Database CA Report

Part 1- Conceptual Design

1. Introduction

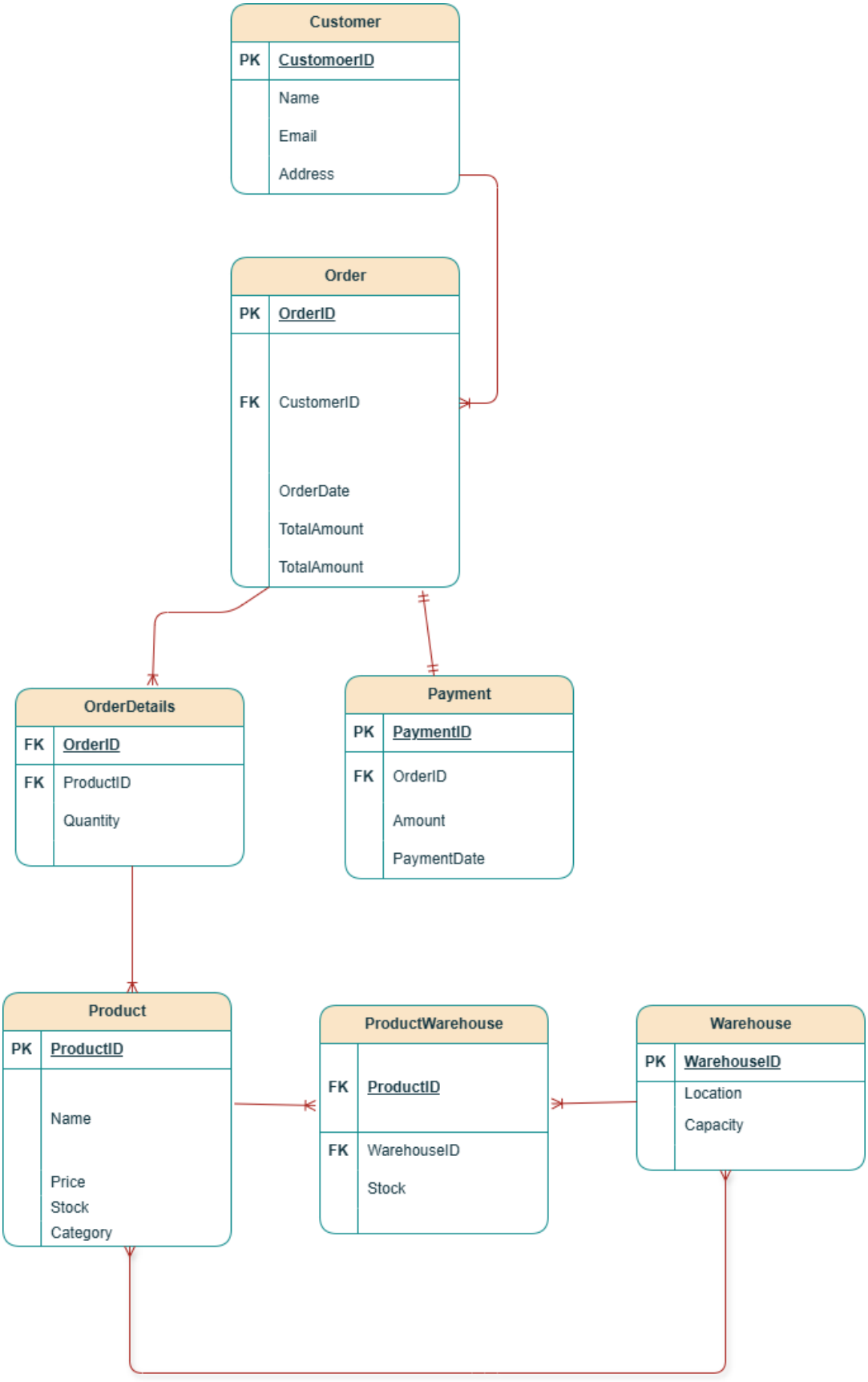
To demonstrate the use of databases, I wish to create a collection and launch a website for jewellery designs. This collection will include pieces like: earrings, necklaces, bracelets, and rings. These will be handcrafted, and the target audience will be women. It can be difficult to manage information such as orders, inventory etc. Therefore, building a database system to keep everything organised is important.

The information managed by the system is:

1. **Customer Information**: Store details like contact information, preferences, and purchase history.
2. **Product Management**: Classify jewellery items into categories (e.g., earrings, necklaces), along with information such as inventory levels, prices, and descriptions.
3. **Order and Payment Tracking**: Allow customers to order multiple items, enable inventory updates, and manage payment methods and statuses.
4. **Sales Insights**: Generate monthly and quarterly reports to highlight trends, like the most popular items, and aid decisions on inventory and marketing.
5. **Design Records**: Maintain a record of design concepts and production costs for each piece.

By implementing this database system the website can run smoothly. This setup will also free up more time for what truly matters i.e designing beautiful pieces that align with the customers' interests.

2. ER design


Customer
PK
CustomoerID
Name
Email
Address
Order
PK
OrderID
FK
CustomerID
OrderDate
TotalAmount
TotalAmount
OrderDetails
FK
OrderID
FK
ProductID
Quantity
Payment
PK
PaymentID
FK
OrderID
Amount
PaymentDate
Product
PK
ProductID
Name
Price
Stock
Category
ProductWarehouse
FK
ProductID
FK
WarehouseID
Stock
Warehouse
PK
WarehouseID
Location
Capacity

3. Conversion to Relational model

3.1 Introduction

The conceptual design of the online jewelry shop database was converted into a relational model to create normalized tables that support efficient data storage and retrieval. The relational model was designed to meet the requirements of Third Normal Form (3NF) to ensure minimal data redundancy and maintain data integrity. This section details the relational model and its SQL implementation.

3.2 Entities and Relationships

3.2.1. Customer Entity

- **Description**: Stores customer information for the jewelry shop.
- **Attributes**:
  - CustomerID (PK) – Unique identifier for each customer.
  - Name – Name of the customer.
  - Email – Email address of the customer.
  - Address – Physical address of the customer.
- **Relationships**:
  - **One-to-Many**: A customer can place multiple orders. (CustomerID is referenced in the Order table as a foreign key.)

3.2.2. Order Entity

- **Description**: Represents customer orders in the jewelry shop.
- **Attributes**:
  - OrderID (PK) – Unique identifier for each order.
  - CustomerID (FK) – Links the order to a customer.
  - OrderDate – Date the order was placed.
  - TotalAmount – Total monetary value of the order.
- **Relationships**:
  - **One-to-Many**: A customer can place multiple orders, but each order belongs to one customer.
  - **One-to-Many**: An order can have multiple line items, as represented in the OrderDetails table.

3.2.3. Product Entity

- **Description**: Stores details about the jewelry products.
- **Attributes**:
    - ProductID (PK) – Unique identifier for each product.
    - Name – Name of the product.
    - Price – Price of the product.
    - Stock – Current stock level of the product.
    - Category – Category of the product (e.g., necklace, bracelet, ring).
- **Relationships**:
    - **Many-to-Many**: Products are included in multiple orders, represented by the OrderDetails table.
    - **Many-to-Many**: Products are stored in multiple warehouses, represented by the ProductWarehouse table.

3.2.4. OrderDetails Entity

- **Description**: Serves as a junction table for the many-to-many relationship between Order and Product.
- **Attributes**:
    - OrderID (FK) – Links to the Order table.
    - ProductID (FK) – Links to the Product table.
    - Quantity – Number of units of the product in the order.
- **Relationships**:
    - **Many-to-One**: Each line item belongs to one order.
    - **Many-to-One**: Each line item references one product.

3.2.5. Payment Entity

- **Description**: Stores payment information for orders.
- **Attributes**:
    - PaymentID (PK) – Unique identifier for each payment.
    - OrderID (FK) – Links the payment to an order.
    - Amount – Total payment amount.
    - PaymentDate – Date the payment was made.
- **Relationships**:
    - **One-to-One**: Each order is associated with one payment.

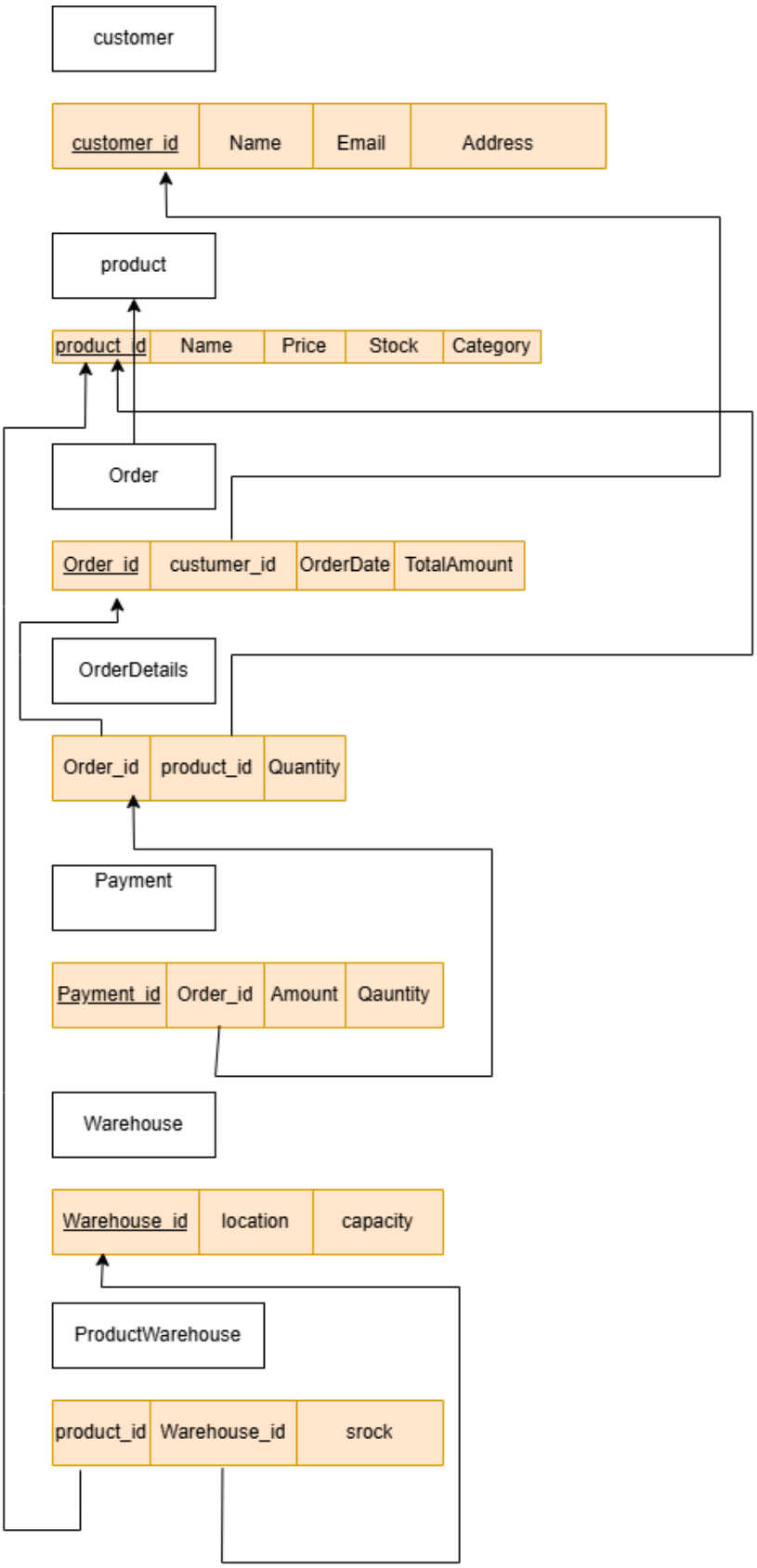
3.2.6. Warehouse Entity

- **Description**: Stores information about the physical locations where products are stored.
- **Attributes**:
    - WarehouseID (PK) – Unique identifier for each warehouse.
    - Location – Physical location of the warehouse.
    - Capacity – Maximum storage capacity of the warehouse.
- **Relationships**:
    - **Many-to-Many**: A warehouse can store multiple products, and a product can be stored in multiple warehouses. (ProductWarehouse table represents this relationship.)

3.2.7. ProductWarehouse Entity

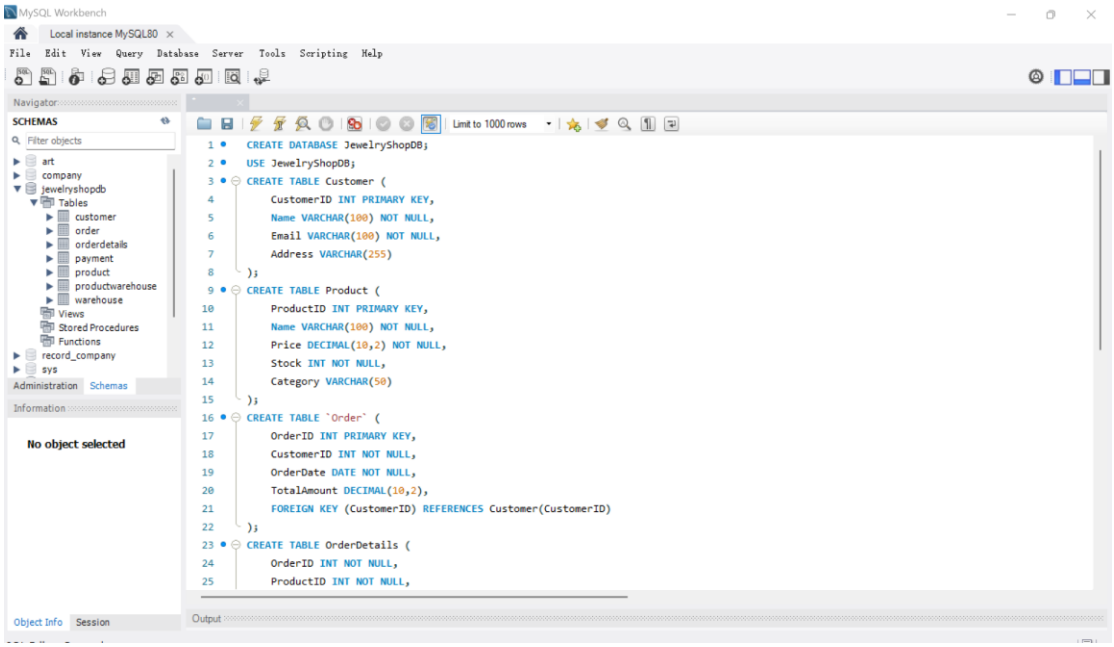
- **Description**: Serves as a junction table for the many-to-many relationship between Product and Warehouse.
- **Attributes**:
    - ProductID (FK) – Links to the Product table.
    - WarehouseID (FK) – Links to the Warehouse table.
    - Stock – Number of units of the product stored in the warehouse.
- **Relationships**:
    - **Many-to-One**: A product can be stored in multiple warehouses.
    - **Many-to-One**: A warehouse can store multiple products.

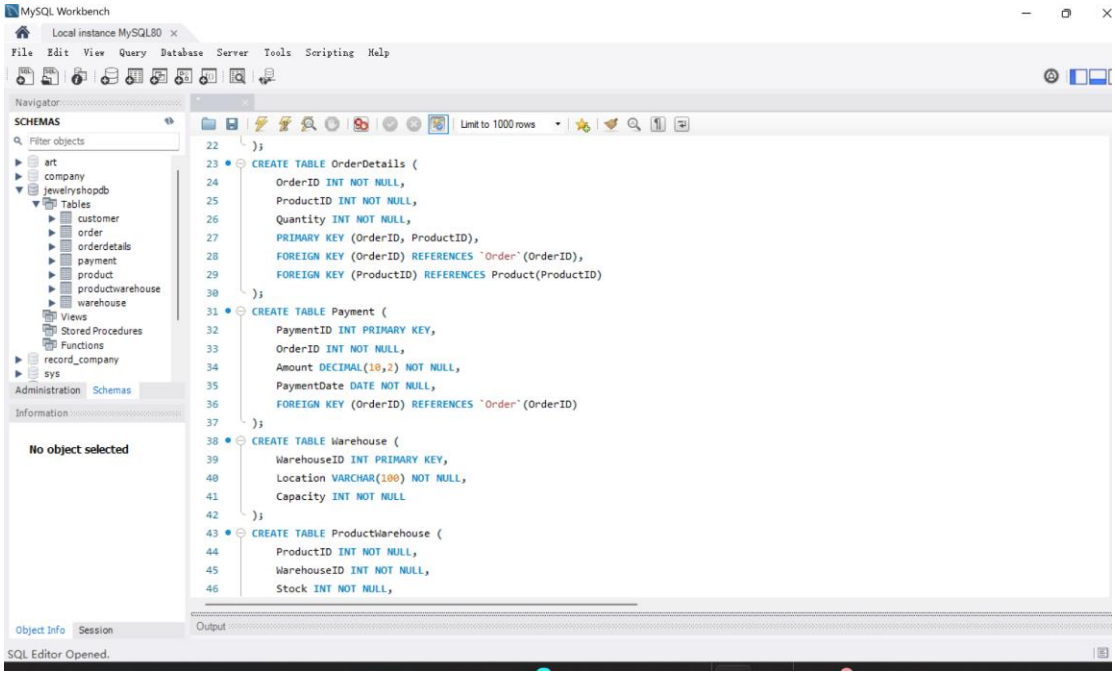
3.3 Relational Model

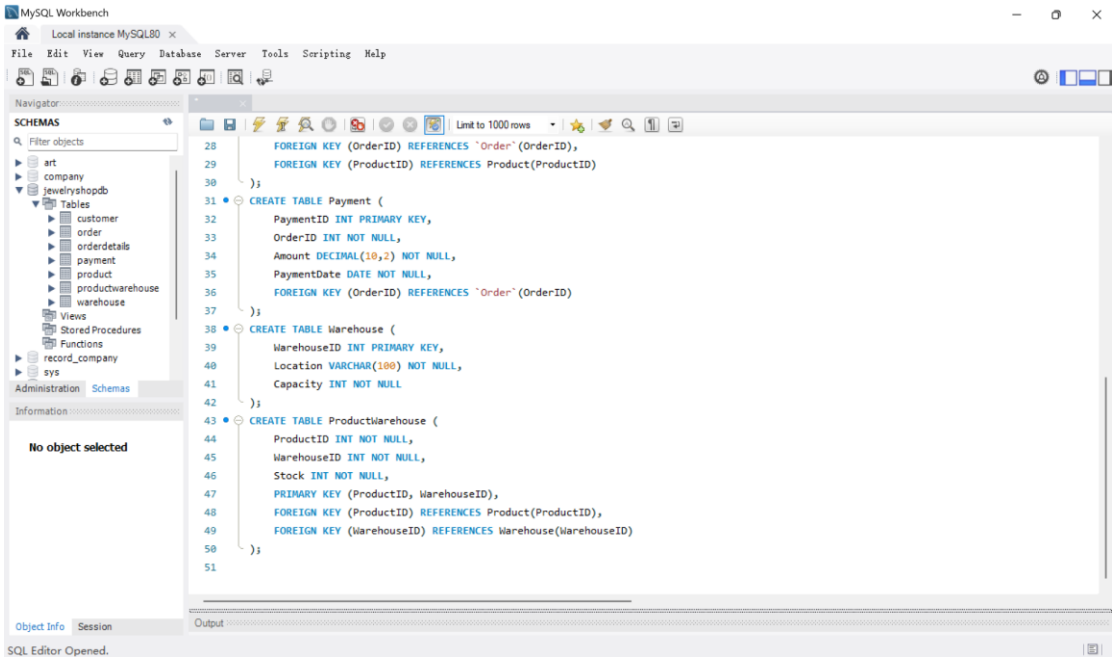


Part 2- Physical design

1. Create the corresponding database using DDL and Create all the necessary tables identified above using DDL

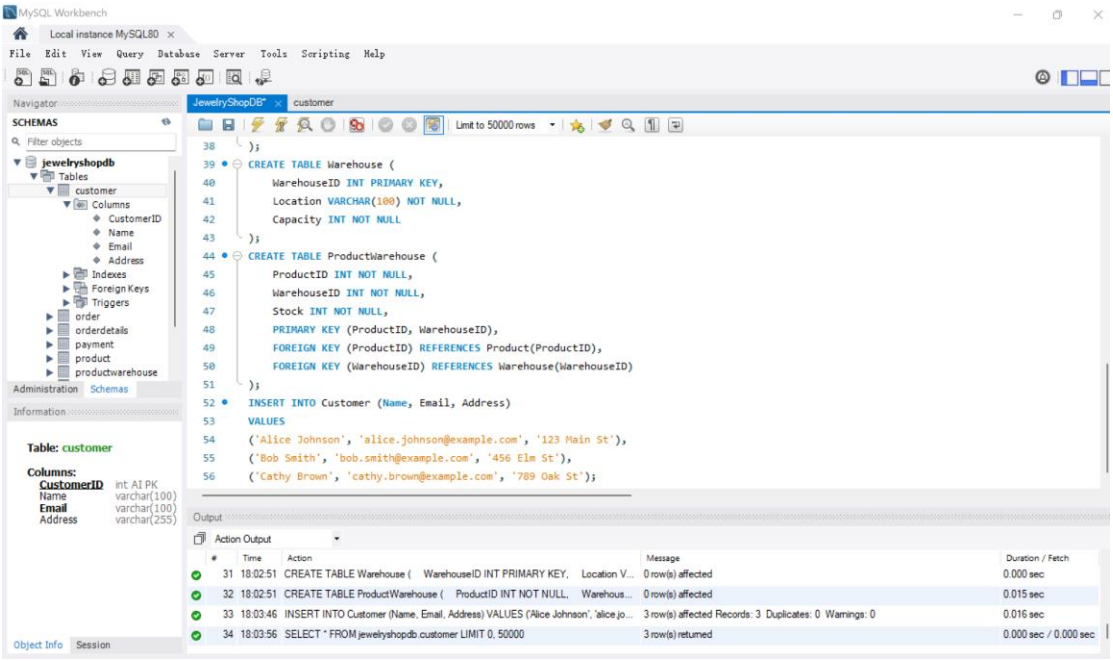


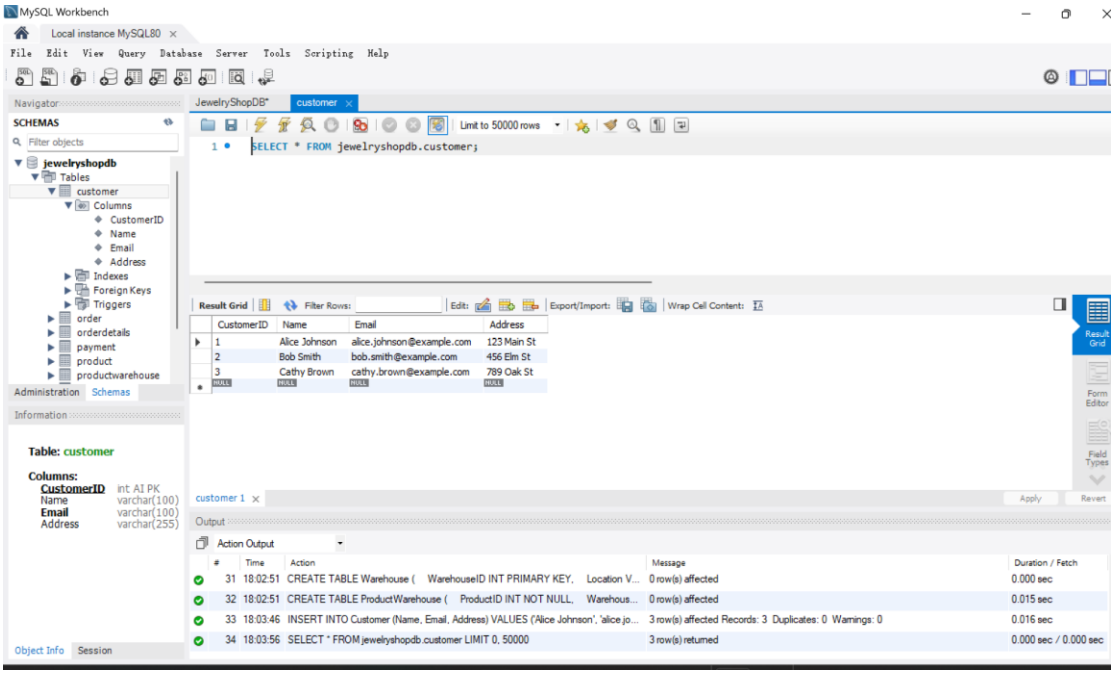

MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
art
company
jewelryshopdb
Tables
customer
order
orderdetails
payment
product
productwarehouse
warehouse
Views
Stored Procedures
Functions
record_company
sys
Administration Schemas
Information
No object selected
Object Info Session
Limit to 1000 rows
);
CREATE TABLE OrderDetails (
OrderID INT NOT NULL,
ProductID INT NOT NULL,
Quantity INT NOT NULL,
PRIMARY KEY (OrderID, ProductID),
FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
CREATE TABLE Payment (
PaymentID INT PRIMARY KEY,
OrderID INT NOT NULL,
Amount DECIMAL(10,2) NOT NULL,
PaymentDate DATE NOT NULL,
FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID)
);
CREATE TABLE Warehouse (
WarehouseID INT PRIMARY KEY,
Location VARCHAR(100) NOT NULL,
Capacity INT NOT NULL
);
CREATE TABLE ProductWarehouse (
ProductID INT NOT NULL,
WarehouseID INT NOT NULL,
Stock INT NOT NULL,
Output
SQL Editor Opened.


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
art
company
jewelryshopdb
Tables
customer
order
orderdetails
payment
product
productwarehouse
warehouse
Views
Stored Procedures
Functions
record_company
sys
Administration Schemas
Information
No object selected
Object Info Session
Limit to 1000 rows
FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
CREATE TABLE Payment (
PaymentID INT PRIMARY KEY,
OrderID INT NOT NULL,
Amount DECIMAL(10,2) NOT NULL,
PaymentDate DATE NOT NULL,
FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID)
);
CREATE TABLE Warehouse (
WarehouseID INT PRIMARY KEY,
Location VARCHAR(100) NOT NULL,
Capacity INT NOT NULL
);
CREATE TABLE ProductWarehouse (
ProductID INT NOT NULL,
WarehouseID INT NOT NULL,
Stock INT NOT NULL,
PRIMARY KEY (ProductID, WarehouseID),
FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
FOREIGN KEY (WarehouseID) REFERENCES Warehouse(WarehouseID)
);
Output
SQL Editor Opened.

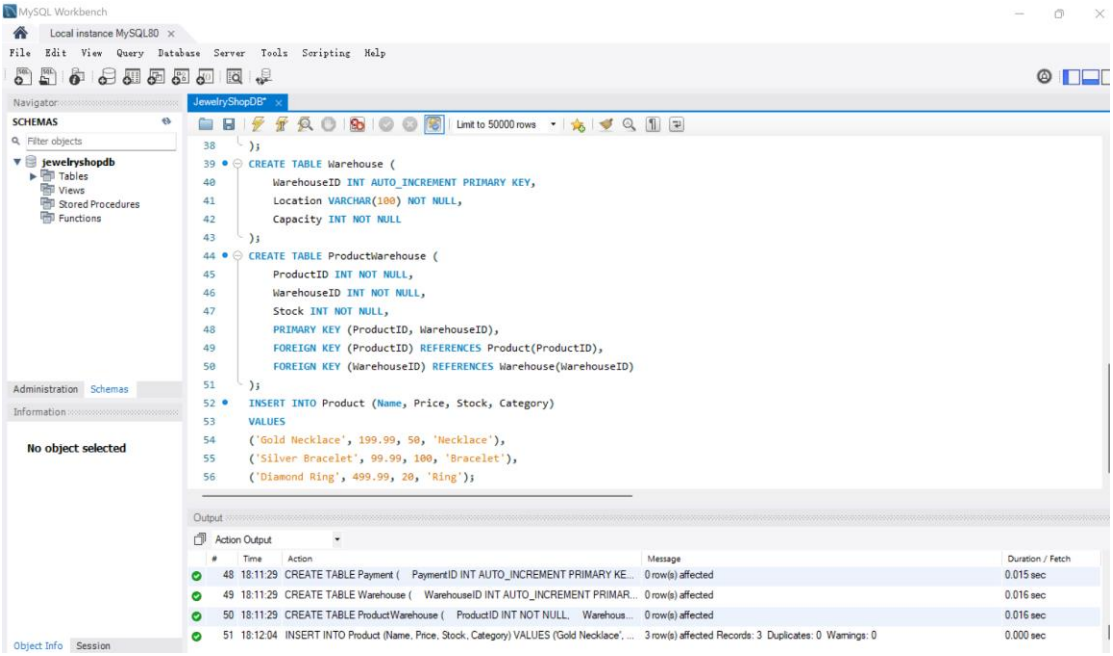
2. Populate Tables Using DML

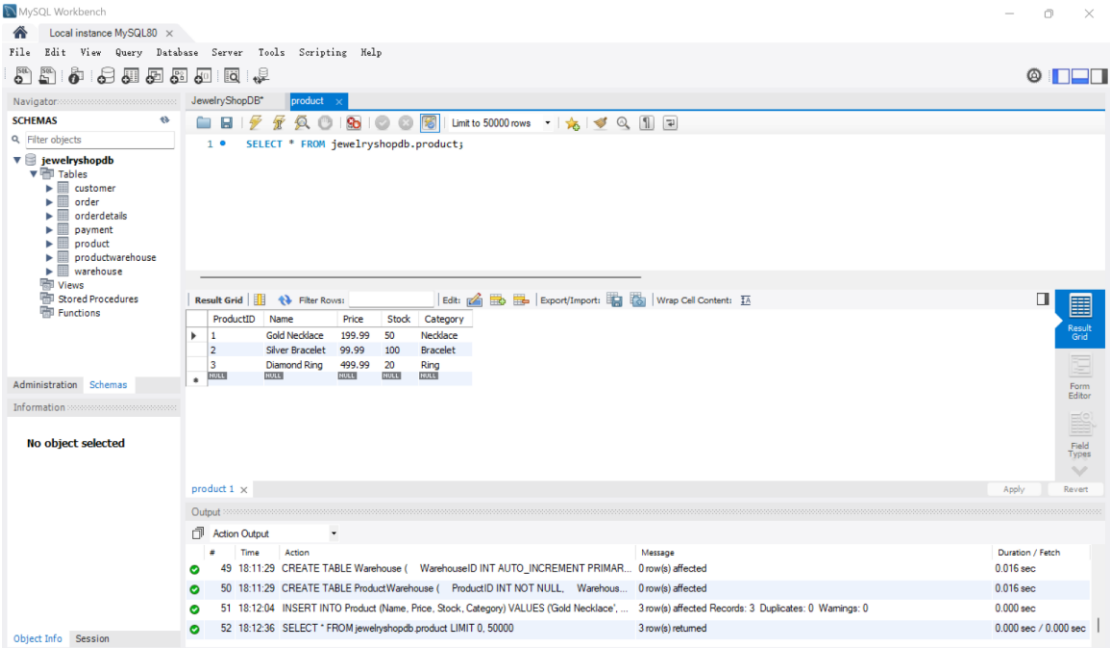
2.1 insert statements in customer table



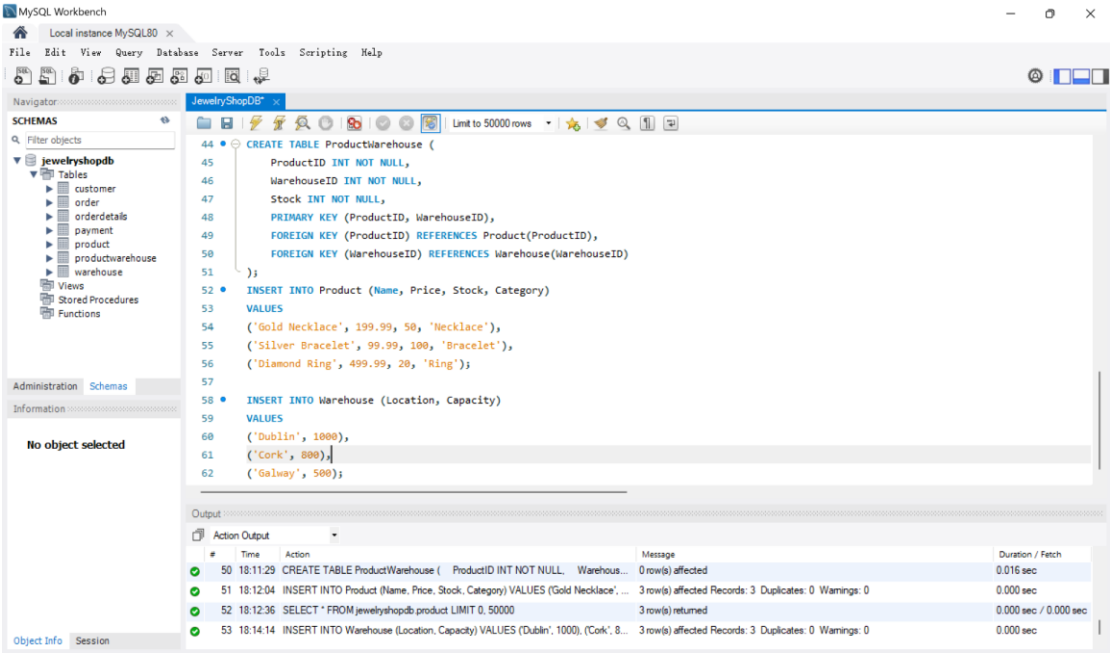


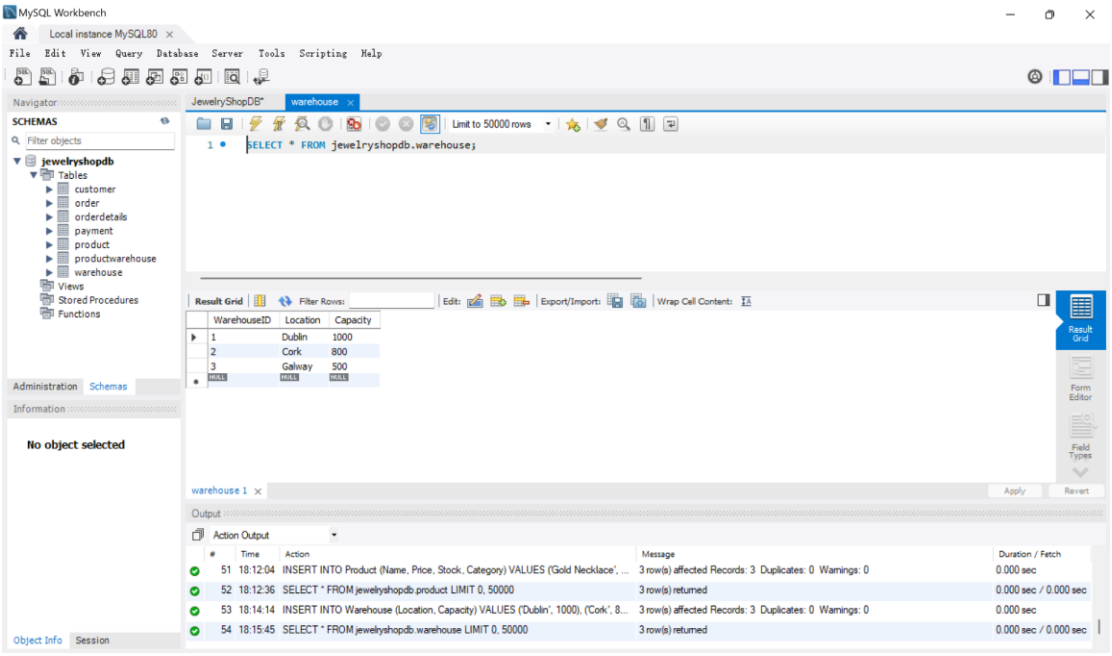
2.2 insert statement in product table





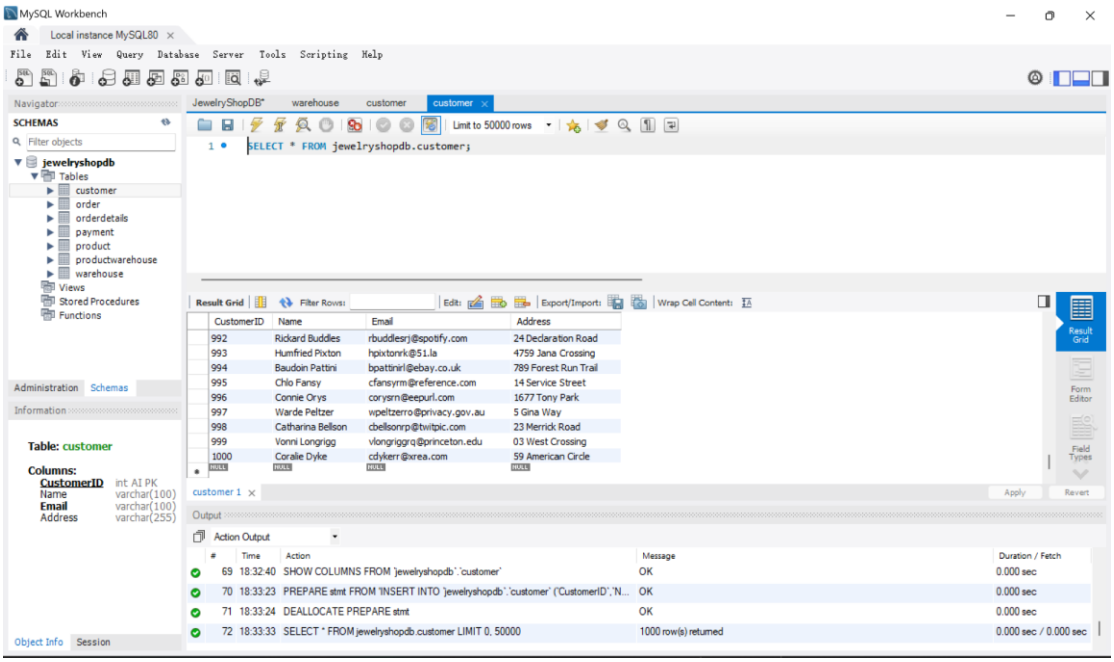
2.3 insert statements in warehouse table



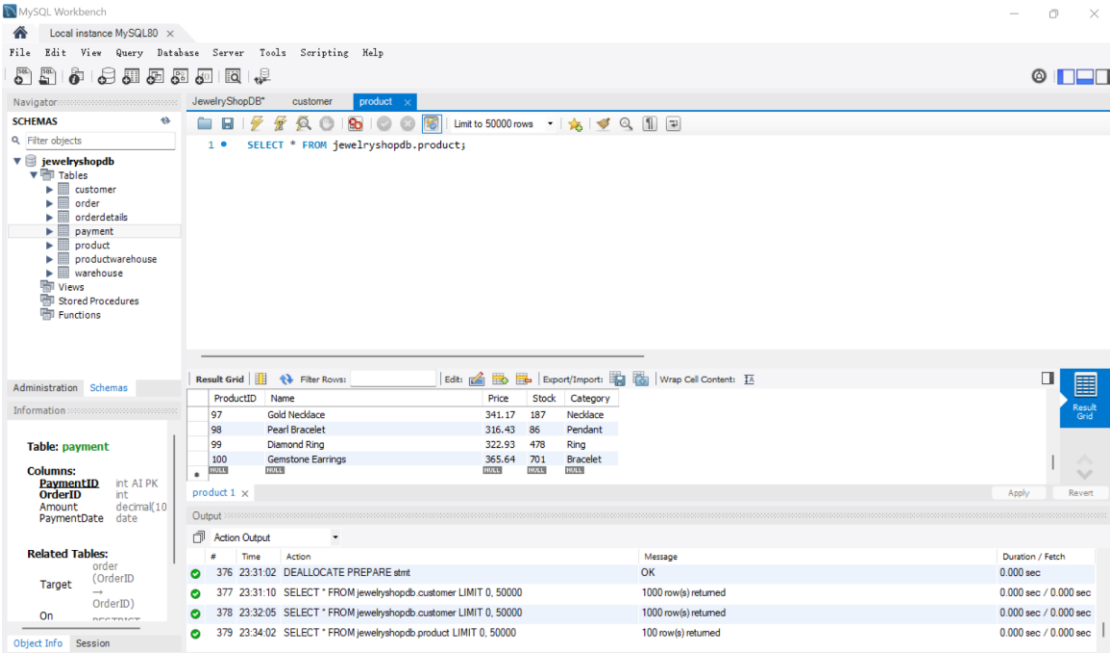


3. Populate database with a large data set representing a one-year transaction

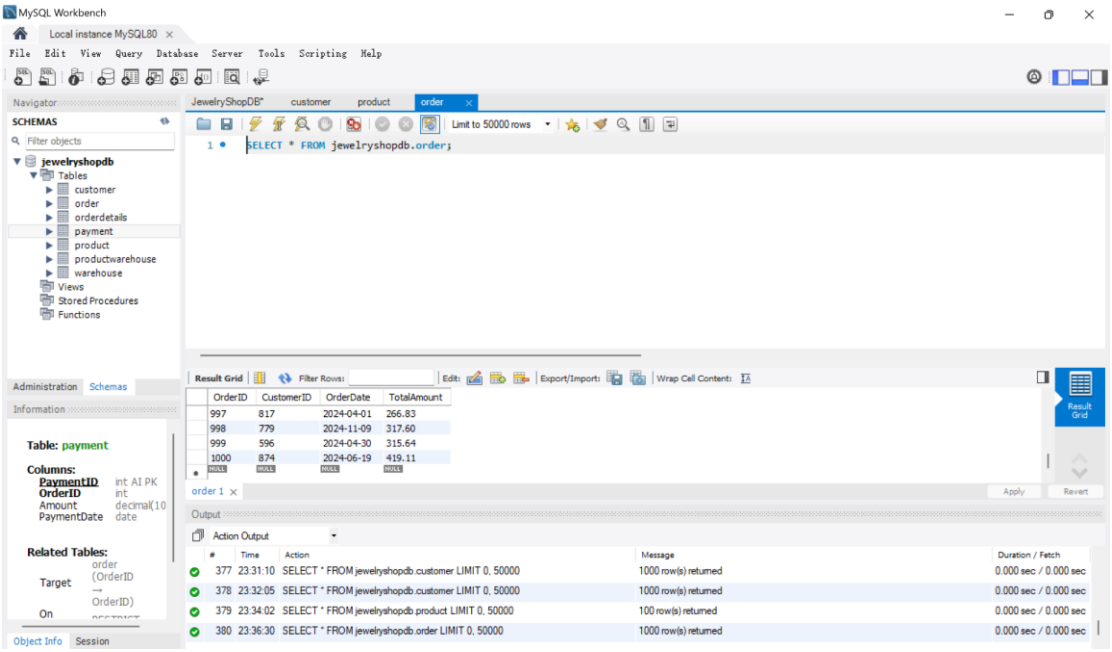
3.1 import customer data



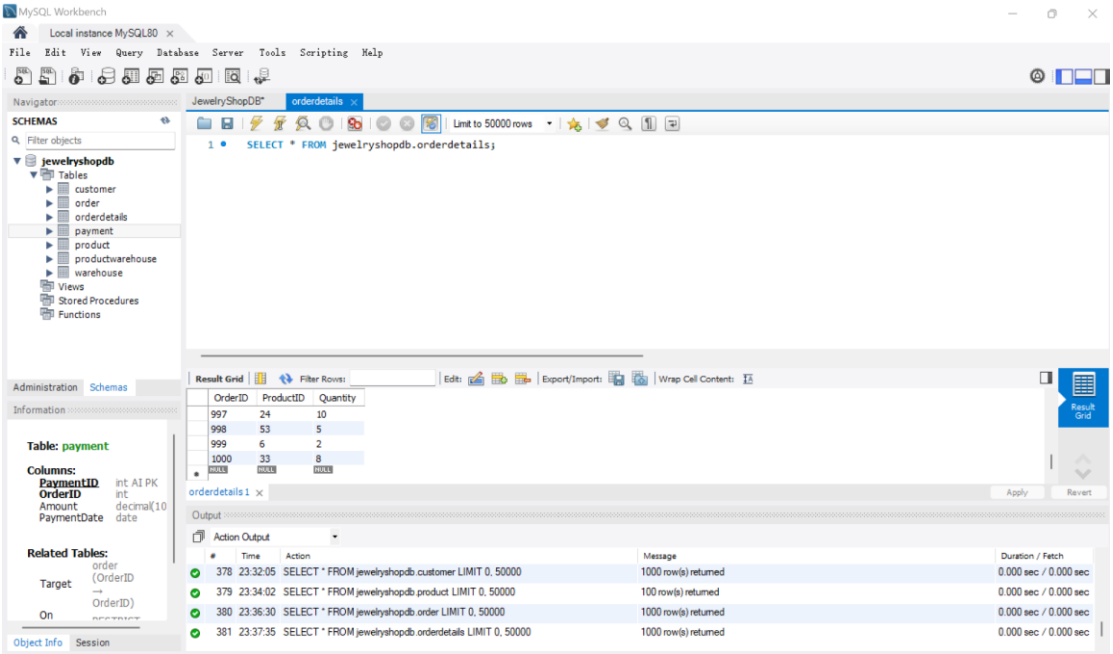
3.2 import product data



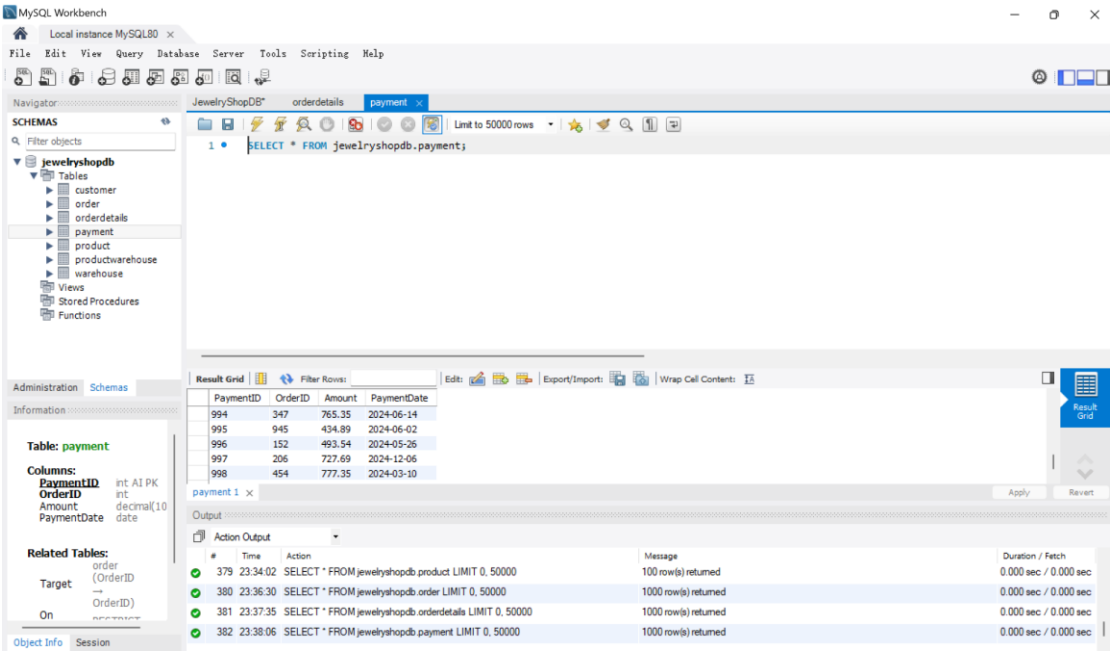
3.3 import order data



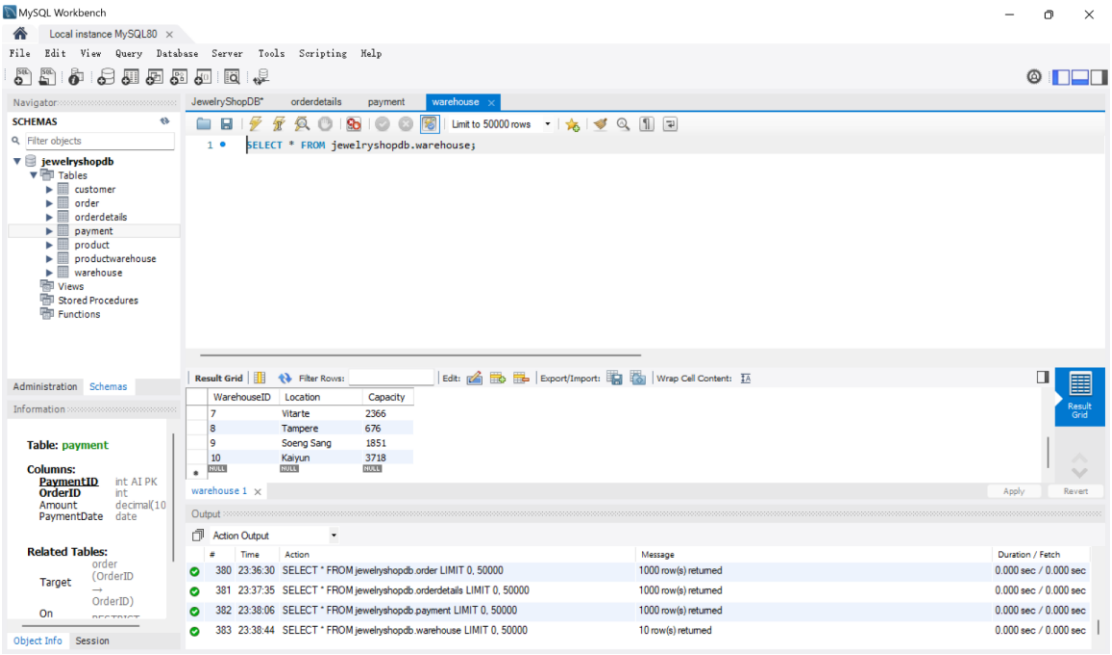
3.4 import orderdetails data



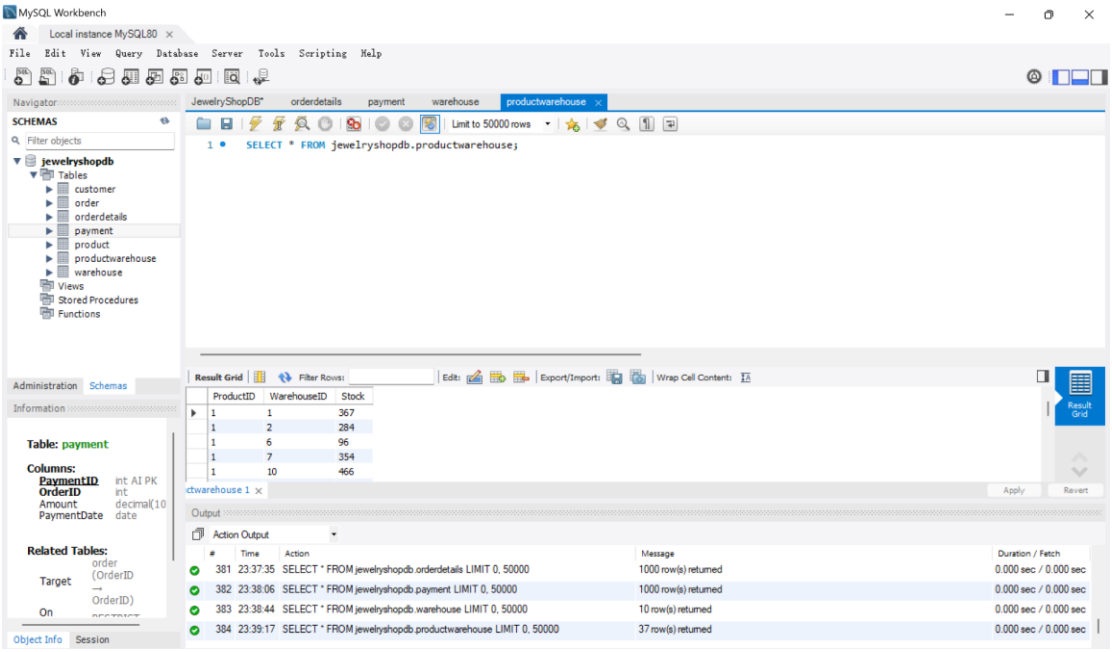
3.5 import payment data



3.6 import warehouse data



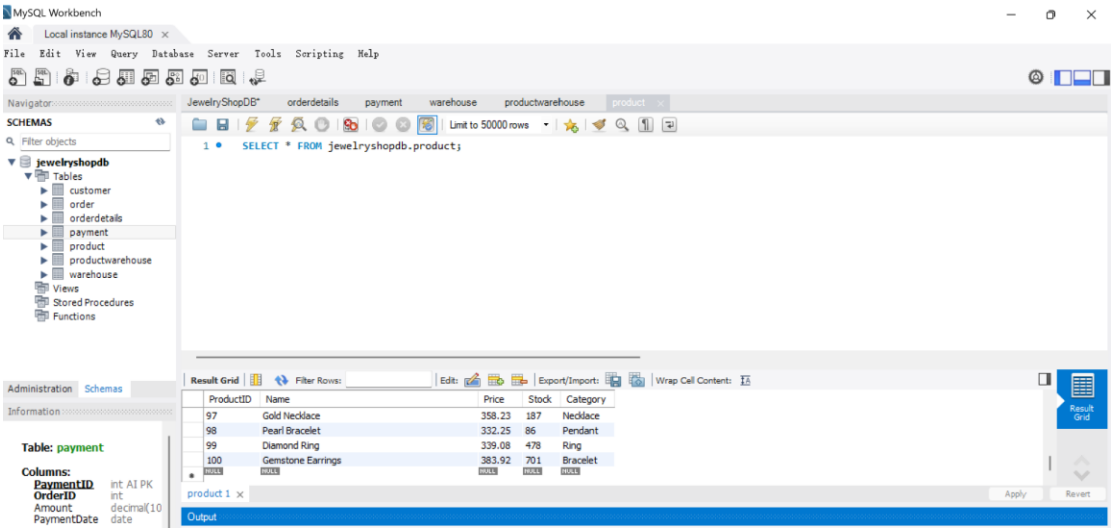
3.7 import productwarehouse data



Part 3- SQL statements

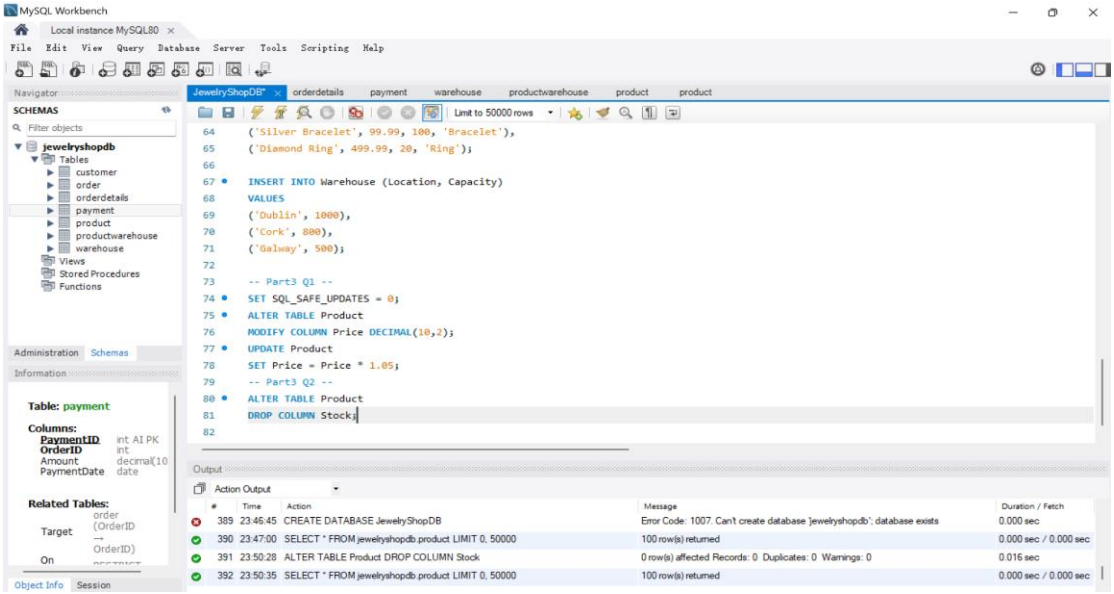
1. update entry

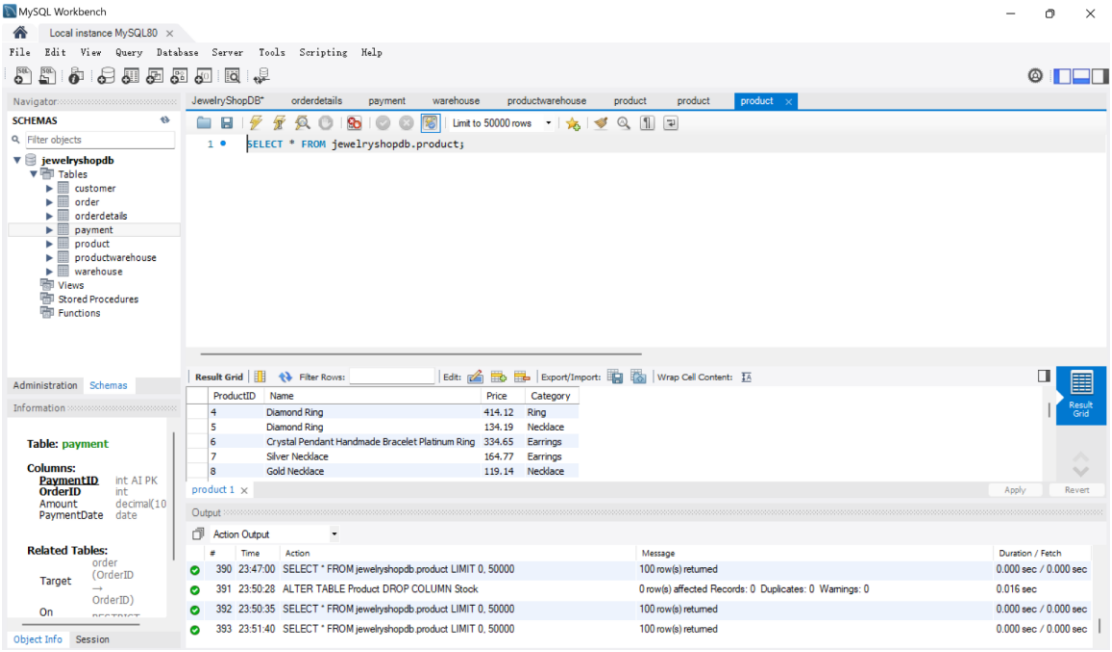
```
74      -- Part3 Q1 --
75  •   SET SQL_SAFE_UPDATES = 0;
76  •   ALTER TABLE Product
77      MODIFY COLUMN Price DECIMAL(10,2);
78  •   UPDATE Product
79      SET Price = Price * 1.05;
```



2. delete entry

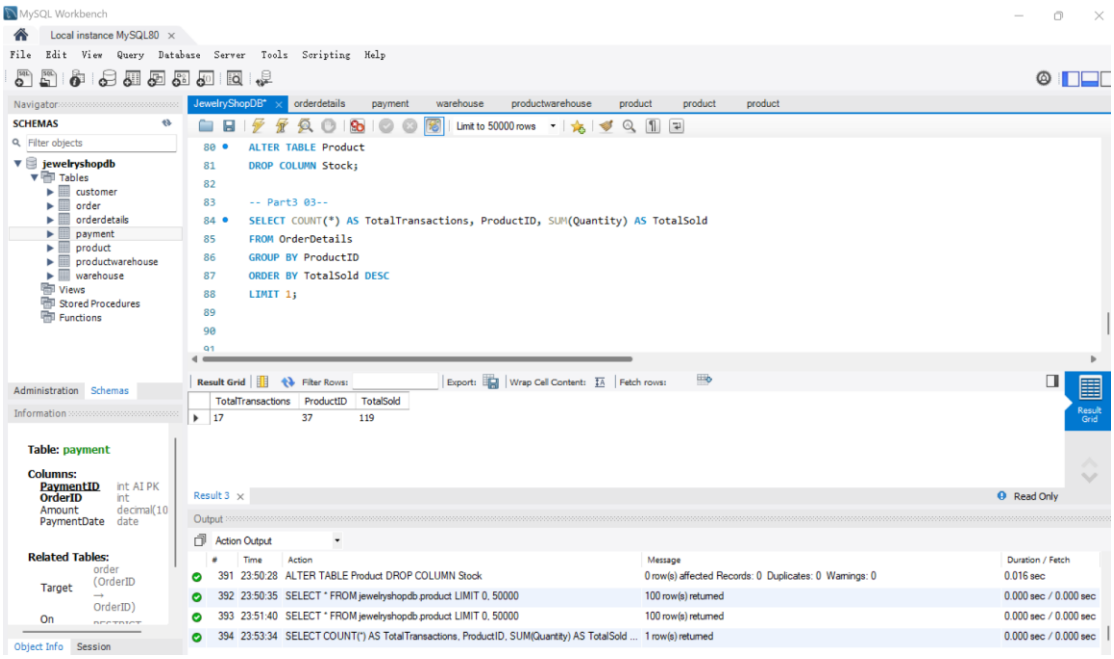
```
80      -- Part3 Q2 --
81 •    ALTER TABLE Product
82      DROP COLUMN Stock;
83
```



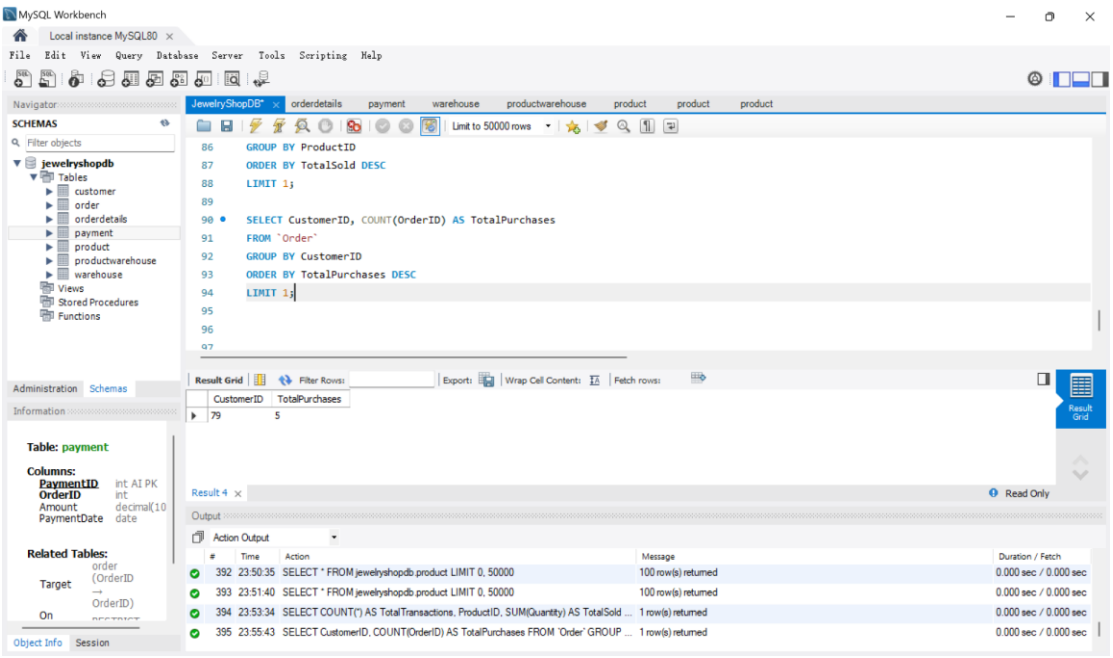


3. Show transactions with top sold/listed entry

```
84      -- Part3 03--
85 •    SELECT COUNT(*) AS TotalTransactions, ProductID, SUM(Quantity) AS TotalSold
86      FROM OrderDetails
87      GROUP BY ProductID
88      ORDER BY TotalSold DESC
89      LIMIT 1;
```

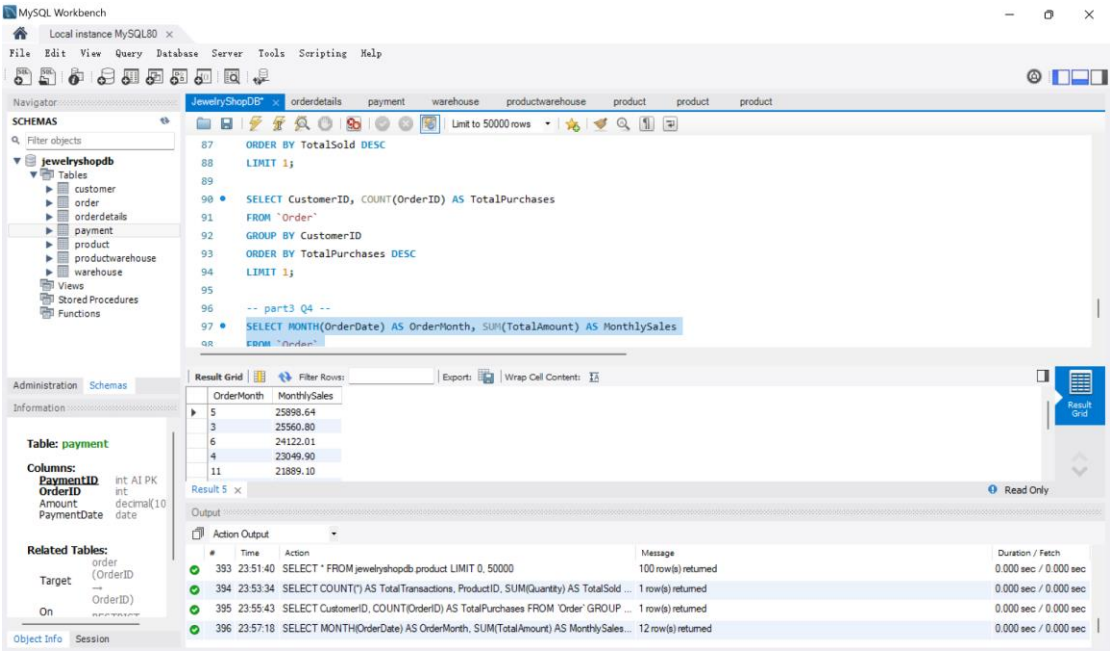


```sql
    SELECT CustomerID, COUNT(OrderID) AS TotalPurchases
    FROM `Order`
    GROUP BY CustomerID
    ORDER BY TotalPurchases DESC
    LIMIT 1;
;
```



4. Order by and Group by

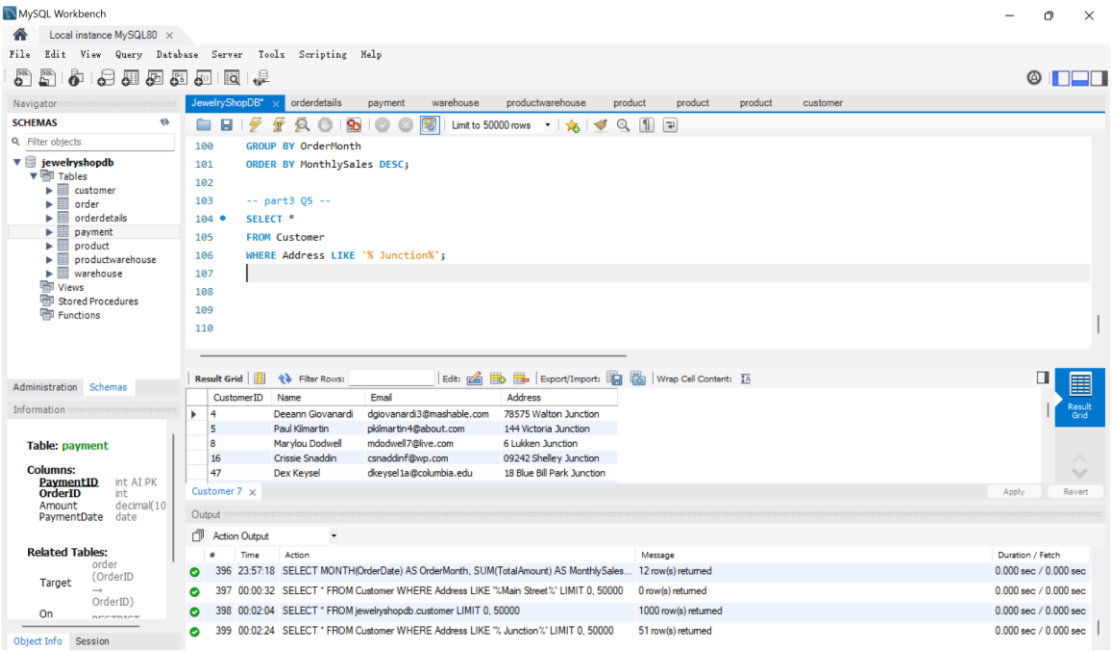
```sql
    -- part3 Q4 --
    SELECT MONTH(OrderDate) AS OrderMonth, SUM(TotalAmount) AS MonthlySales
    FROM `Order`
    GROUP BY OrderMonth
    ORDER BY MonthlySales DESC;
```



5. Pattern matching

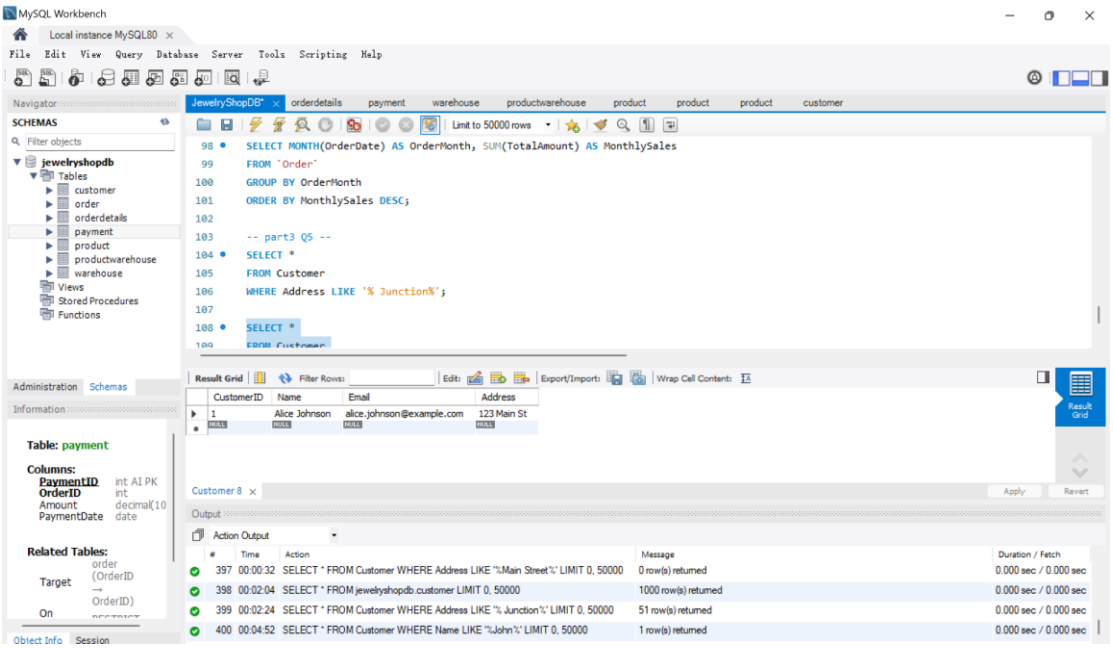
1 .

```
-- part3 Q5 --
SELECT *
FROM Customer
WHERE Address LIKE '% Junction%';
```



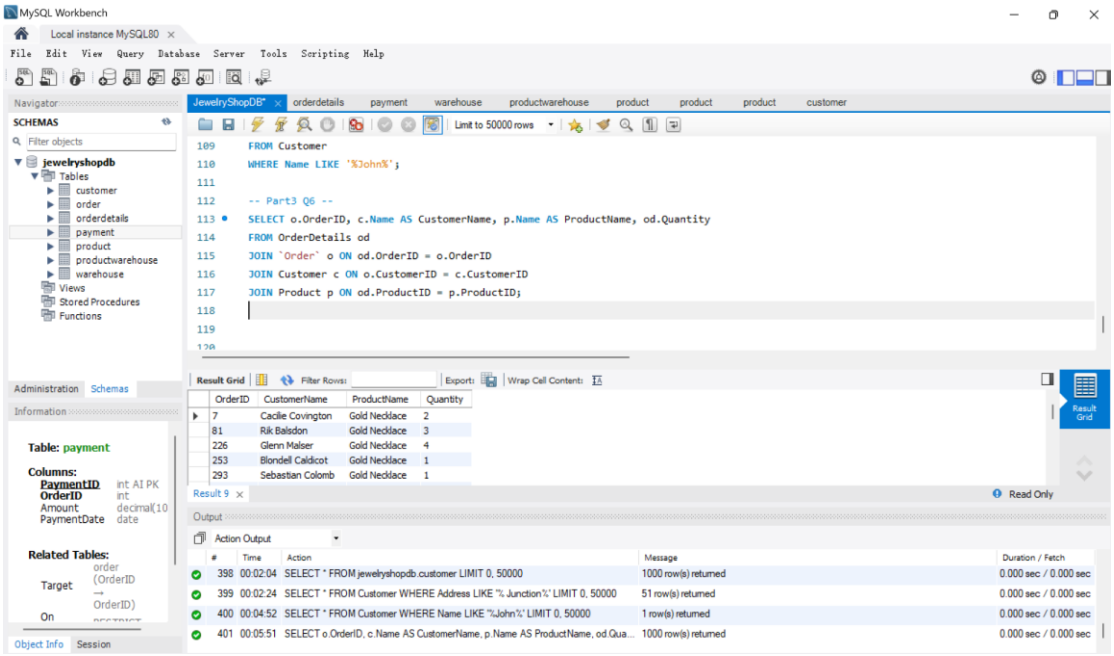
2.

```
108 ●    SELECT *
109      FROM Customer
110      WHERE Name LIKE '%John%';
```



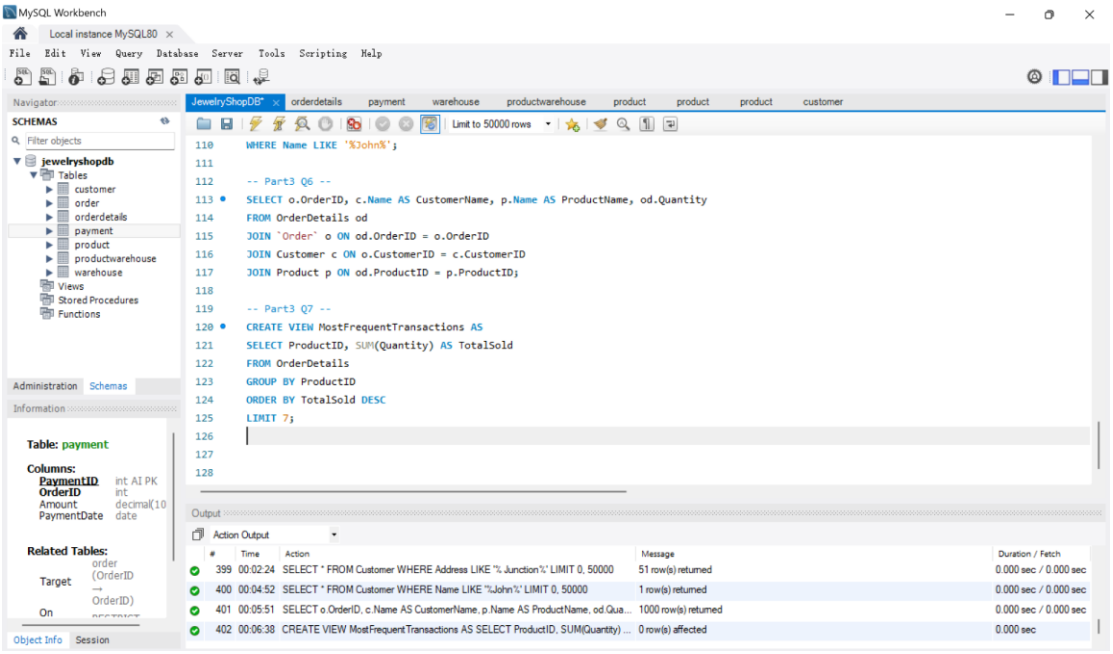
6. Information from multiple tables using Join

```
112     -- Part3 Q6 --
113 •   SELECT o.OrderID, c.Name AS CustomerName, p.Name AS ProductName, od.Quantity
114     FROM OrderDetails od
115     JOIN `Order` o ON od.OrderID = o.OrderID
116     JOIN Customer c ON o.CustomerID = c.CustomerID
117     JOIN Product p ON od.ProductID = p.ProductID;
```



7. View of most recent transactions

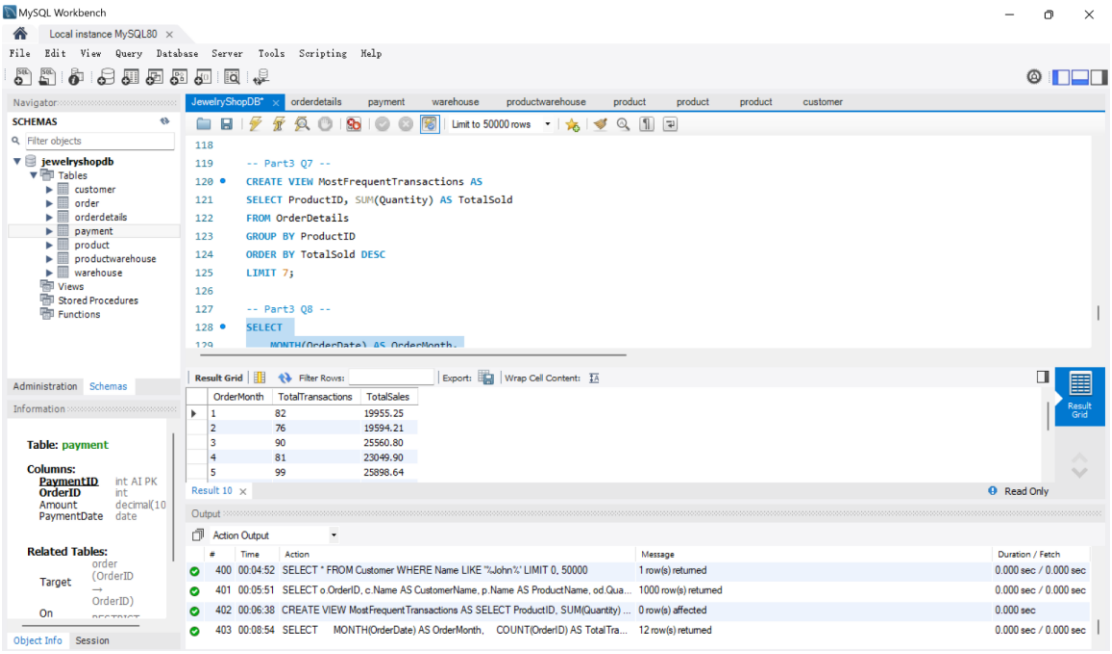
```
119     -- Part3 Q7 --
120 •   CREATE VIEW MostFrequentTransactions AS
121     SELECT ProductID, SUM(Quantity) AS TotalSold
122     FROM OrderDetails
123     GROUP BY ProductID
124     ORDER BY TotalSold DESC
125     LIMIT 7;
```



8. Summary of annual transactions

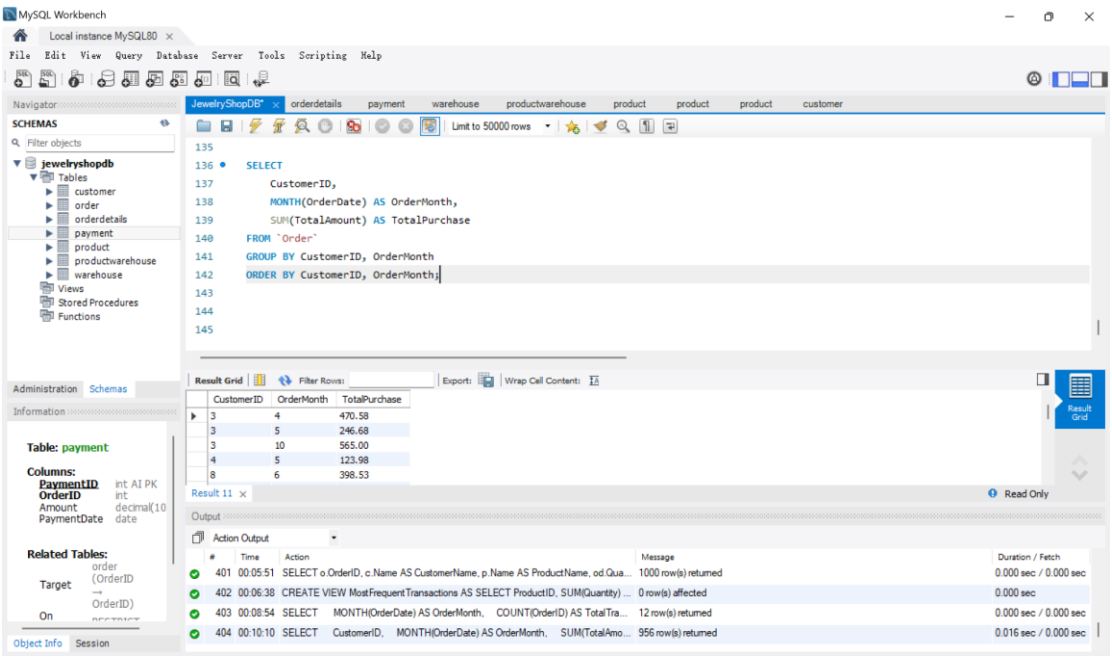
8.1 Total Number of Transactions and Details per Month

```
128  -- Part3 Q8 Shows the total number of transactions with corresponding details every month --
129  SELECT
130      MONTH(OrderDate) AS OrderMonth,
131      COUNT(OrderID) AS TotalTransactions,
132      SUM(TotalAmount) AS TotalSales
133  FROM `Order`
134  GROUP BY OrderMonth
135  ORDER BY OrderMonth;
```



8.2. Customer Purchase Value Per Month

```
137     -- Part3 Q8 Shows customer purchase value per month --
138 ●   SELECT
139         CustomerID,
140         MONTH(OrderDate) AS OrderMonth,
141         SUM(TotalAmount) AS TotalPurchase
142     FROM `Order`
143     GROUP BY CustomerID, OrderMonth
144     ORDER BY CustomerID, OrderMonth;
```



8.3 Product Sales Quantity Per Month

```sql
-- Part3 Q8 Shows name of product and number sold each month --
SELECT
    MONTH(o.OrderDate) AS OrderMonth,
    p.Name AS ProductName,
    SUM(od.Quantity) AS TotalSold
FROM OrderDetails od
JOIN `Order` o ON od.OrderID = o.OrderID
JOIN Product p ON od.ProductID = p.ProductID
GROUP BY OrderMonth, ProductName
ORDER BY OrderMonth, TotalSold DESC;
```