

Beginner Queries.

➤ Define meta data in mysql workbench or any other SQL tool

Metadata in MySQL Workbench or any other SQL tool refers to data about the data. It's essentially information that describes the structure and organization of your database, rather than the actual content of the data itself.

Key components of metadata in MySQL Workbench include:

- Database Structure:
 - **Table names**
 - **Column names**
 - **Data types of columns**
 - **Constraints (e.g., primary keys, foreign keys, unique constraints)**
 - **Indexes**
- Relationships:
 - **How tables are connected (e.g., foreign key relationships)**
- Permissions:
 - **User privileges and access rights**
- Database Objects:
 - **Views, stored procedures, functions, triggers**

Accessing Metadata in MySQL Workbench:

- **Schema Browser: This is the primary tool for viewing and managing metadata. You can see table definitions, create new objects, and modify existing ones.**
- **Database Explorer: This provides a hierarchical view of your database objects, including tables, views, stored procedures, and more.**
- **INFORMATION_SCHEMA: This is a system database that contains metadata about the database itself. You can query its tables to retrieve information about objects, columns, constraints, and more.**

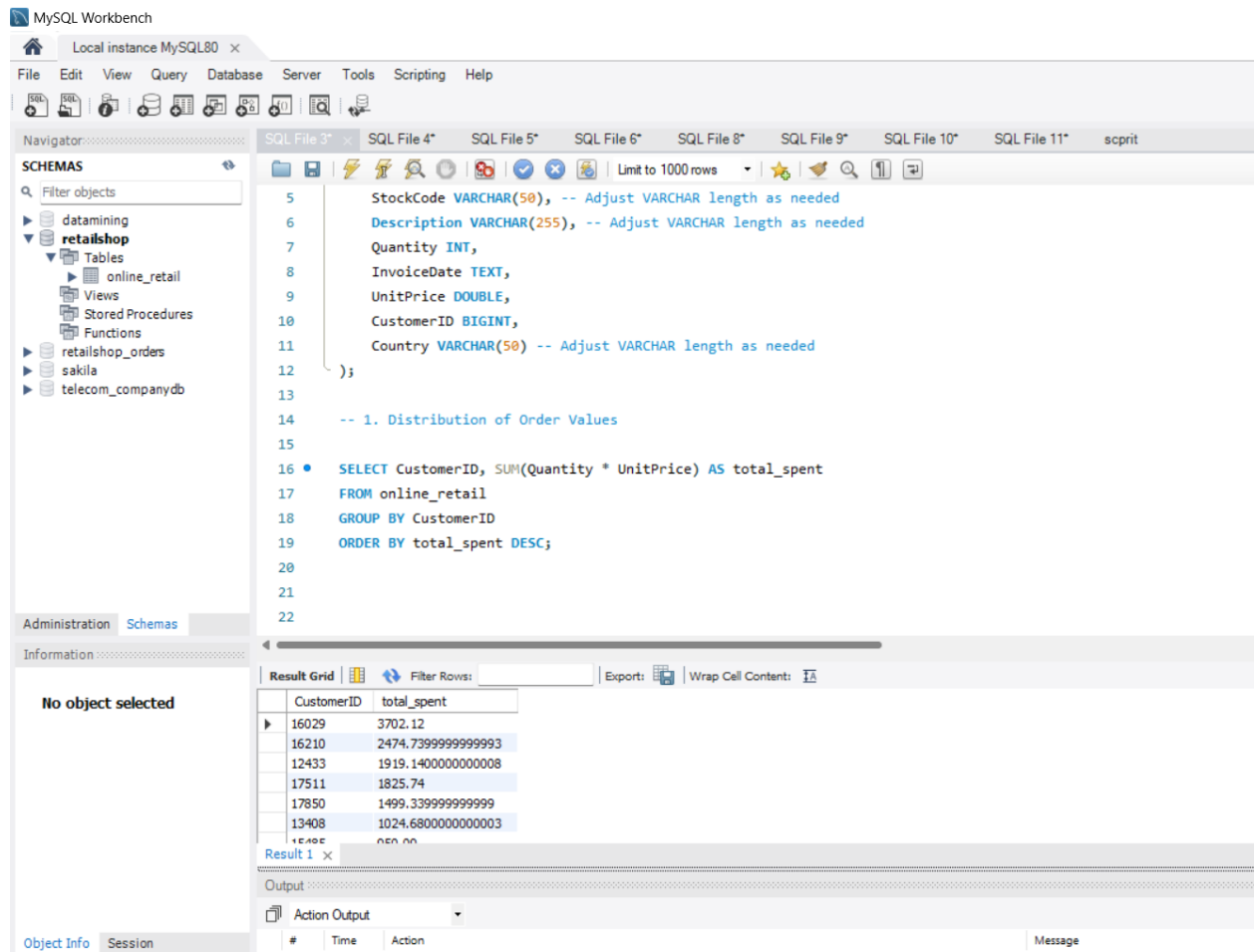
Example:

To view the structure of a table named customers, you can use the following SQL query:

SQL

DESCRIBE customers;

➤ **What is the distribution of order values across all customers in the dataset?**



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'retailshop' expanded, showing 'online_retail' and 'online_retail' tables. The main editor shows a SQL query in 'SQL File 3*' that defines a table structure and then executes a query to find the distribution of order values. The query is as follows:

```
5 StockCode VARCHAR(50), -- Adjust VARCHAR length as needed
6 Description VARCHAR(255), -- Adjust VARCHAR length as needed
7 Quantity INT,
8 InvoiceDate TEXT,
9 UnitPrice DOUBLE,
10 CustomerID BIGINT,
11 Country VARCHAR(50) -- Adjust VARCHAR length as needed
12 );
13
14 -- 1. Distribution of Order Values
15
16 • SELECT CustomerID, SUM(Quantity * UnitPrice) AS total_spent
17 FROM online_retail
18 GROUP BY CustomerID
19 ORDER BY total_spent DESC;
20
21
22
```

The 'Result Grid' at the bottom shows the results of the query, with columns 'CustomerID' and 'total_spent'. The results are sorted in descending order of total spent.

| CustomerID | total_spent |
|------------|---------------------|
| 16029 | 3702.12 |
| 16210 | 2474.7399999999993 |
| 12433 | 1919.14000000000008 |
| 17511 | 1825.74 |
| 17850 | 1499.3399999999999 |
| 13408 | 1024.68000000000003 |
| 16406 | 1024.68000000000003 |

Understanding Order Value Distribution

Order value distribution refers to how the total values of orders are spread out across different customers in a dataset. This can provide insights into customer spending patterns, identifying high-value customers, and understanding overall sales trends.

Calculating Order Value Distribution:

To calculate the distribution of order values across all customers in your dataset, you can use the following SQL query:

```
SELECT CustomerID, SUM(Quantity * UnitPrice) AS total_spent
FROM online_retail
GROUP BY CustomerID
```

ORDER BY total_spent DESC;

This query will:

1. **Group orders by customer ID:** This aggregates all orders for each unique customer.
2. **Calculate total spending:** For each customer, it calculates the sum of the product of Quantity and UnitPrice for all their orders, giving you the total amount they spent.
3. **Order by total spending:** The results are sorted in descending order based on the total_spent column, showing the customers who spent the most first.

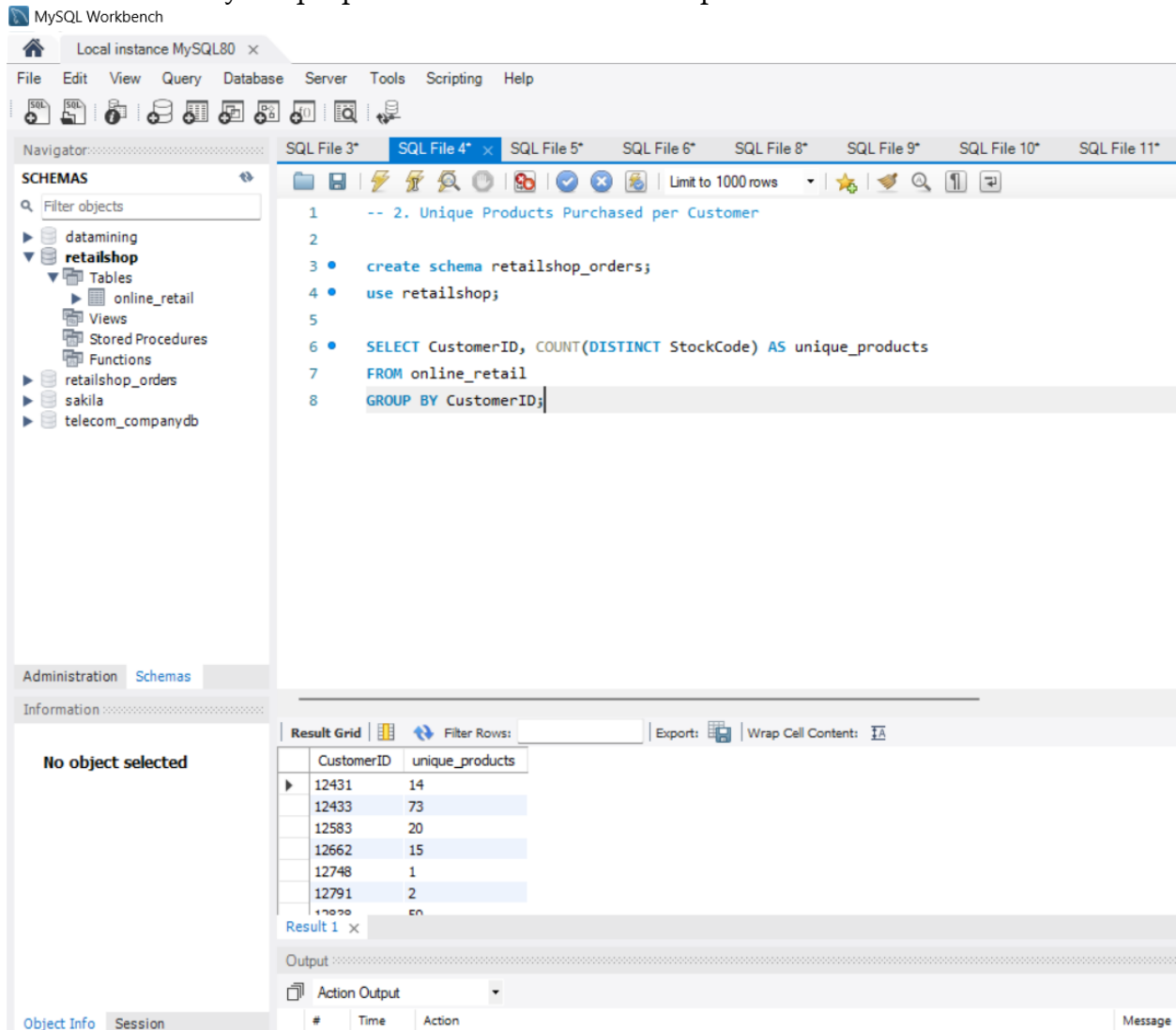
Analyzing the Results:

Once you have the results, you can analyze the distribution of order values using various methods:

- **Visualization:** Create a histogram or bar chart to visualize the distribution. This will help you identify patterns, such as whether the distribution is skewed, normal, or uniform.
- **Summary Statistics:** Calculate summary statistics like the mean, median, mode, standard deviation, and quartiles to get a quantitative understanding of the distribution.
- **Segmentation:** Divide customers into segments based on their total spending (e.g., high-value, mid-value, low-value) to identify customer segments with different spending behaviors.

By analyzing the distribution of order values, you can gain valuable insights into your customers' purchasing habits and identify potential areas for improvement or targeting.

➤ How many unique products has each customer purchased?



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'retailshop' selected. The main editor shows a SQL query in 'SQL File 4'. The query is as follows:

```
1  -- 2. Unique Products Purchased per Customer
2
3  • create schema retailshop_orders;
4  • use retailshop;
5
6  • SELECT CustomerID, COUNT(DISTINCT StockCode) AS unique_products
7  FROM online_retail
8  GROUP BY CustomerID;
```

The 'Result Grid' at the bottom shows the output of the query:

| CustomerID | unique_products |
|------------|-----------------|
| 12431 | 14 |
| 12433 | 73 |
| 12583 | 20 |
| 12662 | 15 |
| 12748 | 1 |
| 12791 | 2 |
| 12800 | 60 |

To determine how many unique products each customer has purchased, you can use the following SQL query:

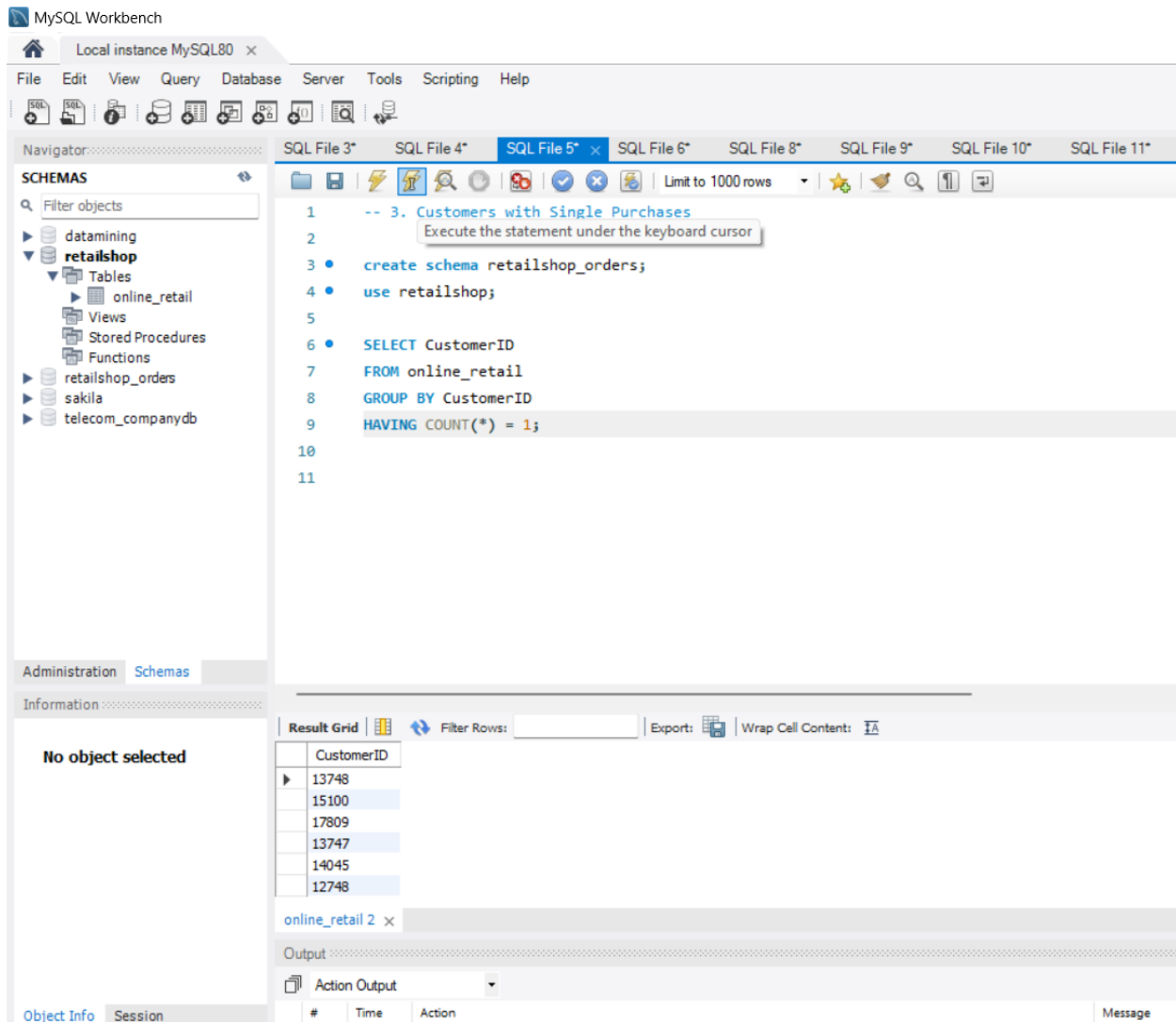
```
SELECT CustomerID, COUNT(DISTINCT StockCode) AS unique_products
FROM online_retail
GROUP BY CustomerID;
```

This query will:

1. **Group orders by customer ID:** This aggregates all orders for each unique customer.
2. **Count unique products:** For each customer, it counts the number of distinct StockCode values, which represent unique products.

The result will be a list of customers and the corresponding number of unique products they have purchased.

- Which customers have only made a single purchase from the company?



To identify customers who have made only a single purchase from the company, you can use the following SQL query:

```
SELECT CustomerID
FROM online_retail
GROUP BY CustomerID
HAVING COUNT(*) = 1;
```

This query will:

1. **Group orders by customer ID:** This aggregates all orders for each unique customer.

2. **Filter for single-purchase customers:** The HAVING clause filters the results to only include customers who have a count of 1, indicating they have made only one purchase.

The result will be a list of customer IDs who have made a single purchase.

- Which products are most commonly purchased together by customers in the dataset?

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with 'retailshop' selected. The main editor displays a SQL query titled '-- 4. Commonly Purchased Products Together'. The query is as follows:

```
-- 4. Commonly Purchased Products Together
use retailshop;
SELECT StockCode_1, StockCode_2, COUNT(*) AS frequency
FROM (
  SELECT
    o1.StockCode AS StockCode_1,
    o2.StockCode AS StockCode_2
  FROM online_retail o1
  JOIN online_retail o2 ON o1.InvoiceNo <> o2.InvoiceNo AND o1.CustomerID = o2.CustomerID
) AS paired_products
GROUP BY StockCode_1, StockCode_2
ORDER BY frequency DESC;
```

Below the query editor, the 'Result Grid' shows the results of the query. The table has three columns: 'StockCode_1', 'StockCode_2', and 'frequency'. The results are as follows:

| StockCode_1 | StockCode_2 | frequency |
|-------------|-------------|-----------|
| 22633 | 85123A | 25 |
| 22632 | 85123A | 25 |
| 22633 | 71053 | 25 |
| 22632 | 71053 | 25 |
| 22633 | 84406B | 25 |
| 22632 | 84406B | 25 |

To identify the products that are most commonly purchased together by customers in the dataset, you can use the following SQL query:

```

SELECT StockCode_1, StockCode_2, COUNT(*) AS frequency
FROM (
  SELECT
    o1.StockCode AS StockCode_1,
    o2.StockCode AS StockCode_2
  FROM online_retail o1
  JOIN online_retail o2 ON o1.InvoiceNo <> o2.InvoiceNo AND o1.CustomerID =
o2.CustomerID
) AS paired_products
GROUP BY StockCode_1, StockCode_2
ORDER BY frequency DESC;

```

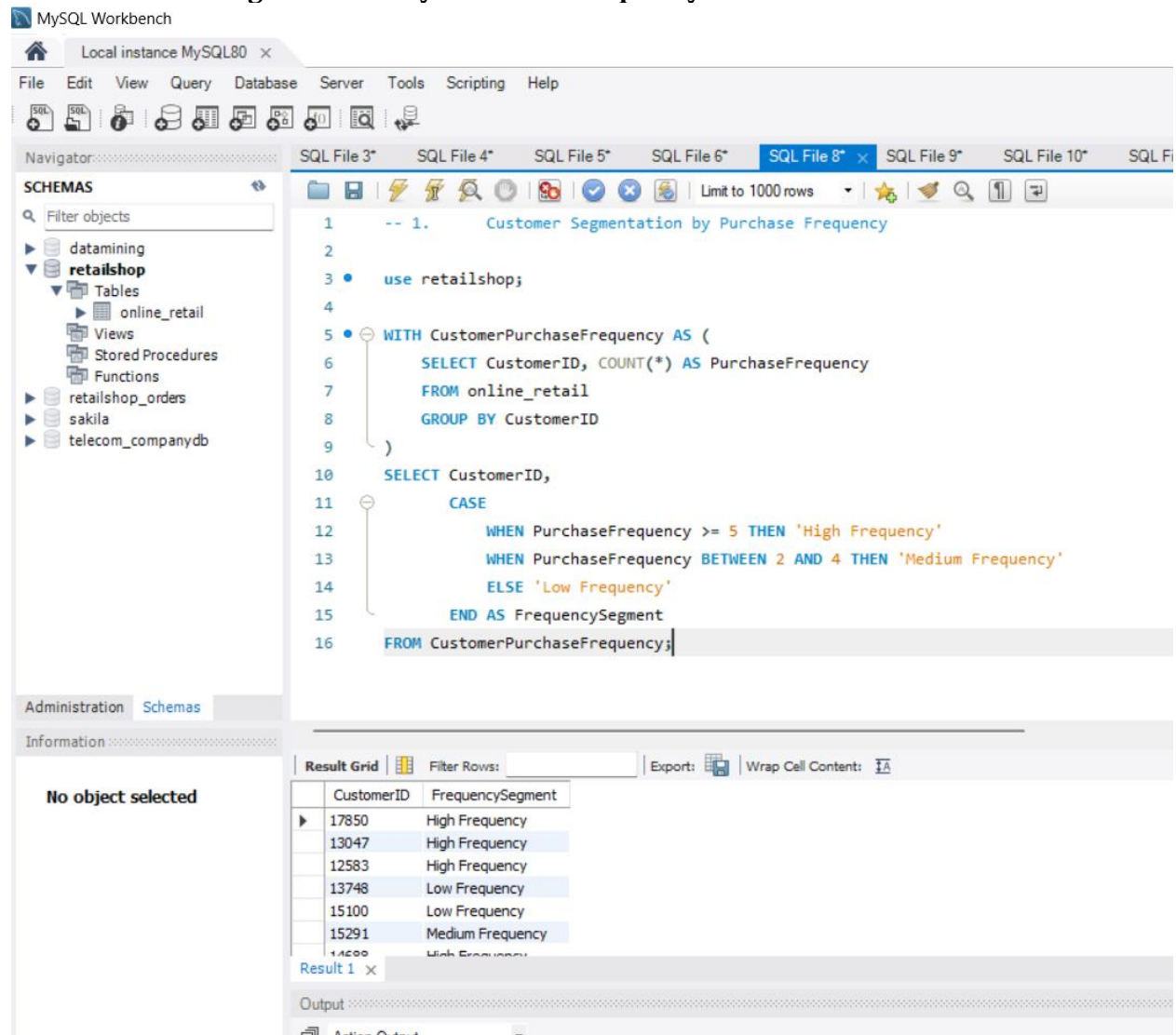
This query will:

1. **Create a temporary table `paired_products`:** This table will contain pairs of products purchased by the same customer in the same order, excluding pairs of the same product.
2. **Group by product pairs:** The GROUP BY clause groups the results by the pairs of products.
3. **Count occurrences:** The COUNT(*) function counts the number of times each product pair appears in the dataset.
4. **Order by frequency:** The results are ordered by the frequency column in descending order, so the most commonly purchased pairs appear first.

The output will show pairs of products and the frequency with which they were purchased together. You can use this information to identify product bundles or cross-selling opportunities.

Advance Queries

1. Customer Segmentation by Purchase Frequency



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' panel with a tree view of databases including 'retailshop'. The main editor window shows a SQL query titled '1. Customer Segmentation by Purchase Frequency'. The query is as follows:

```
-- 1. Customer Segmentation by Purchase Frequency
2
3 use retailshop;
4
5 WITH CustomerPurchaseFrequency AS (
6     SELECT CustomerID, COUNT(*) AS PurchaseFrequency
7     FROM online_retail
8     GROUP BY CustomerID
9 )
10 SELECT CustomerID,
11     CASE
12         WHEN PurchaseFrequency >= 5 THEN 'High Frequency'
13         WHEN PurchaseFrequency BETWEEN 2 AND 4 THEN 'Medium Frequency'
14         ELSE 'Low Frequency'
15     END AS FrequencySegment
16 FROM CustomerPurchaseFrequency;
```

The bottom panel shows the 'Result Grid' with the following data:

| CustomerID | FrequencySegment |
|------------|------------------|
| 17850 | High Frequency |
| 13047 | High Frequency |
| 12583 | High Frequency |
| 13748 | Low Frequency |
| 15100 | Low Frequency |
| 15291 | Medium Frequency |
| 14500 | High Frequency |

Group customers into segments based on their purchase frequency, such as high, medium, and low frequency customers. This can help you identify your most loyal customers and those who need more attention.

```
WITH CustomerPurchaseFrequency AS (
    SELECT CustomerID, COUNT(*) AS PurchaseFrequency
    FROM online_retail
    GROUP BY CustomerID
)
SELECT CustomerID,
    CASE
```



```
    WHEN PurchaseFrequency >= 5 THEN 'High Frequency'
    WHEN PurchaseFrequency BETWEEN 2 AND 4 THEN 'Medium Frequency'
    ELSE 'Low Frequency'
END AS FrequencySegment
FROM CustomerPurchaseFrequency;
```

Explanation:

1. **Calculate Purchase Frequency:** The CustomerPurchaseFrequency common table expression (CTE) counts the number of purchases made by each customer within a specified time period. You can adjust the time period using DATE_SUB as needed.
2. **Assign Frequency Segments:** The main query assigns a segment label to each customer based on their PurchaseFrequency. You can customize the segment criteria based on your specific business needs.

Additional Considerations:

- **Time Period:** Adjust the time period for frequency calculation based on your business requirements.
- **Segment Criteria:** Modify the segment criteria (e.g., high, medium, low) to fit your specific needs.
- **Additional Factors:** Consider incorporating other factors like purchase amount or recency into your segmentation.

By executing this query, you'll get a list of customers categorized into high, medium, and low frequency segments, providing valuable insights for targeted marketing and customer retention strategies.

2. Average Order Value by Country

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas

- datamining
- retailshop
 - Tables
 - online_retail
 - Views
 - Stored Procedures
 - Functions
- retailshop_orders
- sakila
- telecom_companydb

SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 8* SQL File 9* x SQL File 10* SQL File 1*

```
1 -- 2. Average Order Value by Country
2 -- Calculate the average order value for each country to identify where your most valua
3 • use retailshop;
4 • SELECT Country, AVG(Quantity * UnitPrice) AS AverageOrderValue
5 FROM online_retail
6 GROUP BY Country
7 ORDER BY AverageOrderValue DESC;
```

Administration Schemas

Information: No object selected

Result Grid

| Country | AverageOrderValue |
|----------------|--------------------|
| Netherlands | 96.30000000000001 |
| France | 42.793 |
| Norway | 26.2895890410959 |
| Australia | 25.589285714285715 |
| United Kingdom | 22.029572147651113 |
| Germany | 17.432000000000002 |

Result 1 x

Output

Action Output

Calculate the average order value for each country to identify where your most valuable customers are located.

Understanding the Task:

We aim to determine the average order value for each country in the dataset. This can help identify regions with higher-spending customers.

SQL Query:

```
SELECT Country, AVG(Quantity * UnitPrice) AS AverageOrderValue
FROM online_retail
```

GROUP BY Country
ORDER BY AverageOrderValue DESC;

Explanation:

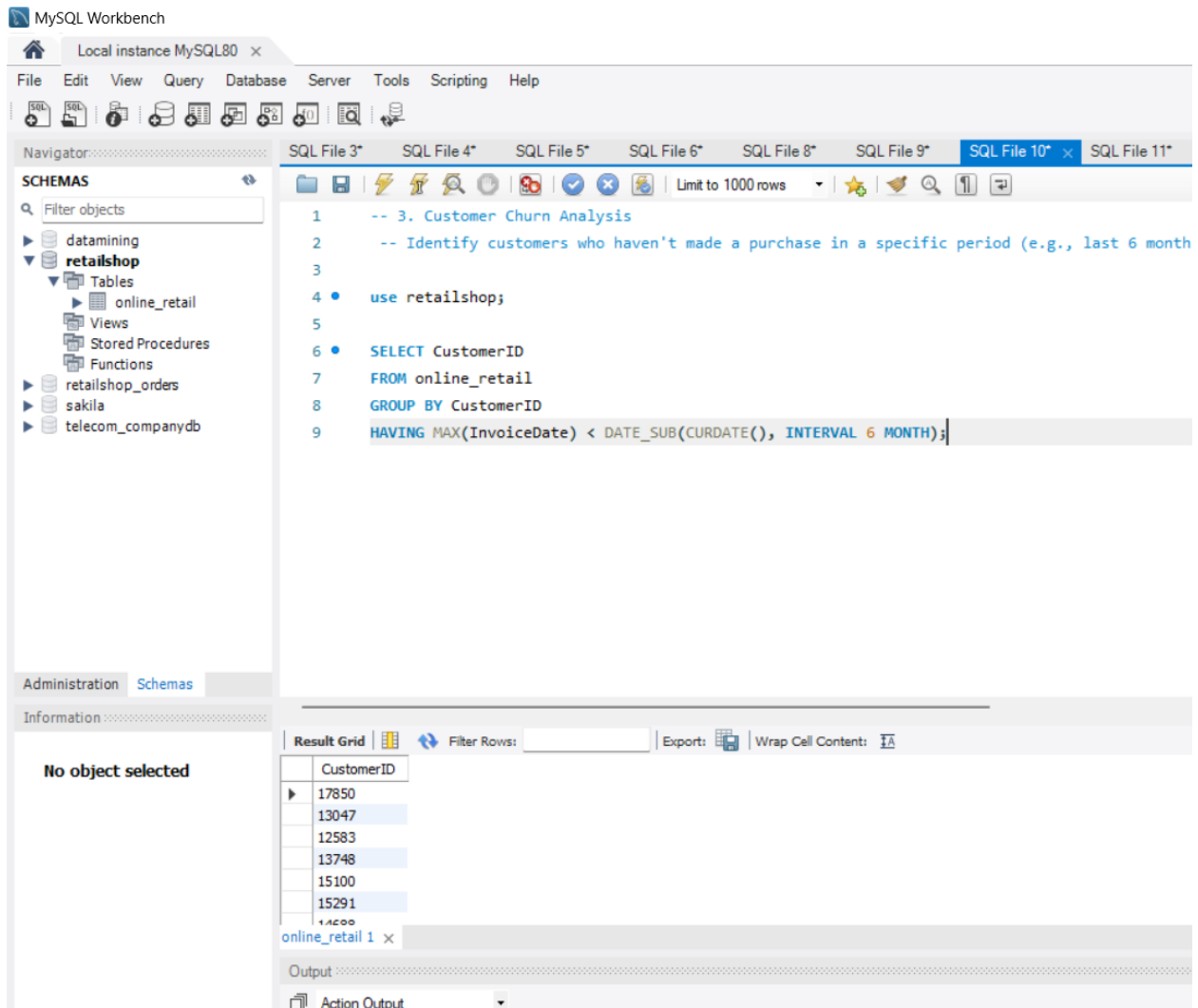
1. **Calculate Total Spending:** For each country, the query calculates the total spending by multiplying Quantity and UnitPrice for each order.
2. **Group by Country:** The results are grouped by Country to get the total spending for each region.
3. **Calculate Average Order Value:** The AVG function calculates the average of the total spending for each country.
4. **Order by Average Order Value:** The results are sorted in descending order to identify countries with the highest average order values.

Additional Considerations:

- **Currency:** If your data includes different currencies, ensure that the UnitPrice values are converted to a common currency before calculating the average.
- **Outliers:** Consider removing outliers (extremely high or low values) to get a more accurate representation of average order values.
- **Time-Based Analysis:** If you have a OrderDate column, you can analyze average order values over time or for specific periods.

By executing this query, you'll get a list of countries and their corresponding average order values, helping you identify regions with higher-spending customers.

3. Customer Churn Analysis



Identify customers who haven't made a purchase in a specific period (e.g., last 6 months) to assess churn.

Understanding the Task:

We aim to identify customers who haven't made a purchase within a specified period, indicating potential churn.

SQL Query:

```
SELECT CustomerID
FROM online_retail
GROUP BY CustomerID
HAVING MAX(InvoiceDate) < DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

Explanation:

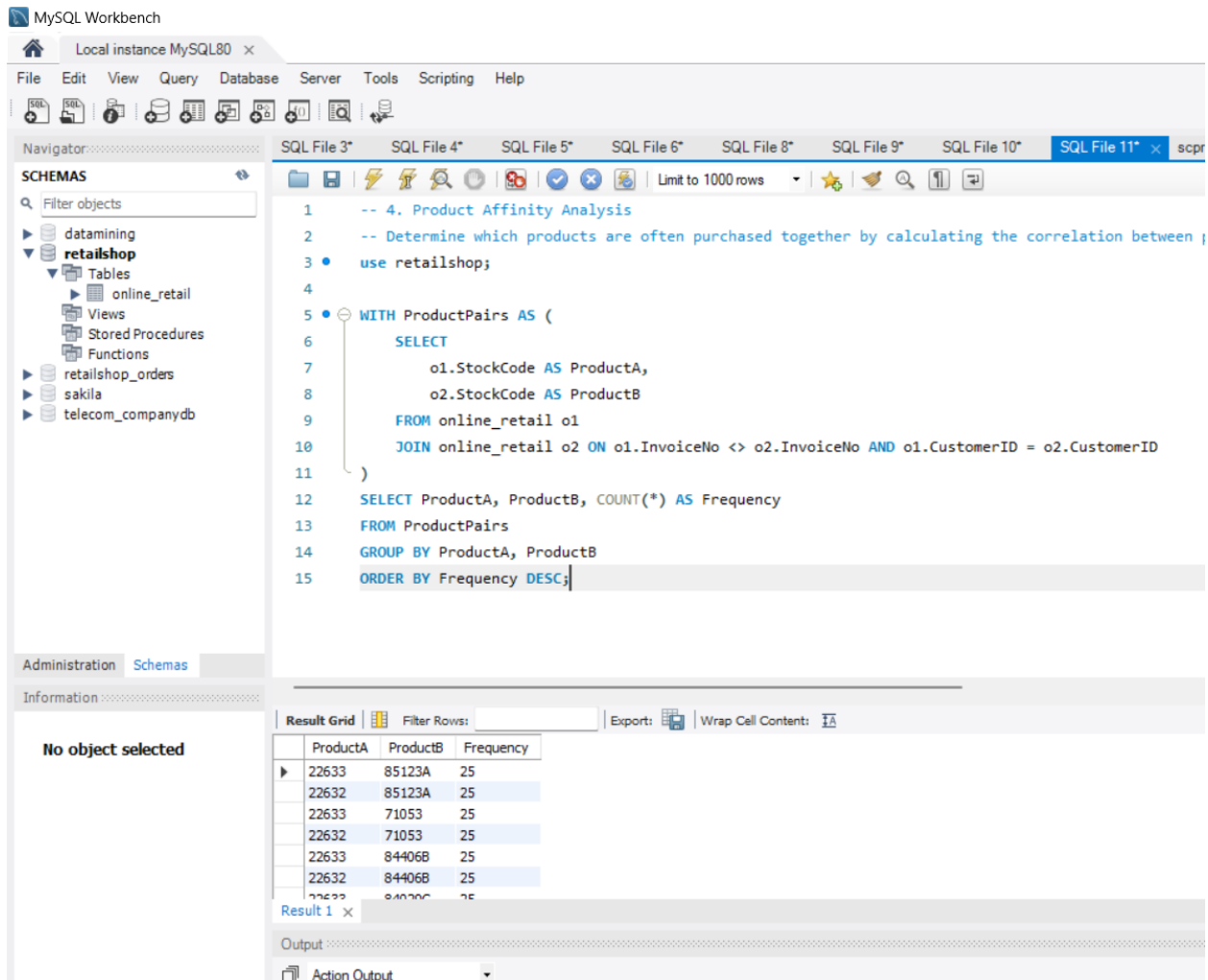
1. **Group by CustomerID:** This groups orders by customer to analyze each customer's purchase history.
2. **Find Last Purchase Date:** MAX(InvoiceDate) finds the most recent purchase date for each customer.
3. **Identify Churned Customers:** The HAVING clause filters for customers whose last purchase date was more than 6 months ago (adjust the interval as needed).

Additional Considerations:

- **Time Period:** Adjust the INTERVAL 6 MONTH to define a different churn threshold.
- **Customer Lifecycle:** Consider other factors like customer lifetime value or purchase frequency when assessing churn.
- **Re-engagement Strategies:** Use the identified churned customers to implement re-engagement campaigns.

By executing this query, you'll get a list of customer IDs who haven't made a purchase in the specified period, indicating potential churn.

4. Product Affinity Analysis



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'retailshop' database selected. The main editor shows a SQL query for Product Affinity Analysis. The query uses a Common Table Expression (CTE) named 'ProductPairs' to identify pairs of products purchased together from the 'online_retail' table. The results are displayed in the 'Result Grid' at the bottom, showing columns for ProductA, ProductB, and Frequency.

```
1  -- 4. Product Affinity Analysis
2  -- Determine which products are often purchased together by calculating the correlation between
3  use retailshop;
4
5  WITH ProductPairs AS (
6      SELECT
7          o1.StockCode AS ProductA,
8          o2.StockCode AS ProductB
9      FROM online_retail o1
10     JOIN online_retail o2 ON o1.InvoiceNo <> o2.InvoiceNo AND o1.CustomerID = o2.CustomerID
11 )
12 SELECT ProductA, ProductB, COUNT(*) AS Frequency
13 FROM ProductPairs
14 GROUP BY ProductA, ProductB
15 ORDER BY Frequency DESC;
```

| ProductA | ProductB | Frequency |
|----------|----------|-----------|
| 22633 | 85123A | 25 |
| 22632 | 85123A | 25 |
| 22633 | 71053 | 25 |
| 22632 | 71053 | 25 |
| 22633 | 84406B | 25 |
| 22632 | 84406B | 25 |

Determine which products are often purchased together by calculating the correlation between product purchases.

Understanding the Task:

We aim to identify products that are frequently purchased together, suggesting a correlation or affinity between them. This can help in product recommendations and cross-selling strategies.

SQL Query:

```
WITH ProductPairs AS (
  SELECT
    o1.StockCode AS ProductA,
    o2.StockCode AS ProductB
```

```
FROM online_retail o1
JOIN online_retail o2 ON o1.InvoiceNo <> o2.InvoiceNo AND o1.CustomerID = o2.CustomerID
)
SELECT ProductA, ProductB, COUNT(*) AS Frequency
FROM ProductPairs
GROUP BY ProductA, ProductB
ORDER BY Frequency DESC;
```

Explanation:

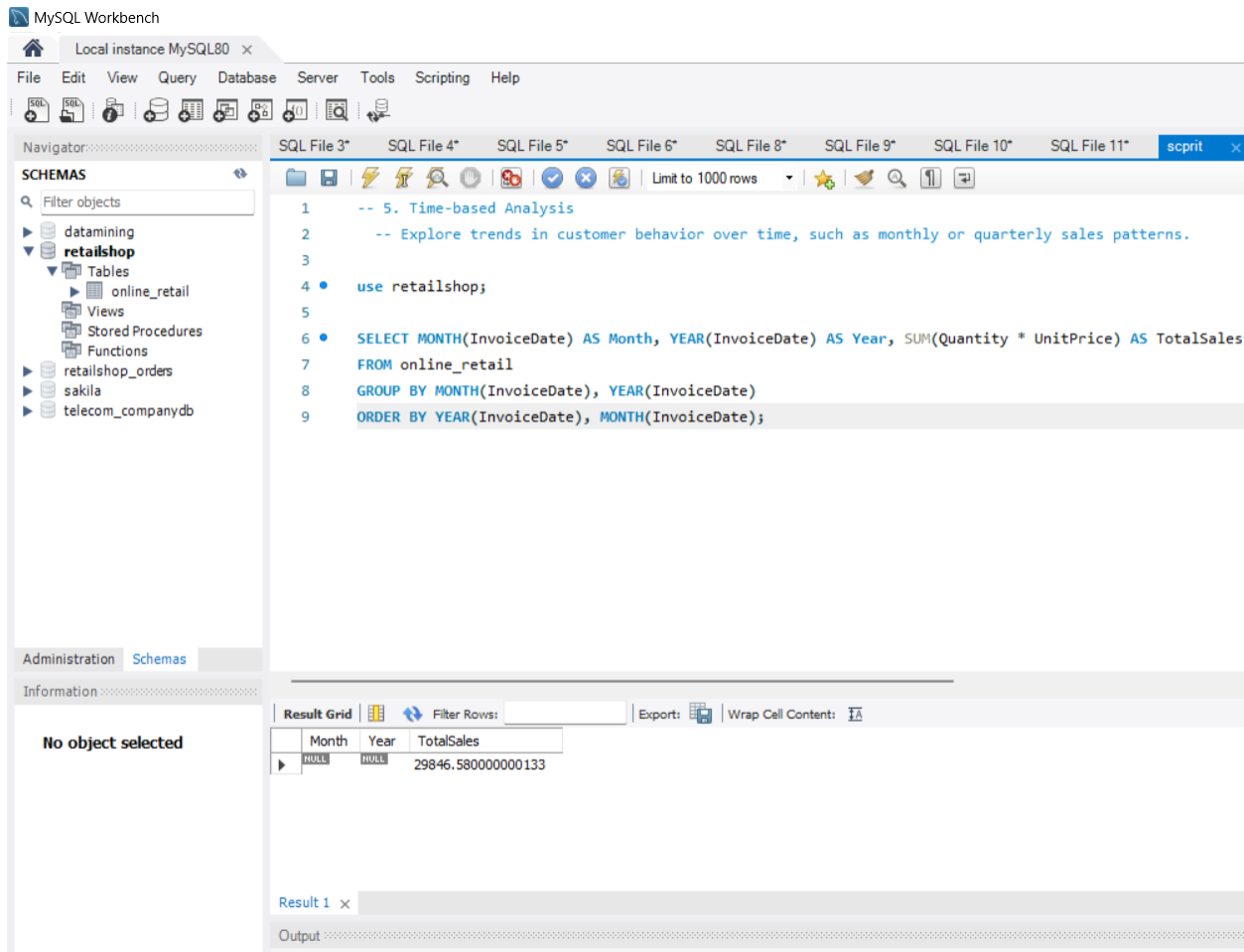
1. **Create Product Pairs:** The ProductPairs common table expression (CTE) identifies pairs of products purchased by the same customer in the same order, excluding pairs of the same product.
2. **Group by Product Pairs:** The GROUP BY clause groups the results by the product pairs.
3. **Count Occurrences:** The COUNT(*) function counts the number of times each product pair appears in the dataset.
4. **Order by Frequency:** The results are ordered by the Frequency column in descending order, so the most commonly purchased pairs appear first.

Additional Considerations:

- **Correlation Measures:** While this query identifies frequent co-purchases, you might want to explore more advanced correlation measures like Jaccard similarity or cosine similarity for a deeper analysis.
- **Product Recommendations:** Use the identified product affinities to recommend products to customers based on their past purchases.
- **Data Cleaning:** Ensure that your data is clean and consistent, handling missing values and duplicates.

By executing this query, you'll get a list of product pairs and their frequency, providing insights into product affinities that can be used for various marketing and sales strategies.

5. Time-based Analysis



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'retailshop' selected. The main editor shows a SQL query in 'SQL File 3*':

```
1  -- 5. Time-based Analysis
2  -- Explore trends in customer behavior over time, such as monthly or quarterly sales patterns.
3
4  • use retailshop;
5
6  • SELECT MONTH(InvoiceDate) AS Month, YEAR(InvoiceDate) AS Year, SUM(Quantity * UnitPrice) AS TotalSales
7  FROM online_retail
8  GROUP BY MONTH(InvoiceDate), YEAR(InvoiceDate)
9  ORDER BY YEAR(InvoiceDate), MONTH(InvoiceDate);
```

The bottom panel shows the 'Result Grid' with the following data:

| Month | Year | TotalSales |
|-------|------|--------------------|
| NULL | NULL | 29846.580000000133 |

Explore trends in customer behavior over time, such as monthly or quarterly sales patterns.

Understanding the Task:

We aim to analyze how customer behavior changes over time, focusing on monthly or quarterly sales patterns.

SQL Query:

```
SELECT MONTH(InvoiceDate) AS Month, YEAR(InvoiceDate) AS Year, SUM(Quantity * UnitPrice)
AS TotalSales
FROM online_retail
GROUP BY MONTH(InvoiceDate), YEAR(InvoiceDate)
ORDER BY YEAR(InvoiceDate), MONTH(InvoiceDate);
```


Explanation:

1. **Extract Month and Year:** The MONTH and YEAR functions extract the month and year from the InvoiceDate column.
2. **Calculate Total Sales:** For each month and year combination, the query calculates the total sales by summing the product of Quantity and UnitPrice.
3. **Group by Month and Year:** The results are grouped by month and year to analyze sales trends over time.
4. **Order by Time:** The results are ordered by year and then month to provide a chronological view of sales data.

Additional Considerations:

- **Time Granularity:** Adjust the time granularity (e.g., daily, weekly) based on your specific analysis needs.
- **Seasonal Patterns:** Consider seasonal factors that might influence sales patterns (e.g., holidays, weather).
- **Trend Analysis:** Use time series analysis techniques to identify trends, seasonality, and other patterns in the data.

By executing this query, you'll get a breakdown of sales by month and year, allowing you to identify trends, seasonal patterns, and potential areas for improvement.