Retail Store Inventory Management System

Milestone: Project proposal

Group 7
ROHAN PRAKASH KRISHNA PRAKASH
NISHCHAY LINGE GOWDA

857-746-9940
617-792-8408

krishnaprakash.r@northeaster.edu
lingegowda.n@northeastern.edu

Percentage of Effort Contributed by Student1: 50%
Percentage of Effort Contributed by Student2: 50%

Signature of Student 1: Rohan Prakash Krishna Prakash
Signature of Student 2: Nishchay Linge Gowda

Retail Store Inventory Management System

Rohan Prakash Krishna Prakash and Nishchay Linge Gowda

Problem Statement:

With the rapid growth of retail operations and the increasing need to manage stock efficiently, a well-structured Retail Store Inventory Management System has become essential. Retail stores must track inventory levels, orders, sales, and deliveries accurately, ensuring that the right products are available to meet customer demands. Poorly managed inventory systems can lead to overstocking, causing excess costs, or stockouts, resulting in lost sales opportunities and customer dissatisfaction. The aim of this system is to optimize inventory processes through automation, real-time data collection, and analytics. By analysing key factors such as purchase trends, sales velocity, and supply chain logistics, the system ensures a balanced stock flow and reduces human errors in tracking, ordering, and restocking inventory. The system must also enhance supply chain efficiency by automating the reordering process when stock reaches certain thresholds, ensuring that the retail store never runs out of essential products. Additionally, the system can provide real-time updates on stock movements, enabling store managers to make informed decisions based on accurate, up-to-date data. This avoids overstocking or stockouts, leading to increased customer satisfaction, smoother operations, and ultimately, higher profitability for the business.

Theory for Inventory management for a retail store:

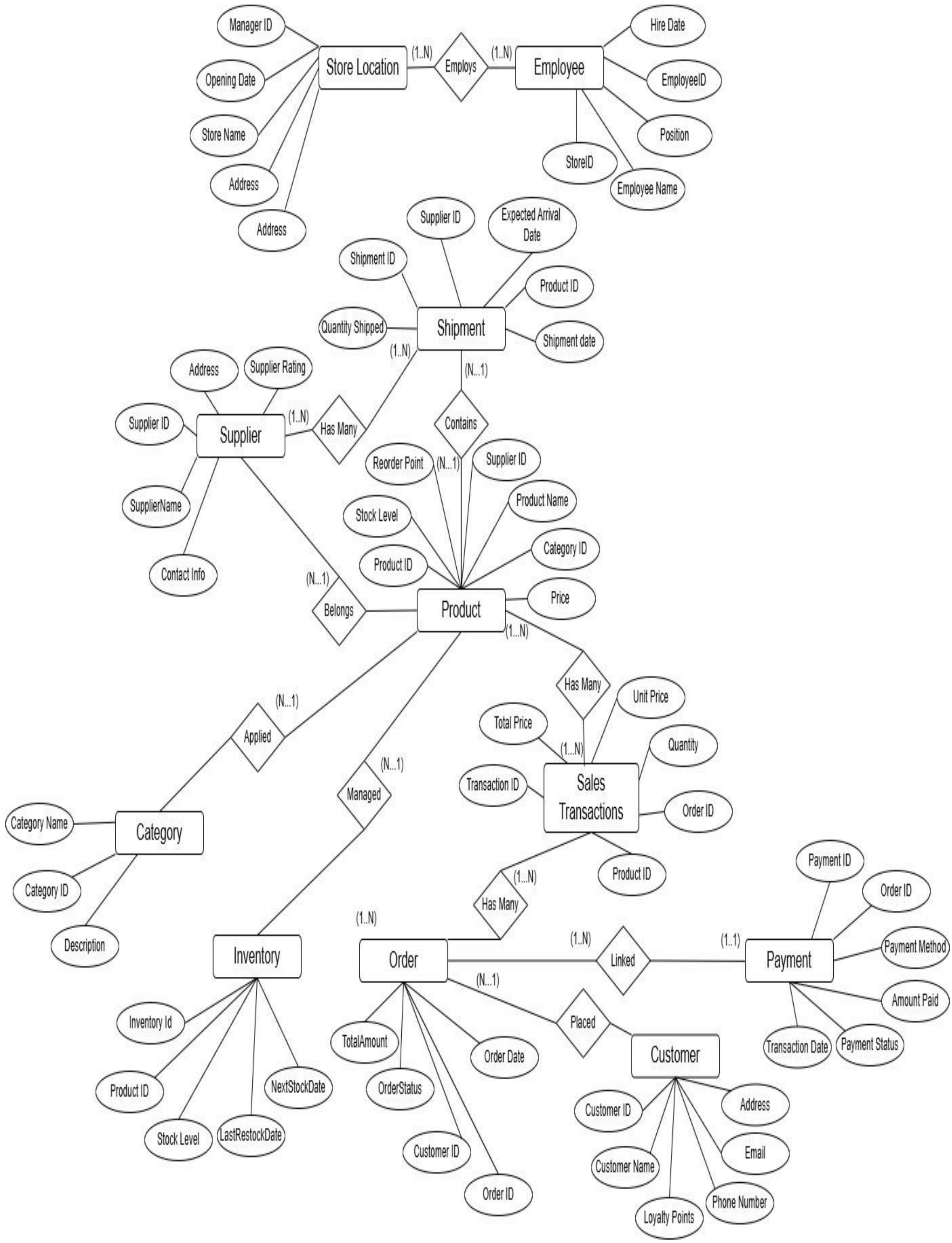
A well-structured Retail Store Inventory Management System is critical to meeting the dynamic demands of modern retail environments. As retail operations expand and customer expectations increase, it becomes essential to accurately track inventory levels, orders, sales, and deliveries. This system must prevent inefficiencies such as overstocking, which leads to increased holding costs, or stockouts, which result in lost sales and reduced customer satisfaction. To optimize these processes, the inventory management system should employ automation, real-time data analytics, and predictive modelling. This ensures seamless synchronization between the store's inventory levels and consumer demand, allowing store managers to manage stock more effectively. By collecting and analysing data on factors such as purchase trends, sales velocity, and supply chain logistics, the system can provide insights into inventory needs, reducing human errors in ordering, restocking, and managing stock levels. The system should also automate key processes such as reordering products when stock reaches pre-set thresholds, ensuring that essential items are never out of stock. Real-time updates on stock movements enable store managers to make quick and informed decisions, while an integrated dashboard offers insights into supply chain operations, helping to improve performance and reduce operational costs.

Additional Information

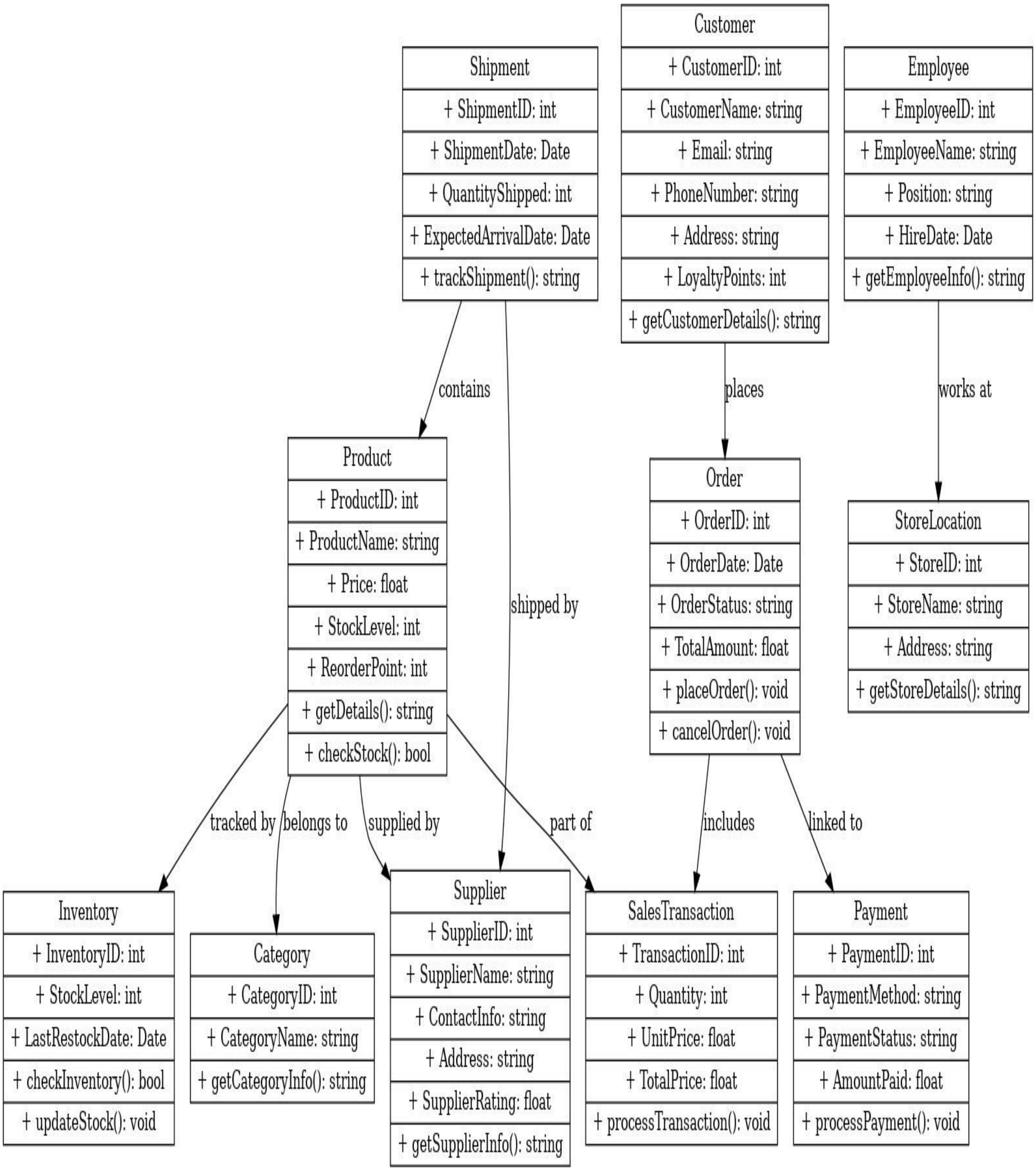
- **Product & Category:** One category has many products, while each product belongs to one category.
- **Product & Supplier**: A product is supplied by one supplier, while a supplier can supply many products.
- **Product & Sales Transaction:** A product can appear in many sales transactions.
- **Customer & Order:** A customer can place many orders, but each order is linked to only one customer.
- **Order & Sales Transaction:** Each order can contain many transactions (line items).
- **Customer Feedback:** Customers can give feedback on multiple products.
- **Shipment:** Each shipment involves a specific product and is sent by a supplier.
- **Discount:** Discounts can apply to multiple products (many-to-many relationship).

Conceptual Model (EER and UML Model)

- **EER Model**


Manager ID
Opening Date
Store Name
Address
Address
Store Location
(1..N)
Employs
(1..N)
Employee
Hire Date
EmployeeID
Position
StoreID
Employee Name
Supplier ID
Expected Arrival Date
Shipment ID
Product ID
Quantity Shipped
Shipment
Shipment date
(1..N)
(N...1)
Address
Supplier Rating
Supplier ID
Supplier
(1..N)
Has Many
Contains
Reorder Point
(N...1)
Supplier ID
SupplierName
Product Name
Stock Level
Category ID
Contact Info
Product ID
(N...1)
Belongs
Product
Price
(1...N)
Has Many
Unit Price
Total Price
(N...1)
Applied
Quantity
(1...N)
(N...1)
Transaction ID
Sales Transactions
Order ID
Managed
Category Name
Category
Product ID
Category ID
(1...N)
Payment ID
Has Many
Order ID
(1..N)
Description
(1..N)
(1..1)
Payment Method
Inventory
Order
Linked
Payment
(N...1)
Amount Paid
Inventory Id
Placed
TotalAmount
Transaction Date
Payment Status
Order Date
Customer
Product ID
NextStockDate
OrderStatus
Address
Customer ID
Stock Level
LastRestockDate
Customer ID
Email
Customer Name
Order ID
Phone Number
Loyalty Points

- **UML Model**


Customer
+ CustomerID: int
+ CustomerName: string
+ Email: string
+ PhoneNumber: string
+ Address: string
+ LoyaltyPoints: int
+ getCustomerDetails(): string
Shipment
+ ShipmentID: int
+ ShipmentDate: Date
+ QuantityShipped: int
+ ExpectedArrivalDate: Date
+ trackShipment(): string
Employee
+ EmployeeID: int
+ EmployeeName: string
+ Position: string
+ HireDate: Date
+ getEmployeeInfo(): string
contains
places
works at
Product
+ ProductID: int
+ ProductName: string
+ Price: float
+ StockLevel: int
+ ReorderPoint: int
+ getDetails(): string
+ checkStock(): bool
shipped by
Order
+ OrderID: int
+ OrderDate: Date
+ OrderStatus: string
+ TotalAmount: float
+ placeOrder(): void
+ cancelOrder(): void
StoreLocation
+ StoreID: int
+ StoreName: string
+ Address: string
+ getStoreDetails(): string
tracked by
belongs to
supplied by
part of
includes
linked to
Inventory
+ InventoryID: int
+ StockLevel: int
+ LastRestockDate: Date
+ checkInventory(): bool
+ updateStock(): void
Category
+ CategoryID: int
+ CategoryName: string
+ getCategoryInfo(): string
Supplier
+ SupplierID: int
+ SupplierName: string
+ ContactInfo: string
+ Address: string
+ SupplierRating: float
+ getSupplierInfo(): string
SalesTransaction
+ TransactionID: int
+ Quantity: int
+ UnitPrice: float
+ TotalPrice: float
+ processTransaction(): void
Payment
+ PaymentID: int
+ PaymentMethod: string
+ PaymentStatus: string
+ AmountPaid: float
+ processPayment(): void

Project Milestone - Logical Model (Relational Model)

Step 1: Identify Entities and Attributes In this step, we list each entity from the EER diagram and its corresponding attributes.

Entities and Their Attributes:

1. Category
   - Attributes: Category_ID (PK), Category Name, Description
2. Product
   - Attributes: Product_ID (PK), Product_Name, Stock_Level, Price, Reorder Point, Category_ID(FK), Supplier_ID (FK), Stock_Level, Price, Reorder Point
3. Supplier
   - Attributes: Supplier_ID (PK), Supplier_Name, Contact_Info, Address, Supplier Rating
4. Customer
   - Attributes: Customer_ID (PK), Customer_Name, Loyalty Point, Phone Number, Address, Email
5. Order
   - Attributes: Order_ID (PK), Customer_ID(FK), Order_Date, Total_Amount, Order_Status
6. Sales Transaction
   - Attributes: Transaction_ID (PK), Order_ID (FK), Product_ID (FK), Quantity, Unit_Price, Total Price
7. Store Location
   - Attributes: Store_ID (PK), Adddress, Store Name, Opening Dated, Manager_ID (FK)
8. Shipment
   - Attributes: Shipment_ID (PK), Product_ID(FK), Supplier_ID (FK), Shipment_Date, Quantity Shipped, Expected Arrival Date
9. Employee
   - Attributes: Employee_ID (PK), Position, Employee Name, Store_ID (FK), Hire Date
10. Inventory

- Attributes: Inventory_ID (PK), Product_ID (FK), Stock Level, Last Restock Date, Next Restock Date

11. Payment
    - Attributes: Payment_ID (PK), Order_ID (FK), Payment Method, Transcation Date, Payment Status, Amount Paid

Step 2: Define Relationships Between Entities

The relationships connect entities according to the EER design. Below are descriptions of each relationship with cardinalities and explanations.

Relationships:

1. Category and Product

Assumption: Each product belongs to one category, but a category can have multiple products.

Cardinality: One-to-Many (1:N)

Explanation: This relationship allows multiple products to be associated with a single category. The Category_ID in the Product table is a foreign key referencing the Category table.

2. Product and Supplier

Assumption: Each product is primarily supplied by one supplier, but a supplier can supply multiple products.

Cardinality: One-to-Many (1:N)

Explanation: Suppliers may offer various products, but each product is associated with one primary supplier for tracking purposes. The Supplier_ID in the Product table is a foreign key referencing the Supplier table.

3. Product and Sales Transaction

Assumption: A product can be sold in multiple transactions, and each transaction involves one product.

Cardinality: One-to-Many (1:N)

Explanation: Each sales transaction records the quantity and price of a single product being sold. The Sales Transaction table has a foreign key Product_ID that links to the Product table, establishing a one-to-many relationship.

4. Customer and Order

Assumption: A customer can place multiple orders, but each order is linked to only one customer.

Cardinality: One-to-Many (1:N)

Explanation: Customers can make multiple purchases over time. Each order is recorded with a unique Order_ID in the Order table, which contains a foreign key Customer_ID referencing the Customer table.

5. Order and Sales Transaction

Assumption: Each order can contain multiple items (transactions), and each transaction represents a single product in a specific quantity.

Cardinality: One-to-Many (1:N)

Explanation: Orders can contain multiple products. The Sales Transaction table links each line item in an order to a specific Order_ID in the Order table, enabling detailed tracking of all items within an order.

6. Store Location and Employee

Assumption: Each store has multiple employees, but each employee is assigned to only one store.

Cardinality: One-to-Many (1:N)

Explanation: Employees are assigned to specific stores, and the Employee table has a foreign key Store_ID that references the Store Location table. This ensures that each employee is associated with a particular store.

7. Store Location and Order

Assumption: Each order is linked to one store location where the transaction was made.

Cardinality: One-to-Many (1:N)

Explanation: Orders are placed at specific store locations, and each order is associated with only one store. A Store_ID foreign key in the Order table references the Store Location table.

8. Store Location and Inventory

Assumption: Each store has its own inventory tracking system for products.

Cardinality: One-to-Many (1:N)

Explanation: Each store location keeps track of its stock levels and restocking details in the Inventory table. The Store_ID in Inventory links each inventory record to a specific store.

9. Product and Inventory

Assumption: Each product is tracked individually for its stock level at each store location.

Cardinality: One-to-Many (1:N)

Explanation: Each Inventory entry represents stock levels for a specific product at a store. The Product_ID in the Inventory table is a foreign key that references the Product table.

10. Supplier and Shipment

Assumption: Each shipment is sent by one supplier but can involve multiple products.

Cardinality: One-to-Many (1:N)

Explanation: Suppliers fulfill shipments, which can vary in products and quantities. Each shipment entry in the Shipment table has a Supplier_ID field, which links to the supplying entity in the Supplier table.

11. Product and Shipment

Assumption: Each shipment contains a specific quantity of a single product from a supplier.

Cardinality: One-to-Many (1:N)

Explanation: Each shipment record tracks a product's delivery details from a supplier. The Shipment table includes Product_ID as a foreign key, linking each shipment entry to the respective product.

12. Order and Payment

Assumption: Each order is linked to one payment transaction, but payments are specific to individual orders.

Cardinality: One-to-One (1:1)

Explanation: Each order has one corresponding payment entry recorded in the Payment table. The Order_ID in the Payment table ensures that each payment is uniquely associated with a single order.

Step 3: Convert Entities to Relational Tables

This is the relational model representation for each entity with all primary and foreign keys established.

Relational Tables:

1. Category(Category_ID (PK), Category_Name, Description)

Explanation: Stores general information about each category. Category_ID is the primary key.

2. Product(Product_ID (PK), Product_Name, Stock_Level, Price, Reorder_Point, Category_ID (FK) → Category(Category_ID), Supplier_ID (FK) → Supplier(Supplier_ID))

Explanation: Represents each product with details like stock level, price, and reorder point. Category_ID and Supplier_ID are foreign keys that reference Category and Supplier, respectively.

3. Supplier(Supplier_ID (PK), Supplier_Name, Contact_Info, Address, Supplier_Rating)

Explanation: Holds details of each supplier. Supplier_ID is the primary key.

4. Customer(Customer_ID (PK), Customer_Name, Loyalty_Point, Phone_Number, Address, Email)

Explanation: Represents customer information, with Customer_ID as the primary key.

5. Order(Order_ID (PK), Customer_ID (FK) → Customer(Customer_ID), Order_Date, Total_Amount, Order_Status)

Explanation: Stores order details, linking each order to a customer. Customer_ID is a foreign key that references Customer.

6. Sales Transaction(Transaction_ID (PK), Order_ID (FK) → Order(Order_ID), Product_ID (FK) → Product(Product_ID), Quantity, Unit_Price, Total_Price)

Explanation: Represents individual items within an order. Each transaction links to a specific order and product through Order_ID and Product_ID foreign keys.

7. Store Location(Store_ID (PK), Address, Store_Name, Opening_Date, Manager_ID (FK) → Employee(Employee_ID))

Explanation: Contains details of store locations. Manager_ID links to the manager assigned to the store, referencing the Employee table.

8. Shipment(Shipment_ID (PK), Product_ID (FK) → Product(Product_ID), Supplier_ID (FK) → Supplier(Supplier_ID), Shipment_Date, Quantity_Shipped, Expected_Arrival_Date)

Explanation: Records shipment details for each product delivered by suppliers. Product_ID and Supplier_ID are foreign keys linking to Product and Supplier tables.

9. Employee(Employee_ID (PK), Position, Employee_Name, Store_ID (FK) → Store Location(Store_ID), Hire_Date)

Explanation: Represents employee details, linking each employee to a specific store. Store_ID is a foreign key referencing the Store Location table.

10. Inventory(Inventory_ID (PK), Product_ID (FK) → Product(Product_ID), Stock_Level

Last_Restock_Date, Next_Restock_Date)

Explanation: Tracks product stock levels for each product in inventory. Product_ID links each entry to a product in the Product table.

11. Payment(Payment_ID (PK), Order_ID (FK) → Order(Order_ID), Payment_Method, Transaction_Date, Payment_Status, Amount_Paid)

Explanation: Stores payment information for each order. Order_ID is a foreign key linking payments to orders in the Order table.

Step 4 : Specialization Details

Specialization Type

Type: **Disjoint Specialization**

Definition: In disjoint specialization, each entity can belong to only one specialized subtype at a time. This is appropriate when an entity instance is mutually exclusive to a single subtype.

Specialization for **Employee**

Suppose that the Employee entity is specialized into two subtypes based on job roles: Manager and Salesperson. Each employee can only belong to one role, so this specialization is disjoint.

General Entity: Employee

Subtype Entities: Manager, Salesperson

Relationship: Each subtype (Manager and Salesperson) has a 1:1 relationship with the Employee entity, indicating that each employee is either a manager or a salesperson, but not both.

Attributes:

1) **Employee (General Entity)**
   - **Attributes:** Employee_ID (PK), Position, Employee_Name, Store_ID (FK), Hire_Date
   - Purpose: Holds general information common to all employees, including name, position, and store location.

2) **Manager (Subtype of Employee)**
   - **Attributes:** Employee_ID (PK, FK) → Employee(Employee_ID), Manager_Rating, Department
   - **Purpose:** Contains attributes specific to managers, such as rating and department, with a primary key that also serves as a foreign key linking it to the Employee table.

3) **Salesperson (Subtype of Employee)**
   - **Attributes:** Employee_ID (PK, FK) → Employee(Employee_ID), Sales_Target, Commission
   - **Purpose:** Contains attributes unique to salespersons, including sales target and commission information. Employee_ID acts as both a primary and foreign key, linking to the Employee table.

Specialization for **Payment**

If Payment is specialized into different payment methods, such as Credit_Card, Cash, and Bank_Transfer, this is also a disjoint specialization, where each payment can belong to only one subtype.

General Entity: Payment

Subtype Entities: Credit_Card, Cash, Bank_Transfer

Relationship: Each subtype has a 1:1 relationship with the Payment entity, meaning each payment record will be associated with only one specific payment method.

Attributes:

4) **Payment (General Entity)**
   - **Attributes:** Payment_ID (PK), Order_ID (FK), Payment_Method, Transaction_Date, Payment_Status, Amount_Paid
   - **Purpose:** Stores common payment details for all payment methods, such as transaction date, status, and amount paid.

5) Credit_Card (Subtype of Payment)

- **Attributes:** Payment_ID (PK, FK) → Payment(Payment_ID), Card_Number, Card_Holder_Name, Expiration_Date
- **Purpose:** Holds credit card-specific information, such as card number and holder's name, with Payment_ID linking to the Payment table.

6) Cash (Subtype of Payment)

- **Attributes:** Payment_ID (PK, FK) → Payment(Payment_ID), Cash_Received_By, Change_Given
- **Purpose:** Stores details relevant to cash transactions, such as the person receiving payment and any change returned.

7) Bank_Transfer (Subtype of Payment)

- **Attributes:** Payment_ID (PK, FK) → Payment(Payment_ID), Bank_Name, Account_Number, Transaction_Reference
- **Purpose:** Contains details specific to bank transfers, including bank name, account number, and transaction reference.

Step 5 : Normalization:

1. Category

Columns: Category_ID (PK), Category_Name, Description

Explanation:

1NF: Each attribute contains atomic values (single values per field).

2NF: Since Category_ID is the only primary key, all other attributes (Category_Name and Description) are fully dependent on Category_ID.

3NF: There are no transitive dependencies, as Category_Name and Description depend only on Category_ID.

Conclusion: The Category table is in 3NF.

2. Product

Columns: Product_ID (PK), Product_Name, Stock_Level, Price, Reorder_Point, Category_ID (FK), Supplier_ID (FK)

Explanation:

1NF: All attributes are atomic (e.g., Product_Name is a single value).

2NF: Each non-key attribute (Product_Name, Stock_Level, Price, Reorder_Point) is fully dependent on Product_ID, the primary key.

3NF: No transitive dependencies are present, as each non-key attribute depends only on Product_ID.

Conclusion: The Product table is in 3NF.

3. Supplier

Columns: Supplier_ID (PK), Supplier_Name, Contact_Info, Address, Supplier_Rating

Explanation:

1NF: Each attribute contains atomic values (e.g., Contact_Info stores a single contact detail).

2NF: All attributes are fully dependent on the primary key Supplier_ID.

3NF: No transitive dependencies are present, so each non-key attribute depends only on Supplier_ID.

Conclusion: The Supplier table is in 3NF.

4. Customer

Columns: Customer_ID (PK), Customer_Name, Loyalty_Point, Phone_Number, Address

Email

Explanation:

1NF: Attributes are atomic, containing only single values.

2NF: Each non-key attribute (e.g., Loyalty_Point, Phone_Number) depends fully on Customer_ID.

3NF: No transitive dependencies, as each non-key attribute is dependent only on Customer_ID.

Conclusion: The Customer table is in 3NF.

5. Order

Columns: Order_ID (PK), Customer_ID (FK), Order_Date, Total_Amount, Order_Status

Explanation:

1NF: All columns contain atomic values (e.g., Order_Date contains a single date value).

2NF: All non-key attributes are fully dependent on Order_ID.

3NF: No transitive dependencies, so each attribute is only dependent on Order_ID.

Conclusion: The Order table is in 3NF.

6. Sales Transaction

Columns: Transaction_ID (PK), Order_ID (FK), Product_ID (FK), Quantity, Unit_Price, Total_Price

Explanation:

1NF: Attributes are atomic, with each column containing a single value.

2NF: Each non-key attribute depends entirely on the primary key Transaction_ID.

3NF: No transitive dependencies, as each attribute depends only on Transaction_ID.

Conclusion: The Sales Transaction table is in 3NF.

7. Store Location

Columns: Store_ID (PK), Address, Store_Name, Opening_Date, Manager_ID (FK)

Explanation:

1NF: Each attribute contains atomic values (e.g., Address is a single address).

2NF: Each attribute is fully dependent on the primary key Store_ID.

3NF: No transitive dependencies, so each non-key attribute depends only on Store_ID.

Conclusion: The Store Location table is in 3NF.

8. Shipment

Columns: Shipment_ID (PK), Product_ID (FK), Supplier_ID (FK), Shipment_Date, Quantity_Shipped, Expected_Arrival_Date

Explanation:

1NF: All columns contain atomic values (e.g., Quantity_Shipped is a single numeric value).

2NF: Each non-key attribute depends fully on the primary key Shipment_ID.

3NF: No transitive dependencies, as each attribute depends only on Shipment_ID.

Conclusion: The Shipment table is in 3NF.

9. Employee

Columns: Employee_ID (PK), Position, Employee_Name, Store_ID (FK), Hire_Date

Explanation:

1NF: Attributes contain atomic values (e.g., Employee_Name is a single name).

2NF: All non-key attributes depend fully on the primary key Employee_ID.

3NF: No transitive dependencies, so each non-key attribute depends only on Employee_ID.

Conclusion: The Employee table is in 3NF.

10. Inventory

Columns: Inventory_ID (PK), Product_ID (FK), Stock_Level, Last_Restock_Date, Next_Restock_Date

Explanation:

1NF: Each attribute is atomic (e.g., Stock_Level is a single value).

2NF: Each non-key attribute depends fully on Inventory_ID.

3NF: No transitive dependencies exist, as each attribute depends only on Inventory_ID.

Conclusion: The Inventory table is in 3NF.

11. Payment

Columns: Payment_ID (PK), Order_ID (FK), Payment_Method, Transaction_Date, Payment_Status, Amount_Paid

Explanation:

1NF: All columns contain atomic values.

2NF: Each attribute depends fully on the primary key Payment_ID.

3NF: No transitive dependencies exist, as each attribute depends only on Payment_ID.

Conclusion: The Payment table is in 3NF

Project Milestone - Implementation in MySQL

Create Tables for the entities:

--Create Category Table

```
CREATE TABLE Category (
    Category_ID INT PRIMARY KEY,
    Category_Name VARCHAR(255),
    Description TEXT
);
```

-- Create Product Table

```
CREATE TABLE Product (
    Product_ID INT PRIMARY KEY,
    Product_Name VARCHAR(255),
    Stock_Level INT,
    Price DECIMAL(10, 2),
    Reorder_Point INT,
    Category_ID INT,
    Supplier_ID INT,
    FOREIGN KEY (Category_ID) REFERENCES Category(Category_ID),
    FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID)
);
```

-- Create Supplier Table

```
CREATE TABLE Supplier (
    Supplier_ID INT PRIMARY KEY,
    Supplier_Name VARCHAR(255),
    Contact_Info VARCHAR(255),
    Address TEXT,
    Supplier_Rating DECIMAL(3, 2)
);
```

-- Create Customer Table

```
CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY,
    Customer_Name VARCHAR(255),
    Loyalty_Point  INT,
    Phone_Number VARCHAR(15),
    Address TEXT,
    Email VARCHAR(255)
);
```

-- Create Order Table

```
CREATE TABLE Order (
   Order_ID INT PRIMARY KEY,
   Customer_ID  INT,
   Order_Date DATE,
   Total_Amount DECIMAL(10, 2),
   Order_Status VARCHAR(50),
   FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);
```

-- Create Sales Transaction Table

```
CREATE TABLE Sales_Transaction (
   Transaction_ID INT PRIMARY KEY,
   Order_ID INT,
   Product_ID INT,
   Quantity INT,
   Unit_Price DECIMAL(10, 2),
   Total_Price DECIMAL(10, 2),
   FOREIGN KEY (Order_ID) REFERENCES Order(Order_ID),
   FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);
```

-- Create Store Location Table

```
CREATE TABLE Store_Location (
    Store_ID INT PRIMARY KEY,
    Address TEXT,
    Store_Name VARCHAR(255),
    Opening_Date DATE,
    Manager_ID INT
);
```

-- Create Shipment Table

```
CREATE TABLE Shipment (
    Shipment_ID INT PRIMARY KEY,
    Product_ID INT,
    Supplier_ID INT,
    Shipment_Date DATE,
    Quantity_Shipped INT,
    Expected_Arrival_Date DATE,
    FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),
    FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID)
);
```

-- Create Employee Table

```
CREATE TABLE Employee (
    Employee_ID INT PRIMARY KEY,
    Position VARCHAR(50),
    Employee_Name VARCHAR(255),
    Store_ID INT,
    Hire_Date DATE,
    FOREIGN KEY (Store_ID) REFERENCES Store_Location(Store_ID));
```

-- Create Inventory Table

```
CREATE TABLE Inventory (
    Inventory_ID INT PRIMARY KEY,
    Product_ID INT,
    Stock_Level INT,
    Last_Restock_Date DATE,
    Next_Restock_Date DATE,
    FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);
```

-- Create Payment Table

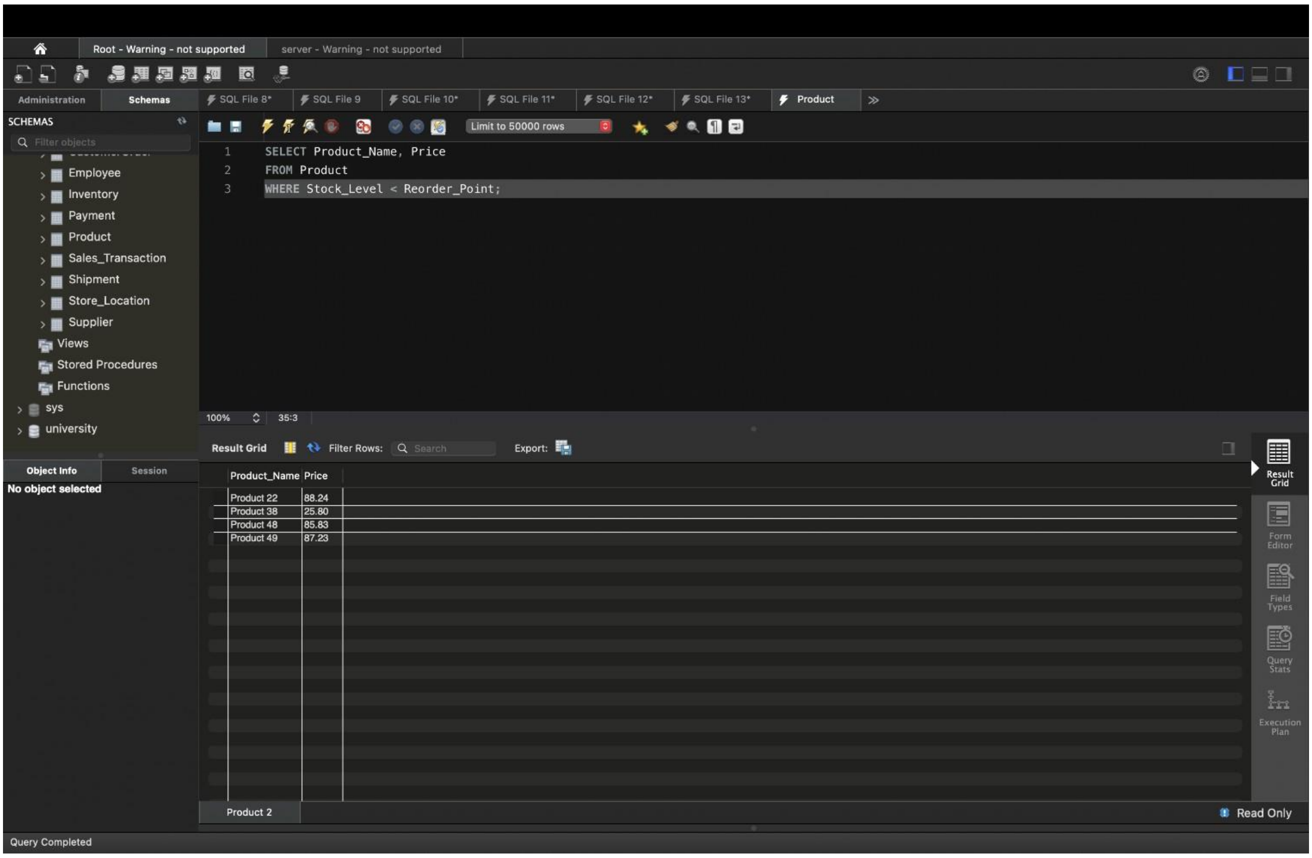
```
CREATE TABLE Payment (
    Payment_ID INT PRIMARY KEY,
    Order_ID INT,
    Payment_Method VARCHAR(50),
    Transaction_Date DATE,
    Payment_Status VARCHAR(50),
    Amount_Paid DECIMAL(10, 2),
    FOREIGN KEY (Order_ID) REFERENCES Order(Order_ID)
);
```

Implementation in MySQL

Queries:

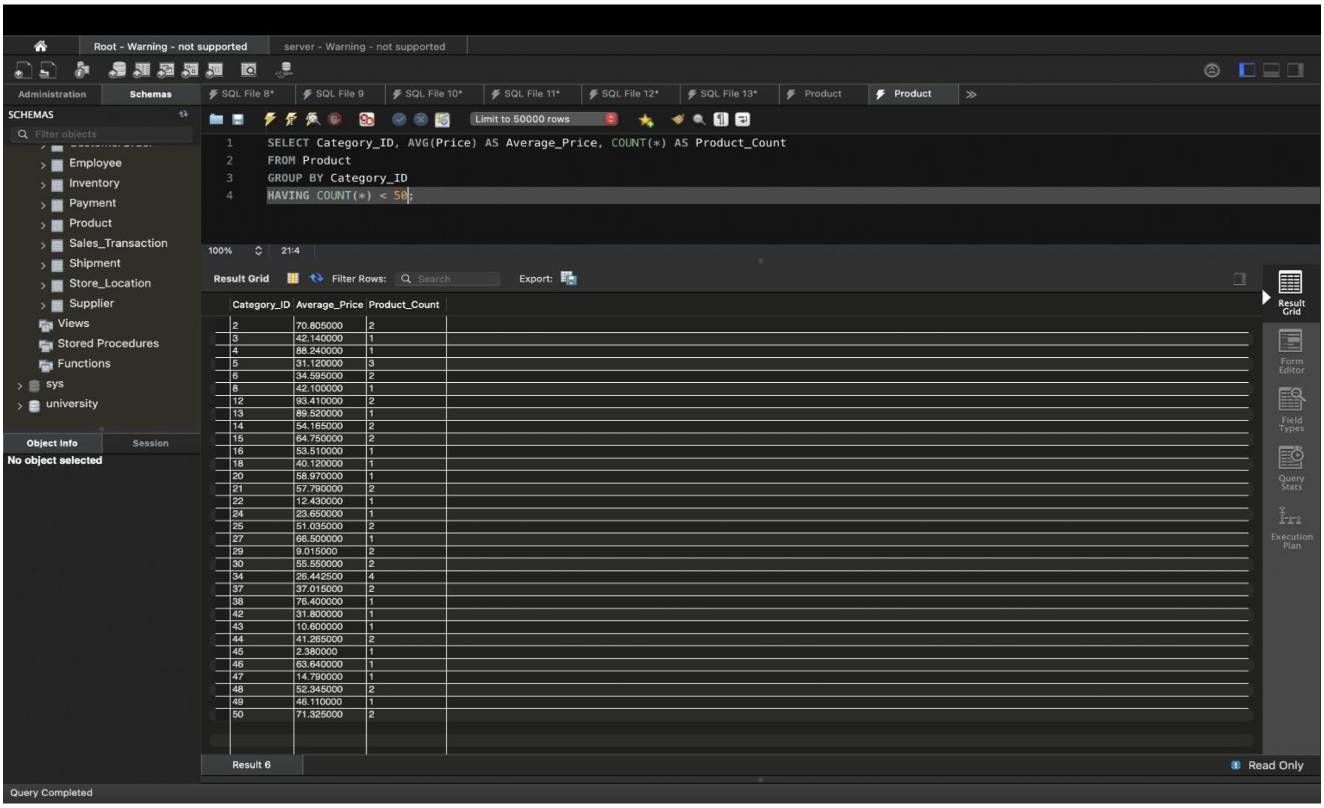
Simple Query:

SELECT Product_Name, Price
FROM Product
WHERE Stock_Level < Reorder_Point;



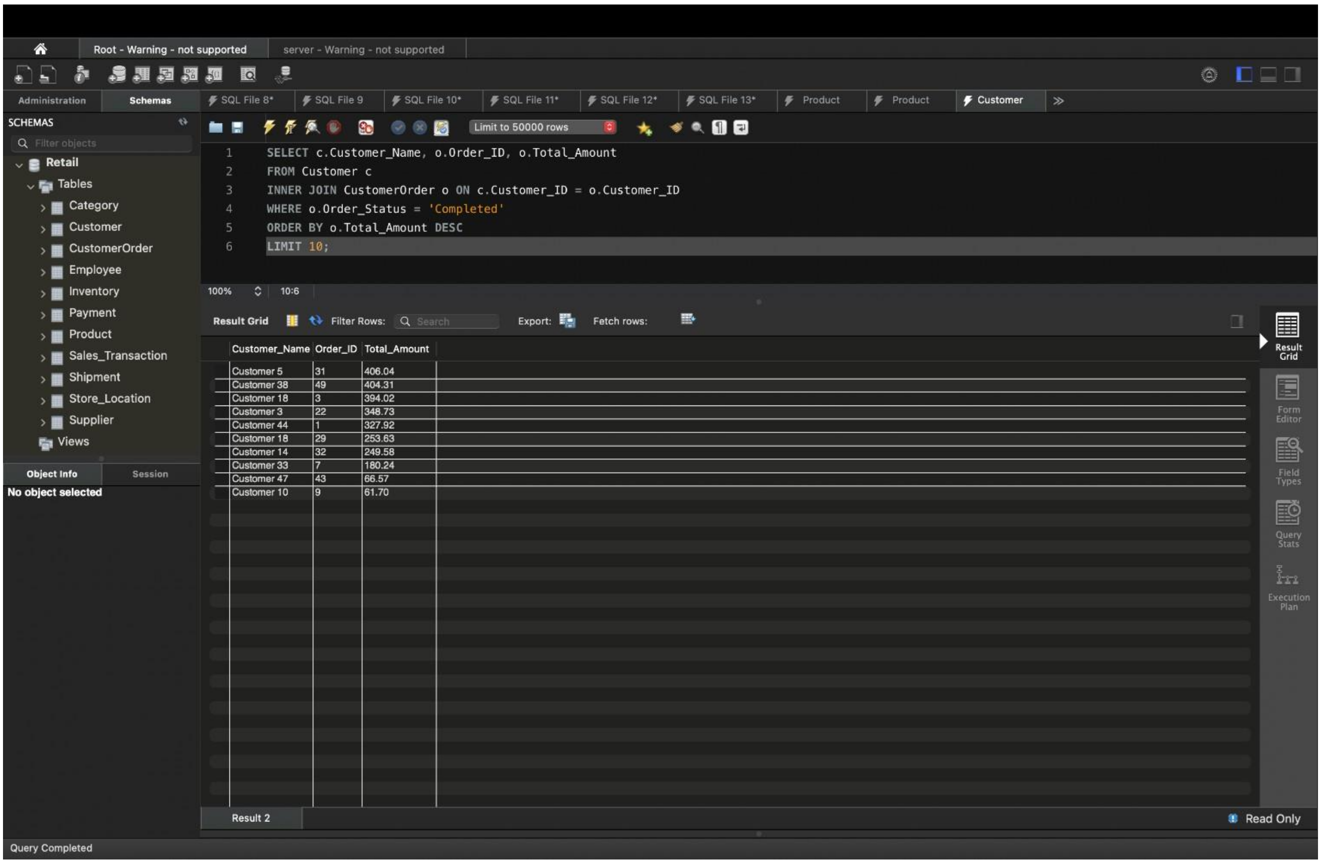
Aggregate Query:

SELECT Category_ID, AVG(Price) AS Average_Price, COUNT(*) AS Product_Count
FROM Product
GROUP BY Category_ID
HAVING COUNT(*) < 50;



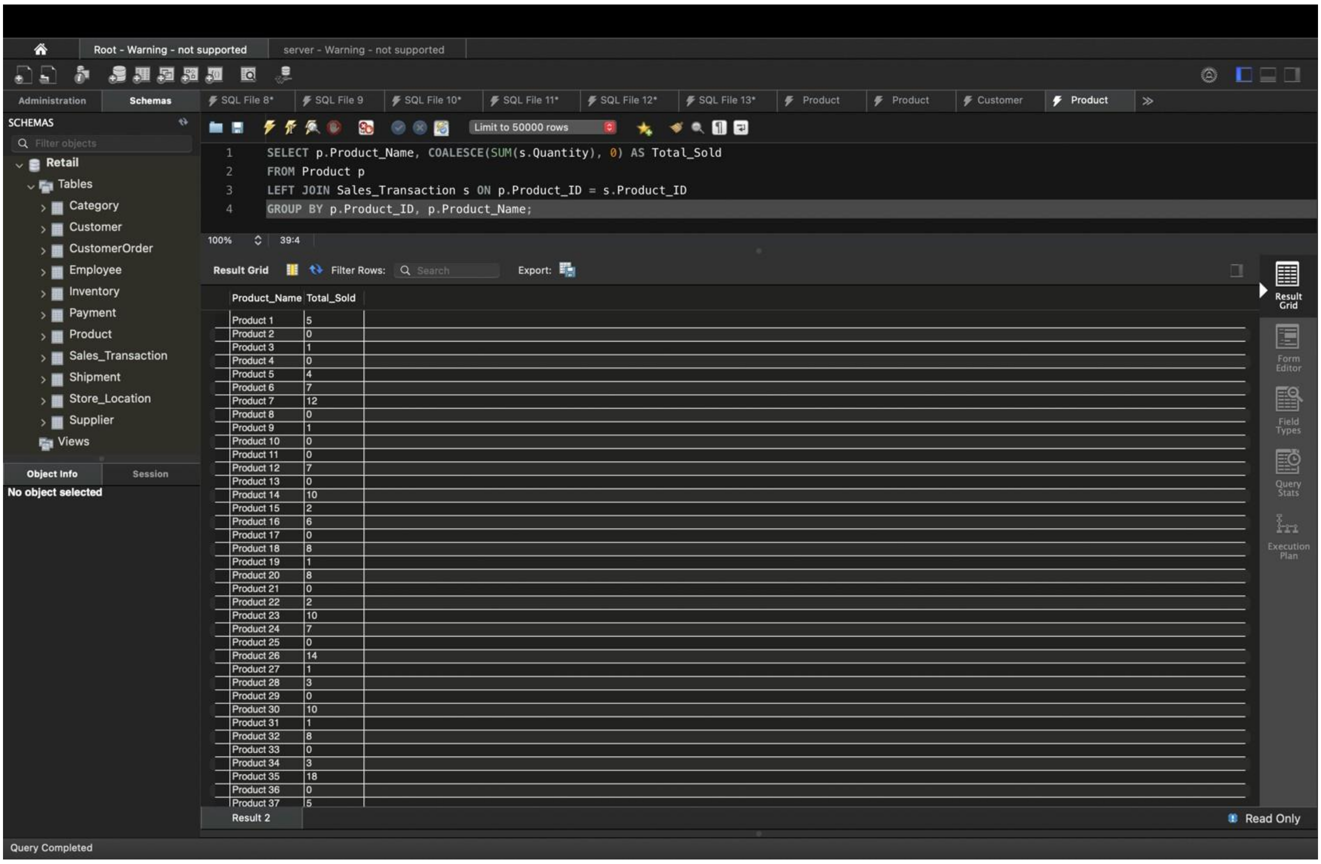
Inner Join:

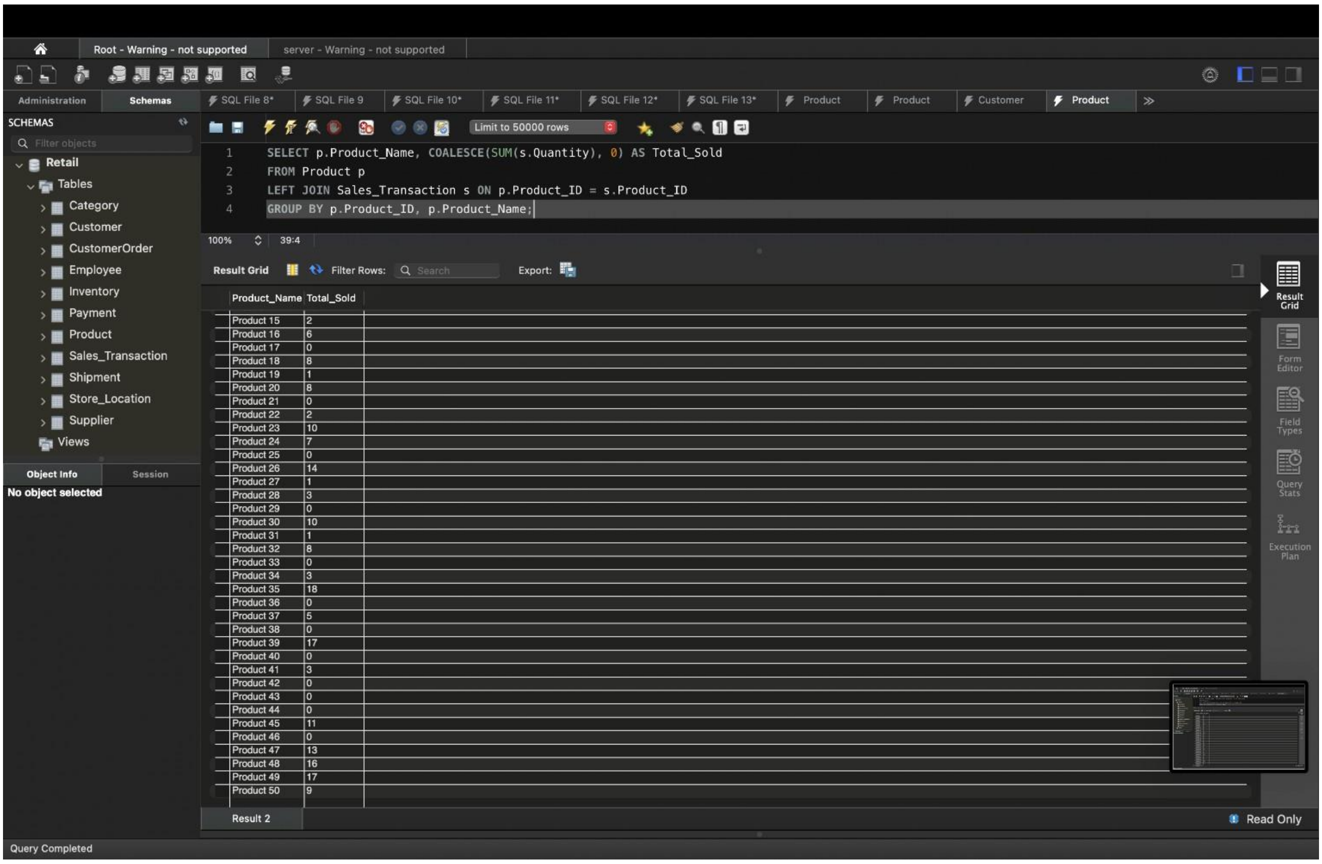
SELECT c.Customer_Name, o.Order_ID, o.Total_Amount
FROM Customer c
INNER JOIN CustomerOrder o ON c.Customer_ID = o.Customer_ID
WHERE o.Order_Status = 'Completed'
ORDER BY o.Total_Amount DESC
LIMIT 10;



Outer Join:

SELECT p.Product_Name, COALESCE(SUM(s.Quantity), 0) AS Total_Sold
FROM Product p
LEFT JOIN Sales_Transaction s ON p.Product_ID = s.Product_ID
GROUP BY p.Product_ID, p.Product_Name;

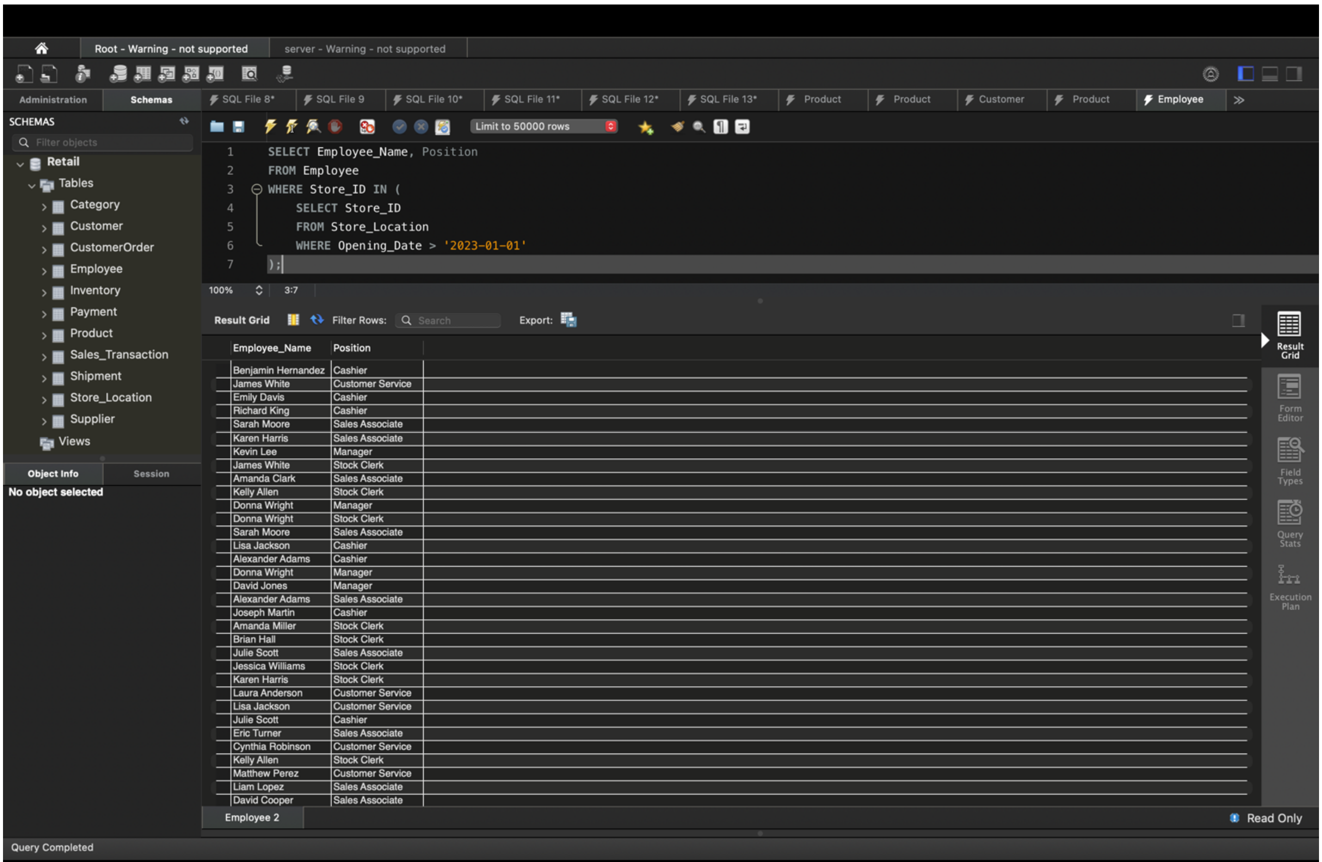


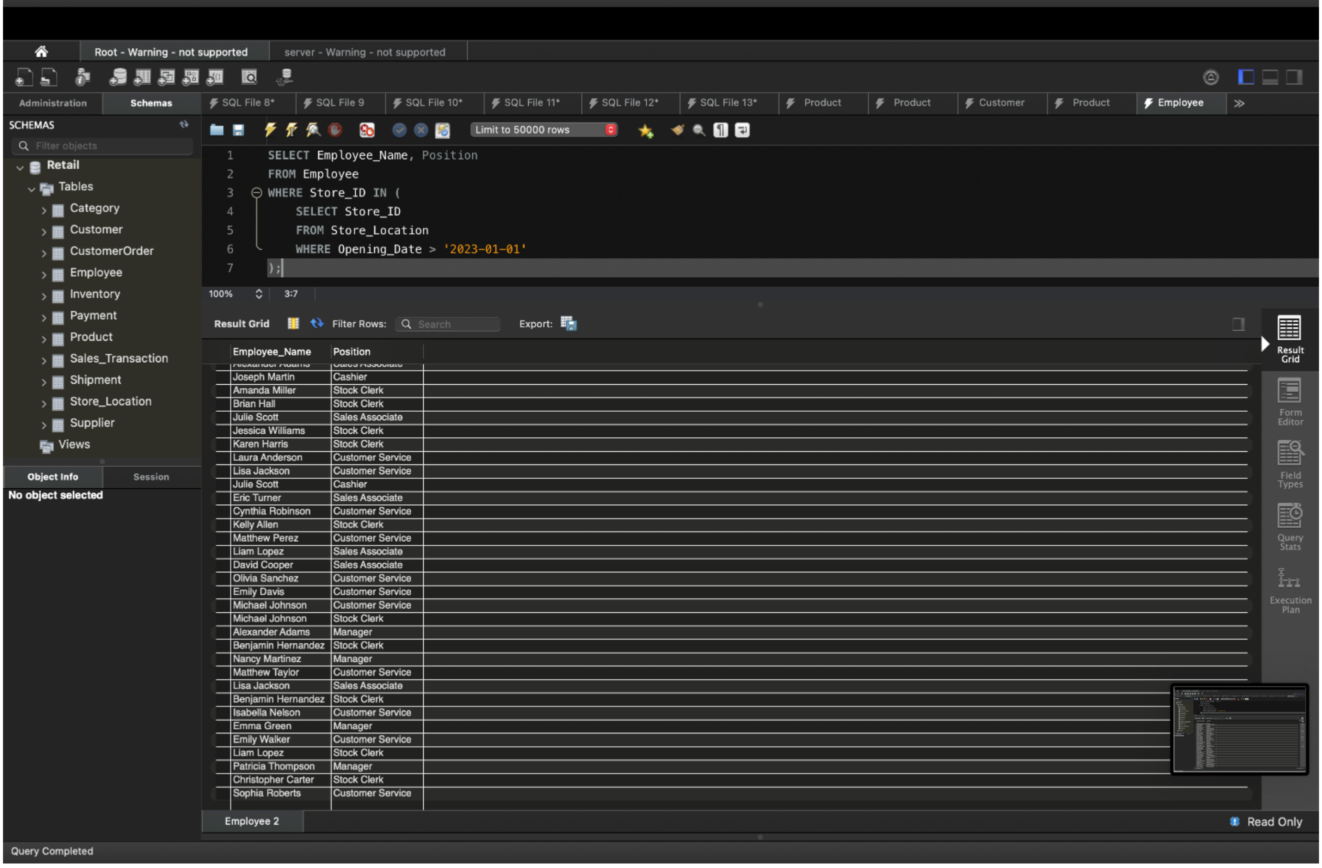


Nested Query:

```
SELECT Employee_Name, Position
FROM Employee
WHERE Store_ID IN (
    SELECT Store_ID
    FROM Store_Location
    WHERE Opening_Date > '2023-01-01'
);
```

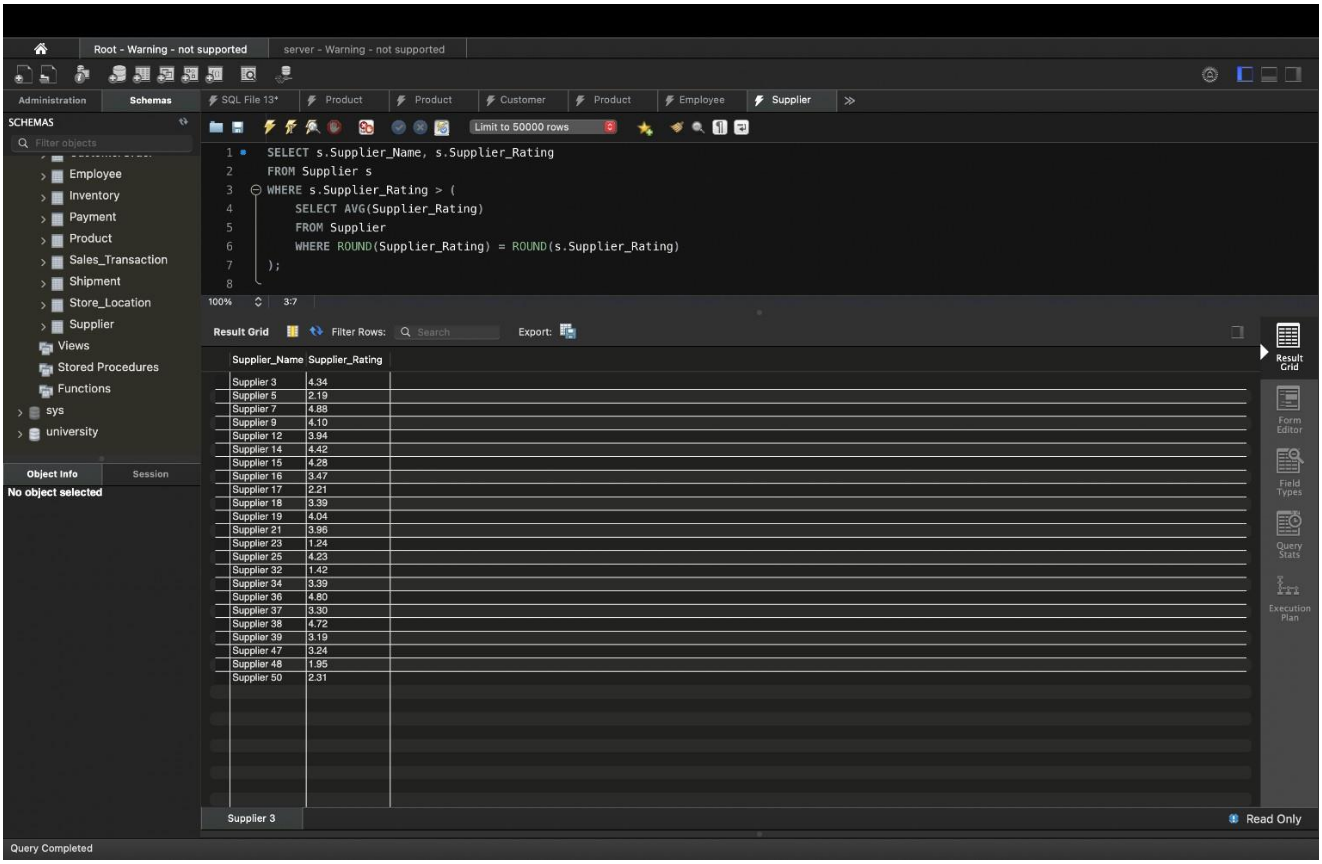
IE 6700 Data Management for Analytics





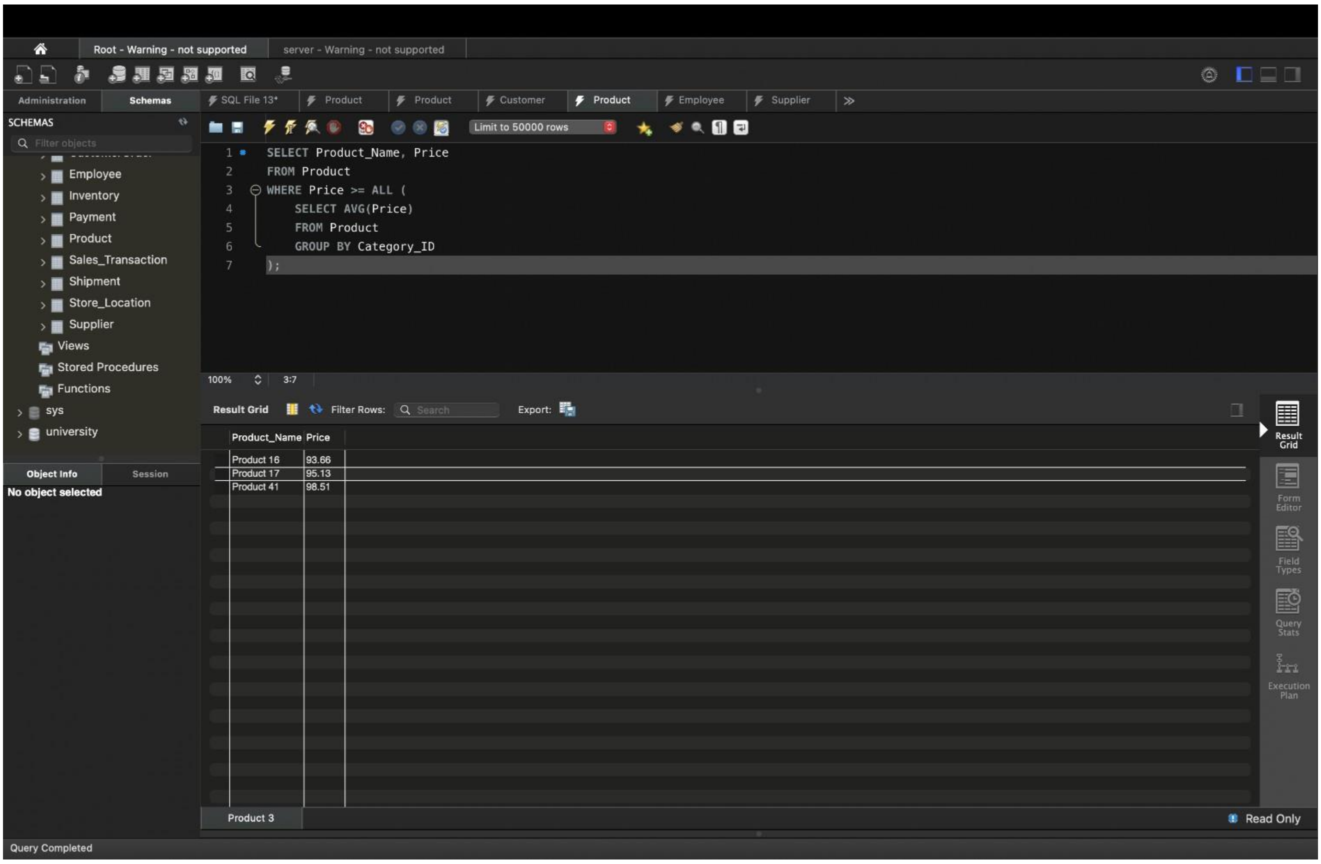
Correlated Query:

```
SELECT s.Supplier_Name, s.Supplier_Rating
FROM Supplier s
WHERE s.Supplier_Rating > (
   SELECT AVG(Supplier_Rating)
   FROM Supplier
   WHERE ROUND(Supplier_Rating) = ROUND(s.Supplier_Rating)
);
```



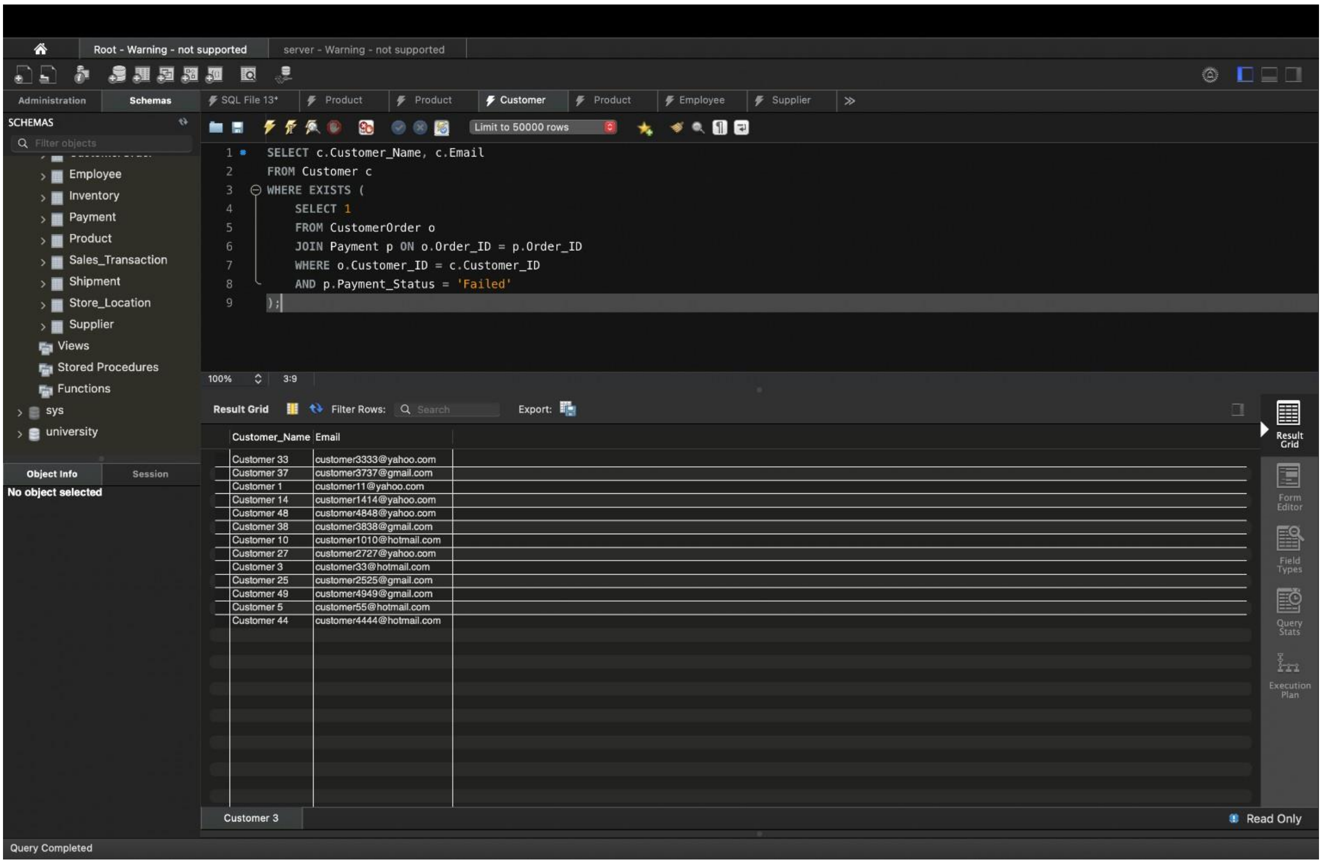
=ALL Query:

```
SELECT Product_Name, Price
FROM Product
WHERE Price >= ALL (
   SELECT AVG(Price)
   FROM Product
   GROUP BY Category_ID
);
```



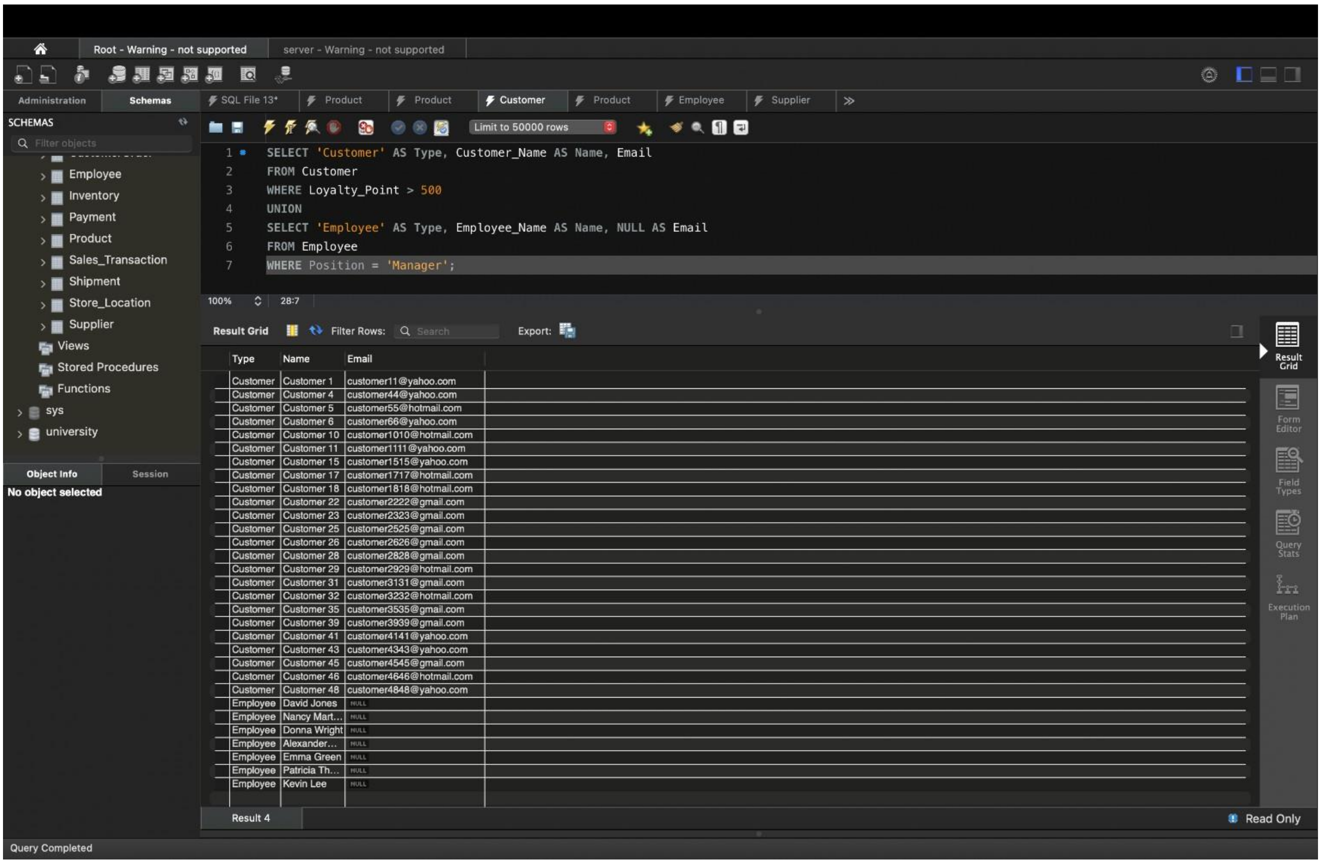
EXISTS Query:

```
SELECT c.Customer_Name, c.Email
FROM Customer c
WHERE EXISTS (
   SELECT 1
   FROM CustomerOrder o
   JOIN Payment p ON o.Order_ID = p.Order_ID
   WHERE o.Customer_ID = c.Customer_ID
   AND p.Payment_Status = 'Failed'
);
```



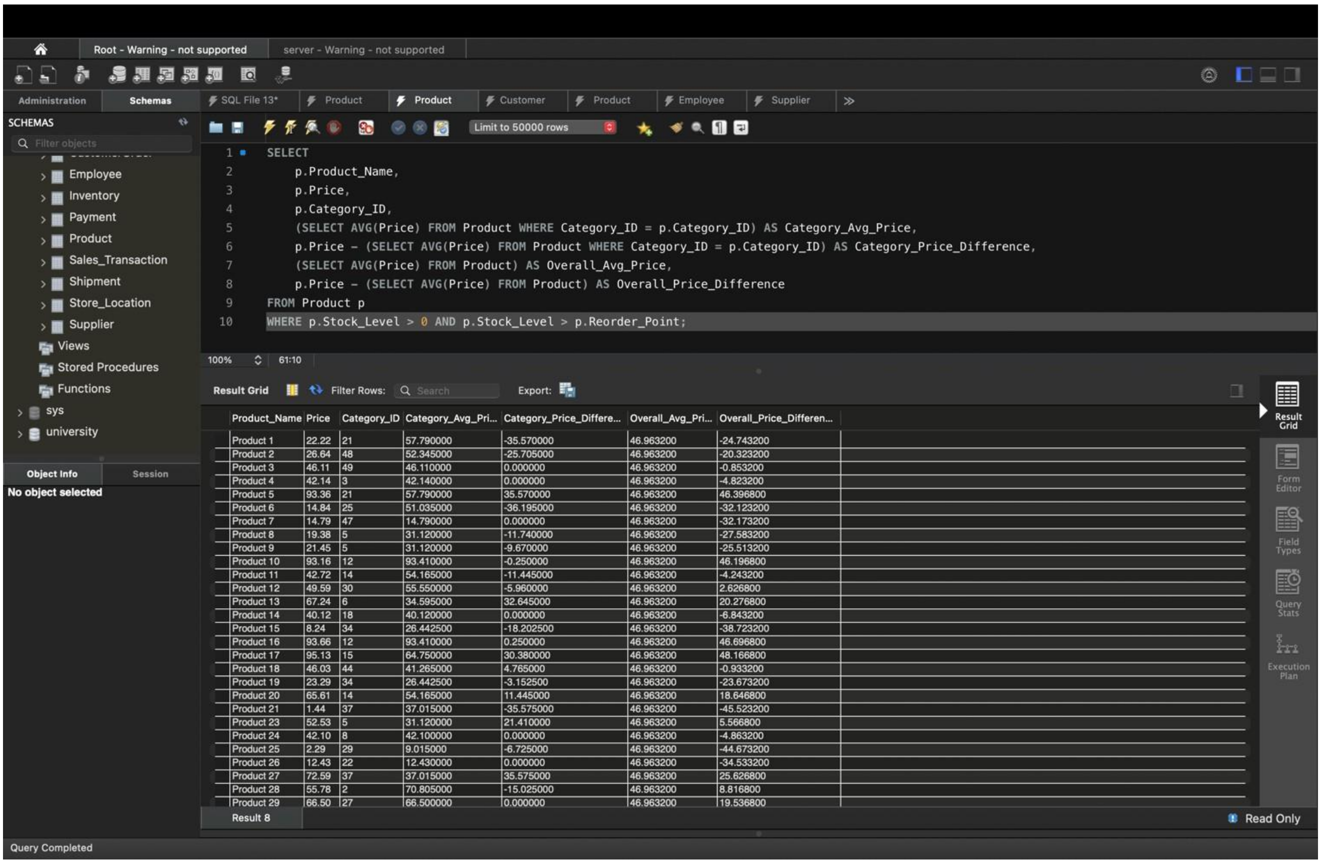
Set Operation (UNION):

SELECT 'Customer' AS Type, Customer_Name AS Name, Email
FROM Customer
WHERE Loyalty_Point > 500
UNION
SELECT 'Employee' AS Type, Employee_Name AS Name, NULL AS Email
FROM Employee
WHERE Position = 'Manager';

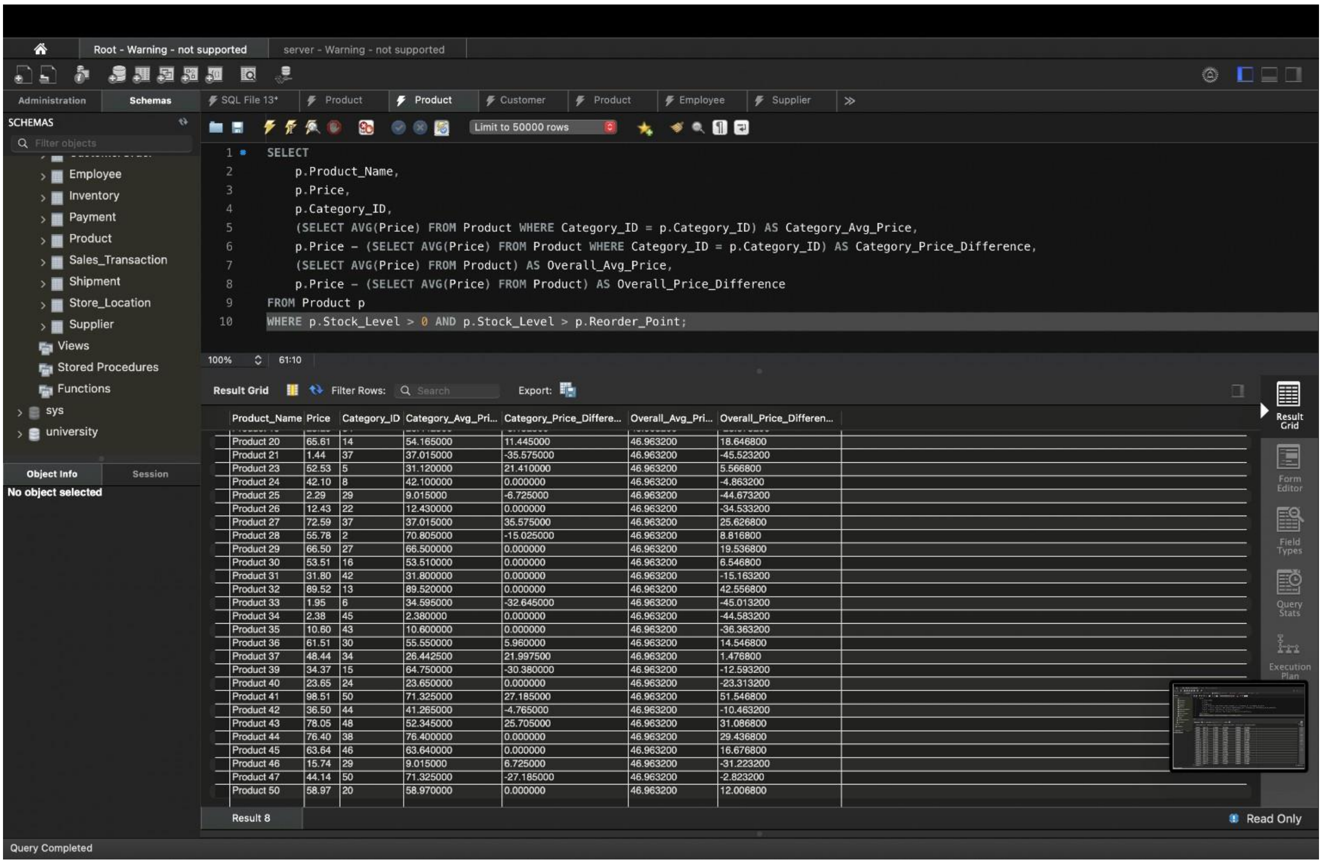


Subquery in SELECT and FROM:

SELECT
p.Product_Name,
p.Price,
p.Category_ID,
(SELECT AVG(Price) FROM Product WHERE Category_ID = p.Category_ID) AS Category_Avg_Price,
p.Price - (SELECT AVG(Price) FROM Product WHERE Category_ID = p.Category_ID) AS Category_Price_Difference,
(SELECT AVG(Price) FROM Product) AS Overall_Avg_Price,
p.Price - (SELECT AVG(Price) FROM Product) AS Overall_Price_Difference
FROM Product p
WHERE p.Stock_Level > 0 AND p.Stock_Level > p.Reorder_Point;



Product_Name	Price	Category_ID	Category_Avg_Pri...	Category_Price_Differe...	Overall_Avg_Pri...	Overall_Price_Differen...
Product 1	22.22	21	57.790000	-35.570000	46.963200	-24.743200
Product 2	26.64	48	52.345000	-25.705000	46.963200	-20.323200
Product 3	46.11	49	46.110000	0.000000	46.963200	-0.853200
Product 4	42.14	3	42.140000	0.000000	46.963200	-4.823200
Product 5	93.36	21	57.790000	35.570000	46.963200	46.396800
Product 6	14.84	25	51.035000	-36.195000	46.963200	-32.123200
Product 7	14.79	47	14.790000	0.000000	46.963200	-32.173200
Product 8	19.38	5	31.120000	-11.740000	46.963200	-27.583200
Product 9	21.45	5	31.120000	-9.670000	46.963200	-25.513200
Product 10	93.16	12	93.410000	-0.250000	46.963200	46.196800
Product 11	42.72	14	54.165000	-11.445000	46.963200	-4.243200
Product 12	49.59	30	55.550000	-5.960000	46.963200	2.626800
Product 13	67.24	6	34.595000	32.645000	46.963200	20.276800
Product 14	40.12	18	40.120000	0.000000	46.963200	-6.843200
Product 15	8.24	34	26.442500	-18.202500	46.963200	-38.723200
Product 16	93.66	12	93.410000	0.250000	46.963200	46.696800
Product 17	95.13	15	64.750000	30.380000	46.963200	48.166800
Product 18	46.03	44	41.265000	4.765000	46.963200	-0.933200
Product 19	23.29	34	26.442500	-3.152500	46.963200	-23.673200
Product 20	65.61	14	54.165000	11.445000	46.963200	18.646800
Product 21	1.44	37	37.015000	-35.575000	46.963200	-45.523200
Product 23	52.53	5	31.120000	21.410000	46.963200	5.566800
Product 24	42.10	8	42.100000	0.000000	46.963200	-4.863200
Product 25	2.29	29	9.015000	-6.725000	46.963200	-44.673200
Product 26	12.43	22	12.430000	0.000000	46.963200	-34.533200
Product 27	72.59	37	37.015000	35.575000	46.963200	25.626800
Product 28	55.78	2	70.805000	-15.025000	46.963200	8.816800
Product 29	66.50	27	66.500000	0.000000	46.963200	19.536800



Database Access via Python

The database is accessed using Python and visualization of analyzed data is shown below.
The connection of MySQL to Python is done using mysql.connector, followed by converting the list into a dataframe using pandas library and using matplotlib to plot the graphs for the analytics.

retail

November 25, 2024

[3]:
```python
import mysql.connector
import pandas as pd
import matplotlib.pyplot as plt



def connect_to_database():
    try:
        conn = mysql.connector.connect(
            host="localhost",
        user="root",
        password="Chinnu@262",
        database="Retail"
        )
        print("Connection successful!")
        return conn
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

# Query 1: Get top 10 products by sales quantity
def get_top_products():
    conn = connect_to_database()
    query = """
    SELECT p.Product_Name, SUM(s.Quantity) as Total_Quantity
    FROM Product p
    JOIN Sales_Transaction s ON p.Product_ID = s.Product_ID
    GROUP BY p.Product_ID, p.Product_Name
    ORDER BY Total_Quantity DESC
    LIMIT 10
    """
    df = pd.read_sql_query(query, conn)
    conn.close()
    return df

# Query 2: Get customer loyalty points distribution
def get_customer_loyalty():
```

```python
    conn = connect_to_database()
    query = """
    SELECT Customer_Name, Loyalty_Point
    FROM Customer
    ORDER BY Loyalty_Point DESC
    """
    df = pd.read_sql_query(query, conn)
    conn.close()
    return df

# Query 3: Get supplier ratings
def get_supplier_ratings():
    conn = connect_to_database()
    query = """
    SELECT Supplier_Name, Supplier_Rating
    FROM Supplier
    """
    df = pd.read_sql_query(query, conn)
    conn.close()
    return df

# Create visualizations
def create_visualizations():
    # Scatter plot: Top 10 products by sales quantity
    top_products = get_top_products()
    plt.figure(figsize=(12, 6))
    plt.scatter(top_products['Product_Name'], top_products['Total_Quantity'])
    plt.title('Top 10 Products by Sales Quantity')
    plt.xlabel('Product Name')
    plt.ylabel('Total Quantity Sold')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

    # Histogram: Customer loyalty points distribution
    customer_loyalty = get_customer_loyalty()
    plt.figure(figsize=(10, 6))
    plt.hist(customer_loyalty['Loyalty_Point'], bins=20, edgecolor='black')
    plt.title('Distribution of Customer Loyalty Points')
    plt.xlabel('Loyalty Points')
    plt.ylabel('Number of Customers')
    plt.show()

    # Pie chart: Supplier ratings
    supplier_ratings = get_supplier_ratings()
    rating_counts = supplier_ratings['Supplier_Rating'].value_counts()
    plt.figure(figsize=(10, 8))
```

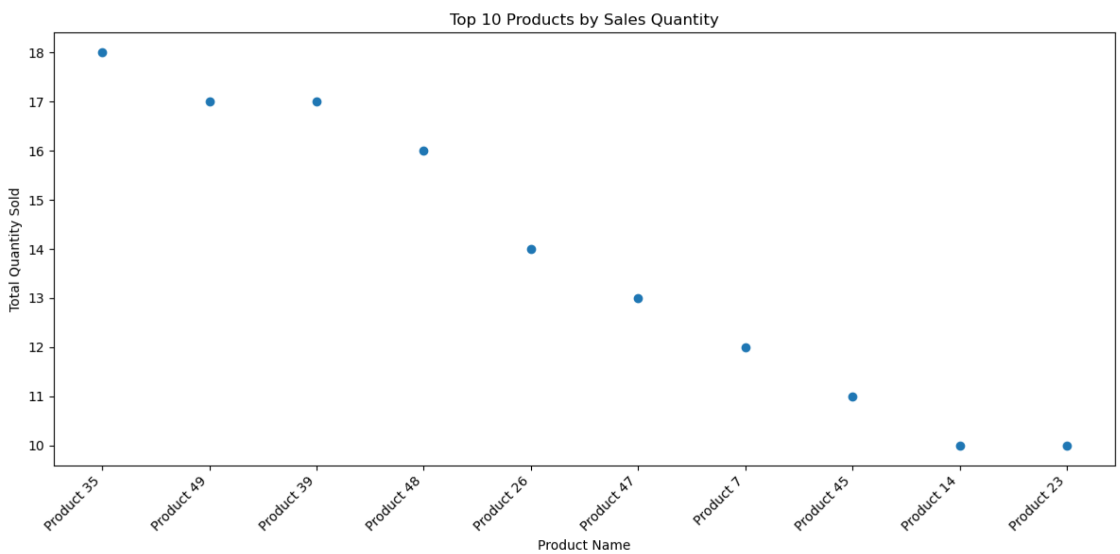
```
    plt.pie(rating_counts.values, labels=rating_counts.index, autopct='%1.
↪1f%%', startangle=90)
    plt.title('Distribution of Supplier Ratings')
    plt.axis('equal')
    plt.show()

# Main function
def main():
    print("Connecting to the Retail database and generating visualizations...")
    create_visualizations()
    print("Visualizations complete.")

if __name__ == "__main__":
    main()
```

```
Connecting to the Retail database and generating visualizations…
Connection successful!
```

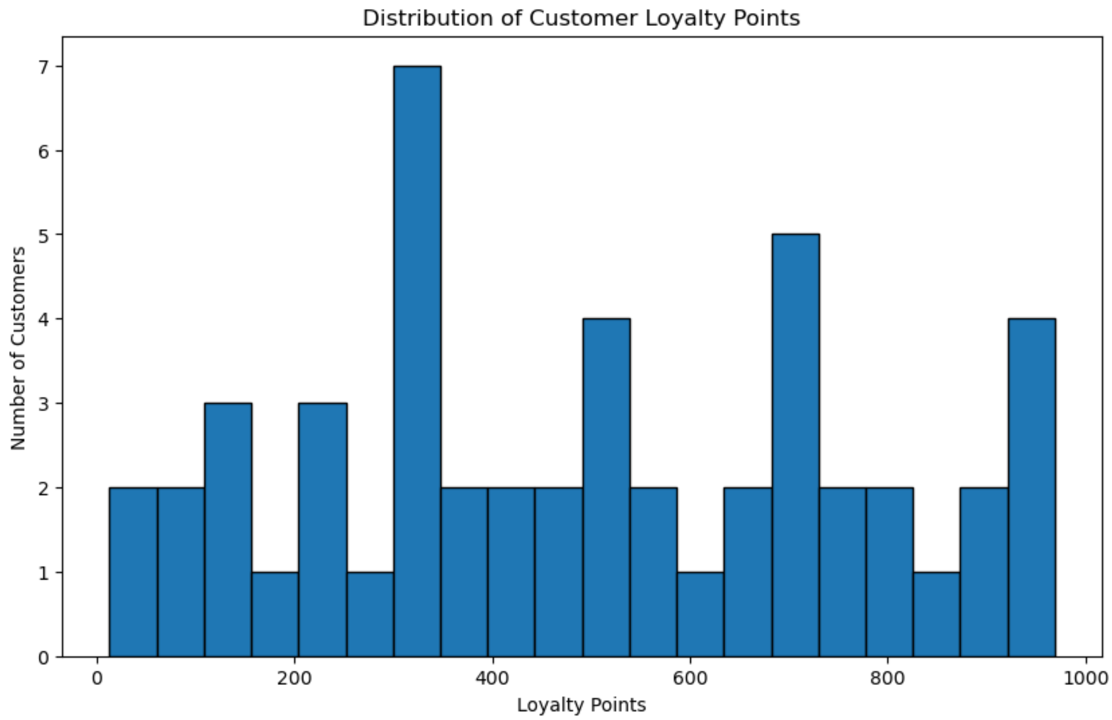
```
/var/folders/dk/ty44k4f97tl78kr03dymv92c0000gn/T/ipykernel_10022/3071242231.py:3
2: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection)
or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are
not tested. Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```




```
Connection successful!
```

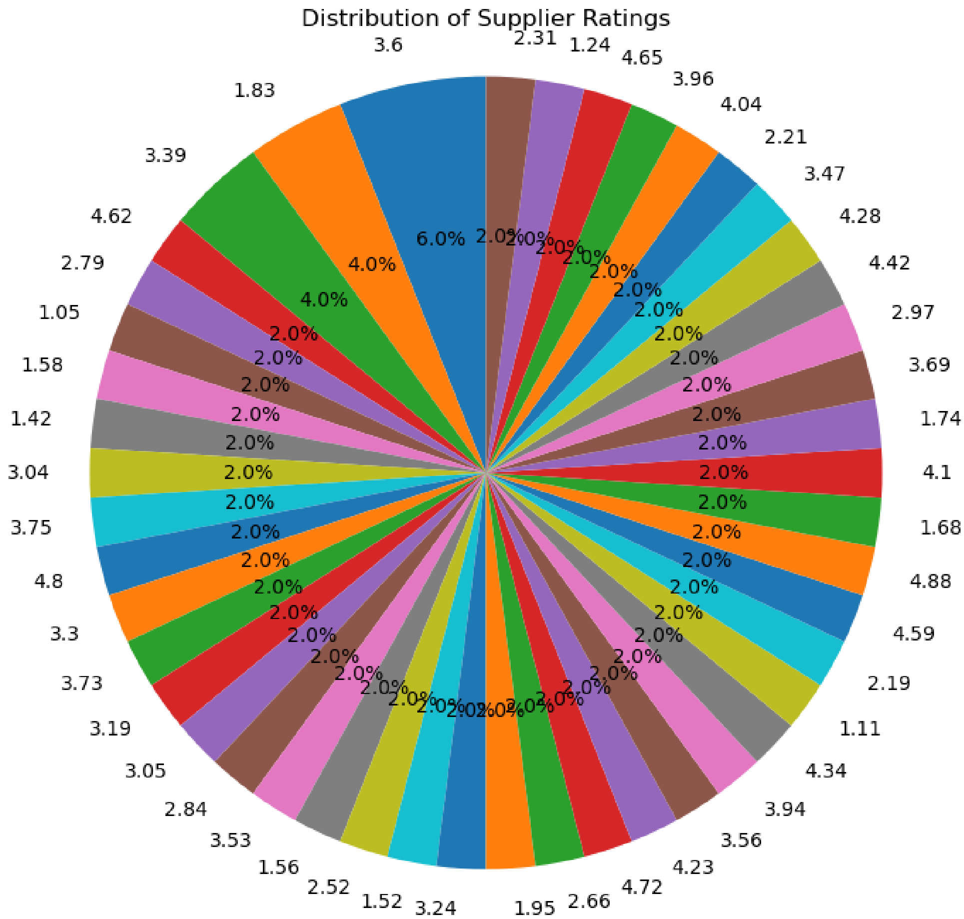
```
/var/folders/dk/ty44k4f97tl78kr03dymv92c0000gn/T/ipykernel_10022/3071242231.py:4
4: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection)
or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are
not tested. Please consider using SQLAlchemy.
```

```
  df = pd.read_sql_query(query, conn)
```




```
/var/folders/dk/ty44k4f97tl78kr03dymv92c0000gn/T/ipykernel_10022/3071242231.py:5
5: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection)
or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are
not tested. Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```

```
Connection successful!
```




```
Visualizations complete.
```

[1]:
```
pip install mysql-connector-python
```

```
Requirement already satisfied: mysql-connector-python in
/opt/anaconda3/lib/python3.12/site-packages (9.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```

```