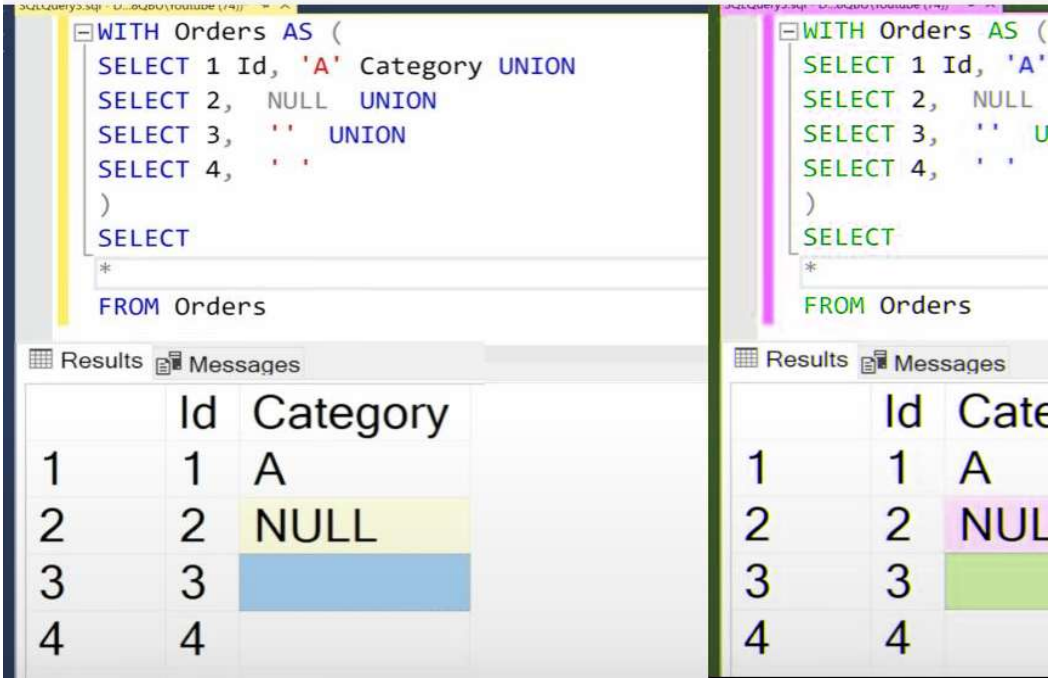
means nothing, unknown!

EMPTY STRING ''

String value has zero characters

BLANK SPACE ' '

String value has one or more space characters



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

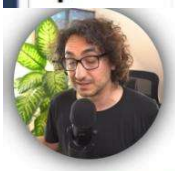
Results Messages

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

```sql
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



	NULL	Empty String	Blank Space
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	= ''	= ' '

DATA POLICY

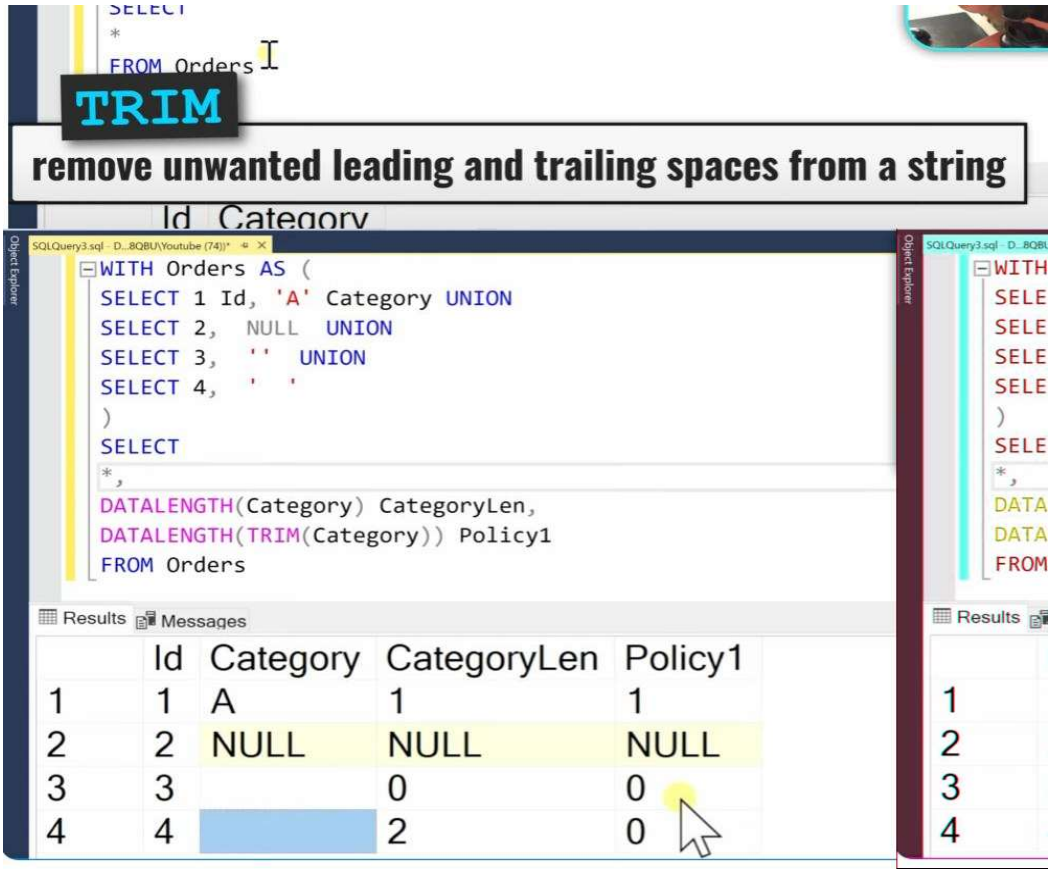
Set of rules that defines

how data should be handled.

#1 DATA POLICY

Only use NULLs and empty strings,

but avoid blank spaces.



#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
```

Results Messages

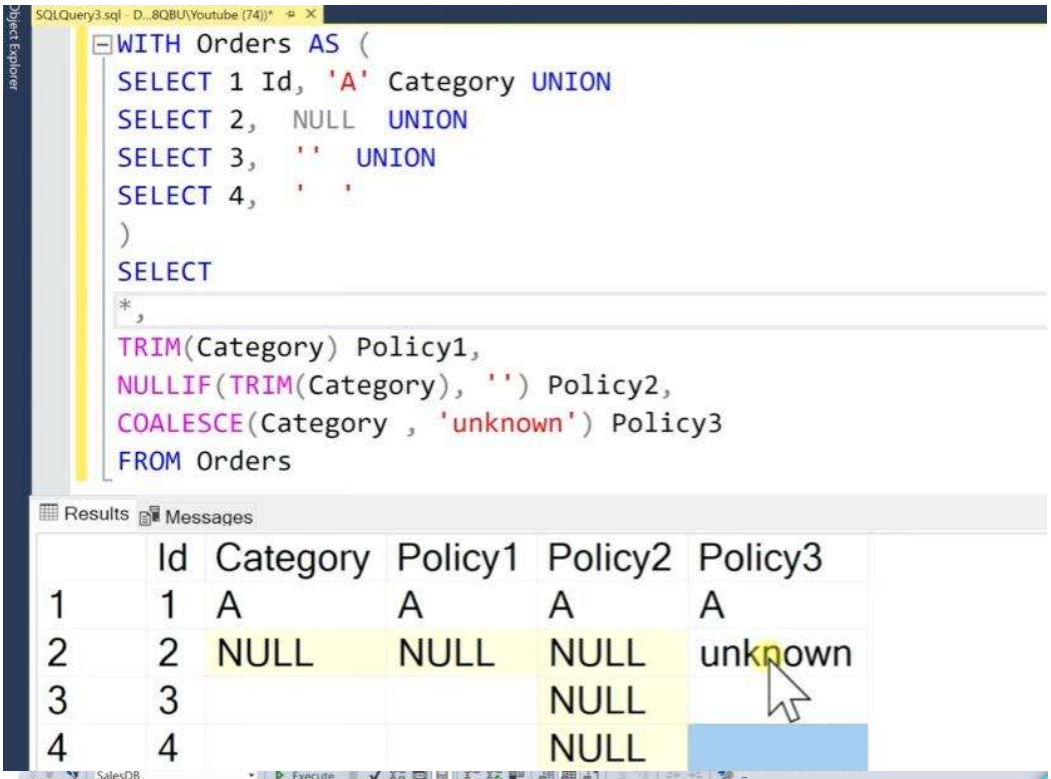
	Id	Category	Policy1	Policy2
1	1	A	A	A
2	2	NULL	NULL	NULL
3	3			NULL
4	4			NULL

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

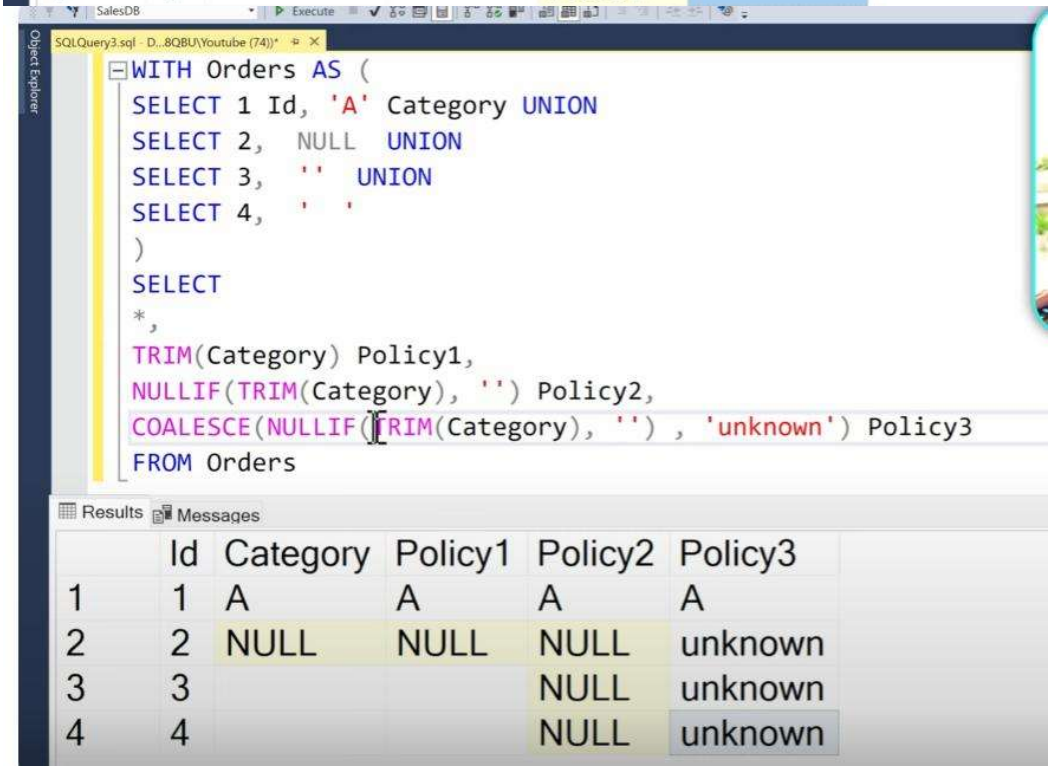
avoid using nulls, empty strings, and blank spaces.



```
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	



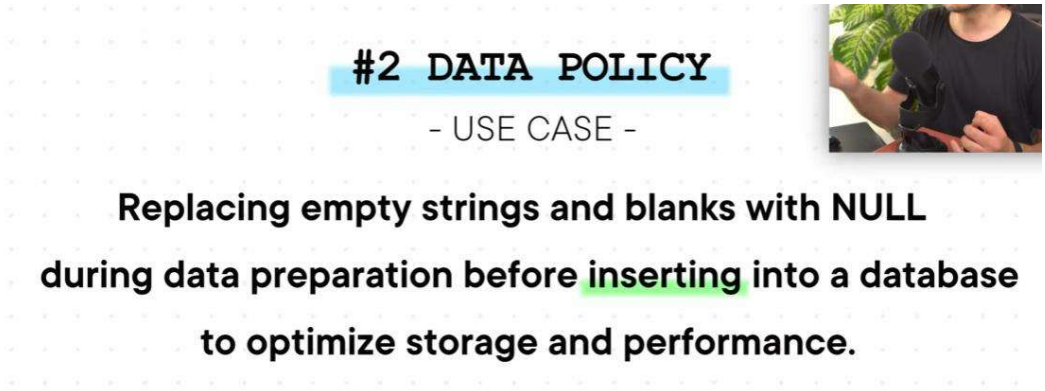
```
WITH Orders AS (
SELECT 1 Id, 'A' Category UNION
SELECT 2,  NULL  UNION
SELECT 3,  ''  UNION
SELECT 4,  '  '
)
SELECT
*,
TRIM(Category) Policy1,
NULLIF(TRIM(Category), '') Policy2,
COALESCE(NULLIF(TRIM(Category), '') , 'unknown') Policy3
FROM Orders
```

Results | Messages

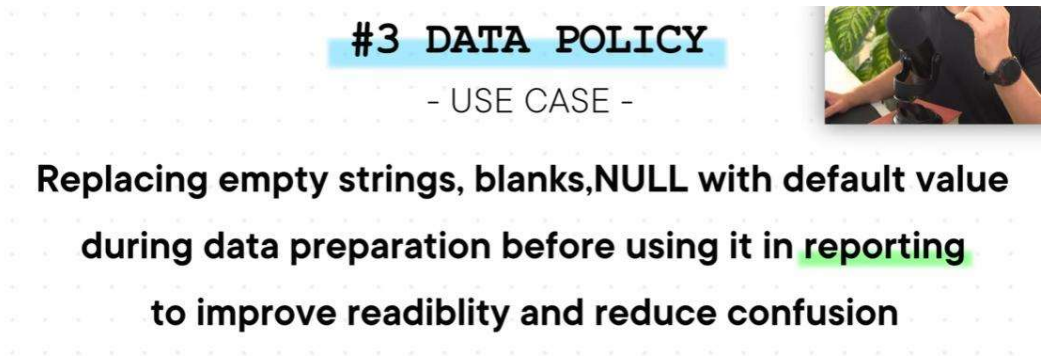
	Id	Category	Policy1	Policy2	Policy3
1	1	A	A	A	A
2	2	NULL	NULL	NULL	unknown
3	3			NULL	unknown
4	4			NULL	unknown

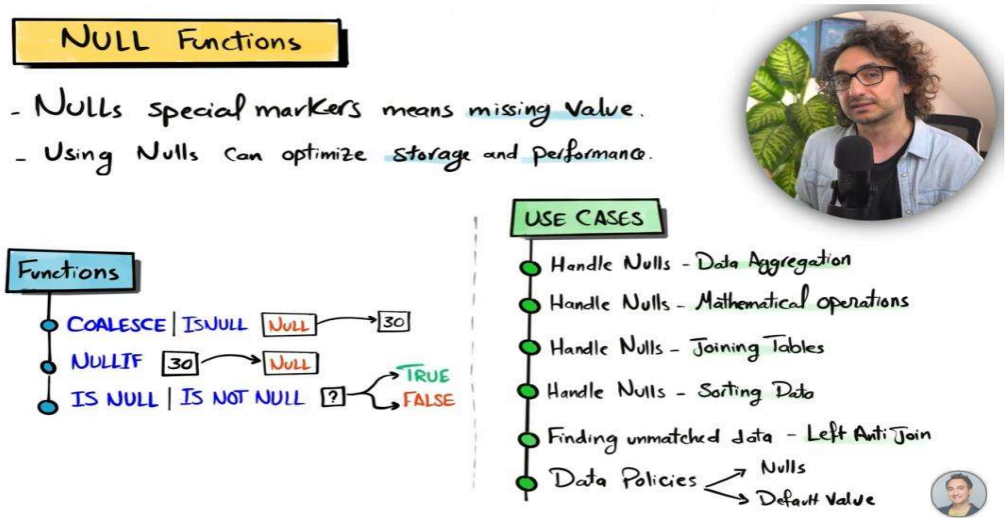
First trim data then convert empty string with null value then finally replace all with value unknown

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

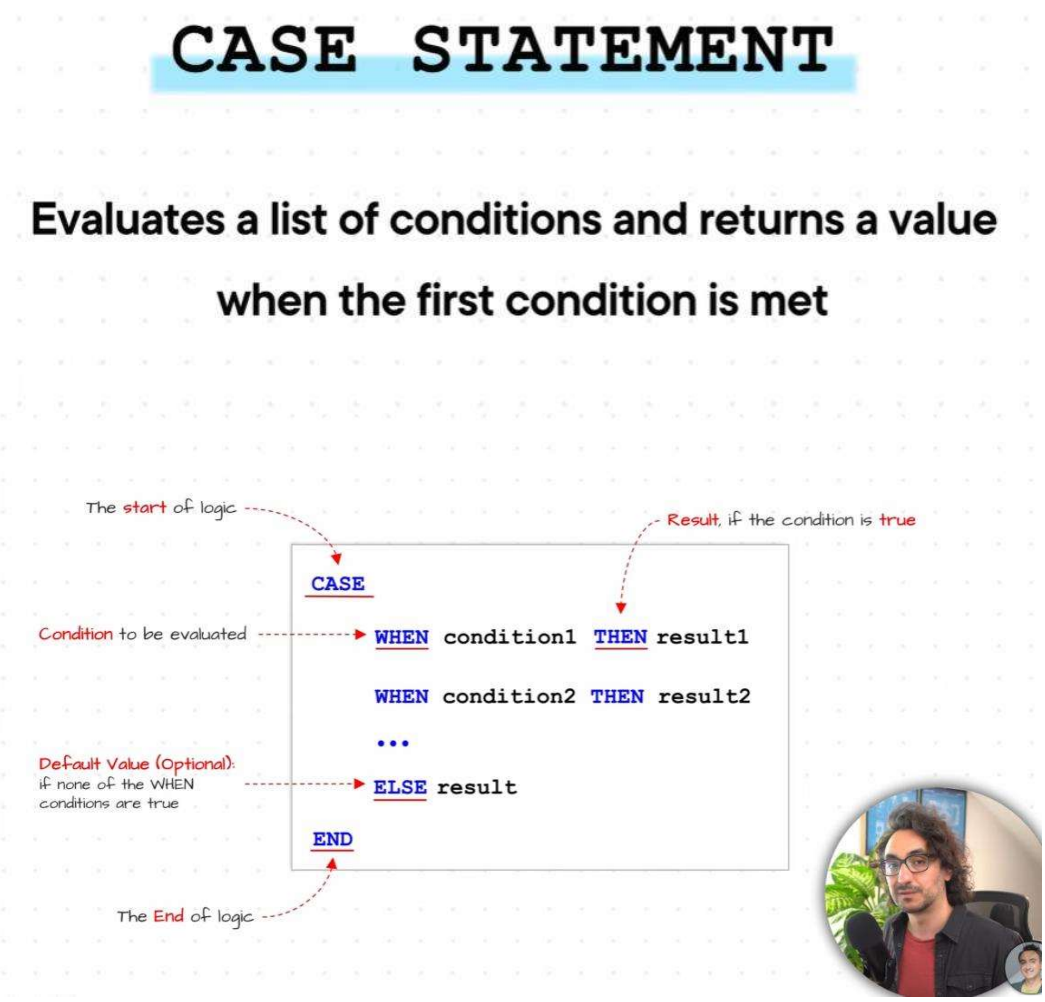


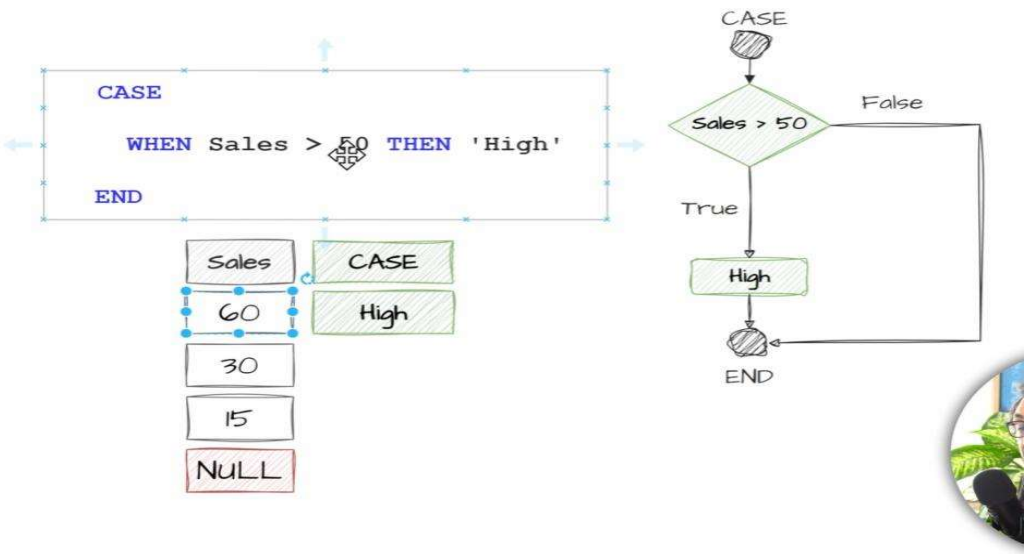
Used when we do data preparation before showing report like poweBI and tableau
Last step before show data to user.
Because null inside report is very hard so we used like this - used policy 3



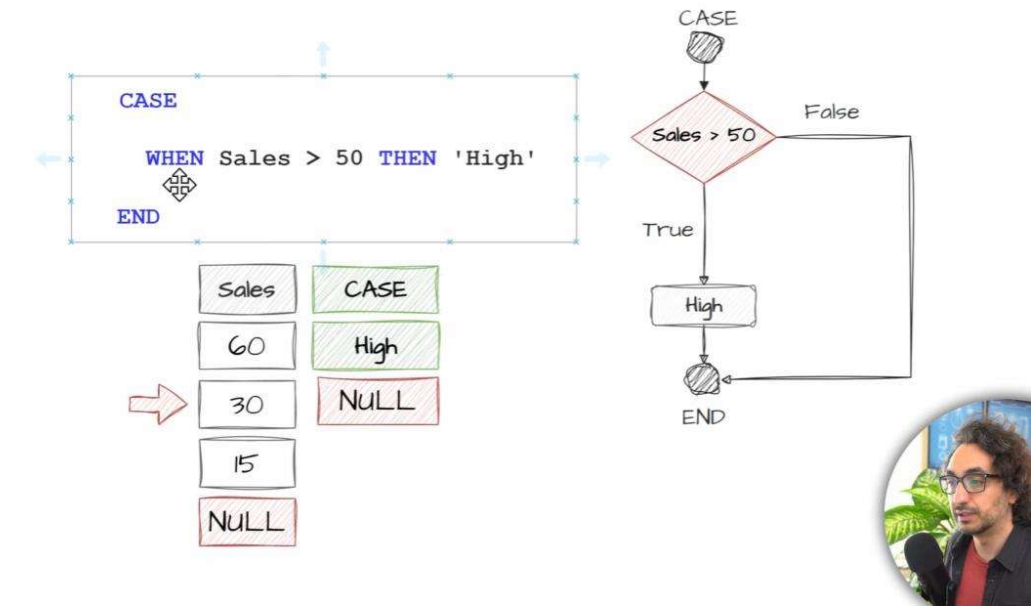


Case - When Statements

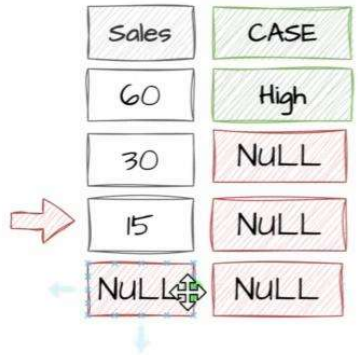


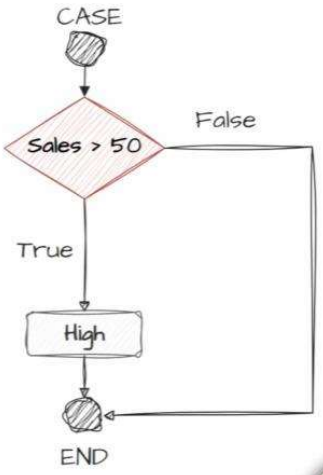


We get Null as output because we did not used else statement like if when condition not satisfied then what to do

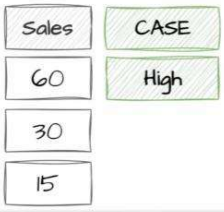


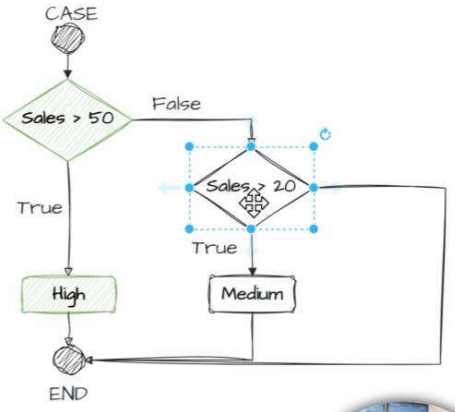
```
CASE
  WHEN Sales > 50 THEN 'High'
END
```





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

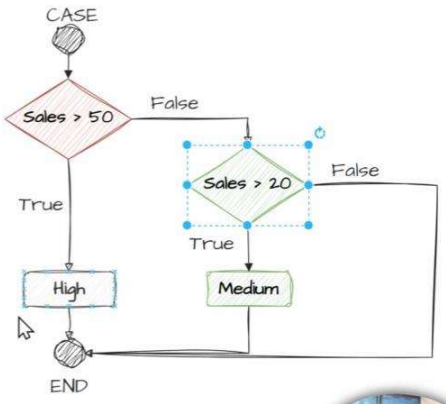




SQL execution stops once the first condition is met

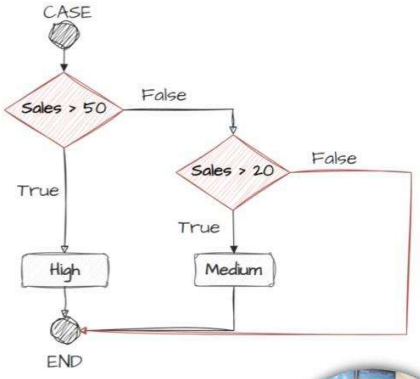
```
CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
END
```

Sales	CASE
60	High
30	Medium
15	
NULL	



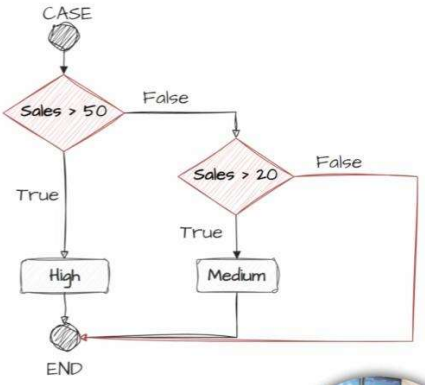
```
CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
END
```

Sales	CASE
60	High
30	Medium
15	NULL
NULL	



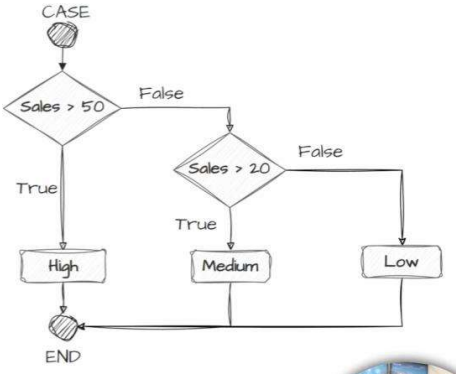
```
CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
END
```

Sales	CASE
60	High
30	Medium
15	NULL
NULL	NULL



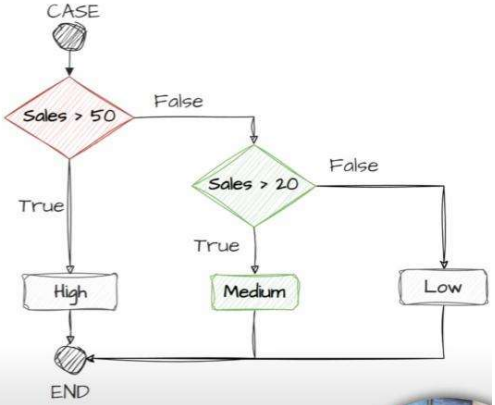
```
CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
  ELSE 'Low'
END
```

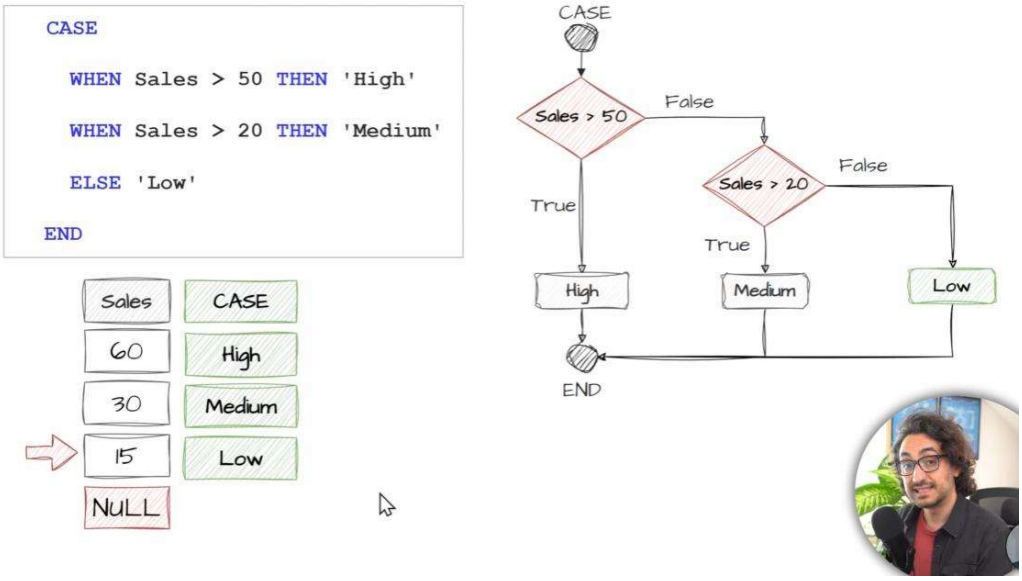
Sales	CASE
60	High
30	
15	
NULL	

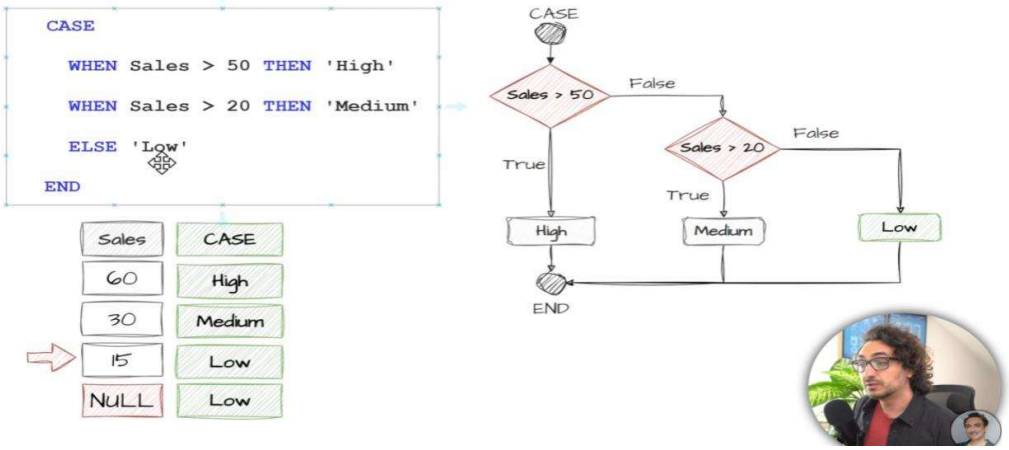


```
CASE
  WHEN Sales > 50 THEN 'High'
  WHEN Sales > 20 THEN 'Medium'
  ELSE 'Low'
END
```

Sales	CASE
60	High
30	Medium
15	
NULL	







Usecases of Case Statement -

1. DataTransformation

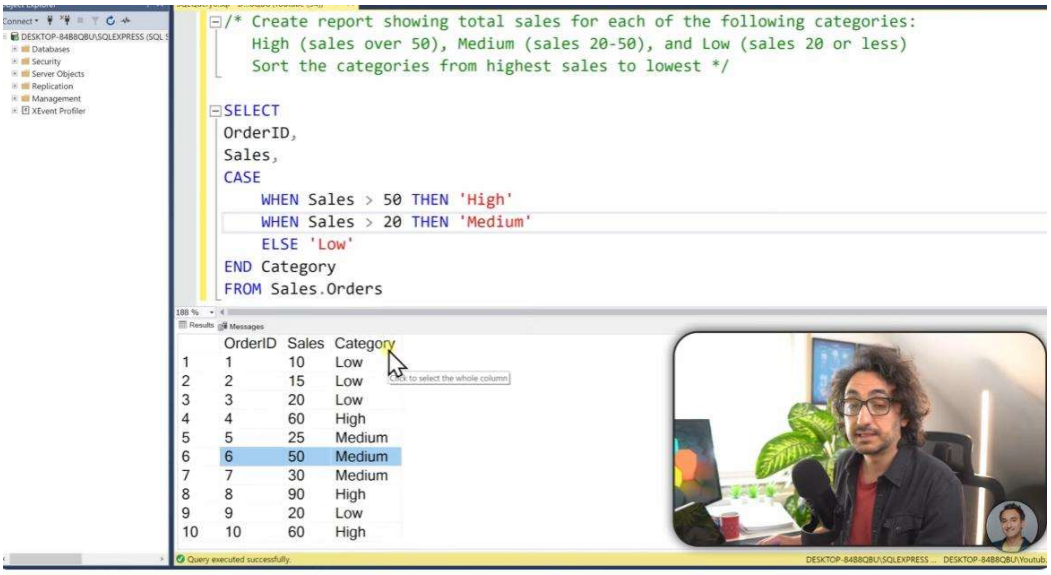
CATEGORIZING DATA

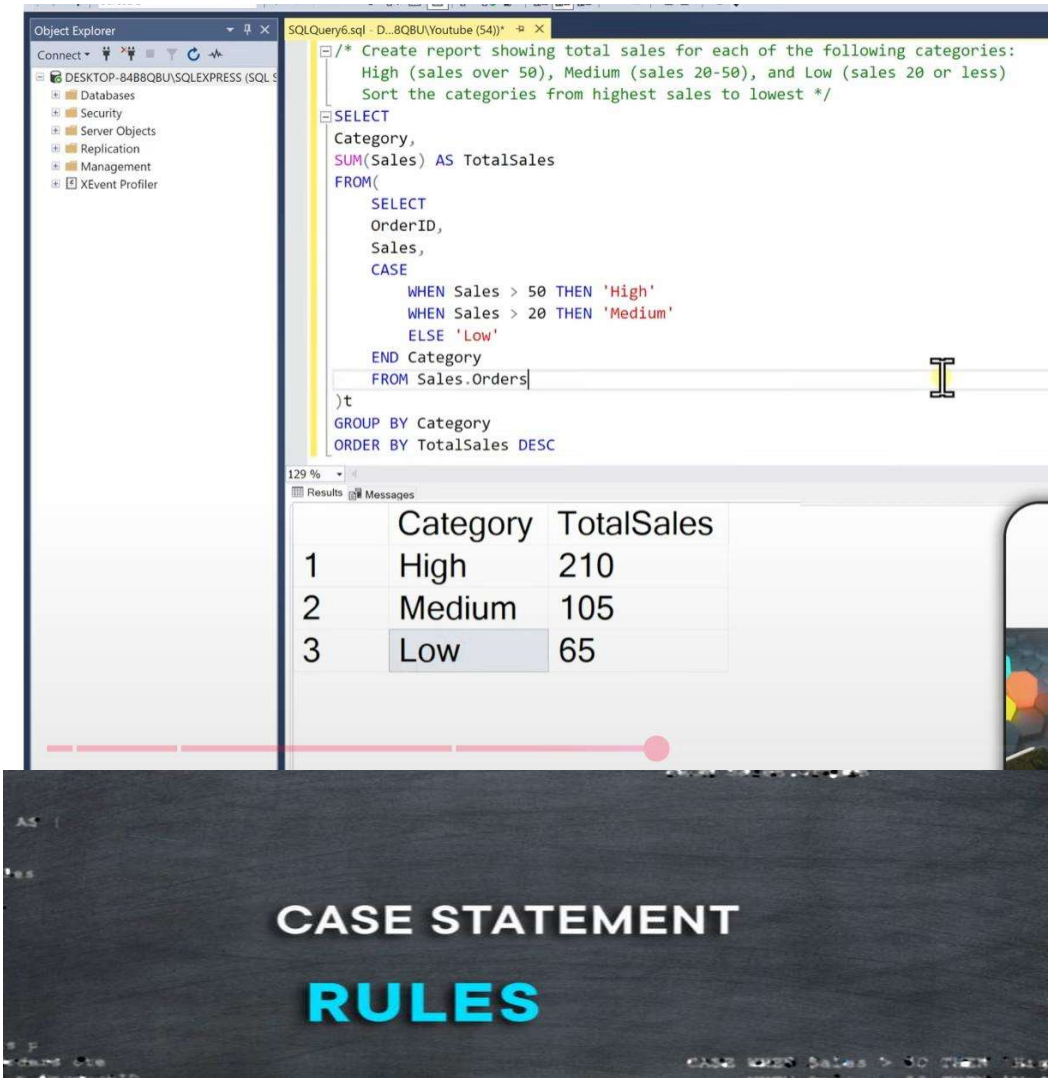
Group the data into different categories based on certain conditions.

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 beased on categories

Only case when

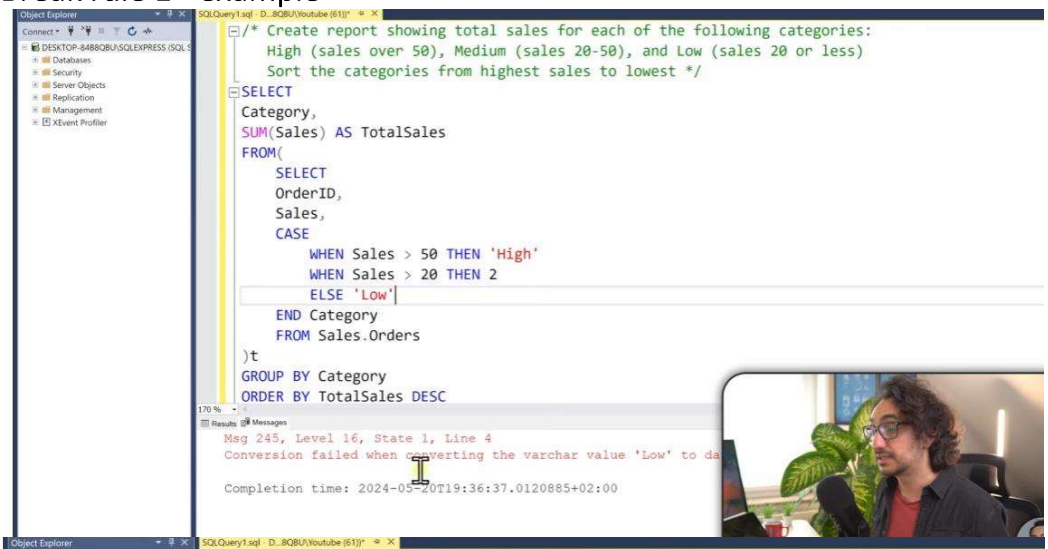

/* Create report showing total sales for each of the following categories:
High (sales over 50), Medium (sales 20-50), and Low (sales 20 or less)
Sort the categories from highest sales to lowest */
SELECT
OrderID,
Sales,
CASE
WHEN Sales > 50 THEN 'High'
WHEN Sales > 20 THEN 'Medium'
ELSE 'Low'
END Category
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

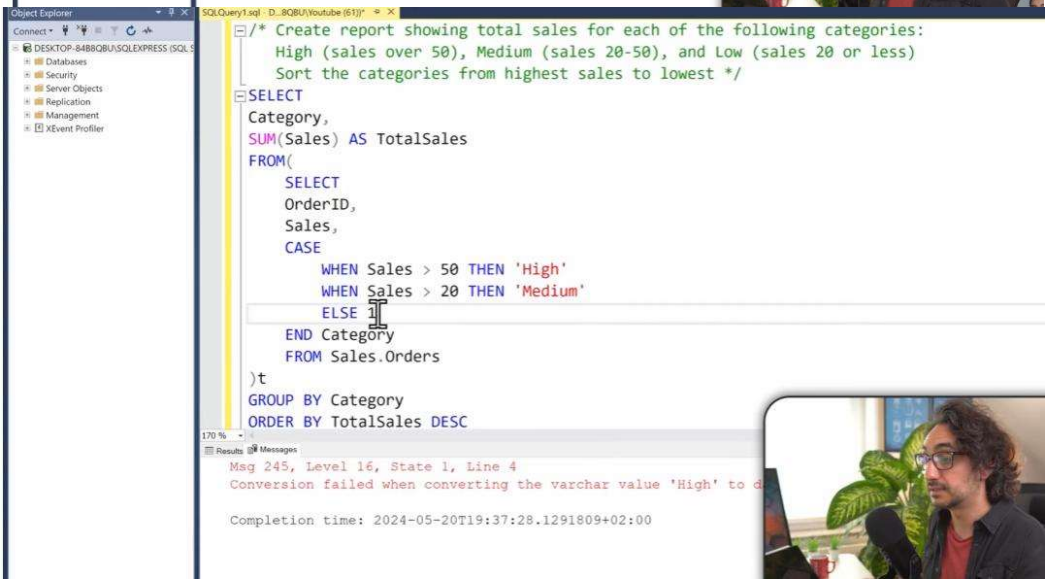

Object Explorer
Connect
DESKTOP-84B8QBU\SQLEXPRESS (SQL S
Databases
Security
Server Objects
Replication
Management
XEvent Profiler
SQLQuery6.sql - D...8QBU\Youtube (54))*
/* Create report showing total sales for each of the following categories:
High (sales over 50), Medium (sales 20-50), and Low (sales 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
129 %
Results
Messages
Category TotalSales
1 High 210
2 Medium 105
3 Low 65
CASE STATEMENT
RULES

Rule 1 :

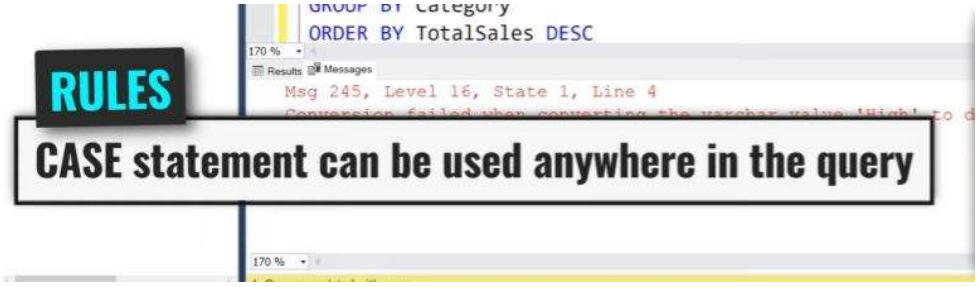
The data type of the results must be matching

Break rule 1 - example

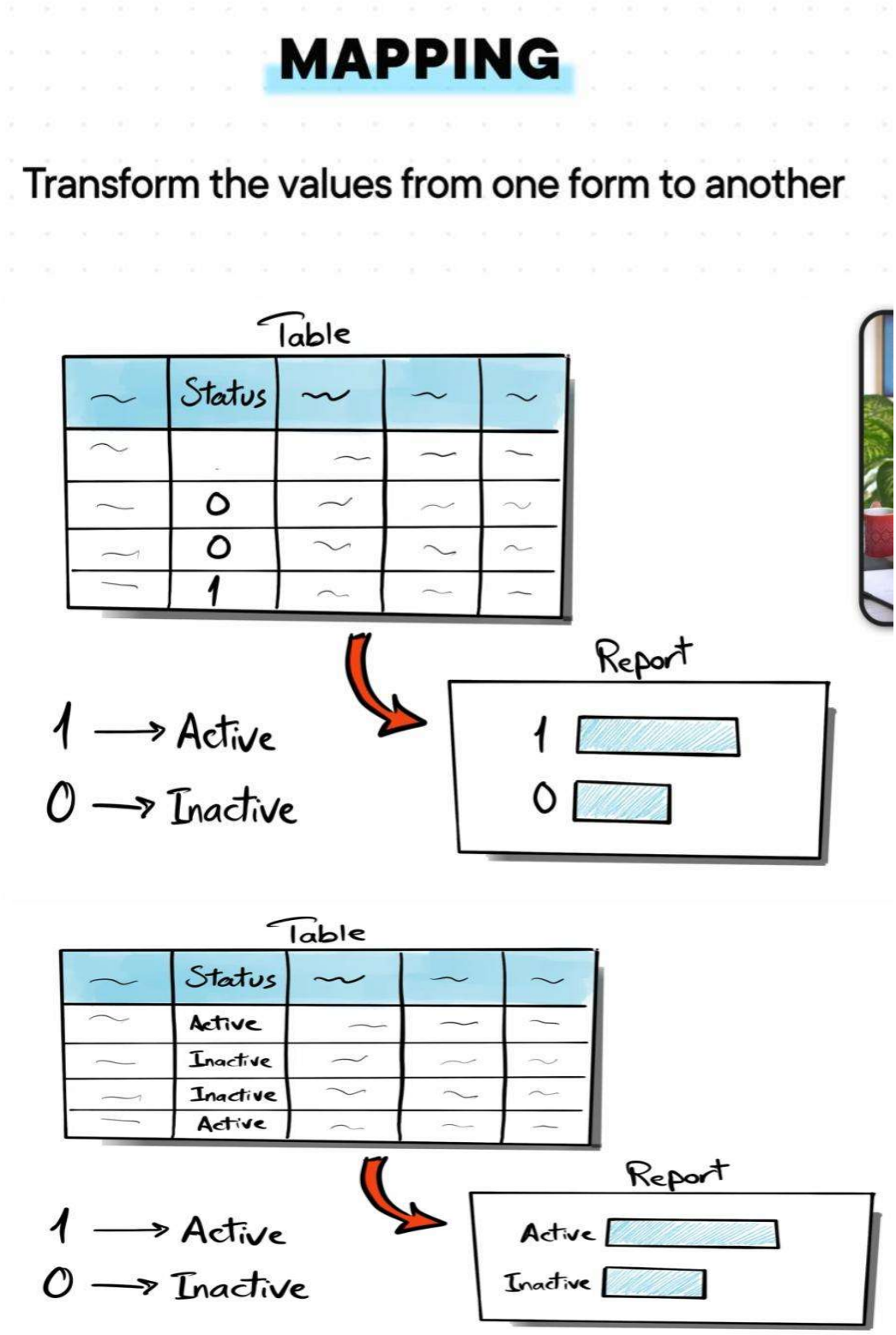


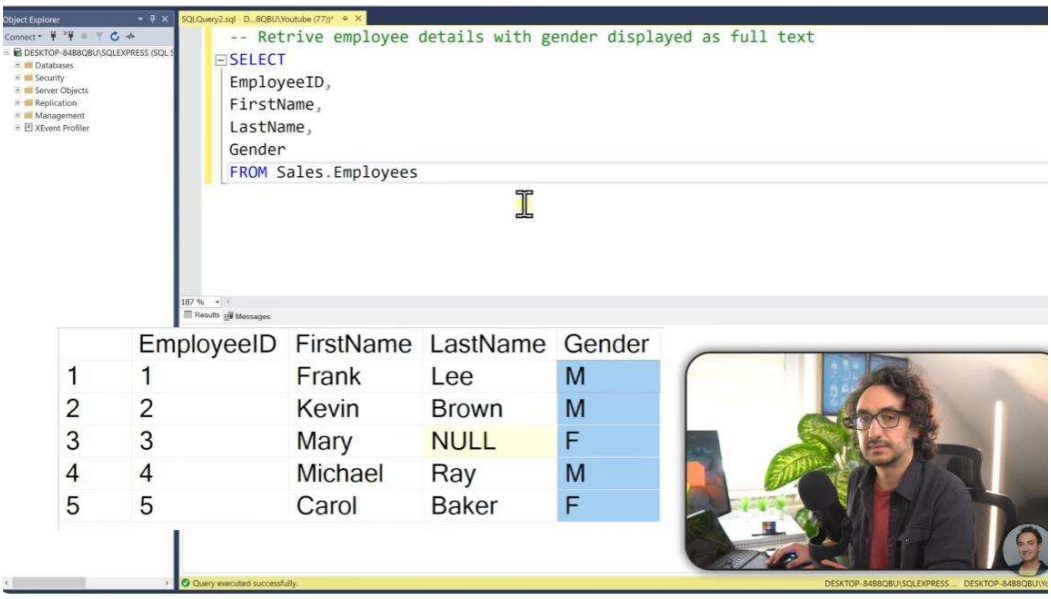


Rule 2 -:

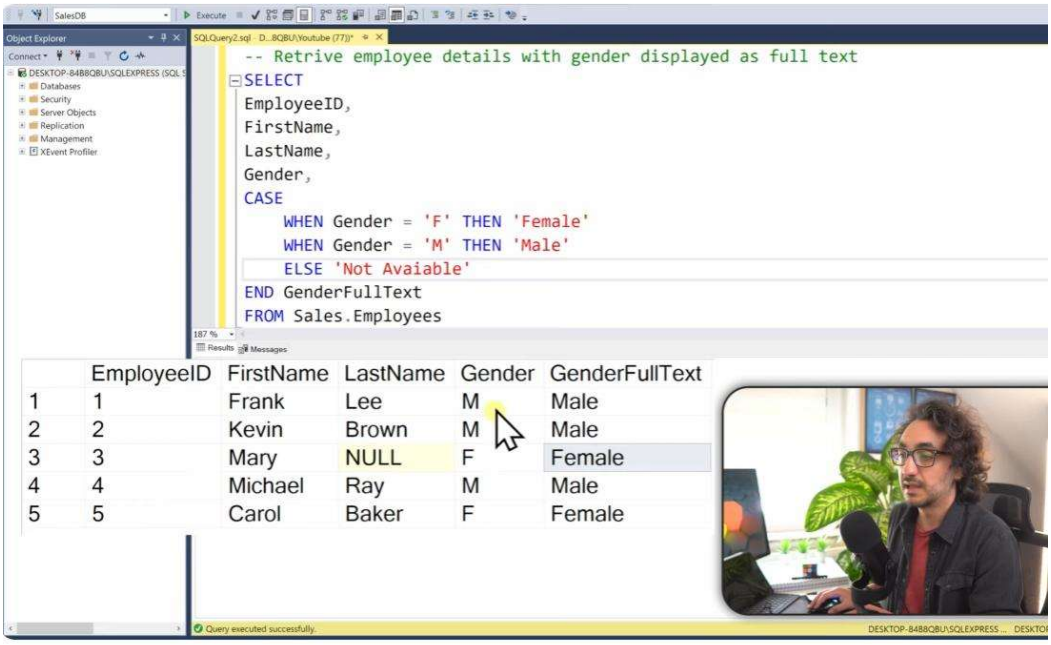


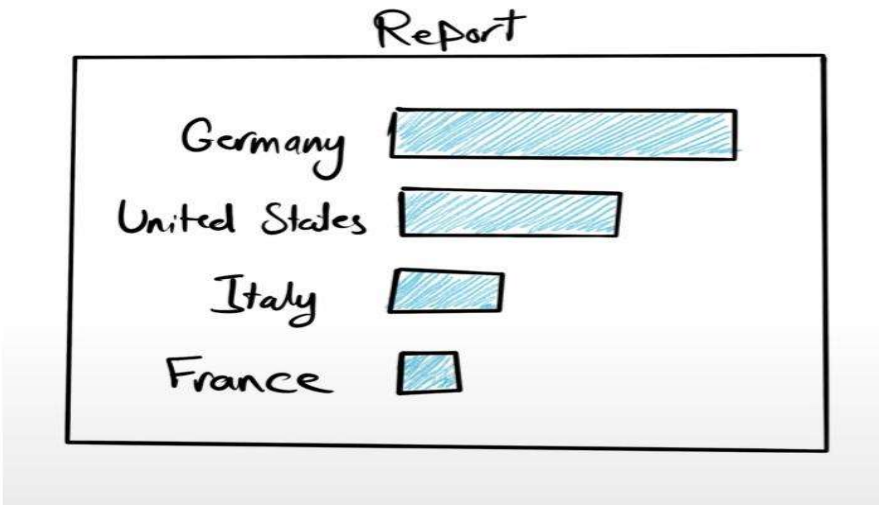
Mapping Values : Usecase of case statement





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





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