### https://chatgpt.com/share/67439a82-85a4-8009-a220-b7529db9a3c0

### Why Use a CASE Statement in SQL?

The CASE statement in SQL is used for **conditional logic** within queries. It allows you to perform conditional transformations, create new calculated columns, or make decisions based on specific conditions. It works like an "if-then-else" statement in programming.

#### Common Use Cases:

1. **Data Transformation**: Convert values in a column to something more meaningful.
2. **Custom Aggregations**: Apply conditions while aggregating data.
3. **Dynamic Categorization**: Assign categories based on conditions.
4. **Simplify Queries**: Perform conditional logic directly in SQL without complex subqueries.

### Syntax of CASE Statement

The basic syntax is:

sql

Copy code

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

...

ELSE resultN

END

* CASE: Begins the statement.
* WHEN: Specifies a condition to evaluate.
* THEN: Specifies the result if the WHEN condition is true.
* ELSE: Provides a default result if no conditions are met (optional).
* END: Marks the end of the CASE statement.

### Example 1: Simple Categorization

Suppose we have a table called Orders:

| OrderID | OrderAmount |
| --- | --- |
| 1 | 500 |
| 2 | 1200 |
| 3 | 300 |
| 4 | 900 |

We want to classify orders into categories based on the OrderAmount:

* "Small" for amounts < 500
* "Medium" for amounts between 500 and 1000
* "Large" for amounts > 1000

**Query:**

sql

Copy code

SELECT

OrderID,

OrderAmount,

CASE

WHEN OrderAmount < 500 THEN 'Small'

WHEN OrderAmount BETWEEN 500 AND 1000 THEN 'Medium'

ELSE 'Large'

END AS OrderCategory

FROM Orders;

**Result:**

| OrderID | OrderAmount | OrderCategory |
| --- | --- | --- |
| 1 | 500 | Medium |
| 2 | 1200 | Large |
| 3 | 300 | Small |
| 4 | 900 | Medium |

### Key Benefits of CASE:

* Versatile and simplifies conditional logic in SQL.
* Useful for creating dynamic values or labels.
* Can be used in SELECT, WHERE, ORDER BY, and other clauses.

### Valid Use Cases and Examples of CASE Statement in SQL

The CASE statement is versatile and can be used in many scenarios for conditional logic in SQL. Below are valid cases with practical examples:

### 1. ****Dynamic Categorization****

Categorize data based on conditions.

#### Example:

Categorize employees based on salary.

**Query:**

sql

Copy code

SELECT

EmployeeID,

Salary,

CASE

WHEN Salary < 3000 THEN 'Low Income'

WHEN Salary BETWEEN 3000 AND 7000 THEN 'Middle Income'

ELSE 'High Income'

END AS IncomeCategory

FROM Employees;

**Result:**

| EmployeeID | Salary | IncomeCategory |
| --- | --- | --- |
| 101 | 2500 | Low Income |
| 102 | 4500 | Middle Income |
| 103 | 9000 | High Income |

### 2. ****Conditional Aggregation****

Apply conditions within aggregate functions.

#### Example:

Count the number of products sold above and below $100.

**Query:**

sql

Copy code

SELECT

SUM(CASE WHEN Price < 100 THEN 1 ELSE 0 END) AS ProductsBelow100,

SUM(CASE WHEN Price >= 100 THEN 1 ELSE 0 END) AS ProductsAbove100

FROM Sales;

**Result:**

| ProductsBelow100 | ProductsAbove100 |
| --- | --- |
| 150 | 75 |

### 3. ****Conditional Updates****

Update data based on specific conditions.

#### Example:

Update the status of orders based on delivery date.

**Query:**

sql

Copy code

UPDATE Orders

SET OrderStatus =

CASE

WHEN DeliveryDate IS NULL THEN 'Pending'

WHEN DeliveryDate <= GETDATE() THEN 'Delivered'

ELSE 'In Transit'

END;

**Effect:**

* Orders with no delivery date → Pending
* Orders delivered on or before today → Delivered
* Future delivery dates → In Transit

### 4. ****Custom Sorting****

Sort results dynamically based on conditions.

#### Example:

Sort customers based on priority status and order count.

**Query:**

sql

Copy code

SELECT

CustomerID,

OrderCount,

CASE

WHEN OrderCount > 50 THEN 1

WHEN OrderCount BETWEEN 20 AND 50 THEN 2

ELSE 3

END AS Priority

FROM Customers

ORDER BY Priority;

**Result:**

| CustomerID | OrderCount | Priority |
| --- | --- | --- |
| 105 | 60 | 1 |
| 102 | 30 | 2 |
| 103 | 15 | 3 |

### 5. ****Data Transformation****

Transform raw data into readable formats.

#### Example:

Convert numerical grades into letter grades.

**Query:**

sql

Copy code

SELECT

StudentID,

Score,

CASE

WHEN Score >= 90 THEN 'A'

WHEN Score >= 80 THEN 'B'

WHEN Score >= 70 THEN 'C'

WHEN Score >= 60 THEN 'D'

ELSE 'F'

END AS Grade

FROM Students;

**Result:**

| StudentID | Score | Grade |
| --- | --- | --- |
| 201 | 85 | B |
| 202 | 72 | C |
| 203 | 95 | A |

### 6. ****Null Value Replacement****

Replace NULL values with a default value.

#### Example:

Replace NULL in product descriptions with "No Description".

**Query:**

sql

Copy code

SELECT

ProductID,

COALESCE(

CASE

WHEN Description IS NULL THEN 'No Description'

ELSE Description

END, 'No Description') AS ProductDescription

FROM Products;

**Result:**

| ProductID | ProductDescription |
| --- | --- |
| 301 | "Premium Watch" |
| 302 | "No Description" |

### 7. ****Conditional Filtering****

Filter rows based on complex conditions.

#### Example:

Select orders based on different conditions for regions.

**Query:**

sql

Copy code

SELECT

OrderID,

Region,

CASE

WHEN Region = 'North' AND OrderAmount > 1000 THEN 'High Value North'

WHEN Region = 'South' AND OrderAmount < 500 THEN 'Low Value South'

ELSE 'Other'

END AS OrderCategory

FROM Orders;

**Result:**

| OrderID | Region | OrderCategory |
| --- | --- | --- |
| 501 | North | High Value North |
| 502 | South | Low Value South |
| 503 | East | Other |

### 8. ****Custom Error Handling****

Display custom messages for invalid data.

#### Example:

Mark invalid ages in a dataset.

**Query:**

sql

Copy code

SELECT

PersonID,

Age,

CASE

WHEN Age < 0 THEN 'Invalid Age'

WHEN Age > 120 THEN 'Invalid Age'

ELSE 'Valid Age'

END AS AgeStatus

FROM People;

**Result:**

| PersonID | Age | AgeStatus |
| --- | --- | --- |
| 1 | 25 | Valid Age |
| 2 | -5 | Invalid Age |
| 3 | 130 | Invalid Age |

### Summary of Use Cases:

1. Categorization (e.g., Low/Medium/High income).
2. Aggregation with conditions.
3. Conditional updates for specific rows.
4. Dynamic sorting.
5. Data transformation or formatting.
6. Replacing NULL values.
7. Filtering complex data sets.
8. Error detection or custom messages.

The CASE statement is a powerful tool that simplifies many complex operations in SQL queries.

what is meant by COALESCE say it crisply.

###### ChatGPT said:

COALESCE is a SQL function that returns the **first non-NULL value** from a list of expressions. It is often used to handle NULL values by providing a default value.

### Syntax:

sql

Copy code

COALESCE(expression1, expression2, ..., expressionN)

* Evaluates the expressions in order.
* Returns the first non-NULL value.
* If all values are NULL, it returns NULL.

### Example:

#### Table: Users

| UserID | FirstName | MiddleName | LastName |
| --- | --- | --- | --- |
| 1 | John | NULL | Doe |
| 2 | NULL | Michael | Smith |
| 3 | NULL | NULL | Brown |

#### Query:

sql

Copy code

SELECT

UserID,

COALESCE(FirstName, MiddleName, LastName, 'Unknown') AS DisplayName

FROM Users;

#### Result:

| UserID | DisplayName |
| --- | --- |
| 1 | John |
| 2 | Michael |
| 3 | Brown |

### Key Points:

* **Efficient for handling NULL values**.
* Often used for fallback or default value logic.

##### You said:

SELECT OrderID, COALESCE(CustomerName, CASE WHEN CustomerName IS NULL THEN 'Unknown Customer' END) AS CustomerName FROM Orders; what is the output for this

###### ChatGPT said:

The SQL query provided is **unnecessary and incorrect** because the COALESCE function is redundant and conflicts with the CASE logic. Let's break this down:

### Query:

sql

Copy code

SELECT

OrderID,

COALESCE(CustomerName,

CASE

WHEN CustomerName IS NULL THEN 'Unknown Customer'

END) AS CustomerName

FROM Orders;

### Why It Is Incorrect:

1. **COALESCE with a CASE Statement:**
   * COALESCE(CustomerName, ...) already checks if CustomerName is NULL.
   * The CASE block inside COALESCE is redundant because it's doing the same check (WHEN CustomerName IS NULL).
2. **Output Conflict:**
   * If CustomerName is NULL, COALESCE will already handle it. The CASE logic is redundant and doesn't add value.
   * This makes the query unnecessarily complicated.

### Simplified and Correct Query:

sql

Copy code

SELECT

OrderID,

COALESCE(CustomerName, 'Unknown Customer') AS CustomerName

FROM Orders;

### Result:

#### Assuming Table Orders:

| OrderID | CustomerName |
| --- | --- |
| 1 | John Doe |
| 2 | NULL |
| 3 | Alice |
| 4 | NULL |

#### Output:

| OrderID | CustomerName |
| --- | --- |
| 1 | John Doe |
| 2 | Unknown Customer |
| 3 | Alice |
| 4 | Unknown Customer |

### Key Takeaway:

Use COALESCE alone to handle NULL values directly. Combining it with CASE for the same condition is redundant.

##### You said:

SELECT SUM(CASE WHEN Price < 100 THEN 1 ELSE 0 END) AS ProductsBelow100, SUM(CASE WHEN Price >= 100 THEN 1 ELSE 0 END) AS ProductsAbove100 FROM Sales; I am saying my understanding correct me if I am wrong. There is a column named Price which contains values greater than and less than 100. The first statement is when the price is less than 100 add 1 in the ProductBelow100 and the second statement says add 1 if the Product is above 100.

###### ChatGPT said:

Your understanding is mostly correct, but let me refine it for clarity:

* The CASE statement is checking the condition for each row in the Sales table, specifically in the Price column.
* The SUM function is then summing up the results of these checks (1 for true, 0 for false) to count the number of rows that meet each condition.

### Breakdown of the Query:

#### Query:

sql

Copy code

SELECT

SUM(CASE WHEN Price < 100 THEN 1 ELSE 0 END) AS ProductsBelow100,

SUM(CASE WHEN Price >= 100 THEN 1 ELSE 0 END) AS ProductsAbove100

FROM Sales;

#### Explanation:

1. CASE WHEN Price < 100 THEN 1 ELSE 0 END:
   * For each row:
     + If Price is less than 100, the CASE statement returns 1.
     + Otherwise, it returns 0.
   * This creates a temporary column of 1s and 0s for rows where the price is below 100.
   * SUM adds up all the 1s in this temporary column to count how many products have a Price < 100.
2. CASE WHEN Price >= 100 THEN 1 ELSE 0 END:
   * Similarly:
     + If Price is greater than or equal to 100, the CASE statement returns 1.
     + Otherwise, it returns 0.
   * SUM adds up all the 1s to count how many products have a Price >= 100.
3. **Result Columns**:
   * ProductsBelow100: The total count of rows where Price < 100.
   * ProductsAbove100: The total count of rows where Price >= 100.

### Example Input Data (Sales Table):

| SaleID | Price |
| --- | --- |
| 1 | 50 |
| 2 | 200 |
| 3 | 75 |
| 4 | 150 |

### Step-by-Step Execution:

1. **Temporary Columns from** CASE:

| SaleID | Price | ProductsBelow100 (Temp) | ProductsAbove100 (Temp) |
| --- | --- | --- | --- |
| 1 | 50 | 1 | 0 |
| 2 | 200 | 0 | 1 |
| 3 | 75 | 1 | 0 |
| 4 | 150 | 0 | 1 |

1. **Sum of Temporary Columns**:

| ProductsBelow100 | ProductsAbove100 |
| --- | --- |
| 2 | 2 |

### Final Output:

| ProductsBelow100 | ProductsAbove100 |
| --- | --- |
| 2 | 2 |

### Refined Understanding:

* The query doesn't "add 1 in the column." Instead:
  + It evaluates the condition for each row.
  + If the condition is true, the CASE returns 1; otherwise, it returns 0.
  + These 1s and 0s are summed up to count the rows meeting the condition.

You are correct in spirit, but now you have a more precise understanding of how it works! Let me know if you have further questions. 😊

##### You said:

Select Sum(CASE WHEN Price > 100 Then 1 Else 0 END) AS PRICE\_BELOW\_100 I will say the order of execution correct me if i am wrong. First From sales will execute then Case statement will execute then sum and finally select.

###### ChatGPT said:

Your understanding of the order of execution is close, but let me refine it for clarity. In SQL, the query execution follows a well-defined **logical order of operations** that is slightly different from the way we write it. Here's the correct order of execution for your query:

### Query:

sql

Copy code

SELECT

SUM(CASE WHEN Price > 100 THEN 1 ELSE 0 END) AS PRICE\_BELOW\_100

FROM Sales;

### Correct Order of Execution:

1. FROM **Clause**:
   * The table Sales is identified, and all rows are selected for processing.
2. CASE **Statement (Part of** SELECT**)**:
   * For each row in the table, the CASE statement is evaluated:
     + If Price > 100, it returns 1.
     + Otherwise, it returns 0.
3. SUM **Aggregation**:
   * The results from the CASE (which are 1s and 0s) are summed up to produce the total count where Price > 100.
4. SELECT **Clause**:
   * The final result of the SUM function is retrieved and presented as PRICE\_BELOW\_100.

### Key Points to Note:

* The CASE statement operates on each row individually, evaluating conditions as rows are processed.
* **Aggregations (like** SUM**)** operate after the CASE has produced its results for all rows.
* Finally, the SELECT clause formats and presents the output.

### Refined Understanding:

* Yes, the FROM **clause** executes first, as you mentioned.
* The CASE **statement** operates on the rows fetched by the FROM clause.
* Then, the SUM **function** aggregates the values resulting from the CASE statement.
* Finally, the SELECT **clause** displays the aggregated result.

You were correct in spirit! This refined explanation aligns with SQL’s logical processing order. Let me know if you have more questions! 😊

##### You said:

UPDATE Orders SET OrderStatus = CASE WHEN DeliveryDate IS NULL THEN 'Pending' WHEN DeliveryDate <= GETDATE() THEN 'Delivered' ELSE 'In Transit' END; I cant understand what is this GETDATE() function is doing and i don't know how the DeliveryDate column will look like please explain me crisply with some clarity.

###### ChatGPT said:

Let me explain the query step by step with clarity:

### ****What is**** GETDATE()****?****

* GETDATE() is a SQL Server function that returns the **current date and time** of the system where the SQL Server is running.
* It is often used for comparisons, filtering, or logging the current date/time in queries.

### ****Understanding the Query:****

sql

Copy code

UPDATE Orders

SET OrderStatus =

CASE

WHEN DeliveryDate IS NULL THEN 'Pending'

WHEN DeliveryDate <= GETDATE() THEN 'Delivered'

ELSE 'In Transit'

END;

1. **Purpose of the Query**:
   * Updates the OrderStatus column in the Orders table based on the value of the DeliveryDate column and the current date/time (GETDATE()).
2. **How It Works**:
   * WHEN DeliveryDate IS NULL:
     + If the DeliveryDate column is NULL (no delivery date recorded), the status is updated to 'Pending'.
   * WHEN DeliveryDate <= GETDATE():
     + If the delivery date is **less than or equal to** the current date/time, it means the delivery has already happened or is happening today. The status is updated to 'Delivered'.
   * ELSE:
     + If the delivery date is **greater than** the current date (i.e., a future date), the status is updated to 'In Transit'.

### ****How the**** DeliveryDate ****Column Might Look Like:****

#### Example Data Before Update:

| OrderID | DeliveryDate | OrderStatus |
| --- | --- | --- |
| 1 | NULL | NULL |
| 2 | 2024-11-20 10:00:00 | NULL |
| 3 | 2024-11-24 15:00:00 | NULL |
| 4 | 2024-11-26 08:00:00 | NULL |

#### Assuming Current Date is 2024-11-24 (from GETDATE()):

* **Row 1**: DeliveryDate = NULL → Status becomes 'Pending'.
* **Row 2**: DeliveryDate < 2024-11-24 → Status becomes 'Delivered'.
* **Row 3**: DeliveryDate = 2024-11-24 → Status becomes 'Delivered'.
* **Row 4**: DeliveryDate > 2024-11-24 → Status becomes 'In Transit'.

#### Example Data After Update:

| OrderID | DeliveryDate | OrderStatus |
| --- | --- | --- |
| 1 | NULL | Pending |
| 2 | 2024-11-20 10:00:00 | Delivered |
| 3 | 2024-11-24 15:00:00 | Delivered |
| 4 | 2024-11-26 08:00:00 | In Transit |

### ****Key Points to Remember****:

* GETDATE() dynamically fetches the current date/time.
* The DeliveryDate column likely stores a date/time value for when the delivery is scheduled or occurred.
* The query updates the OrderStatus based on whether the delivery is in the past, today, or in the future.

Let me know if you need more details! 😊

##### You said:

Select ProductID, COALESCE(Description, 'No Description') From Product; we can write this query instead of this right? SELECT ProductID, COALESCE( CASE WHEN Description IS NULL THEN 'No Description' ELSE Description END, 'No Description') AS ProductDescription FROM Products;

###### ChatGPT said:

Yes, you're correct that the first query is simpler and more appropriate for the task, and the second query is unnecessarily redundant. Let me explain why.

### Query 1 (Simple and Correct):

sql

Copy code

SELECT

ProductID,

COALESCE(Description, 'No Description') AS ProductDescription

FROM Products;

* **What it does**:
  + It checks if Description is NULL.
  + If Description is NULL, it substitutes 'No Description'.
  + Otherwise, it simply returns the value of Description.

### Query 2 (Unnecessarily Redundant):

sql

Copy code

SELECT

ProductID,

COALESCE(

CASE

WHEN Description IS NULL THEN 'No Description'

ELSE Description

END, 'No Description') AS ProductDescription

FROM Products;

* **What it does**:
  + The CASE statement already checks if Description is NULL and substitutes 'No Description' if true.
  + The COALESCE function does the same check again, making the CASE redundant.
  + It's overcomplicated and redundant because the COALESCE function alone handles NULL values efficiently.

### Why Query 1 is Better:

1. **Simpler**: COALESCE is specifically designed to handle NULL substitution, so no need for additional logic.
2. **Readable**: The intent of replacing NULL values is clear and concise.
3. **Efficient**: Avoids unnecessary computations like evaluating a CASE block.

### Recommendation:

Always prefer the first query for simplicity and performance:

sql

Copy code

SELECT

ProductID,

COALESCE(Description, 'No Description') AS ProductDescription

FROM Products;

It achieves the same result in a cleaner and more efficient way. Let me know if you'd like further clarification! 😊

##### You said:

Okay, I think i studied the Case statement clearly. Now test my understanding with some questions, if you want me to write queries show me the table and ask me a question and also say what will be the output of my code if wrong correct it.

###### ChatGPT said:

Great! Let's test your understanding with some scenarios. I’ll provide tables and questions for you to write queries. After you write your query, I’ll evaluate it and correct it if needed.

### ****Table:**** Orders

| OrderID | CustomerName | OrderAmount | OrderDate | DeliveryDate |
| --- | --- | --- | --- | --- |
| 1 | Alice | 250 | 2024-11-20 | 2024-11-25 |
| 2 | Bob | 500 | 2024-11-22 | 2024-11-24 |
| 3 | NULL | 750 | 2024-11-23 | NULL |
| 4 | Charlie | 1200 | 2024-11-18 | 2024-11-22 |
| 5 | NULL | 100 | 2024-11-24 | 2024-11-30 |

### ****Question 1: Categorize Orders Based on OrderAmount****

Write a query to create a new column called OrderCategory:

* "Small Order" if OrderAmount < 300.
* "Medium Order" if OrderAmount >= 300 AND OrderAmount <= 800.
* "Large Order" if OrderAmount > 800.

### ****Question 2: Replace NULL CustomerName****

Write a query to replace NULL in the CustomerName column with 'Guest' and create a new column called UpdatedCustomerName.

### ****Question 3: Check Delivery Status****

Write a query to create a new column called DeliveryStatus:

* "Pending" if DeliveryDate is NULL.
* "Delivered" if DeliveryDate <= '2024-11-24'.
* "In Transit" if DeliveryDate > '2024-11-24'.

### ****Expected Output Examples****

#### Question 1 Output:

| OrderID | OrderAmount | OrderCategory |
| --- | --- | --- |
| 1 | 250 | Small Order |
| 2 | 500 | Medium Order |
| 3 | 750 | Medium Order |
| 4 | 1200 | Large Order |
| 5 | 100 | Small Order |

#### Question 2 Output:

| OrderID | CustomerName | UpdatedCustomerName |
| --- | --- | --- |
| 1 | Alice | Alice |
| 2 | Bob | Bob |
| 3 | NULL | Guest |
| 4 | Charlie | Charlie |
| 5 | NULL | Guest |

#### Question 3 Output:

| OrderID | DeliveryDate | DeliveryStatus |
| --- | --- | --- |
| 1 | 2024-11-25 | In Transit |
| 2 | 2024-11-24 | Delivered |
| 3 | NULL | Pending |
| 4 | 2024-11-22 | Delivered |
| 5 | 2024-11-30 | In Transit |

Write your queries! If you're wrong, I'll correct you and explain why. 😊

##### You said:

Question1: Select \* Case ()

###### ChatGPT said:

It seems like you're trying to begin writing

##### You said:

Question1: Select OrderID, OrderAmount Case when OrderAmount < 300 THEN 'Small Order', when OrderAmount BETWEEN 300 AND 800 THEN 'Medium Order', Else 'Large Order', END AS OrderCategory FROM Orders; Ouestion 2: Select OrderID, CustomerName Coelsce(CustomerName, 'Guest') AS UpdatedCustomerName From Orders; Question3: Select OrderID, DeleiveryDate CASE WHEN DeliveryDate = NULL Then 'Pending', WHEN DeliveryDate <= '2024-11-24' THEN 'Delivered', WHEN DeliveryDate >= '2024-11-24' THEN 'In Transit' FROM Orders;

###### ChatGPT said:

Let’s review your queries one by one, identify errors, and provide corrected versions along with explanations.

### ****Question 1: Your Query****

sql

Copy code

Select OrderID, OrderAmount

Case

when OrderAmount < 300 THEN 'Small Order',

when OrderAmount BETWEEN 300 AND 800 THEN 'Medium Order',

Else 'Large Order',

END AS OrderCategory

FROM Orders;

#### Issues:

1. CASE **Syntax Error**:
   * The CASE statement should not have commas (,) after each THEN clause or the ELSE clause.
   * Replace commas with proper syntax.
2. **Formatting**:
   * Missing a comma between OrderID, OrderAmount and the CASE statement.

#### Correct Query:

sql

Copy code

SELECT

OrderID,

OrderAmount,

CASE

WHEN OrderAmount < 300 THEN 'Small Order'

WHEN OrderAmount BETWEEN 300 AND 800 THEN 'Medium Order'

ELSE 'Large Order'

END AS OrderCategory

FROM Orders;

#### Expected Output:

| OrderID | OrderAmount | OrderCategory |
| --- | --- | --- |
| 1 | 250 | Small Order |
| 2 | 500 | Medium Order |
| 3 | 750 | Medium Order |
| 4 | 1200 | Large Order |
| 5 | 100 | Small Order |

### ****Question 2: Your Query****

sql

Copy code

Select OrderID, CustomerName

Coelsce(CustomerName, 'Guest') AS UpdatedCustomerName

From Orders;

#### Issues:

1. **Typo in** COALESCE:
   * The function name is misspelled as Coelsce. It should be COALESCE.
2. **Missing Comma**:
   * There’s no comma between OrderID, CustomerName and COALESCE.

#### Correct Query:

sql

Copy code

SELECT

OrderID,

CustomerName,

COALESCE(CustomerName, 'Guest') AS UpdatedCustomerName

FROM Orders;

#### Expected Output:

| OrderID | CustomerName | UpdatedCustomerName |
| --- | --- | --- |
| 1 | Alice | Alice |
| 2 | Bob | Bob |
| 3 | NULL | Guest |
| 4 | Charlie | Charlie |
| 5 | NULL | Guest |

### ****Question 3: Your Query****

sql

Copy code

Select OrderID, DeleiveryDate

CASE

WHEN DeliveryDate = NULL Then 'Pending',

WHEN DeliveryDate <= '2024-11-24' THEN 'Delivered',

WHEN DeliveryDate >= '2024-11-24' THEN 'In Transit'

FROM Orders;

#### Issues:

1. **Column Name Typo**:
   * DeleiveryDate is misspelled. It should be DeliveryDate.
2. NULL **Comparison**:
   * You cannot compare NULL with =. Use IS NULL instead.
3. **Extra Comma**:
   * Commas are not allowed in the CASE block between conditions.
4. **Missing** END **for** CASE:
   * Every CASE statement must end with END.
5. **Formatting**:
   * Missing a comma between OrderID and CASE.

#### Correct Query:

sql

Copy code

SELECT

OrderID,

DeliveryDate,

CASE

WHEN DeliveryDate IS NULL THEN 'Pending'

WHEN DeliveryDate <= '2024-11-24' THEN 'Delivered'

ELSE 'In Transit'

END AS DeliveryStatus

FROM Orders;

#### Expected Output:

| OrderID | DeliveryDate | DeliveryStatus |
| --- | --- | --- |
| 1 | 2024-11-25 | In Transit |
| 2 | 2024-11-24 | Delivered |
| 3 | NULL | Pending |
| 4 | 2024-11-22 | Delivered |
| 5 | 2024-11-30 | In Transit |

### Summary of Corrections:

1. **Question 1**:
   * Fix CASE syntax by removing commas and adding proper formatting.
2. **Question 2**:
   * Fix typo in COALESCE.
   * Add proper commas between columns.
3. **Question 3**:
   * Fix NULL comparison using IS NULL.
   * Correct the CASE syntax and include END.

Let me know if you have additional questions or need further clarification! 😊