Northwind Database (SQLite Project)

By: Fatma Mohamed Hashem Abdel Fattah
Ins.: Asmaa El-Taher

Overview:

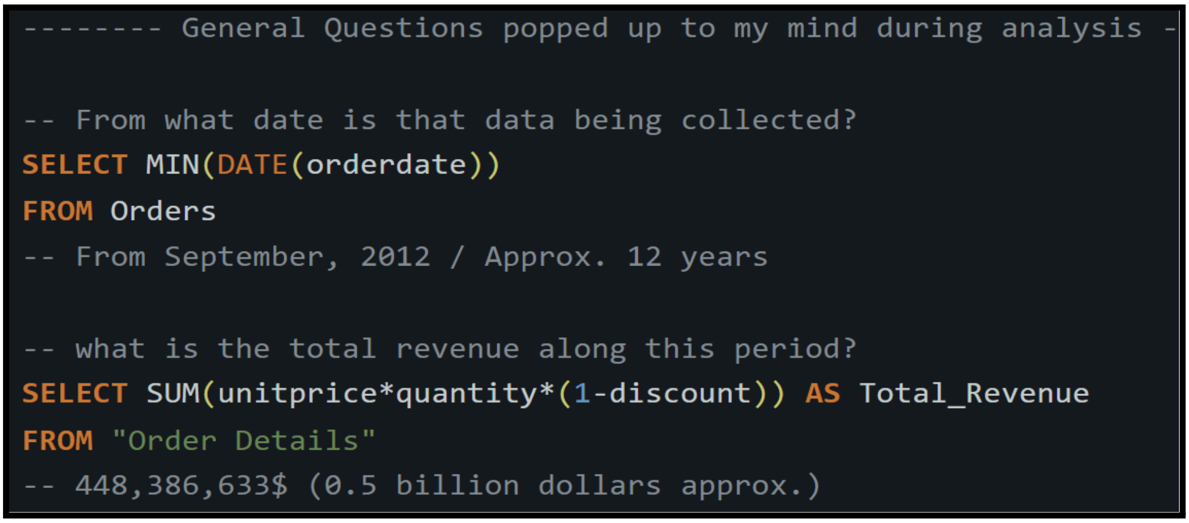
This report provides an in-depth analysis of the Northwind database, focusing on uncovering insights into customer behavior, order patterns, and product performance. The study aims to address key business questions, segment customers based on RFM analysis and order value, and highlight significant trends in geographic and product data.

The analysis includes:

- **General Insights:** Key questions such as the data collection timeline and total revenue generated.
- **Customer Segmentation:** Detailed segmentation based on Recency, Frequency, and Monetary (RFM) values to identify customer segments like Champions, Potential Loyalists, and At-Risk customers.
- **Order Value Analysis:** Segmentation of customers into High, Medium, and Low-Value categories based on their average order value.
- **Geographic Insights:** Examination of customer distribution and segment performance across countries.
- **Product Performance:** Analysis of product revenue, category distribution, and identifying top-performing products.
- **Employee Performance:** Analysis of employees based on revenue generated, number of orders processed, and the average of order value.

The insights derived from this analysis will aid in understanding customer behavior, optimizing business strategies, and enhancing decision-making to drive growth.

- **General Insights:**
  - At first, I answered some general questions about the data like the data collection timeline, and the total revenue generated.

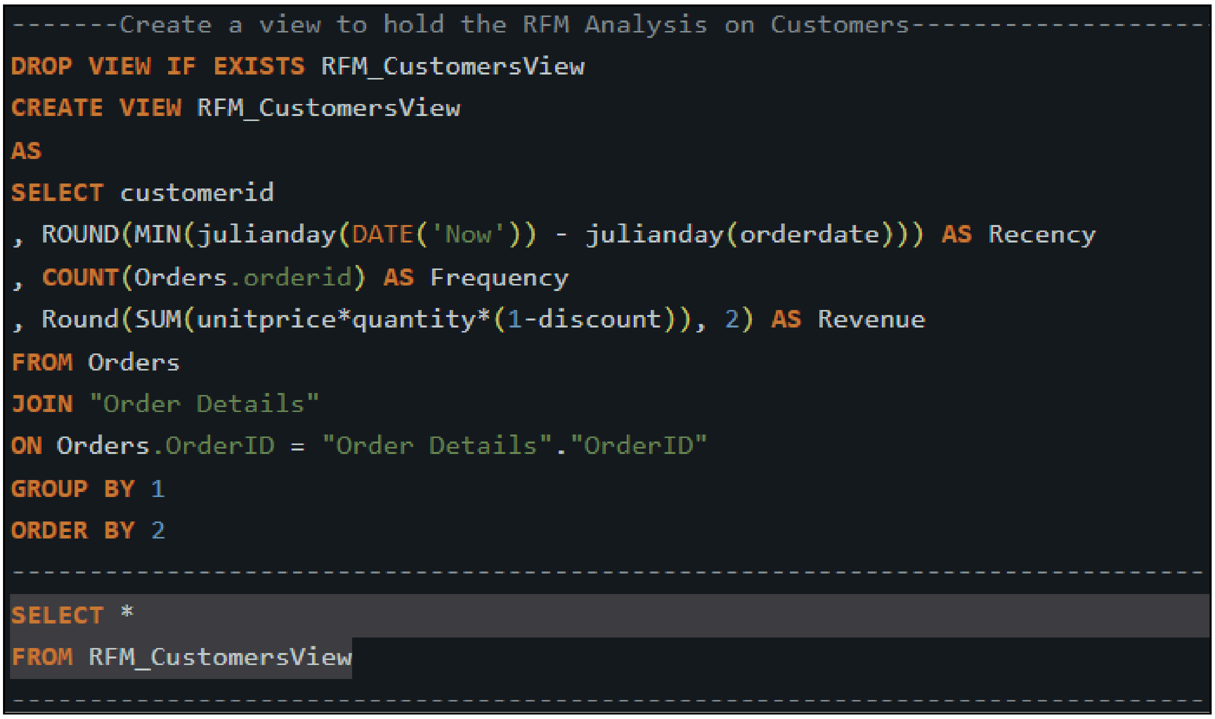


```
-------- General Questions popped up to my mind during analysis -

-- From what date is that data being collected?
SELECT MIN(DATE(orderdate))
FROM Orders
-- From September, 2012 / Approx. 12 years

-- what is the total revenue along this period?
SELECT SUM(unitprice*quantity*(1-discount)) AS Total_Revenue
FROM "Order Details"
-- 448,386,633$ (0.5 billion dollars approx.)
```

I divided the steps of analysis into 4 aspects: customer-wise, order-wise, product-wise, and employee-wise. I applied my learned SQL skills to generate answers to my curious questions.

- **Customer Segmentation:**
  - **RFM Analysis:**
    - **Objective:** I wanted to categorize customers into [Champions, Potential loyalists, At-Risk] categories by several factors such as:
      - Their **Recency:** the number of days since their last order.
      - Their **Frequency:** How often did the customer order from us.
      - The **revenue** that the customer causes.

To calculate these metrics I used this query on the ***[Orders]*** table joined with the ***[Order Details]*** table. In addition, I created a **view** of this query to ease the process of categorization meanafter.

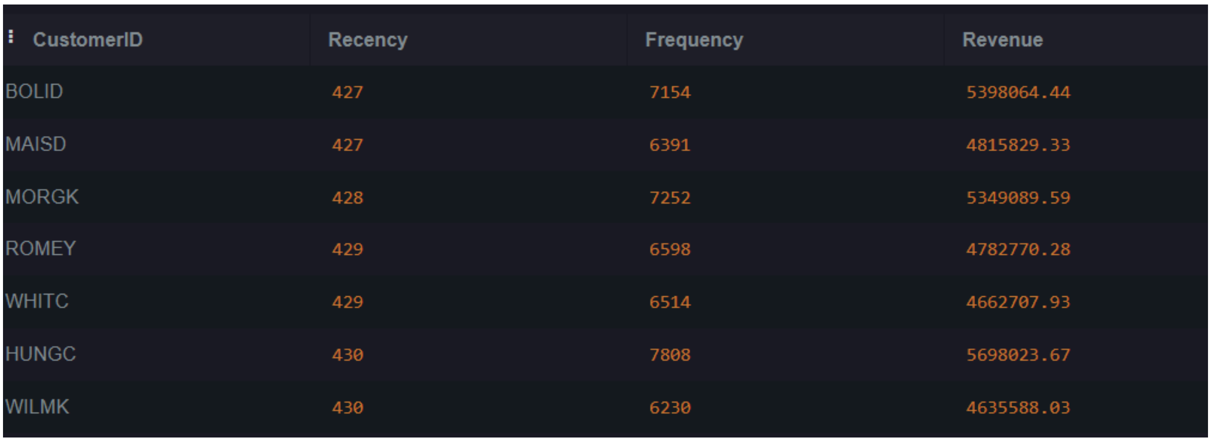


```
-------Create a view to hold the RFM Analysis on Customers-------------------
DROP VIEW IF EXISTS RFM_CustomersView
CREATE VIEW RFM_CustomersView
AS
SELECT customerid
, ROUND(MIN(julianday(DATE('Now')) - julianday(orderdate))) AS Recency
, COUNT(Orders.orderid) AS Frequency
, Round(SUM(unitprice*quantity*(1-discount)), 2) AS Revenue
FROM Orders
JOIN "Order Details"
ON Orders.OrderID = "Order Details"."OrderID"
GROUP BY 1
ORDER BY 2
-----------------------------------------------------------------------------
SELECT *
FROM RFM_CustomersView
-----------------------------------------------------------------------------
```

Explanation of the query:

- *To calculate the **recency,** I used the **julianday()** function on the date of the very first order using the **MIN()** function and subtracting from the **DATE('NOW')** to get the number of days since the first order*
- *For Frequency, We just need to COUNT() the number of orders made along the data timeline*
- *For **Revenue**, we **SUM()**, for all orders made by a customer, the (unit-price multiplied by the quantity of the items of the order and finally apply any discount ratio there)*
- ***Joined** between the **[Orders]** table and the **[Order Details]** table one the **customer_id** column*
- *And finally **GROUP BY** each customer*

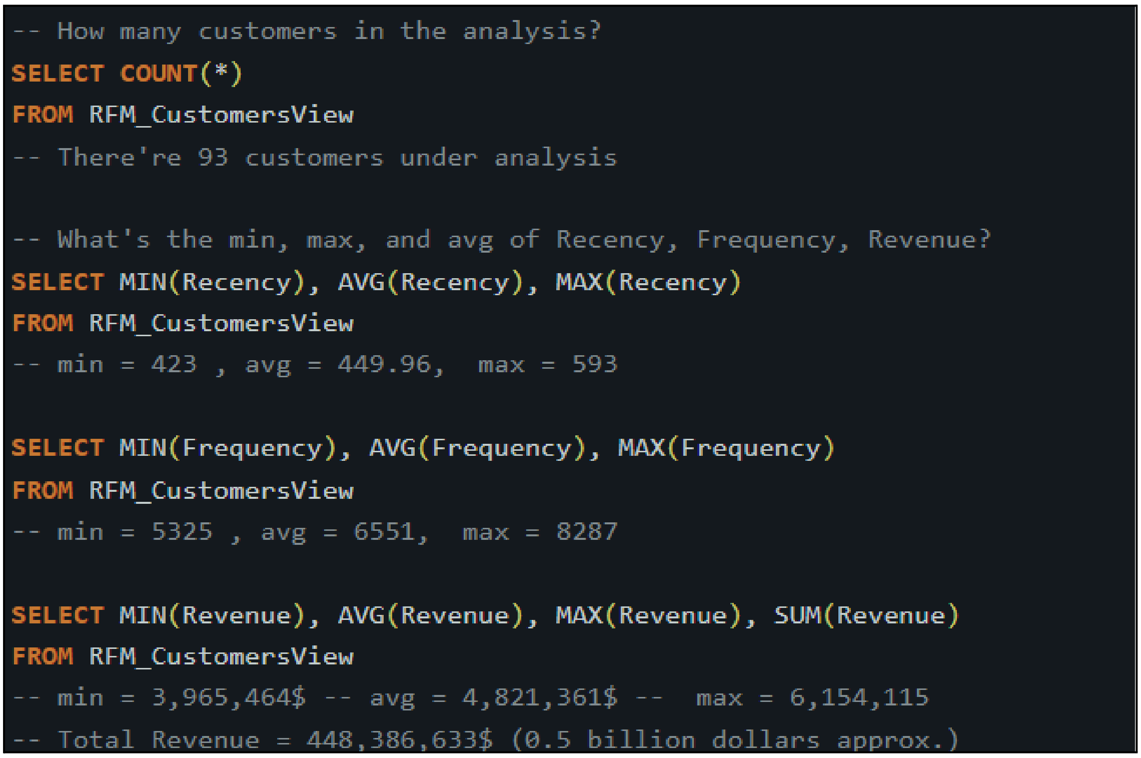
Query Result:



CustomerID	Recency	Frequency	Revenue
BOLID	427	7154	5398064.44
MAISD	427	6391	4815829.33
MORGK	428	7252	5349089.59
ROMEY	429	6598	4782770.28
WHITC	429	6514	4662707.93
HUNGC	430	7808	5698023.67
WILMK	430	6230	4635588.03

As shown, each customer has their recency, frequency, and revenue calculated

- **Customer Segmentation by the RFM:**
    - How can we think of the customer segmentation process for the previous query result?
        - My way of thinking was to have a look at the dispersion and average of this data (i.e. calculate the min, max, avg)



```sql
-- How many customers in the analysis?
SELECT COUNT(*)
FROM RFM_CustomersView
-- There're 93 customers under analysis

-- What's the min, max, and avg of Recency, Frequency, Revenue?
SELECT MIN(Recency), AVG(Recency), MAX(Recency)
FROM RFM_CustomersView
-- min = 423 , avg = 449.96,  max = 593

SELECT MIN(Frequency), AVG(Frequency), MAX(Frequency)
FROM RFM_CustomersView
-- min = 5325 , avg = 6551,  max = 8287

SELECT MIN(Revenue), AVG(Revenue), MAX(Revenue), SUM(Revenue)
FROM RFM_CustomersView
-- min = 3,965,464$ -- avg = 4,821,361$ --  max = 6,154,115
-- Total Revenue = 448,386,633$ (0.5 billion dollars approx.)
```

Awesome, Now having these values about the data in mind we can divide the customers into segments

- **The Customer Segmentation Methodology I decided on was:**
    - **For Recency (The lower the better)**:

        - Champions: recency < 440

        - Potential Loyalists: recency between 440 and 495

        - At Risk: recency > 495

    - **For Frequency (The higher the better):**

        - Champions: Freq > 7200

        - Potential Loyalists: Freq between 6200 and 7200

        - At Risk: Freq < 6000

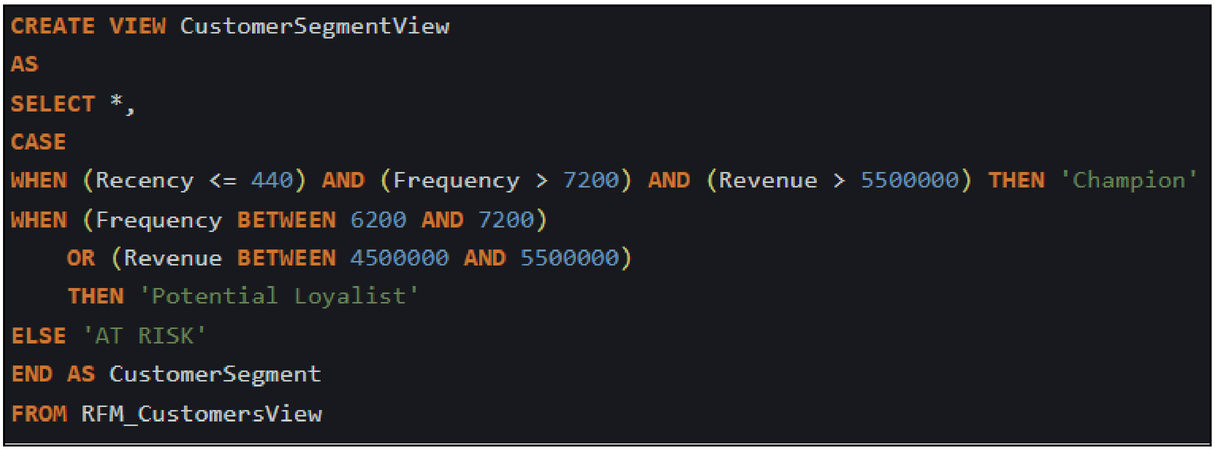
    - **For Revenue (The higher the better):**

        - Champions: Rev > 5.5 million dollars

        - Potential Loyalists: Freq between 4.5 and 5.5 million dollars

        - At Risk: Rev < 4.5 million dollars

Again, I created a view for the query doing the categorization work:



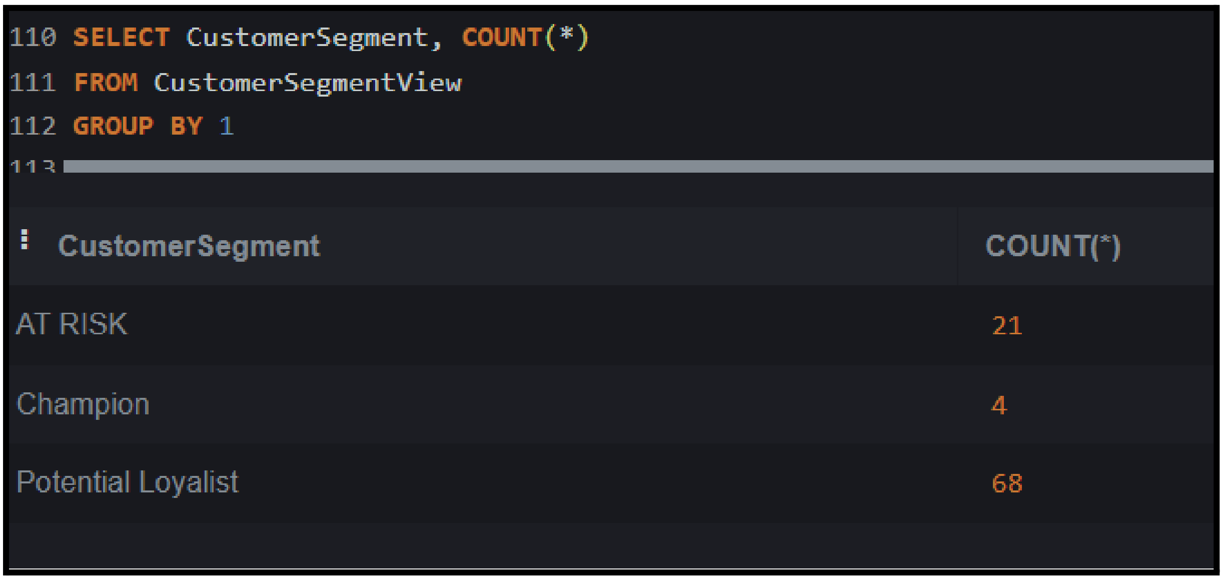
```
CREATE VIEW CustomerSegmentView
AS
SELECT *,
CASE
WHEN (Recency <= 440) AND (Frequency > 7200) AND (Revenue > 5500000) THEN 'Champion'
WHEN (Frequency BETWEEN 6200 AND 7200)
    OR (Revenue BETWEEN 4500000 AND 5500000)
    THEN 'Potential Loyalist'
ELSE 'AT RISK'
END AS CustomerSegment
FROM RFM_CustomersView
```

Explanation of the query:

*Here, I used the **CASE** statement to check in which category will a customer with a certain recency, frequency, and revenue be.*

Query Result:

*I used the **GROUP BY** clause to see how many customers reside in each category*



```
SELECT CustomerSegment, COUNT(*)
FROM CustomerSegmentView
GROUP BY 1
```

CustomerSegment	COUNT(*)
AT RISK	21
Champion	4
Potential Loyalist	68

Insight:

- Only 4 customers are **champions**
- 68 are **potential loyalists** - this is a great share of loyalists.
- **21** customers are **at-risk** representing about **22%** of our customers are **at risk!**

Recommendations:

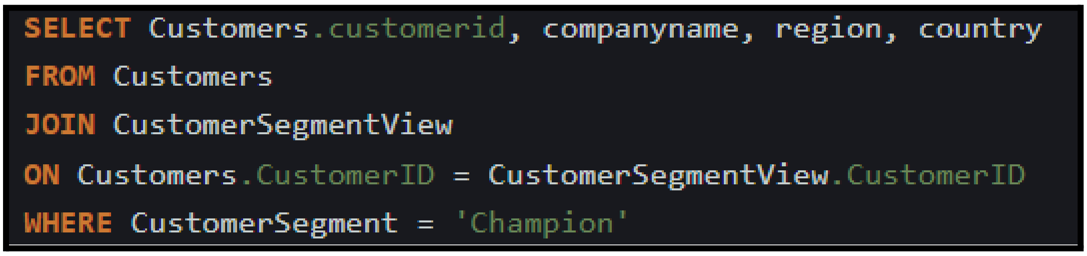
- Work on converting the **potential loyalists into champions** by providing the **68** potential loyalists with **loyalty programs**, **discounts** for frequent purchases, or **incentives** for referrals.
- Develop **retention campaigns** focused on the **21 at-risk customers.**

- ## Some Customer Segments Geographic Analysis

For further analysis of the customer segments, let's see how our customer categories are distributed...

○ Champions:

- I selected the geographic info for customers with ***ids*** that are in the ***customer_ids*** of the **champions**.

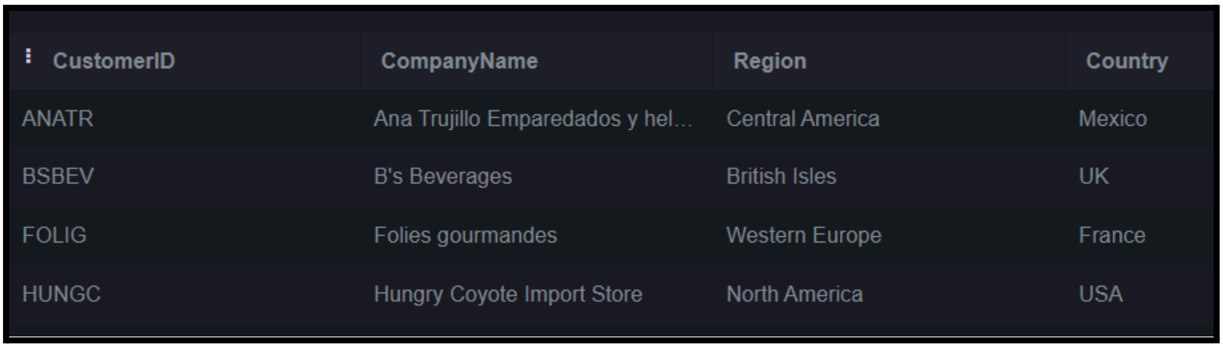


```sql
SELECT Customers.customerid, companyname, region, country
FROM Customers
JOIN CustomerSegmentView
ON Customers.CustomerID = CustomerSegmentView.CustomerID
WHERE CustomerSegment = 'Champion'
```

Explanation of the query:

Here, in this query, I used ***Join*** *between the* ***[Customer]*** *table and the* ***[CustomerSegmentView]*** *on the customerid and filtered on the* ***champions.***

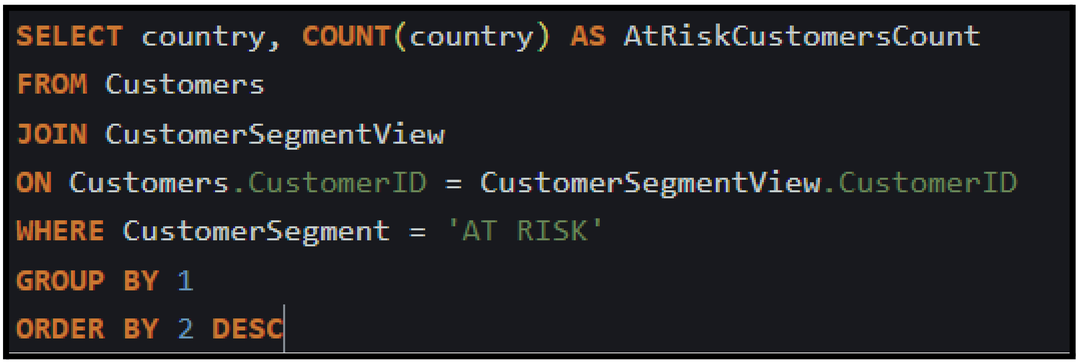
Query Result:



CustomerID	CompanyName	Region	Country
ANATR	Ana Trujillo Emparedados y hel...	Central America	Mexico
BSBEV	B's Beverages	British Isles	UK
FOLIG	Folies gourmandes	Western Europe	France
HUNGC	Hungry Coyote Import Store	North America	USA

Insight: They're **distributed** over **different countries** i.e. Mexico, UK, France, and USA.

○ At Risk Customers:



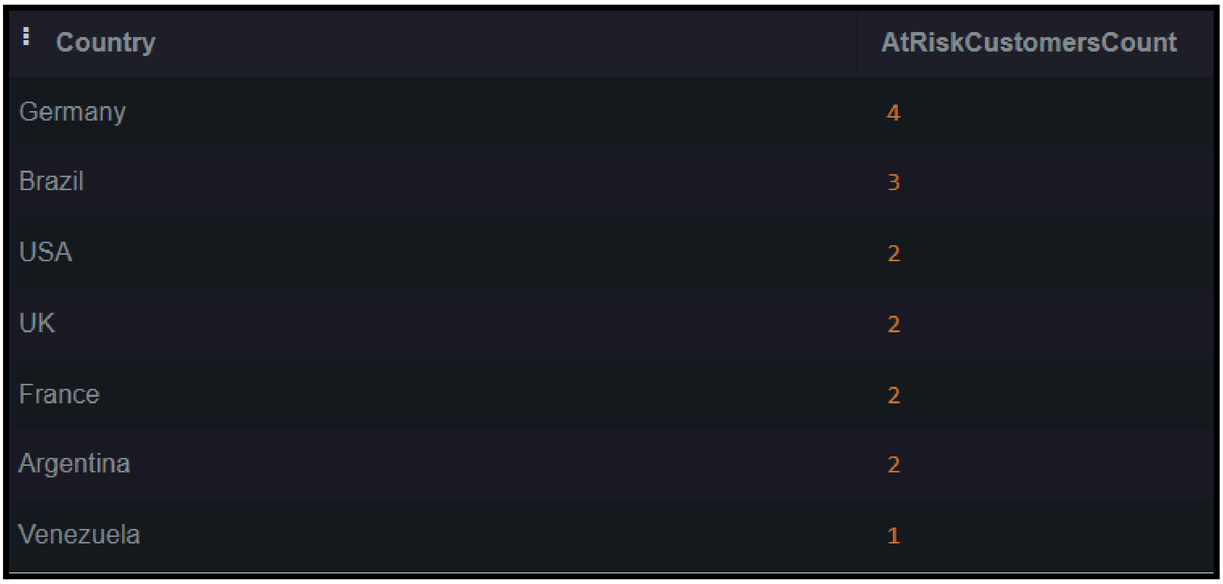
```sql
SELECT country, COUNT(country) AS AtRiskCustomersCount
FROM Customers
JOIN CustomerSegmentView
ON Customers.CustomerID = CustomerSegmentView.CustomerID
WHERE CustomerSegment = 'AT RISK'
GROUP BY 1
ORDER BY 2 DESC
```

Explanation of the query:

Here, in this query, I used ***Join*** *between the* ***[Customer]*** *table and the* ***[CustomerSegmentView]*** *on the customerid and filtered on the* ***at-risk.***

Then, Grouping *For each country, to* ***COUNT()*** *how many* ***at-risk*** *customers reside there*

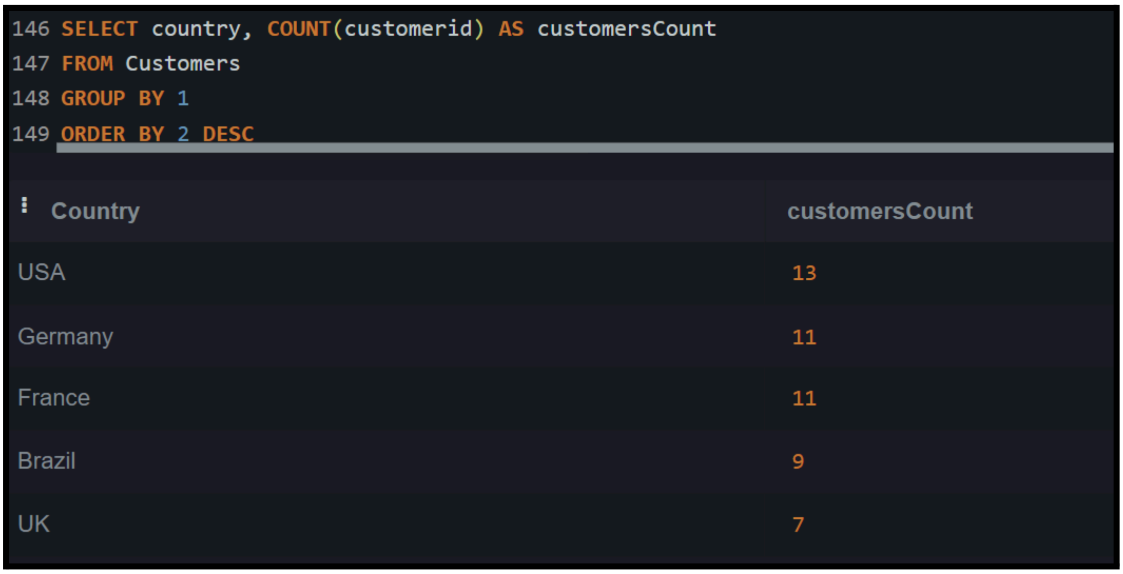
Query Result:



Country	AtRiskCustomersCount
Germany	4
Brazil	3
USA	2
UK	2
France	2
Argentina	2
Venezuela	1

Insight: Germany has the highest number of **At-Risk** customers

- **Let's see the general distribution of customers across the countries**

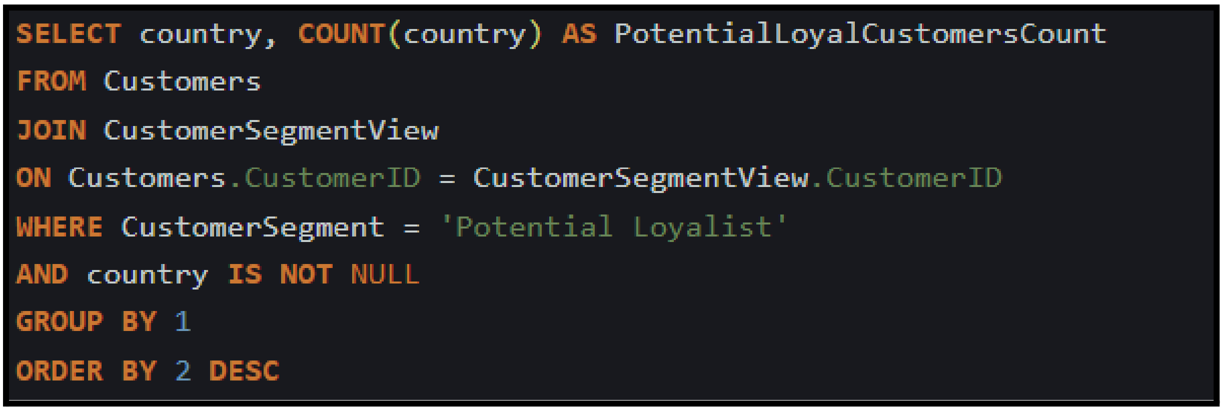


```sql
SELECT country, COUNT(customerid) AS customersCount
FROM Customers
GROUP BY 1
ORDER BY 2 DESC
```

Country	customersCount
USA	13
Germany	11
France	11
Brazil	9
UK	7

Insight: 4 of the 11 customers in **Germany** are **AT RISK!!**

- **Potential Loyalists:**



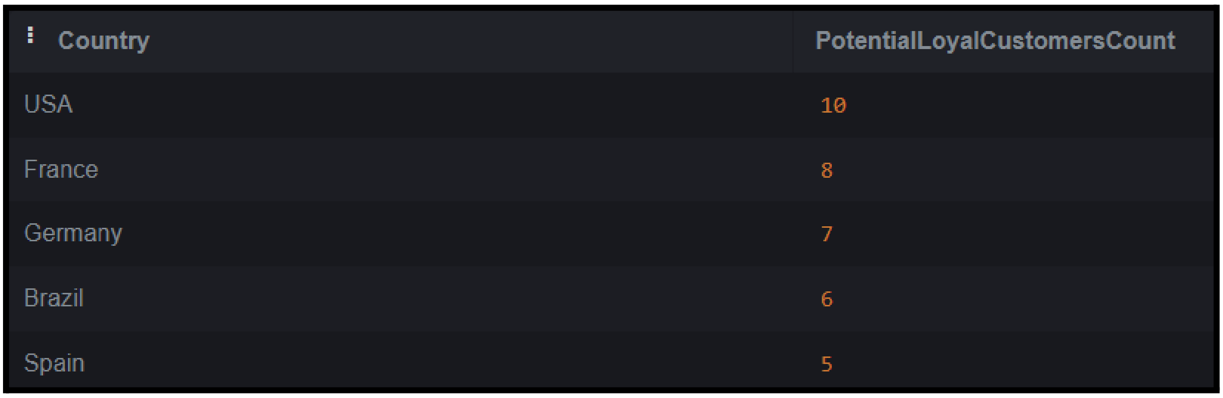
```
SELECT country, COUNT(country) AS PotentialLoyalCustomersCount
FROM Customers
JOIN CustomerSegmentView
ON Customers.CustomerID = CustomerSegmentView.CustomerID
WHERE CustomerSegment = 'Potential Loyalist'
AND country IS NOT NULL
GROUP BY 1
ORDER BY 2 DESC
```

Explanation of the query:

*Here, in this query, I used **Join** between the **[Customer]** table and the **[CustomerSegmentView]** on the customerid and filtered on the **potential loyalist.***

***Then, Grouping** For each country, to **COUNT()** how many **potential loyal customers** reside there.*

Query Result:



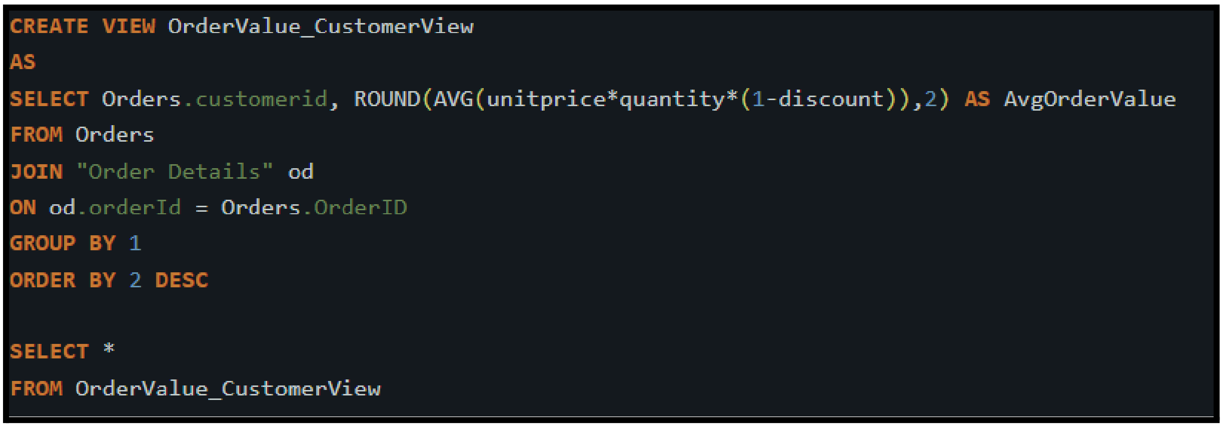
Country	PotentialLoyalCustomersCount
USA	10
France	8
Germany	7
Brazil	6
Spain	5

Insight: 10 of **13** customers in the **USA** are **potential loyalists.**

Recommendation: Capitalize on the strong potential in countries like the USA by implementing strategies to convert the **10 potential loyalists** into **champions**.

- **Customer Segmentation:**
  - **OrderValue-Wise:**
    - **Objective:** I wanted to categorize customers based on their average order revenue value into [‘High-value’, ‘Medium-value’, ‘Low-value’] categories.

For this, I created a view to hold the Average order value for each customer in my database:

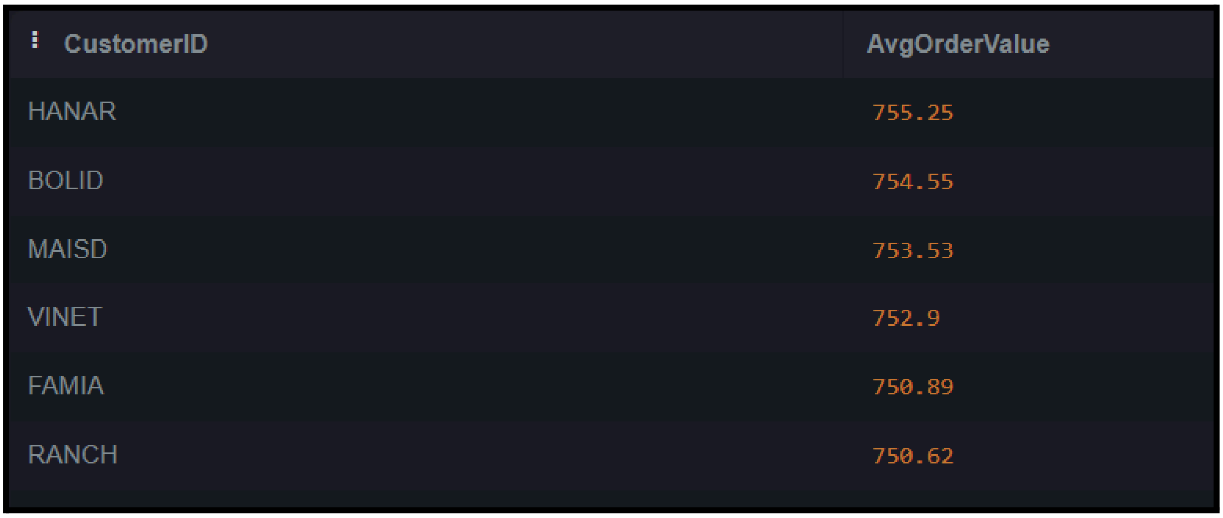


```
CREATE VIEW OrderValue_CustomerView
AS
SELECT Orders.customerid, ROUND(AVG(unitprice*quantity*(1-discount)),2) AS AvgOrderValue
FROM Orders
JOIN "Order Details" od
ON od.orderId = Orders.OrderID
GROUP BY 1
ORDER BY 2 DESC

SELECT *
FROM OrderValue_CustomerView
```

Explanation of the query:

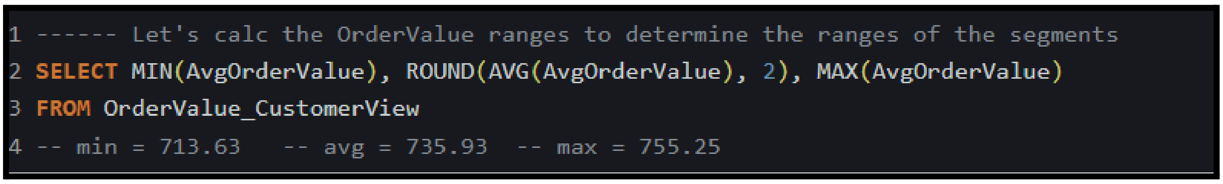
- *To calculate the **AvgOrderValue,** I used the **AVG()** function on the orders prices for each **customer_id***
- ***GROUP BY** clause to map the aggregation **AVG()** function on each **customer orders.***

Query Result:



CustomerID	AvgOrderValue
HANAR	755.25
BOLID	754.55
MAISD	753.53
VINET	752.9
FAMIA	750.89
RANCH	750.62

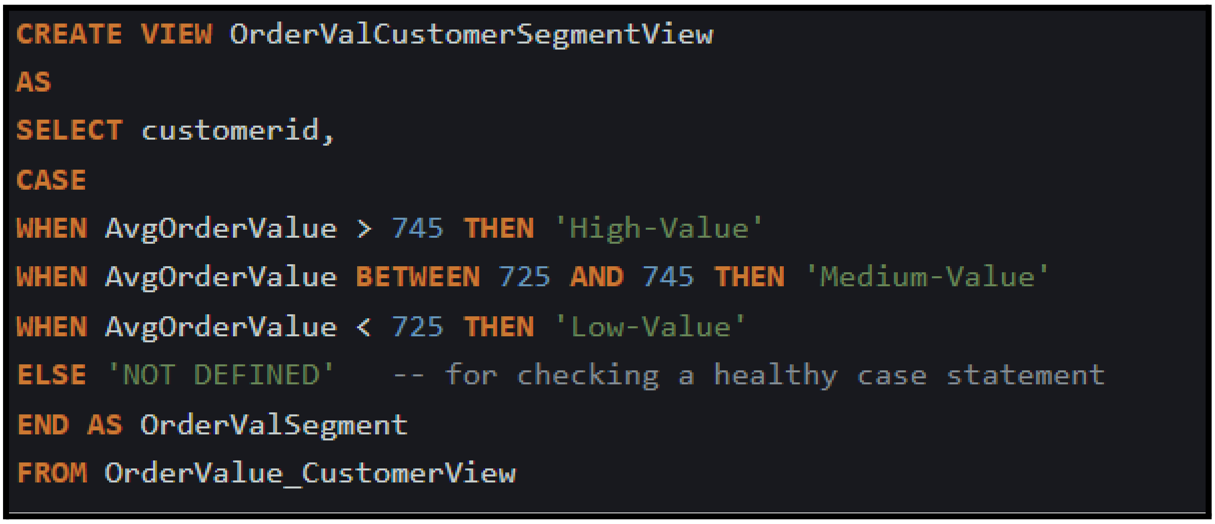
- **Customer Segmentation by their Average Order Value:**
  - I followed the same technique of segmentation that I used in the RFM customer segmentation
    - Let’s look at the dispersion of the customers' AvgOrderValue to determine the segments ranges



```
------ Let's calc the OrderValue ranges to determine the ranges of the segments
SELECT MIN(AvgOrderValue), ROUND(AVG(AvgOrderValue), 2), MAX(AvgOrderValue)
FROM OrderValue_CustomerView
-- min = 713.63    -- avg = 735.93   -- max = 755.25
```

  - **Defining Segments Boundaries:**
    - **For High-Value:**
      - AvgOrderValue > 745$
    - **For Medium-Value:**
      - AvgOrderValue between 725$ and 745
    - **For low-Value:**
      - AvgOrderValue < 725$

I created a view for the query doing the categorization work:



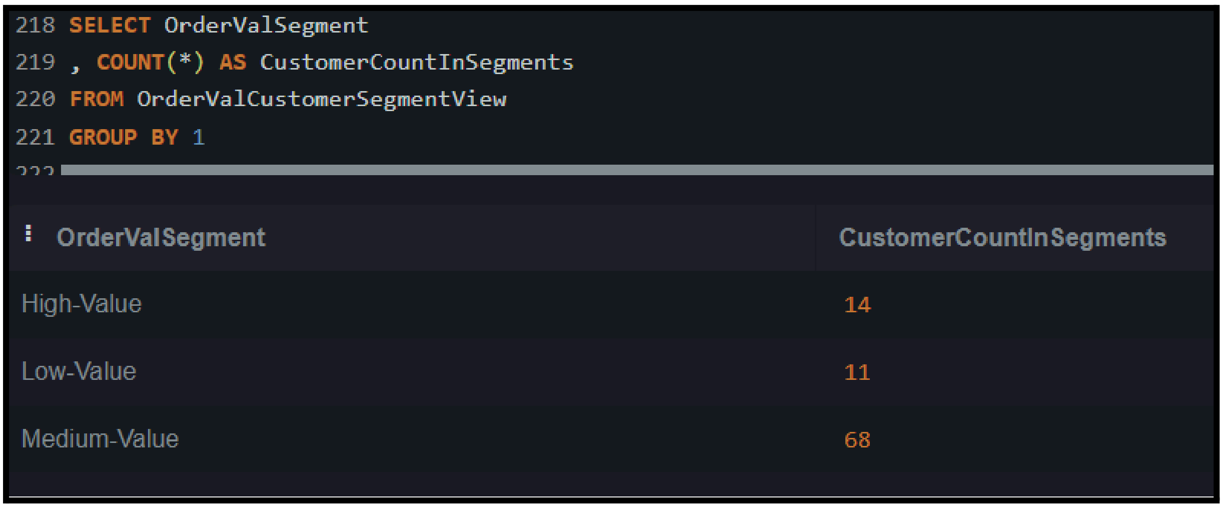
```
CREATE VIEW OrderValCustomerSegmentView
AS
SELECT customerid,
CASE
WHEN AvgOrderValue > 745 THEN 'High-Value'
WHEN AvgOrderValue BETWEEN 725 AND 745 THEN 'Medium-Value'
WHEN AvgOrderValue < 725 THEN 'Low-Value'
ELSE 'NOT DEFINED'   -- for checking a healthy case statement
END AS OrderValSegment
FROM OrderValue_CustomerView
```

Explanation of the query:

*Here, I used the **CASE** statement to check in which category will a customer with a certain **AvgOrderValue** be.*

Query Result:

*I used the **GROUP BY** clause to see how many customers reside in each category.*



```
SELECT OrderValSegment
, COUNT(*) AS CustomerCountInSegments
FROM OrderValCustomerSegmentView
GROUP BY 1
```

OrderValSegment	CustomerCountInSegments
High-Value	14
Low-Value	11
Medium-Value	68

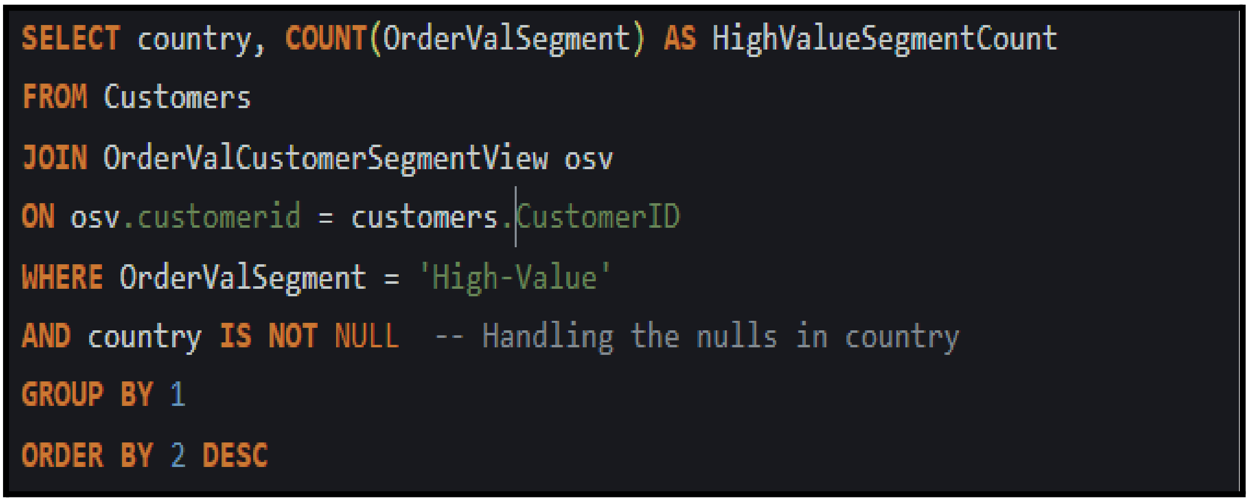
Insight:

- Most of our customers are of **Medium-OrderValue** Class about **73%**
- **11%** are of **Low-OrderValue** Class

- ## Some Customer Segments Geographic Analysis

For further analysis of the customer segments, let's see how our customer categories are distributed…

○ High-Value Class:

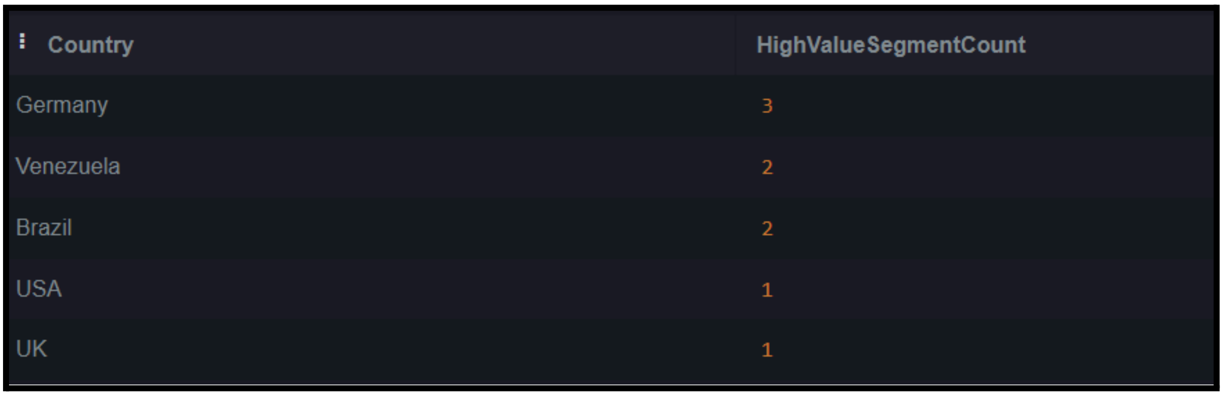


```
SELECT country, COUNT(OrderValSegment) AS HighValueSegmentCount
FROM Customers
JOIN OrderValCustomerSegmentView osv
ON osv.customerid = customers.CustomerID
WHERE OrderValSegment = 'High-Value'
AND country IS NOT NULL  -- Handling the nulls in country
GROUP BY 1
ORDER BY 2 DESC
```

Explanation of the query:

- **JOINing** between the ***[Customers]*** table and the ***[OrderValCustomerSegmentView]*** *then* ***Grouping*** *to count the customers with* *High-OrderValue* *in each* ***Country***

Query Result:

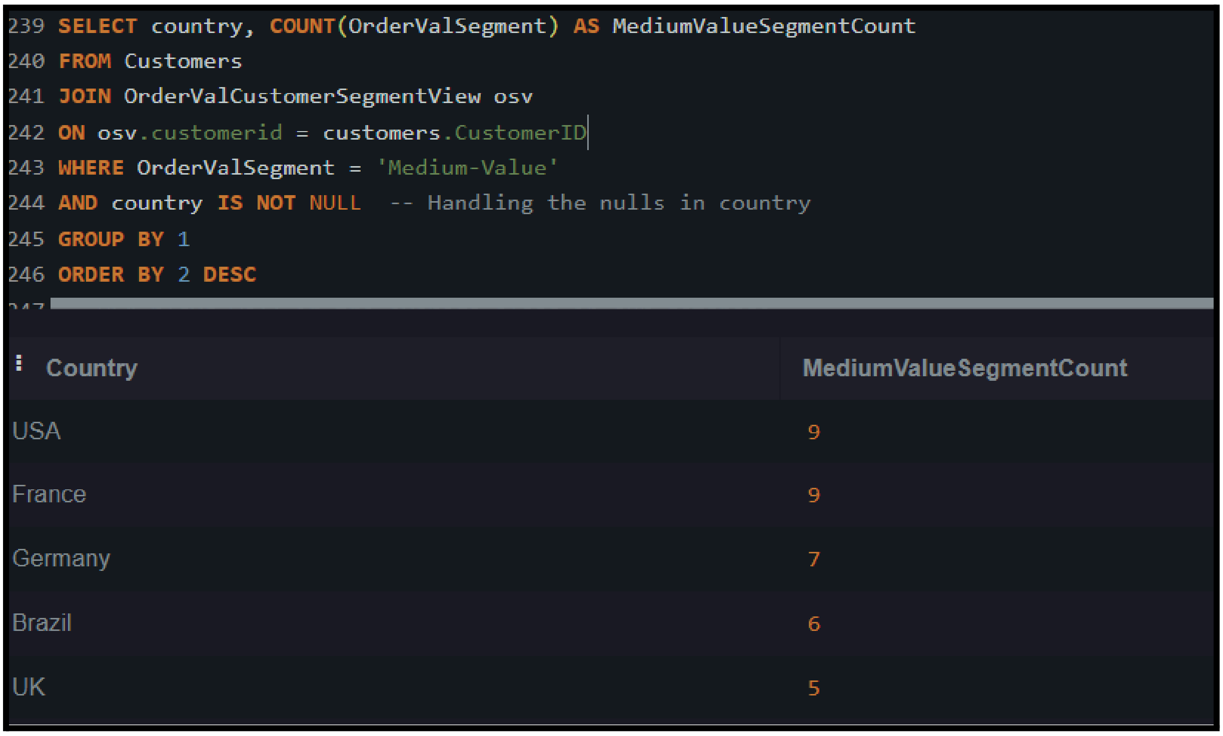


Country	HighValueSegmentCount
Germany	3
Venezuela	2
Brazil	2
USA	1
UK	1

Insight:

- **Germany** has the highest number of customers with High-OrderValue (**3** of Germany's **11** customers)
- **Venezuela** with only **4** customers had **2** of them in the High-OrderValue Class **!!INTERESTING!!**

○ Medium-Value Class:



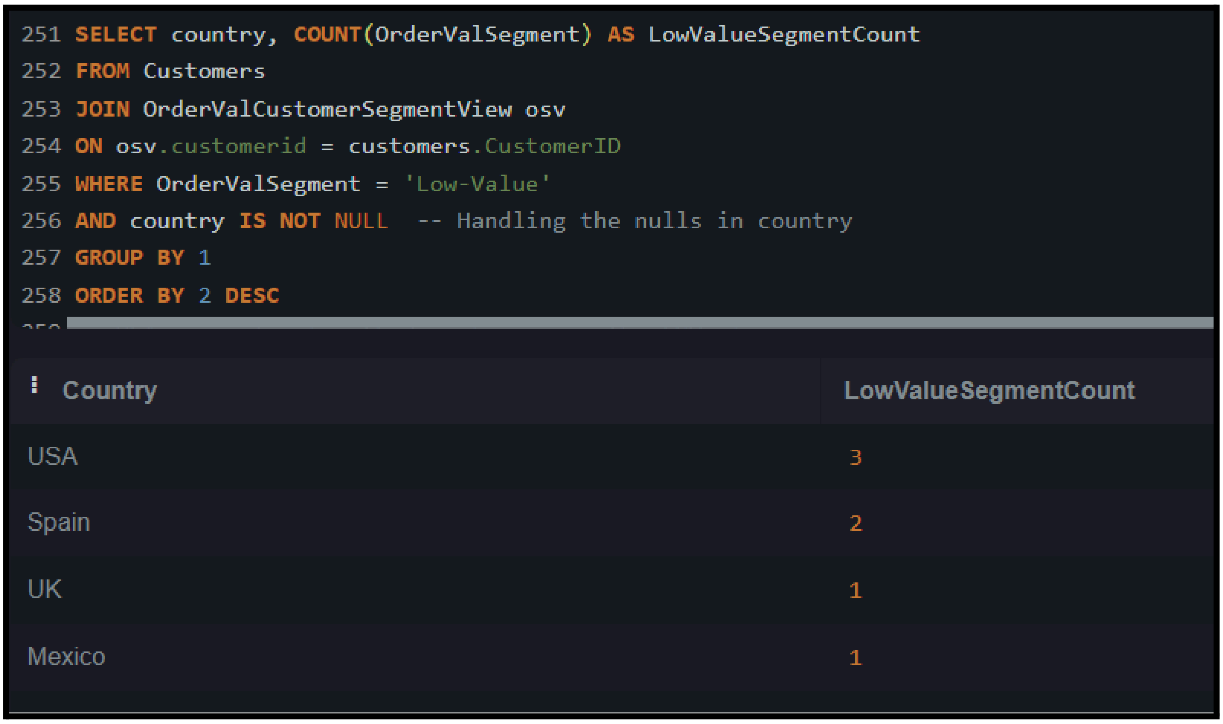
```sql
SELECT country, COUNT(OrderValSegment) AS MediumValueSegmentCount
FROM Customers
JOIN OrderValCustomerSegmentView osv
ON osv.customerid = customers.CustomerID
WHERE OrderValSegment = 'Medium-Value'
AND country IS NOT NULL  -- Handling the nulls in country
GROUP BY 1
ORDER BY 2 DESC
```

Country	MediumValueSegmentCount
USA	9
France	9
Germany	7
Brazil	6
UK	5

Insight:

- **USA** plays well in the Medium class in our database.
- **Germany** comes in the **top 3** countries with Medium-Value Class
- **All countries** that had appeared in the **Champion** segment are in the **top 5** in the Medium-Value Segment

○ Low-Value Class:



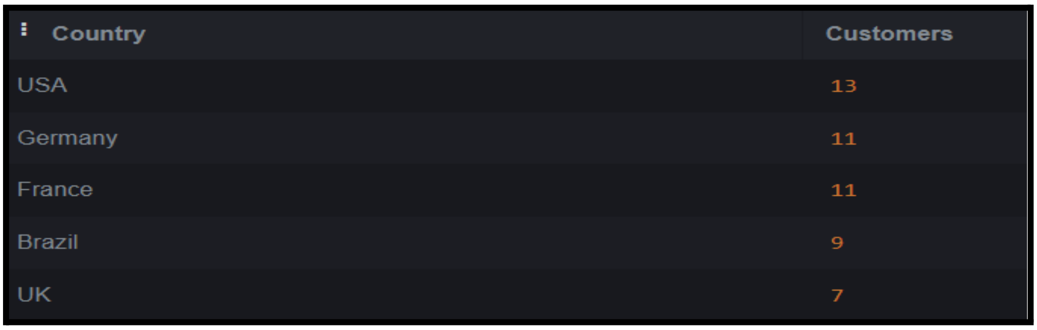
```sql
SELECT country, COUNT(OrderValSegment) AS LowValueSegmentCount
FROM Customers
JOIN OrderValCustomerSegmentView osv
ON osv.customerid = customers.CustomerID
WHERE OrderValSegment = 'Low-Value'
AND country IS NOT NULL  -- Handling the nulls in country
GROUP BY 1
ORDER BY 2 DESC
```

Country	LowValueSegmentCount
USA	3
Spain	2
UK	1
Mexico	1

Insight:

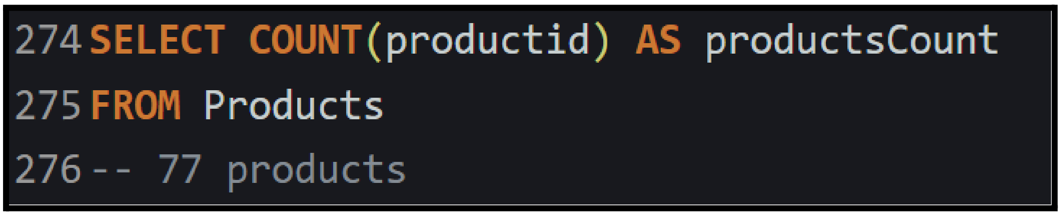
- The **USA** comes **first** with **3** customers in the Low class.

  PS: the **USA** is the **highest** country with customers in our database.



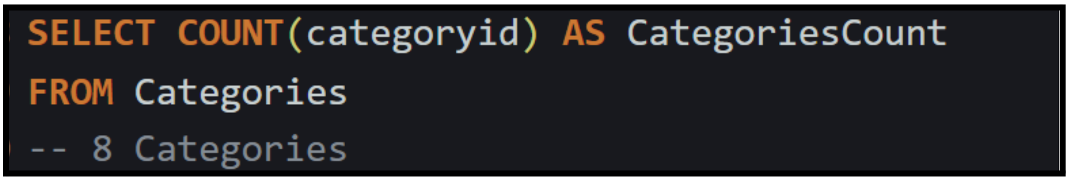
Country	Customers
USA	13
Germany	11
France	11
Brazil	9
UK	7

- **Product Analysis:**
  - **Some Warm-Up Product Analysis:**
    - **How many products in my database?**



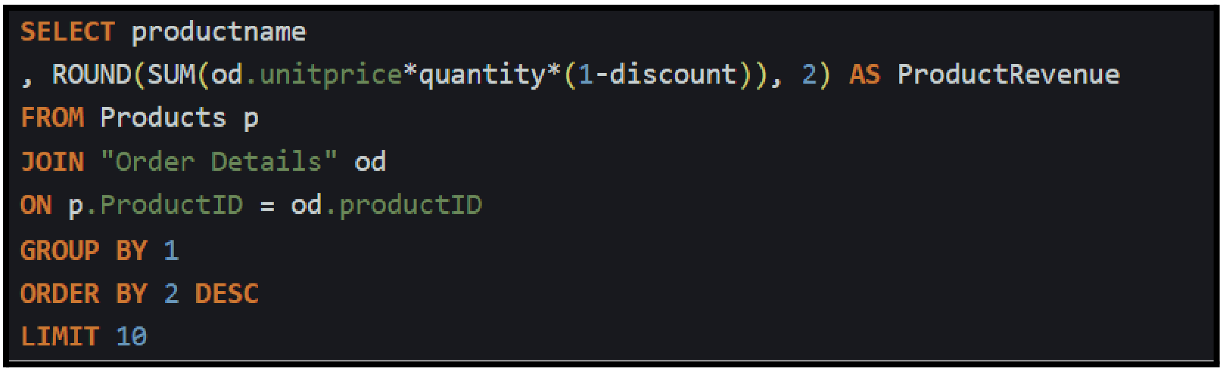
```sql
SELECT COUNT(productid) AS productsCount
FROM Products
-- 77 products
```

    - **How many product Categories?**



```sql
SELECT COUNT(categoryid) AS CategoriesCount
FROM Categories
-- 8 Categories
```

  - **Top 10 products (Revenue-wise):**

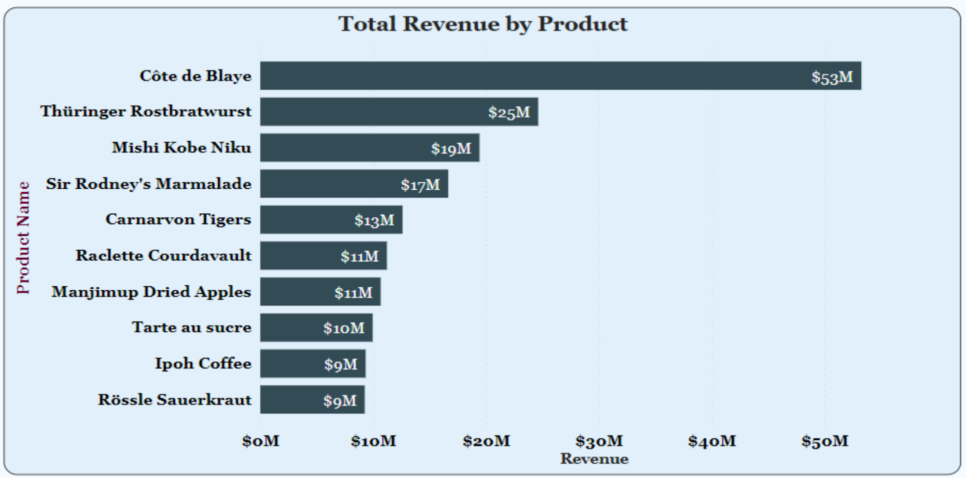


```sql
SELECT productname
, ROUND(SUM(od.unitprice*quantity*(1-discount)), 2) AS ProductRevenue
FROM Products p
JOIN "Order Details" od
ON p.ProductID = od.productID
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
```

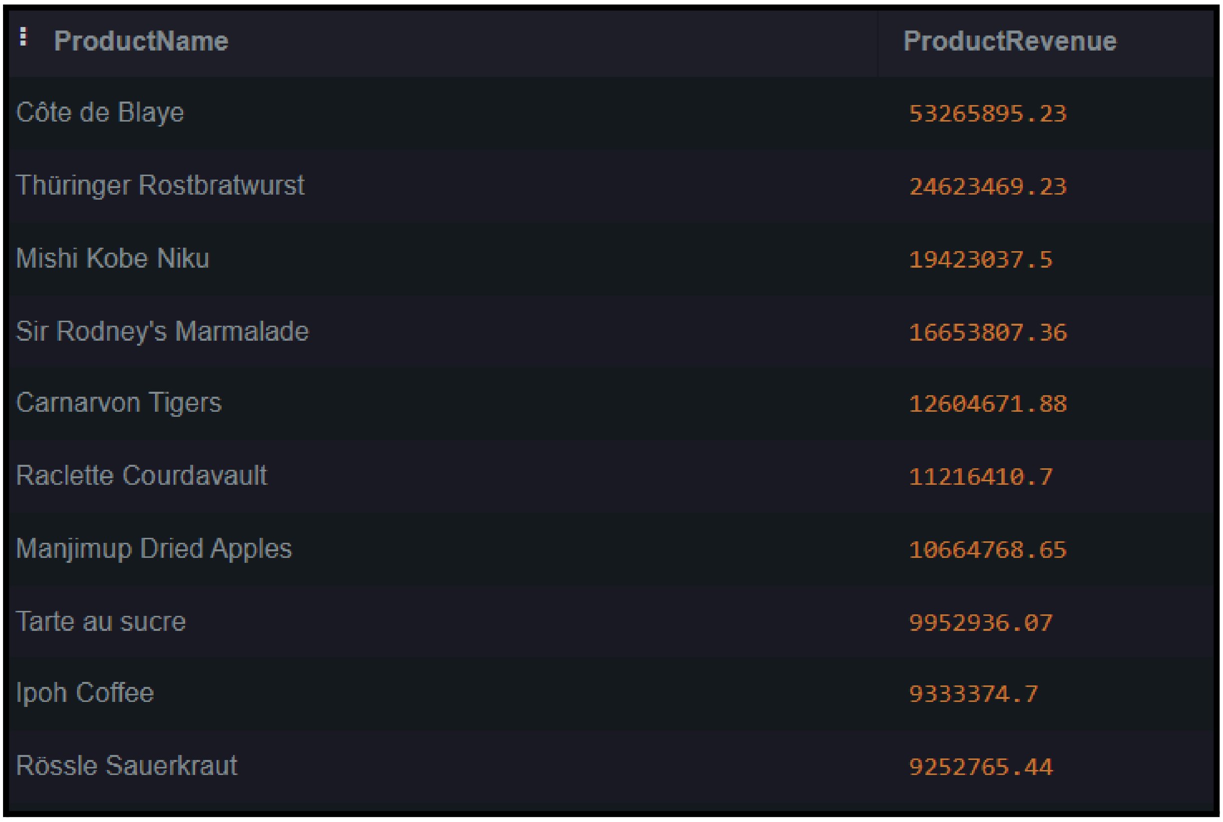
Explanation of the query:

- ***JOINed** between **[Products]** table and **[Order Details]** table to **aggregate** over each product and find the revenue generated by it.*

***Query Result:** (visualization for better result presentation)*

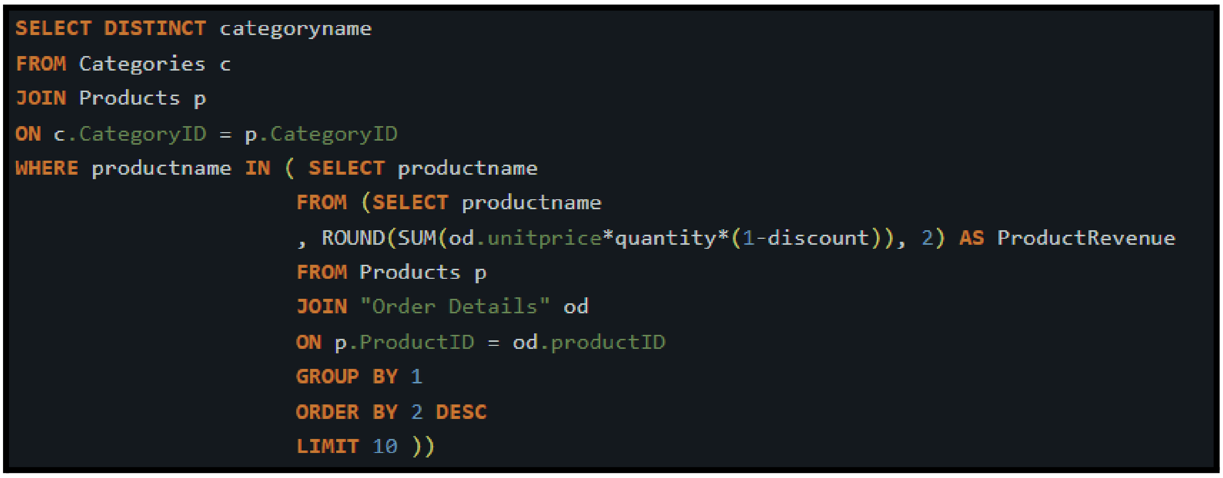


Query Result:



ProductName	ProductRevenue
Côte de Blaye	53265895.23
Thüringer Rostbratwurst	24623469.23
Mishi Kobe Niku	19423037.5
Sir Rodney's Marmalade	16653807.36
Carnarvon Tigers	12604671.88
Raclette Courdavault	11216410.7
Manjimup Dried Apples	10664768.65
Tarte au sucre	9952936.07
Ipoh Coffee	9333374.7
Rössle Sauerkraut	9252765.44

- **In what categories are my top revenue generating products?**



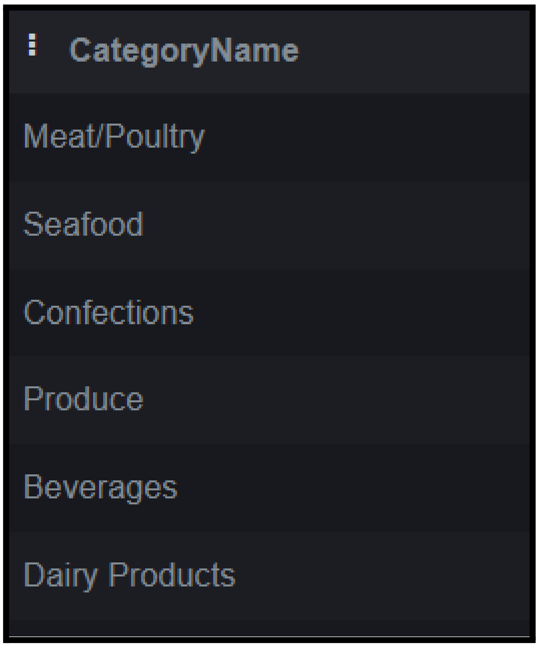
```
SELECT DISTINCT categoryname
FROM Categories c
JOIN Products p
ON c.CategoryID = p.CategoryID
WHERE productname IN ( SELECT productname
                      FROM (SELECT productname
                      , ROUND(SUM(od.unitprice*quantity*(1-discount)), 2) AS ProductRevenue
                      FROM Products p
                      JOIN "Order Details" od
                      ON p.ProductID = od.productID
                      GROUP BY 1
                      ORDER BY 2 DESC
                      LIMIT 10 ))
```

Explanation of the query:

- *JOINing between **[Products]** table and **[Categories]** table to select the **category_name***
- ***Filtering** on the **product_names** in the **"top_products" subquery** that we already created in the previous step.*

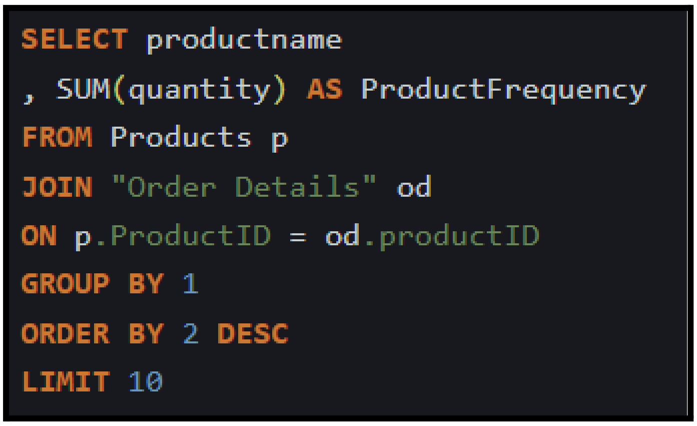
Query Result:

The top products are from these categories:



CategoryName
Meat/Poultry
Seafood
Confections
Produce
Beverages
Dairy Products

- ### Top 10 products (Frequency-wise):

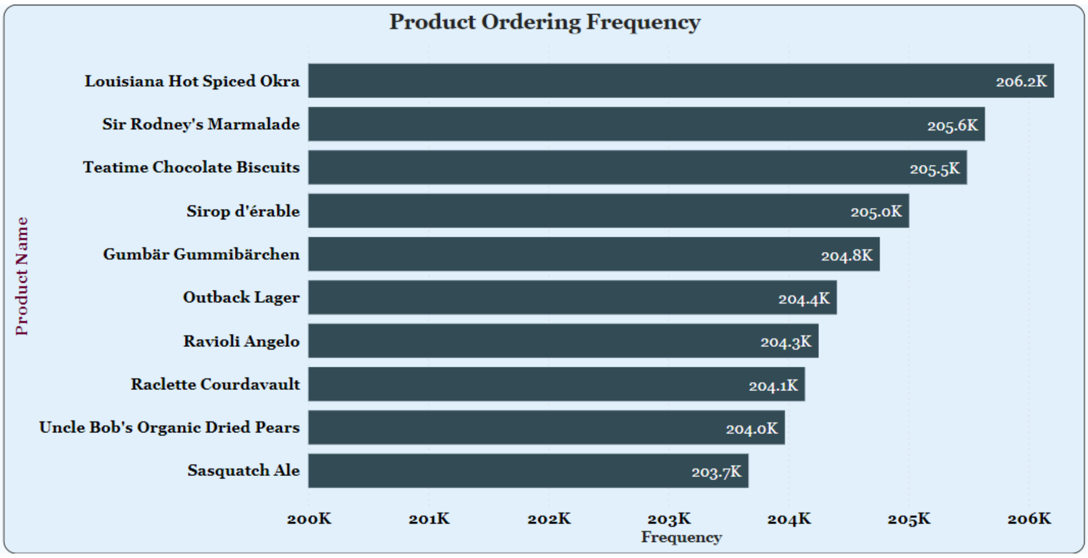


```
SELECT productname
, SUM(quantity) AS ProductFrequency
FROM Products p
JOIN "Order Details" od
ON p.ProductID = od.productID
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
```

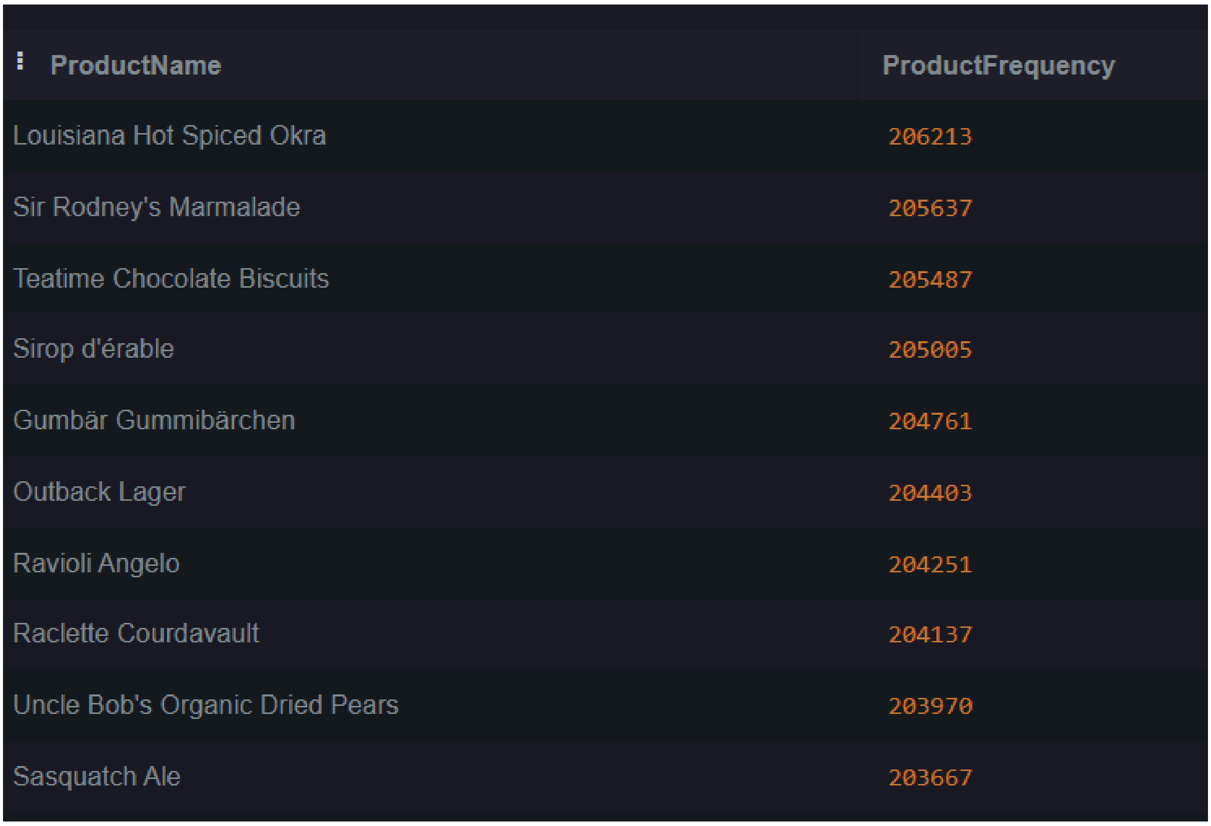
Explanation of the query:

- ***SUM()*** *the quantities sold for each product.*
- *After* ***JOIN****ing the* ***[Products]*** *table with the* ***[Order Details]*** *table, ordering* *<u>descending</u>*, *limiting top* ***10.***

Query Result:

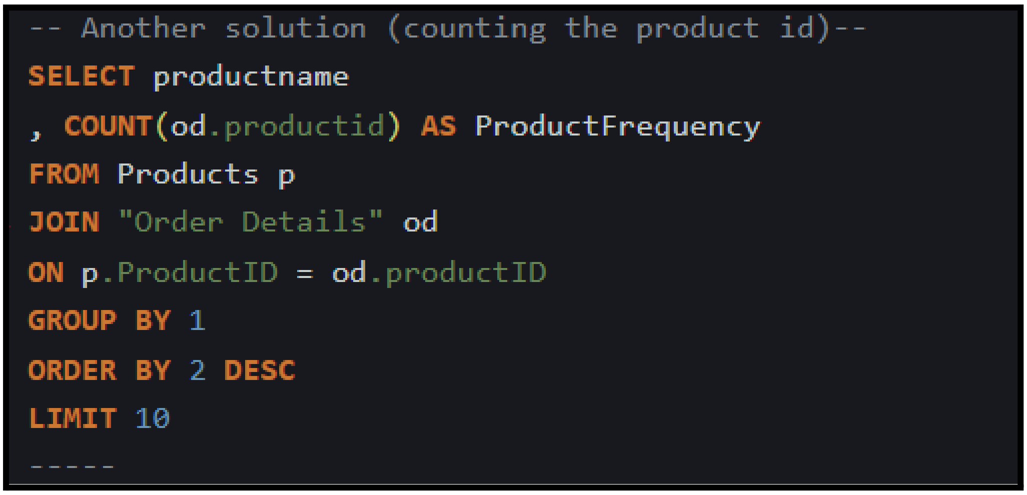


Query result:



ProductName	ProductFrequency
Louisiana Hot Spiced Okra	206213
Sir Rodney's Marmalade	205637
Teatime Chocolate Biscuits	205487
Sirop d'érable	205005
Gumbär Gummibärchen	204761
Outback Lager	204403
Ravioli Angelo	204251
Raclette Courdavault	204137
Uncle Bob's Organic Dried Pears	203970
Sasquatch Ale	203667

- ***Another Solution :***

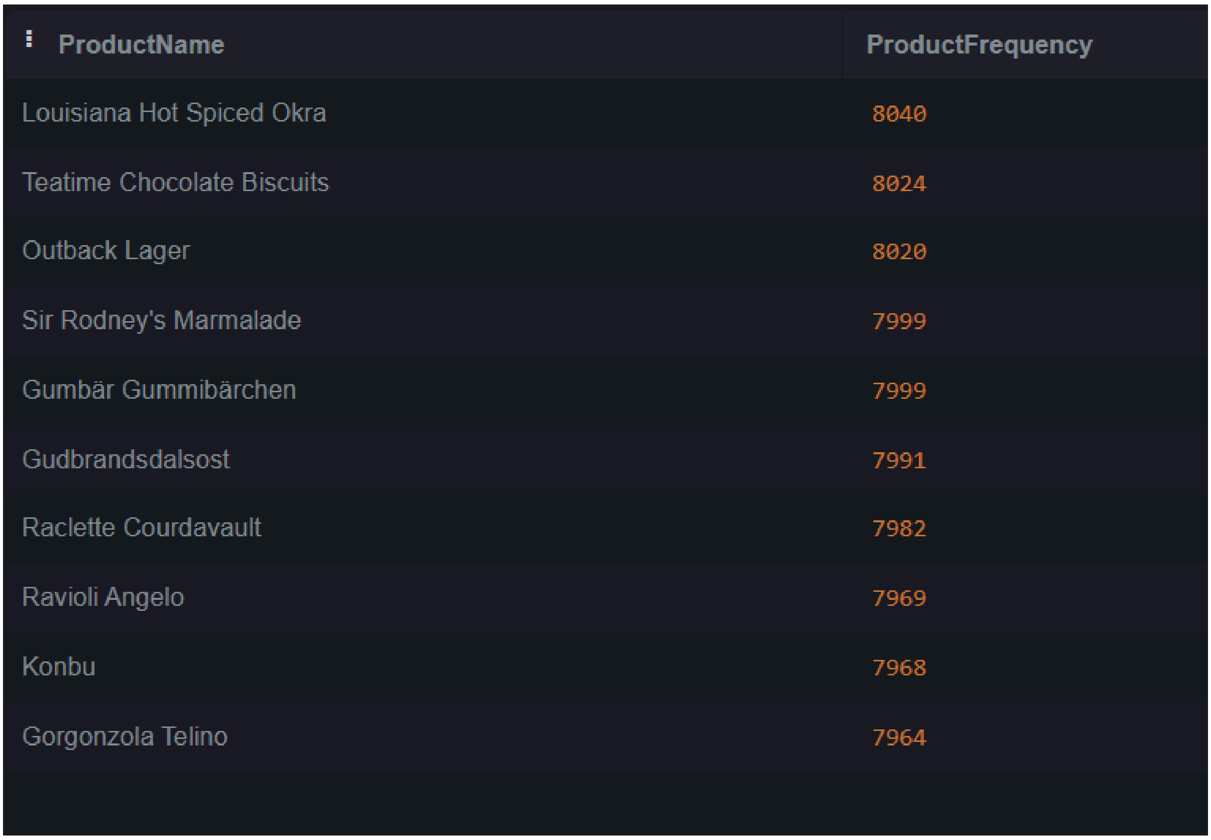


```
-- Another solution (counting the product id)--
SELECT productname
, COUNT(od.productid) AS ProductFrequency
FROM Products p
JOIN "Order Details" od
ON p.ProductID = od.productID
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
-----
```

Explanation of the query:

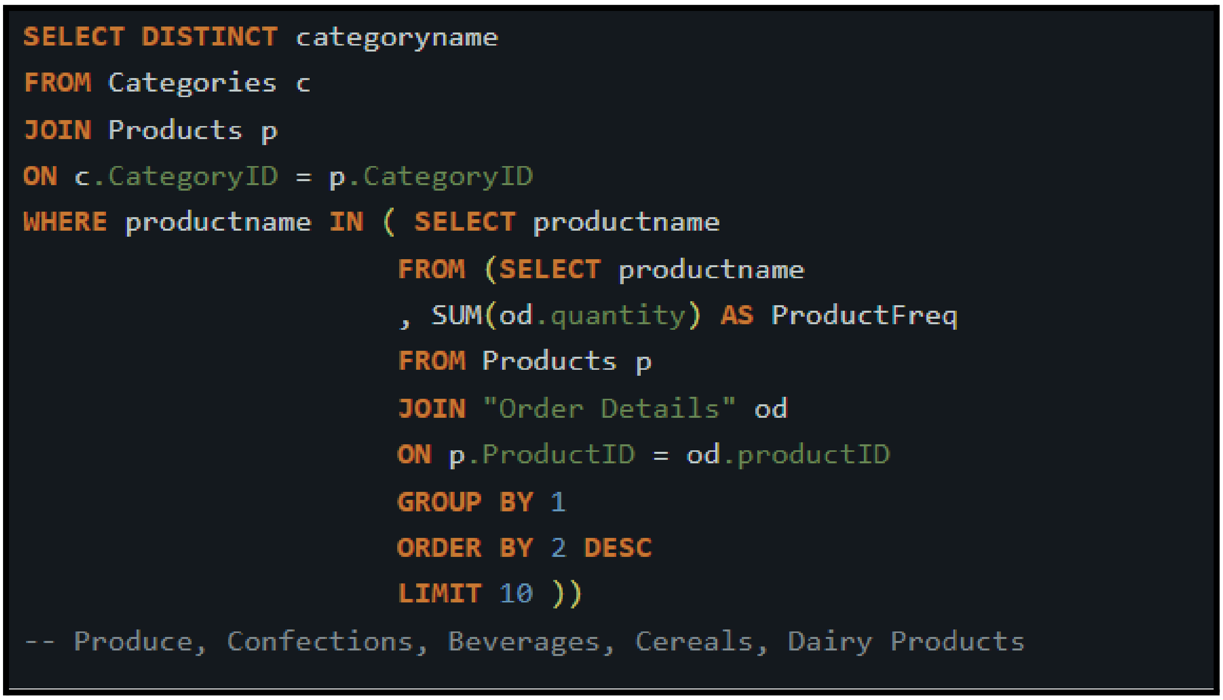
- ***COUNTing** the number of times each product appears in the **[Order Details]** table .*
- *After **JOINing** the **[Products]** table with the **[Order Details]** table, ordering <u>descending</u>, limiting top **10.***

Query Result



ProductName	ProductFrequency
Louisiana Hot Spiced Okra	8040
Teatime Chocolate Biscuits	8024
Outback Lager	8020
Sir Rodney's Marmalade	7999
Gumbär Gummibärchen	7999
Gudbrandsdalsost	7991
Raclette Courdavault	7982
Ravioli Angelo	7969
Konbu	7968
Gorgonzola Telino	7964

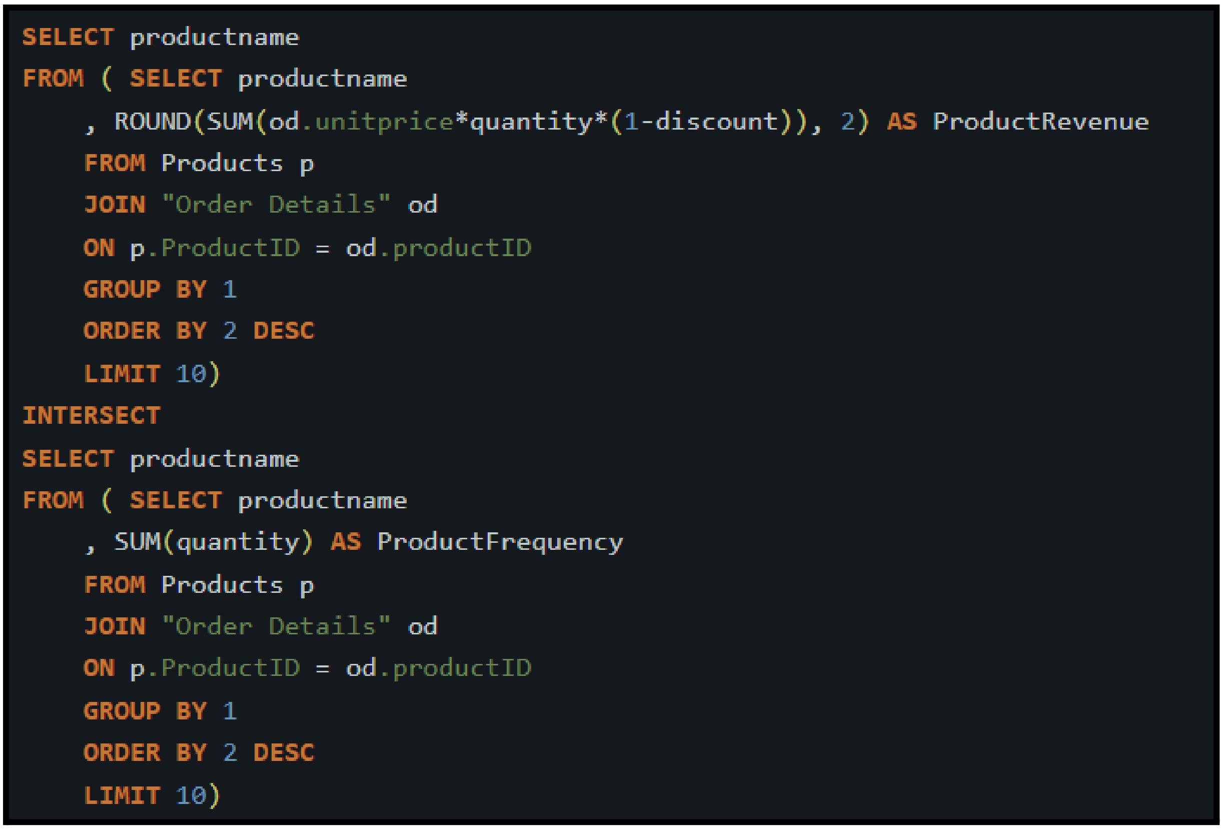
- **In what categories are my top frequently ordered products?**



```
SELECT DISTINCT categoryname
FROM Categories c
JOIN Products p
ON c.CategoryID = p.CategoryID
WHERE productname IN ( SELECT productname
                      FROM (SELECT productname
                      , SUM(od.quantity) AS ProductFreq
                      FROM Products p
                      JOIN "Order Details" od
                      ON p.ProductID = od.productID
                      GROUP BY 1
                      ORDER BY 2 DESC
                      LIMIT 10 ))
-- Produce, Confections, Beverages, Cereals, Dairy Products
```

Insight: Some **top** product categories are **Dairy** and **Confections**

- **Do the most frequent ordered products make the highest revenue??**



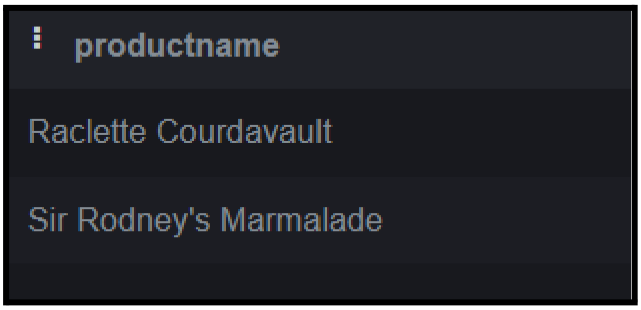
```
SELECT productname
FROM ( SELECT productname
    , ROUND(SUM(od.unitprice*quantity*(1-discount)), 2) AS ProductRevenue
    FROM Products p
    JOIN "Order Details" od
    ON p.ProductID = od.productID
    GROUP BY 1
    ORDER BY 2 DESC
    LIMIT 10)
INTERSECT
SELECT productname
FROM ( SELECT productname
    , SUM(quantity) AS ProductFrequency
    FROM Products p
    JOIN "Order Details" od
    ON p.ProductID = od.productID
    GROUP BY 1
    ORDER BY 2 DESC
    LIMIT 10)
```

Explanation of the query:

- ***INTERSECT****ing between the two previous queries (top 10 products in revenue) and (top 10 products in frequency)*

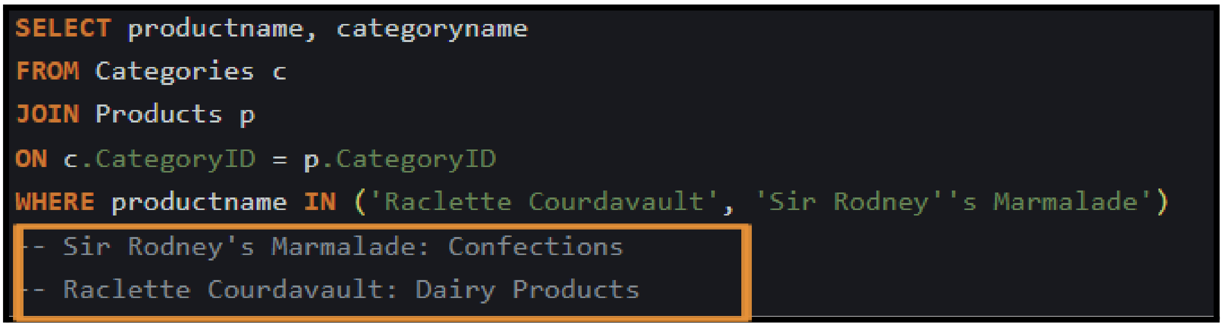
Query Result:

Those two products happened to generate the highest revenue and being ordered frequently.



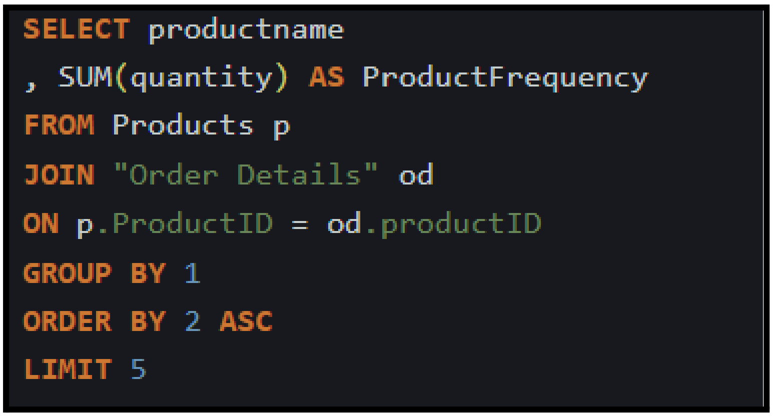
productname
Raclette Courdavault
Sir Rodney's Marmalade

- **From what category are these products? (high revenue, frequency)**



```
SELECT productname, categoryname
FROM Categories c
JOIN Products p
ON c.CategoryID = p.CategoryID
WHERE productname IN ('Raclette Courdavault', 'Sir Rodney''s Marmalade')
-- Sir Rodney's Marmalade: Confections
-- Raclette Courdavault: Dairy Products
```

- **What are the least 5 products with volume sales (Slow Movers)?**

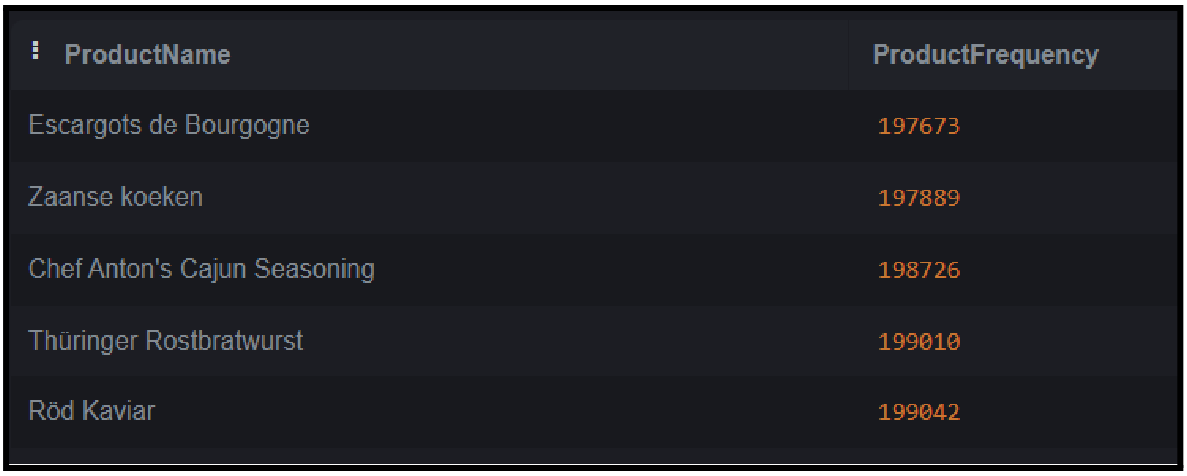


```
SELECT productname
, SUM(quantity) AS ProductFrequency
FROM Products p
JOIN "Order Details" od
ON p.ProductID = od.productID
GROUP BY 1
ORDER BY 2 ASC
LIMIT 5
```

Explanation of the query:

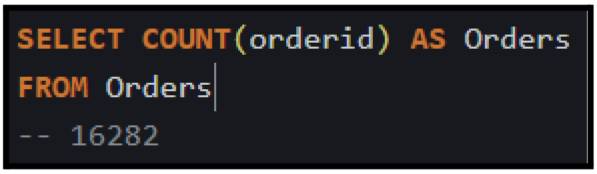
- ***SUM()** the quantities sold for each product.*
- *After **JOIN**ing the **[Products]** table with the **[Order Details]** table, ordering <u>ascending</u>, limiting the first **5***

Query Result:



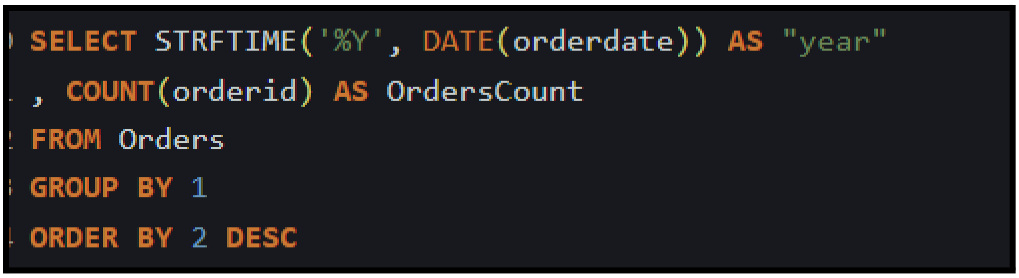
ProductName	ProductFrequency
Escargots de Bourgogne	197673
Zaanse koeken	197889
Chef Anton's Cajun Seasoning	198726
Thüringer Rostbratwurst	199010
Röd Kaviar	199042

- **Order Analysis:**
    - **Some Warm-up Order Analysis:**
        - **How many orders are there?**



```sql
SELECT COUNT(orderid) AS Orders
FROM Orders
-- 16282
```

    - **Seasonality Order Analysis:**
        - **What's the distribution of these orders over years?**

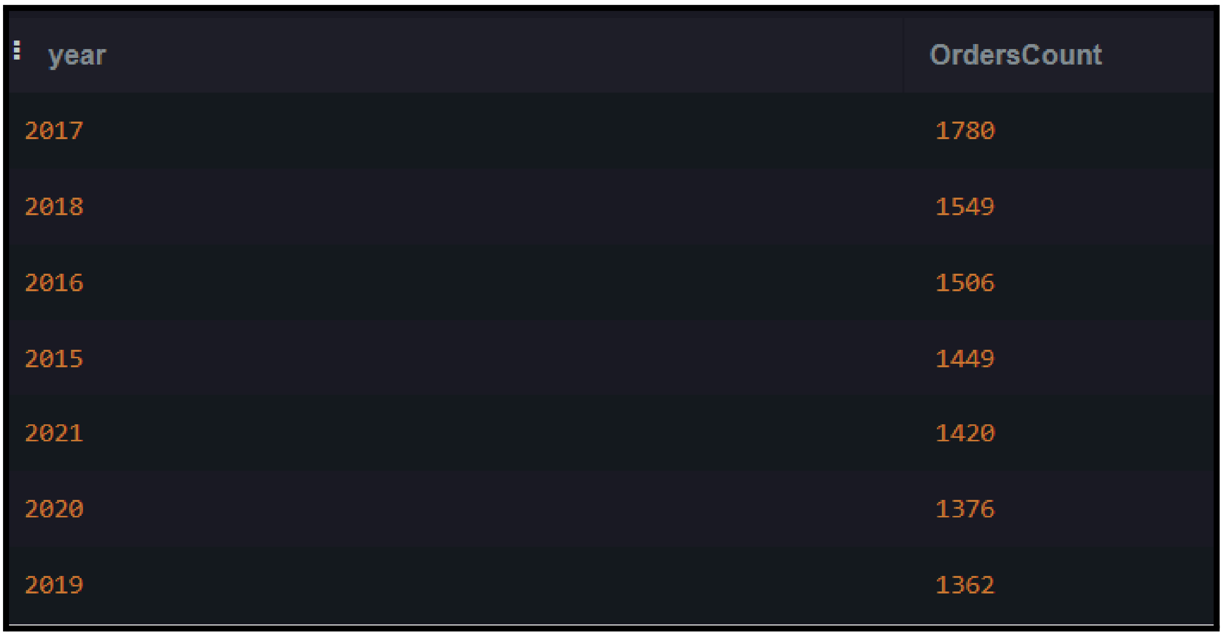


```sql
SELECT STRFTIME('%Y', DATE(orderdate)) AS "year"
, COUNT(orderid) AS OrdersCount
FROM Orders
GROUP BY 1
ORDER BY 2 DESC
```

Query Explanation:

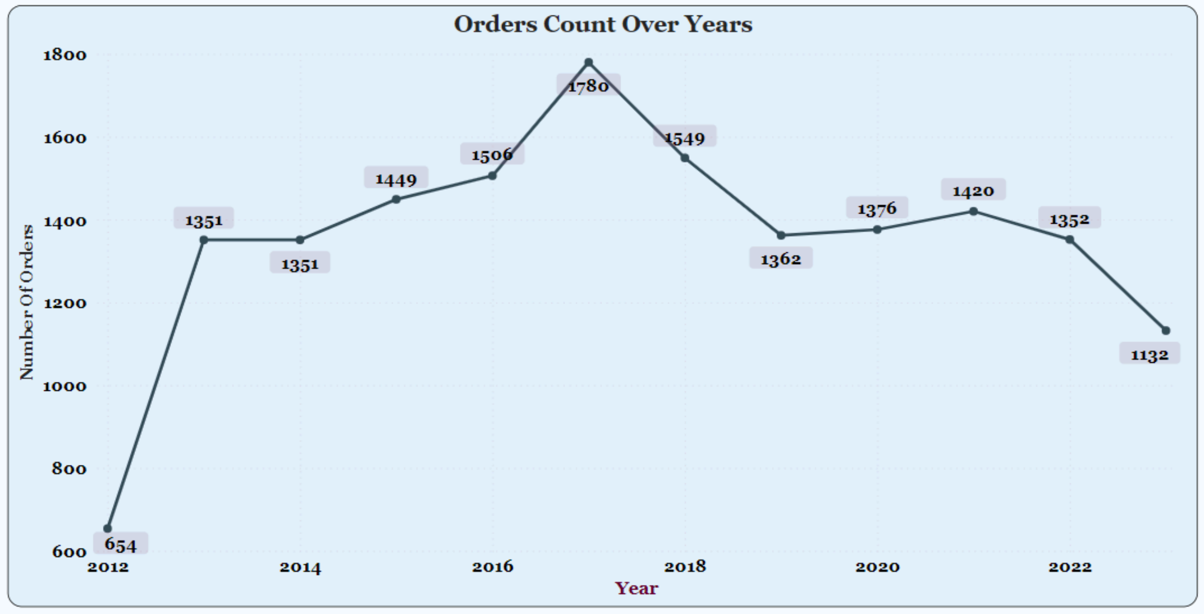
- ***Grouping*** *by the* ***year*** *which is extracted from the orderdate using the* ***STRFTIME()****, whereas for each year , we're* ***counting*** *the number of orders purchased*

Query Result:



year	OrdersCount
2017	1780
2018	1549
2016	1506
2015	1449
2021	1420
2020	1376
2019	1362

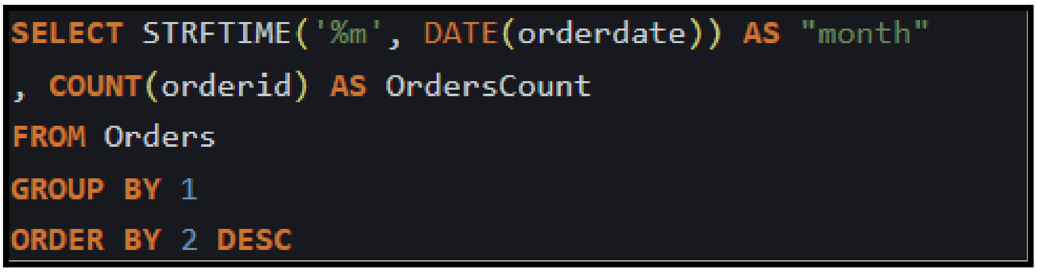
Query Result: *(line chart to visualize the orders count across years from **2012** to **2023**)*



Insight:

- From 2012 to 2016, there was an increasing pattern in the orders made
- In **2017**, we had the highest peak in orders, after which, Orders made started to decade.
- In 2023, we are still decreasing in the number of orders made!

- ■ **What's the distribution of these orders over months?**

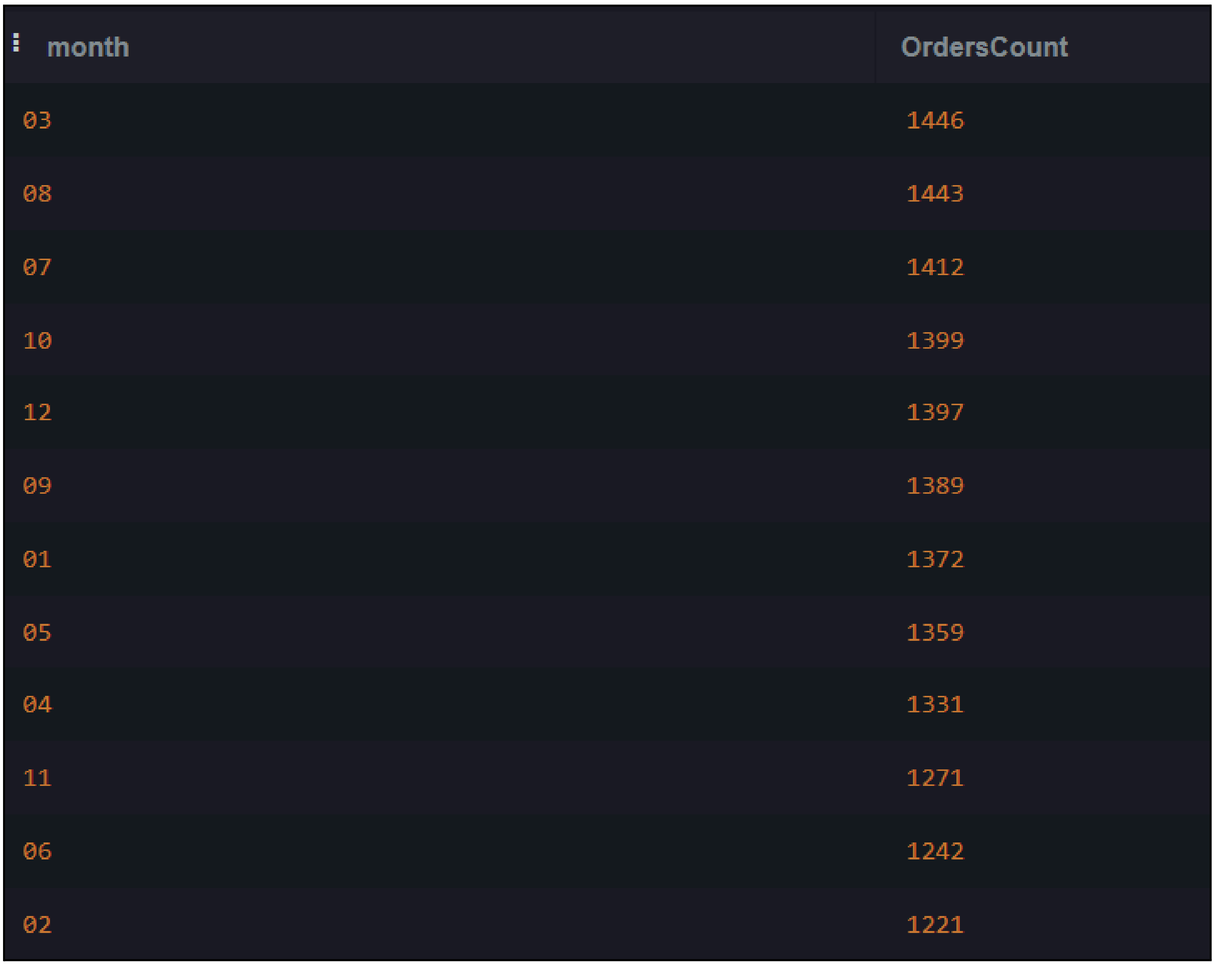


```
SELECT STRFTIME('%m', DATE(orderdate)) AS "month"
, COUNT(orderid) AS OrdersCount
FROM Orders
GROUP BY 1
ORDER BY 2 DESC
```

Query Explanation:

- ***Grouping*** *by the* ***month*** *which is extracted from the orderdate using the* ***STRFTIME()****, whereas for each* <u>*month*</u> *we're* ***counting*** *the* <u>*number of orders purchased*</u>

Query Result:

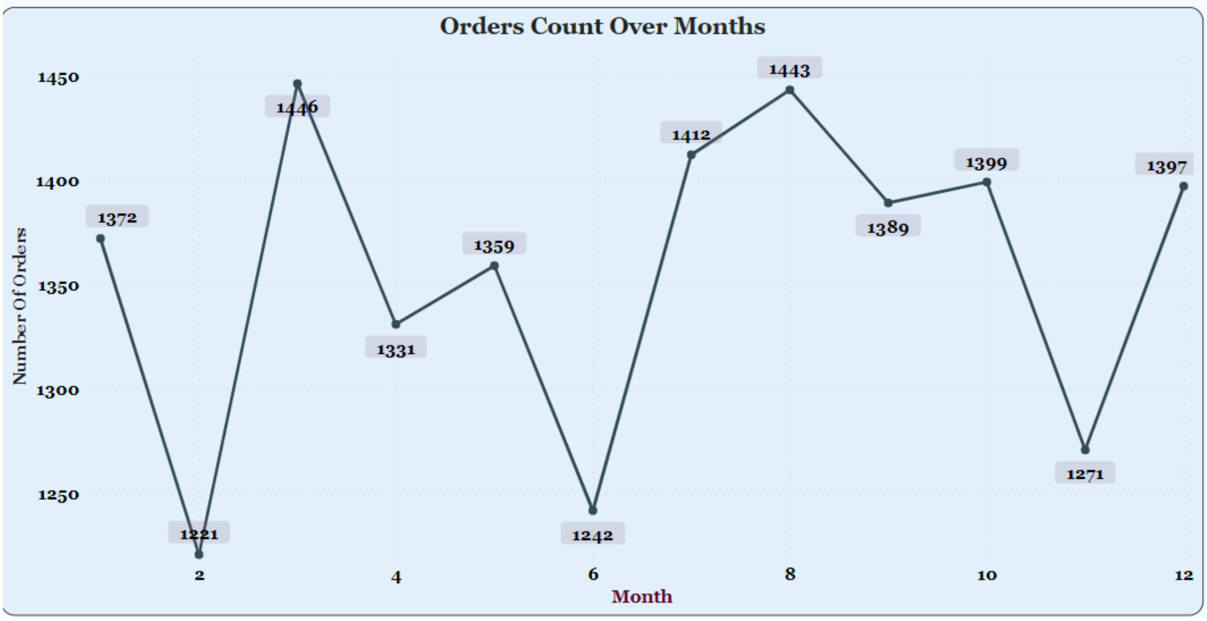


month	OrdersCount
03	1446
08	1443
07	1412
10	1399
12	1397
09	1389
01	1372
05	1359
04	1331
11	1271
06	1242
02	1221

Insight:

- We made most orders in **March** and **August**.

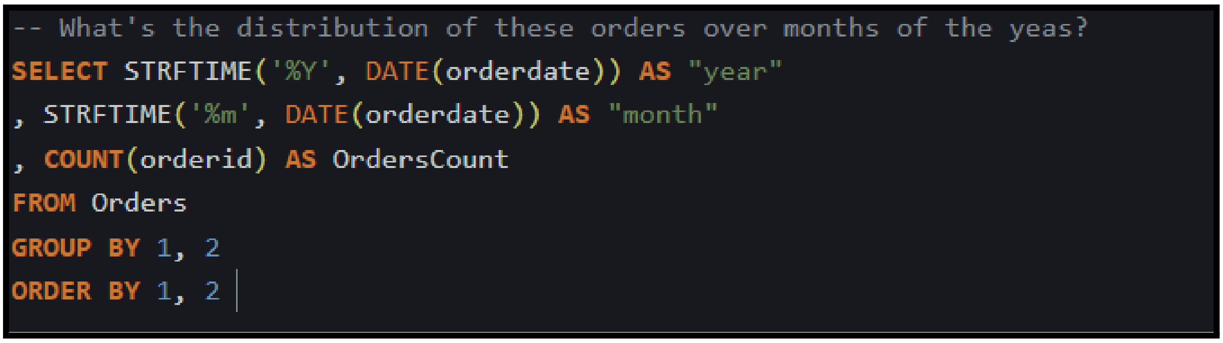
Query Result: *(line chart to visualize the orders count across months from **Jan** to **Dec**)*



Insight:

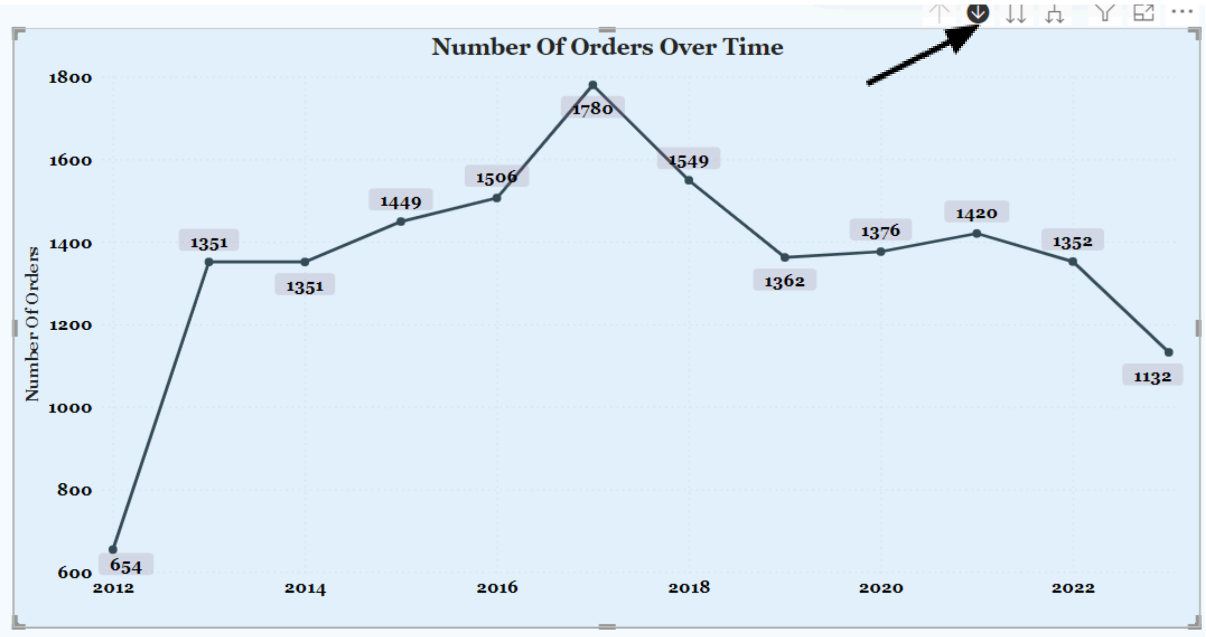
- Number of orders is so **fluctuating** across months.

- **L**et's have a closer look on the Number of orders across months in each year.



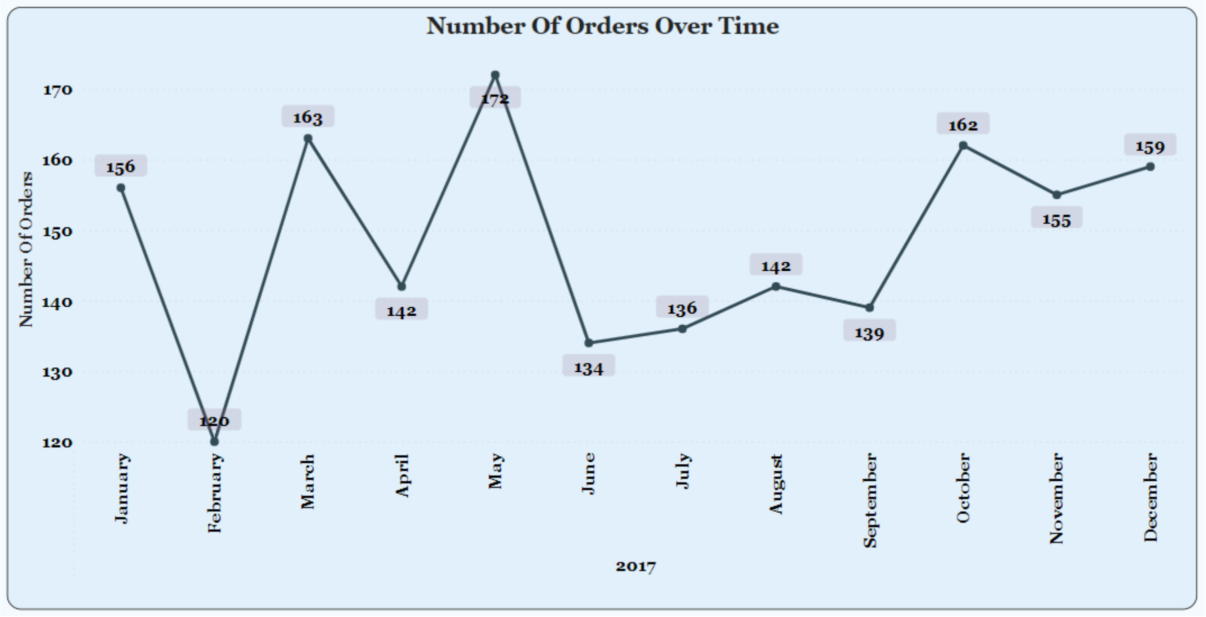
```
-- What's the distribution of these orders over months of the yeas?
SELECT STRFTIME('%Y', DATE(orderdate)) AS "year"
, STRFTIME('%m', DATE(orderdate)) AS "month"
, COUNT(orderid) AS OrdersCount
FROM Orders
GROUP BY 1, 2
ORDER BY 1, 2
```

Query Result: *(line chart to visualize the orders count across Time)*



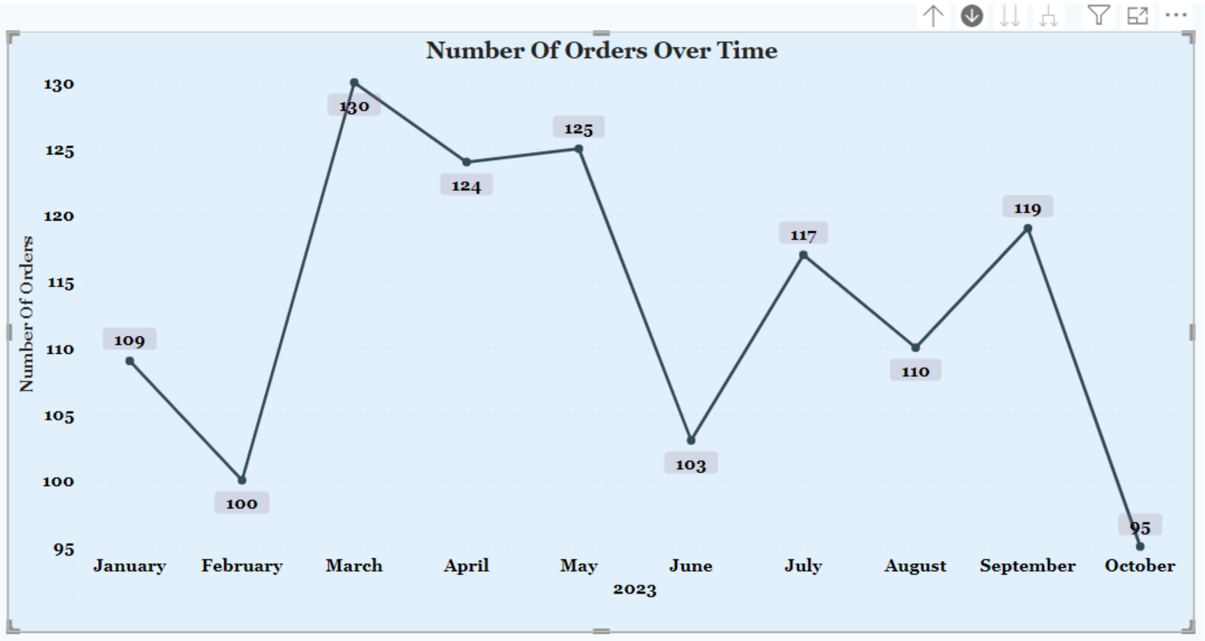
- Let’s **drill down** in this visual across years:

  - In 2017, with the highest orders made:



- Orders were fluctuating between 2017’s months with peaks in May and March.

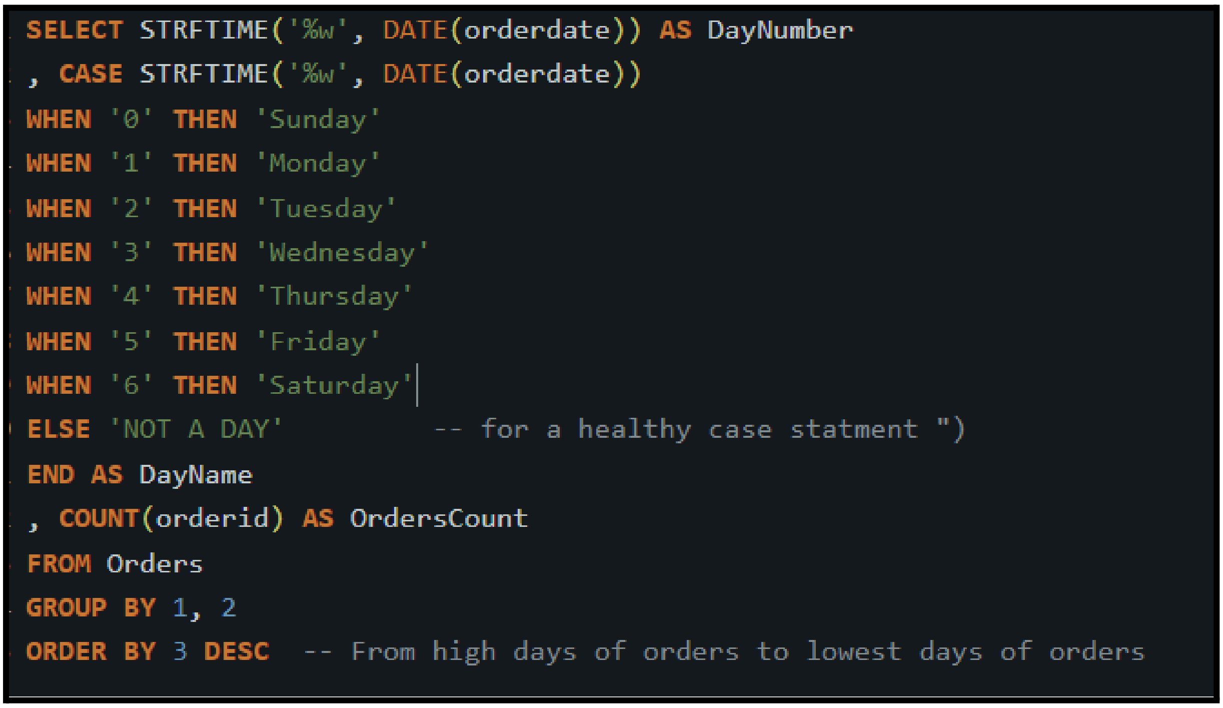
- In 2023, the past year with a below average orders made:



Insight:

- Seems like **May** and **March** are the top months in our orders made in 2023

○ Day-of-the-Week Analysis: the most popular order days

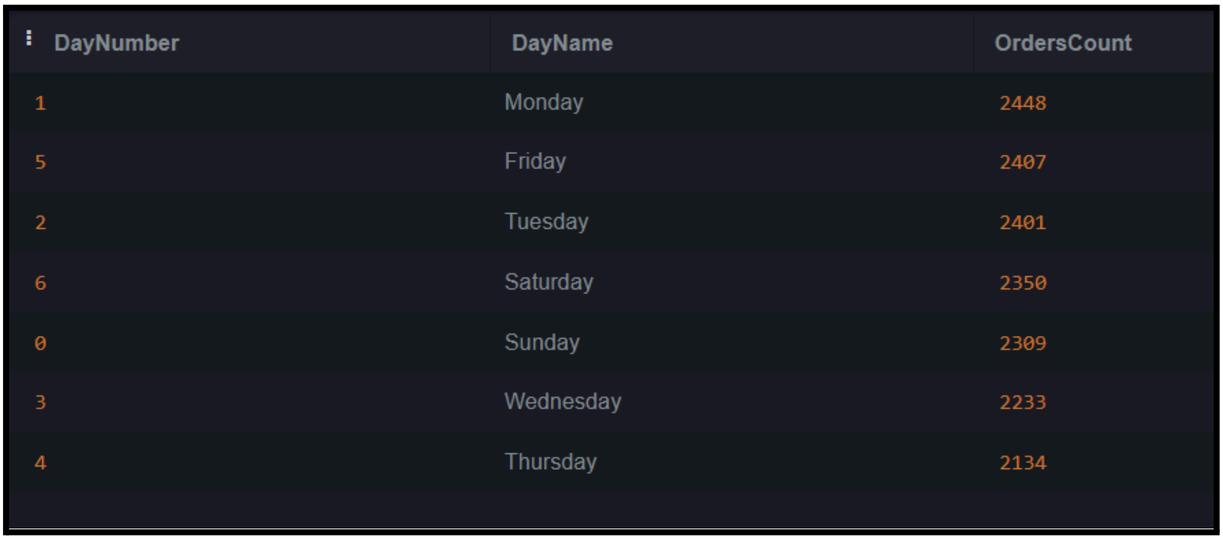


```
SELECT STRFTIME('%w', DATE(orderdate)) AS DayNumber
, CASE STRFTIME('%w', DATE(orderdate))
WHEN '0' THEN 'Sunday'
WHEN '1' THEN 'Monday'
WHEN '2' THEN 'Tuesday'
WHEN '3' THEN 'Wednesday'
WHEN '4' THEN 'Thursday'
WHEN '5' THEN 'Friday'
WHEN '6' THEN 'Saturday'
ELSE 'NOT A DAY'          -- for a healthy case statment ")
END AS DayName
, COUNT(orderid) AS OrdersCount
FROM Orders
GROUP BY 1, 2
ORDER BY 3 DESC  -- From high days of orders to lowest days of orders
```

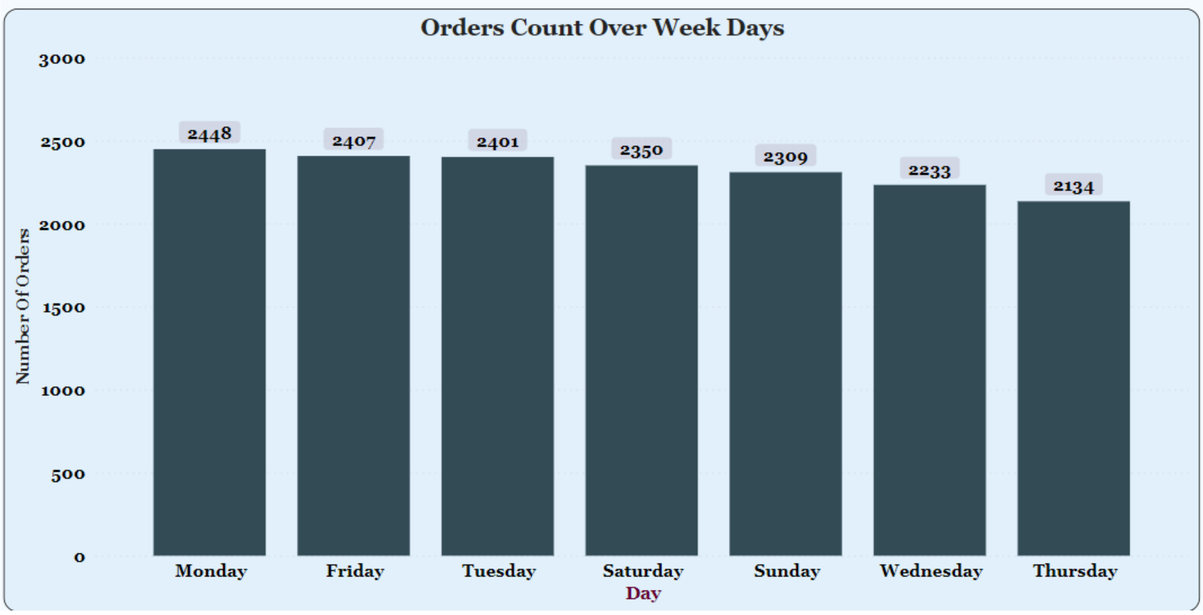
Query Explanation:

- ***Grouping** by the **weekday** which is extracted from the orderdate using the **STRFTIME()**, whereas for each day we're **counting** the number of orders purchased*
- *I used the **CASE** statement to map the day number to its **day name** for a proper result presentation.*

Query Result:

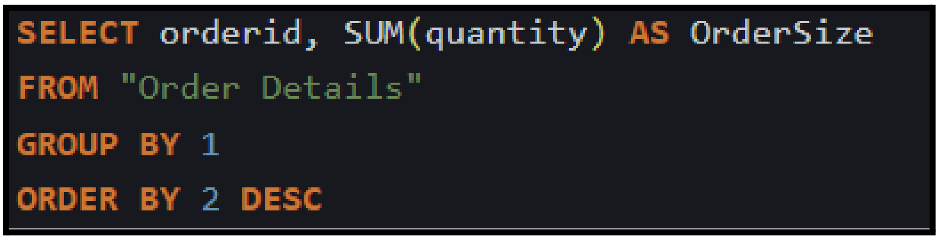


DayNumber	DayName	OrdersCount
1	Monday	2448
5	Friday	2407
2	Tuesday	2401
6	Saturday	2350
0	Sunday	2309
3	Wednesday	2233
4	Thursday	2134



Insight: Most of the orders are purchased on **Mondays.**

- **Order Size distribution Analysis:**

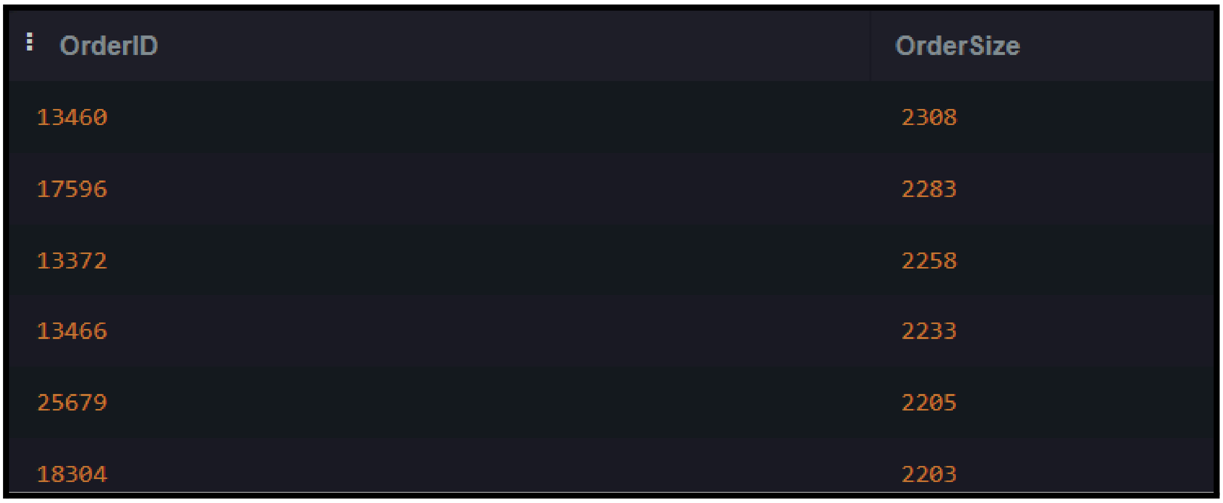


```sql
SELECT orderid, SUM(quantity) AS OrderSize
FROM "Order Details"
GROUP BY 1
ORDER BY 2 DESC
```

Query Explanation:

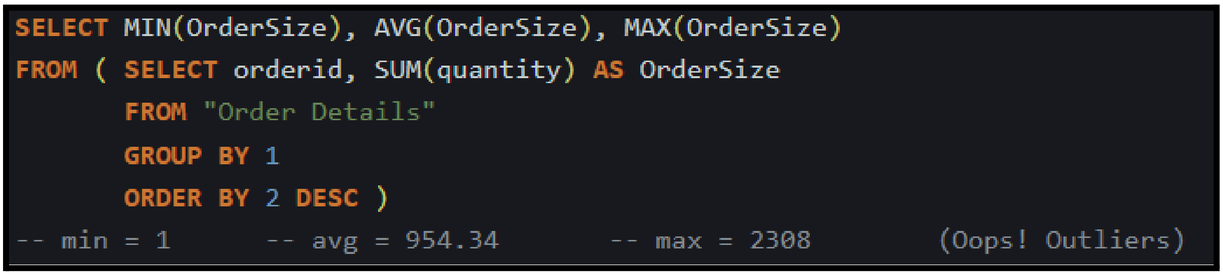
- *Easily done, we're **Grouping** by each order and getting the number of **items** purchased in each of them.*

Query Result:



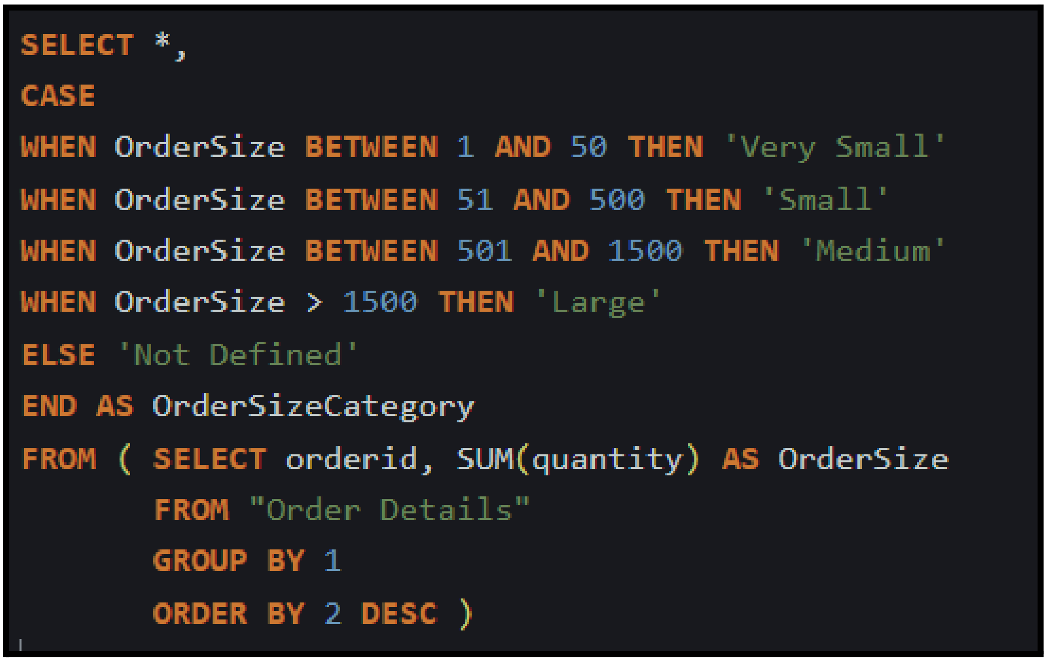
OrderID	OrderSize
13460	2308
17596	2283
13372	2258
13466	2233
25679	2205
18304	2203

- Since we have over **16K** orders, this result about the order size per order isn't so 'telling' or 'representive'!
- So, I thought of Categorizing the Orders by their size.
    - As usual, Let's check the dispersion of the Orders sizes as such:



```sql
SELECT MIN(OrderSize), AVG(OrderSize), MAX(OrderSize)
FROM ( SELECT orderid, SUM(quantity) AS OrderSize
       FROM "Order Details"
       GROUP BY 1
       ORDER BY 2 DESC )
-- min = 1      -- avg = 954.34      -- max = 2308         (Oops! Outliers)
```

- We have some **outliers** in our data, so I think of Categorizing into **4** categories which are **[small - medium - large]** and a category for the **{very small}** orders.

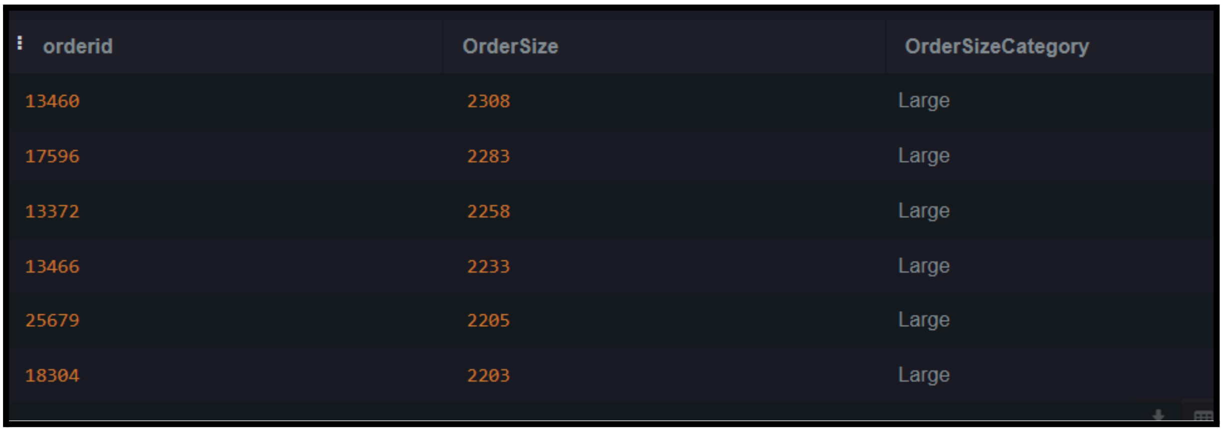


```
SELECT *,
CASE
WHEN OrderSize BETWEEN 1 AND 50 THEN 'Very Small'
WHEN OrderSize BETWEEN 51 AND 500 THEN 'Small'
WHEN OrderSize BETWEEN 501 AND 1500 THEN 'Medium'
WHEN OrderSize > 1500 THEN 'Large'
ELSE 'Not Defined'
END AS OrderSizeCategory
FROM ( SELECT orderid, SUM(quantity) AS OrderSize
       FROM "Order Details"
       GROUP BY 1
       ORDER BY 2 DESC )
```

Query Explanation:

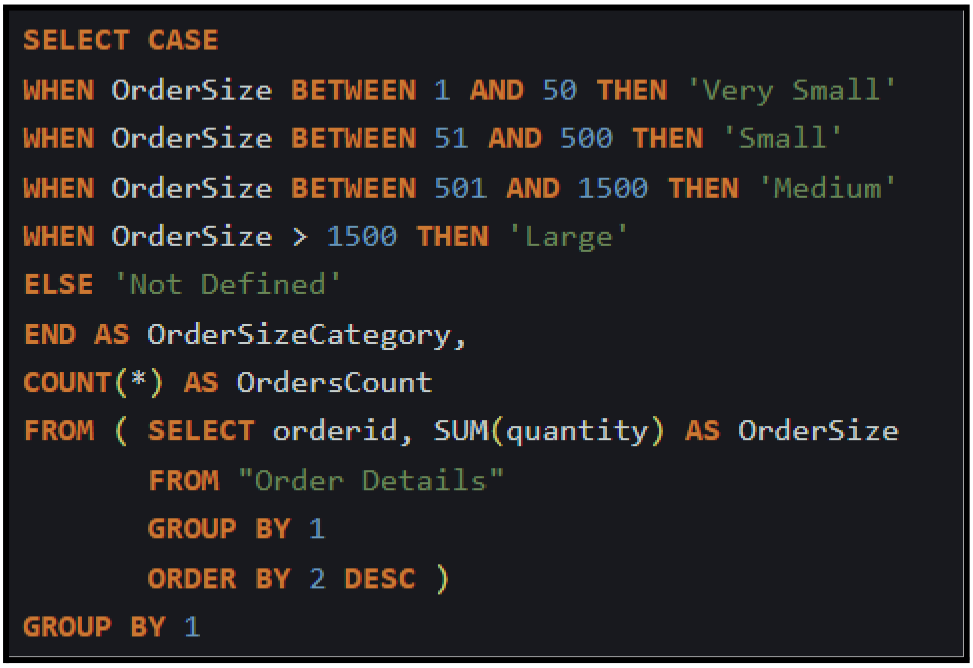
- *From the <u>previous query</u> calculating the **OrderSize**, I used the **CASE** statment on the **OrderSize** to define the Categories boundaries and <u>assign each order to a certain category</u> from the **4** possible categories.*

Query Result:



orderid	OrderSize	OrderSizeCategory
13460	2308	Large
17596	2283	Large
13372	2258	Large
13466	2233	Large
25679	2205	Large
18304	2203	Large

- ***Now, Let's build on that and count the orders in each Category***

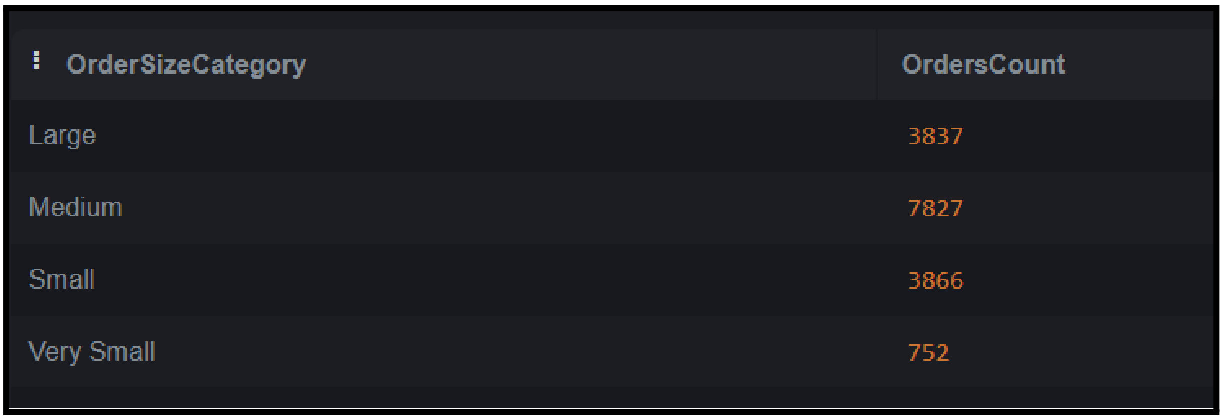


```
SELECT CASE
WHEN OrderSize BETWEEN 1 AND 50 THEN 'Very Small'
WHEN OrderSize BETWEEN 51 AND 500 THEN 'Small'
WHEN OrderSize BETWEEN 501 AND 1500 THEN 'Medium'
WHEN OrderSize > 1500 THEN 'Large'
ELSE 'Not Defined'
END AS OrderSizeCategory,
COUNT(*) AS OrdersCount
FROM ( SELECT orderid, SUM(quantity) AS OrderSize
       FROM "Order Details"
       GROUP BY 1
       ORDER BY 2 DESC )
GROUP BY 1
```

Query Explanation:

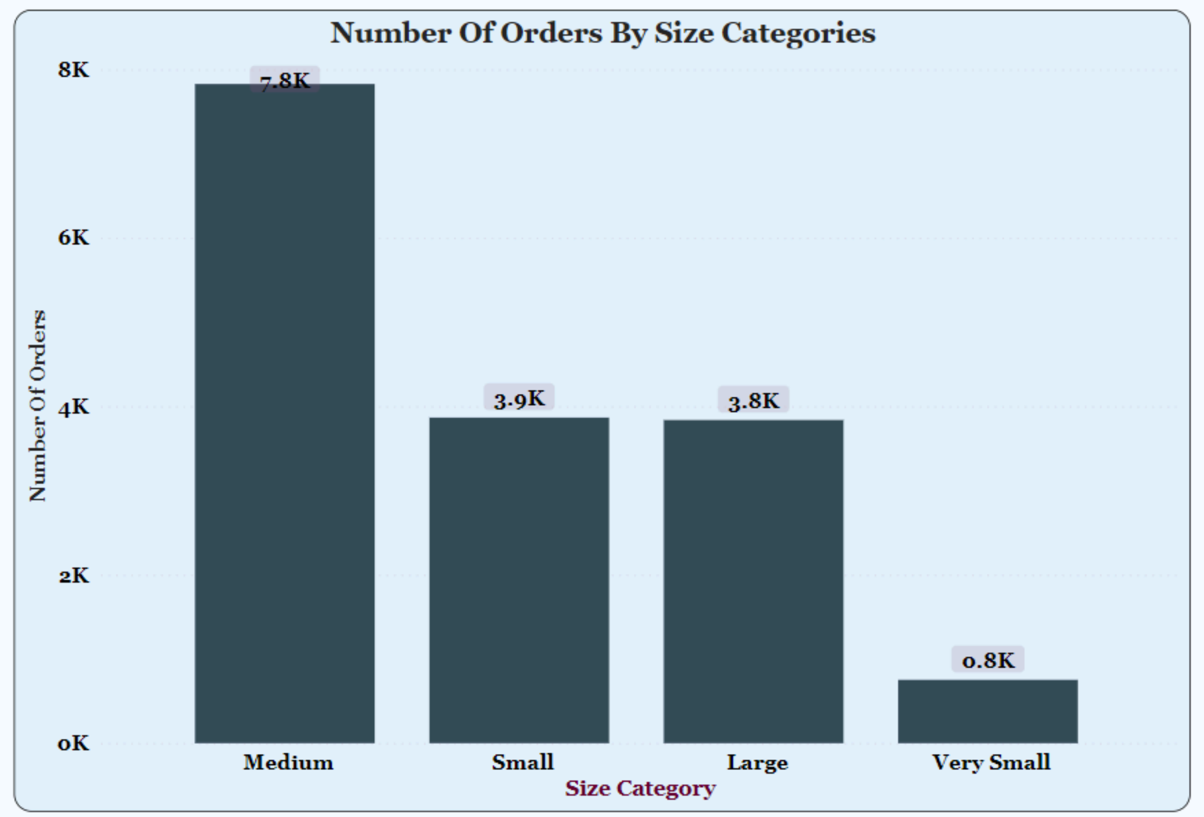
- *Here, I just **Grouped** by the **OrderSizeCategory** and **COUNTed** the orders in each, to see the diversity.*

Query Result:



OrderSizeCategory	OrdersCount
Large	3837
Medium	7827
Small	3866
Very Small	752

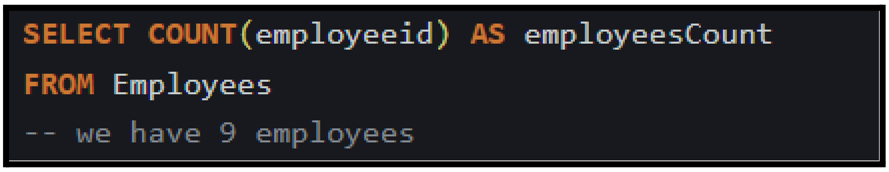
Query Result: *(Bar Chart)*



Insight:

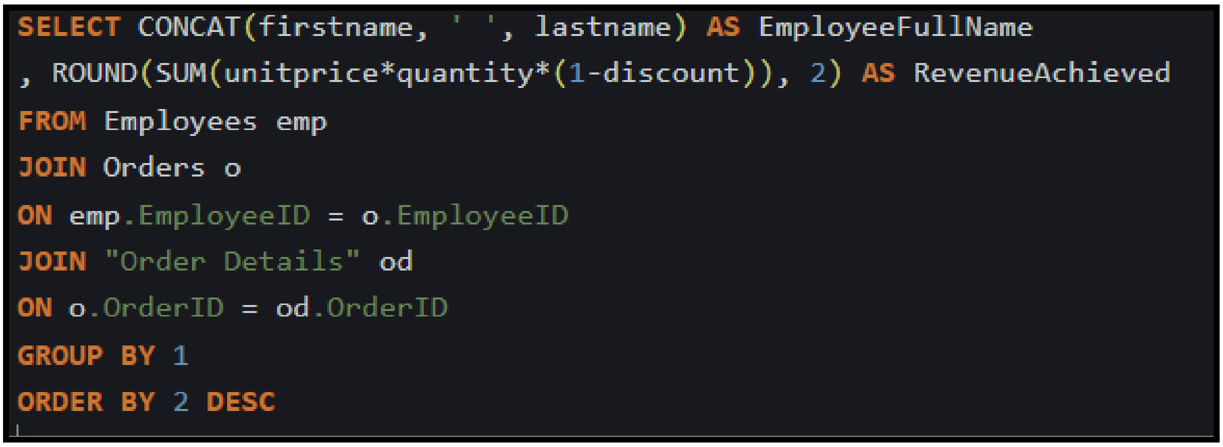
- About **50%** of the Orders are of **Medium Size.**
- **5%** of the orders are **“Very Small”** Orders.

- **Employee Performance Analysis:**
  - **Some Warm-up Employee Analysis:**
    - **At first, how many employees work for my company?**



```sql
SELECT COUNT(employeeid) AS employeesCount
FROM Employees
-- we have 9 employees
```

  - **Analysis Of Employees by [Revenue Achieved]:**

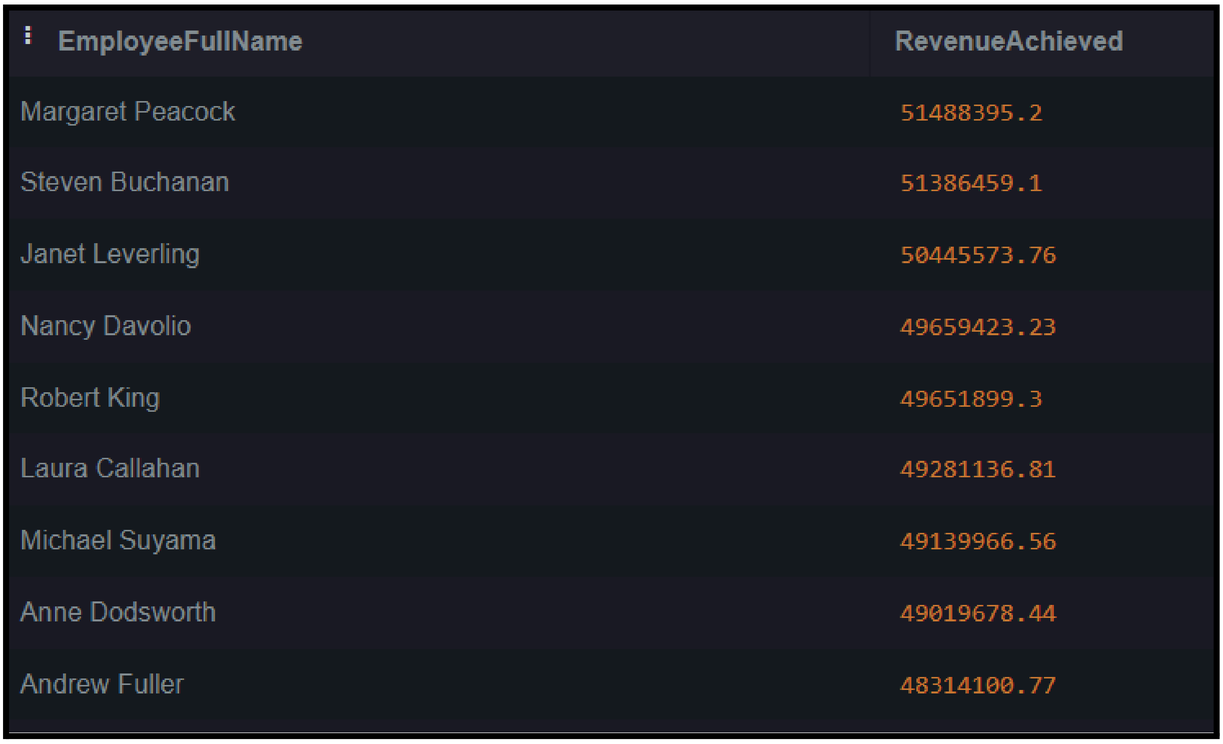


```sql
SELECT CONCAT(firstname, ' ', lastname) AS EmployeeFullName
, ROUND(SUM(unitprice*quantity*(1-discount)), 2) AS RevenueAchieved
FROM Employees emp
JOIN Orders o
ON emp.EmployeeID = o.EmployeeID
JOIN "Order Details" od
ON o.OrderID = od.OrderID
GROUP BY 1
ORDER BY 2 DESC
```

Query Explanation:

- *I **JOIN**ed between the **[Employees]** table and the **[“Orders”]** table and the **[“Order Details”]** table to get the **SUM()** of money obtained by **each employee.***
- *So, we **GROUP**ed by the employee name that is the **CONCAT** between **{fname}** and **{Lname}** of the employee.*

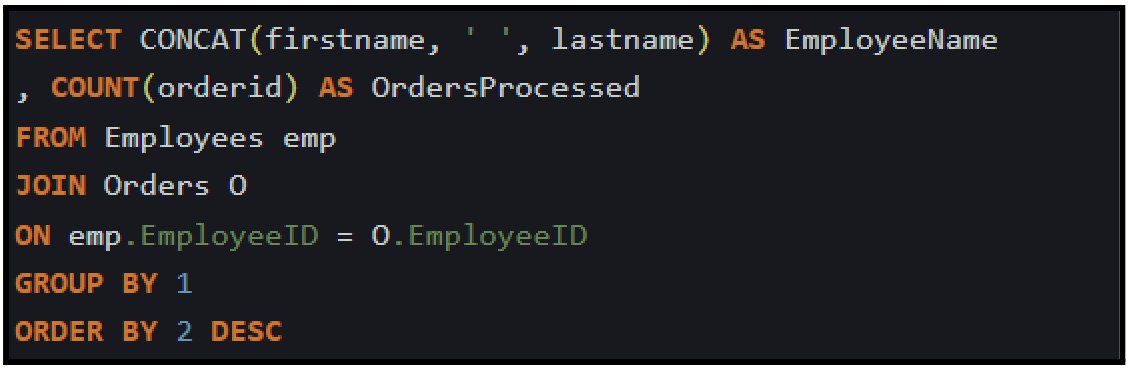
Query Result:



EmployeeFullName	RevenueAchieved
Margaret Peacock	51488395.2
Steven Buchanan	51386459.1
Janet Leverling	50445573.76
Nancy Davolio	49659423.23
Robert King	49651899.3
Laura Callahan	49281136.81
Michael Suyama	49139966.56
Anne Dodsworth	49019678.44
Andrew Fuller	48314100.77

Insight: Margaret Peacock is the **TOP** Employee bringing **money** to our company.

- **Analysis Of Employees by [Number of Orders Processed]:**

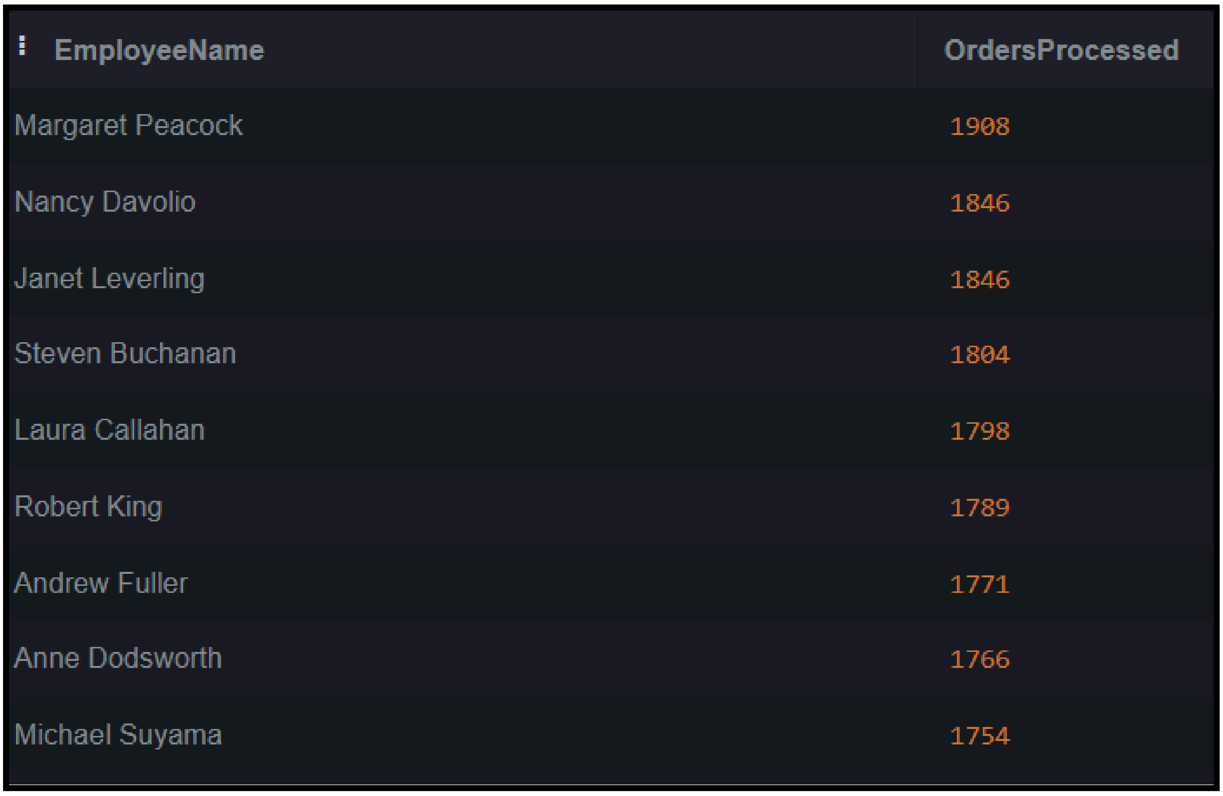


```
SELECT CONCAT(firstname, ' ', lastname) AS EmployeeName
, COUNT(orderid) AS OrdersProcessed
FROM Employees emp
JOIN Orders O
ON emp.EmployeeID = O.EmployeeID
GROUP BY 1
ORDER BY 2 DESC
```

Query Explanation:

- *I **JOIN**ed between the **[Employees]** table and the **[“Orders”]** table to COUNT the number of orders processed by **each employee**.*

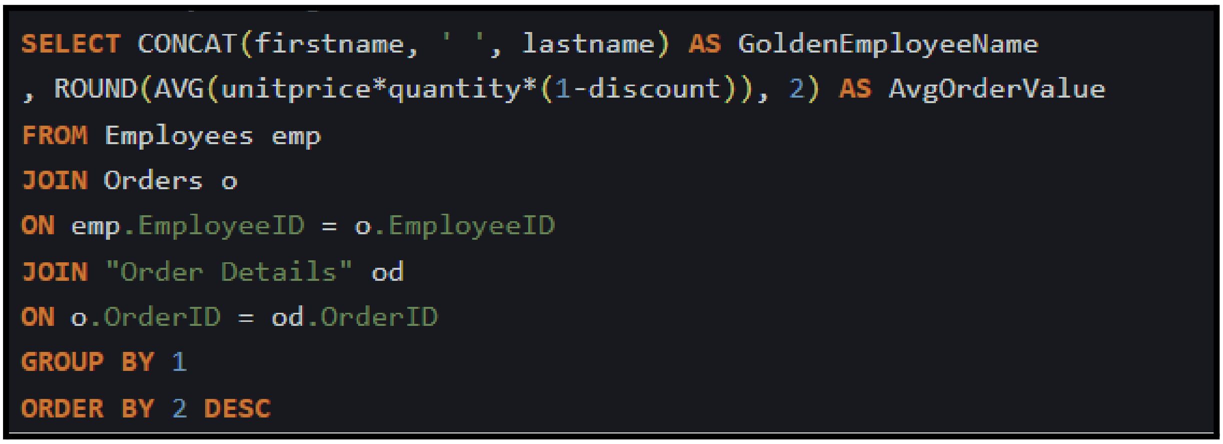
Query Result:



EmployeeName	OrdersProcessed
Margaret Peacock	1908
Nancy Davolio	1846
Janet Leverling	1846
Steven Buchanan	1804
Laura Callahan	1798
Robert King	1789
Andrew Fuller	1771
Anne Dodsworth	1766
Michael Suyama	1754

Insight: Margaret Peacock is again the **TOP** Employee closing **deals** to our company. |PS: (Yayy!)

- ### Analysis Of Employees by [Average Order Value]:



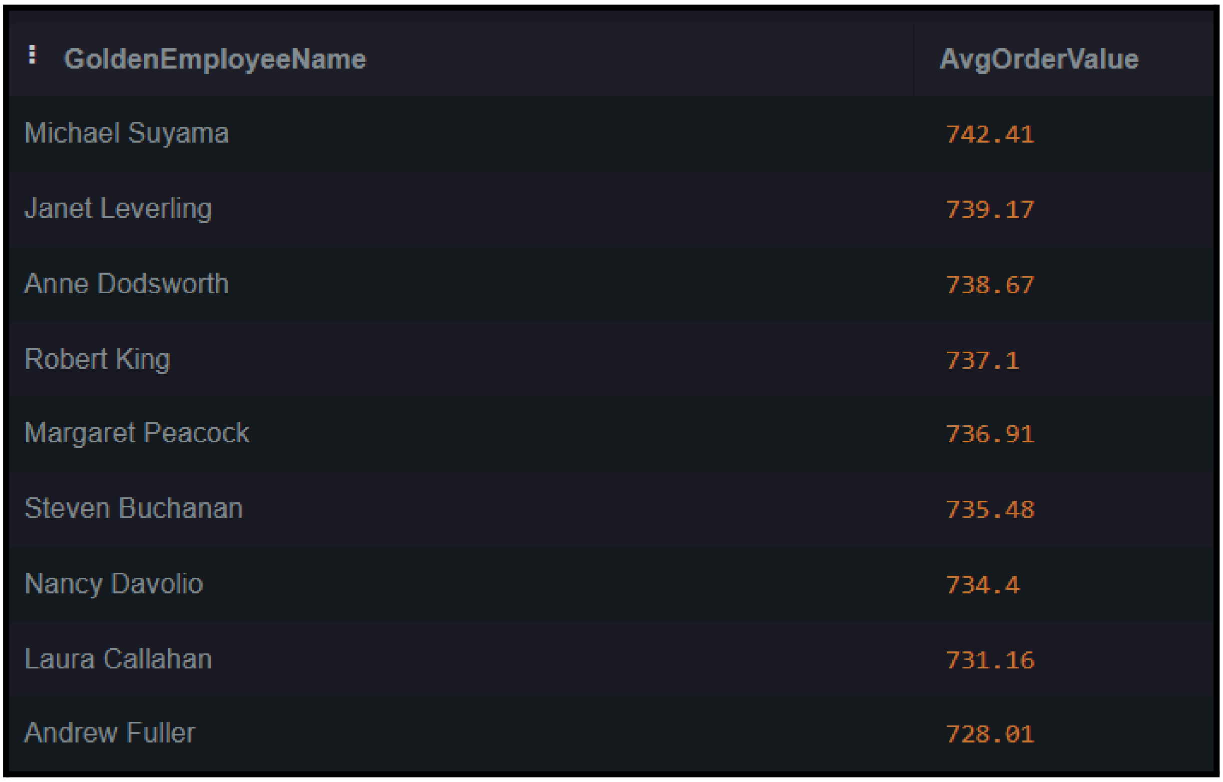
```
SELECT CONCAT(firstname, ' ', lastname) AS GoldenEmployeeName
, ROUND(AVG(unitprice*quantity*(1-discount)), 2) AS AvgOrderValue
FROM Employees emp
JOIN Orders o
ON emp.EmployeeID = o.EmployeeID
JOIN "Order Details" od
ON o.OrderID = od.OrderID
GROUP BY 1
ORDER BY 2 DESC
```

Query Explanation:

- *I **JOIN**ed between the **[Employees]** table and the **[“Orders”]** table and the **[“Order Details”]** table to get the **AVG** of OrderValue achieved by **each employee.***

- *we **GROUP**ed by the employee name that is the **CONCAT** between **{fname}** and **{Lname}** of the employee.*

Query Result:



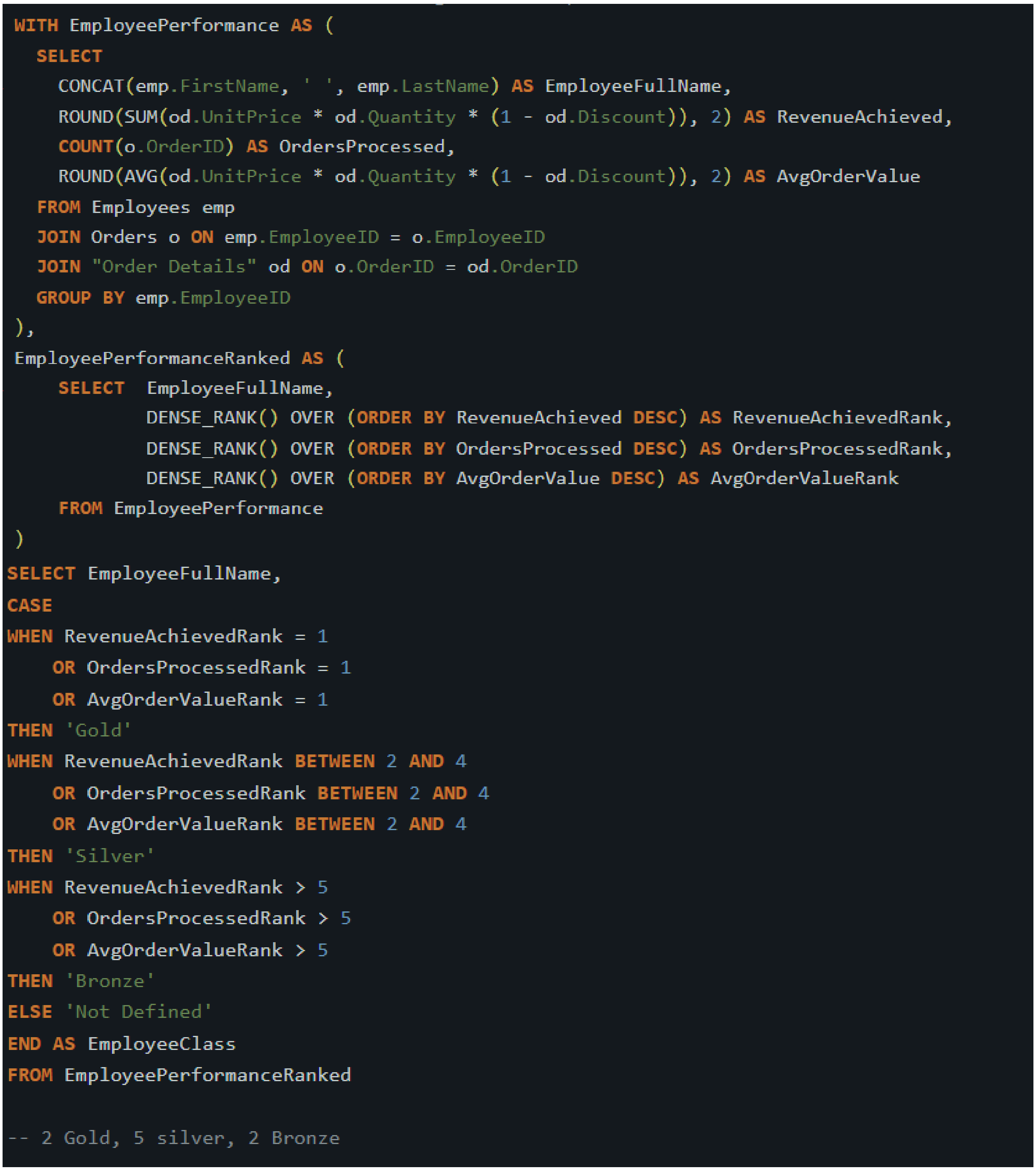
GoldenEmployeeName	AvgOrderValue
Michael Suyama	742.41
Janet Leverling	739.17
Anne Dodsworth	738.67
Robert King	737.1
Margaret Peacock	736.91
Steven Buchanan	735.48
Nancy Davolio	734.4
Laura Callahan	731.16
Andrew Fuller	728.01

Insight:

- **Micheal Suyama** makes the **highest** Average Order Value.
- **Margaret Peacock**, who was the top revenue achiever and the top deals breaker, makes an **average** Average Order Value
  - *This indicates that his high revenue achievments depend mainly on the high number of orders processed.*

- I think of Categorizing my employees into ***[Gold, Silver, Bronze]*** employees based on the **performance metrics** calculated.

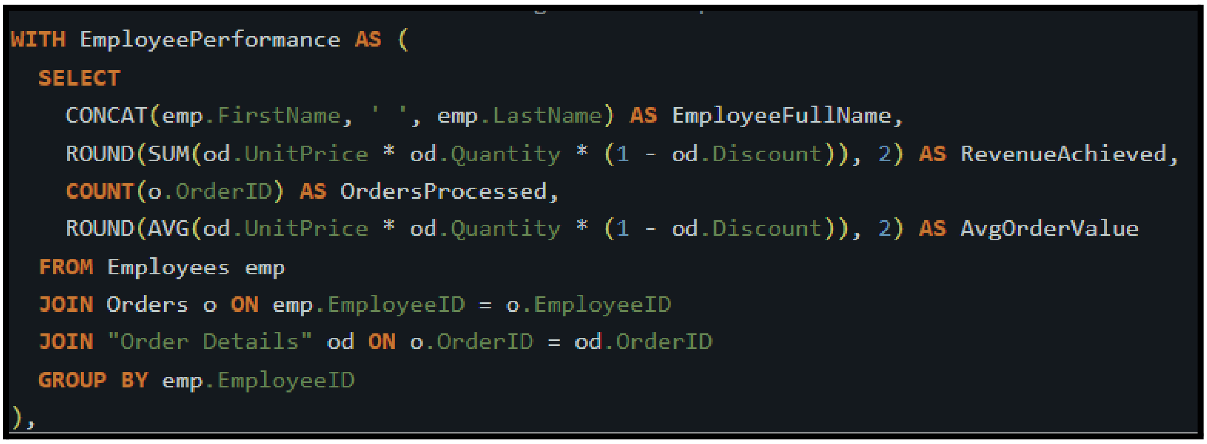
– In this query I used the **CTE** to help me build the logic of the query I needed



```sql
WITH EmployeePerformance AS (
  SELECT
    CONCAT(emp.FirstName, ' ', emp.LastName) AS EmployeeFullName,
    ROUND(SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)), 2) AS RevenueAchieved,
    COUNT(o.OrderID) AS OrdersProcessed,
    ROUND(AVG(od.UnitPrice * od.Quantity * (1 - od.Discount)), 2) AS AvgOrderValue
  FROM Employees emp
  JOIN Orders o ON emp.EmployeeID = o.EmployeeID
  JOIN "Order Details" od ON o.OrderID = od.OrderID
  GROUP BY emp.EmployeeID
),
EmployeePerformanceRanked AS (
    SELECT  EmployeeFullName,
            DENSE_RANK() OVER (ORDER BY RevenueAchieved DESC) AS RevenueAchievedRank,
            DENSE_RANK() OVER (ORDER BY OrdersProcessed DESC) AS OrdersProcessedRank,
            DENSE_RANK() OVER (ORDER BY AvgOrderValue DESC) AS AvgOrderValueRank
    FROM EmployeePerformance
)
SELECT EmployeeFullName,
CASE
WHEN RevenueAchievedRank = 1
    OR OrdersProcessedRank = 1
    OR AvgOrderValueRank = 1
THEN 'Gold'
WHEN RevenueAchievedRank BETWEEN 2 AND 4
    OR OrdersProcessedRank BETWEEN 2 AND 4
    OR AvgOrderValueRank BETWEEN 2 AND 4
THEN 'Silver'
WHEN RevenueAchievedRank > 5
    OR OrdersProcessedRank > 5
    OR AvgOrderValueRank > 5
THEN 'Bronze'
ELSE 'Not Defined'
END AS EmployeeClass
FROM EmployeePerformanceRanked

-- 2 Gold, 5 silver, 2 Bronze
```

– Let’s Break it down:

– At first, I created this helping EmployeePerformance CTE

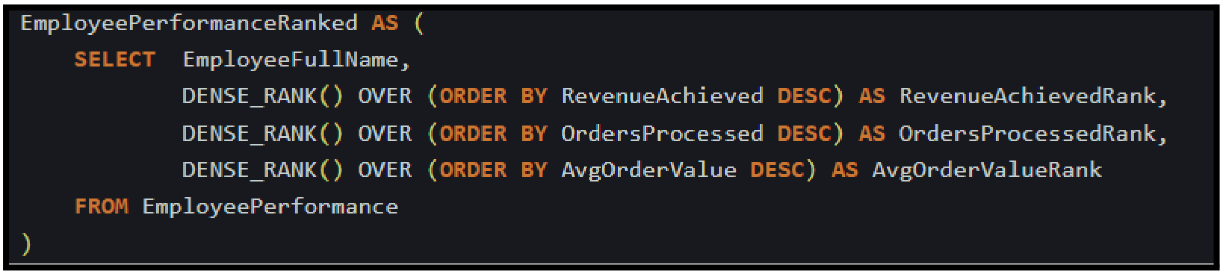


```
WITH EmployeePerformance AS (
  SELECT
    CONCAT(emp.FirstName, ' ', emp.LastName) AS EmployeeFullName,
    ROUND(SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)), 2) AS RevenueAchieved,
    COUNT(o.OrderID) AS OrdersProcessed,
    ROUND(AVG(od.UnitPrice * od.Quantity * (1 - od.Discount)), 2) AS AvgOrderValue
  FROM Employees emp
  JOIN Orders o ON emp.EmployeeID = o.EmployeeID
  JOIN "Order Details" od ON o.OrderID = od.OrderID
  GROUP BY emp.EmployeeID
),
```

which:

- Combines data from ***[Employees]*** and ***[Order Details]*** tables to calculate the performance metrics for each employee:
    - Revenue Achieved: Total revenue generated by an employee.
    - Orders Processed: Total number of orders processed by the employee.
    - AvgOrderValue: Average revenue per order.
- **Group**ed data by ***employee_id*** to get aggregated metrics for each employee

– Then, Using the past table output, I ranked employees by performance:

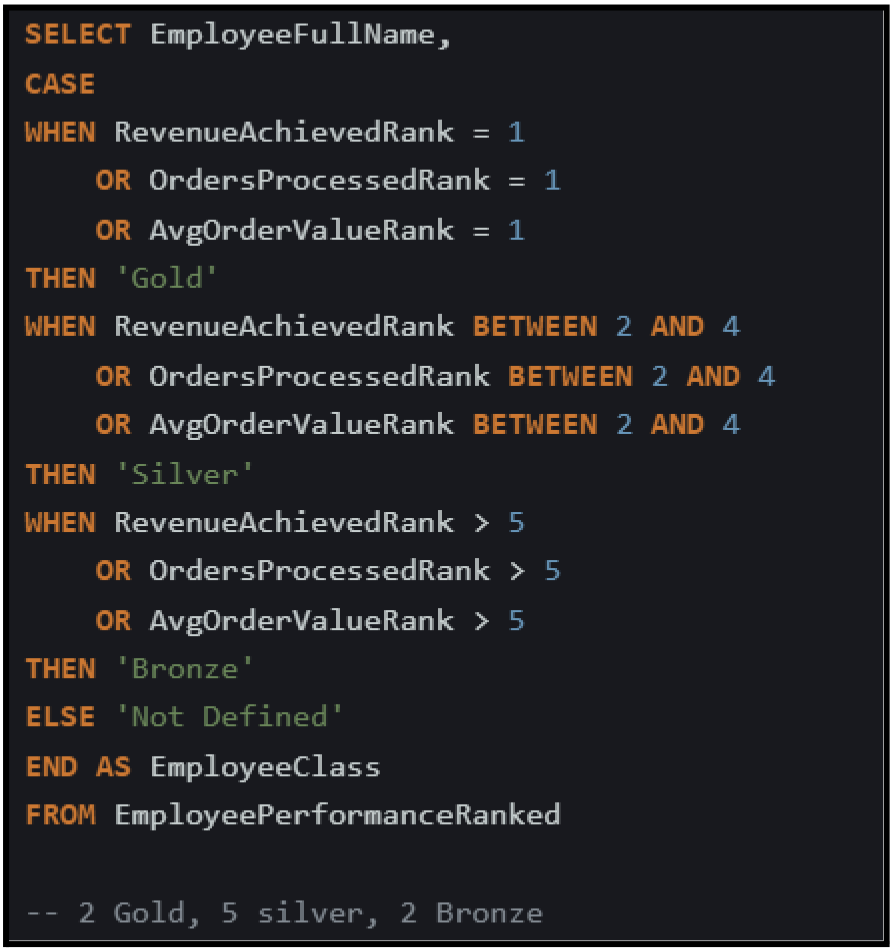


```
EmployeePerformanceRanked AS (
    SELECT  EmployeeFullName,
            DENSE_RANK() OVER (ORDER BY RevenueAchieved DESC) AS RevenueAchievedRank,
            DENSE_RANK() OVER (ORDER BY OrdersProcessed DESC) AS OrdersProcessedRank,
            DENSE_RANK() OVER (ORDER BY AvgOrderValue DESC) AS AvgOrderValueRank
    FROM EmployeePerformance
)
```

This ranking is done by the built-in **DENSE_RANK()** window function that gives the highest employee in revenue a rank of **1** then the next highest a rank of **2** and so on,

*“I used this function in particular because **it ensures no gaps** in ranks, I mean if two employees tie for rank 1, the next rank will be 2 for the second highest employee after those employees tied in rank 1”*

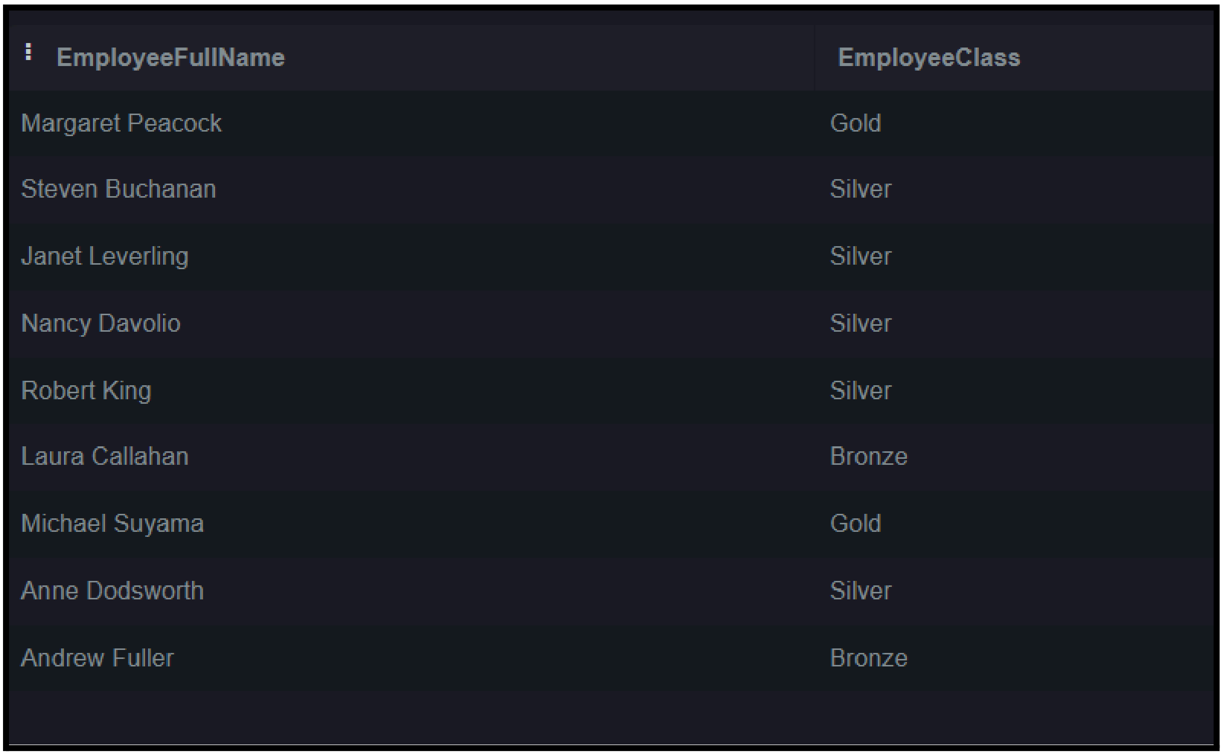
– Finally, I categorized employees into performance levels



```
SELECT EmployeeFullName,
CASE
WHEN RevenueAchievedRank = 1
    OR OrdersProcessedRank = 1
    OR AvgOrderValueRank = 1
THEN 'Gold'
WHEN RevenueAchievedRank BETWEEN 2 AND 4
    OR OrdersProcessedRank BETWEEN 2 AND 4
    OR AvgOrderValueRank BETWEEN 2 AND 4
THEN 'Silver'
WHEN RevenueAchievedRank > 5
    OR OrdersProcessedRank > 5
    OR AvgOrderValueRank > 5
THEN 'Bronze'
ELSE 'Not Defined'
END AS EmployeeClass
FROM EmployeePerformanceRanked

-- 2 Gold, 5 silver, 2 Bronze
```

- Categorizeing employees based on their ranks:
    - **Gold**: At least one metric rank is **1** (top performance).
    - **Silver**: At least one metric rank is **2 or 3** (high performance).
    - **Bronze**: All ranks are **4 or lower** (moderate to low performance).

Final Output:



EmployeeFullName	EmployeeClass
Margaret Peacock	Gold
Steven Buchanan	Silver
Janet Leverling	Silver
Nancy Davolio	Silver
Robert King	Silver
Laura Callahan	Bronze
Michael Suyama	Gold
Anne Dodsworth	Silver
Andrew Fuller	Bronze

Insight:

- **2 Gold employees**
- **5 Silver employees**
- **2 Bronze employees**

- **Northwind Database Analysis Recap**

  The Northwind database project involved analyzing various aspects of a retail business to provide actionable insights for improving operations, customer engagement, and overall performance.

- **Key Business Recommendations:**
  - Reward and retain Gold employees while training others to improve.
  - Convert "Potential Loyalists" into "Champions" with personalized offers.
  - Focus inventory and marketing efforts on top-ranked products.
  - Expand operations in regions with high growth potential.