Online Optical Lens Delivery Service

Report File

Group 12

Aditya Kumar
Nishant Upadhyay
8573398150
8573518056

kumar.aditya1@northeastern.edu
upadhyay.nis@northeastern.edu

Percentage contributed by Student 1: 50%

Signature of Student 1: Aditya Kumar

Percentage contributed by Student 2: 50%

Signature of Student 2: Nishant Upadhyay

Executive Summary

Our innovative online optical lens delivery service is designed to transform the contact lens market by addressing challenges faced by customers, such as cumbersome prescription verification, limited product availability, and the inconvenience of visiting physical stores. These problems often result in delays, limited choices, and decreased customer satisfaction. The goal of this project is to provide a seamless, user-friendly platform that simplifies the process of purchasing contact lenses. The platform prioritizes ease of use by allowing customers to create profiles, upload prescriptions, and browse personalized product catalogs. Advanced features such as secure payments, flexible shipping options, and subscription-based reordering ensure a hassle-free experience.

To meet this goal, the requirements for the platform include:

A comprehensive database management system (DBMS) to handle customer profiles, prescriptions, and order history.

Integration of real-time inventory tracking to prevent shortages and ensure availability.

Automated prescription verification to speed up the ordering process.

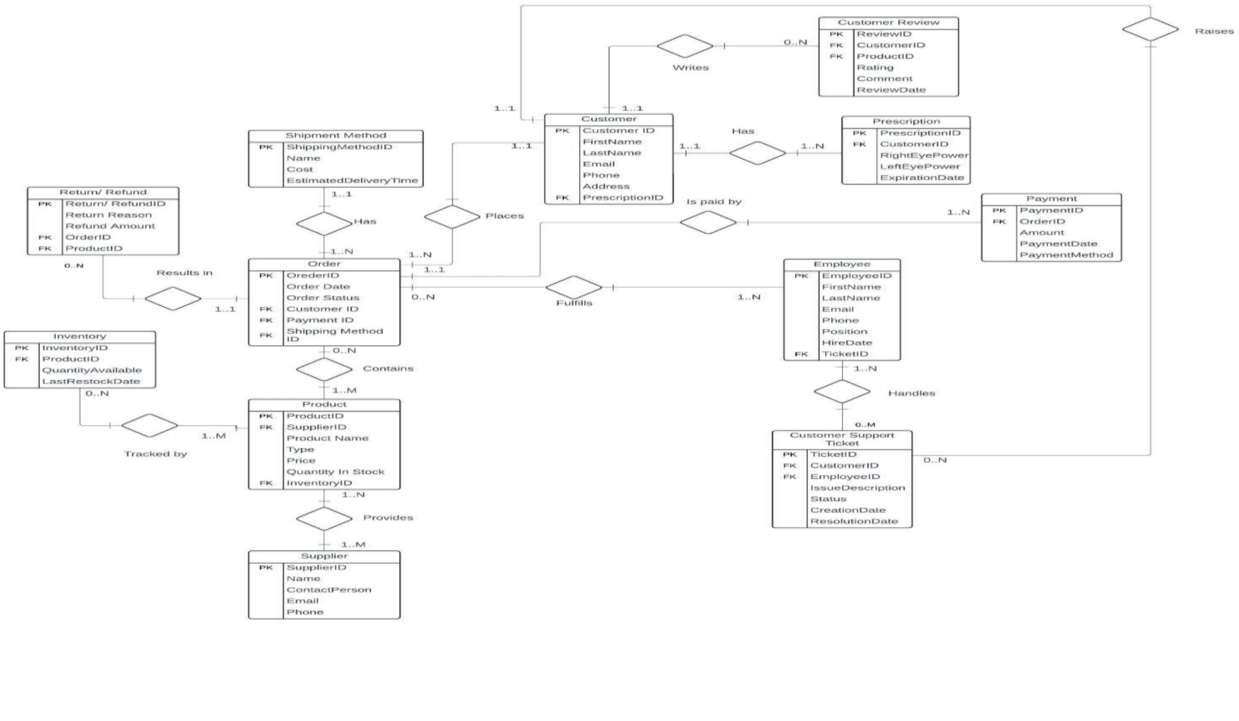
Supplier management tools to streamline restocking and maintain a diverse product range.

Secure payment gateways and support for multiple shipping options to enhance user convenience.

Features for order tracking, returns, and refunds to provide transparency and reliability.

A subscription-based model for automated reordering to promote customer retention.

Conceptual Data Modeling: EER Diagram



Conceptual Data Modeling: UML Diagram

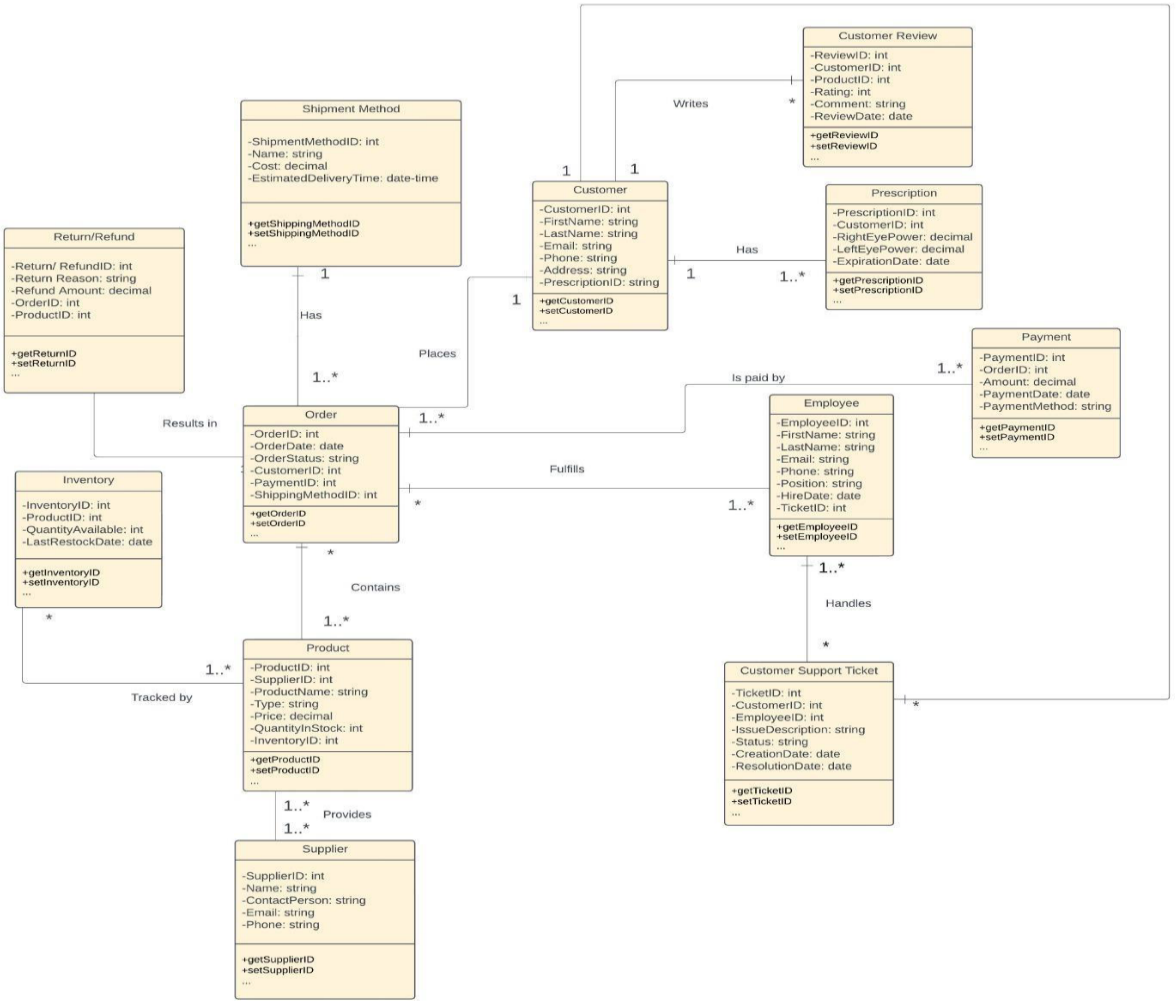
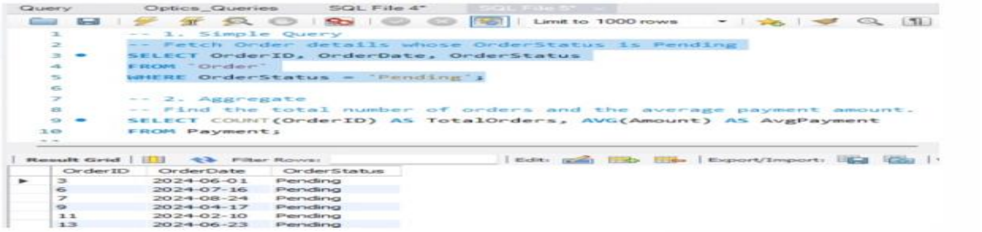


	TABLE NAME	PRIMARY KEY	ATTRIBUTES
1.	Order	OrderID	*OrderID*, OrderDate, OrderStatus, CustomerID, PaymentID, ShippingMethodID
2.	Customer	CustomerID	*CutsomerID*, FirstName, LastName, Email, Phone, Address, PrescriptionID
3.	Employee	EmployeeID	*EmployeeID*, FirstName, LastName, Email, Phone, Position, HireDate, TicketID

4.	Prescription	PrescriptionID	*PrescriptionID*, CustomerID, RightEyePower, LeftEyePower, ExpirationDate
5.	Payment	PaymentID	*PaymentID*, OrderID, Amount, PaymentDate, PaymentMethod
6.	Inventory	InventoryID	InventoryID, ProductID, QuantityAvailable, LastRestockDate
7.	ShippingMethod	ShipmentMethodID	ShipmentMethodID, Name, Cost, EstimatedDeliveryTime
8.	Supplier	SupplierID	SupplierID, Name, Contact, Email, Phone
9.	Product	ProductID	*ProductID*, SupplierID, ProductName, Type, Price, QuantityInStock, InventoryID
10.	CustomerReview	ReviewID	*ReviewID*, CustomerID, ProductID, Rating, Comment, ReviewDate
11.	CustomerSupportTicket	TicketID	*TicketID*, CustomerID, EmployeeID, IssueDescription, Status, CreationDate, ResolutionDate
12.	Return/Refund	Return/RefundID	*Return/RefundID*, ReturnReason, Refund, OrderID, ProductID
13.	Contains	OrderID, ProductID	*OrderID, ProductID*
14.	Provides	SupplierID, ProductID	*SupplierID, ProductID*
15.	Handles	EmployeeID, TicketID	*EmployeeID, TicketID*
16.	TrackedBy	InventoryID, ProductID	*InventoryID, ProductID*, Quantity, LastRestockDate
17.	Fulfills	OrderID,EmployeeID	*OrderID,EmployeeID*

Implementation of Relation Model via MySQL

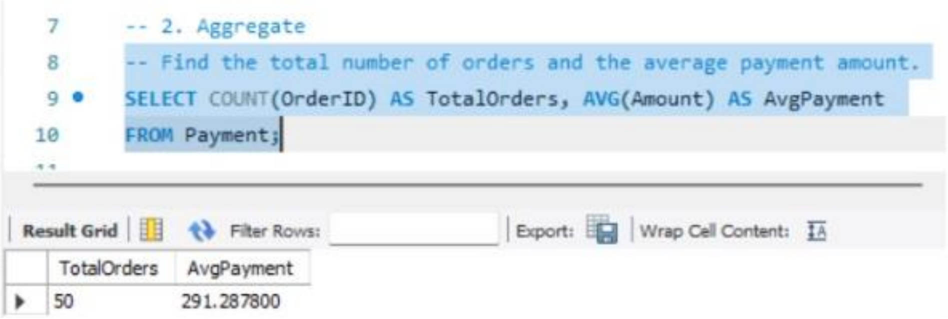
- Q1 : Fetch Order details whose OrderStatus is Pending:



```sql
-- 1. Simple Query
-- Fetch Order details whose OrderStatus is Pending
SELECT OrderID, OrderDate, OrderStatus
FROM `Order`
WHERE OrderStatus = 'Pending';

-- 2. Aggregate
-- Find the total number of orders and the average payment amount.
SELECT COUNT(OrderID) AS TotalOrders, AVG(Amount) AS AvgPayment
FROM Payment;
```

OrderID	OrderDate	OrderStatus
3	2024-06-01	Pending
6	2024-07-16	Pending
7	2024-08-24	Pending
9	2024-04-17	Pending
11	2024-02-10	Pending
13	2024-06-23	Pending

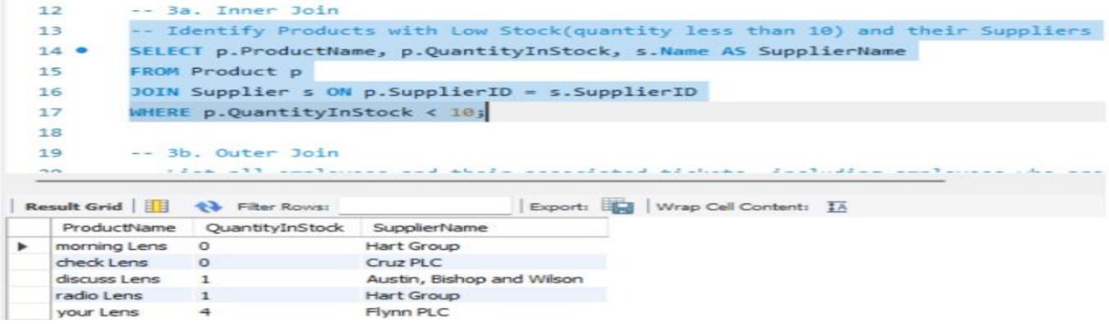
- Q2: Find the total number of orders and the average payment amount.



```sql
-- 2. Aggregate
-- Find the total number of orders and the average payment amount.
SELECT COUNT(OrderID) AS TotalOrders, AVG(Amount) AS AvgPayment
FROM Payment;
```

TotalOrders	AvgPayment
50	291.287800

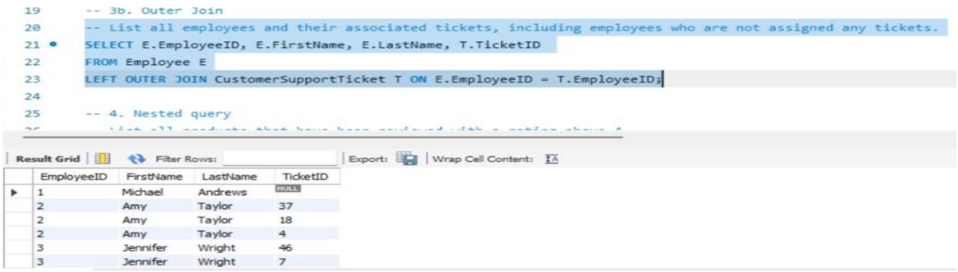
- Q3a: Identify Products with Low stock (quantity less than 10) and their Suppliers



```sql
-- 3a. Inner Join
-- Identify Products with Low Stock(quantity less than 10) and their Suppliers
SELECT p.ProductName, p.QuantityInStock, s.Name AS SupplierName
FROM Product p
JOIN Supplier s ON p.SupplierID = s.SupplierID
WHERE p.QuantityInStock < 10;

-- 3b. Outer Join
```

ProductName	QuantityInStock	SupplierName
morning Lens	0	Hart Group
check Lens	0	Cruz PLC
discuss Lens	1	Austin, Bishop and Wilson
radio Lens	1	Hart Group
your Lens	4	Flynn PLC

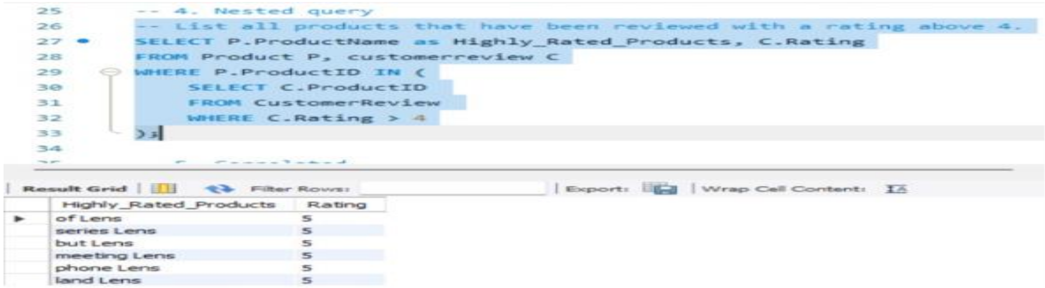
- Q3b: List all employees and their associated tickets, including employees who are not assigned any tickets.



```sql
-- 3b. Outer Join
-- List all employees and their associated tickets, including employees who are not assigned any tickets.
SELECT E.EmployeeID, E.FirstName, E.LastName, T.TicketID
FROM Employee E
LEFT OUTER JOIN CustomerSupportTicket T ON E.EmployeeID = T.EmployeeID;

-- 4. Nested query
```

EmployeeID	FirstName	LastName	TicketID
1	Michael	Andrews	NULL
2	Amy	Taylor	37
2	Amy	Taylor	18
2	Amy	Taylor	4
3	Jennifer	Wright	46
3	Jennifer	Wright	7

Q4: List all products that have been reviewed with a rating above 4.

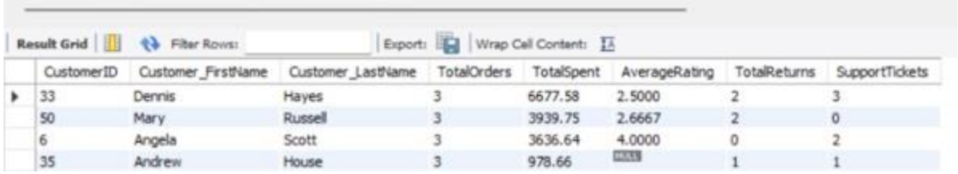


```
-- 4. Nested query
-- List all products that have been reviewed with a rating above 4.
SELECT P.ProductName as Highly_Rated_Products, C.Rating
FROM Product P, customerreview C
WHERE P.ProductID IN (
    SELECT C.ProductID
    FROM CustomerReview
    WHERE C.Rating > 4
);
```

Highly_Rated_Products	Rating
of Lens	5
series Lens	5
but Lens	5
meeting Lens	5
phone Lens	5
land Lens	5

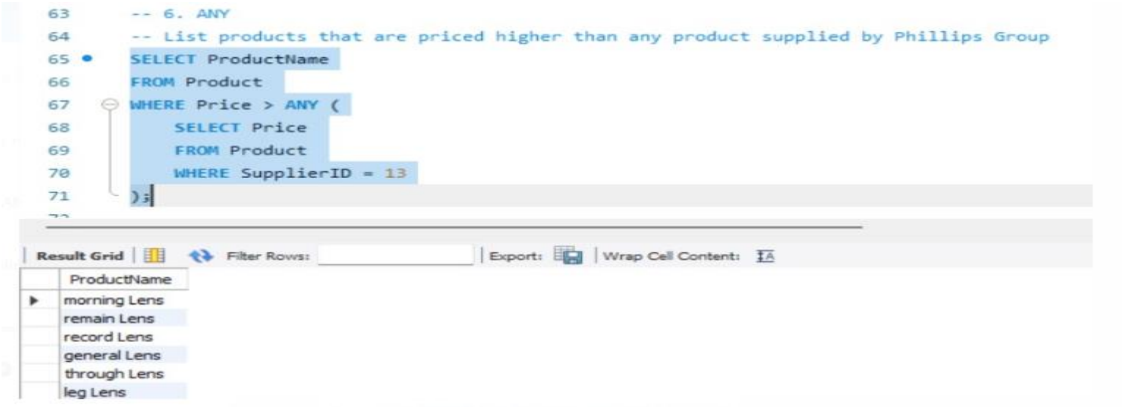
- Q5: Querying detailed summary of customer activity, including total orders, spending, average ratings, returns, and support ticket interactions for customers with more than 2 orders.

```
SELECT
    C.CustomerID,
    C.FirstName as Customer_FirstName,
    C.LastName as Customer_LastName,
    COUNT(DISTINCT O.OrderID) AS TotalOrders,
    SUM(P.Amount) AS TotalSpent,
    AVG(CR.Rating) AS AverageRating,
    COUNT(DISTINCT R.ReturnRefundID) AS TotalReturns,
    COUNT(DISTINCT T.TicketID) AS SupportTickets
FROM Customer C
LEFT JOIN Order O ON O.CustomerID = C.CustomerID
LEFT JOIN Payment P ON P.OrderID = O.OrderID
LEFT JOIN CustomerReview CR ON CR.CustomerID = C.CustomerID
LEFT JOIN Product P2 ON CR.ProductID = P2.ProductID
LEFT JOIN Return_Refund R ON R.OrderID = O.OrderID
LEFT JOIN CustomerSupportTicket T ON T.CustomerID = C.CustomerID
WHERE (
    SELECT COUNT(*)
    FROM Order O2
    WHERE O2.CustomerID = C.CustomerID
) > 2
GROUP BY C.CustomerID
ORDER BY TotalSpent DESC;
```



CustomerID	Customer_FirstName	Customer_LastName	TotalOrders	TotalSpent	AverageRating	TotalReturns	SupportTickets
33	Dennis	Hayes	3	6677.58	2.5000	2	3
50	Mary	Russell	3	3939.75	2.6667	2	0
6	Angela	Scott	3	3636.64	4.0000	0	2
35	Andrew	House	3	978.66	NULL	1	1

- Q6: List products that are priced higher than any product supplied by Phillips Group.

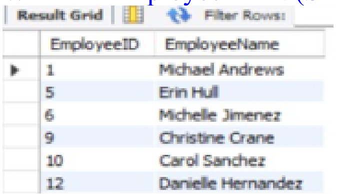


```sql
-- 6. ANY
-- List products that are priced higher than any product supplied by Phillips Group
SELECT ProductName
FROM Product
WHERE Price > ANY (
    SELECT Price
    FROM Product
    WHERE SupplierID = 13
);
```

ProductName
morning Lens
remain Lens
record Lens
general Lens
through Lens
leg Lens

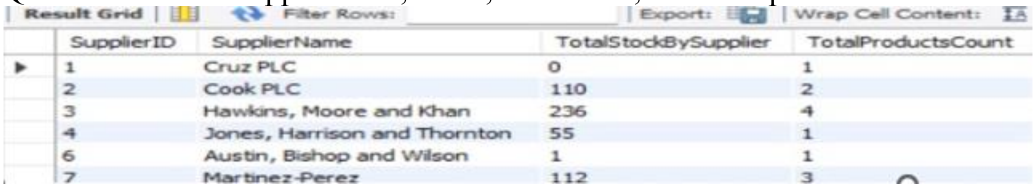
- Q7 : List all employees who either handle customer support tickets or fulfill orders.

```sql
SELECT EmployeeID, CONCAT(FirstName, ' ', LastName) AS EmployeeName
FROM Employee
WHERE EmployeeID IN (SELECT EmployeeID FROM Handles)
UNION
SELECT EmployeeID, CONCAT(FirstName, ' ', LastName) AS EmployeeName
FROM Employee
WHERE EmployeeID IN (SELECT EmployeeID FROM Fulfills);
```



EmployeeID	EmployeeName
1	Michael Andrews
5	Erin Hull
6	Michelle Jimenez
9	Christine Crane
10	Carol Sanchez
12	Danielle Hernandez

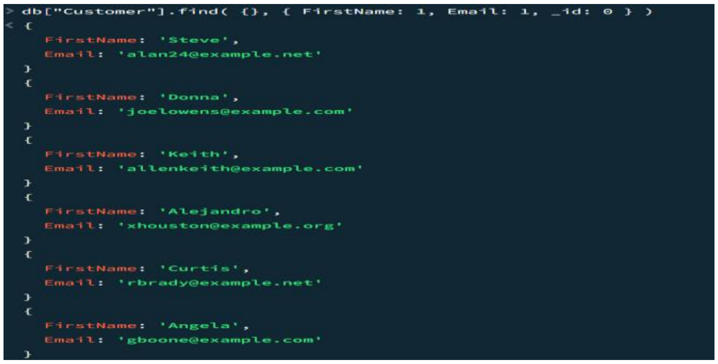
Q8: Retrieve each supplier's ID, name, total stock, and total product count



SupplierID	SupplierName	TotalStockBySupplier	TotalProductsCount
1	Cruz PLC	0	1
2	Cook PLC	110	2
3	Hawkins, Moore and Khan	236	4
4	Jones, Harrison and Thornton	55	1
6	Austin, Bishop and Wilson	1	1
7	Martinez-Perez	112	3

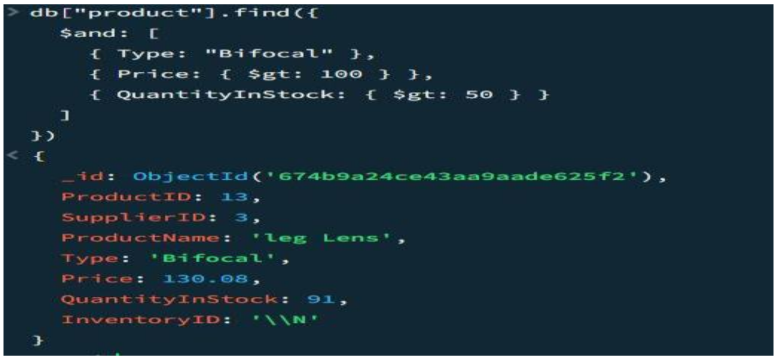
Implementation of Relational model via NoSQL

Q1: Fetching Customer’s First name and Email address from Customer collection



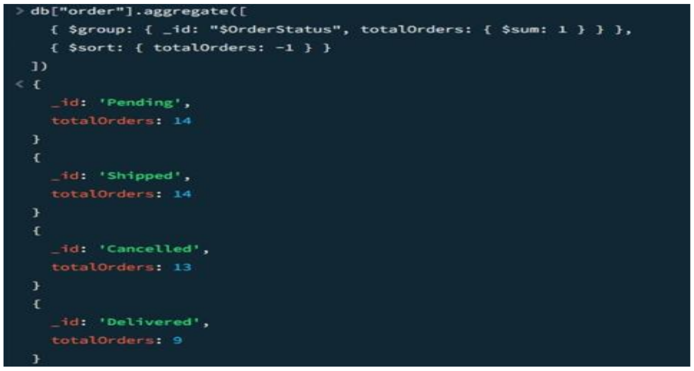
```
> db["Customer"].find( {}, { FirstName: 1, Email: 1, _id: 0 } )
< {
    FirstName: 'Steve',
    Email: 'alan24@example.net'
  }
  {
    FirstName: 'Donna',
    Email: 'joelowens@example.com'
  }
  {
    FirstName: 'Keith',
    Email: 'allenkeith@example.com'
  }
  {
    FirstName: 'Alejandro',
    Email: 'xhouston@example.org'
  }
  {
    FirstName: 'Curtis',
    Email: 'rbrady@example.net'
  }
  {
    FirstName: 'Angela',
    Email: 'gboone@example.com'
  }
```

Q2: Query Bifocal Lens type whose price is greater than $100 and Quantity is greater than $50



```
> db["product"].find({
    $and: [
      { Type: "Bifocal" },
      { Price: { $gt: 100 } },
      { QuantityInStock: { $gt: 50 } }
    ]
  })
< {
    _id: ObjectId('674b9a24ce43aa9aade625f2'),
    ProductID: 13,
    SupplierID: 3,
    ProductName: 'leg Lens',
    Type: 'Bifocal',
    Price: 130.08,
    QuantityInStock: 91,
    InventoryID: '\\N'
  }
```

Q3: How many orders exist for each orderStatus, sorted by the highest count?

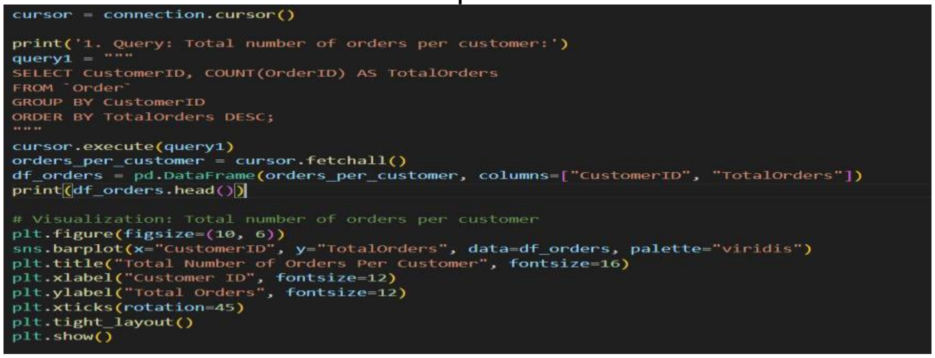


```
> db["order"].aggregate([
    { $group: { _id: "$OrderStatus", totalOrders: { $sum: 1 } } },
    { $sort: { totalOrders: -1 } }
  ])
< {
    _id: 'Pending',
    totalOrders: 14
  }
  {
    _id: 'Shipped',
    totalOrders: 14
  }
  {
    _id: 'Cancelled',
    totalOrders: 13
  }
  {
    _id: 'Delivered',
    totalOrders: 9
  }
```

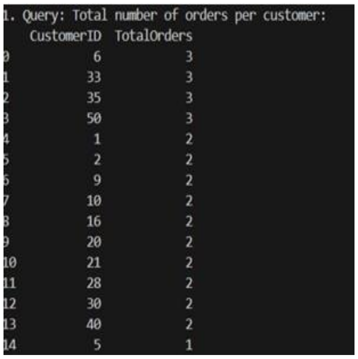
Database Access via Python

The MySQL database was accessed using the mysql.connector python library. A connection was established using the connect method, where the necessary credentials, including the host (localhost), user (root), password (2001), and database name (optics), were provided. These parameters authenticated the connection to the MySQL server, enabling seamless communication between Python and the database. This connection allowed the execution of SQL queries to retrieve and manipulate data, which could then be analyzed using Pandas and visualized using libraries like Matplotlib and Seaborn. This integration facilitated efficient data handling and insightful visualizations.

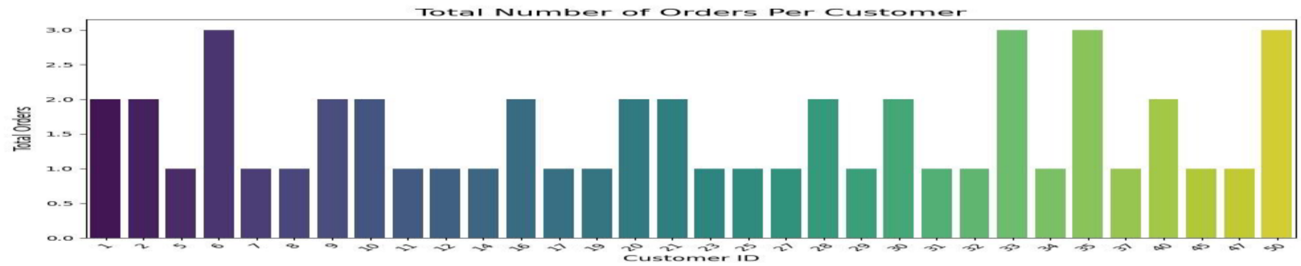
Q1. Find Total number of Orders per customer.



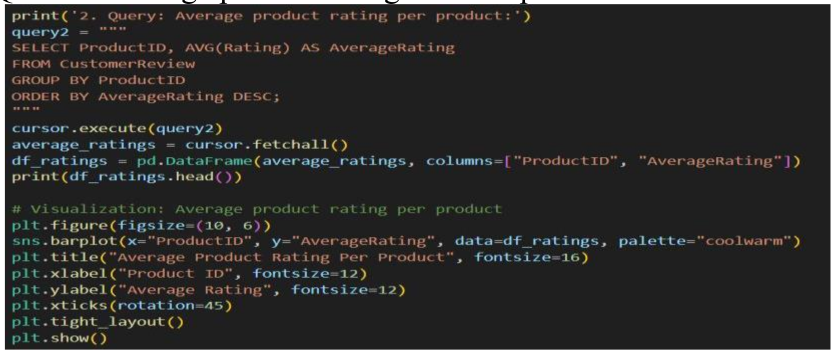
```
cursor = connection.cursor()

print('1. Query: Total number of orders per customer:')
query1 = """
SELECT CustomerID, COUNT(OrderID) AS TotalOrders
FROM `Order`
GROUP BY CustomerID
ORDER BY TotalOrders DESC;
"""
cursor.execute(query1)
orders_per_customer = cursor.fetchall()
df_orders = pd.DataFrame(orders_per_customer, columns=["CustomerID", "TotalOrders"])
print(df_orders.head())

# Visualization: Total number of orders per customer
plt.figure(figsize=(10, 6))
sns.barplot(x="CustomerID", y="TotalOrders", data=df_orders, palette="viridis")
plt.title("Total Number of Orders Per Customer", fontsize=16)
plt.xlabel("Customer ID", fontsize=12)
plt.ylabel("Total Orders", fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



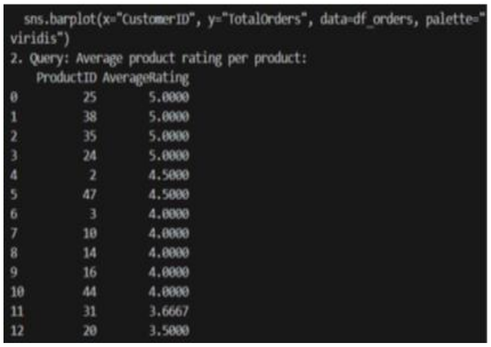
```
1. Query: Total number of orders per customer:
   CustomerID  TotalOrders
0           6            3
1          33            3
2          35            3
3          50            3
4           1            2
5           2            2
6           9            2
7          10            2
8          16            2
9          20            2
10         21            2
11         28            2
12         30            2
13         40            2
14          5            1
```



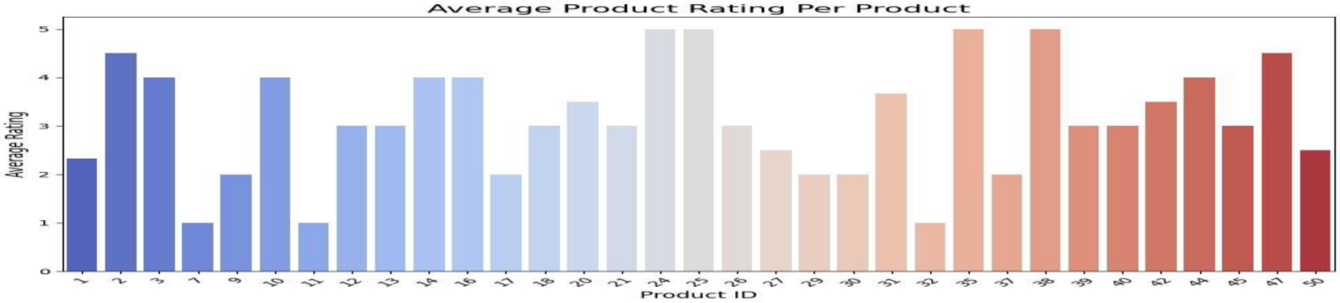
Q2. Find average product rating for each product.



```
print('2. Query: Average product rating per product:')
query2 = """
SELECT ProductID, AVG(Rating) AS AverageRating
FROM CustomerReview
GROUP BY ProductID
ORDER BY AverageRating DESC;
"""
cursor.execute(query2)
average_ratings = cursor.fetchall()
df_ratings = pd.DataFrame(average_ratings, columns=["ProductID", "AverageRating"])
print(df_ratings.head())

# Visualization: Average product rating per product
plt.figure(figsize=(10, 6))
sns.barplot(x="ProductID", y="AverageRating", data=df_ratings, palette="coolwarm")
plt.title("Average Product Rating Per Product", fontsize=16)
plt.xlabel("Product ID", fontsize=12)
plt.ylabel("Average Rating", fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```
 sns.barplot(x="CustomerID", y="TotalOrders", data=df_orders, palette="
viridis")
2. Query: Average product rating per product:
   ProductID AverageRating
0         25        5.0000
1         38        5.0000
2         35        5.0000
3         24        5.0000
4          2        4.5000
5         47        4.5000
6          3        4.0000
7         10        4.0000
8         14        4.0000
9         16        4.0000
10        44        4.0000
11        31        3.6667
12        20        3.5000
```



Graph Explanation: This graph represents the **Average Product Rating Per Product**, with **Product ID** on the x-axis and **Average Rating** on the y-axis

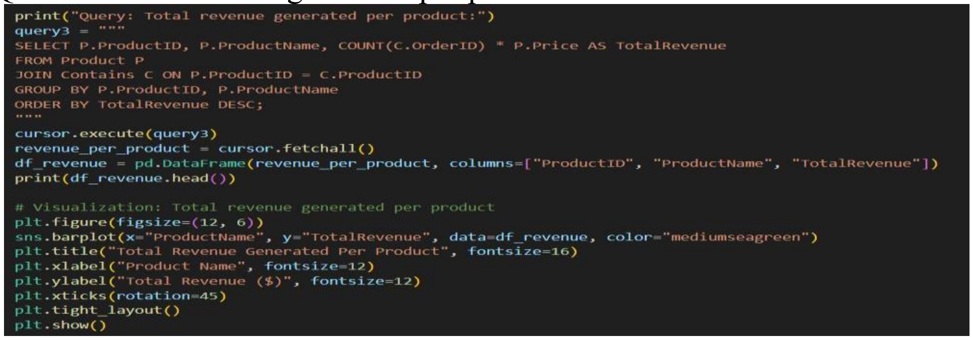
Analysis:

Products such as those with IDs 24, 25, 35, and 38 have ratings close to or at the maximum value of 5, indicating strong customer satisfaction.

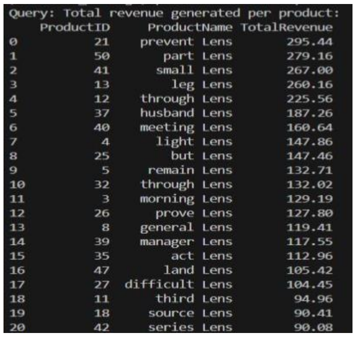
Products like 7, 11, and 32 have significantly lower ratings, indicating potential quality or experience issues that need to be addressed.

There is a noticeable variation in average ratings across products, suggesting differing levels of customer satisfaction

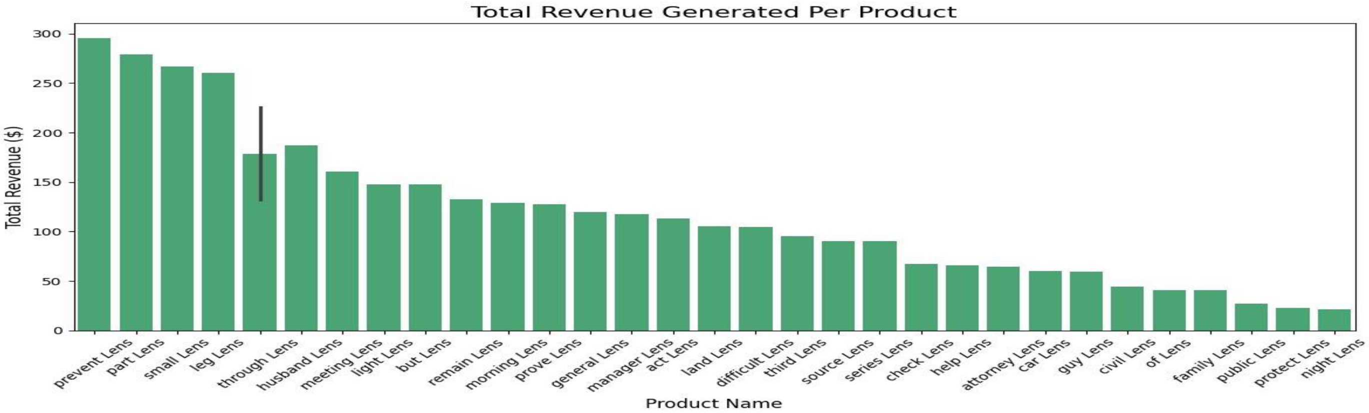
Q3. Find total revenue generated per product



```
print("Query: Total revenue generated per product:")
query3 = """
SELECT P.ProductID, P.ProductName, COUNT(C.OrderID) * P.Price AS TotalRevenue
FROM Product P
JOIN Contains C ON P.ProductID = C.ProductID
GROUP BY P.ProductID, P.ProductName
ORDER BY TotalRevenue DESC;
"""
cursor.execute(query3)
revenue_per_product = cursor.fetchall()
df_revenue = pd.DataFrame(revenue_per_product, columns=["ProductID", "ProductName", "TotalRevenue"])
print(df_revenue.head())

# Visualization: Total revenue generated per product
plt.figure(figsize=(12, 6))
sns.barplot(x="ProductName", y="TotalRevenue", data=df_revenue, color="mediumseagreen")
plt.title("Total Revenue Generated Per Product", fontsize=16)
plt.xlabel("Product Name", fontsize=12)
plt.ylabel("Total Revenue ($)", fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```
Query: Total revenue generated per product:
    ProductID     ProductName TotalRevenue
0          21    prevent Lens       295.44
1          50       part Lens       279.16
2          41      small Lens       267.00
3          13        leg Lens       260.16
4          12    through Lens       225.56
5          37    husband Lens       187.26
6          40    meeting Lens       160.64
7           4      light Lens       147.86
8          25        but Lens       147.46
9           5     remain Lens       132.71
10         32    through Lens       132.02
11          3    morning Lens       129.19
12         26      prove Lens       127.80
13          8    general Lens       119.41
14         39    manager Lens       117.55
15         35        act Lens       112.96
16         47       land Lens       105.42
17         27  difficult Lens       104.45
18         11      third Lens        94.96
19         18     source Lens        90.41
20         42     series Lens        90.08
```

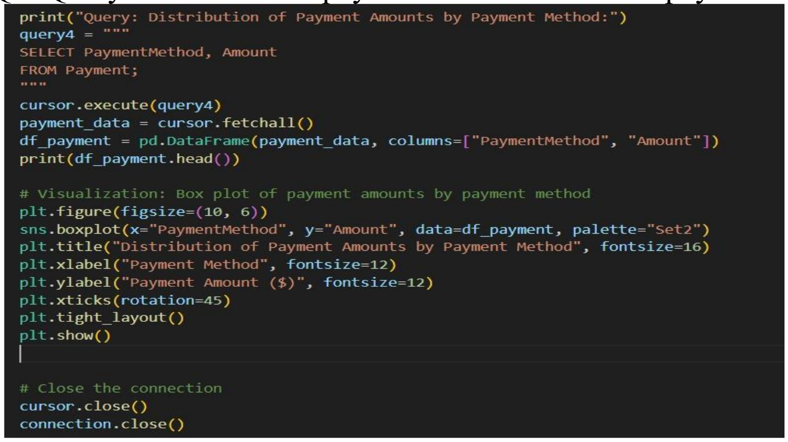


Graph 3: This graph represents the **Total Revenue generated Per Product**, with **Product Name** on the x-axis and **Total Revenue** on the y-axis

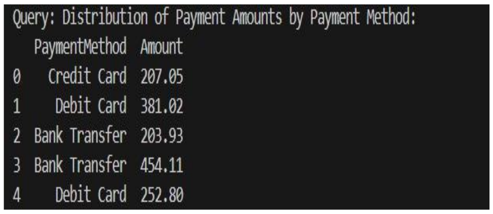
Analysis:

The bar graph highlights a significant disparity in revenue generation among the products, with a few top performers like "prevent Lens," "part Lens," and "small Lens" dominating the sales, while many others contribute minimally. This suggests a reliance on a small subset of products for the majority of revenue. The steep decline in revenue after the top four products indicates opportunities to improve the performance of mid- and low-tier items through strategic repositioning, marketing efforts, or product innovation. Focusing on the top products while addressing underperforming ones could enhance overall profitability.

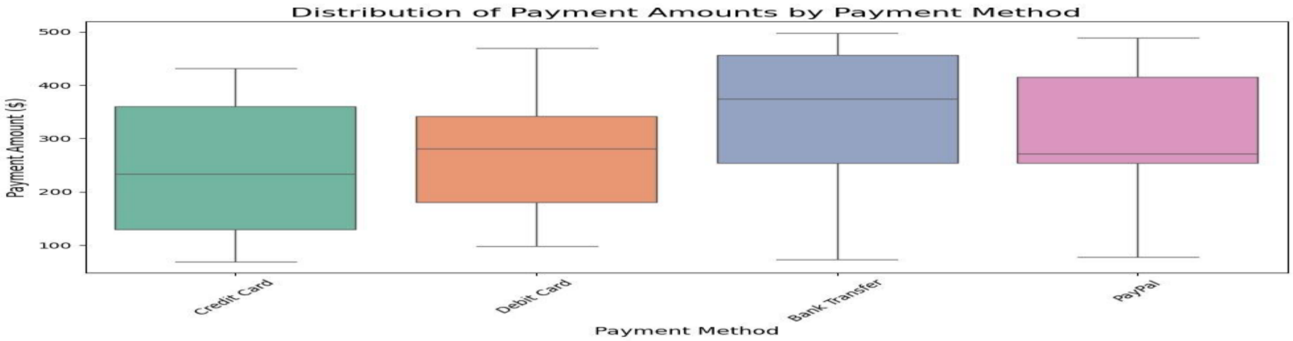
Q4. Query distribution of payment amount based on payment method



```
print("Query: Distribution of Payment Amounts by Payment Method:")
query4 = """
SELECT PaymentMethod, Amount
FROM Payment;
"""
cursor.execute(query4)
payment_data = cursor.fetchall()
df_payment = pd.DataFrame(payment_data, columns=["PaymentMethod", "Amount"])
print(df_payment.head())

# Visualization: Box plot of payment amounts by payment method
plt.figure(figsize=(10, 6))
sns.boxplot(x="PaymentMethod", y="Amount", data=df_payment, palette="Set2")
plt.title("Distribution of Payment Amounts by Payment Method", fontsize=16)
plt.xlabel("Payment Method", fontsize=12)
plt.ylabel("Payment Amount ($)", fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


# Close the connection
cursor.close()
connection.close()
```



```
Query: Distribution of Payment Amounts by Payment Method:
  PaymentMethod  Amount
0   Credit Card  207.05
1    Debit Card  381.02
2 Bank Transfer  203.93
3 Bank Transfer  454.11
4    Debit Card  252.80
```



Graph 4: This box plot illustrates the **Distribution of Payment Amounts by Payment Method**, comparing the spread of payment amounts across four methods: Credit Card, Debit Card, Bank Transfer, and PayPal.

Analysis:

This box plot shows that **Bank Transfer** is preferred for higher payment amounts, with the highest median and widest range. **Credit Card** and **PayPal** exhibit moderate ranges, while **Debit Card** has the lowest median and narrower payments, indicating its use for smaller transactions. Payment method variability suggests **Bank Transfer** is ideal for high-value products, while **Debit Card** suits everyday purchases.

Summary of the Optical Lens Delivery Service Project

The **Optical Lens Delivery Service** simplifies the purchasing process for contact lenses by offering a user-friendly online platform. Customers can create profiles, upload prescriptions, browse personalized catalogs, and make secure payments with flexible shipping and subscription-based reordering options.

The service operates with a robust backend database that handles prescription verification, real-time inventory tracking, supplier management, and employee role assignments. It automates restocking, tracks orders, and efficiently manages returns and refunds. The platform also enables detailed customer analytics to improve engagement and loyalty.

Recommendations

1. Enhance Customer Engagement:
   - Implement loyalty programs for frequent customers, especially those with lower satisfaction ratings.
   - Send personalized incentives to customers like Dennis Hayes, who have high order counts but lower ratings.
2. Improve Product Performance:
   - Analyze products with low ratings to identify and address quality or service issues.
   - Use marketing strategies to increase visibility and sales of underperforming products.

Conclusion

The Optical Lens Delivery Service is poised to transform the online contact lens market by prioritizing convenience, user experience, and operational efficiency. With focused improvements in customer engagement, product performance, and inventory management, the platform can enhance satisfaction and build long-term customer loyalty.