I519--Grad Students--Topic Investigation SUBMISSION

Topic Description: Installation and Usage of RDBMS in MySQL
Team Members: Bhargavi Thanneeru and Sohail Mohammed

Installation:

We successfully installed the MySQL server and Workbench by following tutorials on YouTube and referring to the MySQL documentation.

Challenges Faced:

- As a Mac user, configuring the path for the MySQL server posed an initial challenge. After attempting various solutions, I resolved the issue by appending the (mysql/bin) path to the .zshrc file. Additionally, I installed Microsoft Workbench and established a successful connection to the local instance within the Workbench.

- Encountered restrictions while importing CSV data into MySQL. To address this, I created a configuration file at the specified path (/usr/etc/."my.conf") and made necessary updates based on requirements. Notably, to mitigate server shutdowns caused by prolonged queries, I modified the configuration file to accommodate such constraints.

Data Pre - Preprocessing:

We obtained a denormalized dataset, specifically the "Sample Sales Data," from Kaggle, accessible through the following
link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data
This dataset comprises 25 columns and contains 2823 records.

Data Cleaning:

In the initial phase of data preprocessing, we pruned the dataset to focus on essential columns for our targeted data insights and analysis. The following columns were deemed non-essential and subsequently removed from the original dataset:

OrderLineNumber
Phone
AddressLine1
AddressLine2
Territory
State

By eliminating these columns, we aimed to streamline the dataset and enhance its relevance to the specific analytical goals we have set. This process ensures that the subsequent analysis is conducted on a more refined and purposeful dataset.

Schema of the original data

```
CREATE TABLE orders (
OrderId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
OrderNumber VARCHAR(20),
QuantityOrdered INT,
ItemPrice DECIMAL(8,2),
OrderLineNumber INT,
Sales DECIMAL(8,2),
OrderDate DATE,
Status VARCHAR(15),
Qtr_Id INT,
Month_Id INT,
Year_Id INT,
ProductCategory VARCHAR(25),
ProductPrice DECIMAL(8,2),
ProductCode VARCHAR(25),
CustomerName VARCHAR(65),
Phone VARCHAR(20),
AddressLine1 VARCHAR(60),
AddressLine2 VARCHAR(60),
City VARCHAR(25),
State VARCHAR(20),
PostalCode VARCHAR(15),
Country VARCHAR(25),
Territory VARCHAR(20),
Sales_Contact_LastName VARCHAR(30),
Sales_Contact_FirstName VARCHAR(20),
DealSize VARCHAR(10)
);
```

During the data cleaning and normalization process, we addressed values containing special characters by removing them, thereby ensuring a cleaner dataset for analysis.

Normalization involved breaking down the dataset into four distinct tables:

1. Sales Table: Contains information related to sales transactions.

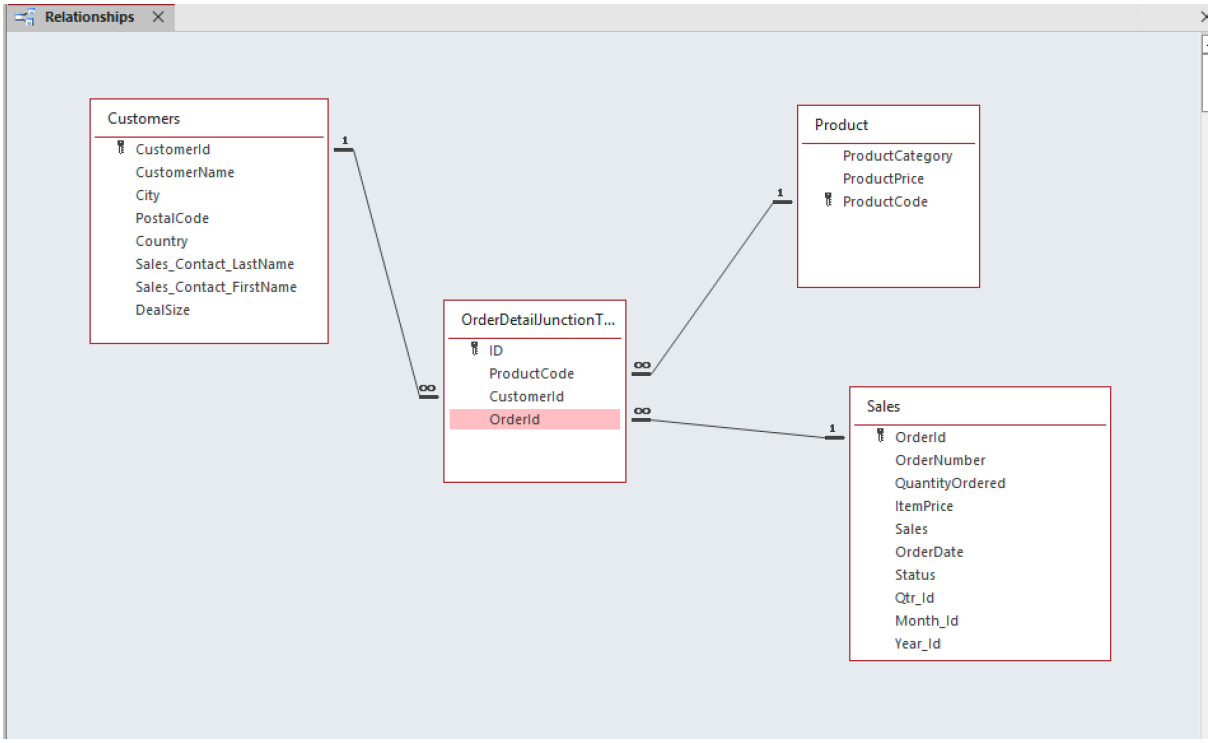
2. Product Table: Contains details about products.

3. Customer Table: Contains information about customers.

4. OrderDetailJunction Table: Serves as a junction table connecting Sales, Product, and Customer tables, facilitating the relationships between these entities.

This normalization approach enhances data organization, reduces redundancy, and establishes relationships between tables, providing a structured and efficient foundation for analysis and insights.

The below is the Relationship diagram:



Customer Table

```
CREATE TABLE customer(
  CustomerId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  CustomerName VARCHAR(65),
  City VARCHAR(25),
  PostalCode VARCHAR(15),
  Country VARCHAR(25),
  Sales_Contact_LastName VARCHAR(30),
  Sales_Contact_FirstName VARCHAR(20),
  DealSize VARCHAR(10));
```

Sales Table

```
CREATE TABLE sales (
  OrderId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  OrderNumber VARCHAR(20),
  QuantityOrdered INT,
  ItemPrice DECIMAL(8,2),
  Sales DECIMAL(8,2),
  OrderDate DATE,
  Status VARCHAR(15),
```

```
  Qtr_Id INT,
  Month_Id INT,
  Year_Id INT);
```

Product Table

```
CREATE TABLE product(
  ProductCode VARCHAR(25) PRIMARY KEY,
  ProductCategory VARCHAR(25),
  ProductPrice DECIMAL(8,2));
```

Junction Table

```
CREATE TABLE orderDetailJunction (
  OrderDetailId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  ProductCode VARCHAR(25),
  CustomerId INT,
  OrderId INT,

  FOREIGN KEY (ProductCode) REFERENCES product(ProductCode),
  FOREIGN KEY (CustomerId) REFERENCES customer(CustomerId),
  FOREIGN KEY (OrderId) REFERENCES sales(OrderId)
);
```

Duplicate Removal:

- Product Table: Duplicates were removed from the product table, resulting in 109 unique records.
- Customer Table: Duplicates were eliminated from the customer table, resulting in 254 unique records.

preprocess.ipynb | sales_data.csv

Code | Python 3 (ipykernel)

[11]:
```
# !pip install mysql-connector-python
```

[2]:
```
import pandas as pd
```

[38]:
```
df = pd.read_csv('sales_data.csv')
```

[39]:
```
# 2/24/2003 0:00
df['OrderDate'] = pd.to_datetime(df['OrderDate'], format='%m/%d/%Y %H:%M').dt.strftime('%Y-%m-%d')
```

[40]:
```
df['OrderDate']
```

[40]:
```
0       2003-02-24
1       2003-05-07
2       2003-07-01
3       2003-08-25
4       2003-10-10
           ...
2818    2004-12-02
2819    2005-01-31
2820    2005-03-01
2821    2005-03-28
2822    2005-05-06
Name: OrderDate, Length: 2823, dtype: object
```

[89]:
```
len(df)
```

[89]:
```
2823
```

[52]:
```
import pandas as pd
import mysql.connector

# Replace 'your_database', 'your_username', 'your_password', 'your_host' with your actual MySQL database details
database_name = 'orders_db'
username = 'root'
password = 'password'
host = 'localhost'

# Connect to MySQL
connection = mysql.connector.connect(
```

preprocess.ipynb | sales_data.csv

Code | Python 3 (ipykernel)

[52]:
```
import pandas as pd
import mysql.connector

# Replace 'your_database', 'your_username', 'your_password', 'your_host' with your actual MySQL database details
database_name = 'orders_db'
username = 'root'
password = 'password'
host = 'localhost'

# Connect to MySQL
connection = mysql.connector.connect(
    host=host,
    user=username,
    password=password,
    database=database_name
)


# Insert data into MySQL
cursor = connection.cursor()
for index, row in df.iterrows():
    cursor.execute("""
        INSERT INTO orders (OrderNumber, QuantityOrdered, ItemPrice, OrderLineNumber, Sales, OrderDate, Status, Qtr_Id, Month_Id, Year_Id, ProductCategory, Pr
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, tuple(row))

# Commit the transaction and close the connection
connection.commit()
cursor.close()
connection.close()
```

[49]:
```
# Assuming df is your DataFrame
customer_df = df[['CustomerName', 'City', 'PostalCode', 'Country', 'Sales_Contact_LastName', 'Sales_Contact_FirstName', 'DealSize']]
```

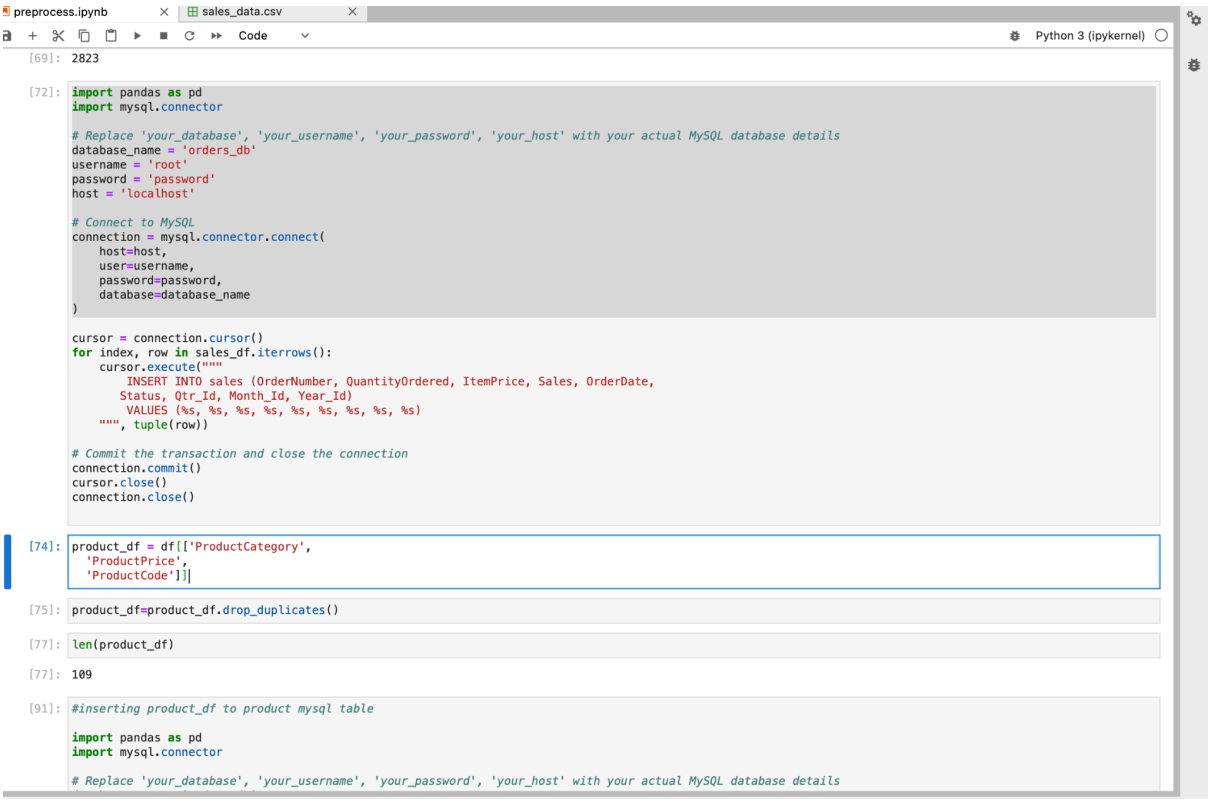
[56]:
```
customer_df = customer_df.drop_duplicates()
```

[58]:
```
len(customer_df)
```

[58]:
```
254
```

[54]:
```
customer_df.columns
```

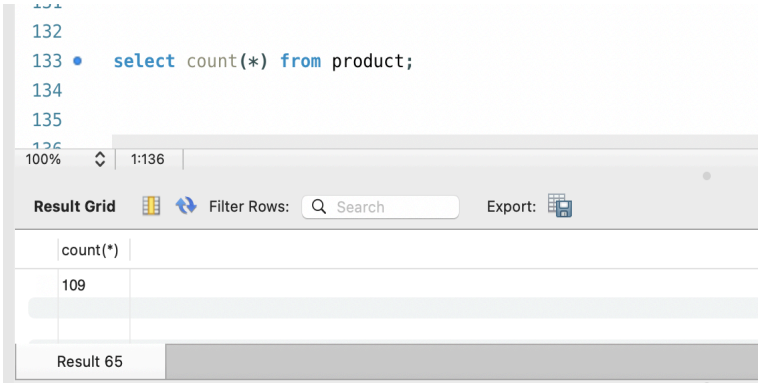
[54]:
```
Index(['CustomerName', 'City', 'PostalCode', 'Country',
       'Sales_Contact_LastName', 'Sales_Contact_FirstName', 'DealSize'],
      dtype='object')
```



Junction Table Creation:

- A junction table named "OrderDetailJunction" was established with "OrderDetailId" as the primary key and "ProductCode," "CustomerId," and "OrderId" as foreign keys referencing their respective tables.

MySQL Table Insertion: The pre-processed and refined DataFrame was successfully inserted into the corresponding MySQL tables. These steps have ensured a clean, structured, and normalized dataset, providing a solid foundation for subsequent analysis and queries.



```
136
137 •   select count(*) from customer;
138
```

100% 1:142

Result Grid | Filter Rows: Search | Export:

count(*)
254

Result 66

SQL Queries

Query 1 - Total sales by product category

```
SELECT ProductCategory, SUM(Sales) AS TotalSales
FROM orders
GROUP BY ProductCategory
ORDER BY TotalSales DESC;
```

```
142
143     -- Query 1 : Get total sales by product category
144
145 •   SELECT ProductCategory, SUM(Sales) AS TotalSales
146     FROM orders
147     GROUP BY ProductCategory
148     ORDER BY TotalSales DESC;
149
150
151
152
```

100% 1:149

Result Grid | Filter Rows: Search | Export:

ProductCategory	TotalSales
Classic Cars	3919615.66
Vintage Cars	1903150.84
Motorcycles	1166388.34
Trucks and Buses	1127789.84
Planes	975003.57
Ships	714437.13
Trains	226243.47

Query 2: Total quantity sold of each product

```
SELECT ProductCode, SUM(QuantityOrdered) AS TotalQty
FROM orders
GROUP BY ProductCode
ORDER BY TotalQty DESC;
```

```
154
155     -- Query 2: Get total quantity sold of each product
156
157 •   SELECT ProductCode, SUM(QuantityOrdered) AS TotalQty
158     FROM orders
159     GROUP BY ProductCode
160     ORDER BY TotalQty DESC;
```

100% 1:163

Result Grid Filter Rows: Search Export:

ProductCode	TotalQty
S18_3232	1774
S24_3856	1052
S18_4600	1031
S700_4002	1029
S12_4473	1024
S24_3949	1008
S18_1097	999
S50_1341	999
S18_2432	998
S18_1342	997
S18_3856	997
S24_2300	996
S18_2319	993
S18_2949	991
S700_2610	990
S24_2840	983
S50_1392	979
S24_1444	976

Query 3: Top 5 customers by total sales amount

```
SELECT c.CustomerName, SUM(o.Sales) AS TotalSales
FROM orders o
JOIN customer c ON o.CustomerId = c.CustomerId
GROUP BY c.CustomerId
ORDER BY TotalSales DESC
LIMIT 5;
```

```
165    -- Query 3: Get top 5 customers by total sales amount
166
167    SELECT c.CustomerName, SUM(o.Sales) AS TotalSales
168    FROM orders o
169    JOIN customer c ON o.CustomerId = c.CustomerId
170    GROUP BY c.CustomerId
171    ORDER BY TotalSales DESC
172    LIMIT 5;
173
```

100% 9:172

Result Grid Filter Rows: Search Export: Fetch rows:

CustomerName	TotalSales
Euro Shopping Channel	912294.11
Mini Gifts Distributors Ltd.	654858.06
Australian Collectors, Co.	200995.41
Muscle Machine Inc	197736.94
La Rochelle Gifts	180124.90

Query 4: Total sales by year

```
SELECT Year_Id, SUM(Sales) AS TotalSales
FROM orders
GROUP BY Year_Id;
```

```
173
174    -- Query 4: Get total sales by year
175
176    SELECT Year_Id, SUM(Sales) AS TotalSales
177    FROM orders
178    GROUP BY Year_Id;
179
```

100% 18:178

Result Grid Filter Rows: Search Export:

Year_Id	TotalSales
2003	3516979.54
2004	4724162.60
2005	1791486.71

Query 5: Order status breakdown

```
SELECT Status, COUNT(*) AS OrderCount
FROM orders
GROUP BY Status;
```

```
181
182 •   SELECT Status, COUNT(*) AS OrderCount
183     FROM orders
184     GROUP BY Status;
185
```

100% 17:184

Result Grid | Filter Rows: Search | Export:

Status	OrderCount
Shipped	2617
Disputed	14
In Process	41
Cancelled	60
On Hold	44
Resolved	47

Query 6: Best selling products

```
SELECT ProductCategory, ProductCode, SUM(QuantityOrdered*ItemPrice) AS
TotalSales
FROM orders
GROUP BY ProductCode, ProductCategory
ORDER BY TotalSales DESC
LIMIT 3;
```

```
188 •   SELECT ProductCategory, ProductCode, SUM(QuantityOrdered*ItemPrice) AS TotalSales
189     FROM orders
190     GROUP BY ProductCode, ProductCategory
191     ORDER BY TotalSales DESC
192     LIMIT 3;
193
```

100% 9:192

Result Grid | Filter Rows: Search | Export: | Fetch rows:

ProductCategory	ProductCode	TotalSales
Classic Cars	S18_3232	176026.63
Classic Cars	S24_3856	103489.89
Trucks and Buses	S18_4600	101835.00

Query 7: Join orders and customers table to get sales by customer city

```
SELECT c.City, SUM(o.Sales) AS TotalSales
FROM orders o
JOIN customer c ON o.CustomerId = c.CustomerId
GROUP BY c.City
ORDER BY TotalSales DESC;
```

```
196 •  SELECT c.City, SUM(o.Sales) AS TotalSales
197    FROM orders o
198    JOIN customer c ON o.CustomerId = c.CustomerId
199    GROUP BY c.City
200    ORDER BY TotalSales DESC;
201
```

100% 26:200

Result Grid Filter Rows: Search Export:

City	TotalSales
Madrid	1082551.44
San Rafael	654858.06
NYC	560787.77
Singapore	288488.41
Paris	268944.68
San Francisco	224358.68
New Bedford	207874.86
Nantes	204304.86
Melbourne	200995.41
Brickhaven	165255.20
San Jose	160010.27
Manchester	157807.81
Boston	154069.66
North Sydney	153996.13
Chatswood	151570.98
Philadelphia	151189.13
Salzburg	149798.63

Query 8: Order status percentage breakdown

```
SELECT Status, COUNT(*) / (SELECT COUNT(*) FROM orders) * 100 AS Percent
FROM orders
GROUP BY Status;
```

```
203    -- Query 8: Get order status percentage breakdown
204 •  SELECT Status, COUNT(*) / (SELECT COUNT(*) FROM orders) * 100 AS Percent
205    FROM orders
206    GROUP BY Status;
207
208
```

100% 17:206

Result Grid Filter Rows: Search Export:

Status	Percent
Shipped	92.7028
Disputed	0.4959
In Process	1.4524
Cancelled	2.1254
On Hold	1.5586
Resolved	1.6649

Query 9: top 5 customers by average order amount

```
SELECT c.CustomerName, AVG(o.Sales) AS AvgOrderAmount
```

```
FROM orders o
JOIN customer c ON o.CustomerId = c.CustomerId
GROUP BY c.CustomerId
ORDER BY AvgOrderAmount DESC
LIMIT 5;
```

```
209    -- Query 9: Get top 5 customers by average order amount
210    SELECT c.CustomerName, AVG(o.Sales) AS AvgOrderAmount
211    FROM orders o
212    JOIN customer c ON o.CustomerId = c.CustomerId
213    GROUP BY c.CustomerId
214    ORDER BY AvgOrderAmount DESC
215    LIMIT 5;
```

100% 9:215

Result Grid Filter Rows: Search Export: Fetch rows:

CustomerName	AvgOrderAmount
Super Scale Inc.	4674.827647
Mini Caravy	4233.604211
La Corne D'abondance, Co.	4226.246957
Royale Belge	4180.012500
Muscle Machine Inc	4119.519583

Query 10: Top 5 customers by average order amount:

```
SELECT c.CustomerName, AVG(o.Sales) AS AvgOrderAmount
FROM orders o
JOIN customer c ON o.CustomerId = c.CustomerId
GROUP BY c.CustomerId
ORDER BY AvgOrderAmount DESC
LIMIT 5;
```

```
217    -- Query 10: Get top 5 customers by average order amount:
218    SELECT c.CustomerName, AVG(o.Sales) AS AvgOrderAmount
219    FROM orders o
220    JOIN customer c ON o.CustomerId = c.CustomerId
221    GROUP BY c.CustomerId
222    ORDER BY AvgOrderAmount DESC
223    LIMIT 5;
224
```

100% 9:223

Result Grid Filter Rows: Search Export: Fetch rows:

CustomerName	AvgOrderAmount
Super Scale Inc.	4674.827647
Mini Caravy	4233.604211
La Corne D'abondance, Co.	4226.246957
Royale Belge	4180.012500
Muscle Machine Inc	4119.519583

Query 11: Sales by month and year

```
SELECT YEAR(OrderDate) AS Year, MONTH(OrderDate) AS Month, SUM(Sales) AS
TotalSales
FROM orders
GROUP BY Year, Month
ORDER BY Year, Month;
```

```
225    -- Query 11: Get sales by month and year
226
227 •  SELECT YEAR(OrderDate) AS Year, MONTH(OrderDate) AS Month, SUM(Sales) AS TotalSales
228    FROM orders
229    GROUP BY Year, Month
230    ORDER BY Year, Month;
231
```

100% 22:230

Result Grid Filter Rows: Search Export:

Year	Month	TotalSales
2003	1	129753.60
2003	2	140836.19
2003	3	174504.90
2003	4	201609.55
2003	5	192673.11
2003	6	168082.56
2003	7	187731.88
2003	8	197809.30
2003	9	263973.36
2003	10	568290.97
2003	11	1029837....
2003	12	261876.46
2004	1	316577.42
2004	2	311419.53
2004	3	205733.73
2004	4	206148.12
2004	5	273438.39
2004	6	286674.22

Query 12: Get moving annual total sales by year

```
SELECT
   Year_Id AS Year,
   SUM(Sales) AS AnnualSales,
   (SELECT SUM(Sales)
    FROM orders
    WHERE YEAR(OrderDate) BETWEEN Year_Id-1 AND Year_Id) AS
MovingAnnualSales
FROM orders
GROUP BY Year_Id;
```

```sql
-- Query 12: Get moving annual total sales by year

SELECT
    Year_Id AS Year,
    SUM(Sales) AS AnnualSales,
    (SELECT SUM(Sales)
     FROM orders
     WHERE YEAR(OrderDate) BETWEEN Year_Id-1 AND Year_Id) AS MovingAnnualSales
FROM orders
GROUP BY Year_Id;
```

100% 18:241

Result Grid Filter Rows: Search Export:

Year	AnnualSales	MovingAnnualSales
2003	3516979.54	10032628.85
2004	4724162.60	10032628.85
2005	1791486.71	10032628.85

Query 13: Get order status percentage breakdown by year

```sql
SELECT o.Year_Id, o.Status, COUNT(*) / (SELECT COUNT(*) FROM orders o2 WHERE o2.Year_Id = o.Year_Id) * 100 AS Percent
FROM orders o
GROUP BY o.Year_Id, o.Status;
```

```sql
-- Query 13: Get order status percentage breakdown by year

SELECT o.Year_Id, o.Status, COUNT(*) / (SELECT COUNT(*) FROM orders o2 WHERE o2.Year_Id = o.Year_Id) * 100 AS Percent
FROM orders o
GROUP BY o.Year_Id, o.Status;
```

100% 30:248

Result Grid Filter Rows: Search Export:

Year_Id	Status	Percent
2003	Shipped	97.6000
2004	Shipped	95.6877
2005	Shipped	74.0586
2005	Disputed	2.9289
2005	In Process	8.5774
2003	Cancelled	1.6000
2004	Cancelled	3.2714
2005	On Hold	7.9498
2003	Resolved	0.8000
2004	On Hold	0.4461
2005	Resolved	6.4854
2004	Resolved	0.5948

Query 14: Sales by country and deal size:

```sql
SELECT c.Country, c.DealSize, SUM(o.Sales) AS TotalSales
FROM customer c
JOIN orders o ON c.CustomerId = o.CustomerId
GROUP BY c.Country, c.DealSize;
```

```
249
250     -- Query 14: Get sales by country and deal size:
251
252     SELECT c.Country, c.DealSize, SUM(o.Sales) AS TotalSales
253     FROM customer c
254     JOIN orders o ON c.CustomerId = o.CustomerId
255     GROUP BY c.Country, c.DealSize;
256
```

100% 32:255

Result Grid Filter Rows: Search Export:

Country	DealSize	TotalSales
USA	Small	467563.46
France	Small	204095.35
France	Medium	543737.98
USA	Medium	2090616.14
Norway	Medium	307463.70
Australia	Medium	265586.87
Finland	Small	225211.53
Austria	Large	149798.63
Australia	Small	305567.11
UK	Small	118008.27
Spain	Large	1082551.44
Sweden	Large	210014.21
Singap…	Large	288488.41
Australia	Large	59469.12
France	Large	363083.19
Canada	Large	75238.92
Japan	Large	120562.74

Query 15: Get sales percentage by country

```
SELECT
   c.Country,
   SUM(o.Sales) AS TotalSales,
   SUM(o.Sales) / (SELECT SUM(Sales) FROM orders) * 100 AS Percentage
FROM
   orders o
      JOIN
   customer c ON o.CustomerId = c.CustomerId
GROUP BY c.Country
ORDER BY TotalSales DESC;
```

```sql
-- Query 15: Get sales percentage by country
SELECT
    c.Country,
    SUM(o.Sales) AS TotalSales,
    SUM(o.Sales) / (SELECT SUM(Sales) FROM orders) * 100 AS Percentage
FROM
    orders o
        JOIN
    customer c ON o.CustomerId = c.CustomerId
GROUP BY c.Country
ORDER BY TotalSales DESC;
```

100% 26:267

Result Grid Filter Rows: Search Export:

Country	TotalSales	Percentage
USA	3627982.83	36.161836
Spain	1215686.92	12.117332
France	1110916.52	11.073035
Australia	630623.10	6.285721
UK	478880.46	4.773230
Italy	374674.31	3.734558
Finland	329581.91	3.285100
Norway	307463.70	3.064637
Singapore	288488.41	2.875502
Denmark	245637.15	2.448383
Canada	224078.56	2.233498
Germany	220472.09	2.197551
Sweden	210014.21	2.093312
Austria	202062.53	2.014054
Japan	188167.81	1.875558
Switzerla...	117713.56	1.173307
Belgium	108412.62	1.080600
Philippines	94015.73	0.937100

Query 16: Get total sales by product category and country

```sql
SELECT
  c.Country,
  p.ProductCategory,
  SUM(o.Sales) AS TotalSales
FROM orders o
JOIN customer c ON o.CustomerId = c.CustomerId
JOIN product p ON o.ProductCode = p.ProductCode
GROUP BY c.Country, p.ProductCategory
ORDER BY TotalSales DESC;
```

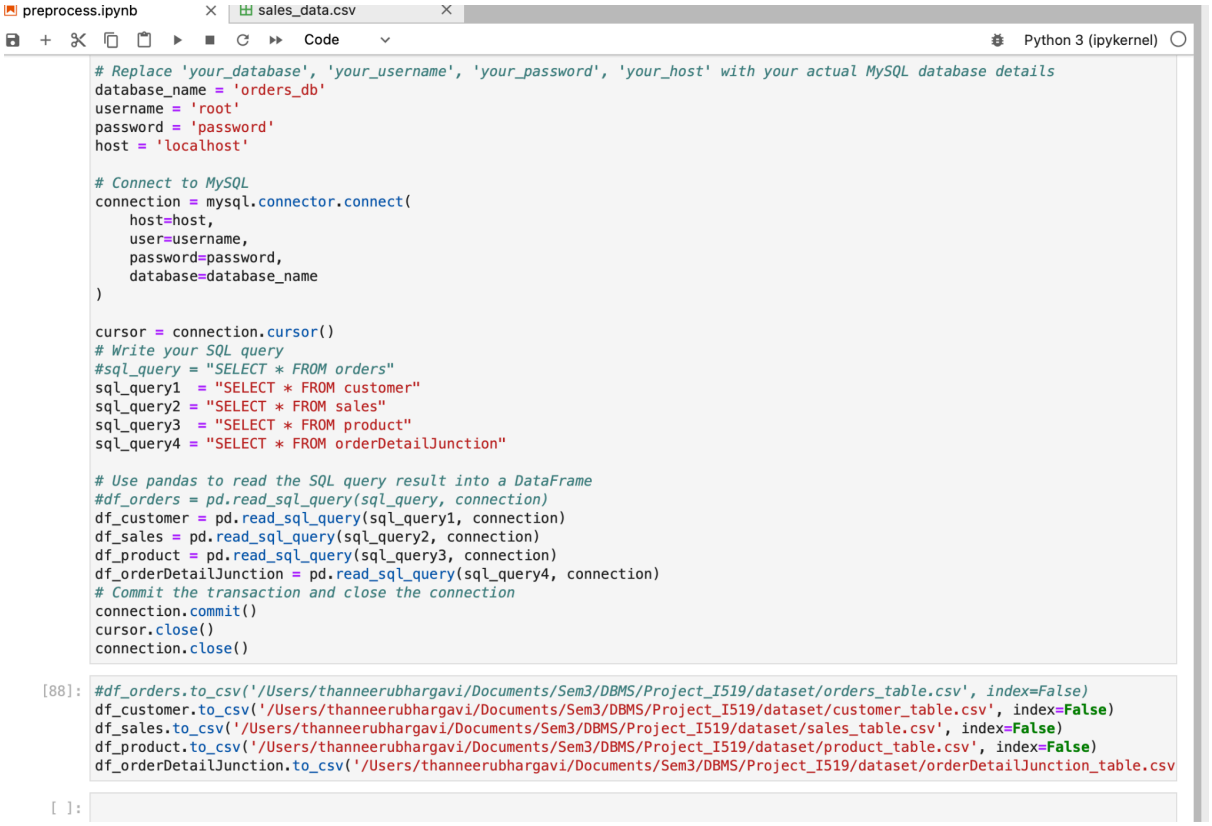
```sql
-- Query 16: Get total sales by product category and country

SELECT
  c.Country,
  p.ProductCategory,
  SUM(o.Sales) AS TotalSales
FROM orders o
JOIN customer c ON o.CustomerId = c.CustomerId
JOIN product p ON o.ProductCode = p.ProductCode
GROUP BY c.Country, p.ProductCategory
ORDER BY TotalSales DESC;
```

100% 26:279

Result Grid Filter Rows: Search Export:

Country	ProductCategory	TotalSales
USA	Classic Cars	1344638.22
USA	Vintage Cars	757755.90
USA	Motorcycles	520371.70
Spain	Classic Cars	476165.15
USA	Trucks and Buses	397842.42
France	Classic Cars	388951.20
USA	Planes	328432.89
Spain	Vintage Cars	229514.51
France	Motorcycles	226390.31
USA	Ships	209688.14
Australia	Classic Cars	193085.54
Australia	Vintage Cars	189555.32
Spain	Trucks and Buses	177556.78
France	Vintage Cars	176609.81
UK	Classic Cars	159377.70
Denm…	Classic Cars	157182.48
Finland	Classic Cars	153552.24
Germ…	Classic Cars	148315.00

Converted SQL Output Data to CSV files for Data Visualization:



preprocess.ipynb | sales_data.csv

Code | Python 3 (ipykernel)

```
# Replace 'your_database', 'your_username', 'your_password', 'your_host' with your actual MySQL database details
database_name = 'orders_db'
username = 'root'
password = 'password'
host = 'localhost'

# Connect to MySQL
connection = mysql.connector.connect(
    host=host,
    user=username,
    password=password,
    database=database_name
)

cursor = connection.cursor()
# Write your SQL query
#sql_query = "SELECT * FROM orders"
sql_query1  = "SELECT * FROM customer"
sql_query2 = "SELECT * FROM sales"
sql_query3  = "SELECT * FROM product"
sql_query4 = "SELECT * FROM orderDetailJunction"

# Use pandas to read the SQL query result into a DataFrame
#df_orders = pd.read_sql_query(sql_query, connection)
df_customer = pd.read_sql_query(sql_query1, connection)
df_sales = pd.read_sql_query(sql_query2, connection)
df_product = pd.read_sql_query(sql_query3, connection)
df_orderDetailJunction = pd.read_sql_query(sql_query4, connection)
# Commit the transaction and close the connection
connection.commit()
cursor.close()
connection.close()
```

[88]:
```
#df_orders.to_csv('/Users/thanneerubhargavi/Documents/Sem3/DBMS/Project_I519/dataset/orders_table.csv', index=False)
df_customer.to_csv('/Users/thanneerubhargavi/Documents/Sem3/DBMS/Project_I519/dataset/customer_table.csv', index=False)
df_sales.to_csv('/Users/thanneerubhargavi/Documents/Sem3/DBMS/Project_I519/dataset/sales_table.csv', index=False)
df_product.to_csv('/Users/thanneerubhargavi/Documents/Sem3/DBMS/Project_I519/dataset/product_table.csv', index=False)
df_orderDetailJunction.to_csv('/Users/thanneerubhargavi/Documents/Sem3/DBMS/Project_I519/dataset/orderDetailJunction_table.csv
```

[ ]:

Tableau Visualizations:

After querying to get some valuable insights we went on to do some visualizations in tableau. The below tableau public link directs you to the dashboard for checking on some insights visually.

https://public.tableau.com/views/INFOI_519Project/Dashboard1?:language=en-US&:display_count=n&:origin=viz_share_link

1) The first visualization talks about total sales of a country and legend has color saturation as visual encoding.

2) The second is a visualization on Total number of orders per each product category, here the categories are colored to their names, using color hue visual encoding.

3) The third visualization is Quantity ordered by each customer in each quarter cycle.