

DM Module Report

Online Retail Segmentation

# Muhammad Abedi

## [Ab3di (github.com)](https://github.com/Ab3di)

# 

# High Impact Skills Development Program

# in Artificial Intelligence, Data Science, and Blockchain

**Summary**

The project aims to enhance their performance by nurturing personalized customer relationships. By harnessing the wealth of information encapsulated within datasets such as "online\_retail," organizations can align their strategies with customer preferences, leading to increased customer satisfaction, strengthened brand loyalty, and ultimately, improved business outcomes. Through such targeted approaches, businesses are better positioned to navigate the intricacies of the modern marketplace and remain competitive in a dynamic and evolving business landscape.

**Introduction**

Customer segmentation stands as a fundamental strategy within the realm of customer analytics, facilitating organizations in effectively categorizing their clientele based on multifaceted attributes such as demographics, purchasing behavior, and customer interactions. This strategic approach, employed by a wide array of enterprises, entails the systematic classification of customers into distinct groups, fostering a deeper understanding of their needs, preferences, and expectations.

In the context of the "module\_project" database and its "online\_retail" table, a diverse set of variables paint a comprehensive picture of customer behavior. This dataset encapsulates an amalgamation of demographic information, purchasing history, and customer interactions, all of which hold the potential to be harnessed for strategic insights. The dataset includes columns such as "InvoiceNo," "StockCode," "Description," "Quantity," "InvoiceDate," "UnitPrice," "CustomerID," and "Country," each contributing a unique facet of customer engagement.

Analysis of this dataset unveils multifarious opportunities for extracting actionable insights. By delving into customer segmentation, businesses can gain a refined understanding of customer behavior patterns. Leveraging attributes such as purchasing frequency, geographical location, and historical interactions, organizations can effectively categorize customers into segments such as "High Frequency," "Medium Frequency," and "Low Frequency" buyers. Such categorizations empower businesses to tailor their marketing efforts, personalize communication, and optimize customer engagement strategies to suit the distinct preferences of each segment.

Furthermore, exploring trends in customer behavior over time uncovers invaluable insights. Through meticulous examination of monthly or quarterly sales patterns, businesses can identify peak periods of customer activity, plan inventory management strategies, and fine-tune marketing campaigns to maximize impact during high-demand periods.

**Beginner Queries**

* Define meta data in mysql workbench

**Database Name**: module\_project

**Table Name**: online\_retail

**Columns:**

1. **InvoiceNo**

Data Type: TEXT

Description: The invoice number associated with the purchase.

1. **StockCode**

Data Type: TEXT

Description: A code that represents the stock of the purchased item.

1. **Description**

Data Type: TEXT

Description: A text description of the purchased item.

1. **Quantity**

Data Type: INT

Description: The quantity of the purchased item in the invoice.

1. **InvoiceDate**

Data Type: TEXT

Description: The date and time when the invoice was created.

1. **UnitPrice**

Data Type: DOUBLE

Description: The unit price of the purchased item.

1. **CustomerID**

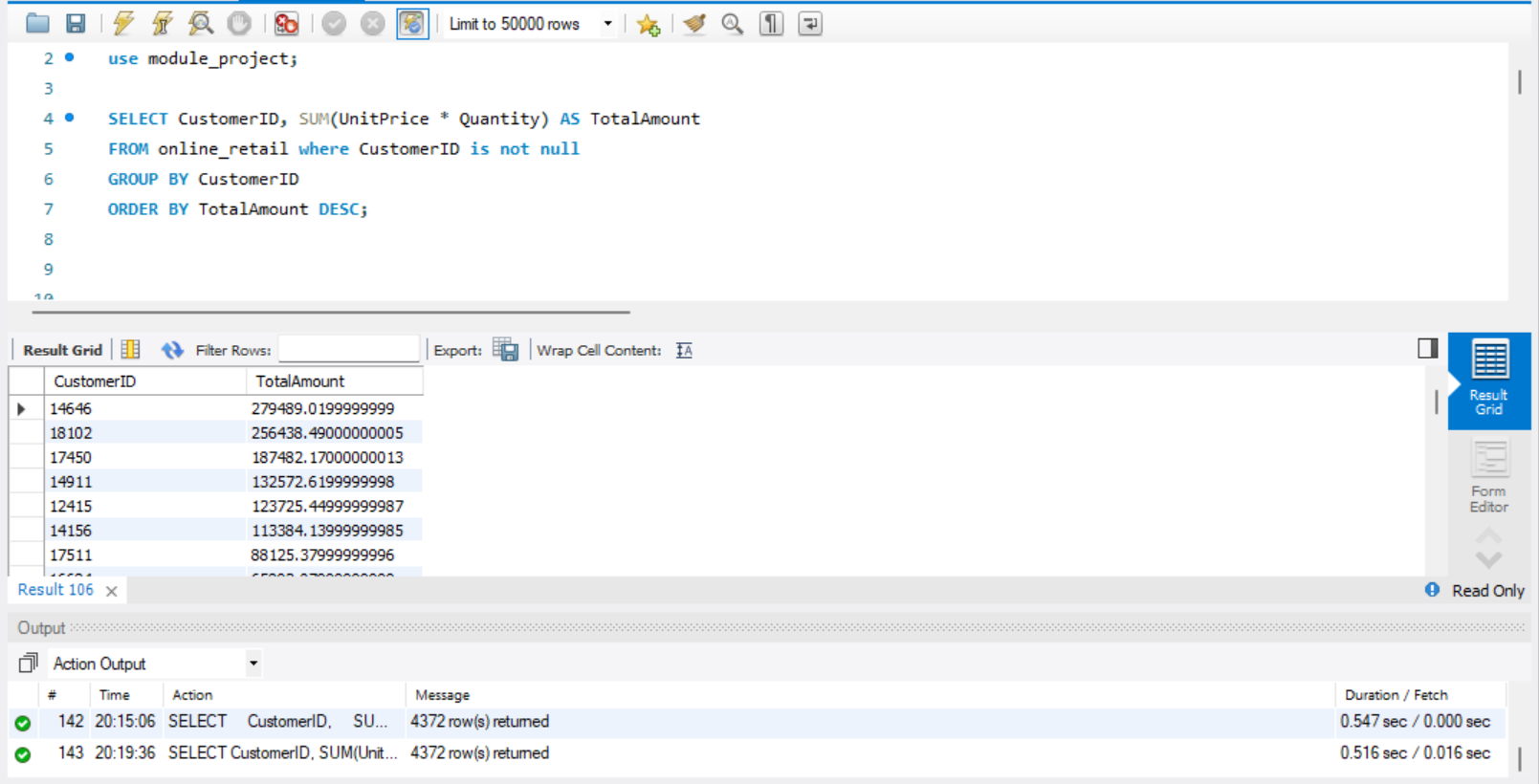
Data Type: INT

Description: The unique identifier of the customer.

1. **Country**

Data Type: TEXT

Description: The country where the customer is located.

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

**Steps:**

**1. SELECT Statement:**

- Select the columns to be displayed in the result.

- Select the `CustomerID` column from the `online\_retail` table.

- Calculate the total amount spent by each customer by multiplying `UnitPrice` and `Quantity` for each row and using the `SUM` aggregate function.

- Alias the calculated total amount as `TotalAmount`.

**2. FROM Clause:**

- Specify the source table for the query as `online\_retail`.

**3. WHERE Clause:**

- Filter the rows where `CustomerID` is not null. This filters out any rows where the `CustomerID` is missing.

**4. GROUP BY Clause**

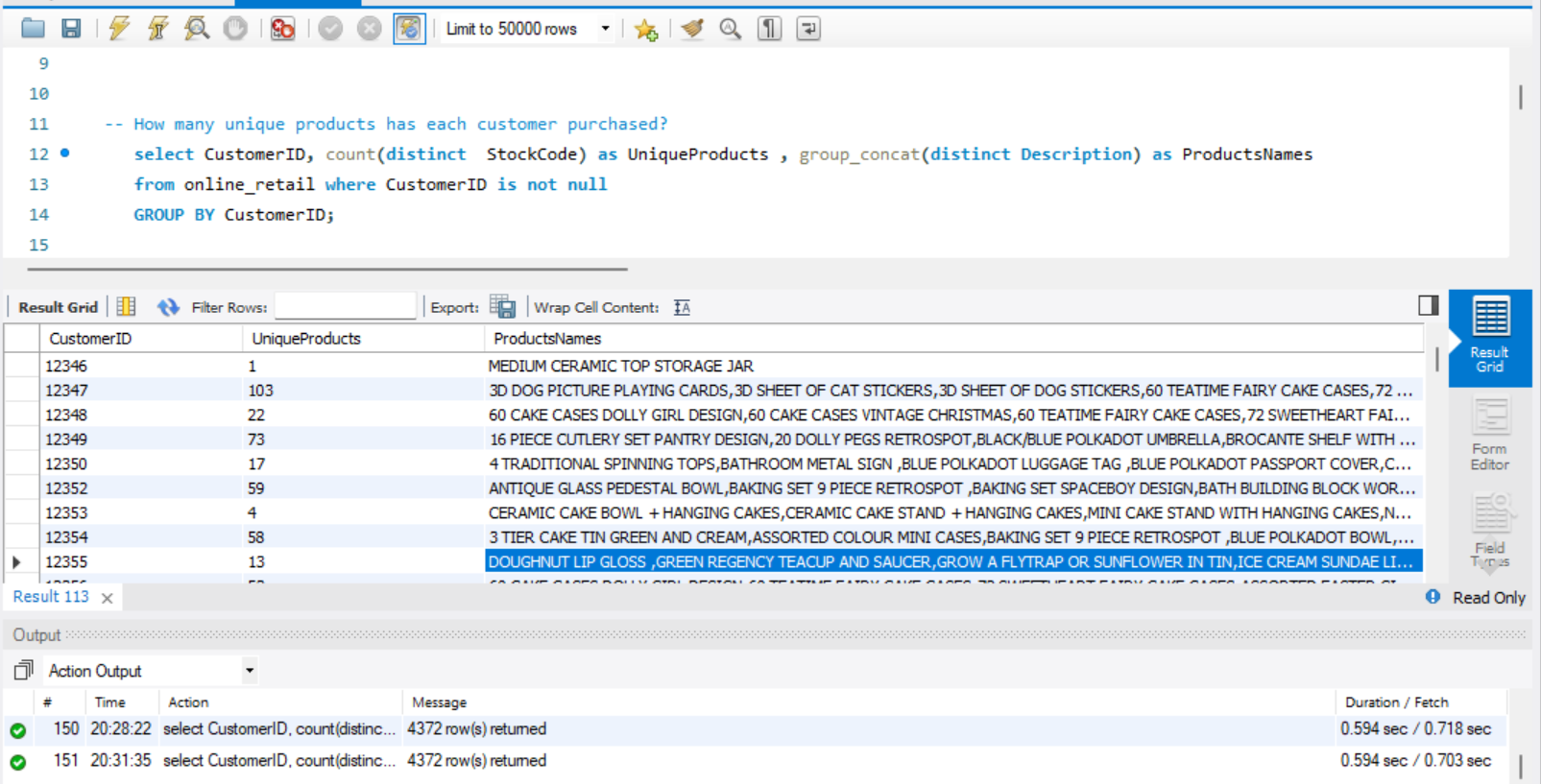
- Group the results by the `CustomerID` column.

- This step is necessary because you want to calculate the total amount for each individual customer.

**5. ORDER BY Clause:**

- Order the results in descending order based on the `TotalAmount`.

- This will ensure that customers who spent the most appear at the top.

* How many unique products has each customer purchased?

**Steps:**

**1. SELECT Statement:**

- Select the columns to be displayed in the result.

- Select the `CustomerID` column from the `online\_retail` table.

- Count the number of distinct `StockCode` values for each customer using the `count` aggregatefunction.

- Alias the calculated count as `UniqueProduct`.

- Use the `group\_concat` function to concatenate distinct `description` values into a comma-separated list for each customer.

- Alias the concatenated list as `ProductsNames`.

**2. FROM Clause:**

- Specify the source table for the query as `online\_retail`.

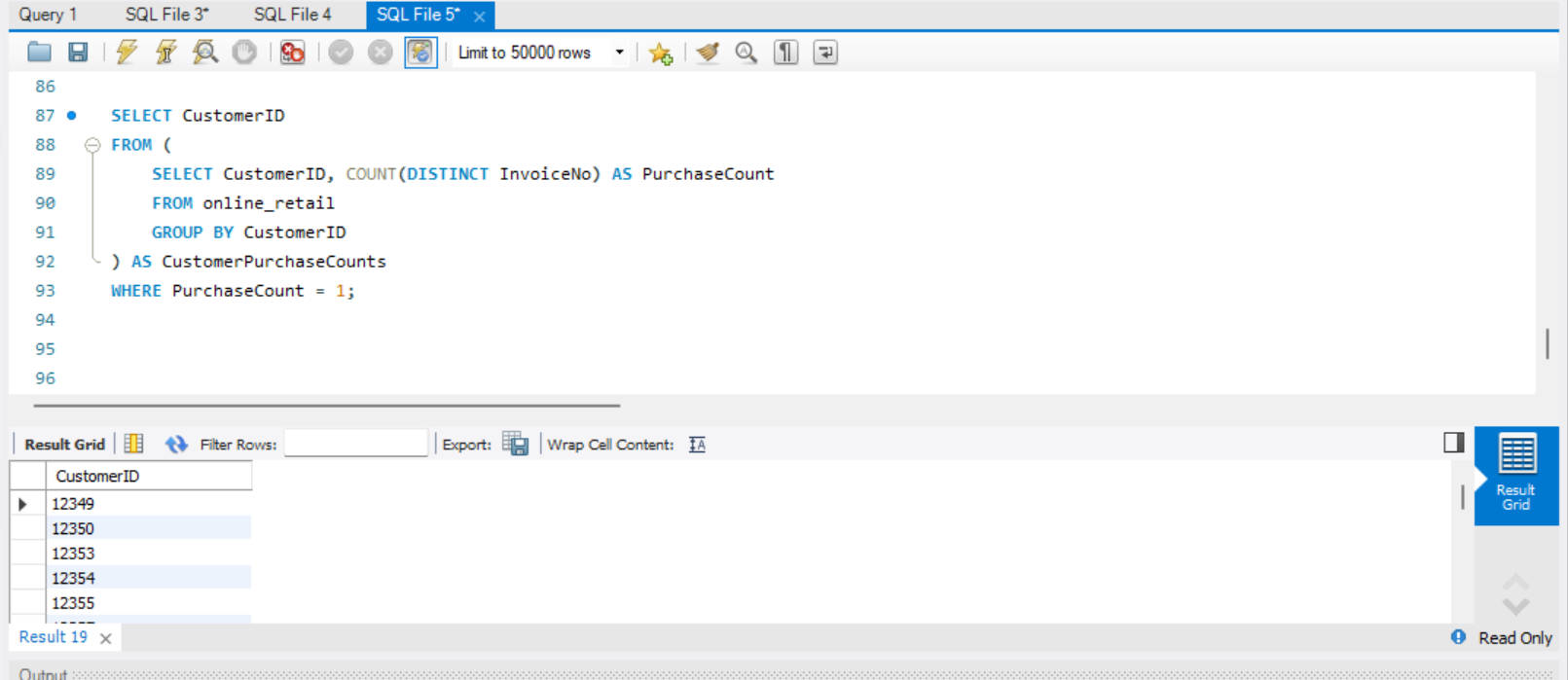
**3. WHERE Clause:**

- Filter the rows where `CustomerID` is not null. This filters out any rows where the `CustomerID` is missing.

**4. GROUP BY Clause:**

- Group the results by the `CustomerID` column.

- This step is necessary because you want to calculate the count of unique products and concatenate product names for each individual customer.

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

**Steps:**

**1. Subquery (Inner Query):**

- In the inner query, the following steps are performed:

- Select the `CustomerID` column from the `online\_retail` table.

- Count the number of distinct `InvoiceNo` values for each customer using the `count` aggregate function.

- Alias the calculated count as `PurchaseCount`.

- Use the `group by` clause to group the results by the `CustomerID` column.

- The result of this subquery is a table that shows each customer's ID and their corresponding purchase count.

**2. Outer Query:**

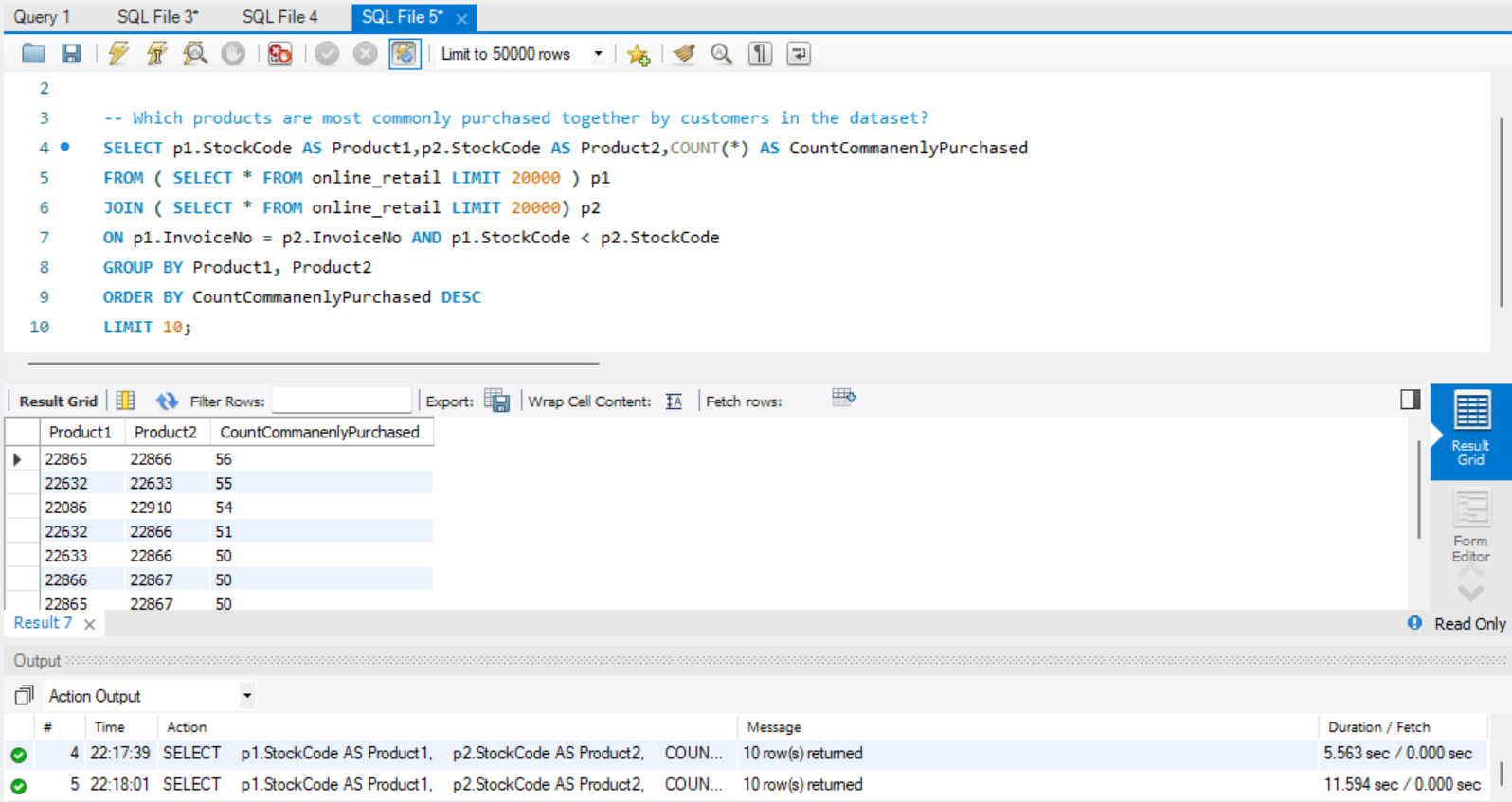
- In the outer query, the following steps are performed:

- The subquery result is treated as a derived table and given an alias `CustomerPurchaseCounts`.

- Select the `CustomerID` column from the `CustomerPurchaseCounts` derived table.

- Filter the results where `PurchaseCount` is equal to 1.

- The final result displays the `CustomerID` of customers who have made only a single purchase.

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

**Steps:**

**1. Subqueries (Inner Queries):**

- Two identical subqueries are used, aliased as `p1` and `p2`.

- Each subquery performs the following steps:

- Select all columns from the `online\_retail` table.

- Use the `LIMIT` clause to limit the result to 20,000 rows. This is done for optimization purposes, likely to reduce the computational load while testing the query.

**2. Join:**

- The two subqueries `p1` and `p2` are used in a join operation.

- The join is based on two conditions:

- Matching `InvoiceNo` values between `p1` and `p2`.

- Ensuring that the `StockCode` in `p1` is less than the `StockCode` in `p2`. This avoids duplicate pairs and ensures each combination is counted only once.

**3. Grouping:**

- The joined result is then grouped by `Product1` (which is the `StockCode` from `p1`) and `Product2` (which is the `StockCode` from `p2`).

- This grouping aggregates the count of how many times each pair of products appears together in the data.

**4. Aggregation:**

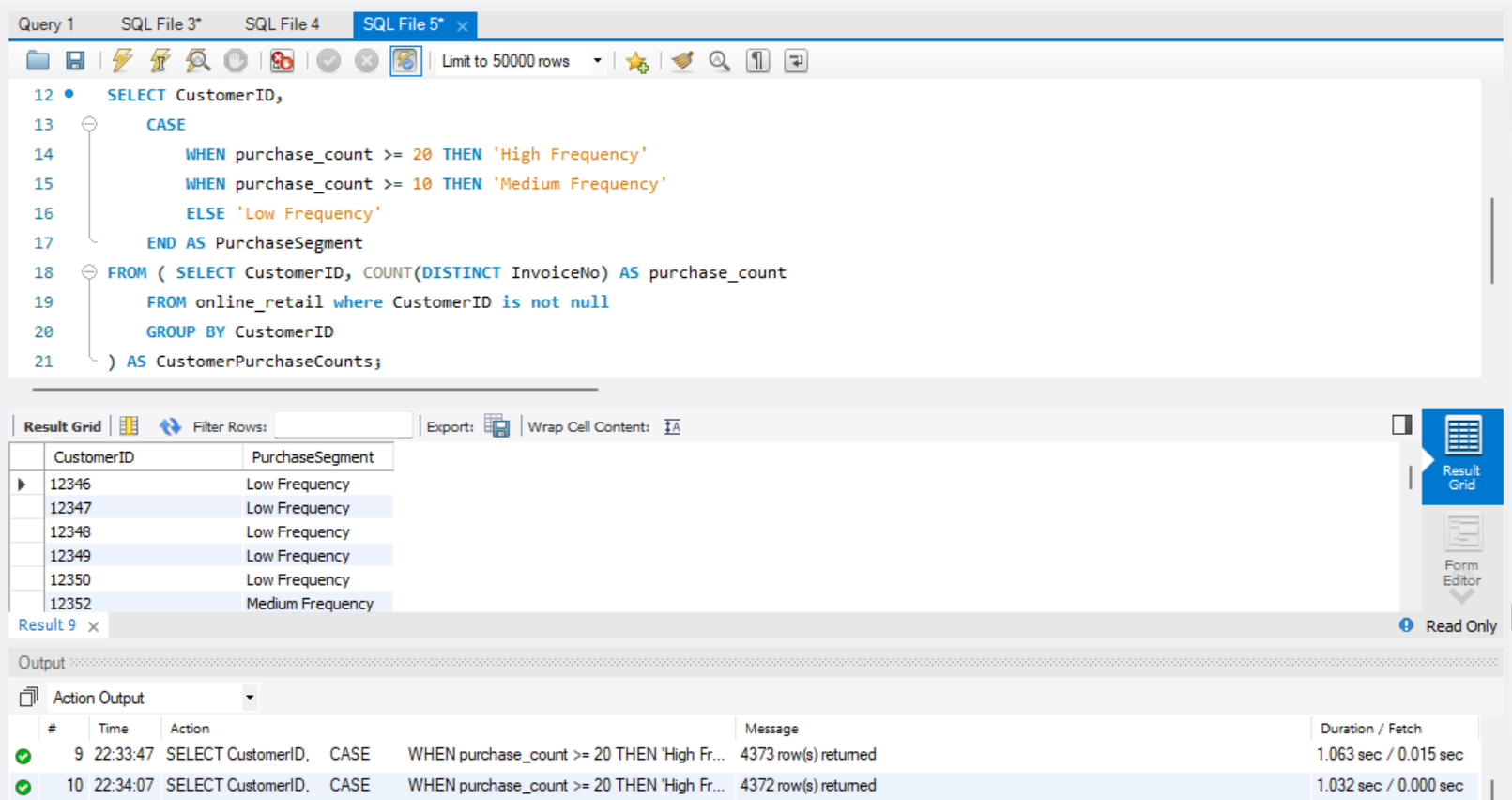
- The `COUNT()` function is used to calculate the number of times each pair of products has been commonly purchased together.

**5. Sorting and Limiting:**

- The results are ordered in descending order based on the `CountCommanenlyPurchased` value.

- The `LIMIT` clause is applied to restrict the output to the top 10 results, showing the pairs of products that are most commonly purchased together.

**Advance Queries**

1. **Customer Segmentation by Purchase Frequency**

**Steps:**

**1. Subquery (Inner Query):**

- The subquery performs the following steps:

- Selects `CustomerID` and calculates the count of distinct `InvoiceNo` values for each customer.

- Uses a filter (`WHERE CustomerID IS NOT NULL`) to exclude rows where `CustomerID` is null.

- Groups the results by `CustomerID`.

**2. Main Query:**

- The main query is based on the results of the subquery.

- It selects `CustomerID` and introduces a calculated column `PurchaseSegment` using the `CASE` statement to categorize customers based on their purchase frequency.

- The `CASE` statement has the following conditions:

- If the `purchase\_count` is greater than or equal to 20, the customer is labeled as 'High Frequency'.

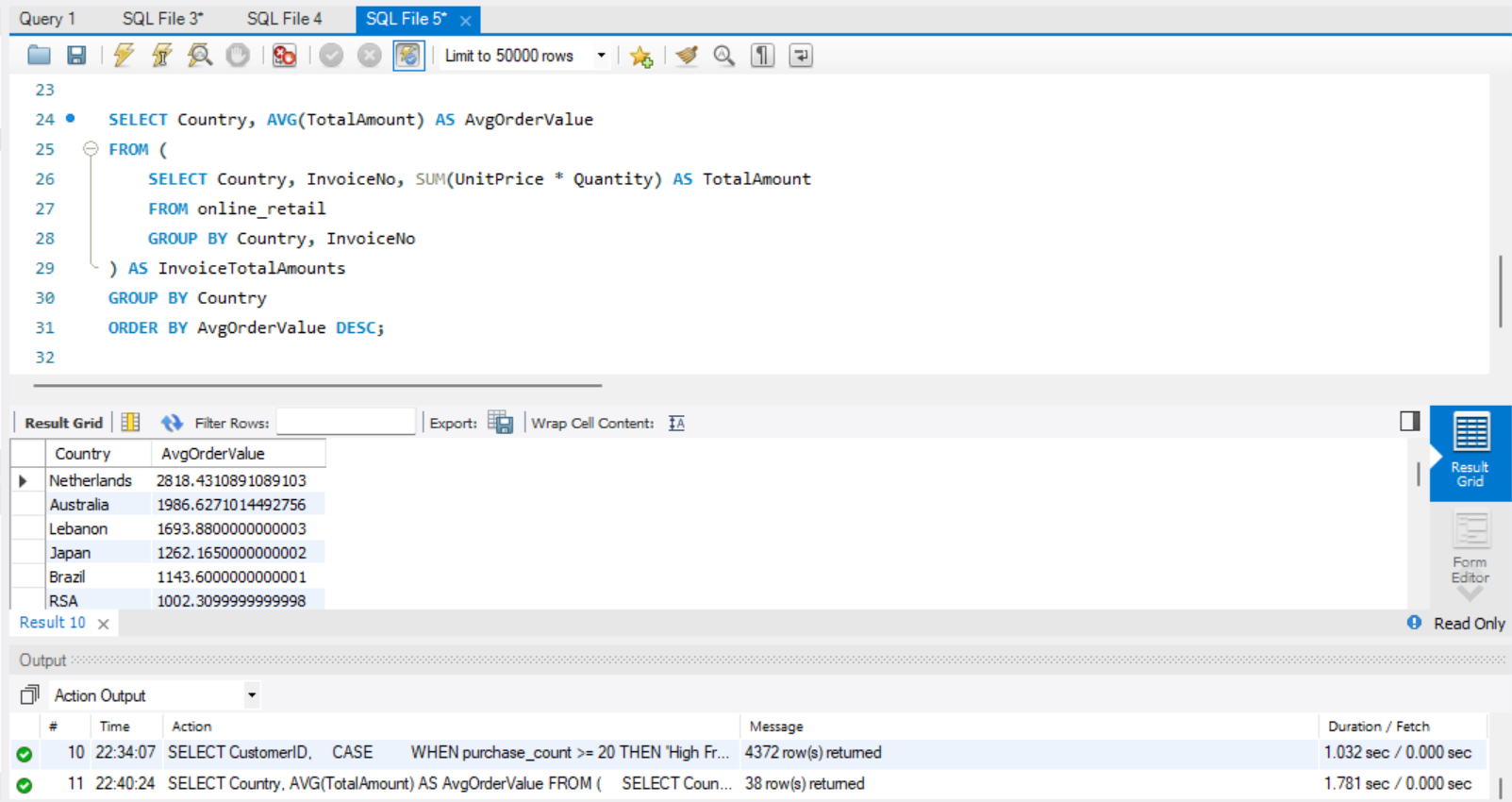
- If the `purchase\_count` is greater than or equal to 10 (but less than 20), the customer is labeled as 'Medium Frequency'.

- For all other cases (purchase count less than 10), the customer is labeled as 'Low Frequency'.

**3. Result:**

- The final result set includes two columns: `CustomerID` and `PurchaseSegment`.

- Each row represents a customer with their corresponding purchase segment based on the defined conditions.

**2. Average Order Value by Country**

**Steps:**

**1. Subquery (Innermost Query):**

- The innermost subquery performs the following steps:

- Selects `Country`, `InvoiceNo`, and calculates the total amount for each invoice by multiplying `UnitPrice` and `Quantity`.

- Uses the `GROUP BY` clause to group the results by `Country` and `InvoiceNo`.

**2. Middle Subquery (Intermediate Query):**

- The middle subquery is based on the results of the innermost subquery.

- It performs the following steps:

- Uses the results from the innermost subquery.

- Calculates the sum of the total amounts for each `Country` and `InvoiceNo` combination.

- Retains the `Country` and calculated `TotalAmount` columns.

**3. Main Query:**

- The main query is based on the results of the middle subquery.

- It selects `Country` and calculates the average order value (`AvgOrderValue`) for each country using the aggregated `TotalAmount` from the middle subquery.

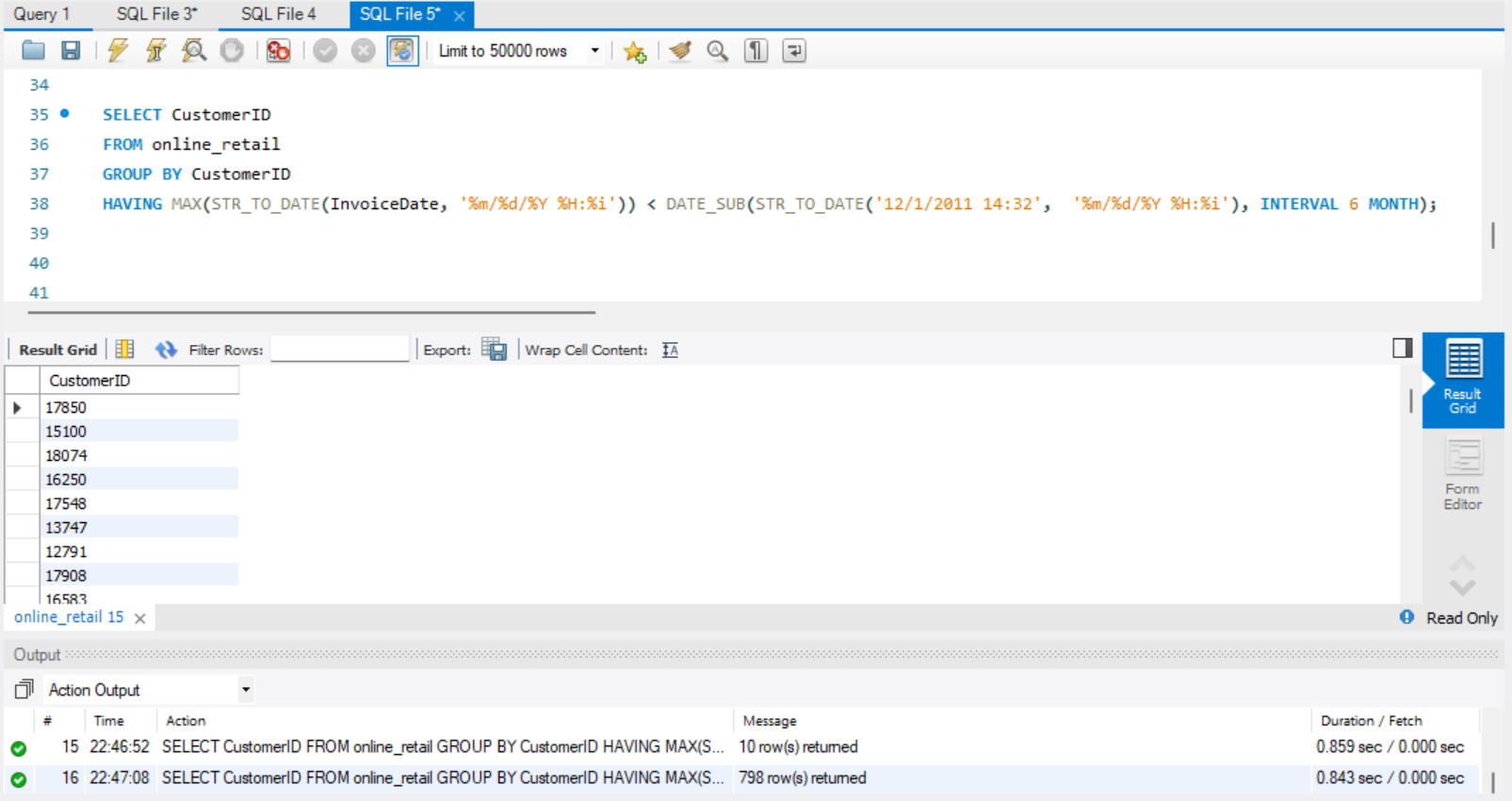
- Uses the `GROUP BY` clause to group the results by `Country`.

- The result also includes an `ORDER BY` clause to sort the countries based on the calculated average order values in descending order.

**4. Result:**

- The final result set includes two columns: `Country` and `AvgOrderValue`.

- Each row represents a country along with its corresponding average order value.

**3. Customer Churn Analysis**

**Steps:**

**1. Subquery (Innermost Query):**

- The innermost subquery performs the following steps:

- Selects `CustomerID` and the maximum invoice date for each customer using the `MAX` function and `STR\_TO\_DATE` function to convert the `InvoiceDate` string to a date format.

- Uses the `GROUP BY` clause to group the results by `CustomerID`.

**2. Main Query:**

- The main query is based on the results of the innermost subquery.

- It selects `CustomerID` from the results of the innermost subquery.

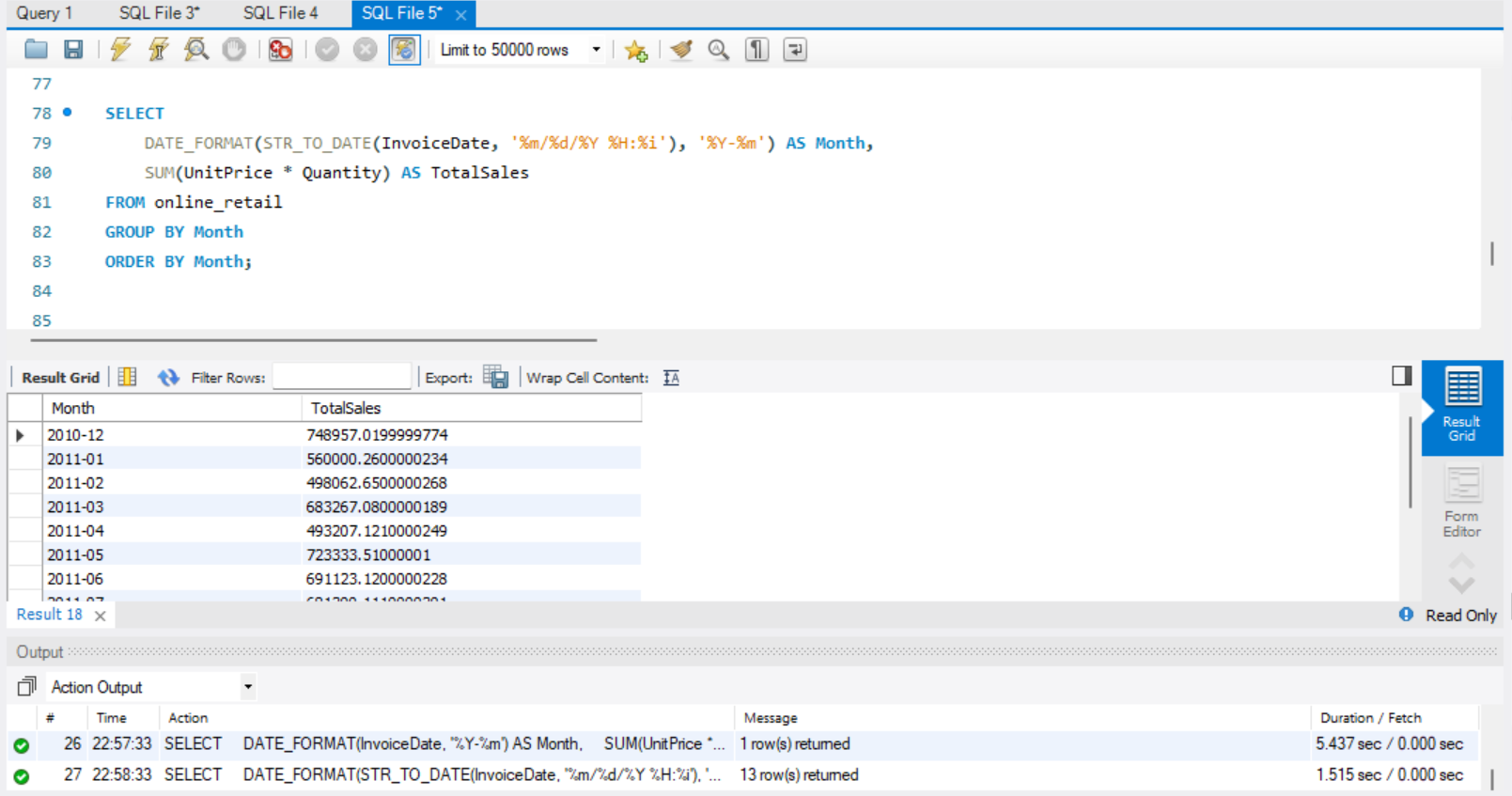
- Uses the `GROUP BY` clause to group the results by `CustomerID`.

- Includes a `HAVING` clause to filter out customers whose maximum invoice date is more than 6 months before the specified date.

**3. Result:**

- The final result set includes one column: `CustomerID`.

- Each row represents a customer who hasn't made a purchase in the last 6 months based on the specified condition.

**5. Time-based Analysis**

**Steps:**

**1. SELECT Clause:**

- The `SELECT` clause selects two columns:

- `DATE\_FORMAT(InvoiceDate, '%Y-%m') AS Month`: This column uses the `DATE\_FORMAT` function to extract and format the year and month from the `InvoiceDate` column in the format 'YYYY-MM'.

- `SUM(UnitPrice Quantity) AS TotalSales`: This column calculates the total sales for each month by multiplying the `UnitPrice` and `Quantity` columns and summing them up.

**2. FROM Clause:**

- The `FROM` clause specifies the source table as `online\_retail`.

**3. GROUP BY Clause:**

- The `GROUP BY` clause groups the data by the formatted `Month` column.

- This allows the aggregation of sales data on a monthly basis.

**4. ORDER BY Clause:**

- The `ORDER BY` clause arranges the results in ascending order of the formatted `Month` column.

- This ensures that the sales data is displayed in chronological order.

**5. Result:**

- The final result set includes two columns: `Month` and `TotalSales`.

- Each row represents a month and its corresponding total sales.

**The End**

**Ab3di**