



ONLINE RETAIL SEGMENTATION

DATA MINING FINAL PROJECT



Presented by
AFTAB NAFEES
GitHub Username
aftab1038

Repository Link
<https://shorturl.at/5tZOof>

Table of Contents

Introduction	03
Beginner Queries	04
Metadata Definition	04
Distribution of Order Values	05
Unique Products Purchased	06
Single Purchase Customers	07
Commonly Purchased Products Together	08
Advance Queries	09
Customer Segmentation by Purchase Frequency	09
Average Order Value by Country	10
Customer Churn Analysis	11
Product Affinity Analysis	12
Time-based Analysis	13
Conclusion	14

Introduction

The objective of this project is to perform customer segmentation using data mining techniques, specifically through SQL. Customer segmentation allows businesses to categorize clients based on various factors such as demographics and purchasing patterns. This segmentation is crucial for tailoring marketing strategies, improving customer satisfaction, and boosting overall business performance.

The dataset used in this project contains transaction-level data from an online retail store, with key variables including:

- **InvoiceNo:** Unique invoice number for each transaction.
- **StockCode:** Unique product code.
- **Description:** Product description.
- **Quantity:** Quantity of the product sold.
- **InvoiceDate:** Date and time of the transaction.
- **UnitPrice:** Price per unit of the product.
- **CustomerID:** Unique identifier for each customer.
- **Country:** Country where the transaction occurred.

Number of Rows: This dataset contains **581587** rows

Beginner Queries

Metadata Definition

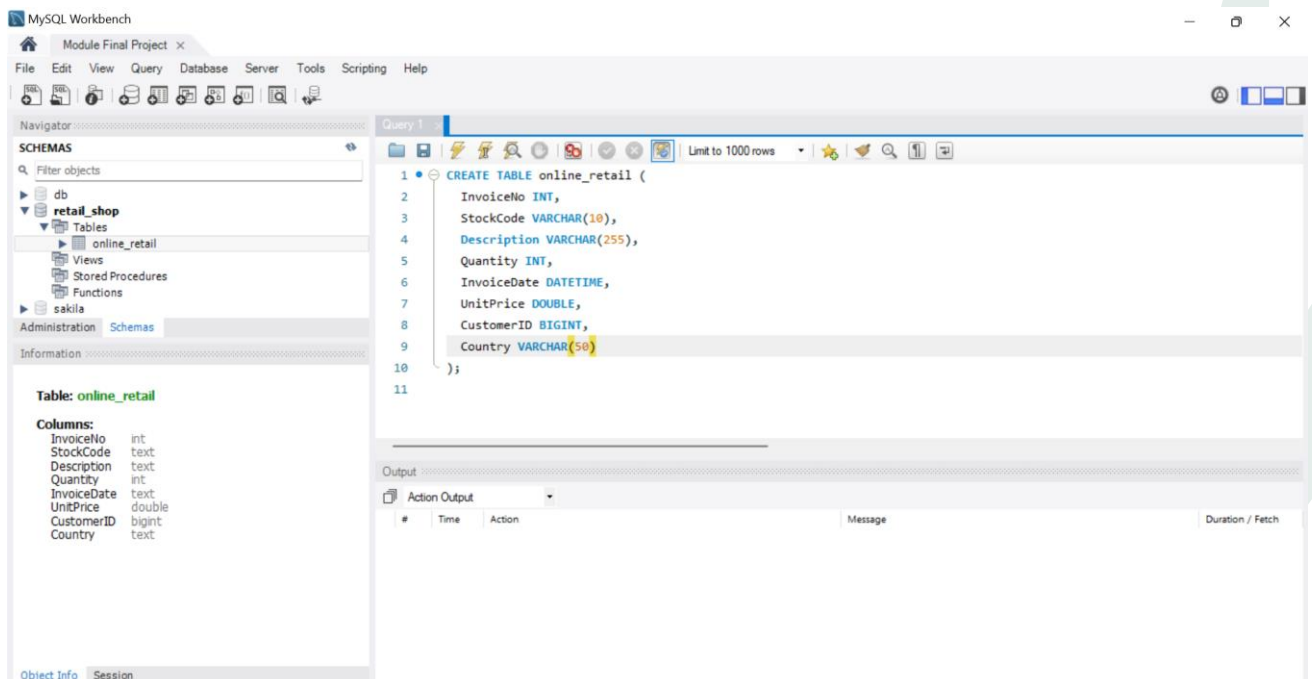
Query:

```
CREATE TABLE online_retail (  
  InvoiceNo VARCHAR(10),  
  StockCode VARCHAR(10),  
  Description VARCHAR(255),  
  Quantity INT,  
  InvoiceDate DATETIME,  
  UnitPrice DECIMAL(10, 2),  
  CustomerID VARCHAR(10),  
  Country VARCHAR(50)  
);
```

Explanation:

This query creates a table structure in SQL, defining the necessary fields to store and manipulate the dataset.

Output:



Distribution of Order Values

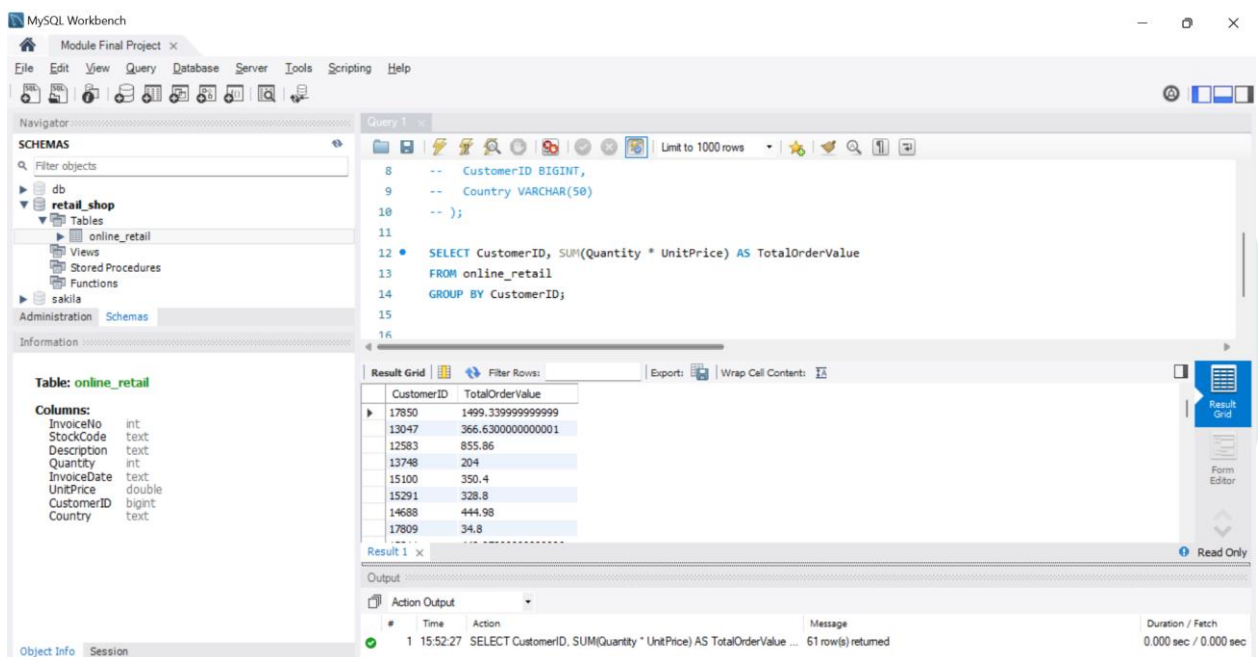
Query:

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

Explanation:

This query calculates the total value of orders placed by each customer. It is crucial for understanding the spending habits of different customer segments.

Output:



The screenshot shows the MySQL Workbench interface. The 'Schemas' pane on the left shows the 'retail_shop' database with the 'online_retail' table selected. The 'Query' editor in the center contains the SQL query. The 'Result Grid' on the right displays the results of the query, showing columns 'CustomerID' and 'TotalOrderValue'. The 'Output' pane at the bottom shows the execution details.

Table: online_retail

Columns:

- InvoiceNo: int
- StockCode: text
- Description: text
- Quantity: int
- InvoiceDate: text
- UnitPrice: double
- CustomerID: bigint
- Country: text

Query 1:

```
-- CustomerID BIGINT,
-- Country VARCHAR(50)
-- );
SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalOrderValue
FROM online_retail
GROUP BY CustomerID;
```

Result Grid:

CustomerID	TotalOrderValue
17850	1499.3399999999999
13047	366.63000000000001
12583	855.86
13748	204
15100	350.4
15291	328.8
14688	444.98
17809	34.8

Output:

#	Time	Action	Message	Duration / Fetch
1	15:52:27	SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalOrderValue ...	61 row(s) returned	0.000 sec / 0.000 sec

Unique Products Purchased

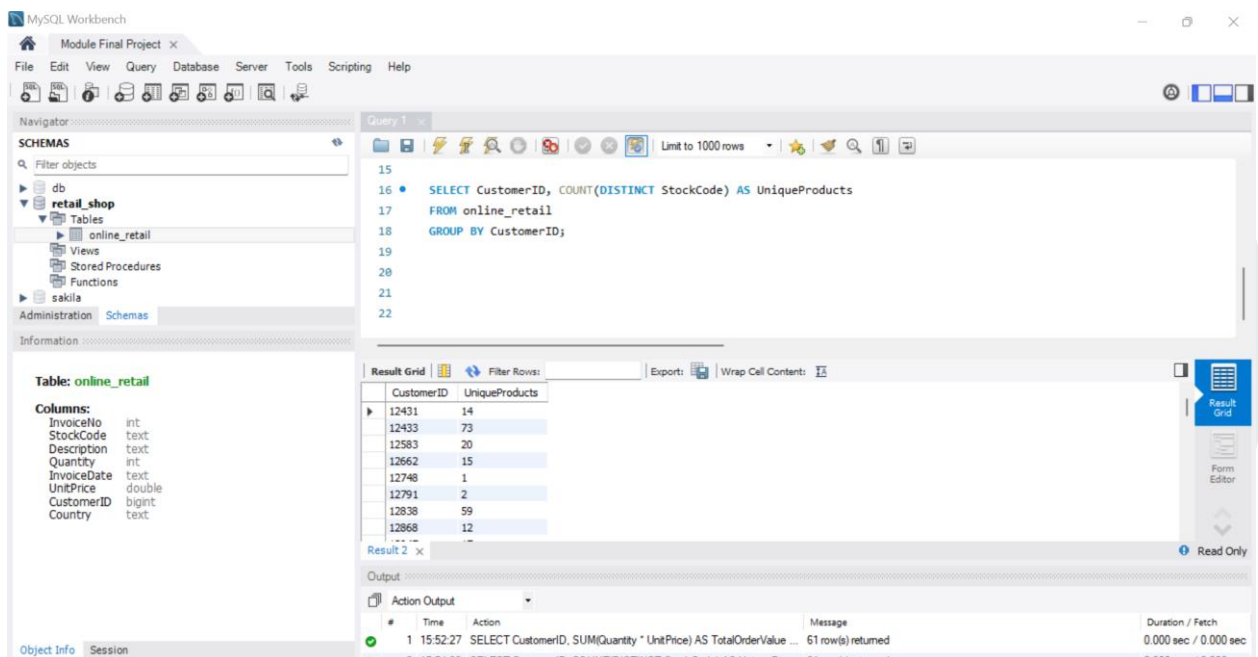
Query:

```
SELECT CustomerID, COUNT(DISTINCT StockCode)
As UniqueProducts
FROM retailGROUP
BY CustomerID;
```

Explanation:

This query determines how many unique products each customer has purchased, offering insights into customer diversity and product reach.

Output:



The screenshot displays the MySQL Workbench interface. The 'Query 1' window contains the following SQL query:

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

The 'Result Grid' shows the output of the query, with columns 'CustomerID' and 'UniqueProducts'. The results are as follows:

CustomerID	UniqueProducts
12431	14
12433	73
12583	20
12662	15
12748	1
12791	2
12838	59
12868	12

The 'Table: online_retail' section on the left lists the columns and their data types:

- InvoiceNo: int
- StockCode: text
- Description: text
- Quantity: int
- InvoiceDate: text
- UnitPrice: double
- CustomerID: bigint
- Country: text

The 'Output' section at the bottom shows the execution details, including the time taken (15:52:27) and the message: 'SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalOrderValue ... 61 row(s) returned'. The duration is 0.000 sec / 0.000 sec.

Single Purchase Customers

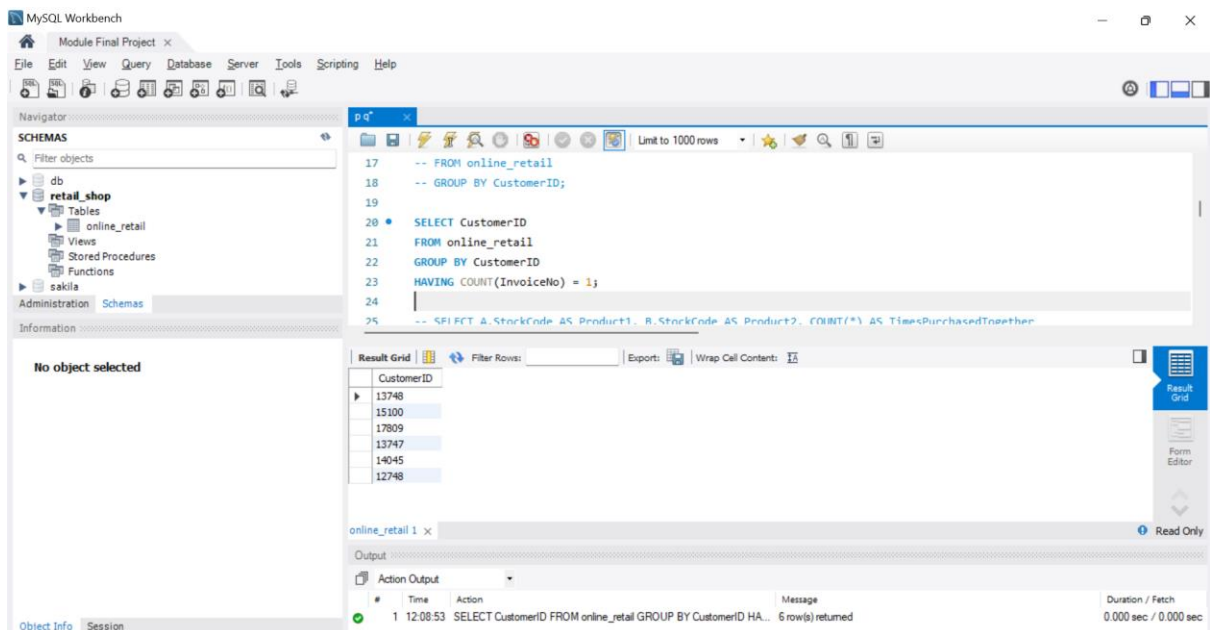
Query:

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

Explanation:

Identifying customers who have made only a single purchase can help in designing targeted retention strategies.

Output:



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'db' expanded, containing 'retail_shop' and 'sakila'. The 'retail_shop' schema is selected, showing 'online_retail' as a table. The main editor window shows a SQL query:

```
-- FROM online_retail
-- GROUP BY CustomerID;

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

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

CustomerID
13748
15100
17809
13747
14045
12748

The bottom status bar indicates the query was executed at 12:08:53, returning 6 rows in 0.000 seconds.

Commonly Purchased Products Together

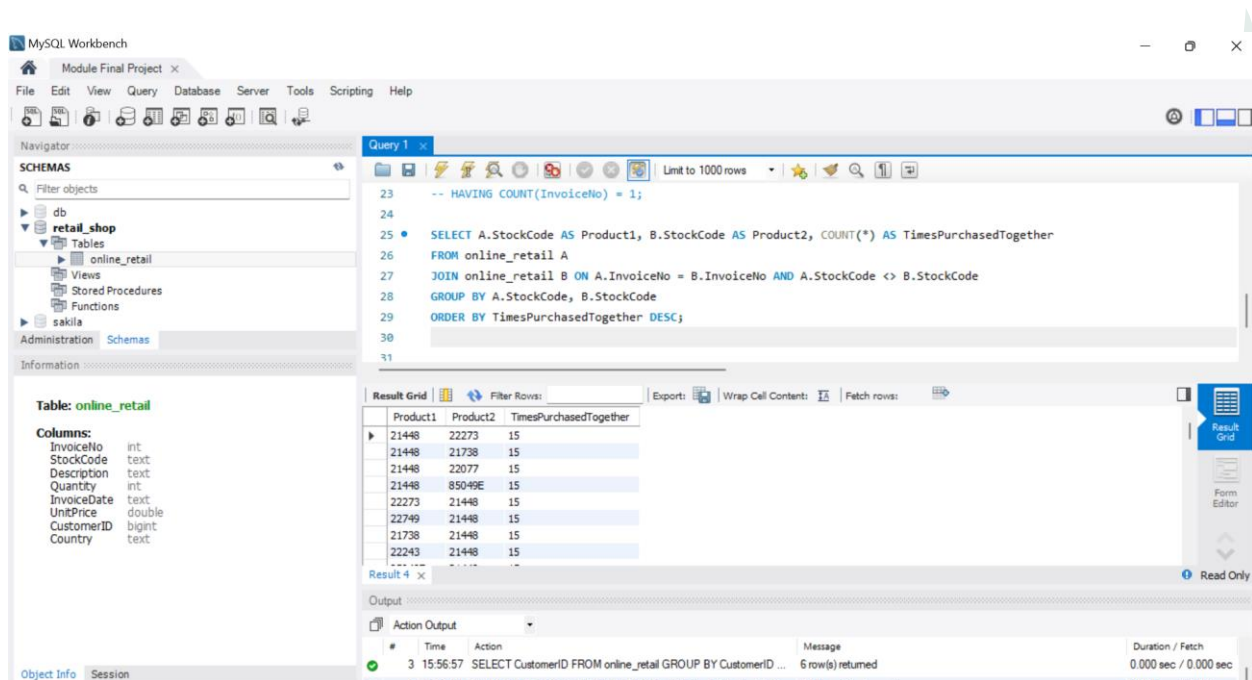
Query:

```
SELECT A.StockCode AS Product1, B.StockCode AS Product2,  
COUNT(*)  
AS TimesPurchasedTogether  
FROM online_retail A  
JOIN online_retail B  
ON A.InvoiceNo = B.InvoiceNo AND  
A.StockCode <> B.StockCode  
GROUP BY A.StockCode, B.StockCode  
ORDER BY TimesPurchasedTogether DESC;
```

Explanation:

This query finds pairs of products that are often purchased together by customers. It helps in identifying product bundling opportunities or understanding customer buying behavior.

Output:



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'Schemas' pane with 'db' and 'sakila' databases. The 'retail_shop' database is selected, showing 'online_retail' as a table. The main editor shows the SQL query. The 'Result Grid' at the bottom displays the query results.

Table: online_retail

Columns:

- InvoiceNo: int
- StockCode: text
- Description: text
- Quantity: int
- InvoiceDate: text
- UnitPrice: double
- CustomerID: bigint
- Country: text

Result Grid:

Product1	Product2	TimesPurchasedTogether
21448	22273	15
21448	21738	15
21448	22077	15
21448	85049E	15
22273	21448	15
22749	21448	15
21738	21448	15
22243	21448	15

Action Output:

#	Time	Action	Message	Duration / Fetch
3	15:56:57	SELECT CustomerID FROM online_retail GROUP BY CustomerID ...	6 row(s) returned	0.000 sec / 0.000 sec

Advance Queries

Customer Segmentation by Purchase Frequency

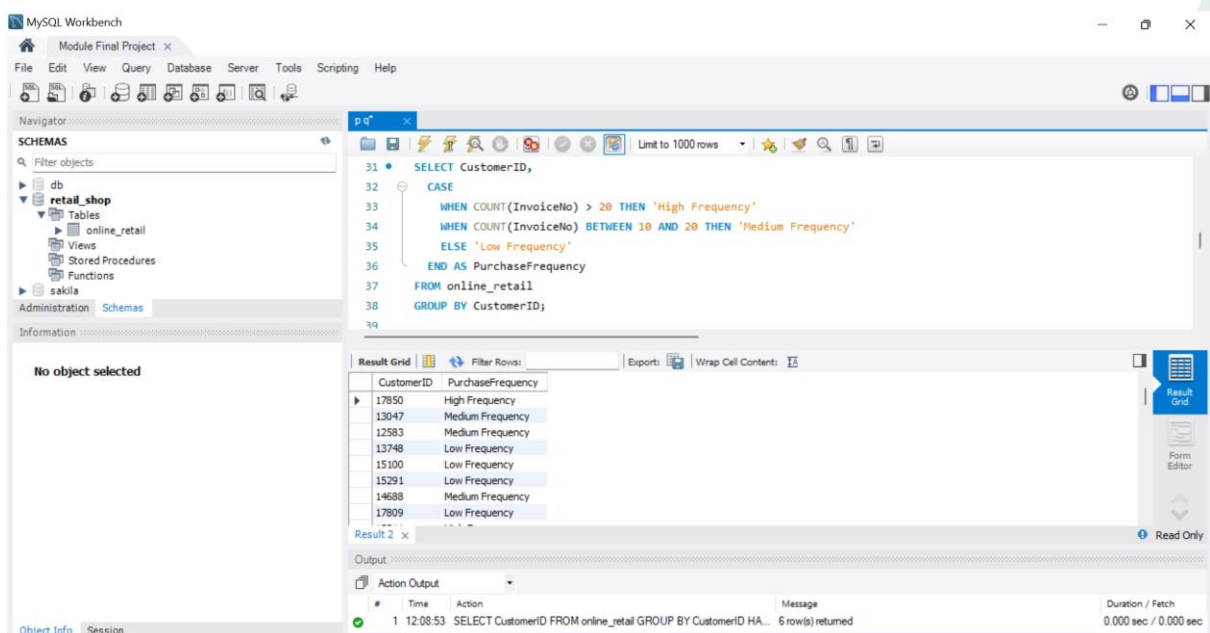
Query:

```
SELECT CustomerID,  
CASE  
    WHEN COUNT(InvoiceNo) > 20 THEN 'High Frequency'  
    WHEN COUNT(InvoiceNo) BETWEEN 10 AND 20 THEN 'Medium  
Frequency'  
    ELSE 'Low Frequency'  
END AS PurchaseFrequency  
FROM online_retail  
GROUP BY CustomerID;
```

Explanation:

Customers are segmented based on their purchase frequency, helping identify loyal customers and those who may need more attention.

Output:



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with 'db' and 'retail_shop' expanded, listing 'online_retail' as a table. The main editor window contains the SQL query for customer segmentation. Below the query, the 'Result Grid' shows the output of the query, displaying columns 'CustomerID' and 'PurchaseFrequency'. The 'Output' panel at the bottom shows the execution details, including the time taken and the number of rows returned.

CustomerID	PurchaseFrequency
17850	High Frequency
13047	Medium Frequency
12583	Medium Frequency
13748	Low Frequency
15100	Low Frequency
15291	Low Frequency
14688	Medium Frequency
17809	Low Frequency

Average Order Value by Country

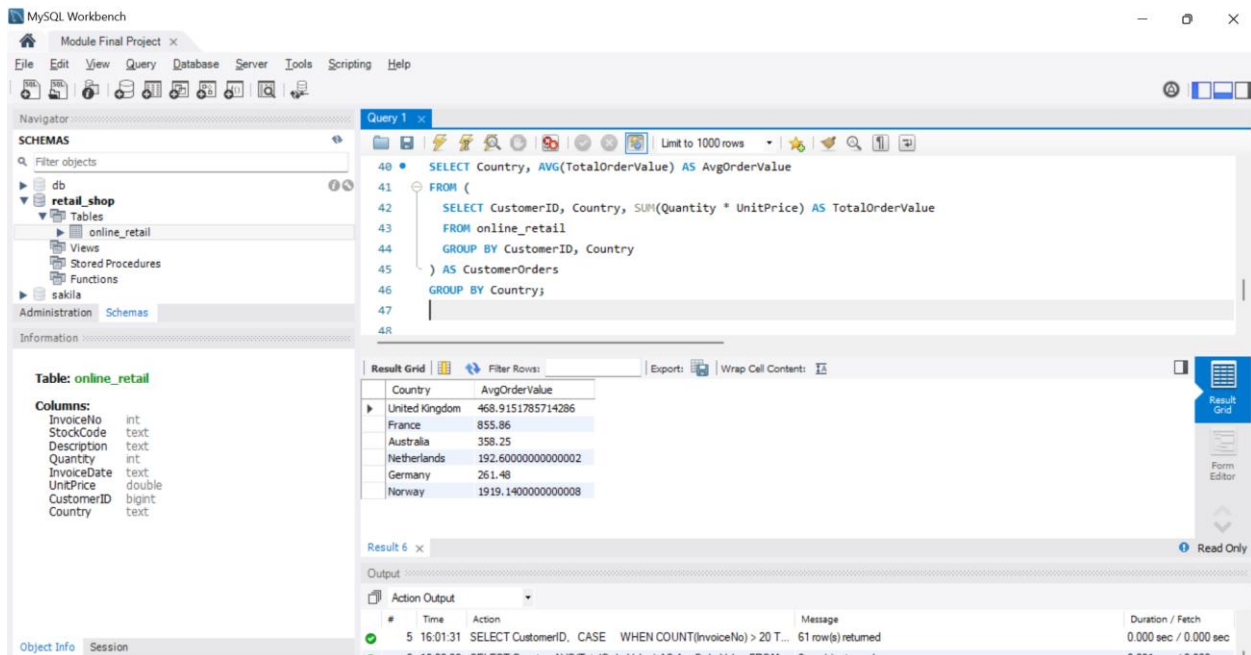
Query:

```
SELECT Country, AVG(TotalOrderValue) AS AvgOrderValue
FROM (
    SELECT CustomerID, Country, SUM(Quantity * UnitPrice)
    AS TotalOrderValue
    FROM online_retail
    GROUP BY CustomerID, Country
) AS CustomerOrders
GROUP BY Country;
```

Explanation:

This query calculates the average order value by country, helping to identify regions with the highest spending customers.

Output:



The screenshot displays the MySQL Workbench interface. The 'Query' tab shows the SQL query for calculating the average order value by country. The 'Result Grid' shows the output of the query, which includes the country and the average order value. The 'Table: online_retail' section lists the columns and their data types. The 'Output' section shows the execution details, including the time taken and the number of rows returned.

Country	AvgOrderValue
United Kingdom	468.9151785714286
France	855.86
Australia	358.25
Netherlands	192.60000000000002
Germany	261.48
Norway	1919.1400000000008

Table: online_retail

Columns:

- InvoiceNo: int
- StockCode: text
- Description: text
- Quantity: int
- InvoiceDate: text
- UnitPrice: double
- CustomerID: bigint
- Country: text

Result 6 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
5	16:01:31	SELECT CustomerID, CASE WHEN COUNT(InvoiceNo) > 20 T...	61 row(s) returned	0.000 sec / 0.000 sec

Customer Churn Analysis

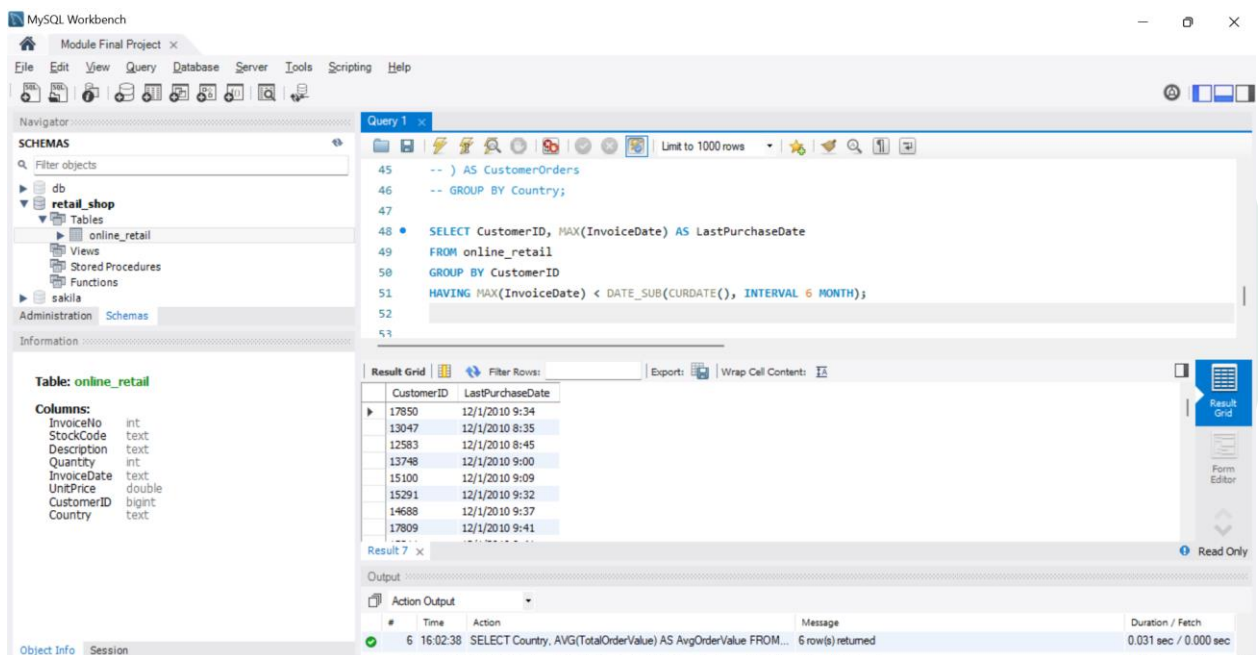
Query:

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

Explanation:

Customers who haven't made a purchase in the last six months are identified, providing a basis for churn analysis.

Output:



The screenshot displays the MySQL Workbench interface. On the left, the 'Navigator' pane shows the database schema with 'db' containing 'retail_shop' and 'sakila'. The 'retail_shop' database is expanded, showing 'online_retail' as a table. The 'Table: online_retail' section lists columns: InvoiceNo (int), StockCode (text), Description (text), Quantity (int), InvoiceDate (text), UnitPrice (double), CustomerID (bigint), and Country (text). The main editor shows a SQL query (Query 1) that selects CustomerID and the maximum InvoiceDate (aliased as LastPurchaseDate) from the online_retail table, grouped by CustomerID, and filtered to show only those customers whose last purchase date is more than 6 months ago. The 'Result Grid' pane shows the output of the query, displaying a list of CustomerID and LastPurchaseDate. The 'Output' pane at the bottom shows the execution message: 'SELECT Country, AVG(TotalOrderValue) AS AvgOrderValue FROM... 6 row(s) returned'.

CustomerID	LastPurchaseDate
17850	12/1/2010 9:34
13047	12/1/2010 8:35
12583	12/1/2010 8:45
13748	12/1/2010 9:00
15100	12/1/2010 9:09
15291	12/1/2010 9:32
14688	12/1/2010 9:37
17809	12/1/2010 9:41

Product Affinity Analysis

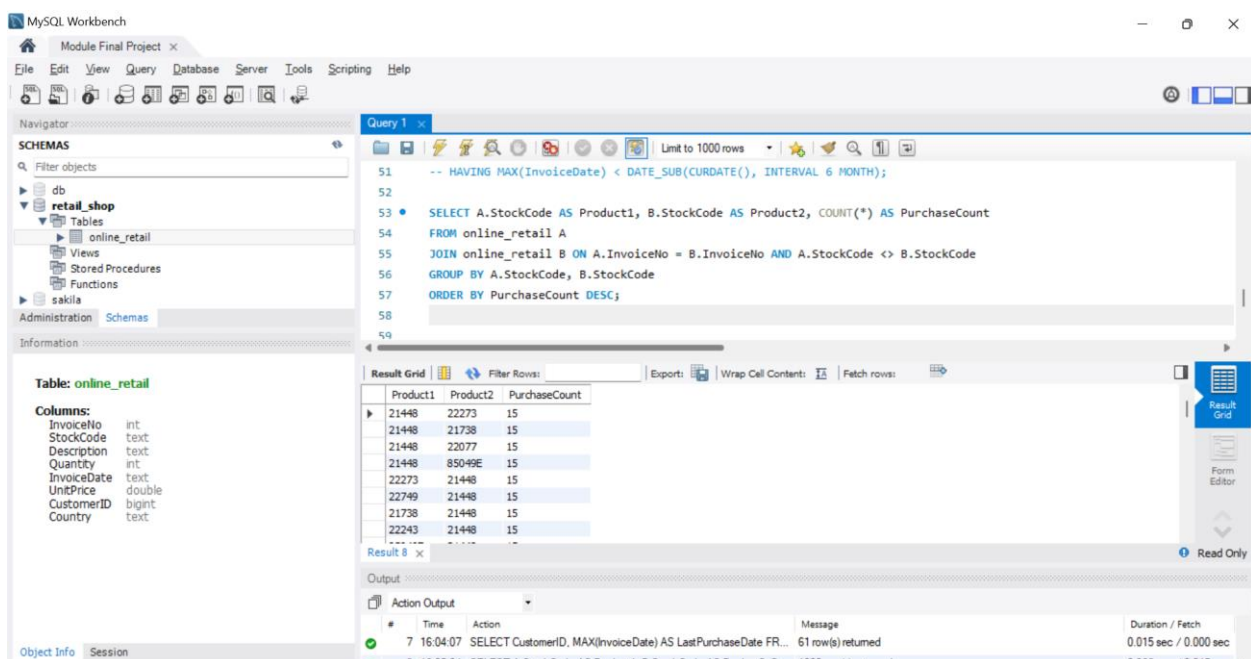
Query:

```
SELECT A.StockCode AS Product1, B.StockCode AS Product2,  
COUNT(*) AS PurchaseCount  
FROM online_retail A  
JOIN online_retail B ON A.InvoiceNo = B.InvoiceNo AND  
A.StockCode <> B.StockCode  
GROUP BY A.StockCode, B.StockCode  
ORDER BY PurchaseCount DESC;
```

Explanation:

This analysis identifies products frequently purchased together, useful for cross-selling strategies.

Output:



The screenshot displays the MySQL Workbench interface. The 'Query 1' window shows the following SQL query:

```
-- HAVING MAX(InvoiceDate) < DATE_SUB(CURDATE(), INTERVAL 6 MONTH);  
52  
53 * SELECT A.StockCode AS Product1, B.StockCode AS Product2, COUNT(*) AS PurchaseCount  
54 FROM online_retail A  
55 JOIN online_retail B ON A.InvoiceNo = B.InvoiceNo AND A.StockCode <> B.StockCode  
56 GROUP BY A.StockCode, B.StockCode  
57 ORDER BY PurchaseCount DESC;  
58  
59
```

The 'Result Grid' shows the following data:

Product1	Product2	PurchaseCount
21448	22273	15
21448	21738	15
21448	22077	15
21448	85049E	15
22273	21448	15
22749	21448	15
21738	21448	15
22243	21448	15

The 'Table: online_retail' structure is shown on the left:

Columns:	
InvoiceNo	int
StockCode	text
Description	text
Quantity	int
InvoiceDate	text
UnitPrice	double
CustomerID	bigint
Country	text

The 'Output' window shows the following message:

```
7 16:04:07 SELECT CustomerID, MAX(InvoiceDate) AS LastPurchaseDate FR... 61 row(s) returned  
Duration / Fetch 0.015 sec / 0.000 sec
```

Time-based Analysis

Query:

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

Explanation:

Monthly sales trends are explored to understand seasonal or temporal patterns in customer behavior.

Output:

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view containing 'db', 'retail_shop', and 'sakila'. The 'retail_shop' schema is expanded, showing 'online_retail' as a table. The 'Table: online_retail' section lists columns: InvoiceNo (int), StockCode (text), Description (text), Quantity (int), InvoiceDate (datetime), UnitPrice (double), CustomerID (bigint), and Country (text). The main query editor shows the following SQL query:

```
65 SELECT YEAR(InvoiceDate) AS Year, MONTH(InvoiceDate) AS Month, SUM(Quantity * UnitPrice) AS MonthlySales  
66 FROM online_retail  
67 GROUP BY Year, Month  
68 ORDER BY Year, Month;  
69  
70  
71
```

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

Year	Month	MonthlySales
2010	12	29846.5800000000133

The 'Action Output' panel at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
2	16:38:58	UPDATE online_retail SET InvoiceDate = STR_TO_DATE(Invoice...	Error Code: 1411. Incorrect datetime value: '2010-12-01 08:26:00' f...	0.000 sec
3	16:39:01	UPDATE online_retail SET InvoiceDate = STR_TO_DATE(Invoice...	Error Code: 1411. Incorrect datetime value: '2010-12-01 08:26:00' f...	0.000 sec
4	16:39:02	UPDATE online_retail SET InvoiceDate = STR_TO_DATE(Invoice...	Error Code: 1411. Incorrect datetime value: '2010-12-01 08:26:00' f...	0.000 sec
5	16:39:12	ALTER TABLE online_retail MODIFY COLUMN InvoiceDate DATE...	1316 row(s) affected Records: 1316 Duplicates: 0 Warnings: 0	0.422 sec
6	16:39:35	SELECT YEAR(InvoiceDate) AS Year, MONTH(InvoiceDate) AS M...	1 row(s) returned	0.000 sec / 0.000 sec

Conclusion

The SQL queries executed in this project provided valuable insights into customer segmentation, purchase behaviors, and potential strategies for improving customer engagement and retention. By leveraging these insights, businesses can better tailor their marketing efforts and improve overall business performance.