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:

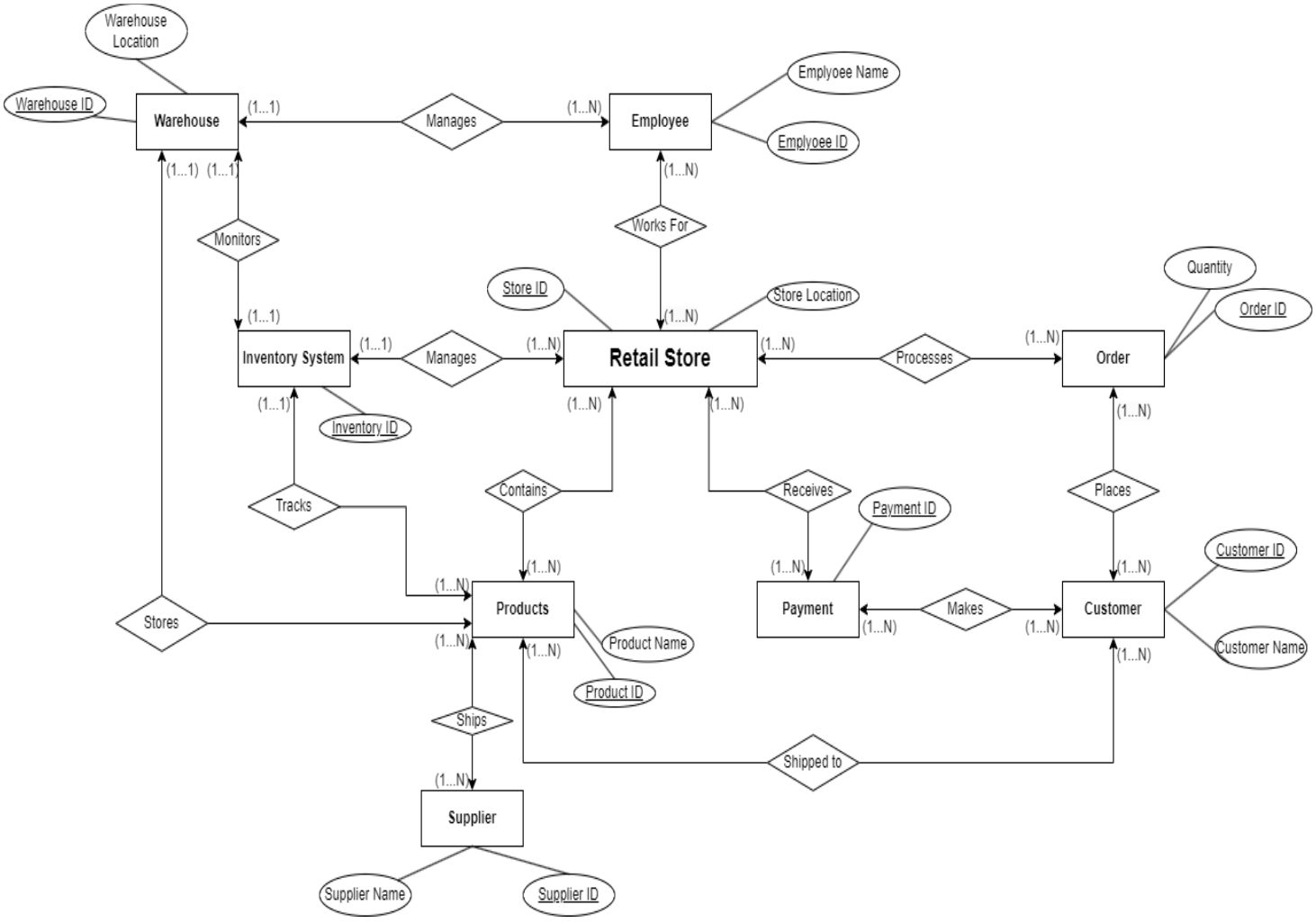
With the rapid growth of retail operations and the increasing need to manage stock efficiently, a well-structured Retail Store Inventory Management System has become essential. Retail stores must track inventory levels, orders, sales, and deliveries accurately, ensuring that the right products are available 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:

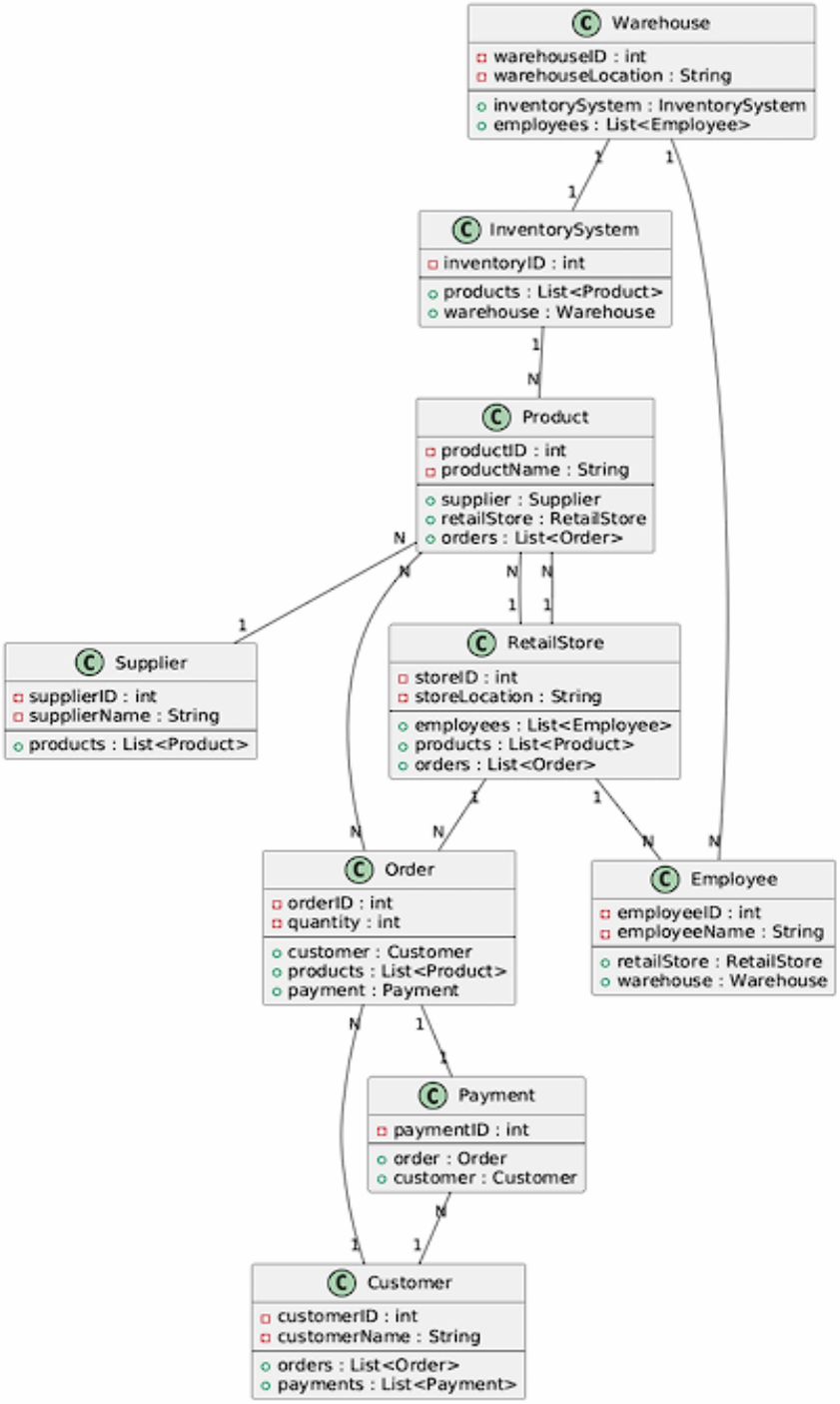
A well-structured Retail Store Inventory Management System is critical to meeting the dynamic demands of modern retail environments. As retail operations expand and customer expectations increase, it becomes essential to accurately track inventory levels, orders, sales, and deliveries. This system must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

Conceptual Model (EER and UML Model)

- **EER Model**



- **UML Model**


Warehouse
warehouseID : int
warehouseLocation : String
inventorySystem : InventorySystem
employees : List<Employee>
InventorySystem
inventoryID : int
products : List<Product>
warehouse : Warehouse
Product
productID : int
productName : String
supplier : Supplier
retailStore : RetailStore
orders : List<Order>
Supplier
supplierID : int
supplierName : String
products : List<Product>
RetailStore
storeID : int
storeLocation : String
employees : List<Employee>
products : List<Product>
orders : List<Order>
Order
orderID : int
quantity : int
customer : Customer
products : List<Product>
payment : Payment
Employee
employeeID : int
employeeName : String
retailStore : RetailStore
warehouse : Warehouse
Payment
paymentID : int
order : Order
customer : Customer
Customer
customerID : int
customerName : String
orders : List<Order>
payments : List<Payment>

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

Entities and Their Attributes:

1. Customer
   - Attributes:
     - Customer_ID (Primary Key)
     - Customer_Name

2. Order
   - Attributes:
     - Order_ID (Primary Key)
     - Quantity

3. Retail Store
   - Attributes:
     - Store_ID (Primary Key)
     - Store_Location

4. Employee
   - Attributes:
     - Employee_ID (Primary Key)
     - Warehouse_ID (Foreign Key)
     - Employee_Name

5. Warehouse
   - Attributes:
     - Warehouse_ID (Primary Key)
     - Warehouse_Location

6. Products
   - Attributes:
     - Product_ID (Primary Key)
     - Warehouse_ID (Foreign Key)
     - Inventory_ID (Foreign Key)
     - Product_Name

7. Supplier
   - Attributes:
     - Supplier_ID (Primary Key)
     - Supplier_Name

8. Payment
   - Attributes:
     - Payment_ID (Primary Key)
     - Payment_Type

9. Inventory System
   - Attributes:
     - Inventory_ID (Primary Key)

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. Warehouse to Inventory System (Manages):

- One warehouse manages one inventory system: (1..1)
- One inventory system is managed by one warehouse: (1..1)
- Final notation: 1:1

2. Warehouse to Employee (Manages):

- One warehouse is managed by many employees: (1..N)
- One employee manages one warehouse: (1..1)
- Final notation: 1:N

3. Warehouse to Inventory System (Monitors):

- One warehouse monitors one inventory system: (1..1)
- One inventory system is monitored by one warehouse: (1..1)
- Final notation: 1:1

4. Warehouse to Products (Stores):

- One warehouse stores many products: (1..N)
- One product is stored in one warehouse: (1..1)
- Final notation: 1:N

5. Retail Store to Warehouse (Manages):

- One retail store manages many warehouses: (1..N)
- One warehouse is managed by one retail store: (1..1)
- Final notation: 1:N

6. Retail Store to Order (Processes):

- One retail store processes many orders: (1..N)
- One order is processed by one retail store: (1..1)
- Final notation: 1:N

7. Retail Store to Products (Contains):

- One retail store contains many products: (1..N)
- One product can be contained in multiple retail stores: (1..N)
- Final notation: M:N

8. Retail Store to Payment (Receives):

- One retail store receives many payments: (1..N)
- One payment is received by one retail store: (1..1)
- Final notation: 1:N

9. Retail Store to Employee (Works For):

- One retail store employs many employees: (1..N)
- One employee works for one retail store: (1..1)
- Final notation: 1:N

10. Products to Inventory System (Tracks):

- One inventory system tracks many products: (1..N)
- One product can be tracked by multiple inventory systems: (1..N)
- Final notation: M:N

11. Products to Customer (Makes):

- One product is purchased by many customers: (1..N)
- One customer purchases many products: (1..N)
- Final notation: M:N

12. Products to Supplier (Ships):

- One supplier ships many products: (1..N)
- One product is shipped by one supplier: (1..1)
- Final notation: 1:N

13. Products to Retail Store (Contained in):

- One product can be contained in many retail stores: (1..N)
- One retail store contains many products: (1..N)
- Final notation: M:N

14. Order to Retail Store (Processes):

- One retail store processes many orders: (1..N)
- One order is processed by one retail store: (1..1)
- Final notation: 1:N

15. Order to Customer (Places):

- One customer places many orders: (1..N)
- One order is placed by one customer: (1..1)
- Final notation: 1:N

16. Payment to Retail Store (Receives):

- One retail store receives many payments: (1..N)
- One payment is linked to one retail store: (1..1)
- Final notation: 1:N

17. Supplier to Products (Ships):

- One supplier ships many products: (1..N)
- One product is shipped by one supplier: (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:

Customer (Customer ID, Customer Name)

Order (Order ID, Quantity)

Processes(*Order ID, Store ID*)

Order ID: foreign key refers to order ID in relation Order. NULL not allowed, on delete/update cascade.
Store ID: foreign key refers to *Store ID* in relation Retail store. NULL not allowed, on delete/update cascade.

Places(*Order ID, Customer ID*)

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.
Order ID: foreign key refers to order ID in relation Order. NULL not allowed, on delete/update cascade.

Retail(StoreID, Store location)

Employee(Emplyoee ID, *warehouse ID,* Employee Name)

Warehouse ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Works For(*Employee ID, Store ID*)

StoreID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.
Employee ID: foreign key refers to *Employee* ID in relation *Employee*. NULL not allowed, on delete/update cascade.

Warehouse(Warehouse ID, Warehouse location)

Products(ProductID, *Warehouse ID , Inventory ID*, Product Name)

Warehouse ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.
Inventory ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Contains(*Product ID, Store ID*)

Store ID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.
Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on

delete/update cascade.

Ships(*Product ID, Supplier ID)*

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.
Supplier ID: foreign key refers to Supplier ID in relation Supplier. NULL not allowed, on delete/update cascade.

Shipped To(*Product ID, customer ID)*

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.
Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.

Makes (*Payment ID, Customer ID*)

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.
Payment ID: foreign key refers to Payment ID in relation Payment. NULL not allowed, on delete/update cascade.

Payment(Payment ID,Payment Type)

Receive (*Store ID, Payment ID*)

StoreID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.
Payment ID: foreign key refers to Payment ID in relation Payment. NULL not allowed, on delete/update cascade.

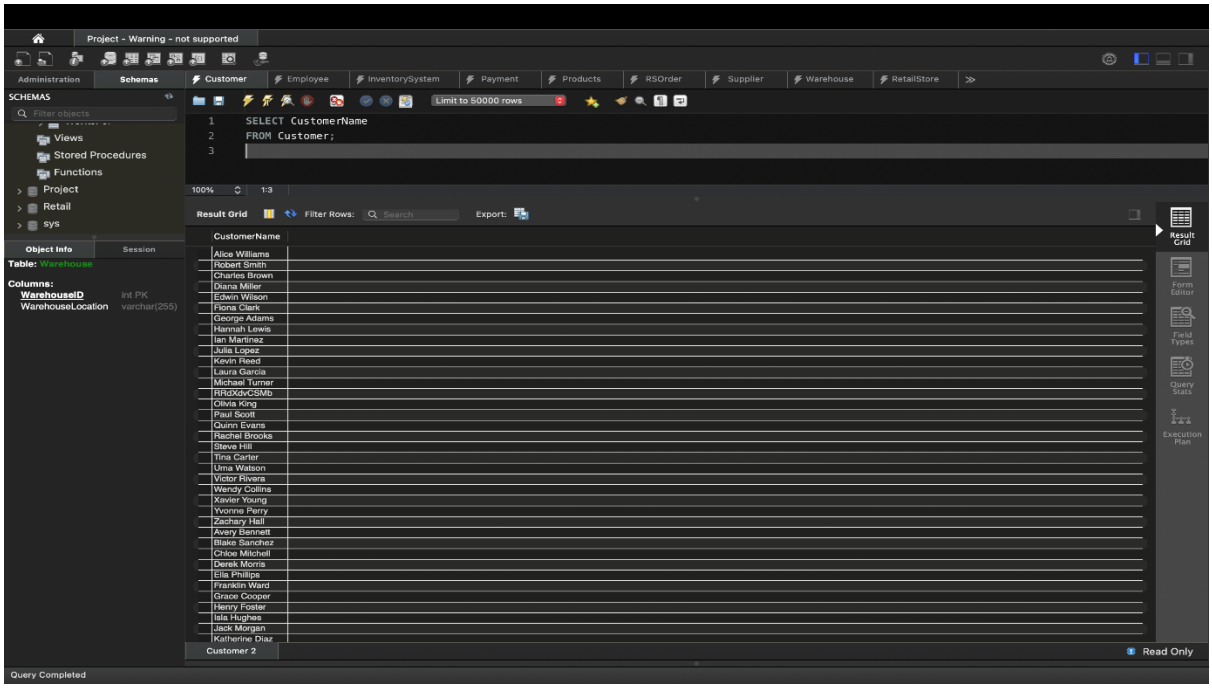
Inventory System(InventoryID)

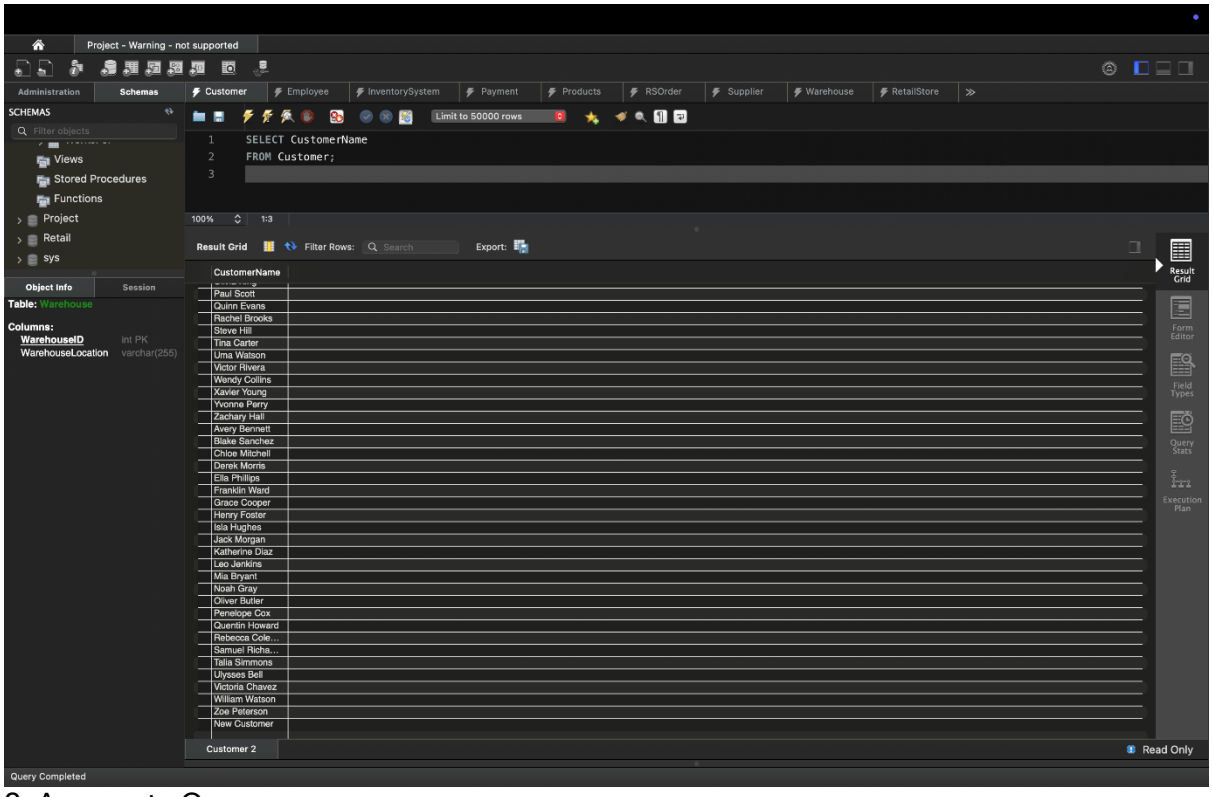
Supplier(Supplier ID, Supplier Name)

Implementation in MySQL Queries:

1. Simple Query

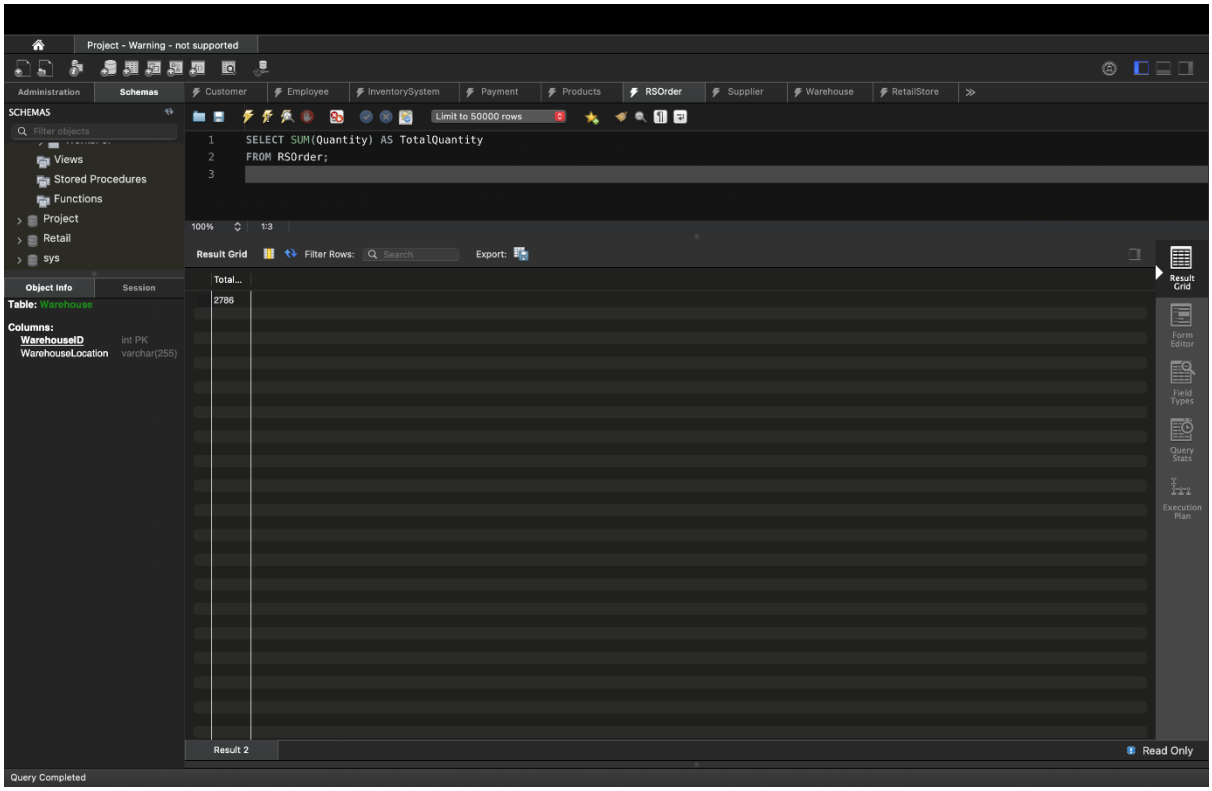
```
SELECT CustomerName
FROM Customer
```





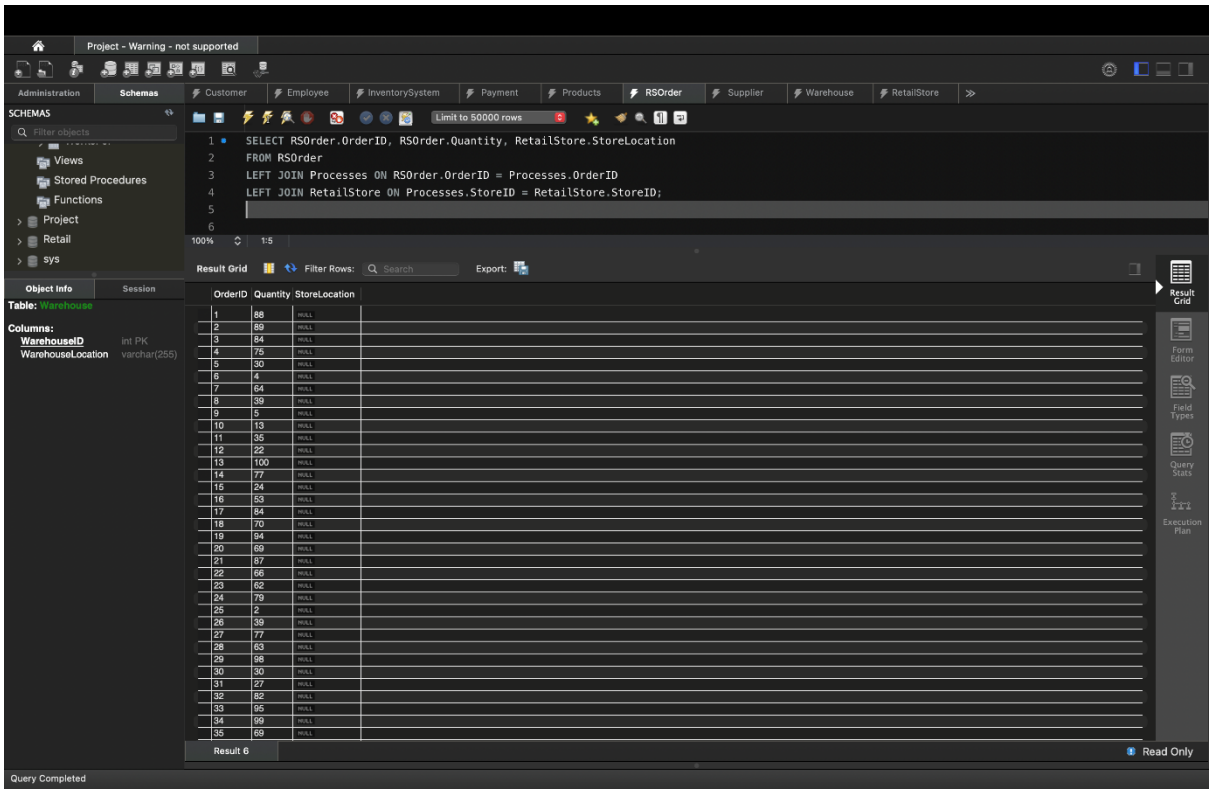
2. Aggregate Query

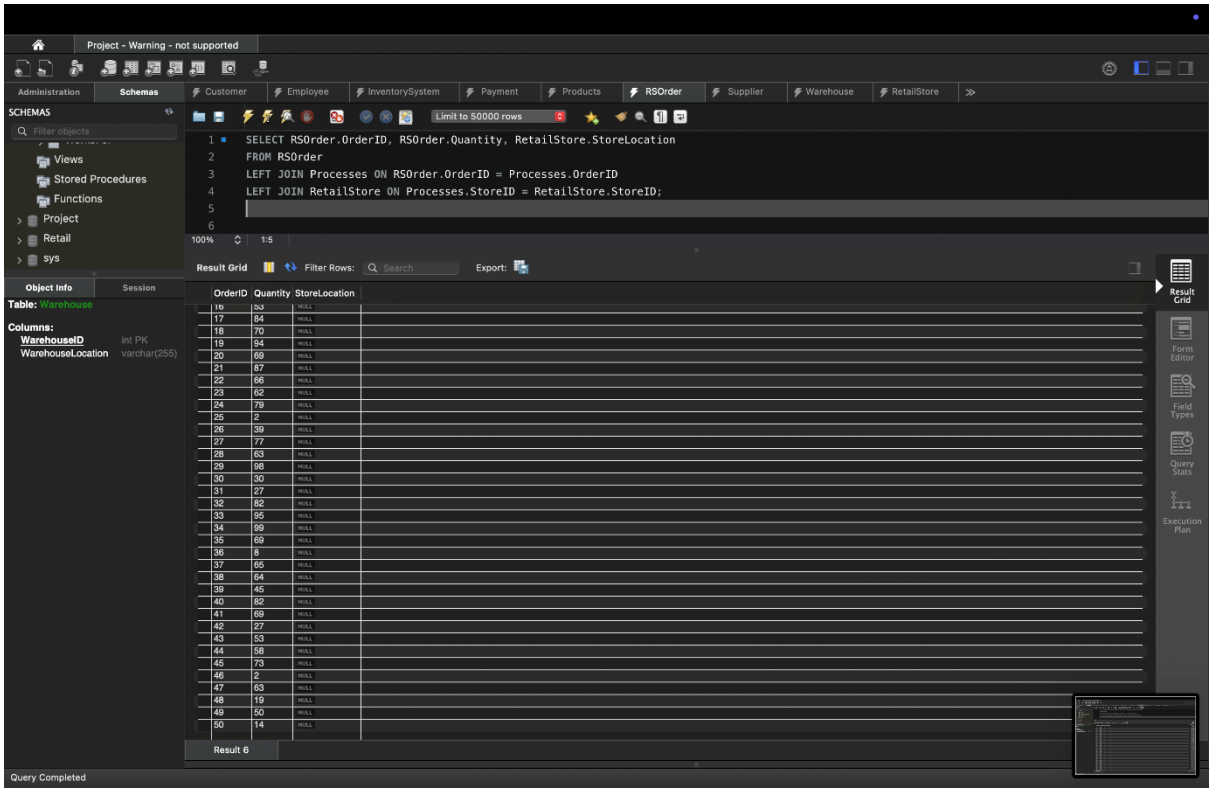
```
SELECT SUM(Quantity) AS TotalQuantity
FROM RSOrder;
```



3. Inner Join Query

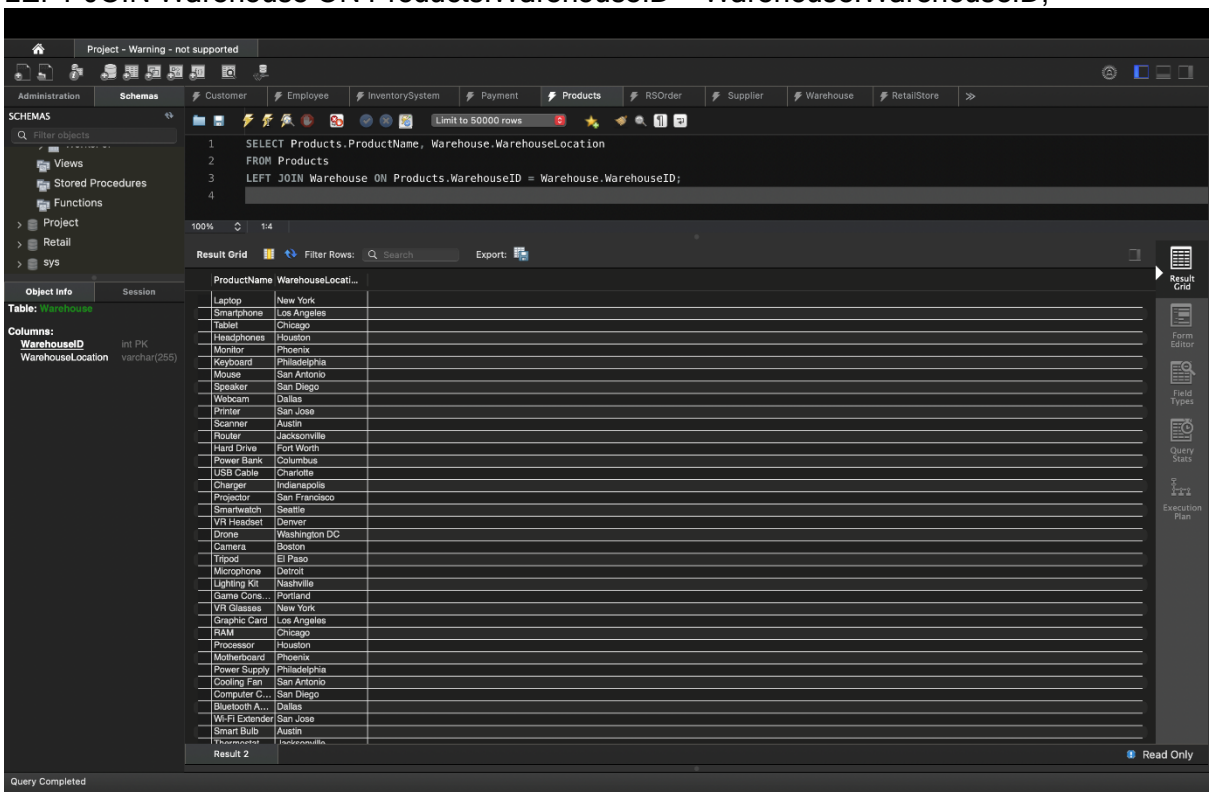
```
SELECT RSOrder.OrderID, RSOrder.Quantity, RetailStore.StoreLocation
FROM RSOrder
INNER JOIN Processes ON RSOrder.OrderID = Processes.OrderID
INNER JOIN RetailStore ON Processes.StoreID = RetailStore.StoreID;
```

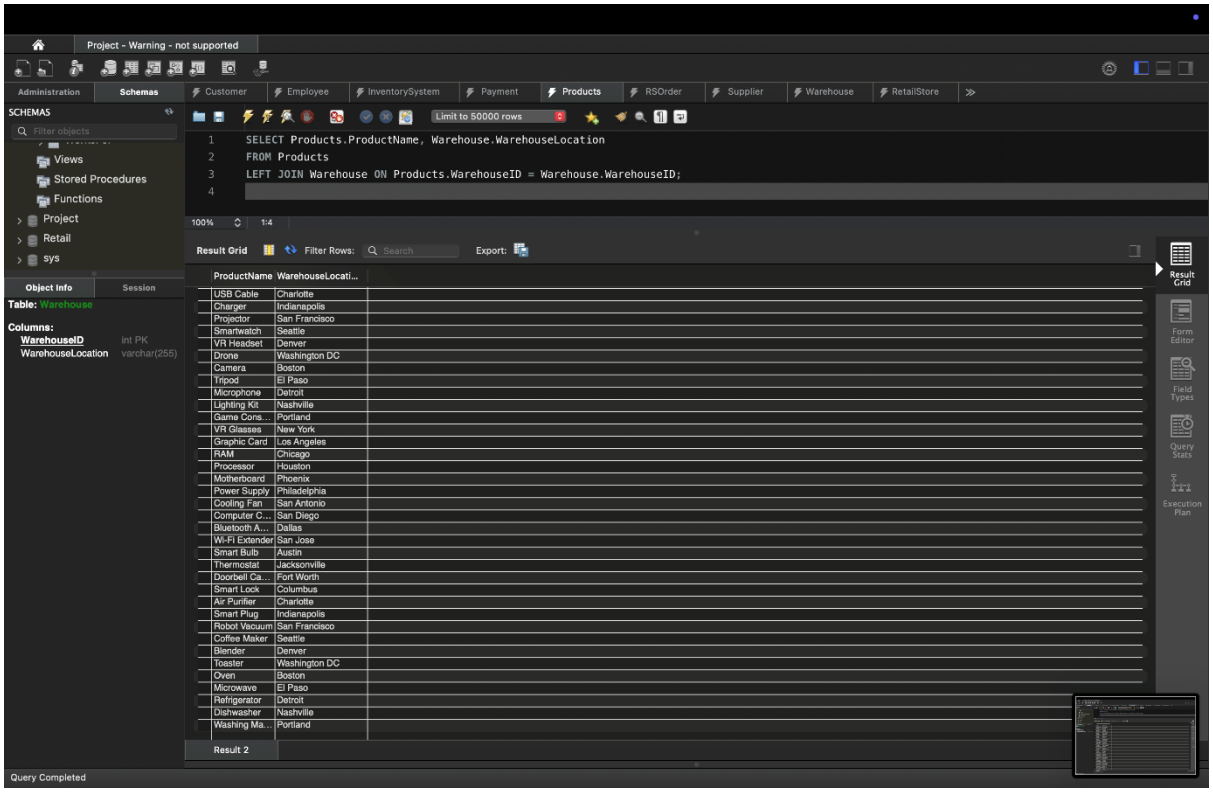




4. Left Outer Join

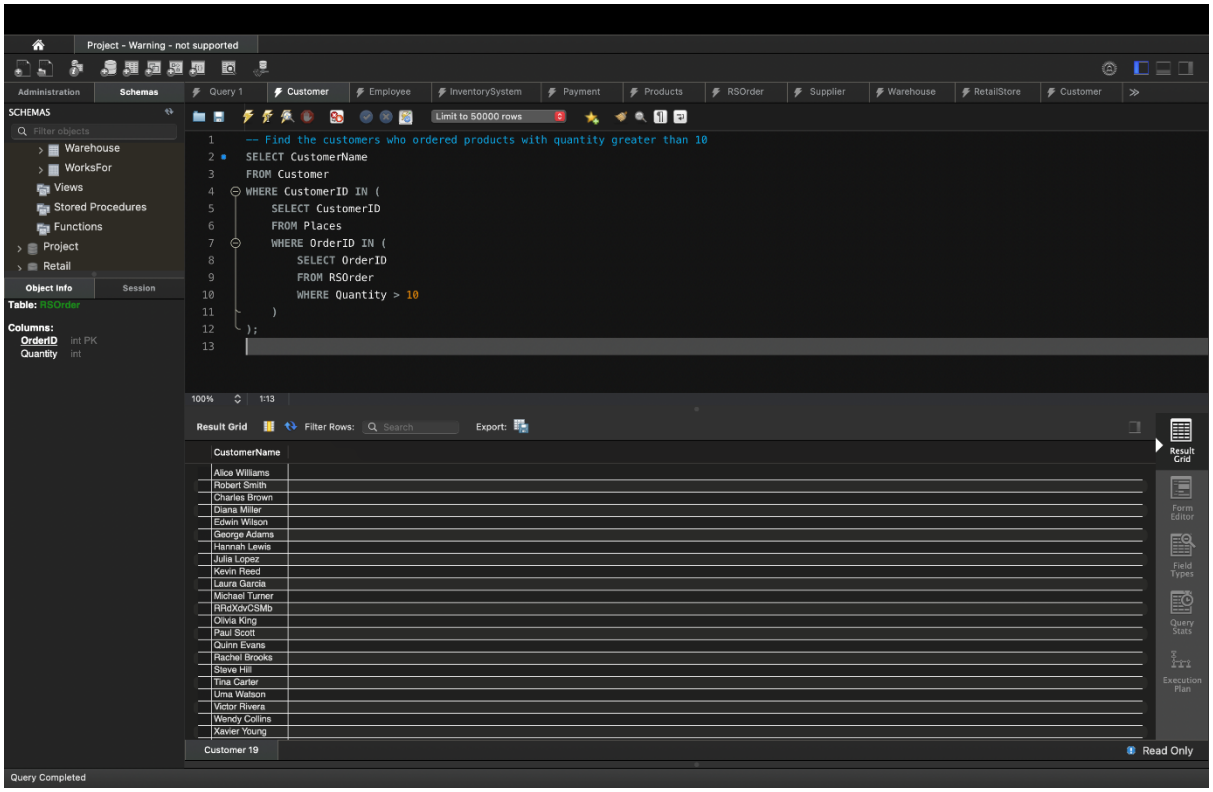
SELECT Products.ProductName, Warehouse.WarehouseLocation
FROM Products
LEFT JOIN Warehouse ON Products.WarehouseID = Warehouse.WarehouseID;

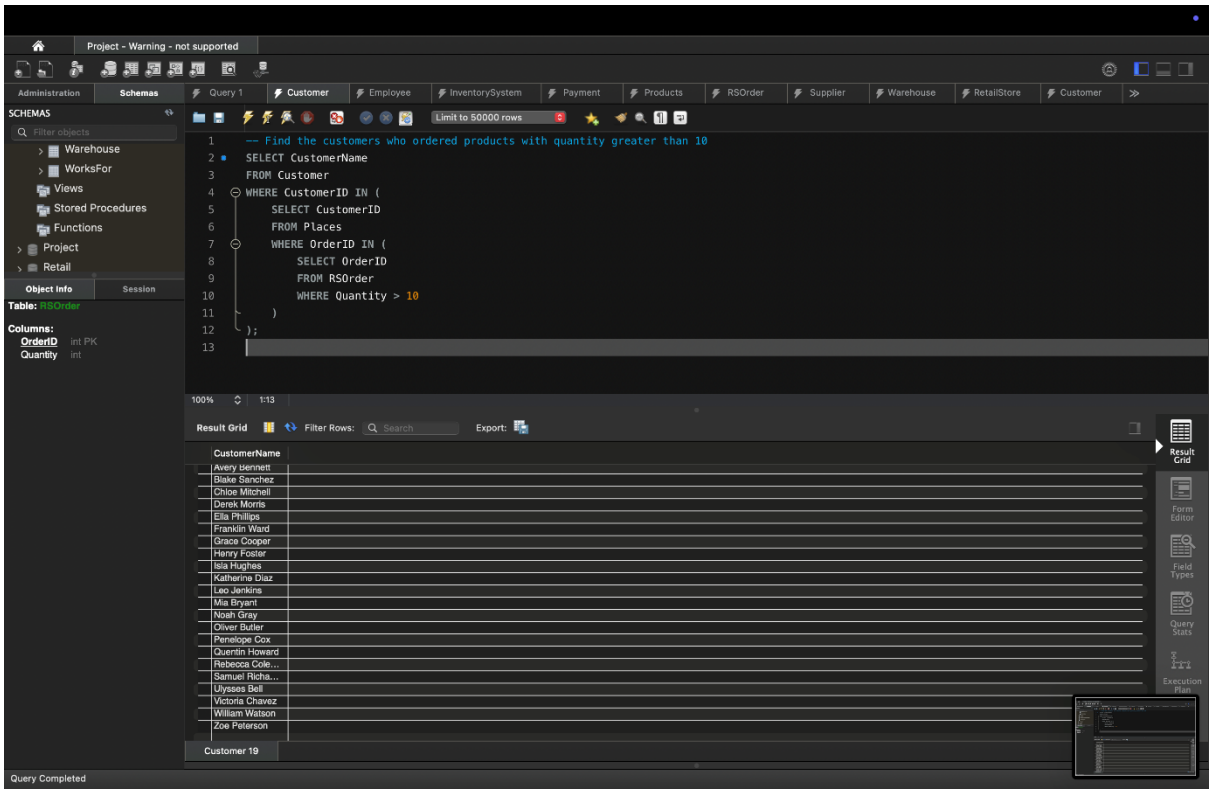




5. Nested Query (Subquery in WHERE):

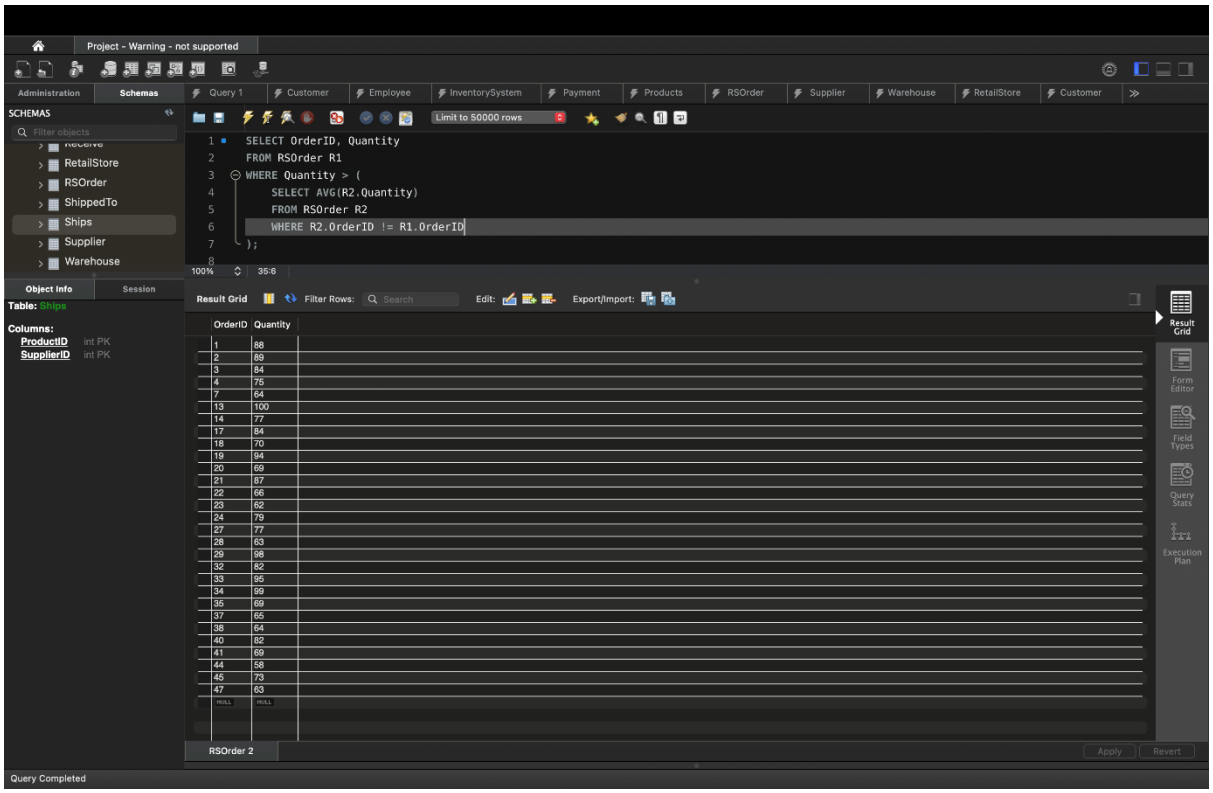
```
SELECT CustomerName
FROM Customer
WHERE CustomerID IN (
    SELECT CustomerID
    FROM Places
    WHERE OrderID IN (
        SELECT OrderID
        FROM RSOrder
        WHERE Quantity > 10
    )
);
```


Project - Warning - not supported
Administration
Schemas
Query 1
Customer
Employee
InventorySystem
Payment
Products
RSOrder
Supplier
Warehouse
RetailStore
Customer
SCHEMAS
Filter objects
Warehouse
WorksFor
Views
Stored Procedures
Functions
Project
Retail
Object Info
Session
Table: RSOrder
Columns:
OrderID int PK
Quantity int
Limit to 50000 rows
-- Find the customers who ordered products with quantity greater than 10
SELECT CustomerName
FROM Customer
WHERE CustomerID IN (
SELECT CustomerID
FROM Places
WHERE OrderID IN (
SELECT OrderID
FROM RSOrder
WHERE Quantity > 10
)
);
100%
1:13
Result Grid
Filter Rows:
Search
Export:
CustomerName
Alice Williams
Robert Smith
Charles Brown
Diana Miller
Edwin Wilson
George Adams
Hannah Lewis
Julia Lopez
Kevin Reed
Laura Garcia
Michael Turner
RRdXtvCSMb
Olivia King
Paul Scott
Quinn Evans
Rachel Brooks
Steve Hill
Tina Carter
Uma Watson
Victor Rivera
Wendy Collins
Xavier Young
Customer 19
Read Only
Query Completed


Project - Warning - not supported
Administration
Schemas
Query 1
Customer
Employee
InventorySystem
Payment
Products
RSOrder
Supplier
Warehouse
RetailStore
Customer
SCHEMAS
Filter objects
Warehouse
WorksFor
Views
Stored Procedures
Functions
Project
Retail
Object Info
Session
Table: RSOrder
Columns:
OrderID int PK
Quantity int
Limit to 50000 rows
-- Find the customers who ordered products with quantity greater than 10
SELECT CustomerName
FROM Customer
WHERE CustomerID IN (
SELECT CustomerID
FROM Places
WHERE OrderID IN (
SELECT OrderID
FROM RSOrder
WHERE Quantity > 10
)
);
100%
1:13
Result Grid
Filter Rows:
Search
Export:
CustomerName
Avery Bennett
Blake Sanchez
Chloe Mitchell
Derek Morris
Ella Phillips
Franklin Ward
Grace Cooper
Henry Foster
Isla Hughes
Katherine Diaz
Leo Jenkins
Mia Bryant
Noah Gray
Oliver Butler
Penelope Cox
Quentin Howard
Rebecca Cole...
Samuel Richa...
Ulysses Bell
Victoria Chavez
William Watson
Zoe Peterson
Customer 19
Query Completed

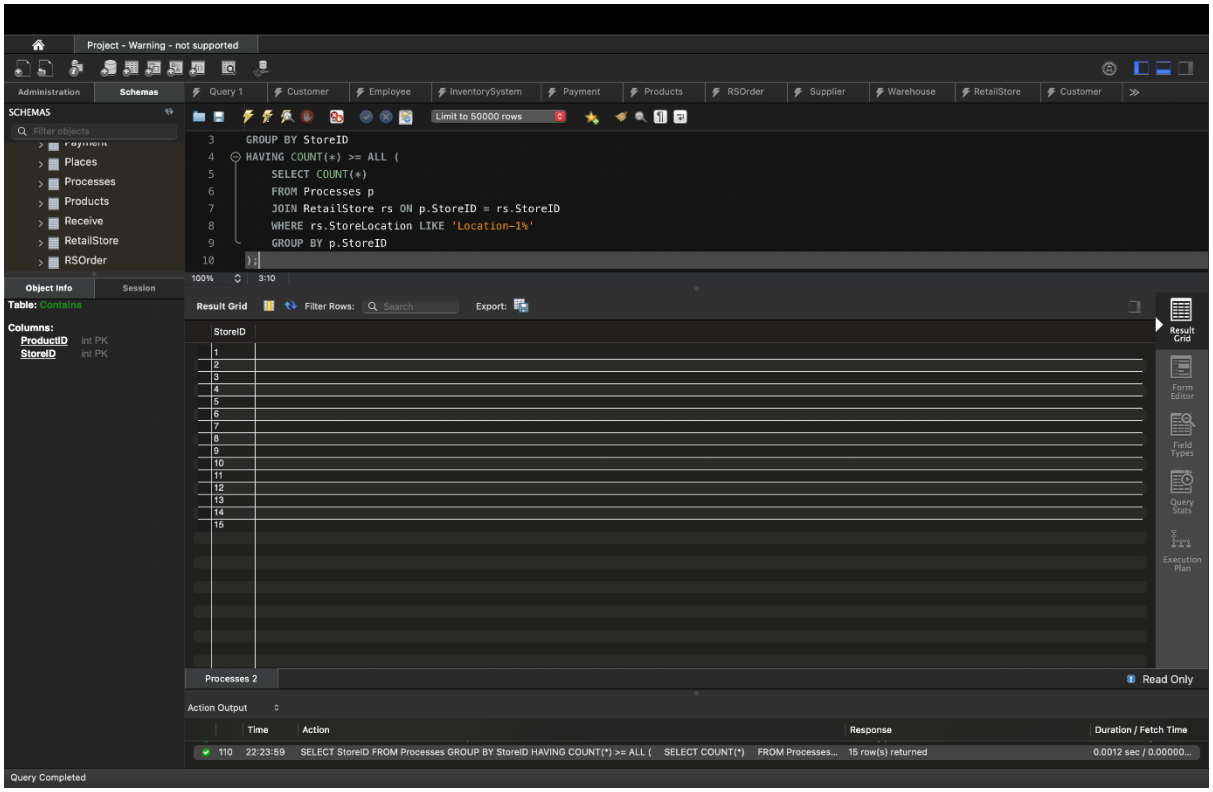
6. correlated query

```
SELECT OrderID, Quantity
FROM RSOrder R1
WHERE Quantity > (
    SELECT AVG(R2.Quantity)
    FROM RSOrder R2
    WHERE R2.OrderID != R1.OrderID
);
```



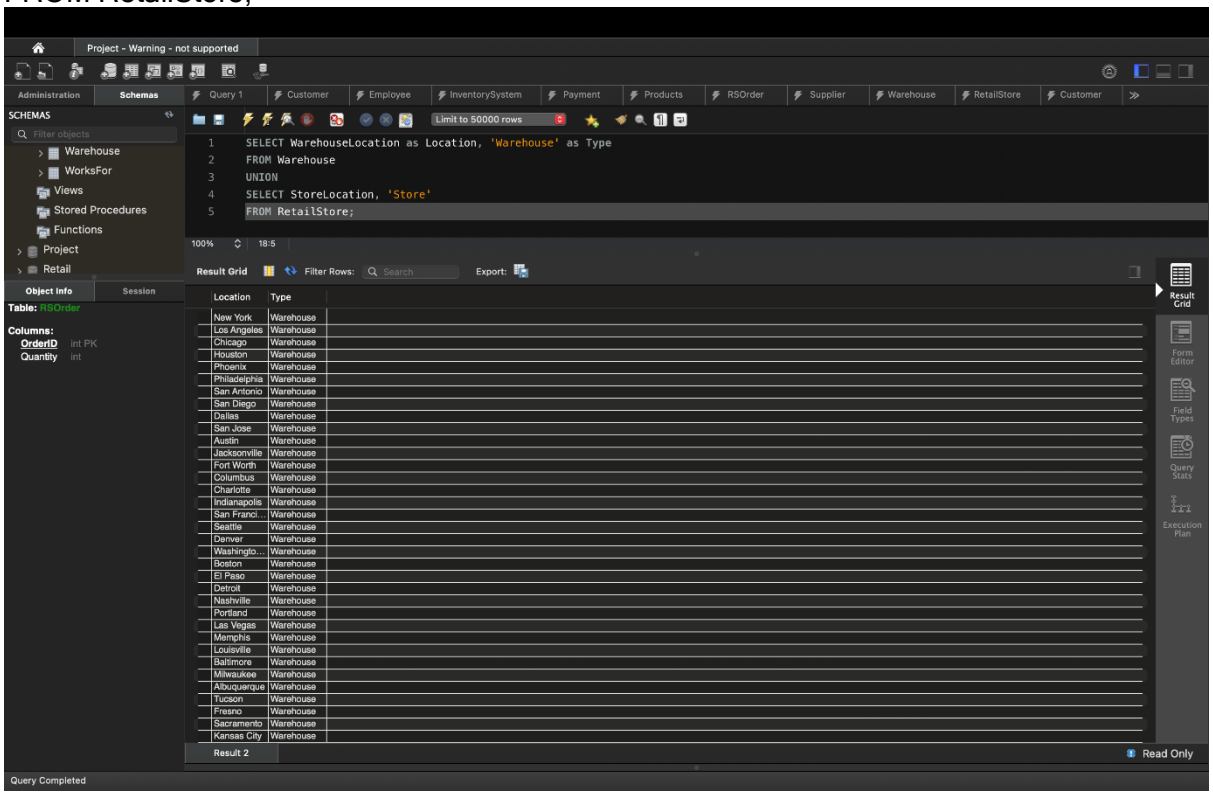
7. ALL Operator

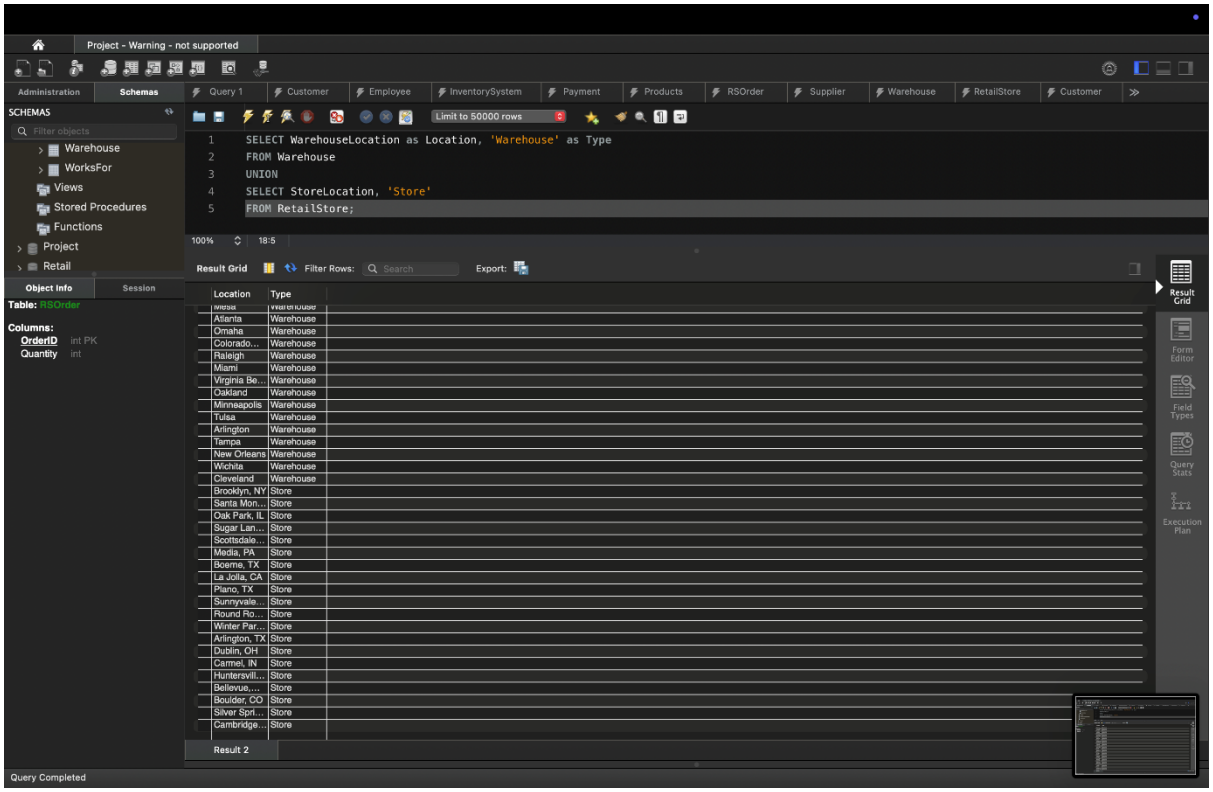
```
GROUP BY StoreID
HAVING COUNT(*) >= ALL (
    SELECT COUNT(*)
    FROM Processes p
    JOIN RetailStore rs ON p.StoreID = rs.StoreID
    WHERE rs.StoreLocation LIKE 'Location-1%'
    GROUP BY p.StoreID
);
```



8. UNION:

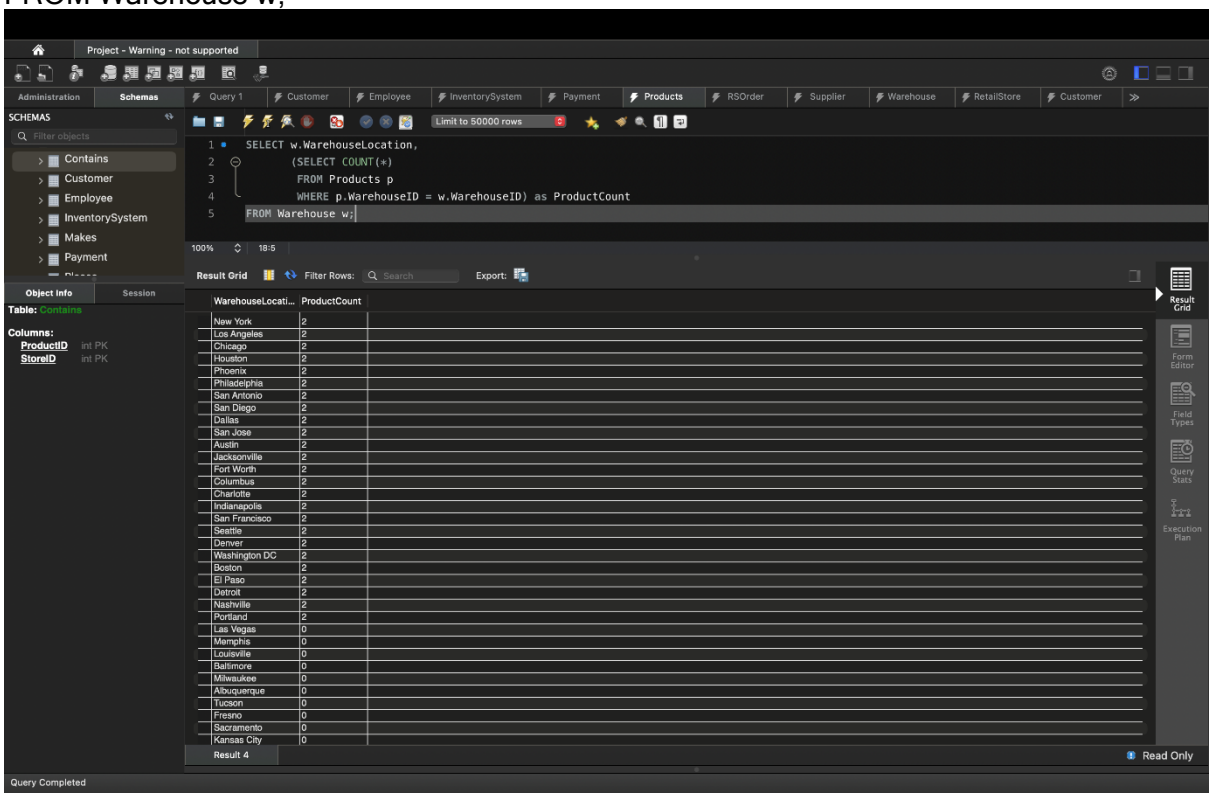
SELECT WarehouseLocation as Location, 'Warehouse' as Type
FROM Warehouse
UNION
SELECT StoreLocation, 'Store'
FROM RetailStore;

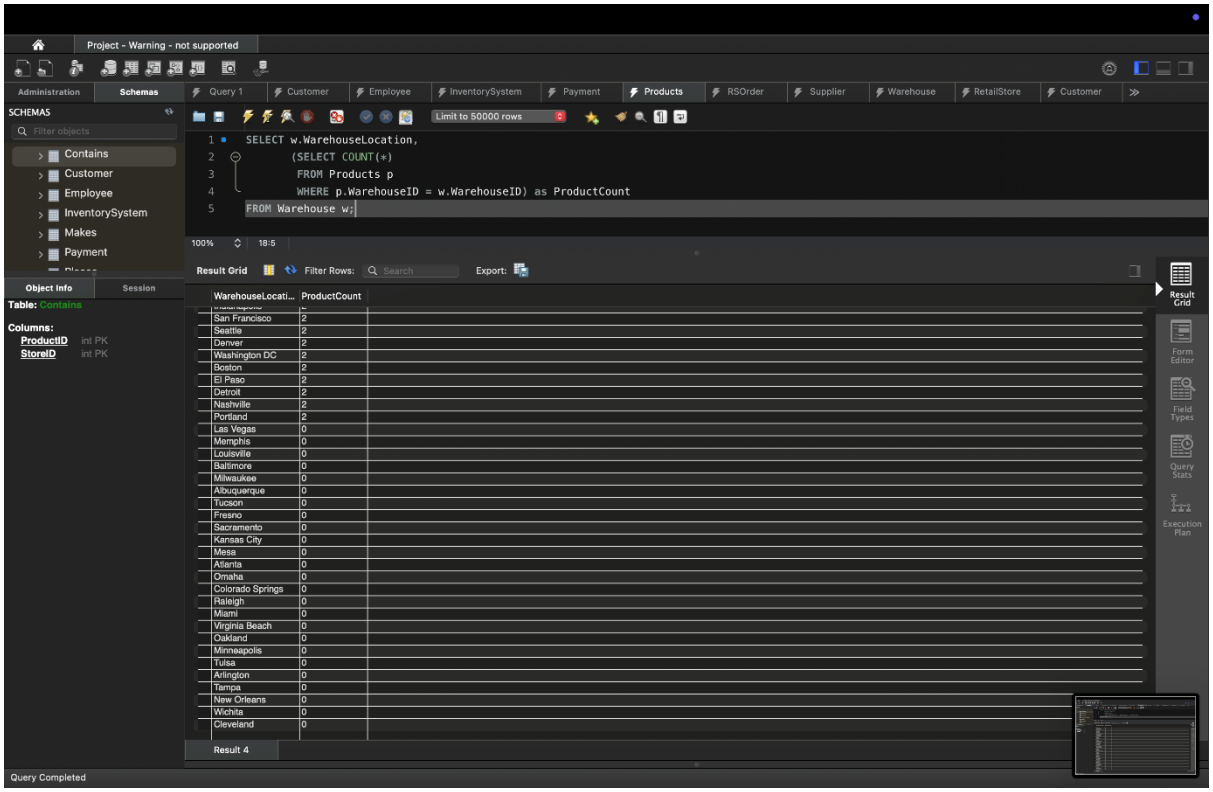




9. Subquery

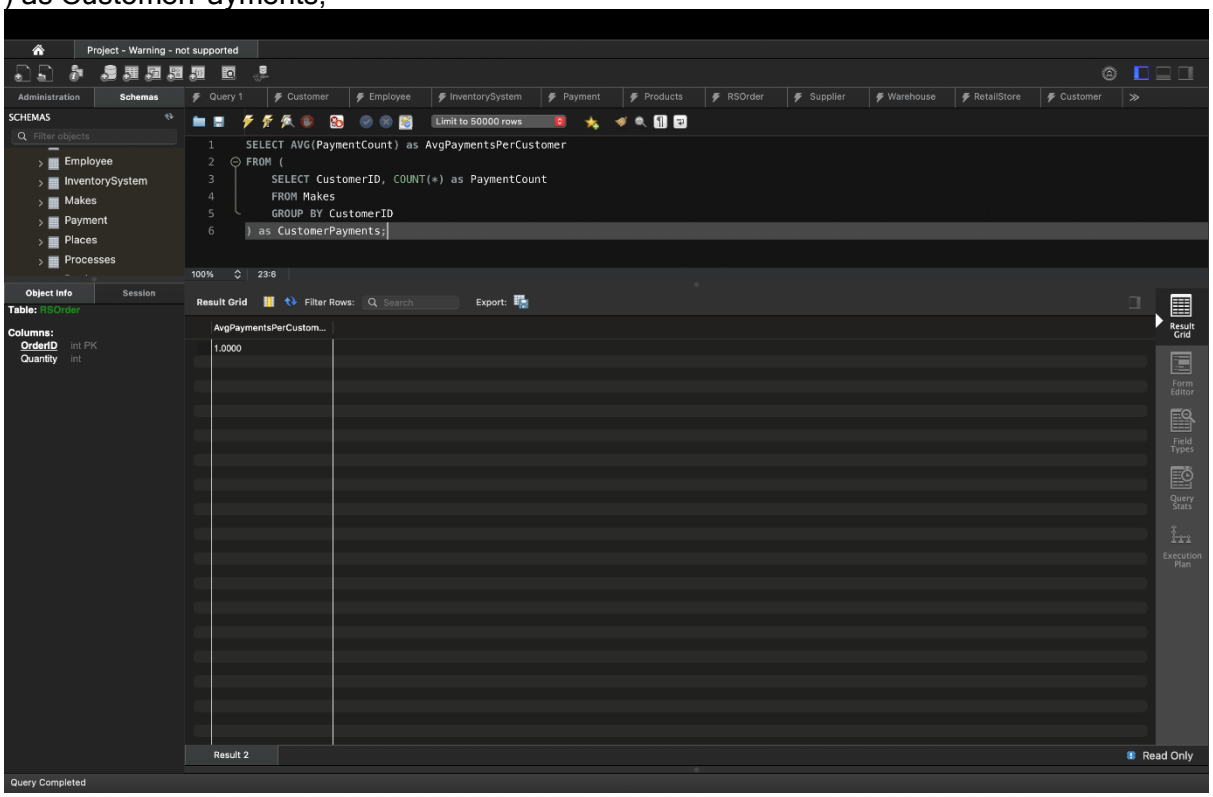
```
SELECT w.WarehouseLocation,
       (SELECT COUNT(*)
        FROM Products p
        WHERE p.WarehouseID = w.WarehouseID) as ProductCount
FROM Warehouse w;
```





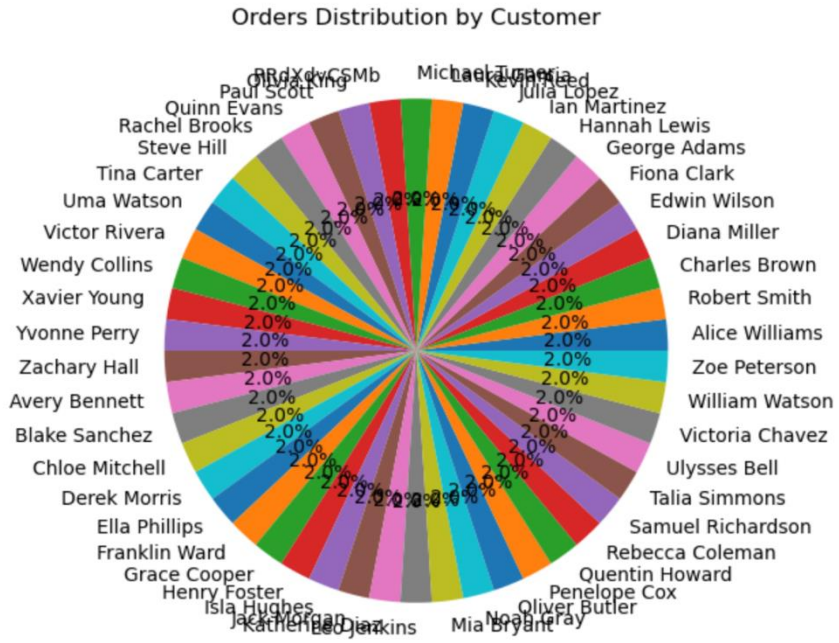
10. Subquery in FROM:

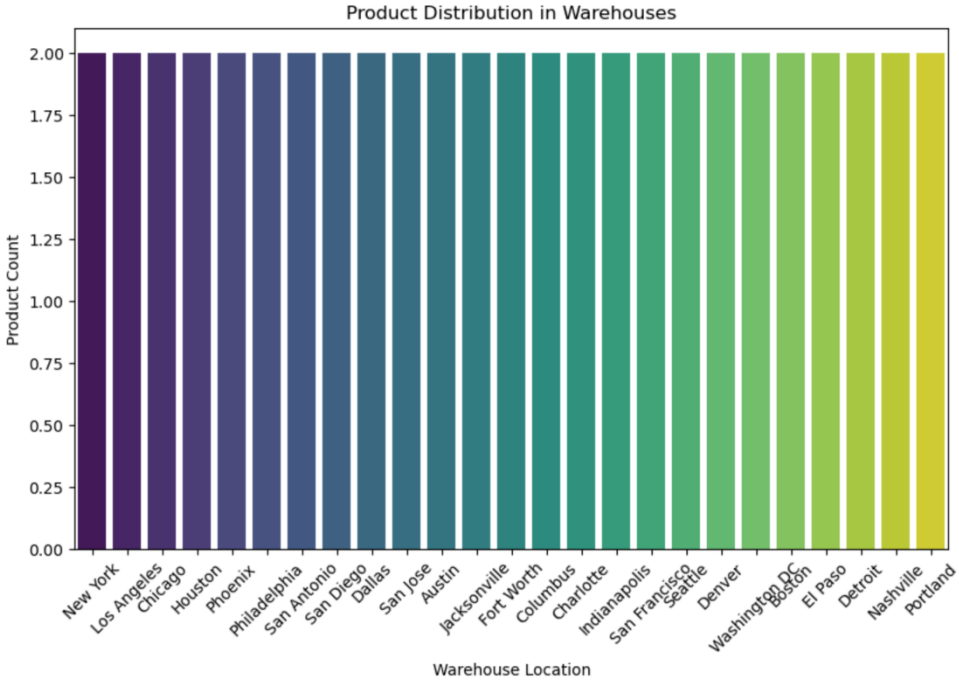
```
SELECT AVG(PaymentCount) as AvgPaymentsPerCustomer
FROM (
    SELECT CustomerID, COUNT(*) as PaymentCount
    FROM Makes
    GROUP BY CustomerID
) as CustomerPayments;
```

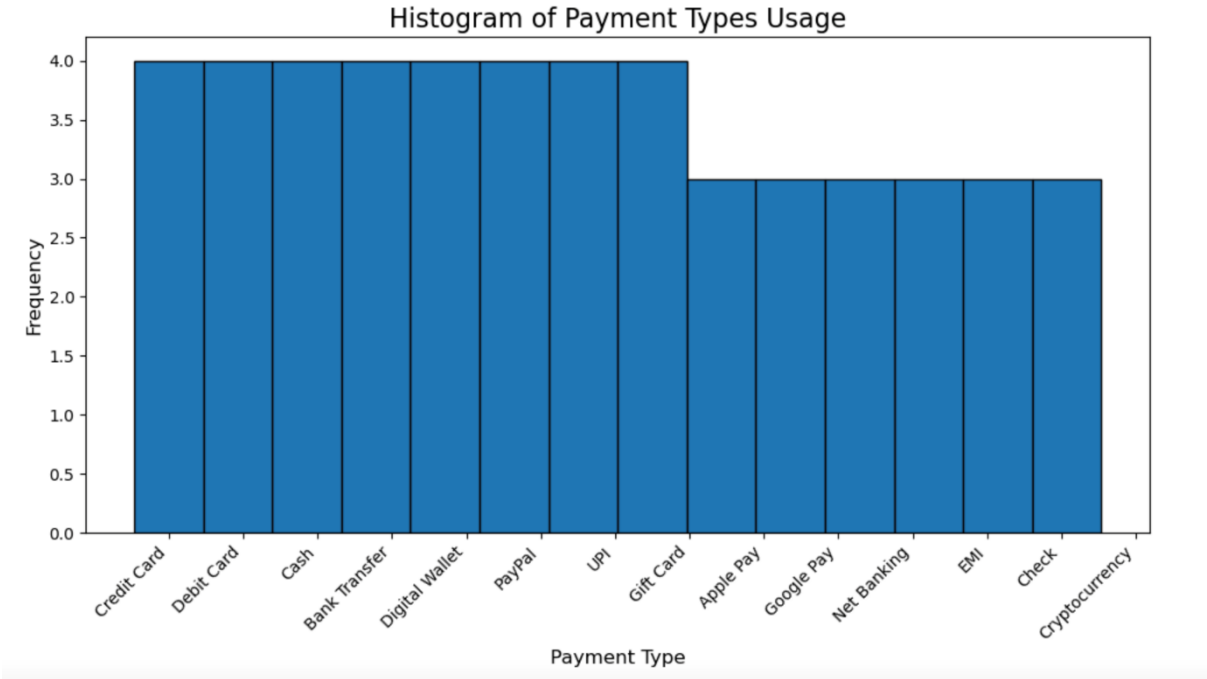


Database Access via Python

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

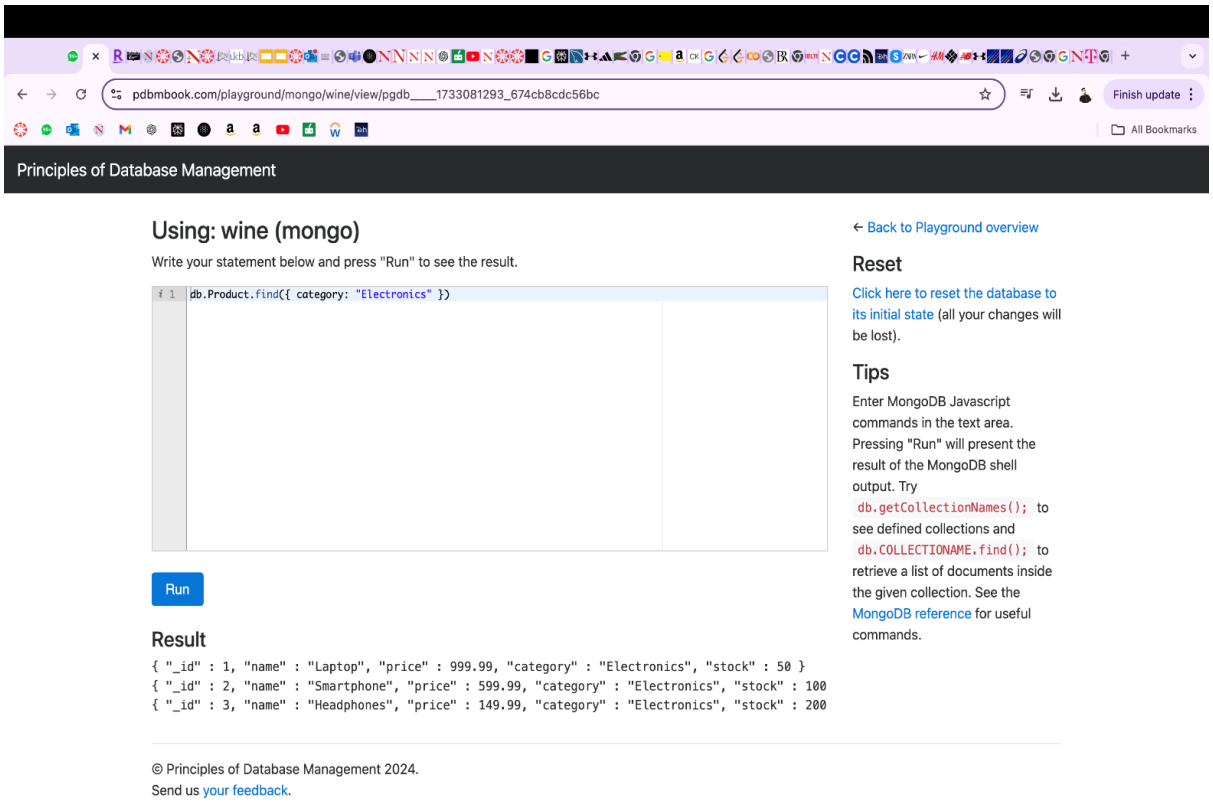
FOR CREATING AND INSERTING TABLE CODE USED:

```
// 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" }
])

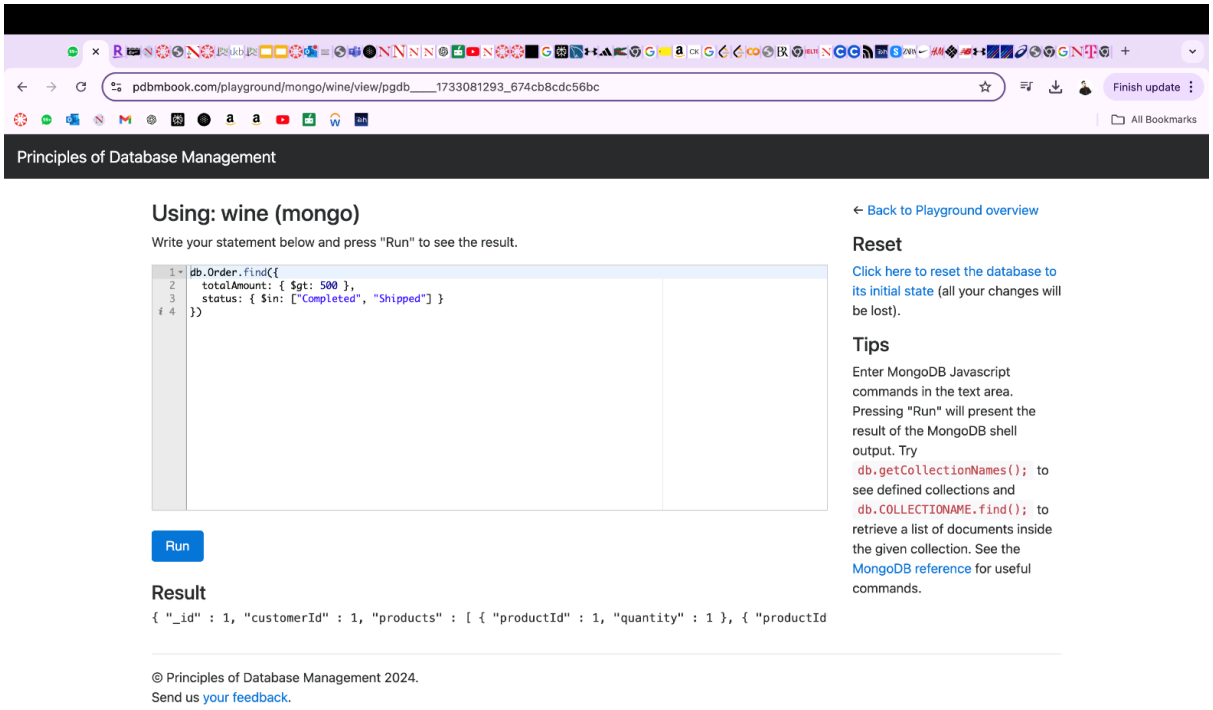
Query 1: SIMPLE QUERY:

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



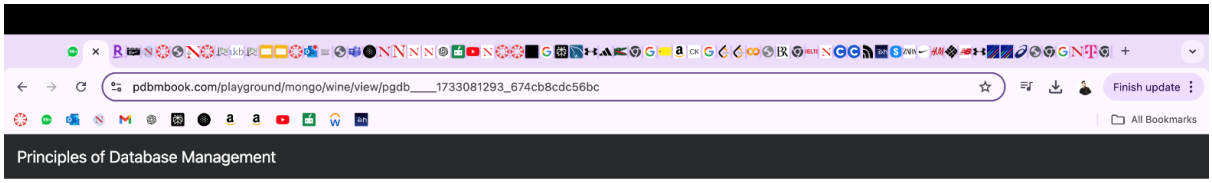
Query 2 : MORE COMPLEX QUERY:

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



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
    }
  }
])
```



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: {
      _id: 1
```

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.COLLECTIONNAME.find();` to retrieve a list of documents inside the given collection. See the MongoDB reference for useful commands.

Send us your feedback.