SUMMARY: CASE STATEMENTS

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand the CASE statements in SQL
- Understand how CASE WHEN can be utilized to create new segments and new columns/flags based on existing columns.
- Understand how case statements can be used for conditional aggregation.

KEY TOPICS AND EXAMPLES:

Understanding the CASE statements in SQL:

MySQL CASE expression is a part of the control flow function that allows us to write an if-else or if-then-else logic to a query. This expression can be used anywhere that uses a valid program or query, such as SELECT, WHERE, ORDER BY clause, etc.

The CASE expression validates various conditions and returns the result when the first condition is true. Once the condition is met, it stops traversing and gives the output. If it does not find any condition true, it executes the else block. When the else block is not found, it returns a NULL value. The main goal of the MySQL CASE statement is to deal with multiple IF statements in the SELECT clause.

Analogy:

Imagine you're a waiter in a restaurant, and your job is to recommend a dessert to customers based on certain conditions or preferences. The CASE statement in MySQL can be represented as the thought process you go through to make the recommendation.
You can have two types of scenarios:

- **Simple Scenario:** This is like considering only the customer's choice of the main course to recommend a dessert. Based on their main course selection, you recommend a specific dessert.

- **Complex Scenario:** This is like considering multiple factors or conditions to recommend a dessert, rather than just the main course. You might take into account whether the customer is vegetarian, vegan, or their age group, and then recommend an appropriate dessert based on these conditions.

The CASE statement in MySQL works similarly. It evaluates a set of conditions or expressions, and based on the first condition that evaluates to true, it returns the corresponding result or value. If none of the conditions are true, it can return a default value.
Just like a waiter follows a decision tree or a set of rules to recommend a dessert based on customer preferences or conditions, the CASE statement in MySQL follows a series of conditional statements to return a specific value or perform a specific action based on the conditions you provide.
The CASE statement is particularly useful in scenarios where you need to categorize data, handle null

values, transform data from one format to another, or perform conditional operations based on specific criteria. It provides a concise and readable way to implement complex conditional logic within SQL queries, making the code more maintainable and easier to understand.

Understanding conditional Value Assignment:

One of the primary uses of the CASE statement is to conditionally assign values based on certain conditions. This is useful when you need to transform data or categorize it based on specific criteria.

Syntax:

```
SELECT column1, column2,
      CASE
            WHEN condition1 THEN result1
            WHEN condition2 THEN result2
            ELSE resultN
      END AS alias_name
FROM table_name;
```

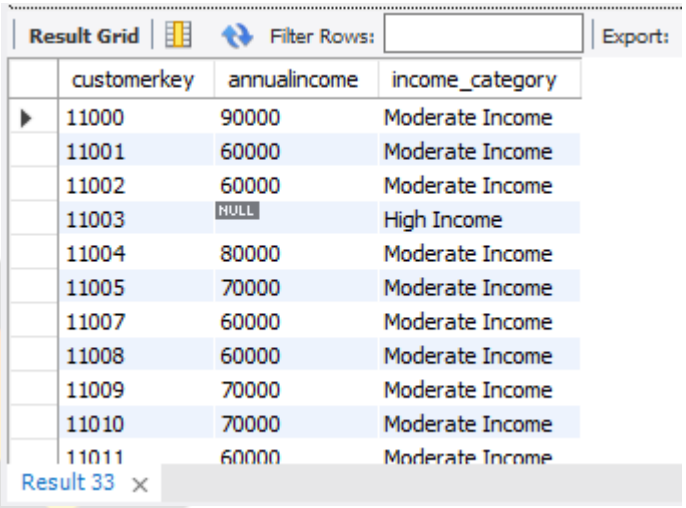
NOTE:

- *condition1, condition2: These are the conditions evaluated sequentially.*
- *result1, result2, resultN: These are the results returned when corresponding conditions are met.*
- *alias_name: This is the name given to the resultant column.*

Example 1:

```
SELECT customerkey, annualincome,
    CASE
        WHEN annualincome < 50000 THEN 'Low Income'
        WHEN annualincome BETWEEN 50000 AND 100000 THEN 'Moderate
Income'
        ELSE 'High Income'
    END AS income_category
FROM customers;
```

Output:



Result Grid | Filter Rows: | Export:

	customerkey	annualincome	income_category
▶	11000	90000	Moderate Income
	11001	60000	Moderate Income
	11002	60000	Moderate Income
	11003	NULL	High Income
	11004	80000	Moderate Income
	11005	70000	Moderate Income
	11007	60000	Moderate Income
	11008	60000	Moderate Income
	11009	70000	Moderate Income
	11010	70000	Moderate Income
	11011	60000	Moderate Income

Result 33 ×

NOTE: The above output might not contain all the returned rows. Also, we can notice an ambiguity in the output. The customers with NULL income are also showing High Income which might not be the best way to represent and this might affect the analysis. Thus, we move to the next use case.

Understanding handling of NULL Values using CASE statement:

The CASE statement can be helpful in handling NULL values in a more meaningful way. You can use it to provide a default value or a specific value when a column or expression evaluates to NULL.

Syntax:

```
SELECT column1, column2,
      CASE
            WHEN column_name IS NULL THEN 'Replacement Value'
            ELSE column_name
      END AS alias_name
FROM table_name;
```

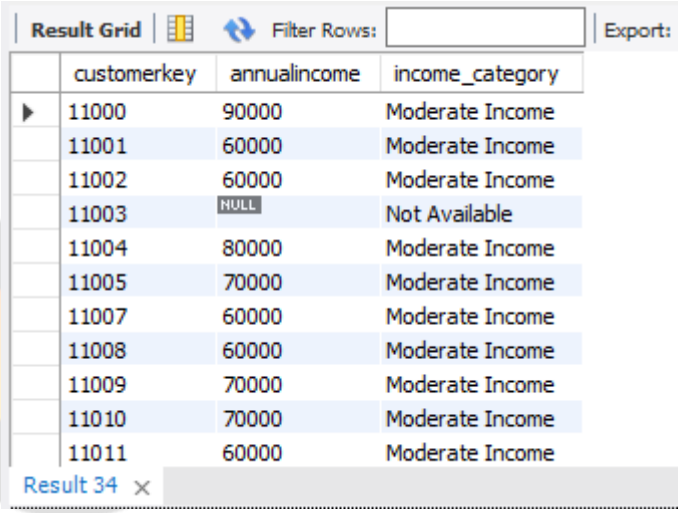
NOTE:

- ***WHEN column_name IS NULL:*** *Checks if the column value is NULL.*
- ***THEN replacement_value:*** *Specifies the value to use if the column value is NULL.*
- ***ELSE column_name:*** *Specifies the value to use if the column value is not NULL.*
- ***AS alias_name:*** *Provides an alias for the resultant column.*

Example 1:

```
SELECT customerkey, annualincome,
    CASE
        WHEN annualincome < 50000 THEN 'Low Income'
        WHEN annualincome BETWEEN 50000 AND 100000 THEN 'Moderate
Income'
        WHEN annualincome IS NULL THEN "Not Available"
        ELSE 'High Income'
    END AS income_category
FROM customers;
```

Output:



Result Grid | Filter Rows: | Export:

customerkey	annualincome	income_category
11000	90000	Moderate Income
11001	60000	Moderate Income
11002	60000	Moderate Income
11003	NULL	Not Available
11004	80000	Moderate Income
11005	70000	Moderate Income
11007	60000	Moderate Income
11008	60000	Moderate Income
11009	70000	Moderate Income
11010	70000	Moderate Income
11011	60000	Moderate Income

Result 34

NOTE: The above output might not contain all the returned rows. In the above output, we can notice the ambiguity that we faced in the previous step is now removed and the NULL values are not considered as High Income.

Understanding the updating of a table using CASE statement:

In MySQL, you can use the CASE statement in an UPDATE query to conditionally update column values based on specific conditions. The most common uses of the CASE statement in UPDATE queries is to update status or category columns based on specific conditions. For example, you can update the status of an order based on the order date, shipping date, or payment status.

Syntax:

```
UPDATE table_name
SET column1 = CASE
                  WHEN condition1 THEN value1
                  WHEN condition2 THEN value2
                  ELSE default_value
              END,
     column2 = CASE
                  WHEN condition1 THEN value3
                  WHEN condition2 THEN value4
                  ELSE default_value
              END
WHERE condition;
```

Example 1:

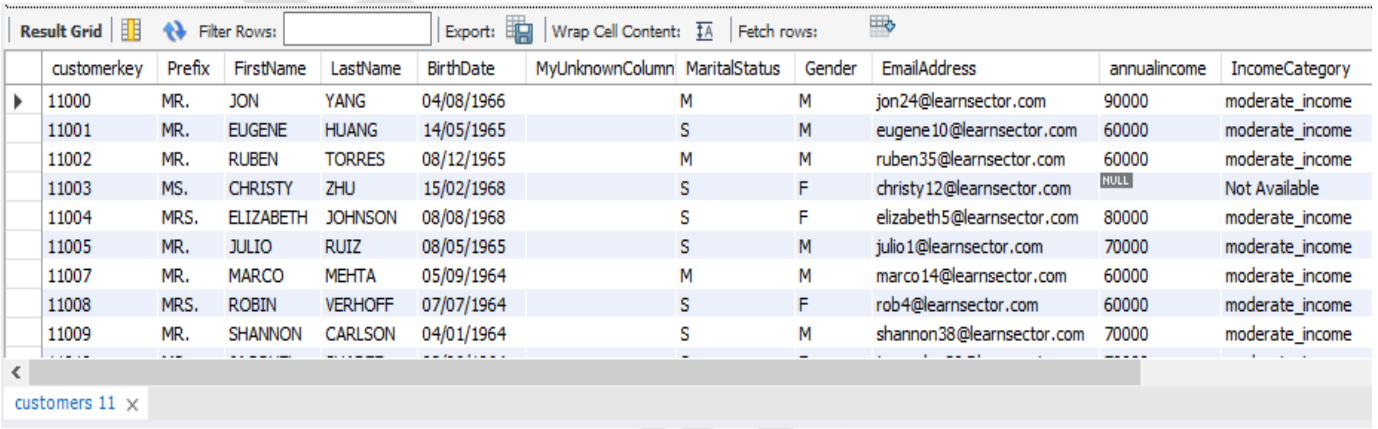
The below query will help us permanently insert a separate column after the annual income column which will create buckets according to the specified conditions as mentioned and will set the NULL values as Not Available.

```
ALTER TABLE customers
ADD COLUMN IncomeCategory varchar(50)
AFTER annualincome;

UPDATE customers
SET IncomeCategory = CASE
                WHEN annualincome < 30000 THEN "low_income"
                WHEN annualincome BETWEEN 30000 and 100000 THEN
"moderate_income"
                WHEN annualincome IS NULL THEN "Not Available"
                ELSE "high_income"
            END;

SELECT * FROM customers;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

customerkey	Prefix	FirstName	LastName	BirthDate	MyUnknownColumn	MaritalStatus	Gender	EmailAddress	annualincome	IncomeCategory
11000	MR.	JON	YANG	04/08/1966		M	M	jon24@learnsector.com	90000	moderate_income
11001	MR.	EUGENE	HUANG	14/05/1965		S	M	eugene10@learnsector.com	60000	moderate_income
11002	MR.	RUBEN	TORRES	08/12/1965		M	M	ruben35@learnsector.com	60000	moderate_income
11003	MS.	CHRISTY	ZHU	15/02/1968		S	F	christy12@learnsector.com	NULL	Not Available
11004	MRS.	ELIZABETH	JOHNSON	08/08/1968		S	F	elizabeth5@learnsector.com	80000	moderate_income
11005	MR.	JULIO	RUIZ	08/05/1965		S	M	julio1@learnsector.com	70000	moderate_income
11007	MR.	MARCO	MEHTA	05/09/1964		M	M	marco14@learnsector.com	60000	moderate_income
11008	MRS.	ROBIN	VERHOFF	07/07/1964		S	F	rob4@learnsector.com	60000	moderate_income
11009	MR.	SHANNON	CARLSON	04/01/1964		S	M	shannon38@learnsector.com	70000	moderate_income

customers 11

Understanding conditional aggregation using CASE statement:

Conditional aggregation using a CASE statement in SQL allows you to perform aggregate functions based on specific conditions within a dataset. This technique is especially useful when you need to apply different aggregation rules depending on certain criteria.

How It Works

- **CASE Statement Integration:** The CASE statement is used within the aggregate function to specify different conditions. Depending on whether these conditions are met, different values can be included or excluded from the aggregation.

- **Aggregate Functions**: Common aggregate functions used with CASE statements include SUM, COUNT, AVG, MIN, and MAX. These functions perform their respective operations on the values that meet the conditions specified in the CASE statement.

- **Condition Specification:** Within the CASE statement, you can specify multiple conditions. Each condition is evaluated, and depending on the result (true or false),

the value is either included in or excluded from the aggregate calculation.

- **Grouping and Aggregation:** The results are typically grouped by one or more columns using the GROUP BY clause. This ensures that the aggregate calculations are performed for each group of data separately.

Syntax:

```
SELECT column1, column2,
      AGGREGATE_FUNCTION(CASE WHEN condition1 THEN value1 WHEN
condition2 THEN value2
            ELSE default_value END) AS alias_name,
    -- You can have multiple such conditional aggregations
      AGGREGATE_FUNCTION(CASE WHEN conditionA THEN valueA WHEN
conditionB THEN valueB
        ...
        ELSE default_value
    END) AS another_alias_name
FROM table_name
GROUP BY column1, column2
```

NOTE:

- ***AGGREGATE_FUNCTION:*** *This is where you specify the aggregate function you want to use, such as SUM, COUNT, AVG, MIN, MAX.*
- ***CASE Statement:*** *The CASE statement defines the conditions and the values to be used based on those conditions.*
- ***Conditions:*** *These are the logical conditions that determine which value is used in the aggregate function.*
- ***Values:*** *These are the values that are aggregated if the corresponding condition is true.*
- ***Default Value:*** *This value is used when none of the specified conditions are met.*
- ***Alias****: An alias name for the resulting column, which makes the output easier to read and use.*

Example 1:

```
SELECT t.region,
      ROUND(AVG(p.productcost), 2) AS AvgProductCost,
      CASE WHEN AVG(p.productcost) > 200 THEN 'High Cost' ELSE 'Low
Cost' END AS CostCategory
FROM products p
JOIN product_subcategories psc ON p.productsubcategorykey =
psc.productsubcategorykey
JOIN product_categories pc ON psc.productcategorykey =
pc.productcategorykey
JOIN returns r ON p.productkey = r.productkey
JOIN territories t ON r.territorykey = t.salesterritorykey
GROUP BY t.region
ORDER BY AvgProductCost DESC;
```

Output:

Result Grid | Filter Rows: | Export:

region	AvgProductCost	CostCategory
Australia	348.27	High Cost
Germany	320.17	High Cost
France	255.09	High Cost
United Kingdom	240.09	High Cost
Southwest	218.9	High Cost
Northwest	202.06	High Cost
Canada	110.18	Low Cost
Southeast	12.19	Low Cost

Result 8

Example 2:

```
SELECT TerritoryKey,
      SUM(CASE
            WHEN orderQuantity > 2 THEN orderquantity
            ELSE 0
            END) AS High_Performance_Sales,
            SUM(CASE
            WHEN orderQuantity BETWEEN 1 AND 2 THEN orderquantity
            ELSE 0
            END) AS Medium_Performance_Sales,
            SUM(CASE
            WHEN orderQuantity < 1 THEN orderquantity
            ELSE 0
            END) AS Low_Performance_Sales
FROM Sales_2017
GROUP BY Territorykey;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TerritoryKey	High_Performance_Sales	Medium_Performance_Sales	Low_Performance_Sales
1	801	5988	0
4	1284	8106	0
6	852	5008	0
10	630	4463	0
8	600	3786	0
9	1185	8302	0
7	498	3738	0
5	0	35	0
3	0	14	0
2	3	21	0

Example 3:

As this section is very important, we have to provide more queries about this concept. Here is this query which helps you understand the concepts more clearly.

```
SELECT EmployeeID,
      SUM(Revenue) AS Total_Revenue,
      SUM(CASE
            WHEN Revenue >= 100000 THEN Revenue * 0.1
            WHEN Revenue BETWEEN 50000 AND 99999 THEN Revenue * 0.05
            ELSE Revenue * 0.02
            END) AS Bonus
FROM Sales
GROUP BY EmployeeID;
```

Example 4:

```
SELECT  Subject,
            COUNT(CASE WHEN Grade >= 60 THEN 1 ELSE NULL END) AS
Pass_Count,
            COUNT(CASE WHEN Grade < 60 THEN 1 ELSE NULL END) AS
Fail_Count
FROM Grades
GROUP BY Subject;
```

Understanding conditional logic in JOINs:

Using complex conditional logic in JOINs can enhance your SQL queries by allowing you to merge tables based on multiple conditions or dynamic criteria.

Syntax:

```
SELECT [columns]
       [CASE expression]
FROM table1
[JOIN type] table2 ON table1.column = table2.column
            [AND (CASE
                        WHEN condition1 THEN true_value
                        WHEN condition2 THEN true_value
                        ELSE false_value
                  END)]
[JOIN type] table3 ON table2.column = table3.column
            [AND (CASE
                        WHEN condition1 THEN true_value
                        WHEN condition2 THEN true_value
                        ELSE false_value
                  END)]
WHERE condition
GROUP BY columns
```

```
ORDER BY columns;
```

NOTE:

- ***SELECT [columns]:*** *Specify the columns you want to retrieve in the result set. You can also include a CASE expression in the SELECT clause to apply conditional logic to the selected columns.*
- ***FROM table1:*** *Specify the main table from which you want to retrieve data.*
- ***[JOIN type] table2 ON table1.column = table2.column:*** *Specify the type of join (INNER JOIN, LEFT JOIN, RIGHT JOIN, CROSS JOIN, etc.) and the second table you want to join. The ON clause specifies the join condition, which is the condition that matches rows from the two tables.*
- ***[AND (CASE WHEN condition1 THEN true_value WHEN condition2 THEN true_value ... ELSE false_value END)]:*** *Optionally, you can include a CASE expression in the ON clause of the join. This CASE expression applies conditional logic to determine whether the join condition should be considered true or false for each row combination. If the condition is true, the true_value is used for the join; otherwise, the false_value is used.*
- ***[JOIN type] table3 ON table2.column = table3.column [AND (CASE ... END)]:*** *You can continue adding additional table joins, specifying the join type, and join condition, and optionally including another CASE expression in the ON clause for each join.*
- ***[WHERE [condition]]:*** *Optionally, you can include a WHERE clause to filter the results based on a specified condition.*
- ***[GROUP BY [columns]]:*** *Optionally, you can include a GROUP BY clause to group the results by one or more columns.*
- ***[ORDER BY [columns]]:*** *Optionally, you can include an ORDER BY clause to sort the results by one or more columns.*

Example 1:

```
SELECT  p.ProductName, pc.CategoryName, psc.SubcategoryName,
            SUM(r.ReturnQuantity) AS TotalReturns,
            CASE
                  WHEN SUM(r.ReturnQuantity) > 50 THEN 'High Returns'
                  WHEN SUM(r.ReturnQuantity) > 25 THEN 'Moderate
Returns'
                  ELSE 'Low Returns'
            END AS ReturnLevel
FROM Products p
JOIN Product_Subcategories psc ON p.ProductSubcategoryKey =
psc.ProductSubcategoryKey
JOIN Product_Categories pc ON psc.ProductCategoryKey =
pc.ProductCategoryKey
LEFT JOIN Returns r ON p.ProductKey = r.ProductKey
        AND (CASE
                WHEN pc.CategoryName = 'Bikes' THEN r.ReturnQuantity >
20
                WHEN pc.CategoryName = 'Components' THEN
r.ReturnQuantity > 5
                ELSE TRUE
```

```
            END)
JOIN Territories t ON r.TerritoryKey = t.SalesTerritoryKey
        AND (CASE
                WHEN t.Region = 'North America' THEN r.ReturnQuantity >
20
                WHEN t.Region = 'Europe' THEN r.ReturnQuantity > 15
                ELSE TRUE
            END)
GROUP BY p.ProductName, pc.CategoryName, psc.SubcategoryName
HAVING SUM(r.ReturnQuantity) > 10
ORDER BY TotalReturns DESC;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ProductName	CategoryName	SubcategoryName	TotalReturns	ReturnLevel
Water Bottle - 30 oz.	Accessories	Bottles and Cages	155	High Returns
Patch Kit/8 Patches	Accessories	Tires and Tubes	95	High Returns
Mountain Tire Tube	Accessories	Tires and Tubes	93	High Returns
Mountain Bottle Cage	Accessories	Bottles and Cages	77	High Returns
Sport-100 Helmet, Red	Accessories	Helmets	70	High Returns
Road Tire Tube	Accessories	Tires and Tubes	67	High Returns
Sport-100 Helmet, Blue	Accessories	Helmets	66	High Returns
Road Bottle Cage	Accessories	Bottles and Cages	56	High Returns
Fender Set - Mountain	Accessories	Fenders	54	High Returns
Sport-100 Helmet, Black	Accessories	Helmets	52	High Returns
HL Mountain Tire	Accessories	Tires and Tubes	49	Moderate R...

Result 15

Highlighting the major use case of the CASE statement:

In MySQL, the CASE statement can be used to simulate pivoting when combined with aggregate functions like SUM, COUNT, MAX, etc. This is because MySQL, like many other SQL databases, does not have a built-in PIVOT function. Instead, you can use CASE within an aggregate function to conditionally aggregate data into separate columns.

Example:

```
SELECT ProductID,
      SUM(CASE WHEN Region = 'North' THEN Amount ELSE 0 END) AS
North_Sales,
      SUM(CASE WHEN Region = 'South' THEN Amount ELSE 0 END) AS
South_Sales,
      SUM(CASE WHEN Region = 'East' THEN Amount ELSE 0 END) AS
East_Sales,
      SUM(CASE WHEN Region = 'West' THEN Amount ELSE 0 END) AS
West_Sales
```

```
FROM Sales
GROUP BY ProductID
ORDER BY ProductID;
```

FUN TIME:

1. **Customer Return Behavior:**

```
SELECT c.customerkey, c.FirstName, c.lastname,
    SUM(COALESCE(r.returnQuantity, 0) AS total_return_quantity,
    CASE
         WHEN SUM(COALESCE(r.returnQuantity, 0) > 5 THEN 'Frequent
Returner'
         WHEN SUM(COALESCE(r.returnQuantity, 0) between 1 and 5 THEN
'Occasional Returner'
         ELSE 'Non-Returner'
     END AS return_behavior
FROM customers c
LEFT JOIN sales_2015 s5 ON c.customerkey = s5.customerkey
LEFT JOIN returns r ON s5.ProductKey = r.ProductKey
GROUP BY c.customerkey, c.FirstName, c.lastname
ORDER BY total_return_quantity DESC;
```

Description:

This query categorizes customers based on their total return quantity. It identifies customers as 'Frequent Returners' if they have returned items more than five times, 'Occasional Returners' if they have returned items between one and five times, and 'Non-Returners' if they have not returned any items. The results are grouped by customer and ordered by the total return quantity in descending order.

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rov

customerkey	FirstName	lastname	total_return_quantity	return_behavior
11605	HEIDI	SUBRAM	21	Frequent Returner
11612	KEITH	ANDERSEN	21	Frequent Returner
12123	WESLEY	LIANG	21	Frequent Returner
12128	KRISTY	MUNOZ	21	Frequent Returner
12214	JULIE	RAJE	21	Frequent Returner
12226	FAITH	WARD	21	Frequent Returner
12264	BRUCE	ASHE	21	Frequent Returner
12270	EBONY	RAMOS	21	Frequent Returner
12286	NATHAN	JENKINS	21	Frequent Returner
12287	ANDRE	ARUN	21	Frequent Returner
12335	FRANCIS...	ASHE	21	Frequent Returner

Result 24

NOTE: The above output might not contain all the returned rows.

2. **Total Return Quantity by Product Subcategory and Category with Segmentation:**

```
SELECT pc.categoryname, ps.subcategoryname,
      SUM(r.returnQuantity) AS total_return_quantity,
      CASE WHEN SUM(r.returnQuantity) > 100 THEN 'High Return'
                  WHEN SUM(r.returnQuantity) > 50 THEN 'Medium Return'
                  ELSE 'Low Return'
      END AS return_segment
FROM products p
JOIN product_subcategories ps ON p.productsubcategorykey =
ps.productsubcategorykey
JOIN product_categories pc ON ps.productcategorykey =
pc.productcategorykey
JOIN returns r ON p.productkey = r.productkey
GROUP BY pc.categoryname, ps.subcategoryname;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

categoryname	subcategoryname	total_return_quantity	return_segment
Bikes	Road Bikes	223	High Return
Bikes	Mountain Bikes	136	High Return
Accessories	Bottles and Cages	288	High Return
Accessories	Tires and Tubes	534	High Return
Accessories	Hydration Packs	25	Low Return
Accessories	Helmets	188	High Return
Clothing	Jerseys	93	Medium Return
Accessories	Fenders	54	Medium Return
Accessories	Bike Racks	8	Low Return
Accessories	Cleaners	25	Low Return
Clothing	Caps	46	Low Return

Result 20

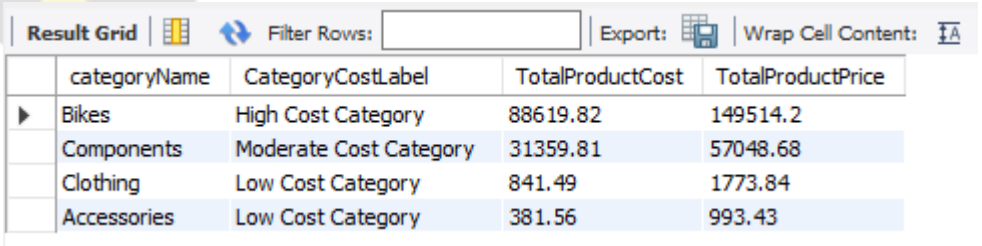
NOTE: The above output might not contain all the returned rows.

3. **Total product cost and total product price for each product category:**

```
SELECT pc.categoryName,
    CASE
        WHEN SUM(COALESCE(p.productCost, 0)) < 10000 THEN 'Low
Cost Category'
        WHEN SUM(COALESCE(p.productCost, 0)) >= 10000 AND
SUM(COALESCE(p.productCost, 0)) < 50000 THEN 'Moderate Cost
Category'
        ELSE 'High Cost Category'
    END AS CategoryCostLabel,
    ROUND(SUM(COALESCE(p.productCost, 0)), 2) AS TotalProductCost,
    ROUND(SUM(COALESCE(p.productPrice, 0)), 2) AS
```

```
TotalProductPrice
FROM
    Products p
    JOIN Product_Subcategories ps ON p.productSubcategoryKey =
ps.productSubcategoryKey
    JOIN Product_Categories pc ON ps.productCategoryKey =
pc.productCategoryKey
GROUP BY pc.categoryName
ORDER BY TotalProductCost DESC;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

categoryName	CategoryCostLabel	TotalProductCost	TotalProductPrice
Bikes	High Cost Category	88619.82	149514.2
Components	Moderate Cost Category	31359.81	57048.68
Clothing	Low Cost Category	841.49	1773.84
Accessories	Low Cost Category	381.56	993.43

NOTE: The above output might not contain all the returned rows.

4. **Total return quantity and average product price for each product subcategory and region:**

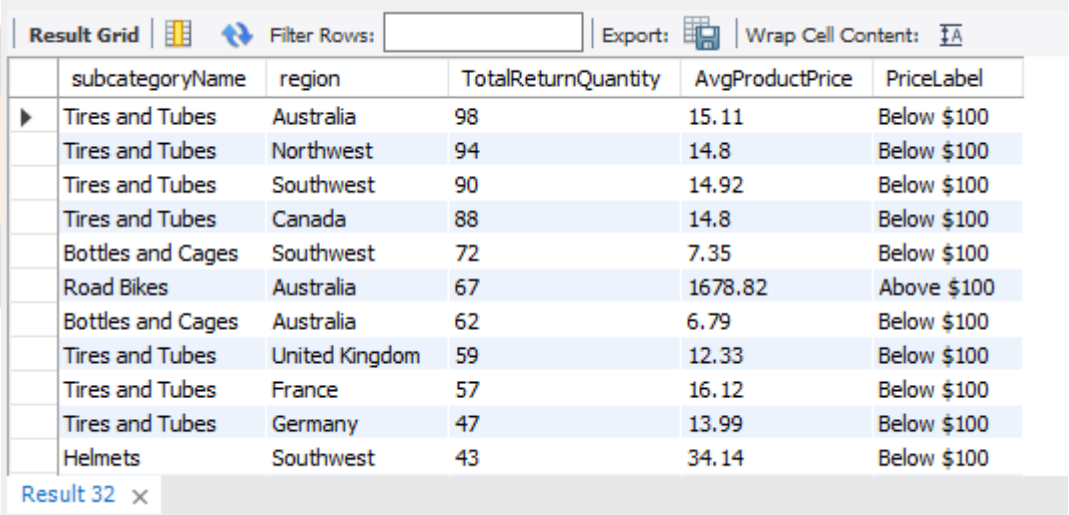
```
SELECT ps.subcategoryName, t.region,
    SUM(r.returnQuantity) AS TotalReturnQuantity,
    ROUND(AVG(p.productPrice), 2) AS AvgProductPrice,
    CASE
        WHEN AVG(p.productPrice) < 100 THEN 'Below $100'
        ELSE 'Above $100'
    END AS PriceLabel
FROM
    Returns r
    JOIN Products p ON r.productKey = p.productKey
    JOIN Product_Subcategories ps ON p.productSubcategoryKey =
ps.productSubcategoryKey
    JOIN Territories t ON r.territoryKey = t.salesterritoryKey
GROUP BY
    ps.subcategoryName, t.region
ORDER BY
    TotalReturnQuantity DESC;
```

Description:

This query helps you retrieve a summarized view of return quantities and average product prices for different product subcategories across geographic regions. It enables you to identify

subcategory-region combinations with high return volumes and categorize them based on whether their average product price falls above or below $100. This information can be useful for analyzing return patterns, identifying areas of concern, and making data-driven decisions related to product pricing and inventory management.

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

subcategoryName	region	TotalReturnQuantity	AvgProductPrice	PriceLabel
Tires and Tubes	Australia	98	15.11	Below $100
Tires and Tubes	Northwest	94	14.8	Below $100
Tires and Tubes	Southwest	90	14.92	Below $100
Tires and Tubes	Canada	88	14.8	Below $100
Bottles and Cages	Southwest	72	7.35	Below $100
Road Bikes	Australia	67	1678.82	Above $100
Bottles and Cages	Australia	62	6.79	Below $100
Tires and Tubes	United Kingdom	59	12.33	Below $100
Tires and Tubes	France	57	16.12	Below $100
Tires and Tubes	Germany	47	13.99	Below $100
Helmets	Southwest	43	34.14	Below $100

Result 32

NOTE: The above output might not contain all the returned rows.

5. **Total order quantity and total revenue for each combination of product category, region, and customer gender:**

```
SELECT pc.categoryName, t.region,
    SUM(s6.orderQuantity) AS TotalOrderQuantity,
    SUM(p.productPrice * s6.orderQuantity) AS TotalRevenue,
    CASE
        WHEN SUM(p.productPrice * s6.orderQuantity) < 50000 THEN
'Low Revenue'
        WHEN SUM(p.productPrice * s6.orderQuantity) >= 50000 AND
SUM(p.productPrice * s6.orderQuantity) < 200000 THEN 'Moderate
Revenue'
        ELSE 'High Revenue'
    END AS RevenueLabel
FROM
    Products p
JOIN Product_Subcategories ps ON p.productSubcategoryKey =
ps.productSubcategoryKey
JOIN Product_Categories pc ON ps.productCategoryKey =
pc.productCategoryKey
JOIN sales_2016 s6 ON p.productKey = s6.productKey
JOIN Territories t ON s6.territoryKey = t.salesterritoryKey
```

```
GROUP BY pc.categoryName, t.region
ORDER BY TotalRevenue DESC;
```

Description:

The query calculates the total order quantity by summing the order quantities from the sales data. It also calculates the total revenue by multiplying the product price with the order quantity and summing the result.
The CASE statement is used to assign a revenue label based on the calculated total revenue for each category-region combination. If the total revenue is below $50,000, it is labeled as "Low Revenue"; if it is between $50,000 and $200,000, it is labeled as "Moderate Revenue"; otherwise, it is labeled as "High Revenue".
The output displays the product category name, geographic region, total order quantity, total revenue, and the assigned revenue label for each category-region combination, sorted in descending order of total revenue.

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

categoryName	region	TotalOrderQuantity	TotalRevenue	RevenueLabel
Bikes	Australia	1663	2777917.7082000435	High Revenue
Bikes	Southwest	1034	1530794.0411000047	High Revenue
Bikes	United Kingdom	750	1155411.0627999995	High Revenue
Bikes	Northwest	671	963440.762299998	High Revenue
Bikes	Germany	606	958352.5952999973	High Revenue
Bikes	France	585	935345.595099998	High Revenue
Bikes	Canada	295	438313.2044999992	High Revenue
Accessories	Southwest	5152	82872.85920000095	Moderate Revenue
Accessories	Australia	4854	77186.72600000112	Moderate Revenue
Accessories	Northwest	3931	62991.96720000163	Moderate Revenue
Accessories	Canada	3778	59060.81500000137	Moderate Revenue

Result 31

NOTE: The above output might not contain all the returned rows.

6. Return Quantity by Region and Product Category:

```
SELECT t.region,
    SUM(CASE WHEN pc.categoryName = 'Bikes' THEN r.returnQuantity
ELSE 0 END) AS BikeReturns,
    SUM(CASE WHEN pc.categoryName = 'Accessories' THEN
r.returnQuantity ELSE 0 END) AS AccessoriesReturns,
    SUM(CASE WHEN pc.categoryName = 'Clothing' THEN
r.returnQuantity ELSE 0 END) AS ClothingReturns,
    SUM(CASE WHEN pc.categoryName = 'Components' THEN
r.returnQuantity ELSE 0 END) AS ComponentReturns
FROM returns r
```

```
JOIN products p ON r.productkey = p.productkey
JOIN product_subcategories psc ON p.productsubcategorykey =
psc.productsubcategorykey
JOIN product_categories pc ON psc.productcategorykey =
pc.productcategorykey
JOIN territories t ON r.territorykey = t.salesterritorykey
GROUP BY t.region
ORDER BY t.region;
```

Description:

This query retrieves the sum of return quantities for different product categories, grouped by geographic region. It calculates the total number of returns for Bikes, Accessories, Clothing, and Components categories across all regions. The output displays the region name along with the corresponding return quantities for each product category, sorted by the region name.

The query uses a CASE statement within the SUM aggregate function to calculate the return quantities for each product category separately. It checks the category name condition and sums the return quantity if the condition matches, or assigns 0 if it doesn't match.

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

region	BikeReturns	AccessoriesReturns	ClothingReturns	ComponentReturns
Australia	125	223	56	0
Canada	22	165	51	0
France	42	121	23	0
Germany	53	88	22	0
Northwest	58	172	40	0
Southeast	0	1	0	0
Southwest	78	231	53	0
United Kingdom	51	129	24	0

Result 4

NOTE: The above output might not contain all the returned rows.