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:

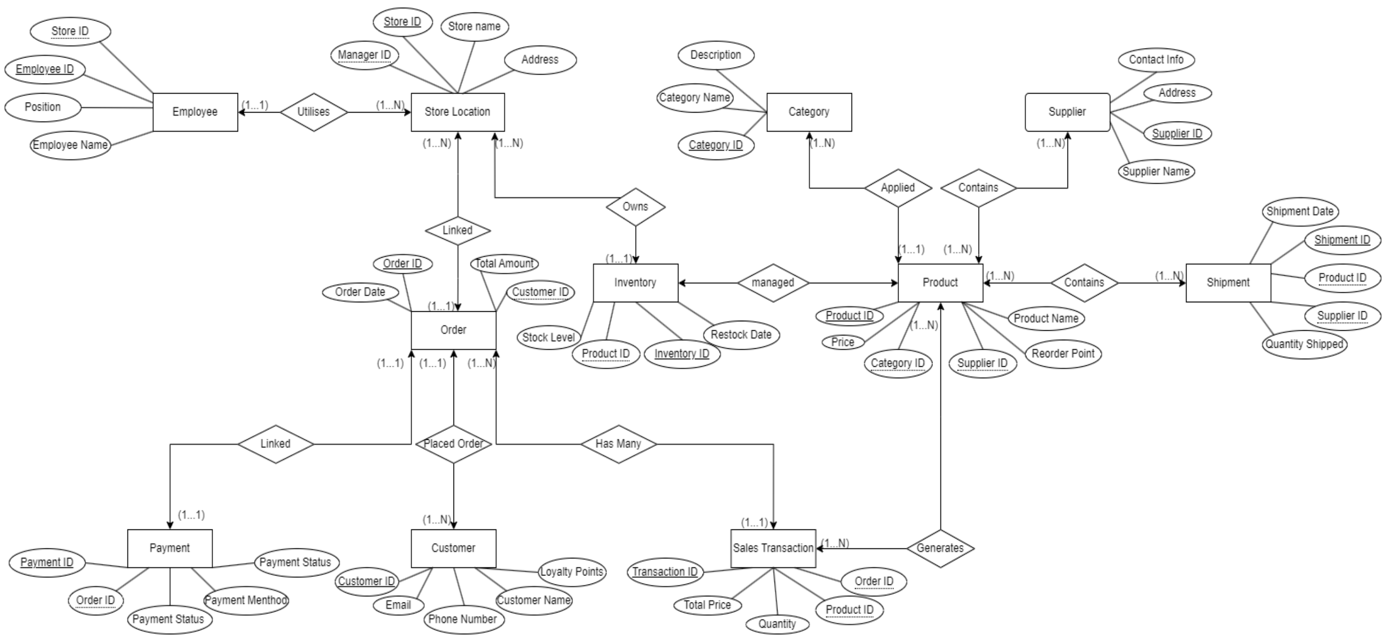
A well-structured Retail Store Inventory Management System is critical to meeting the dynamic demands of modern retail environments. As retail operations expand and customer expectations increase, it becomes essential to accurately track inventory levels, orders, sales, and deliveries. This system must 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

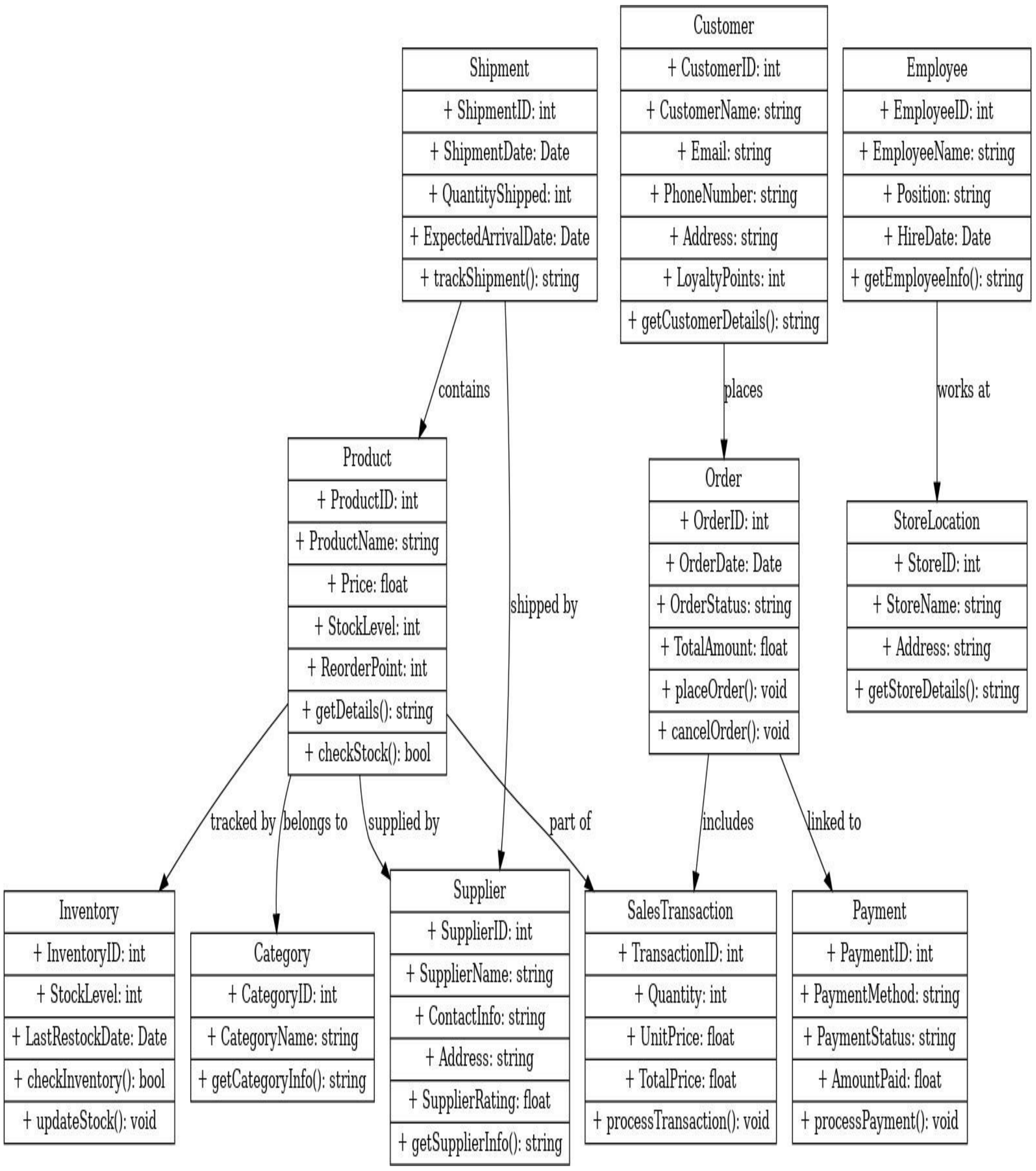
- **Product & Category:** One category has many products, while each product belongs to one category.
- **Product & Supplier**: A product is supplied by one supplier, while a supplier can supply many products.
- **Product & Sales Transaction:** A product can appear in many sales transactions.
- **Customer & 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).
- **Inventory & Store Location:** Each store location holds an inventory of many products.
- **Shipment:** Each shipment involves a specific product and is sent by a supplier.

Conceptual Model (EER and UML Model)

- **EER Model**


Store ID
Employee ID
Position
Employee Name
Employee
(1...1)
Utilises
(1...N)
Store ID
Store name
Manager ID
Address
Store Location
(1...N)
(1...N)
Linked
Owns
Description
Category Name
Category ID
Category
(1..N)
Applied
Contains
Supplier
Contact Info
Address
Supplier ID
Supplier Name
(1...N)
Shipment Date
Shipment ID
Product ID
Supplier ID
Quantity Shipped
Shipment
(1...N)
Contains
(1...N)
Product
(1...1)
(1...N)
Product ID
Price
Category ID
Supplier ID
Product Name
Reorder Point
(1...N)
Inventory
(1...1)
managed
Stock Level
Product ID
Inventory ID
Restock Date
Order ID
Order Date
Total Amount
Customer ID
(1...1)
Order
(1...1)
(1...1)
(1...N)
Linked
Placed Order
Has Many
(1...1)
Payment
Payment ID
Order ID
Payment Status
Payment Menthod
Payment Status
(1...N)
Customer
Customer ID
Email
Phone Number
Customer Name
Loyalty Points
(1...1)
Sales Transaction
(1...N)
Generates
Transaction ID
Total Price
Quantity
Product ID
Order ID

- **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
shipped by
Product
+ ProductID: int
+ ProductName: string
+ Price: float
+ StockLevel: int
+ ReorderPoint: int
+ getDetails(): string
+ checkStock(): bool
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. **Employee to Store Location (Utilizes):**

One store location can have many employees: (1..N)

One employee works at one store location: (1..1)

Final notation: 1:N

2. **Store Location to Order (Linked):**

One store location can have many orders: (1..N)

One order is linked to one store location: (1..1)

Final notation: 1:N

3. **Store Location to Inventory (Owns):**

One store location owns multiple inventory items: (1..N)

One inventory item belongs to one store: (1..1)

Final notation: 1:N

4. **Category to Product (Applied):**

One category can have multiple products: (1..N)

One product belongs to one category: (1..1)

Final notation: 1:N

5. **Supplier to Product (Contains):**

One supplier can supply multiple products: (1..N)

One product can be supplied by multiple suppliers: (1..N)

Final notation: M:N

6. **Product to Shipment (Contains):**

One product can be in multiple shipments: (1..N)

One shipment can contain multiple products: (1..N)

Final notation: M:N

7. **Product to Inventory (Managed):**

One product can be in multiple inventories: (1..N)

One inventory record refers to one product: (1..1)

Final notation: 1:N

8. **Order to Customer (Placed Order):**

One customer can place multiple orders: (1..N)

One order belongs to one customer: (1..1)

Final notation: 1:N

9. **Order to Payment (Linked):**

One order can have one payment: (1..1)

One payment belongs to one order: (1..1)

Final notation: 1:1

10. **Order to Sales Transaction (Generates):**

One order can generate multiple sales transactions: (1..N)

One sales transaction belongs to one order: (1..1)

Final notation: 1:N

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.

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

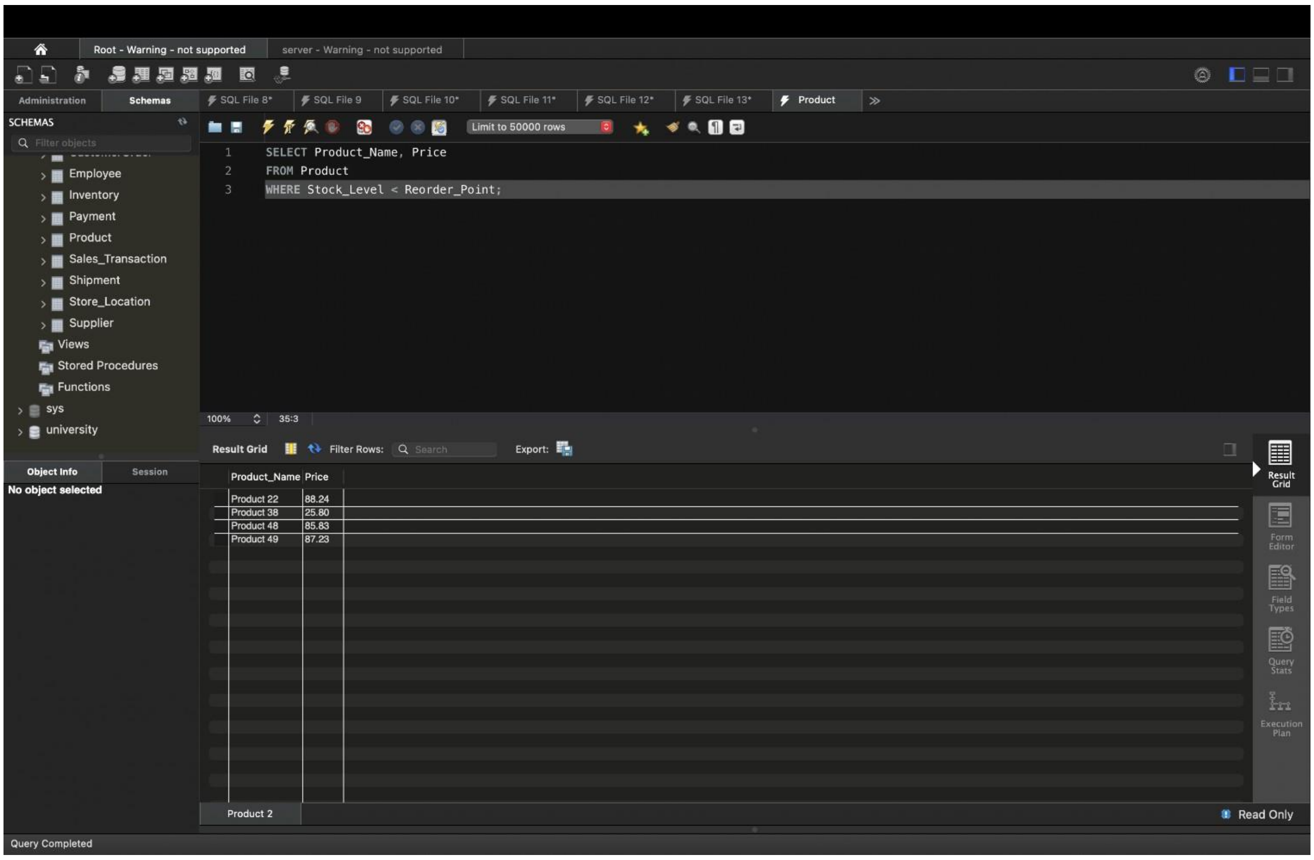
```
CREATE TABLE Payment (
    Payment_ID INT PRIMARY KEY,
    Order_ID INT,
    Payment_Method VARCHAR(50),
    Transaction_Date DATE,
    Payment_Status VARCHAR(50),
    Amount_Paid DECIMAL(10, 2),
    FOREIGN KEY (Order_ID) REFERENCES Order(Order_ID)
);
```

Implementation in MySQL

Queries:

Simple Query:

```
SELECT Product_Name, Price
FROM Product
WHERE Stock_Level < Reorder_Point;
```

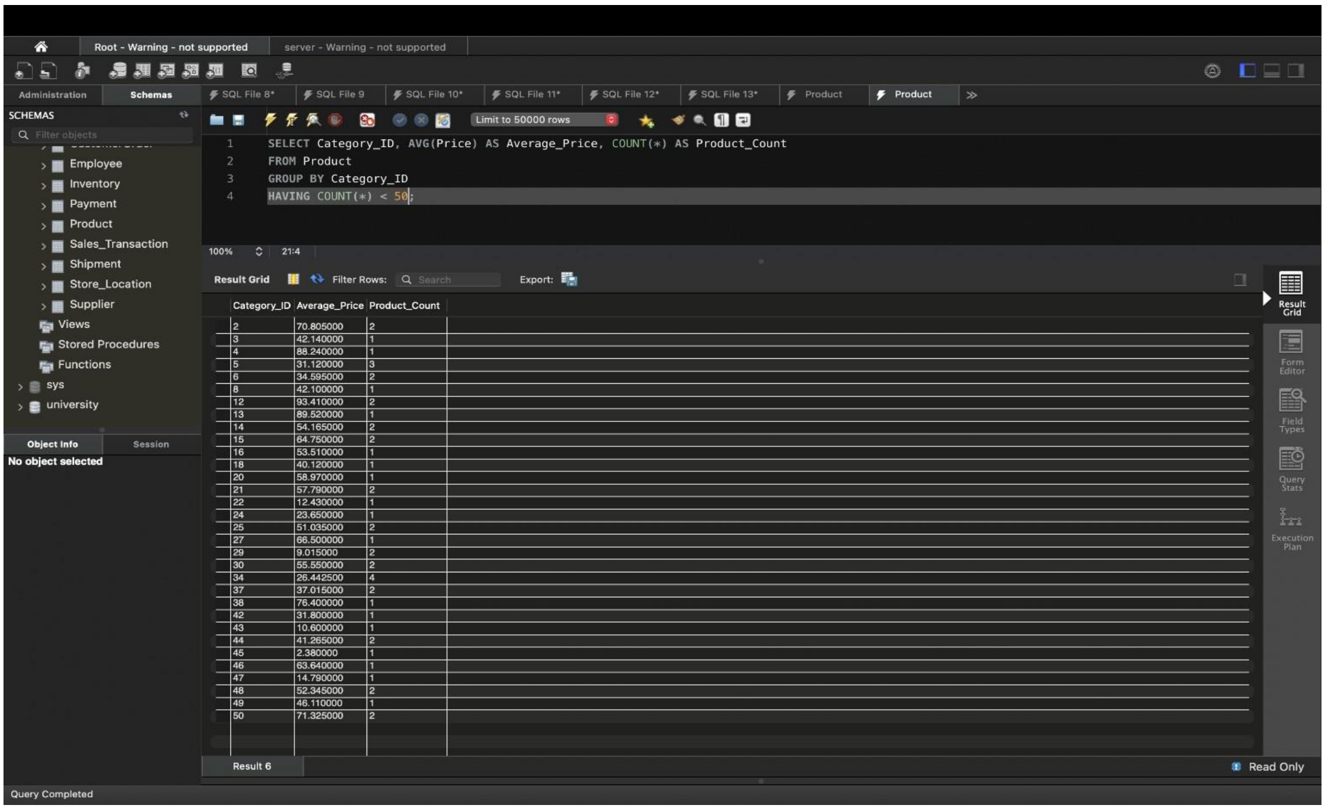


```
SELECT Product_Name, Price
FROM Product
WHERE Stock_Level < Reorder_Point;
```

```
Product_Name  Price
Product 22    88.24
Product 38    25.80
Product 48    85.83
Product 49    87.23
```

Aggregate Query:

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



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

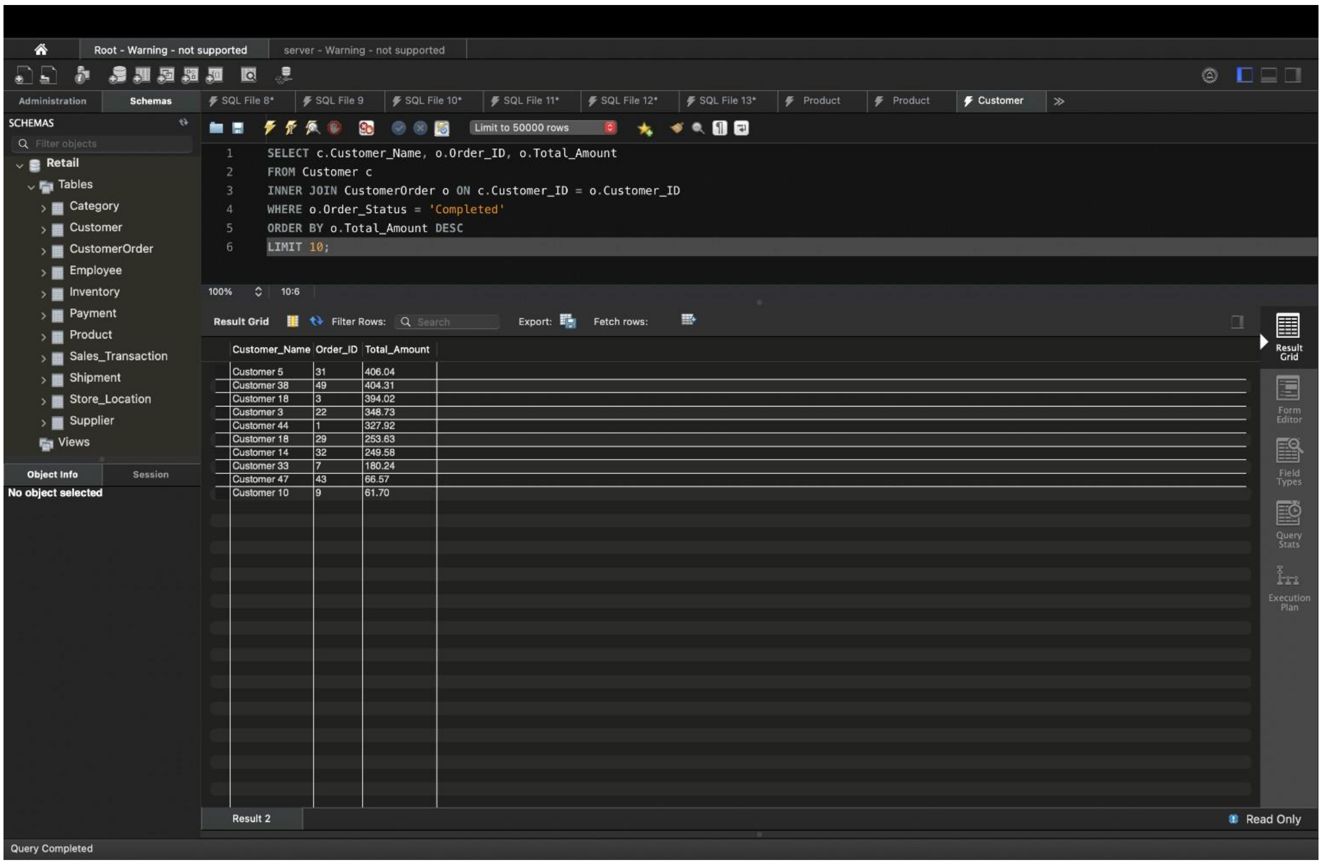
Category_ID	Average_Price	Product_Count
2	70.805000	2
3	42.140000	1
4	88.240000	1
5	31.120000	3
6	34.595000	2
8	42.100000	1
12	93.410000	2
13	89.520000	1
14	54.165000	2
15	64.750000	2
16	53.510000	1
18	40.120000	1
20	58.970000	1
21	57.790000	2
22	12.430000	1
24	23.650000	1
25	51.035000	2
27	66.500000	1
29	9.015000	2
30	55.550000	2
34	26.442500	4
37	37.015000	2
38	76.400000	1
42	31.800000	1
43	10.600000	1
44	41.265000	2
45	2.380000	1
46	63.640000	1
47	14.790000	1
48	52.345000	2
49	46.110000	1
50	71.325000	2

Result 6

Query Completed

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;
```

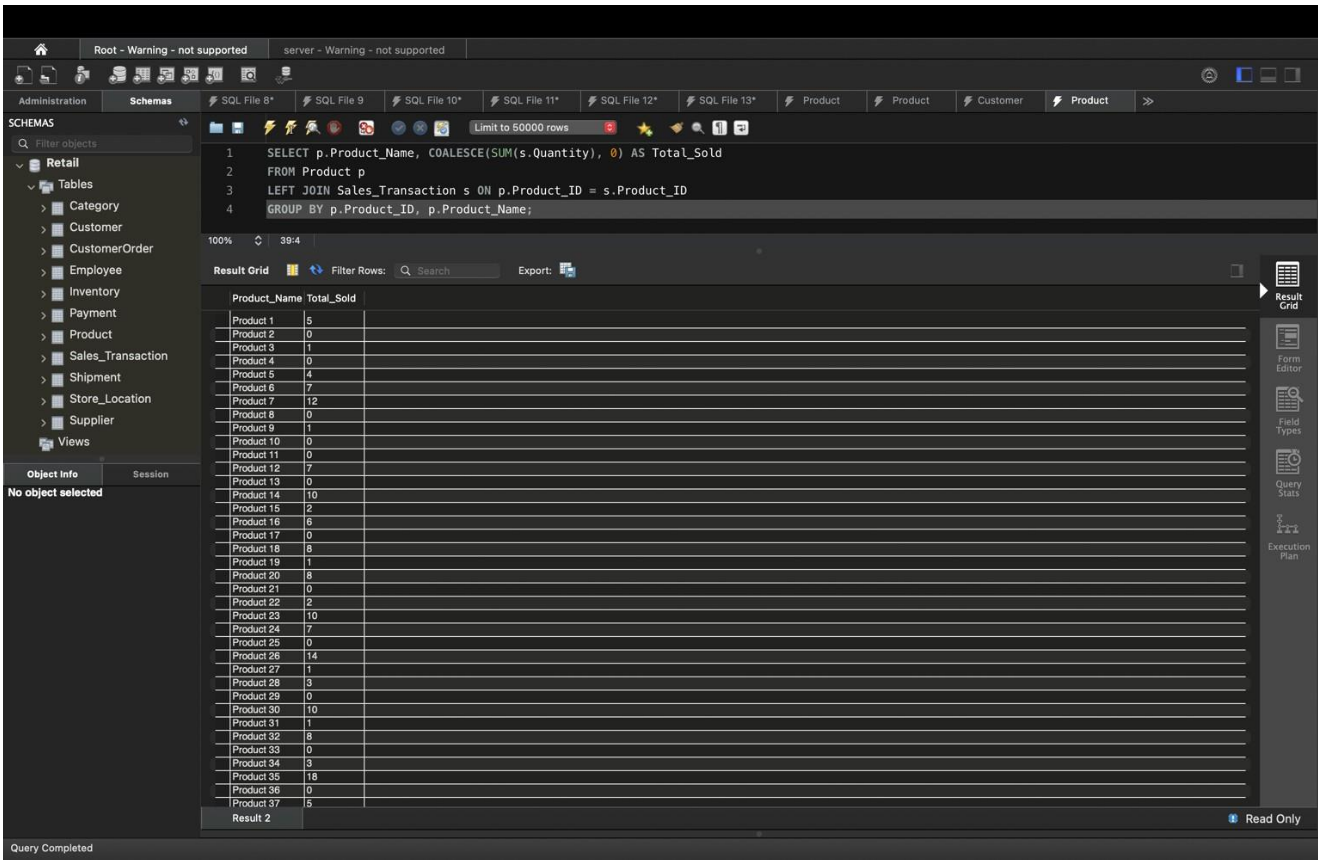


```
SELECT c.Customer_Name, o.Order_ID, o.Total_Amount
FROM Customer c
INNER JOIN CustomerOrder o ON c.Customer_ID = o.Customer_ID
WHERE o.Order_Status = 'Completed'
ORDER BY o.Total_Amount DESC
LIMIT 10;
```

Customer_Name	Order_ID	Total_Amount
Customer 5	31	406.04
Customer 38	49	404.31
Customer 18	3	394.02
Customer 3	22	348.73
Customer 44	1	327.92
Customer 18	29	253.63
Customer 14	32	249.58
Customer 33	7	180.24
Customer 47	43	66.57
Customer 10	9	61.70

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;
```

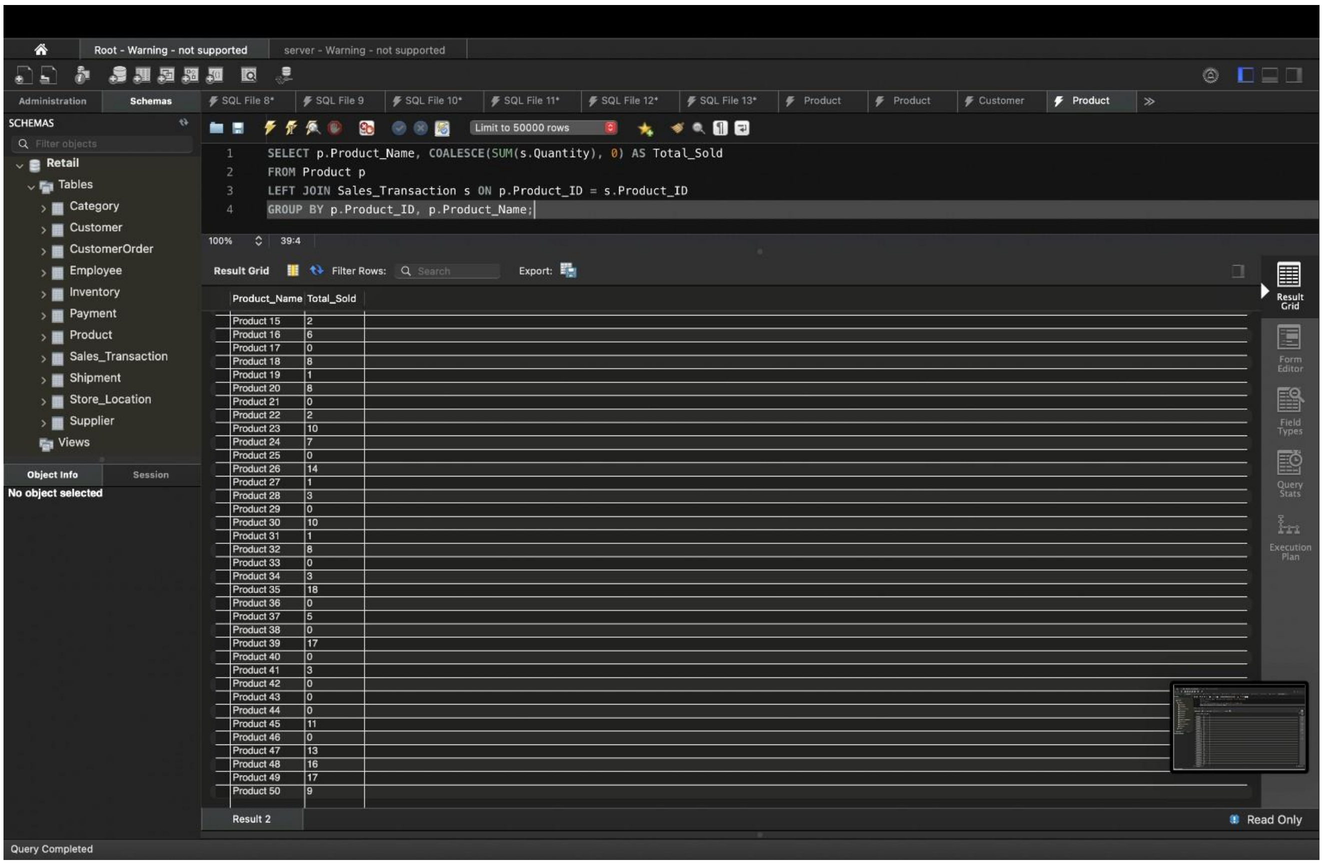


```
SELECT p.Product_Name, COALESCE(SUM(s.Quantity), 0) AS Total_Sold
FROM Product p
LEFT JOIN Sales_Transaction s ON p.Product_ID = s.Product_ID
GROUP BY p.Product_ID, p.Product_Name;
```

Product_Name	Total_Sold
Product 1	5
Product 2	0
Product 3	1
Product 4	0
Product 5	4
Product 6	7
Product 7	12
Product 8	0
Product 9	1
Product 10	0
Product 11	0
Product 12	7
Product 13	0
Product 14	10
Product 15	2
Product 16	6
Product 17	0
Product 18	8
Product 19	1
Product 20	8
Product 21	0
Product 22	2
Product 23	10
Product 24	7
Product 25	0
Product 26	14
Product 27	1
Product 28	3
Product 29	0
Product 30	10
Product 31	1
Product 32	8
Product 33	0
Product 34	3
Product 35	18
Product 36	0
Product 37	5

Result 2

Query Completed

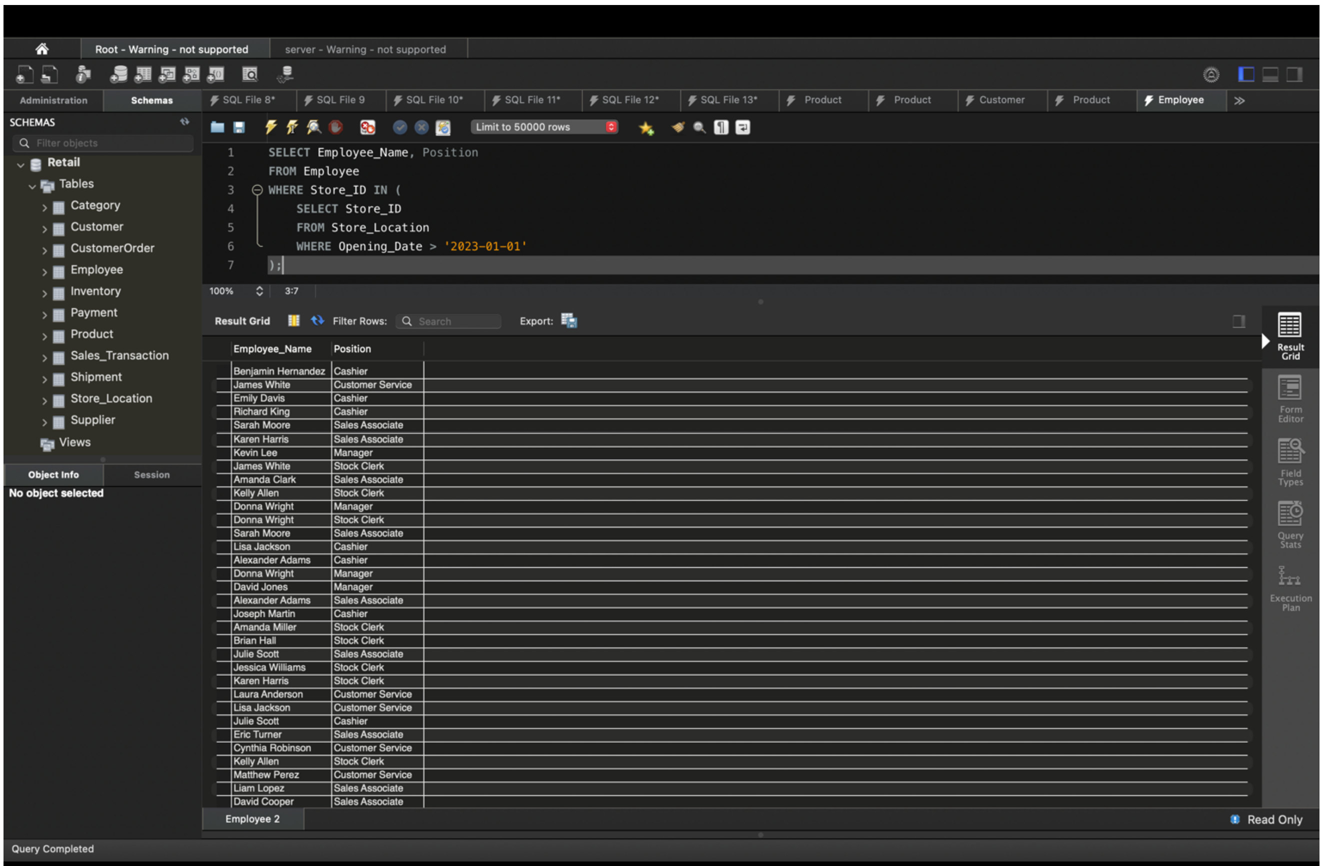


```
SELECT p.Product_Name, COALESCE(SUM(s.Quantity), 0) AS Total_Sold
FROM Product p
LEFT JOIN Sales_Transaction s ON p.Product_ID = s.Product_ID
GROUP BY p.Product_ID, p.Product_Name;
```

Product_Name	Total_Sold
Product 15	2
Product 16	6
Product 17	0
Product 18	8
Product 19	1
Product 20	8
Product 21	0
Product 22	2
Product 23	10
Product 24	7
Product 25	0
Product 26	14
Product 27	1
Product 28	3
Product 29	0
Product 30	10
Product 31	1
Product 32	8
Product 33	0
Product 34	3
Product 35	18
Product 36	0
Product 37	5
Product 38	0
Product 39	17
Product 40	0
Product 41	3
Product 42	0
Product 43	0
Product 44	0
Product 45	11
Product 46	0
Product 47	13
Product 48	16
Product 49	17
Product 50	9

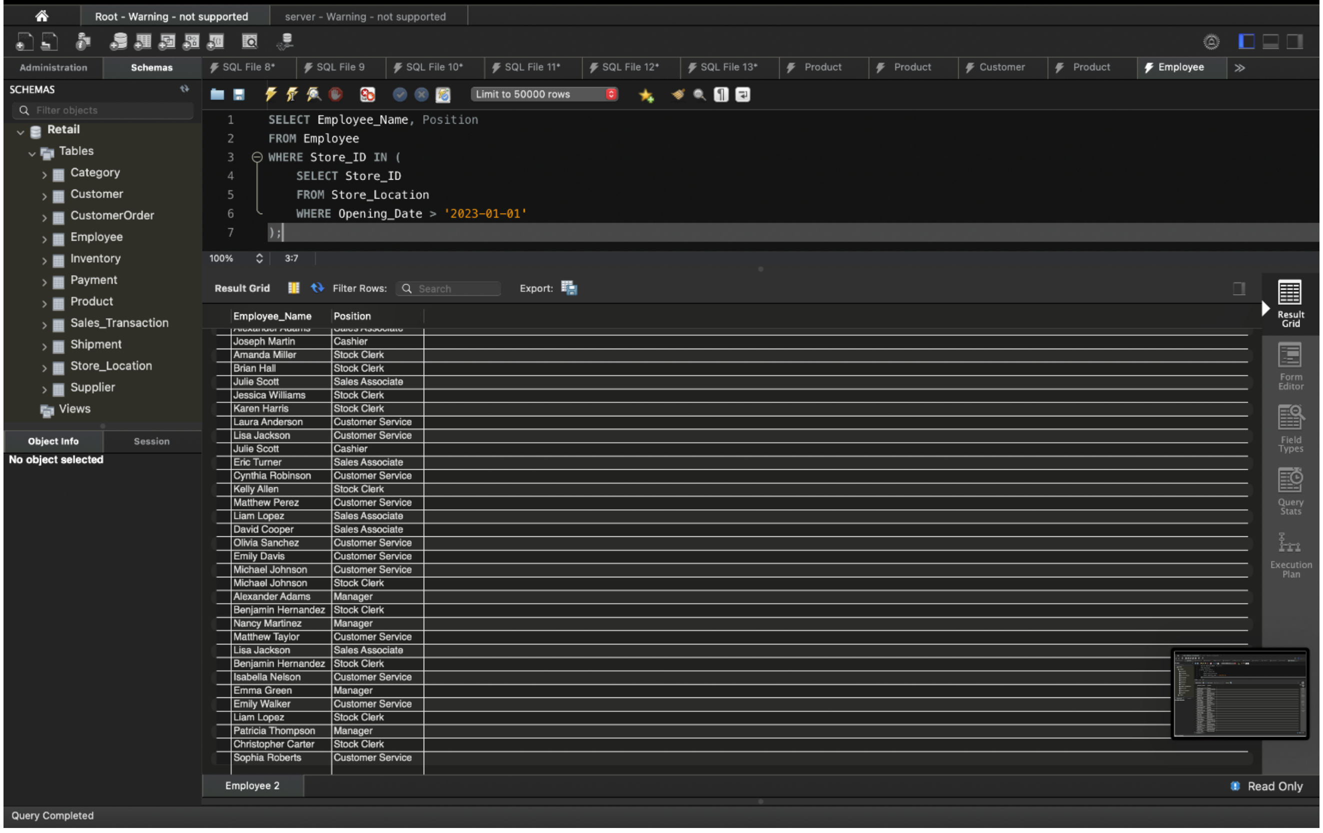
Nested Query:

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



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

Employee_Name	Position
Benjamin Hernandez	Cashier
James White	Customer Service
Emily Davis	Cashier
Richard King	Cashier
Sarah Moore	Sales Associate
Karen Harris	Sales Associate
Kevin Lee	Manager
James White	Stock Clerk
Amanda Clark	Sales Associate
Kelly Allen	Stock Clerk
Donna Wright	Manager
Donna Wright	Stock Clerk
Sarah Moore	Sales Associate
Lisa Jackson	Cashier
Alexander Adams	Cashier
Donna Wright	Manager
David Jones	Manager
Alexander Adams	Sales Associate
Joseph Martin	Cashier
Amanda Miller	Stock Clerk
Brian Hall	Stock Clerk
Julie Scott	Sales Associate
Jessica Williams	Stock Clerk
Karen Harris	Stock Clerk
Laura Anderson	Customer Service
Lisa Jackson	Customer Service
Julie Scott	Cashier
Eric Turner	Sales Associate
Cynthia Robinson	Customer Service
Kelly Allen	Stock Clerk
Matthew Perez	Customer Service
Liam Lopez	Sales Associate
David Cooper	Sales Associate

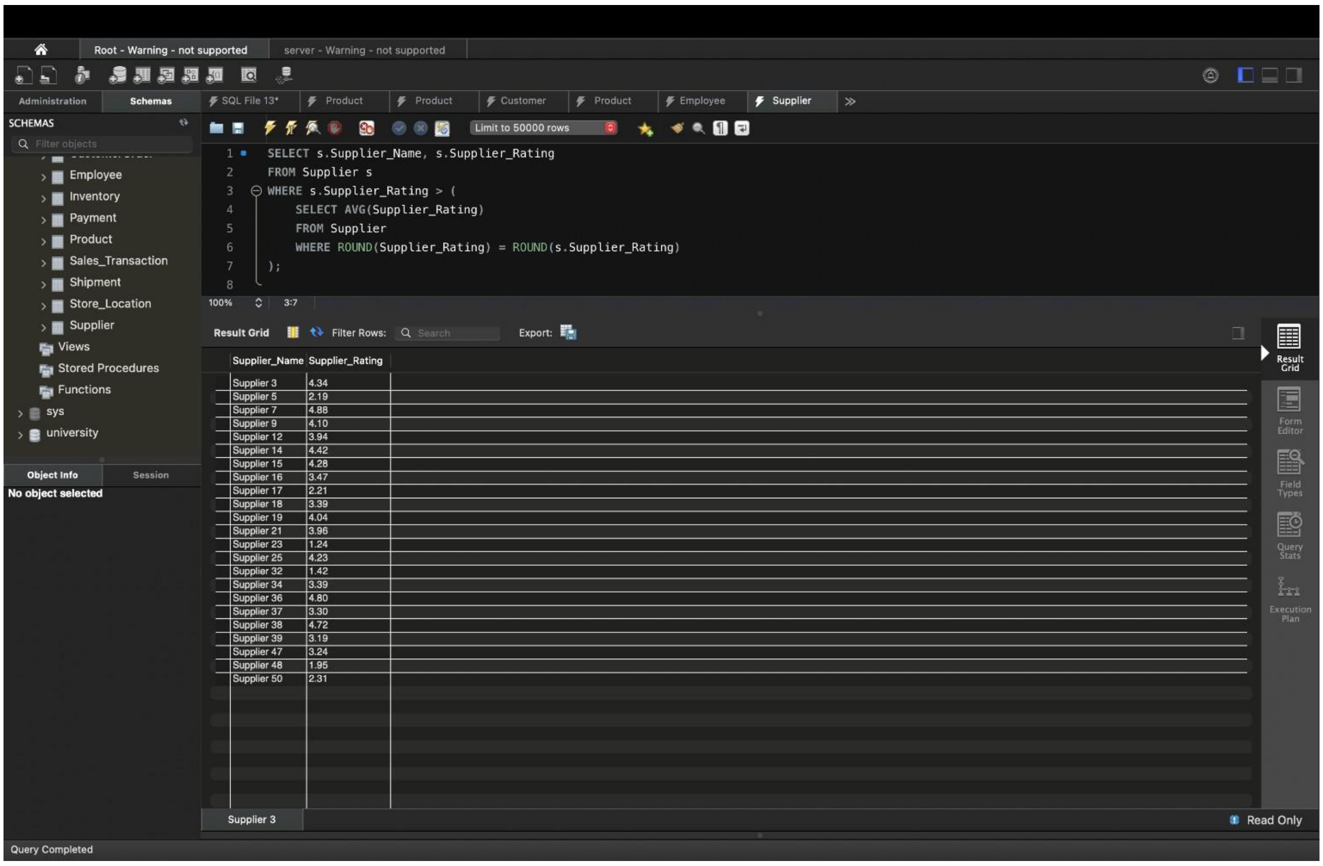


```sql
SELECT Employee_Name, Position
FROM Employee
WHERE Store_ID IN (
    SELECT Store_ID
    FROM Store_Location
    WHERE Opening_Date > '2023-01-01'
);
```

Employee_Name	Position
Alexander Adams	Sales Associate
Joseph Martin	Cashier
Amanda Miller	Stock Clerk
Brian Hall	Stock Clerk
Julie Scott	Sales Associate
Jessica Williams	Stock Clerk
Karen Harris	Stock Clerk
Laura Anderson	Customer Service
Lisa Jackson	Customer Service
Julie Scott	Cashier
Eric Turner	Sales Associate
Cynthia Robinson	Customer Service
Kelly Allen	Stock Clerk
Matthew Perez	Customer Service
Liam Lopez	Sales Associate
David Cooper	Sales Associate
Olivia Sanchez	Customer Service
Emily Davis	Customer Service
Michael Johnson	Customer Service
Michael Johnson	Stock Clerk
Alexander Adams	Manager
Benjamin Hernandez	Stock Clerk
Nancy Martinez	Manager
Matthew Taylor	Customer Service
Lisa Jackson	Sales Associate
Benjamin Hernandez	Stock Clerk
Isabella Nelson	Customer Service
Emma Green	Manager
Emily Walker	Customer Service
Liam Lopez	Stock Clerk
Patricia Thompson	Manager
Christopher Carter	Stock Clerk
Sophia Roberts	Customer Service

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)
);
```



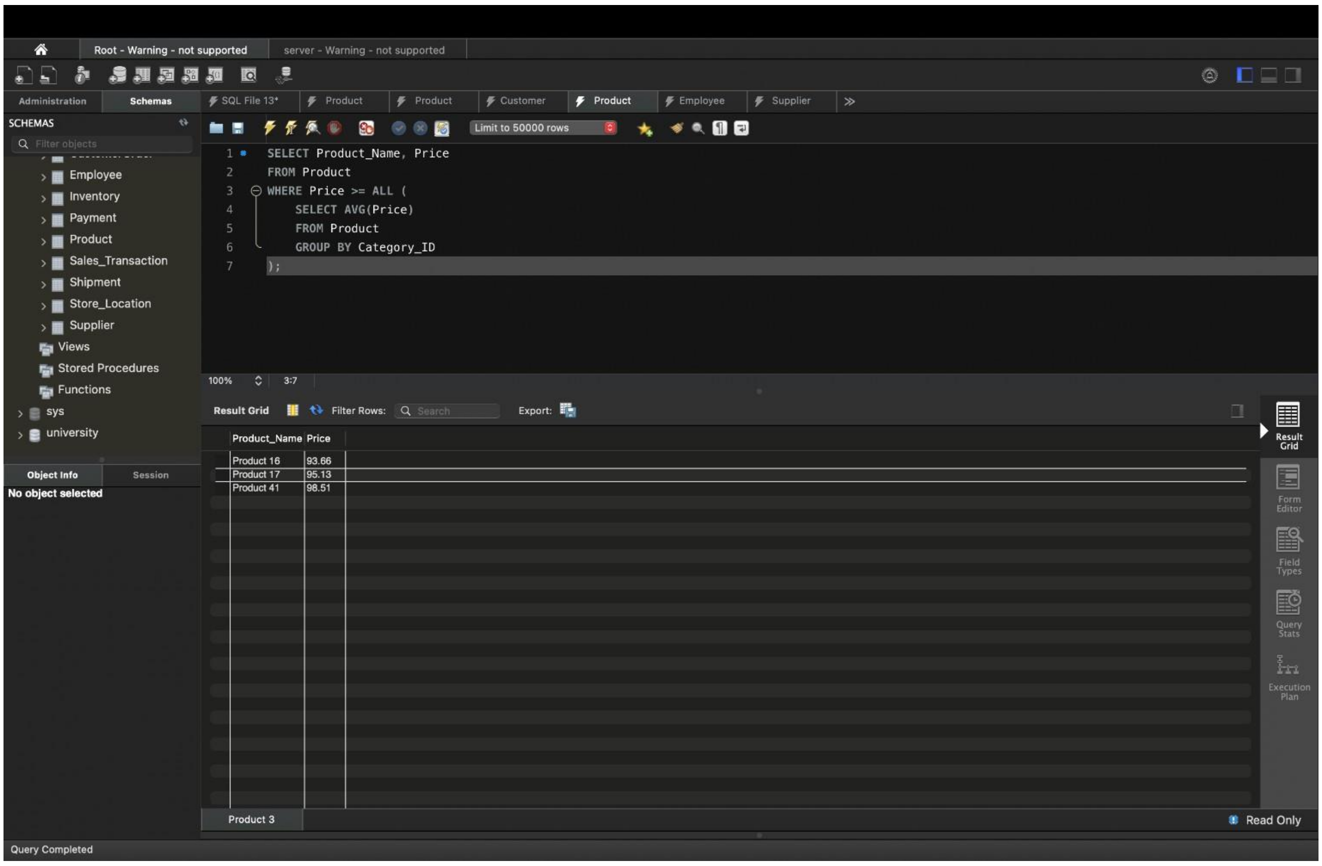
```
SELECT s.Supplier_Name, s.Supplier_Rating
FROM Supplier s
WHERE s.Supplier_Rating > (
    SELECT AVG(Supplier_Rating)
    FROM Supplier
    WHERE ROUND(Supplier_Rating) = ROUND(s.Supplier_Rating)
);
```

Supplier_Name	Supplier_Rating
Supplier 3	4.34
Supplier 5	2.19
Supplier 7	4.88
Supplier 9	4.10
Supplier 12	3.94
Supplier 14	4.42
Supplier 15	4.28
Supplier 16	3.47
Supplier 17	2.21
Supplier 18	3.39
Supplier 19	4.04
Supplier 21	3.96
Supplier 23	1.24
Supplier 25	4.23
Supplier 32	1.42
Supplier 34	3.39
Supplier 36	4.80
Supplier 37	3.30
Supplier 38	4.72
Supplier 39	3.19
Supplier 47	3.24
Supplier 48	1.95
Supplier 50	2.31

Supplier 3

=ALL Query:

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



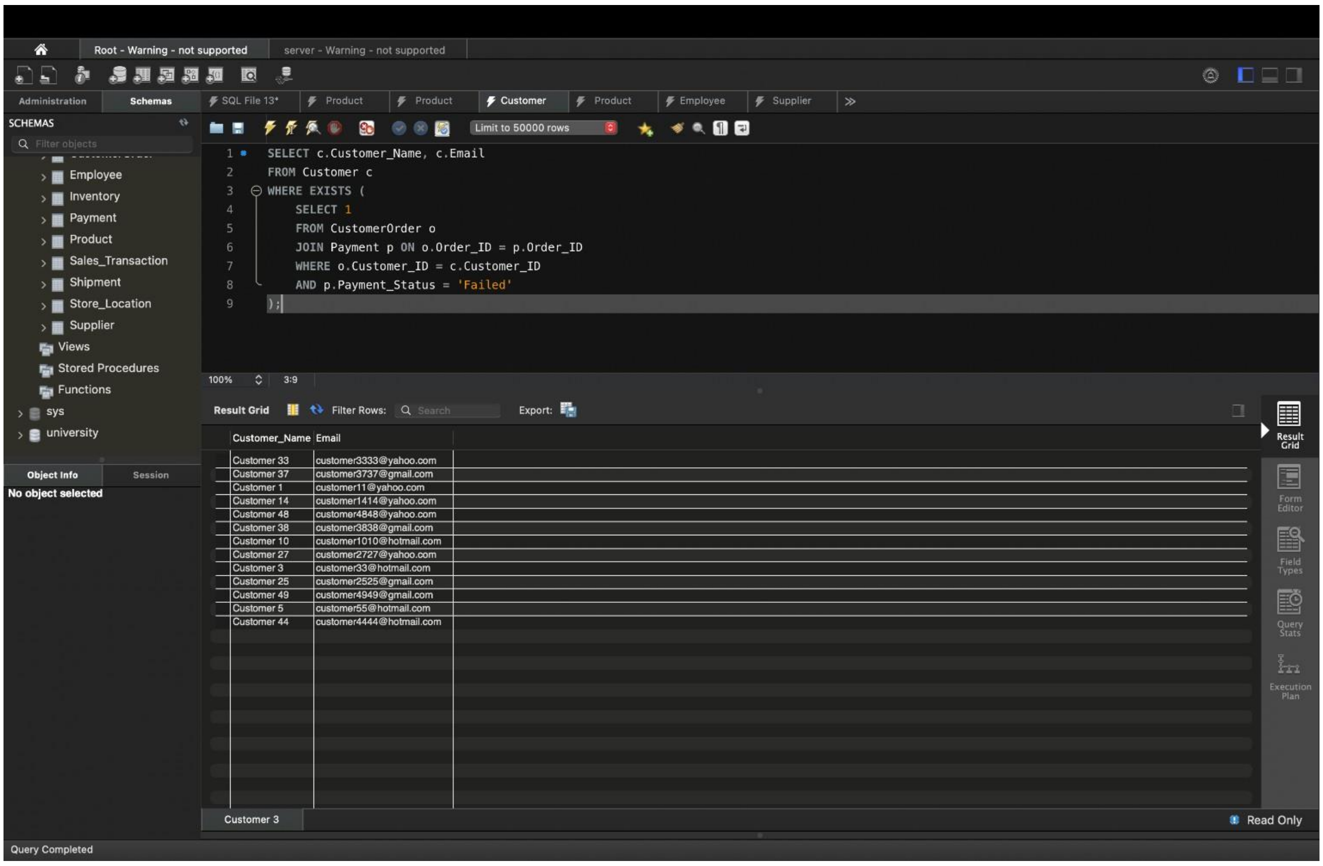
```
SELECT Product_Name, Price
FROM Product
WHERE Price >= ALL (
    SELECT AVG(Price)
    FROM Product
    GROUP BY Category_ID
);
```

Product_Name	Price
Product 16	93.66
Product 17	95.13
Product 41	98.51

Query Completed

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'
);
```



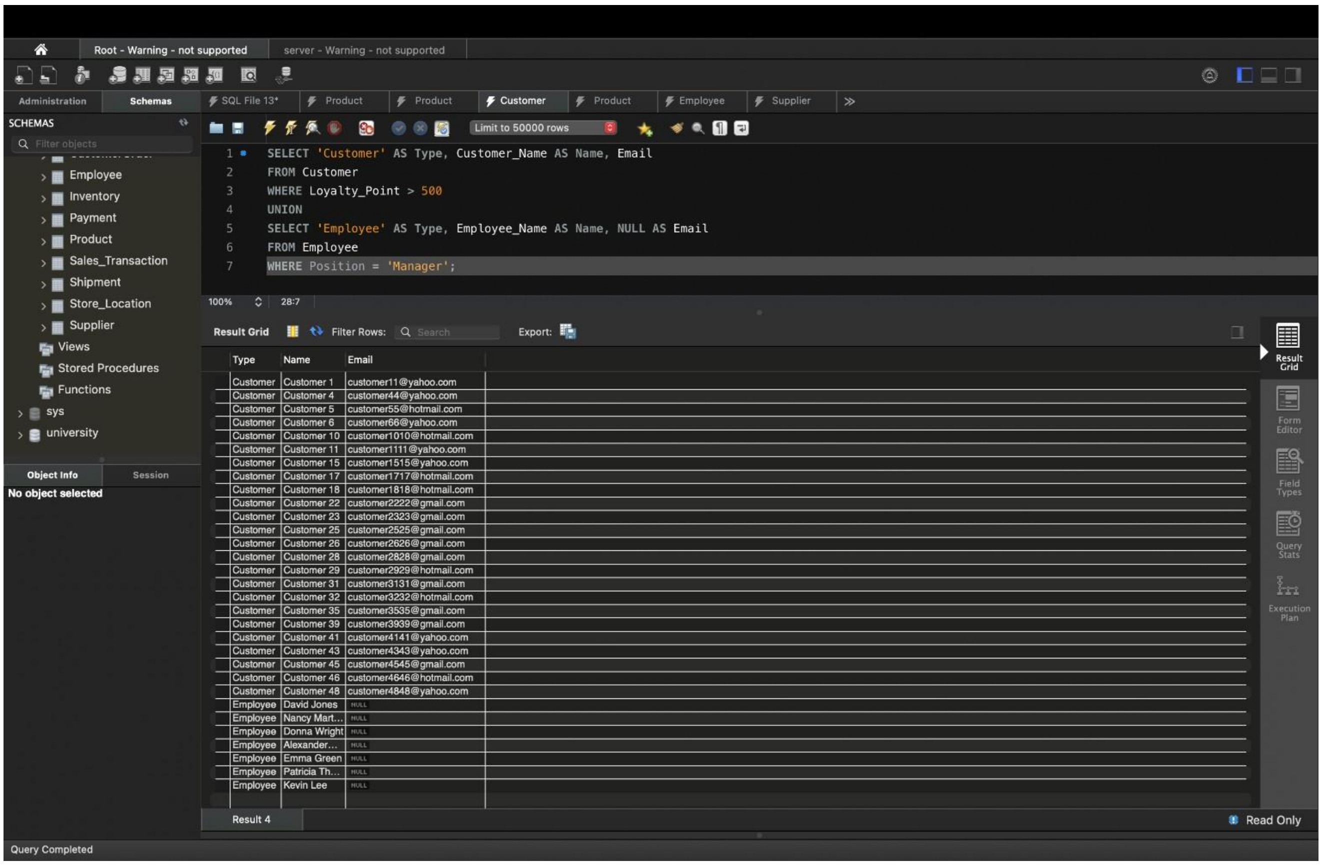
```
SELECT c.Customer_Name, c.Email
FROM Customer c
WHERE EXISTS (
    SELECT 1
    FROM CustomerOrder o
    JOIN Payment p ON o.Order_ID = p.Order_ID
    WHERE o.Customer_ID = c.Customer_ID
    AND p.Payment_Status = 'Failed'
);
```

Customer_Name	Email
Customer 33	customer3333@yahoo.com
Customer 37	customer3737@gmail.com
Customer 1	customer11@yahoo.com
Customer 14	customer1414@yahoo.com
Customer 48	customer4848@yahoo.com
Customer 38	customer3838@gmail.com
Customer 10	customer1010@hotmail.com
Customer 27	customer2727@yahoo.com
Customer 3	customer33@hotmail.com
Customer 25	customer2525@gmail.com
Customer 49	customer4949@gmail.com
Customer 5	customer55@hotmail.com
Customer 44	customer4444@hotmail.com

Query Completed

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';
```



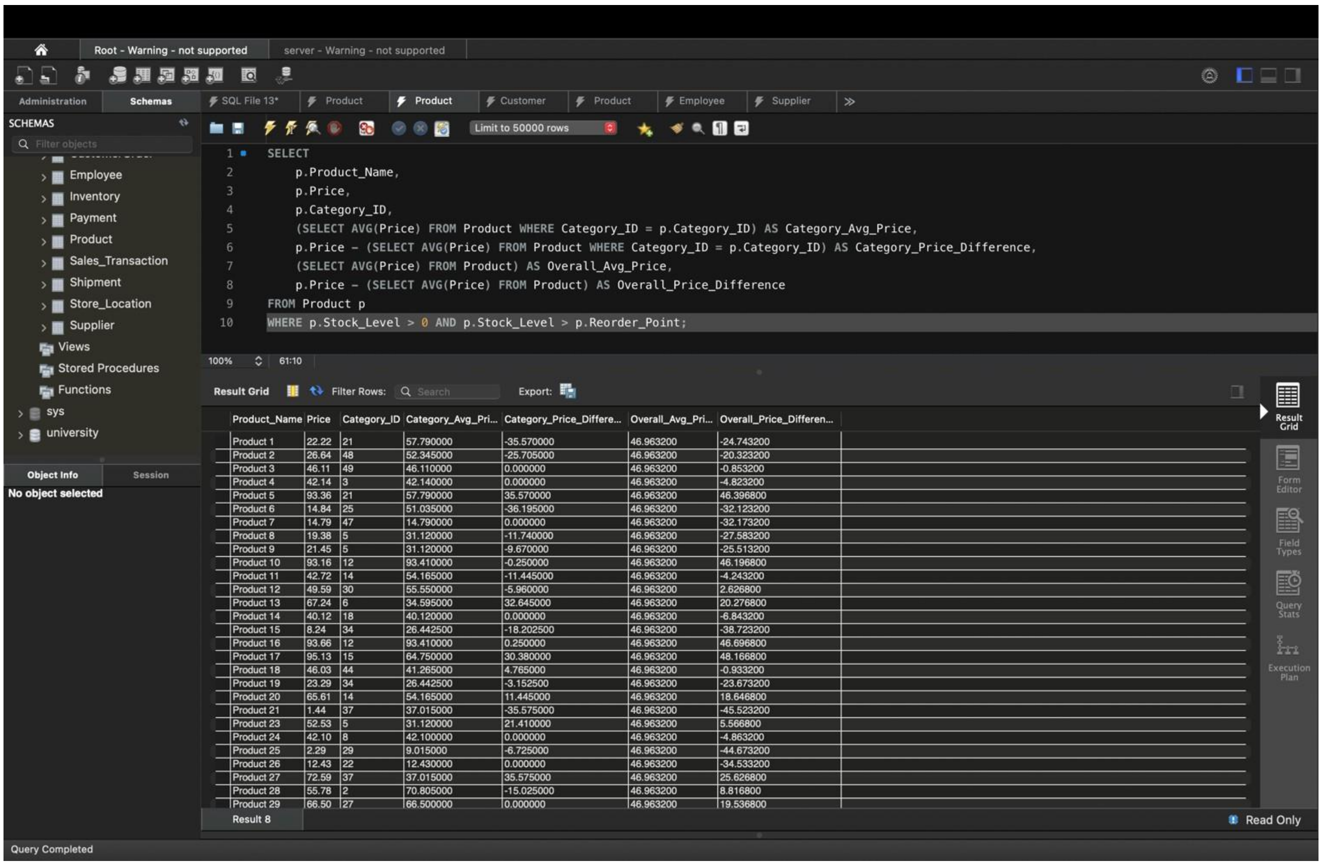
```
SELECT 'Customer' AS Type, Customer_Name AS Name, Email
FROM Customer
WHERE Loyalty_Point > 500
UNION
SELECT 'Employee' AS Type, Employee_Name AS Name, NULL AS Email
FROM Employee
WHERE Position = 'Manager';
```

Type	Name	Email
Customer	Customer 1	customer11@yahoo.com
Customer	Customer 4	customer44@yahoo.com
Customer	Customer 5	customer55@hotmail.com
Customer	Customer 6	customer66@yahoo.com
Customer	Customer 10	customer1010@hotmail.com
Customer	Customer 11	customer1111@yahoo.com
Customer	Customer 15	customer1515@yahoo.com
Customer	Customer 17	customer1717@hotmail.com
Customer	Customer 18	customer1818@hotmail.com
Customer	Customer 22	customer2222@gmail.com
Customer	Customer 23	customer2323@gmail.com
Customer	Customer 25	customer2525@gmail.com
Customer	Customer 26	customer2626@gmail.com
Customer	Customer 28	customer2828@gmail.com
Customer	Customer 29	customer2929@hotmail.com
Customer	Customer 31	customer3131@gmail.com
Customer	Customer 32	customer3232@hotmail.com
Customer	Customer 35	customer3535@gmail.com
Customer	Customer 39	customer3939@gmail.com
Customer	Customer 41	customer4141@yahoo.com
Customer	Customer 43	customer4343@yahoo.com
Customer	Customer 45	customer4545@gmail.com
Customer	Customer 46	customer4646@hotmail.com
Customer	Customer 48	customer4848@yahoo.com
Employee	David Jones	NULL
Employee	Nancy Mart...	NULL
Employee	Donna Wright	NULL
Employee	Alexander...	NULL
Employee	Emma Green	NULL
Employee	Patricia Th...	NULL
Employee	Kevin Lee	NULL

Result 4 · Read Only · Query Completed

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;
```

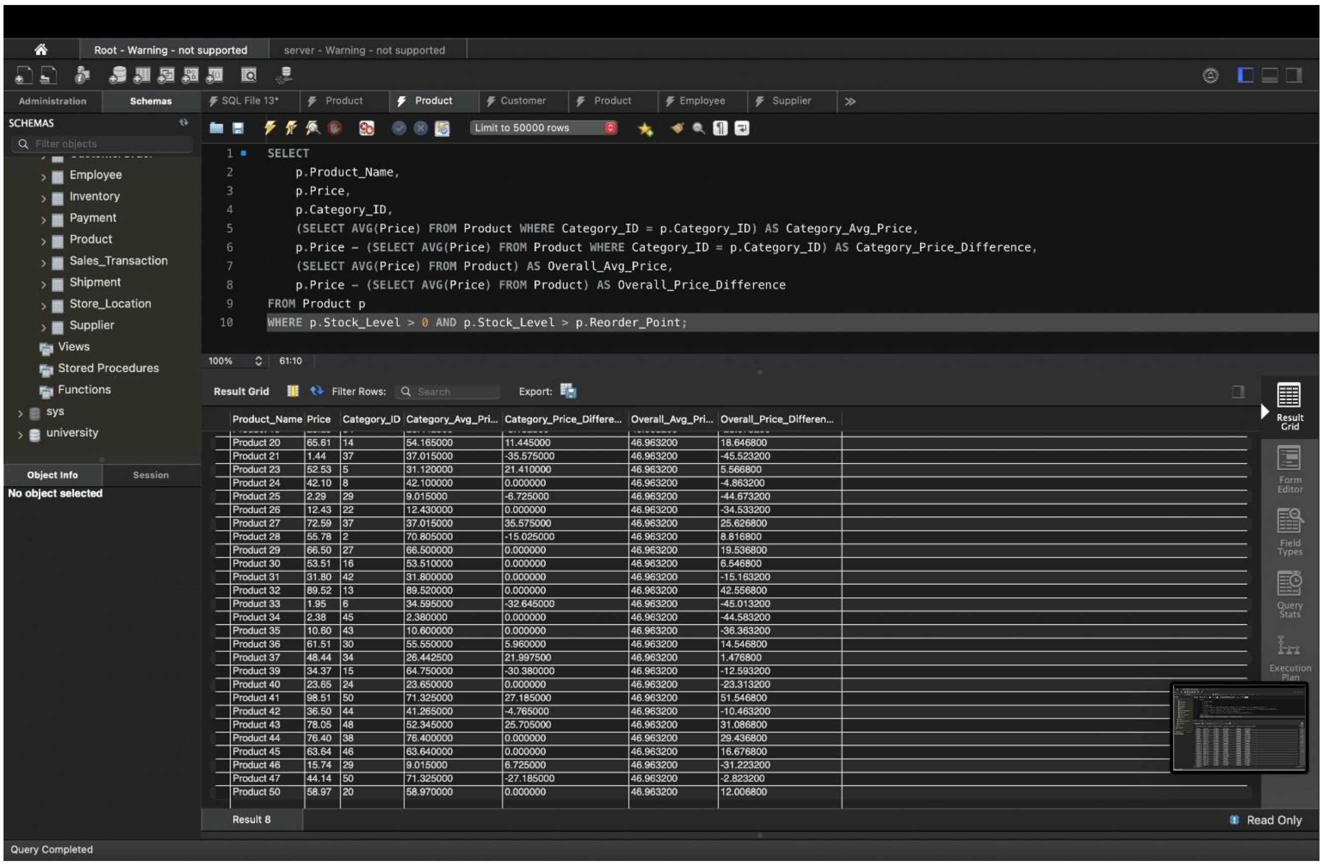


```
SELECT
    p.Product_Name,
    p.Price,
    p.Category_ID,
    (SELECT AVG(Price) FROM Product WHERE Category_ID = p.Category_ID) AS Category_Avg_Price,
    p.Price - (SELECT AVG(Price) FROM Product WHERE Category_ID = p.Category_ID) AS Category_Price_Difference,
    (SELECT AVG(Price) FROM Product) 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

Result 8

Query Completed



```sql
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 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
Product 30	53.51	16	53.510000	0.000000	46.963200	6.546800
Product 31	31.80	42	31.800000	0.000000	46.963200	-15.163200
Product 32	89.52	13	89.520000	0.000000	46.963200	42.556800
Product 33	1.95	6	34.595000	-32.645000	46.963200	-45.013200
Product 34	2.38	45	2.380000	0.000000	46.963200	-44.583200
Product 35	10.60	43	10.600000	0.000000	46.963200	-36.363200
Product 36	61.51	30	55.550000	5.960000	46.963200	14.546800
Product 37	48.44	34	26.442500	21.997500	46.963200	1.476800
Product 39	34.37	15	64.750000	-30.380000	46.963200	-12.593200
Product 40	23.65	24	23.650000	0.000000	46.963200	-23.313200
Product 41	98.51	50	71.325000	27.185000	46.963200	51.546800
Product 42	36.50	44	41.265000	-4.765000	46.963200	-10.463200
Product 43	78.05	48	52.345000	25.705000	46.963200	31.086800
Product 44	76.40	38	76.400000	0.000000	46.963200	29.436800
Product 45	63.64	46	63.640000	0.000000	46.963200	16.676800
Product 46	15.74	29	9.015000	6.725000	46.963200	-31.223200
Product 47	44.14	50	71.325000	-27.185000	46.963200	-2.823200
Product 50	58.97	20	58.970000	0.000000	46.963200	12.006800

Result 8

Query Completed

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.

NoSQL Implementation

FOR CREATING AND INSERTING TABLE CODE USED:

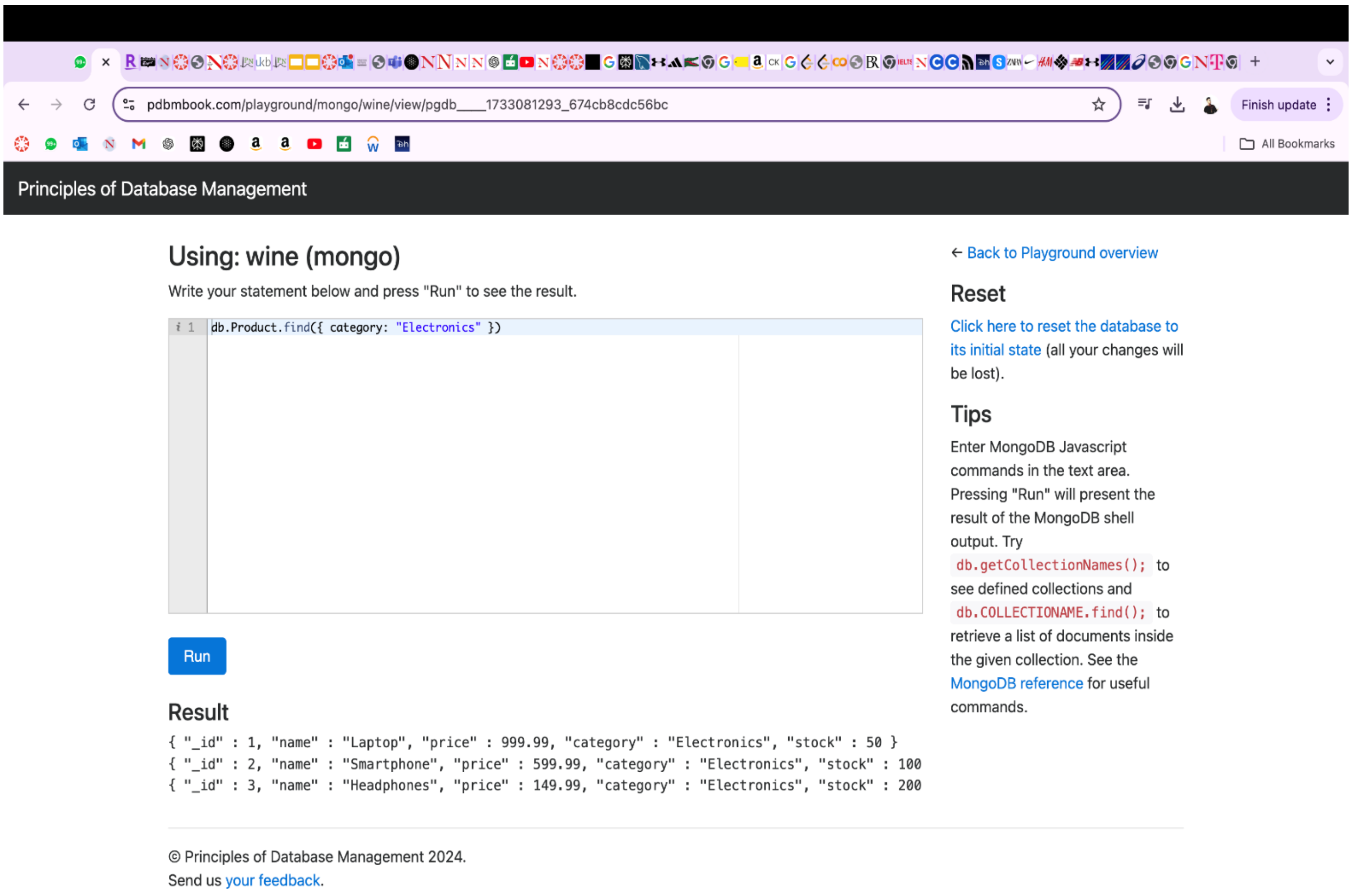
```
// Create Product collection
db.createCollection("Product")
db.Product.insertMany([
  { _id: 1, name: "Laptop", price: 999.99, category: "Electronics", stock: 50 },
  { _id: 2, name: "Smartphone", price: 599.99, category: "Electronics", stock: 100 },
  { _id: 3, name: "Headphones", price: 149.99, category: "Electronics", stock: 200 },
  { _id: 4, name: "T-shirt", price: 19.99, category: "Clothing", stock: 500 },
  { _id: 5, name: "Jeans", price: 59.99, category: "Clothing", stock: 300 }
])

// Create Customer collection
db.createCollection("Customer")
db.Customer.insertMany([
  { _id: 1, name: "John Doe", email: "john@example.com", loyaltyPoints: 100 },
  { _id: 2, name: "Jane Smith", email: "jane@example.com", loyaltyPoints: 150 },
  { _id: 3, name: "Bob Johnson", email: "bob@example.com", loyaltyPoints: 75 }
])

// Create Order collection
db.createCollection("Order")
db.Order.insertMany([
  { _id: 1, customerId: 1, products: [{ productId: 1, quantity: 1 }, { productId: 3, quantity: 2 }], totalAmount:
1299.97, status: "Completed" },
  { _id: 2, customerId: 2, products: [{ productId: 2, quantity: 1 }, { productId: 4, quantity: 3 }], totalAmount:
659.96, status: "Processing" },
  { _id: 3, customerId: 3, products: [{ productId: 5, quantity: 2 }], totalAmount: 119.98, status: "Shipped" }
])
```

Query 1: SIMPLE QUERY:

```
db.Product.find({ category: "Electronics" })
```


pdbmbook.com/playground/mongo/wine/view/pgdb____1733081293_674cb8cdc56bc
Finish update
All Bookmarks
Principles of Database Management
Run
← Back to Playground overview

Using: wine (mongo)

Write your statement below and press "Run" to see the result.

```
db.Product.find({ category: "Electronics" })
```

Result

```
{ "_id" : 1, "name" : "Laptop", "price" : 999.99, "category" : "Electronics", "stock" : 50 }
{ "_id" : 2, "name" : "Smartphone", "price" : 599.99, "category" : "Electronics", "stock" : 100
{ "_id" : 3, "name" : "Headphones", "price" : 149.99, "category" : "Electronics", "stock" : 200
```

Reset

Click here to reset the database to its initial state (all your changes will be lost).

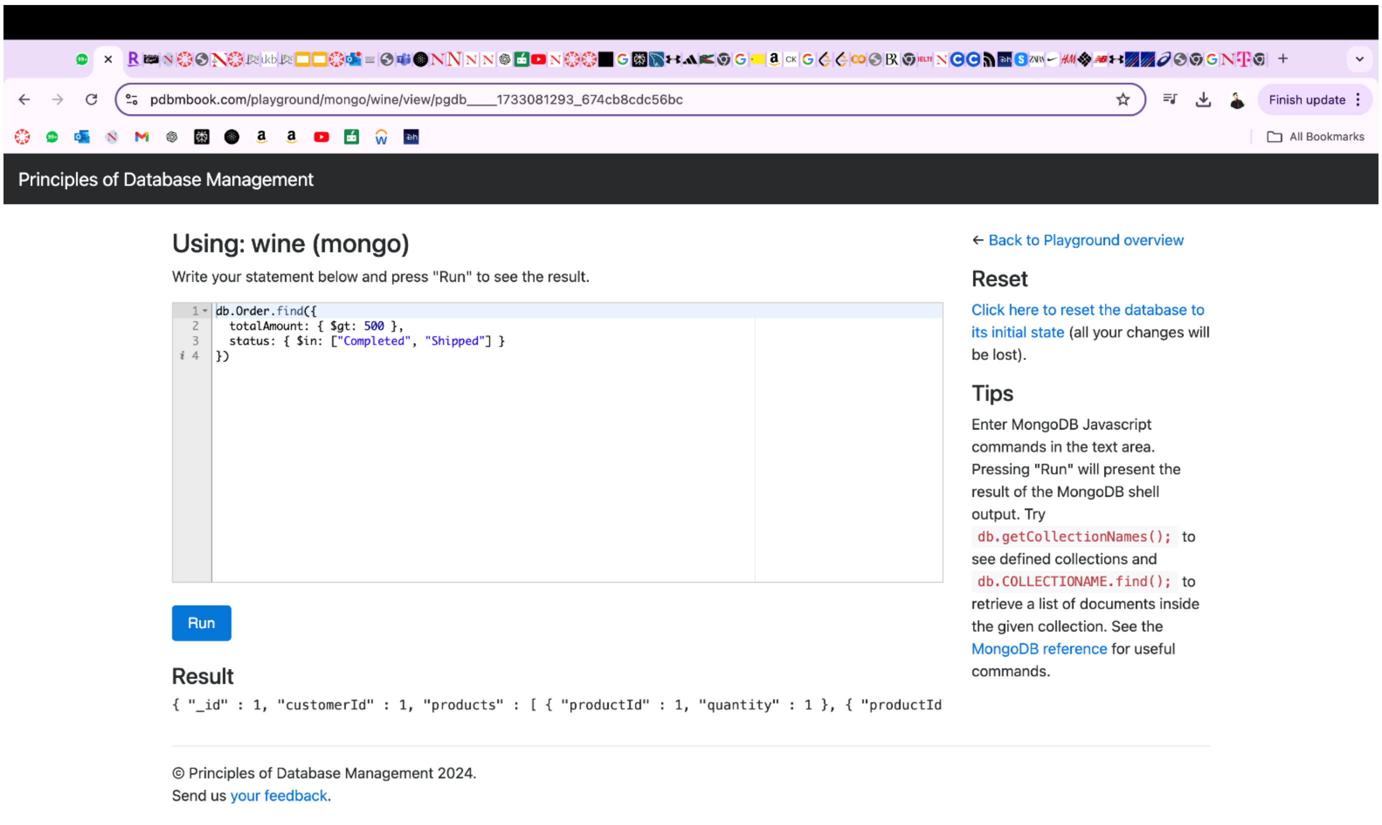
Tips

Enter MongoDB Javascript commands in the text area. Pressing "Run" will present the result of the MongoDB shell output. Try `db.getCollectionNames();` to see defined collections and `db.COLLECTIONAME.find();` to retrieve a list of documents inside the given collection. See the MongoDB reference for useful commands.

© Principles of Database Management 2024.
Send us your feedback.

Query 2 : MORE COMPLEX QUERY:

```
db.Order.find({
  totalAmount: { $gt: 500 },
  status: { $in: ["Completed", "Shipped"] }
})
```


pdbmbook.com/playground/mongo/wine/view/pgdb____1733081293_674cb8cdc56bc
Finish update
All Bookmarks
Run

Principles of Database Management

Using: wine (mongo)

Write your statement below and press "Run" to see the result.

```
db.Order.find({
  totalAmount: { $gt: 500 },
  status: { $in: ["Completed", "Shipped"] }
})
```

Result

```
{ "_id" : 1, "customerId" : 1, "products" : [ { "productId" : 1, "quantity" : 1 }, { "productId
```

← Back to Playground overview

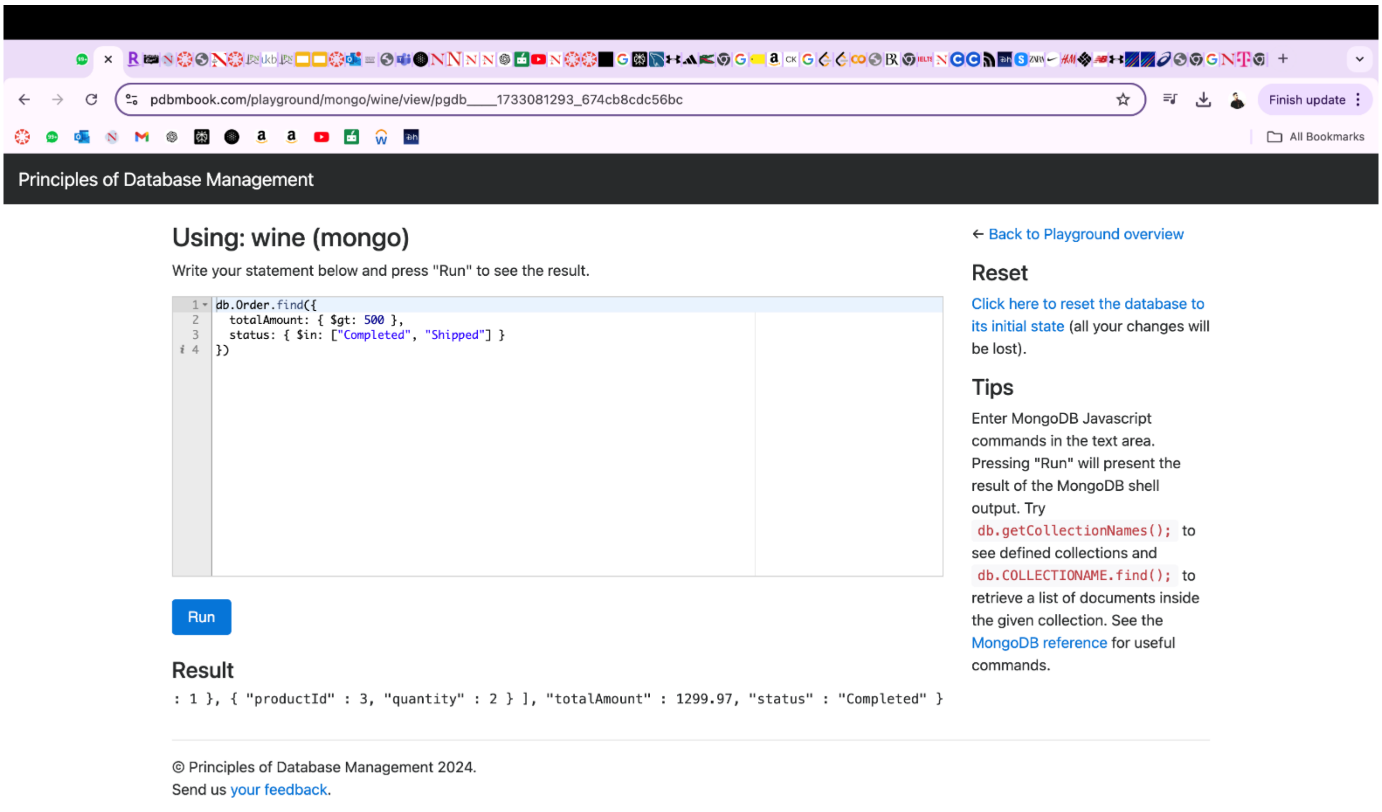
Reset

Click here to reset the database to its initial state (all your changes will be lost).

Tips

Enter MongoDB Javascript commands in the text area. Pressing "Run" will present the result of the MongoDB shell output. Try `db.getCollectionNames();` to see defined collections and `db.COLLECTIONAME.find();` to retrieve a list of documents inside the given collection. See the MongoDB reference for useful commands.

© Principles of Database Management 2024.
Send us your feedback.


pdbmbook.com/playground/mongo/wine/view/pgdb____1733081293_674cb8cdc56bc
Finish update
All Bookmarks
Run

Principles of Database Management

Using: wine (mongo)

Write your statement below and press "Run" to see the result.

```
db.Order.find({
  totalAmount: { $gt: 500 },
  status: { $in: ["Completed", "Shipped"] }
})
```

Result

```
: 1 }, { "productId" : 3, "quantity" : 2 } ], "totalAmount" : 1299.97, "status" : "Completed" }
```

← Back to Playground overview

Reset

Click here to reset the database to its initial state (all your changes will be lost).

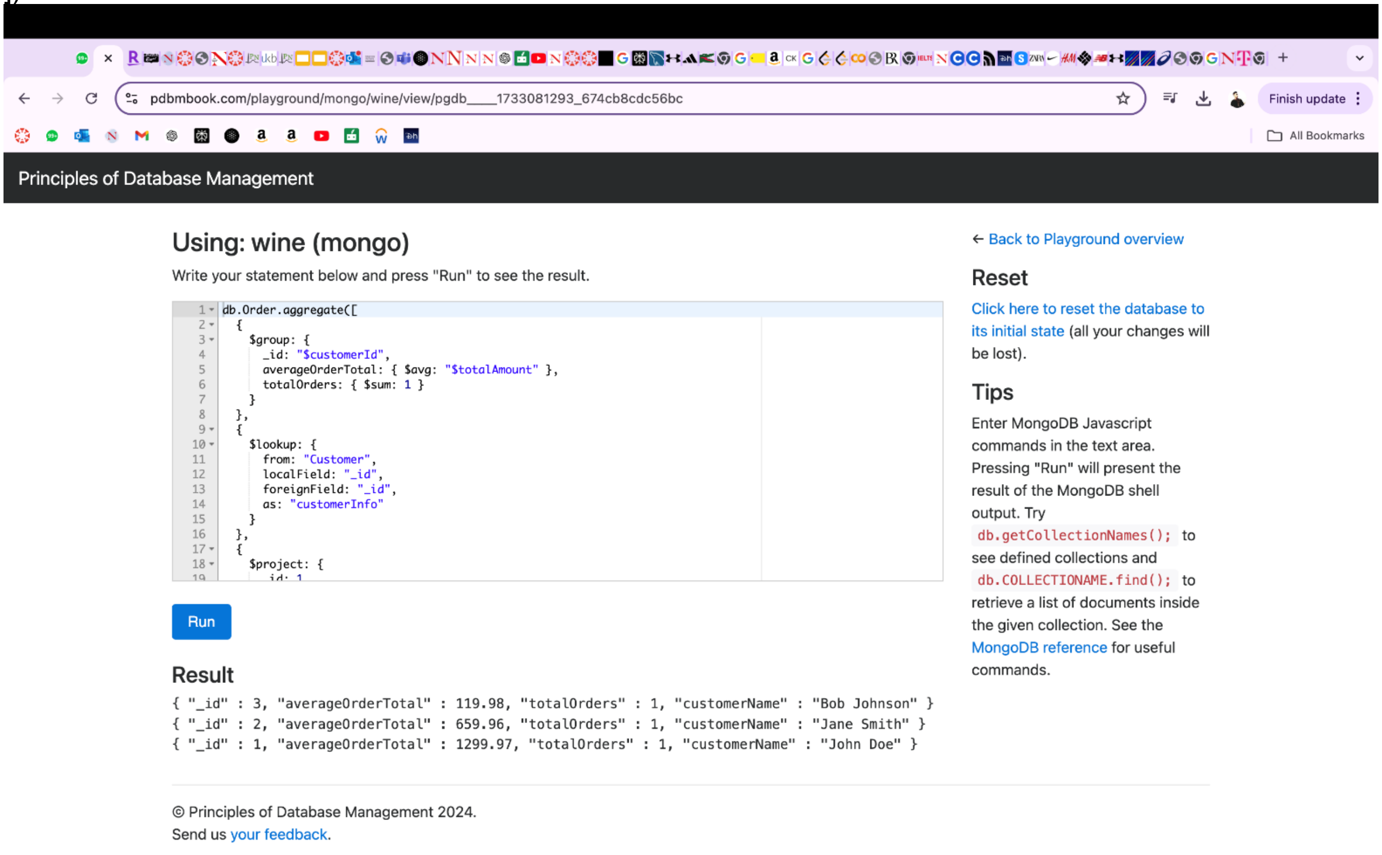
Tips

Enter MongoDB Javascript commands in the text area. Pressing "Run" will present the result of the MongoDB shell output. Try `db.getCollectionNames();` to see defined collections and `db.COLLECTIONAME.find();` to retrieve a list of documents inside the given collection. See the MongoDB reference for useful commands.

© Principles of Database Management 2024.
Send us your feedback.

Query 3: AGGREGATE QUERY:

```
db.Order.aggregate([
 {
   $group: {
     _id: "$customerId",
     averageOrderTotal: { $avg: "$totalAmount" },
     totalOrders: { $sum: 1 }
   }
 },
 {
   $lookup: {
     from: "Customer",
     localField: "_id",
     foreignField: "_id",
     as: "customerInfo"
   }
 },
 {
   $project: {
     _id: 1,
     customerName: { $arrayElemAt: ["$customerInfo.name", 0] },
     averageOrderTotal: 1,
     totalOrders: 1
   }
 }
])
```



pdbmbook.com/playground/mongo/wine/view/pgdb____1733081293_674cb8cdc56bc

Principles of Database Management

Using: wine (mongo)

Write your statement below and press "Run" to see the result.

```
db.Order.aggregate([
  {
    $group: {
      _id: "$customerId",
      averageOrderTotal: { $avg: "$totalAmount" },
      totalOrders: { $sum: 1 }
    }
  },
  {
    $lookup: {
      from: "Customer",
      localField: "_id",
      foreignField: "_id",
      as: "customerInfo"
    }
  },
  {
    $project: {
```

Run

Result

```
{ "_id" : 3, "averageOrderTotal" : 119.98, "totalOrders" : 1, "customerName" : "Bob Johnson" }
{ "_id" : 2, "averageOrderTotal" : 659.96, "totalOrders" : 1, "customerName" : "Jane Smith" }
{ "_id" : 1, "averageOrderTotal" : 1299.97, "totalOrders" : 1, "customerName" : "John Doe" }
```

← Back to Playground overview

Reset

Click here to reset the database to its initial state (all your changes will be lost).

Tips

Enter MongoDB Javascript commands in the text area. Pressing "Run" will present the result of the MongoDB shell output. Try `db.getCollectionNames();` to see defined collections and `db.COLLECTIONAME.find();` to retrieve a list of documents inside the given collection. See the MongoDB reference for useful commands.

© Principles of Database Management 2024.
Send us your feedback.