REPORT ABOUT PROJECT

Muhammad Ali

NUST Campus Gilgit

Contents

About Me

Name: Muhammad Ali

Project Title: Online Retail Segmentation

GitHub Profile Link : https://github.com/Muhammad-Ali-56

NUST
Defining Futures

ONLINE RETAIL SEGMENTATION REPORT

Introduction

In this report, we analyze a dataset containing customer purchase information. The dataset provides insights into customer behaviors, purchase frequencies, order values, and product preferences.

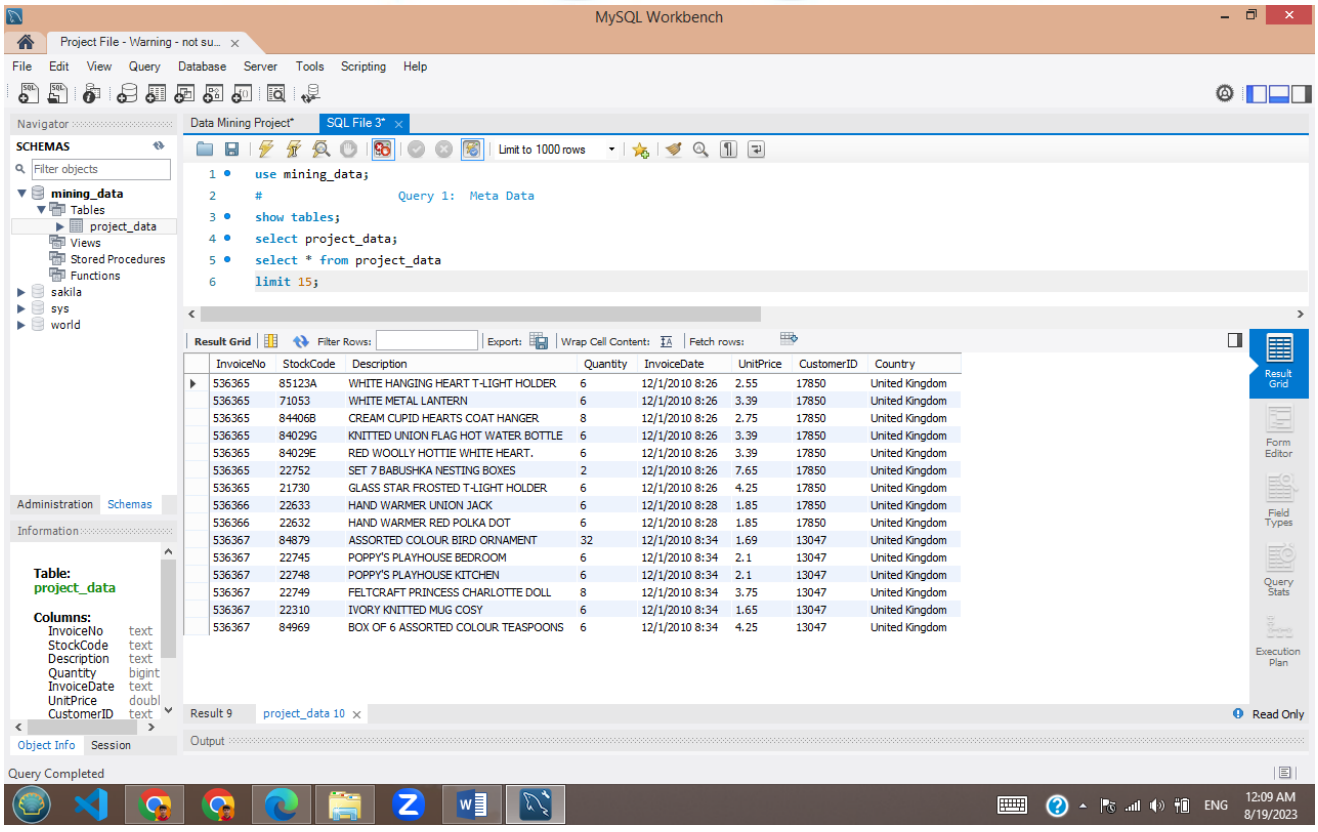
BEGINNER QUERIES

Query 1: Meta Data

```
use mining_data;

#                                   Query 1:  Meta Data

show tables;

select project_data;

select * from project_data;
```

We start by exploring the structure of the dataset. The show tables command reveals the tables present in the "mining_data" database. The subsequent queries retrieve and display all rows from the "project_data" table, allowing us to understand the data format and content.

- The **SHOW TABLES** command is used to list all the tables present in the current database (**mining_data** in this case).
- This query provides a quick overview of the available tables in the database, giving insight into the structure of the dataset.

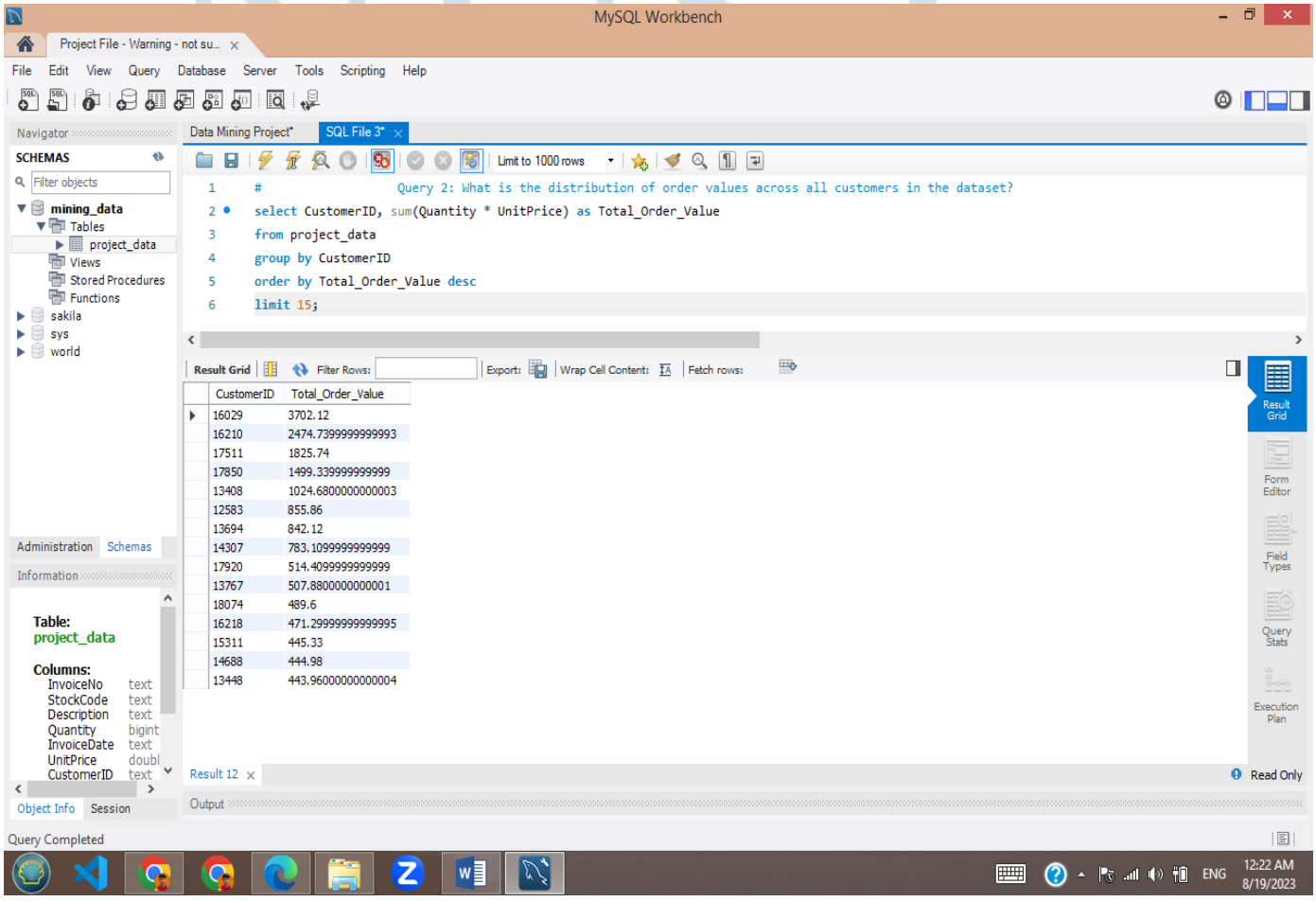


Query 2: Distribution of Order Values

```
# Query 2: What is the distribution of order values across all customers in the dataset?

select CustomerID, sum(Quantity * UnitPrice) as Total_Order_Value

from project_data

group by CustomerID

order by Total_Order_Value desc

limit 15;
```

This query examines the distribution of order values across customers. By calculating the total order value for each customer, we identify the top 15 customers with the highest order values. This insight can help identify the most valuable customers in terms of their purchasing power.

- The query groups the data by **CustomerID**.
- It calculates the total order value for each customer by multiplying **Quantity** with **UnitPrice** and summing the results.
- The **GROUP BY** clause ensures that the calculations are performed per customer.
- The **ORDER BY** clause arranges the results in **descending order** of total order value.
- The **LIMIT 15** clause restricts the output to the top **15** customers with the highest order values.

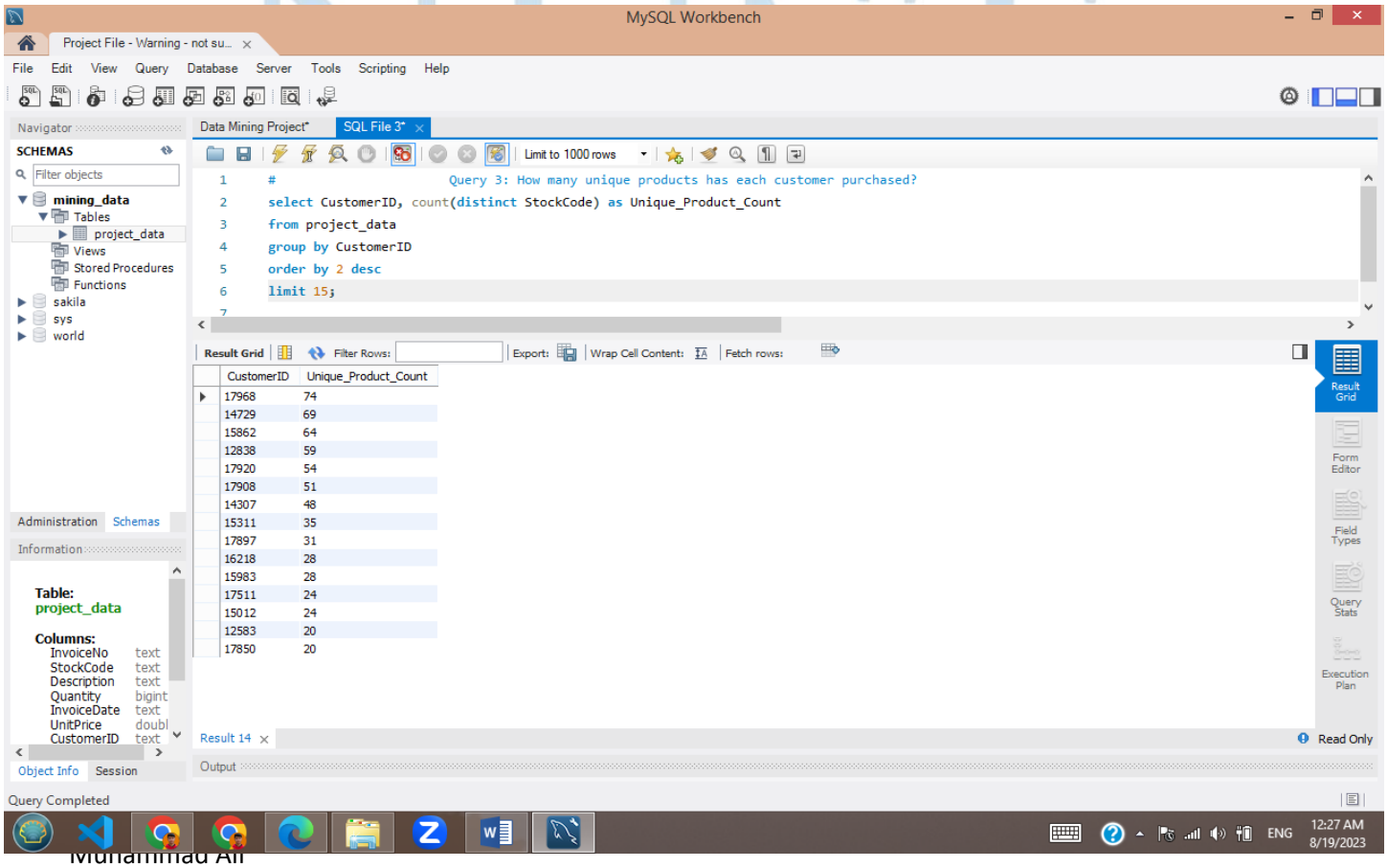


Query 3: Unique Products per Customer

```
#                                    Query 3: How many unique products has each
customer purchased?

select CustomerID, count(distinct StockCode) as Unique_Product_Count

from project_data

group by CustomerID

order by 2 desc;
```

Here, we investigate how many unique products each customer has purchased. The results provide an understanding of customers' variety in product preferences. Customers with a high count of unique products may exhibit diverse interests.

- Similar to the previous query, this query groups the data by **CustomerID**.
- It uses the **COUNT(DISTINCT StockCode)** function to count the number of unique products each customer has purchased.
- The results are **ordered** in **descending order** of unique product count
- The **2** in **ORDER BY** denotes **Unique_Product_count**

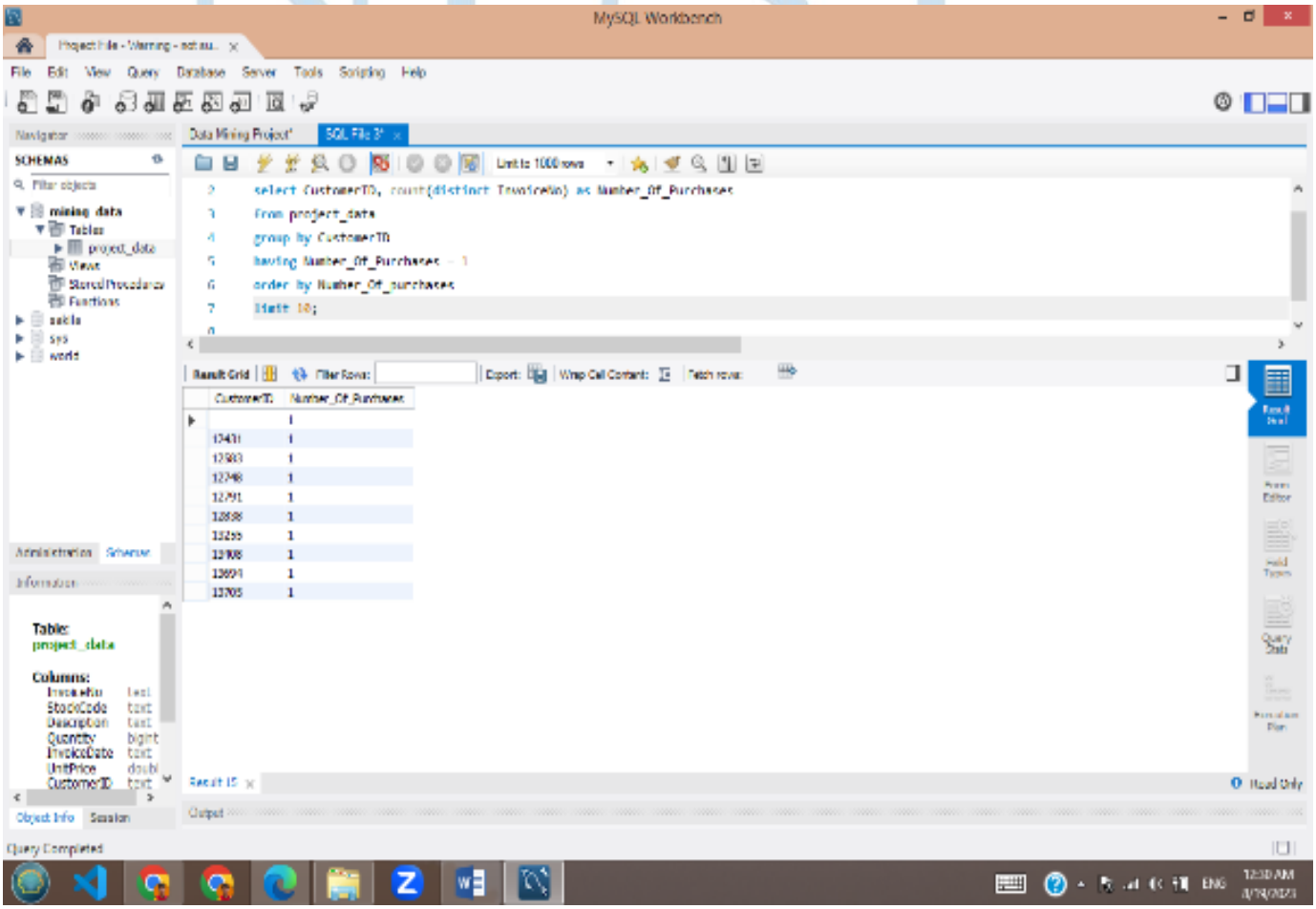


Query 4: Single Purchase Customers

```
# Query 4: Which customers have only made a single purchase from the company?

select CustomerID, count(distinct InvoiceNo) as Number_Of_Purchases

from project_data

group by CustomerID

having Number_Of_Purchases = 1

order by Number_Of_purchases

limit 10;
```

Identifying customers who have made only a single purchase can be crucial for targeted marketing efforts. This query lists customers who have made just one purchase, helping to identify potential areas for improvement in customer retention.

- The query groups the data by **CustomerID**.
- It uses the **COUNT(DISTINCT InvoiceNo)** function to count the number of distinct invoices (purchases) each customer has made.
- The **HAVING** clause filters the results to include only those customers who have made exactly **one purchase**.
- The results are **ordered by** the number of purchases and **limited to the top 10**.



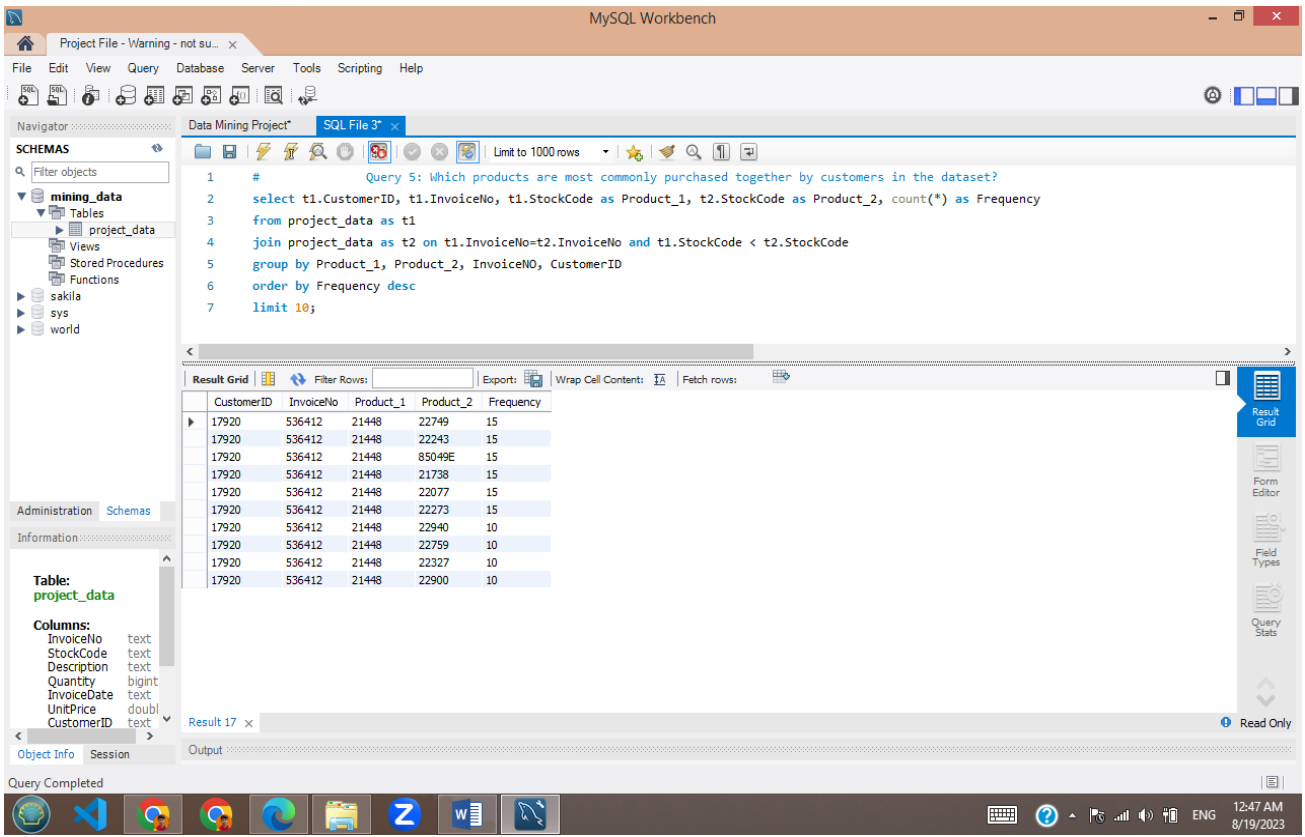
Query 5: Frequently Purchased Product Pairs

 # Query 5: Which products are most commonly purchased together by customers in the dataset?

```
select t1.CustomerID, t1.InvoiceNo, t1.StockCode as Product_1, t2.StockCode as Product_2, count(*) as Frequency
from project_data as t1
join project_data as t2 on t1.InvoiceNo=t2.InvoiceNo and t1.StockCode < t2.StockCode
group by Product_1, Product_2, InvoiceNO, CustomerID
order by Frequency desc
limit 10;
```

By analyzing products commonly purchased together, we can uncover valuable cross-selling opportunities. This query identifies pairs of products frequently purchased together by customers. Such insights can inform product bundling strategies.

- This query uses a **self-join** on the **project_data** table, creating two different aliases (**t1 and t2**) to reference the same table.
- The join condition ensures that the rows being joined have the same **InvoiceNo** but different **StockCode** values.
- The query groups the data by **Product_1**, **Product_2**, **InvoiceNo**, and **CustomerID**.
- The **COUNT(*)** function calculates the **frequency** of each product pair being purchased together.
- The **results** are **ordered** in **descending order** of **frequency** and **limited to the top 1**



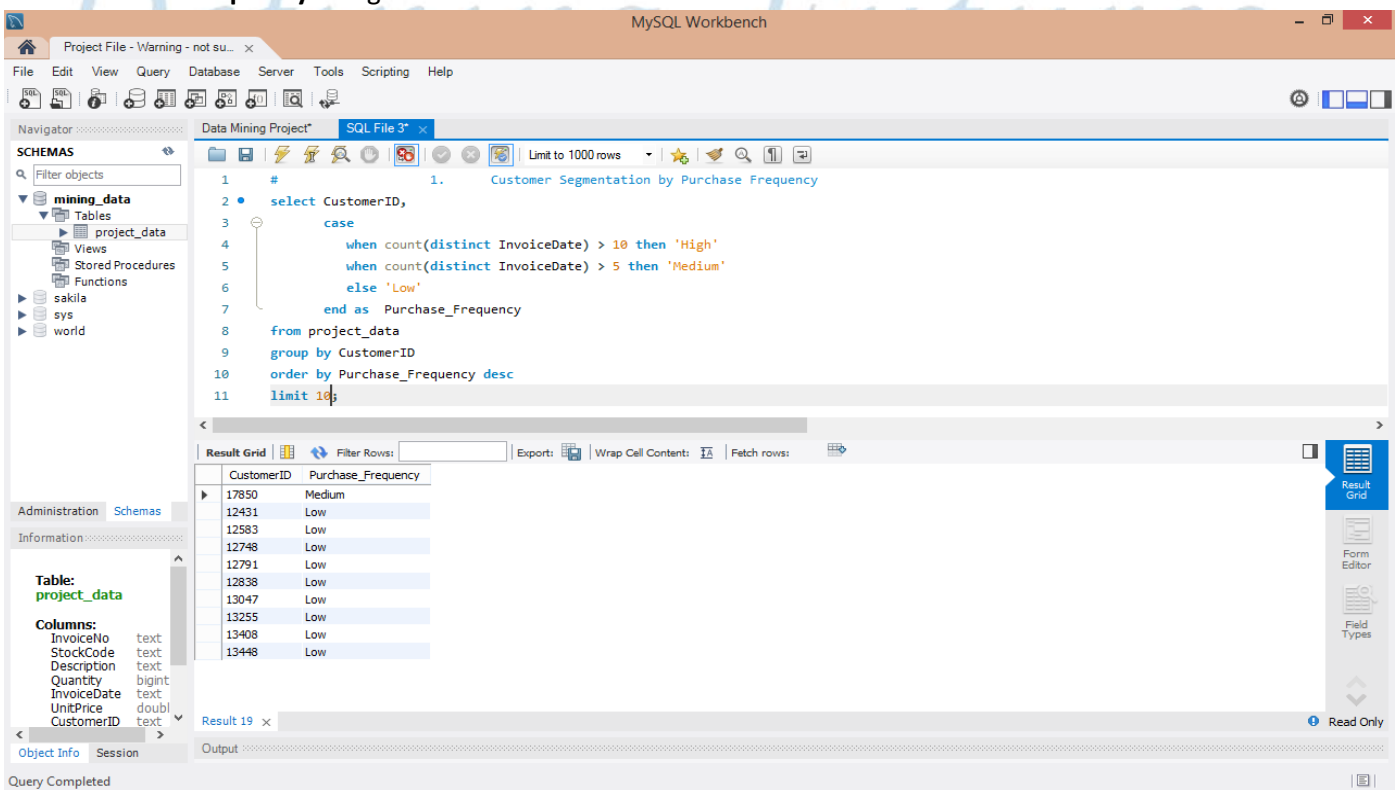
ADVANCED QUERIES

1. Customer Segmentation by Purchase Frequency

```
#                    1.    Customer Segmentation by Purchase Frequency
select CustomerID,
    case
      when count(distinct InvoiceDate) > 10 then 'High'
      when count(distinct InvoiceDate) > 5 then 'Medium'
      else 'Low'
    end as  Purchase_Frequency
from project_data
group by CustomerID
order by Purchase_Frequency desc
limit 10;
```

Segmenting customers based on their purchase frequency provides valuable insights into customer engagement. This query categorizes customers as "**High**," "**Medium**," or "**Low**" frequency based on their purchase history.

- Uses a **CASE** statement to categorize customers based on the number of distinct **InvoiceDate** values.
- Results are **ordered by** purchase **frequency**, showing which customers fall into **high**, **medium**, or **low frequency** categories.

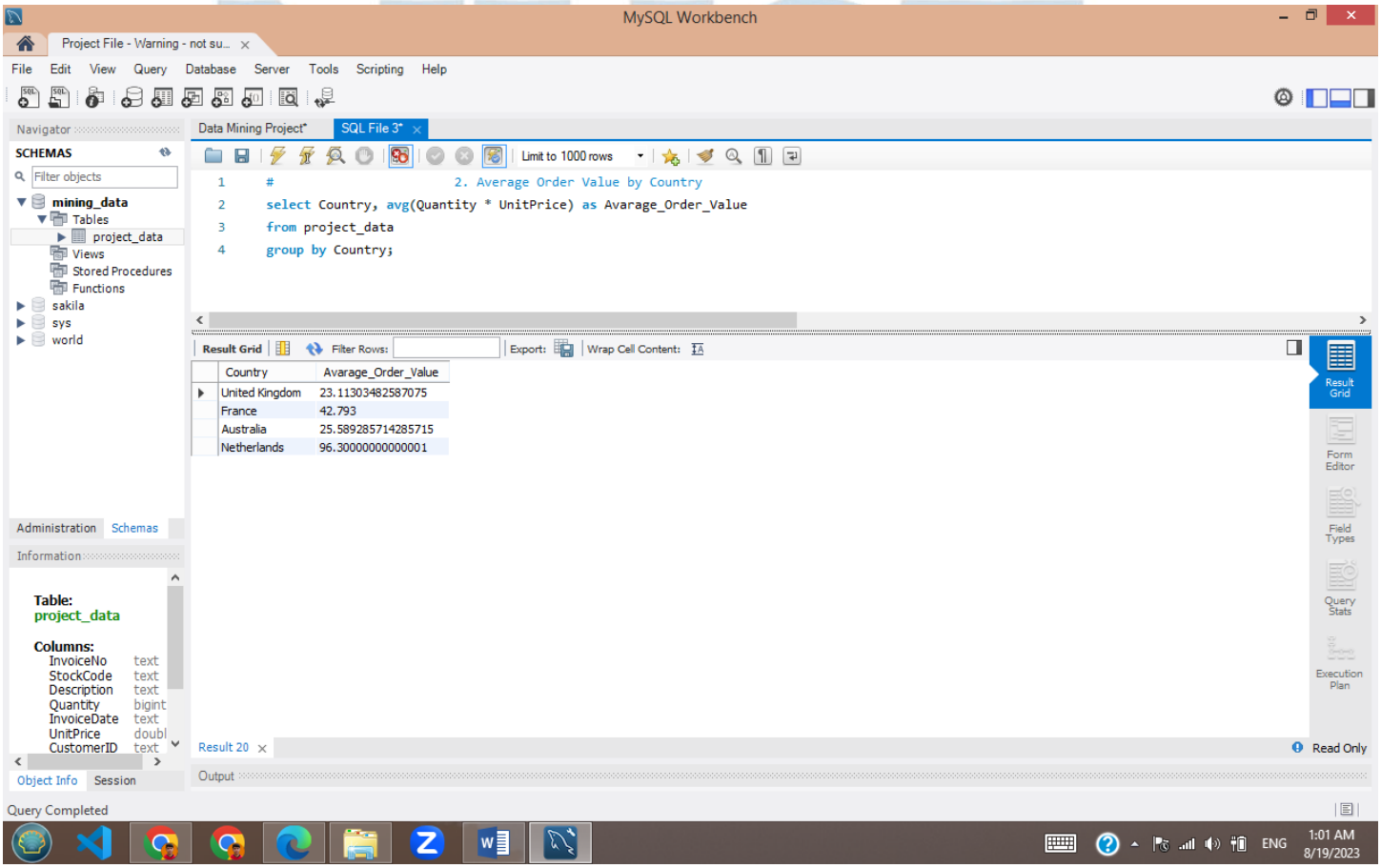


2. Average Order Value by Country

```
#                                    2. Average Order Value by Country

select Country, avg(Quantity * UnitPrice) as Avarage_Order_Value

from project_data

group by Country;
```

Understanding average order values by country can assist in tailoring marketing strategies to different regions. This query calculates the average order value for each country represented in the dataset.

- Groups the data by **Country**.
- Calculates the **average order value** by multiplying **Quantity** with **UnitPrice** and taking the **average.**
- Provides insights into the spending patterns of different countries.

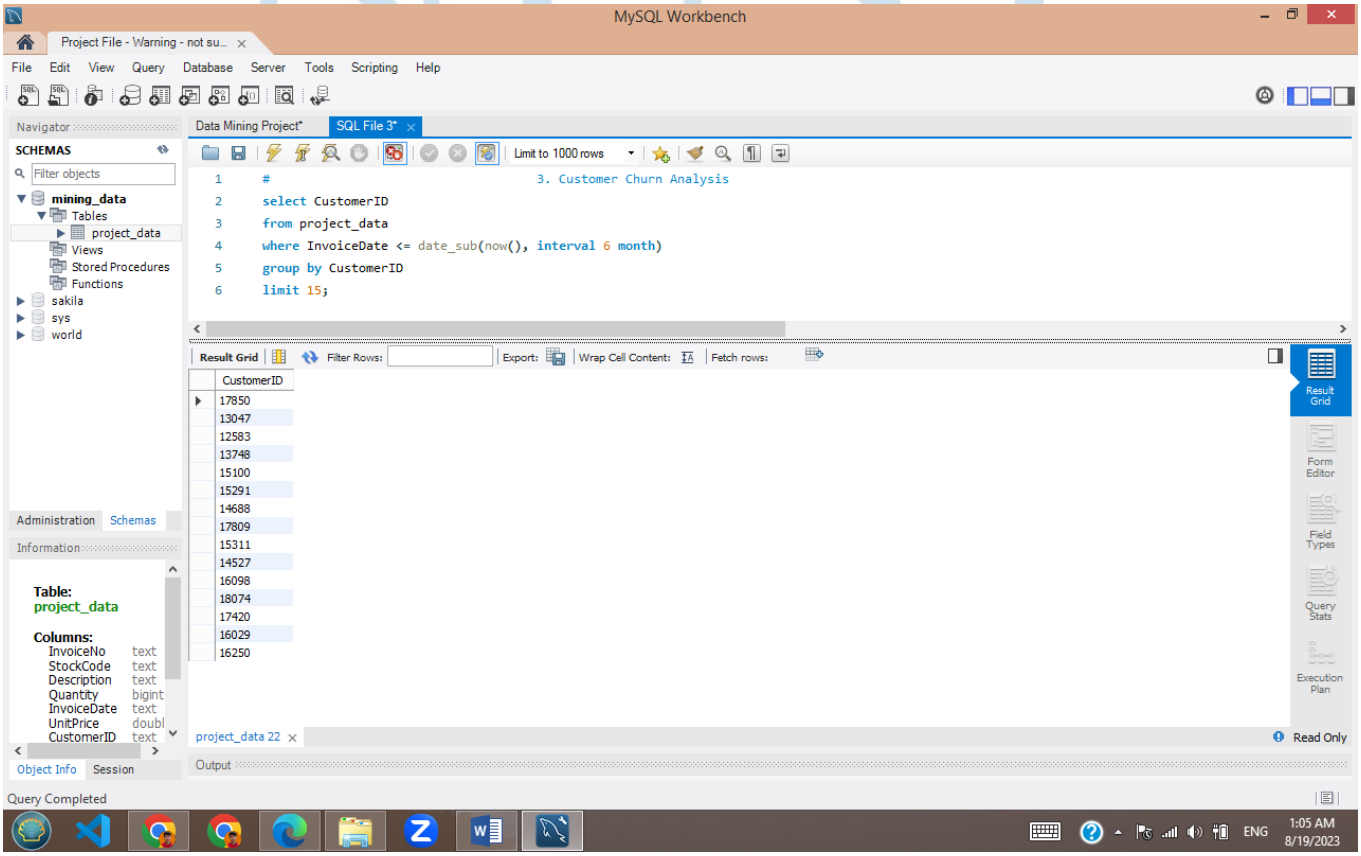


3. Customer Churn Analysis

```
#                                   3. Customer Churn Analysis

select CustomerID

from project_data

where InvoiceDate <= date_sub(now(), interval 6 month)

group by CustomerID

limit 15;
```

Customer churn analysis is essential for customer retention efforts. This query identifies customers who haven't made purchases in the last six months, potentially indicating churn.

- Filters the data to include only customers **who haven't made purchases** in the last 6 months.
- Groups the data by **CustomerID**.
- Identifies customers who might have churned (**stopped purchasing**) based on their recent activity.
- **Limiting up to 15** due to large Data.

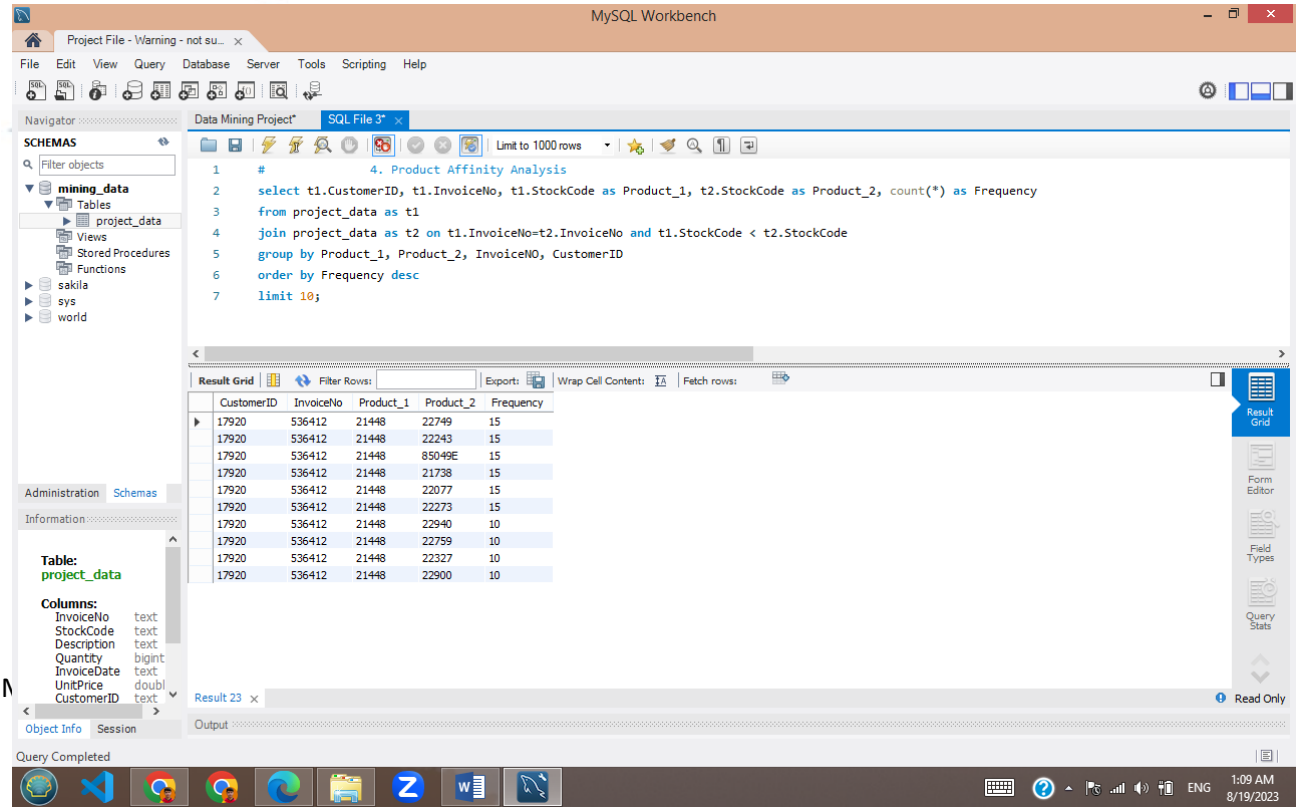


4. Product Affinity Analysis

```
#                                   4. Product Affinity Analysis

select t1.CustomerID, t1.InvoiceNo, t1.StockCode as Product_1, t2.StockCode as Product_2, count(*) as Frequency

from project_data as t1

join project_data as t2 on t1.InvoiceNo=t2.InvoiceNo and t1.StockCode < t2.StockCode

group by Product_1, Product_2, InvoiceNO, CustomerID

order by Frequency desc

limit 10;
```

Similar to Query 5, this query further explores product affinity by examining pairs of products commonly purchased together by customers.

- This query uses a **self-join** on the **project_data** table, creating two different aliases (**t1** and **t2**) to reference the same table.
- The join condition ensures that the rows being joined have the same **InvoiceNo** but different **StockCode** values.
- The query groups the data by **Product_1**, **Product_2**, **InvoiceNo**, and **CustomerID**.
- The **COUNT(*)** function calculates the frequency of each product pair being purchased together.
- The results are **ordered** in **descending order** of **frequency** and **limited** to the **top 10**

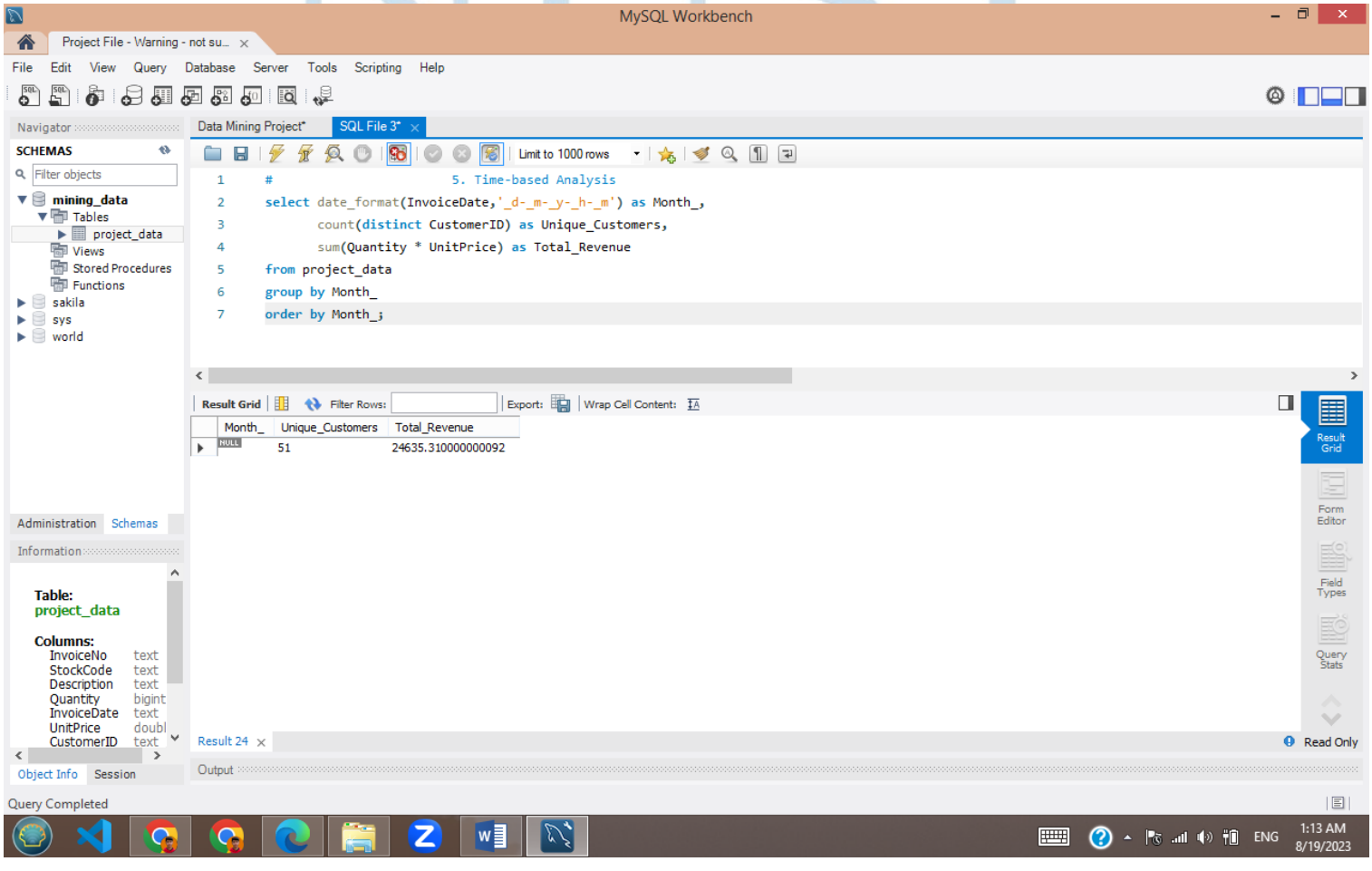


5. Time-based Analysis

```
#                                                    5. Time-based Analysis
select date_format(InvoiceDate,'_d-_m-_y-_h-_m') as Month_,
    count(distinct CustomerID) as Unique_Customers,
    sum(Quantity * UnitPrice) as Total_Revenue
from project_data
group by Month_
order by Month_;
```

Analyzing sales trends over time is crucial for understanding seasonality and making informed business decisions. This query presents a time-based analysis by grouping data by months and summarizing unique customer counts and total revenue for each month.

- Uses the **DATE_FORMAT** function to extract month, day, year, and time components from **InvoiceDate**.
- Groups the data by formatted **date**, providing insights into monthly trends in terms of **unique customers** and **total revenue**



CONCLUSION

The analysis of the dataset provides valuable insights into customer behavior, purchase patterns, and potential business opportunities. The various queries shed light on customer segmentation, product affinity, and time-based trends. These insights can guide marketing strategies, customer retention efforts, and product recommendations to enhance business success.

The End…