E-Commerce Website

Database Management System
Final Report

Course: CPS 510 (Database Systems I)
Section: 6
Group Members:
Rajorshi Sarker, Student Number: 501120195
Karam Al-Chouli, Student Number: 501117243
Choyon Uddin, Student Number: 501031767

Application Description

This E-Commerce Database Management System (DBMS) ensures the efficient and secure data management in an online retail platform. It supports various operations, allowing seamless interaction between users, products, orders, payments, etc.

The DBMS provides a range of functionalities. One important feature is role based user control, which distinguishes between customers and administrators each with their set of permissions and privileges. Administrators have the ability to manage products by adding, editing and deleting them. They can also categorize products based on style, available sizes and colours to enhance navigation. The system also keeps track of product stock levels. If a product is out of stock, customers cannot purchase it until it restocks. It is the responsibility of the administrator to update the inventory once the product is back in stock. Additionally, this application includes a search function that helps customers easily find and browse products by matching tags with their search queries. Customers can also use filters to refine their search based on criteria such as merchandise type, style, size or colour. Furthermore the system handles product reviews and ratings to provide insights into product quality for customers.

Order management is a core function of the DBMS, as it enables users to place orders or cancel existing ones while keeping track of order statuses. Moreover, this system supports payment methods including credit cards and PayPal along with integration with payment gateways, for smooth transaction processing. When it comes to shipping and logistics, the DBMS system calculates expenses, creates shipment labels and tracks them by integrating with shipping companies. In addition to that, customers have the option to update their account information and check their order history.

In the background, this DBMS ensures data integrity by maintaining relationships between entities, such as connecting orders to products via order items and associating payments with orders. Thus, it supports many essential functions and relationships to provide a smooth and efficient online shopping experience.

Potential Functions

Functions	Description
User	Allows users to register, log in or out, and manage their profiles. It also includes role-based access control to differentiate between customers, administrators, and vendors.
Product	Enables the addition, editing, and deletion of products in the system. It allows categorization of products, tracks product numbers, prices, and stock levels, and supports product reviews.
ShoppingCart	Lets users add or remove items from their carts, update quantities, calculate the total cost of items in the cart, and clear the cart when

	needed.
Payment	Involves accepting various payment methods, such as credit cards, and integrating with payment gateways to securely handle payments for orders.
Order	Allows users to create new orders, view order history, track the status of their orders, and cancel orders if necessary.
Shipping	Calculates shipping costs, generates shipping labels, tracks shipments, and integrates with shipping carriers to manage the delivery of products to customers.
Search	Enables users to search for products using keywords or filters.

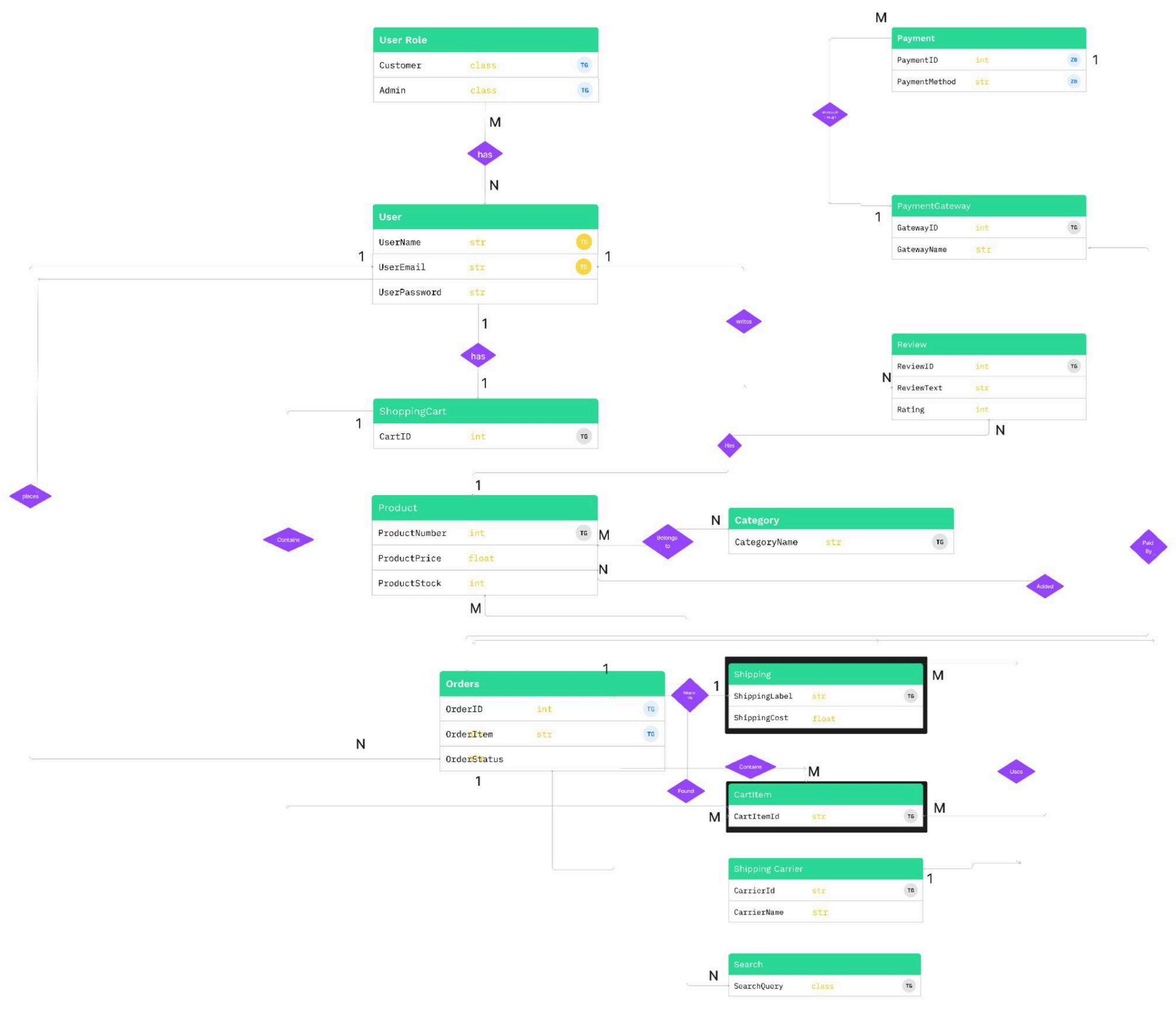
Entities and their Relationships

Entities	Relationships
User ● Username ● UserEmail ● UserPassword	User can have one or more Roles (UserRole)
UserRole ● Customer ● Admin	Each UserRole is associated with one or more Users.
Product ● ProductNumber ● ProductPrice ● ProductStock	Products belong to one or more Categories.
Category ● CategoryName	Categories can contain multiple Products.
Order ● OrderID ● OrderDate ● OrderStatus	Each Order can have multiple OrderItems. Each Order is associated with one Payment. Each Order can have one Shipping record
Payment ● PaymentID ● PaymentMethod	Each PaymentID is associated with one Order. Payments are processed through a PaymentGateway.
PaymentGateway ● GatewayID ● GatewayName	PaymentGateways can be associated with multiple Payments.
ShoppingCart	Each User can have one active ShoppingCart.

● CartID	Each ShoppingCart can have multiple CartItems.
CartItem ● CartItemID	Each CartItem is associated with one Product. Each CartItem belongs to one ShoppingCart.
Shipping ● ShippingLabel ● ShippingCost	Each Order can have one Shipping record. Shipping is associated with a ShippingCarrier.
ShippingCarrier ● CarrierID ● CarrierName	ShippingCarriers can be associated with multiple Shipping records.
Search ● SearchQuery	Each SearchQuery can be associated with multiple Products.
Review ● ReviewID ● ReviewText ● Rating	Each Review is associated with one Product. Each Review includes a Rating.

ER Model

Group Members: Rajorshi Sarker, Student Number: 501120195 Karam Al-Chouli, Student Number: 501117243 Choyon Uddin, Student Number: 501031767



Scheme Design

Part 1: Creating Tables

```
SQL> CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(255) NOT NULL,
    UserEmail VARCHAR(255) NOT NULL,
    UserPassword VARCHAR(255) NOT NULL
);

Table created.
```

```
SQL> CREATE TABLE UserRoles (
   RoleID INT PRIMARY KEY,
   RoleName VARCHAR(50) NOT NULL
);

Table created.

SQL> CREATE TABLE User_UserRoles (
   UserID INT,
   RoleID INT,
   PRIMARY KEY (UserID, RoleID),
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (RoleID) REFERENCES UserRoles(RoleID)
);

Table created.

SQL> CREATE TABLE Categories (
   CategoryID INT PRIMARY KEY,
   CategoryName VARCHAR(100) NOT NULL
);

Table created.

SQL> CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductNumber VARCHAR(20) NOT NULL
   ProductName VARCHAR(255),
   ProductPrice DECIMAL(10, 2) NOT NULL,
   ProductStock INT NOT NULL,
   CategoryID INT,
   FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

Table created.

SQL> CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   OrderDate DATE NOT NULL,
   OrderStatus VARCHAR(50) NOT NULL,
   UserID INT,
   FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

```
Table created.

SQL> CREATE TABLE OrderItems (
   OrderItemID INT PRIMARY KEY,
   OrderID INT,
   ProductID INT,
   OrderQuantity INT NOT NULL,
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

Table created.

SQL> CREATE TABLE Payments (
   PaymentID INT PRIMARY KEY,
   PaymentMethod VARCHAR(100) NOT NULL,
   OrderID INT,
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);

Table created.

SQL> CREATE TABLE PaymentGateways (
   GatewayID INT PRIMARY KEY,
   GatewayName VARCHAR(100) NOT NULL
);

Table created.

SQL> CREATE TABLE Payment_PaymentGateways (
   PaymentID INT,
   GatewayID INT,
   PRIMARY KEY (PaymentID, GatewayID),
   FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
   FOREIGN KEY (GatewayID) REFERENCES PaymentGateways(GatewayID)
);

Table created.

SQL> CREATE TABLE ShoppingCarts (
   CartID INT PRIMARY KEY,
   UserID INT,
   FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

```
Table created.

SQL> CREATE TABLE CartItems (
   CartItemID INT PRIMARY KEY,
   CartID INT,
   ProductID INT,
   CartQuantity INT NOT NULL,
   FOREIGN KEY (CartID) REFERENCES ShoppingCarts(CartID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

Table created.

SQL> CREATE TABLE ShippingCarriers (
   CarrierID INT PRIMARY KEY,
   CarrierName VARCHAR(100) NOT NULL
);

Table created.

SQL> CREATE TABLE Shipping (
   ShippingLabel VARCHAR(50) PRIMARY KEY,
   ShippingCost DECIMAL(10, 2) NOT NULL,
   OrderID INT,
   ShippingCarrierID INT,
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
   FOREIGN KEY (ShippingCarrierID) REFERENCES ShippingCarriers(CarrierID)
);

Table created.

SQL> CREATE TABLE SearchQueries (
   SearchQueryID INT PRIMARY KEY,
   QueryText VARCHAR(255) NOT NULL
);

Table created.

SQL> CREATE TABLE ProductReviews (
   ReviewID INT PRIMARY KEY,
   ReviewText VARCHAR2(4000),
   Rating INT,
   ProductID INT,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

Table created.
```

Part 2: Populating Tables with Data

```
SQL> INSERT INTO Users (UserID, Username, UserEmail, UserPassword)
VALUES (1, 'john_doe', 'john@example.com', 'password123');

1 row created.

SQL> INSERT INTO Users (UserID, Username, UserEmail, UserPassword)
VALUES ('2', 'admin_user', 'admin@example.com', 'adminpass');

1 row created.

SQL> INSERT INTO UserRoles (RoleID, RoleName)
VALUES ('1', 'Customer');

1 row created.

SQL> INSERT INTO UserRoles (RoleID, RoleName)
VALUES ('2', 'Admin');

1 row created.

SQL> INSERT INTO User_UserRoles (UserID, RoleID)
VALUES ('1', '1');

1 row created.

SQL> INSERT INTO User_UserRoles (UserID, RoleID)
VALUES (‘2’, ‘2’);

1 row created.

SQL> INSERT INTO Categories (CategoryID, CategoryName)
VALUES ('1', 'Tops');

1 row created.

SQL> INSERT INTO Categories (CategoryID, CategoryName)
VALUES (2, 'Bottoms');
```

```
1 row created.

SQL> INSERT INTO Products (ProductID, ProductNumber, ProductPrice, ProductStock,
CategoryID)
VALUES ('1', 'P12345', '49.99', '100', '1');

1 row created.

SQL> INSERT INTO Products (ProductID, ProductNumber, ProductPrice, ProductStock,
CategoryID)
VALUES ('2', 'C56789', '29.99', '100', '2');

1 row created.

SQL> INSERT INTO Orders (OrderID, OrderDate, OrderStatus, UserID)
VALUES ('1', '2023-09-15', 'Processing', '1');

1 row created.

SQL> INSERT INTO Orders (OrderID, OrderDate, OrderStatus, UserID)
VALUES ('2','2023-09-13', 'Shipped', '2');

1 row created.

SQL> INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, OrderQuantity)
VALUES ('1', '1', '1', '2');

1 row created.

SQL> INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, OrderQuantity)
VALUES ('2', '2', '2', '1');

1 row created.

SQL> INSERT INTO Payments (PaymentID, PaymentMethod, OrderID)
VALUES ('1', 'Credit Card', '1');

1 row created.

SQL> INSERT INTO Payments (PaymentID, PaymentMethod, OrderID)
VALUES ('2', 'PayPal', '2');

1 row created.
```

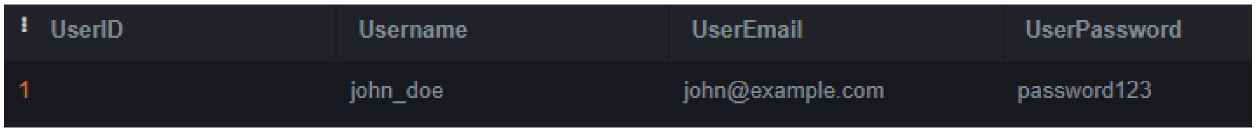
```
SQL> INSERT INTO PaymentGateways (GatewayID, GatewayName)
VALUES ('1', 'Stripe');

1 row created.

SQL> INSERT INTO PaymentGateways (GatewayID, GatewayName)
VALUES ('2', 'PayPal');

1 row created.

SQL> INSERT INTO Payment_PaymentGateways (PaymentID, GatewayID)
VALUES ('1', '1');

1 row created.

SQL> INSERT INTO Payment_PaymentGateways (PaymentID, GatewayID)
VALUES ('2', '2');

1 row created.

SQL> INSERT INTO ShoppingCarts (CartID, UserID)
VALUES ('1', '1');

1 row created.

SQL> INSERT INTO ShoppingCarts (CartID, UserID)
VALUES ('2', '2');

1 row created.

SQL> INSERT INTO CartItems (CartItemID, CartID, ProductID, CartQuantity)
VALUES ('1', '1', '1', ‘2’);

1 row created.

SQL> INSERT INTO CartItems (CartItemID, CartID, ProductID, CartQuantity)
VALUES ('2', '2', '2', '1');

1 row created.

SQL> INSERT INTO ShippingCarriers (CarrierID, CarrierName)
VALUES ('1', 'Canada Post');
```

```
1 row created.

SQL> INSERT INTO ShippingCarriers (CarrierID, CarrierName)
VALUES ('2', 'Fedex');

1 row created.

SQL> INSERT INTO ShippingCarriers (CarrierID, CarrierName)
VALUES ('3', 'UPS');

1 row created.

SQL> INSERT INTO Shipping (ShippingLabel, ShippingCost, OrderID, ShippingCarrierID)
VALUES ('UPS-123456', '10.99', '1', '3');

1 row created.

SQL> INSERT INTO Shipping (ShippingLabel, ShippingCost, OrderID, ShippingCarrierID)
VALUES ('Canada Post-789012', '7.99', '2', '1');

1 row created.

SQL> INSERT INTO SearchQueries (SearchQueryID, QueryText)
VALUES ('1', 'Satin Top');

1 row created.

SQL> INSERT INTO SearchQueries (SearchQueryID, QueryText)
VALUES ('2', 'Denim Skirt');

1 row created.

SQL> INSERT INTO ProductReviews (ReviewID, ReviewText, Rating, ProductID)
VALUES ('1', 'Great product!', '5', '1');

1 row created.

SQL> INSERT INTO ProductReviews (ReviewID, ReviewText, Rating, ProductID)
VALUES ('2', 'Excellent quality for the price', '4', '2');

1 row created.

SQL> COMMIT;
```

Commit complete.

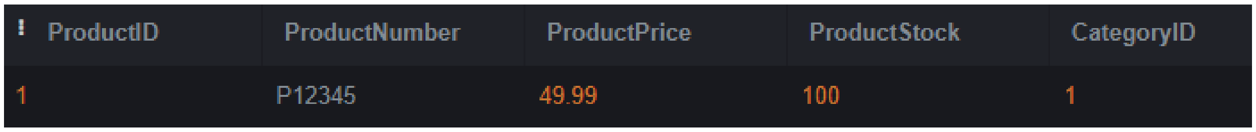
Queries

1. Retrieve User Information by Username:
   SELECT * FROM Users WHERE Username = 'john_doe';



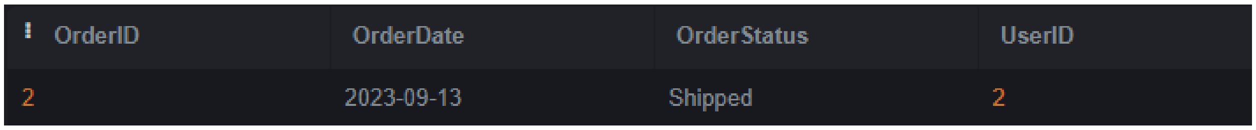
UserID	Username	UserEmail	UserPassword
1	john_doe	john@example.com	password123

2. List Products in a Specific Category:
   SELECT * FROM Products WHERE CategoryID = 1;



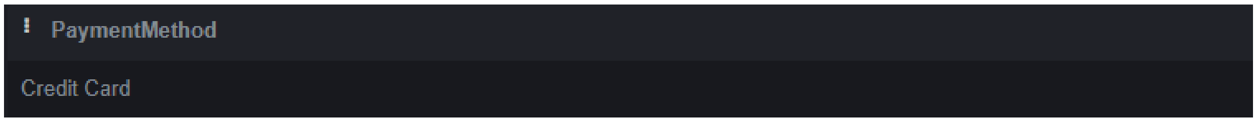
ProductID	ProductNumber	ProductPrice	ProductStock	CategoryID
1	P12345	49.99	100	1

3. Find Orders Shipped on a Specific Date:
   SELECT * FROM Orders WHERE OrderDate = '2023-09-13';



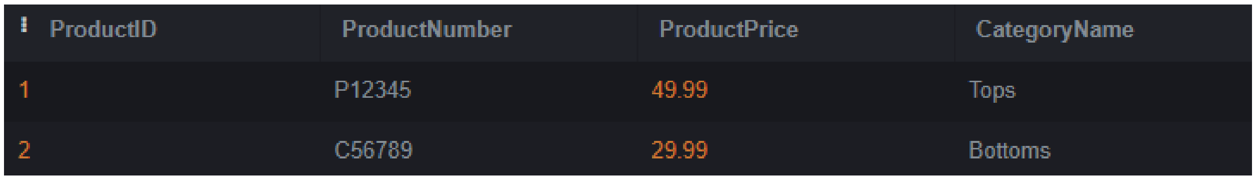
OrderID	OrderDate	OrderStatus	UserID
2	2023-09-13	Shipped	2

4. Retrieve All Payment Methods for an Order:
   SELECT PaymentMethod FROM Payments WHERE OrderID = 1;



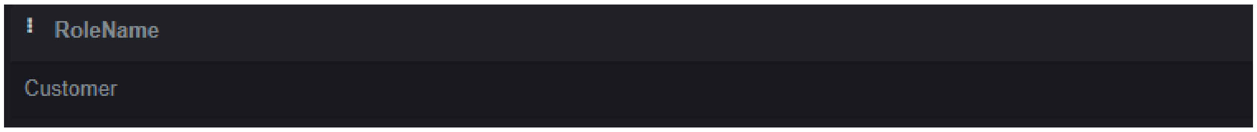
PaymentMethod
Credit Card

5. List Products and Their Categories:
   SELECT Products.ProductID, Products.ProductNumber, Products.ProductPrice, Categories.CategoryName
   FROM Products
   INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;



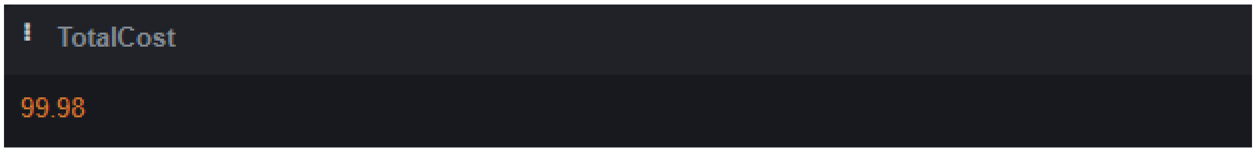
ProductID	ProductNumber	ProductPrice	CategoryName
1	P12345	49.99	Tops
2	C56789	29.99	Bottoms

6. Find All User Roles for a Specific User:
   SELECT UserRoles.RoleName
   FROM UserRoles
   INNER JOIN User_UserRoles ON UserRoles.RoleID = User_UserRoles.RoleID
   WHERE User_UserRoles.UserID = 1;



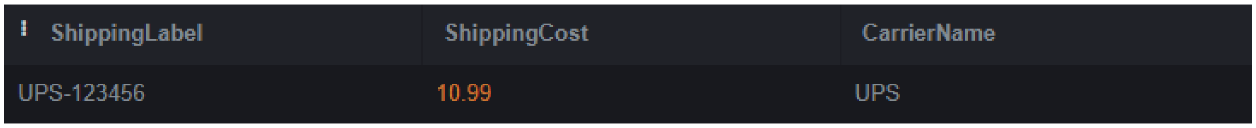
RoleName
Customer

7. Calculate Total Cost of Items in a Shopping Cart:
   SELECT SUM(Products.ProductPrice * CartItems.CartQuantity) AS TotalCost
   FROM CartItems
   INNER JOIN Products ON CartItems.ProductID = Products.ProductID
   WHERE CartItems.CartID = 1;



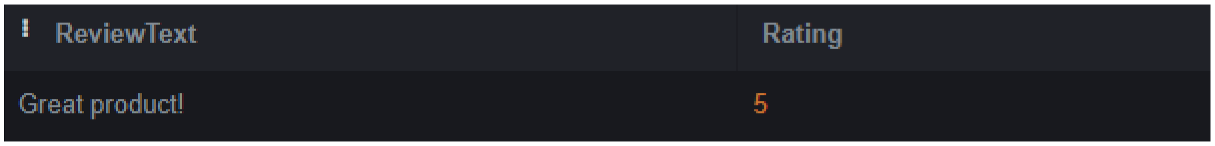
TotalCost
99.98

8. List Shipping Details for an Order:
   SELECT Shipping.ShippingLabel, Shipping.ShippingCost,
   ShippingCarriers.CarrierName
   FROM Shipping
   INNER JOIN ShippingCarriers ON Shipping.ShippingCarrierID =
   ShippingCarriers.CarrierID
   WHERE Shipping.OrderID = 1;



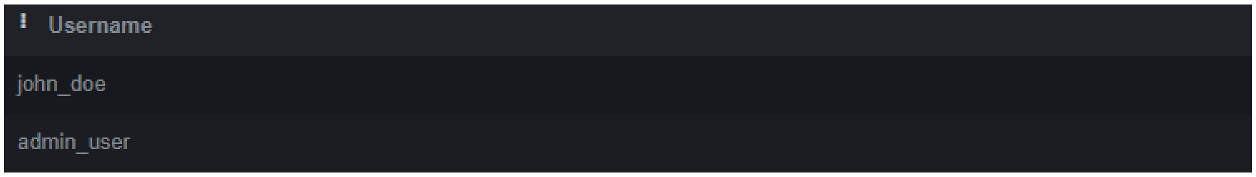
ShippingLabel	ShippingCost	CarrierName
UPS-123456	10.99	UPS

9. Retrieve All Reviews for a Product:
   SELECT ReviewText, Rating
   FROM ProductReviews
   WHERE ProductID = 1;



ReviewText	Rating
Great product!	5

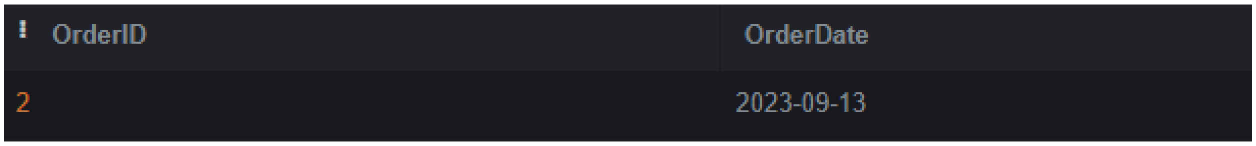
10. List All Users Who Have Placed Orders:
    SELECT DISTINCT Users.Username
    FROM Users
    INNER JOIN Orders ON Users.UserID = Orders.UserID;



Username
john_doe
admin_user

11. Find Orders with a Specific Status:

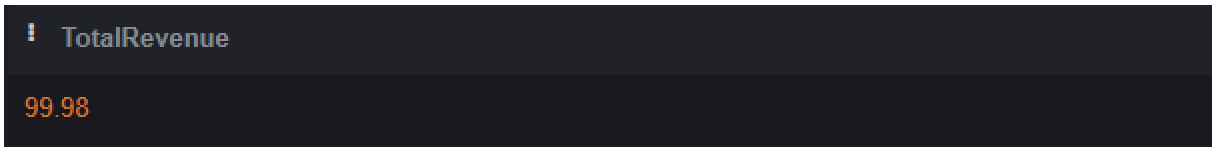
```
SELECT OrderID, OrderDate
FROM Orders
WHERE OrderStatus = 'Shipped';
```



OrderID	OrderDate
2	2023-09-13

12. Calculate Total Revenue for a Specific Category:

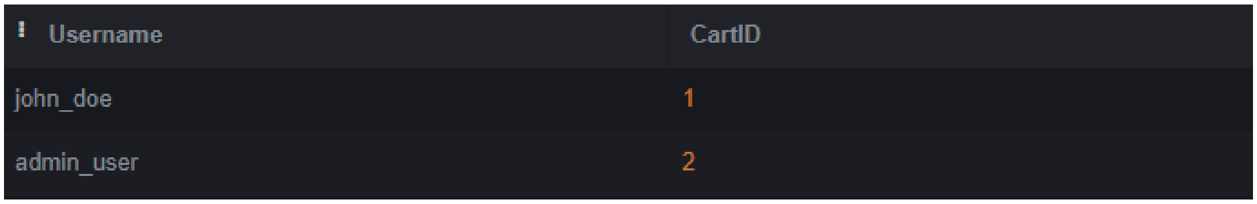
```
SELECT SUM(Products.ProductPrice * OrderItems.OrderQuantity) AS TotalRevenue
FROM Products
INNER JOIN OrderItems ON Products.ProductID = OrderItems.ProductID
WHERE Products.CategoryID = 1;
```



TotalRevenue
99.98

13. List Users with Their Active Shopping Carts:

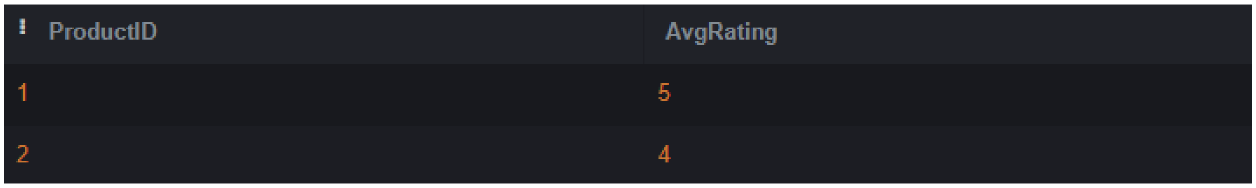
```
SELECT Users.Username, ShoppingCarts.CartID
FROM Users
LEFT JOIN ShoppingCarts ON Users.UserID = ShoppingCarts.UserID;
```



Username	CartID
john_doe	1
admin_user	2

14. Find Average Rating of Each Product:

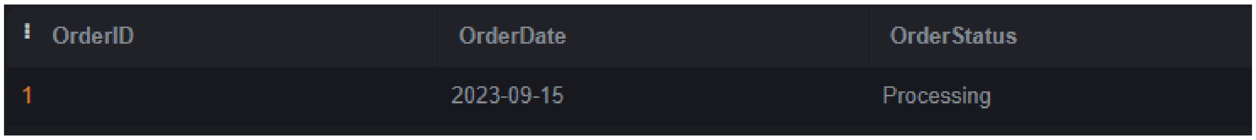
```
SELECT ProductID, AVG(Rating) AS AvgRating
FROM ProductReviews
GROUP BY ProductID;
```



ProductID	AvgRating
1	5
2	4

15. Find Orders Placed by a Specific User:

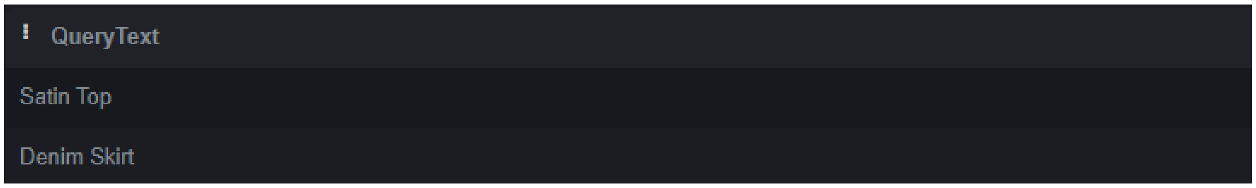
```
SELECT OrderID, OrderDate, OrderStatus
FROM Orders
WHERE UserID = 1;
```



OrderID	OrderDate	OrderStatus
1	2023-09-15	Processing

16. List All Unique Search Queries Used:

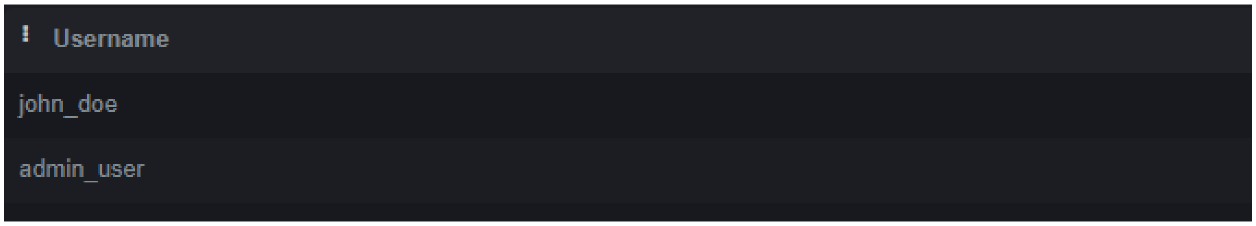
```
SELECT DISTINCT QueryText
FROM SearchQueries;
```



QueryText
Satin Top
Denim Skirt

17. Find Users that Reviewed a Products they Ordered:

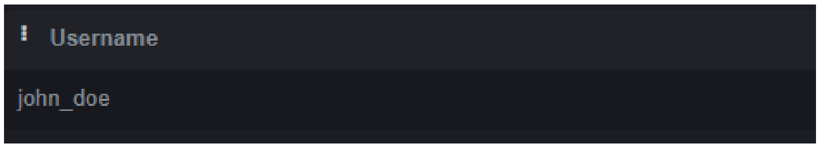
```
SELECT DISTINCT u.Username
FROM Users u
WHERE EXISTS (
SELECT 1
FROM Orders o
WHERE o.UserID = u.UserID
AND EXISTS (
SELECT 1
FROM ProductReviews pr
WHERE pr.ProductID IN (
SELECT ProductID
FROM OrderItems oi
WHERE oi.OrderID = o.OrderID
)
)
);
```



Username
john_doe
admin_user

18. Find Users without Admin Roles:

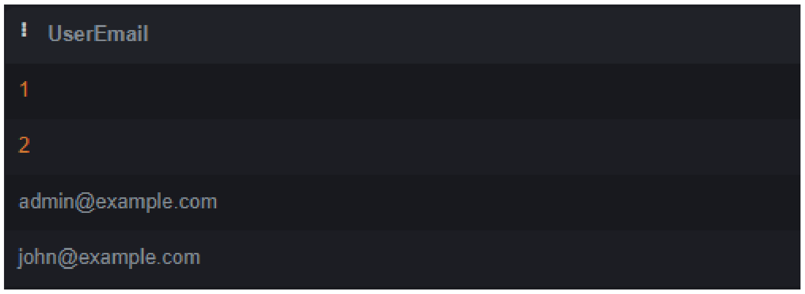
```
SELECT Username
FROM Users u
WHERE NOT EXISTS (
   SELECT 1
   FROM User_UserRoles ur
   WHERE ur.UserID = u.UserID
   AND ur.RoleID = 2
);
```



Username
john_doe

19. Get distinct UserEmails from Users and User_UserRoles tables:

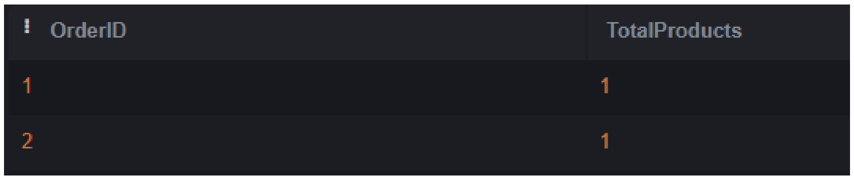
```
SELECT UserEmail FROM Users
UNION
SELECT UserID FROM User_UserRoles;
```



UserEmail
1
2
admin@example.com
john@example.com

20. Calculate the total number of products in each order:

```
SELECT Orders.OrderID, COUNT(OrderItems.ProductID) AS TotalProducts
FROM Orders
LEFT JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
GROUP BY Orders.OrderID;
```



OrderID	TotalProducts
1	1
2	1

Views

1. CartSummary:

```
CREATE VIEW CartSummary AS
SELECT
   ShoppingCarts.CartID,
   Users.Username,
   Products.ProductNumber,
   COUNT(CartItems.CartItemID) AS ItemCount,
   SUM(Products.ProductPrice) AS CartTotal
FROM ShoppingCarts
JOIN Users ON ShoppingCarts.UserID = Users.UserID
JOIN CartItems ON ShoppingCarts.CartID = CartItems.CartID
JOIN Products ON CartItems.ProductID = Products.ProductID
GROUP BY ShoppingCarts.CartID, Users.Username, Products.ProductNumber
```

2. OrderDetails:

```
CREATE VIEW OrderDetails AS
SELECT
   Orders.OrderID,
   Orders.OrderDate,
   Products.ProductID,
   Products.ProductNumber,
   Products.ProductPrice,
   Products.ProductStock
FROM Orders
JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
JOIN Products ON OrderItems.ProductID = Products.ProductID
```

3. OrderShippingView:

```
CREATE VIEW OrderShippingView AS
SELECT
   Orders.OrderID,
   Orders.OrderDate,
   Orders.OrderStatus,
   Shipping.ShippingLabel,
   Shipping.ShippingCost,
   ShippingCarriers.CarrierName
FROM Orders
JOIN Shipping ON Orders.OrderID = Shipping.OrderID
JOIN ShippingCarriers ON Shipping.ShippingCarrierID = ShippingCarriers.CarrierID
```

4. ProductReviewsSummary:

```
CREATE VIEW ProductReviewsSummary AS
SELECT
   Products.ProductID,
   Products.ProductNumber,
   COUNT(ProductReviews.ReviewID) AS ReviewCount,
   AVG(ProductReviews.Rating) AS AverageRating
FROM Products
LEFT JOIN ProductReviews ON Products.ProductID = ProductReviews.ProductID
```

```
GROUP BY Products.ProductID, Products.ProductNumber
```

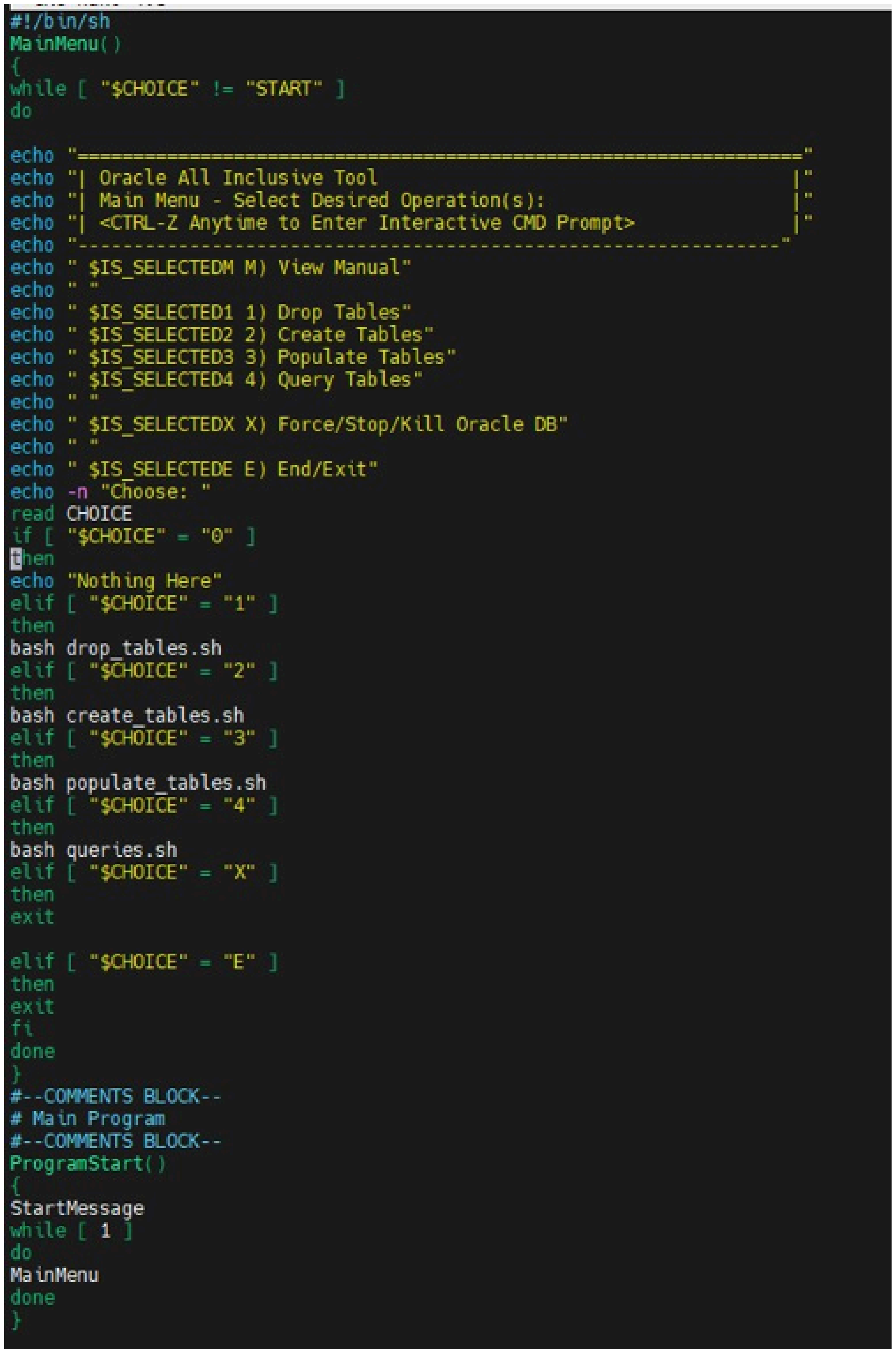
5. SearchQueryStats:

```
CREATE VIEW SearchQueryStats AS
SELECT
   SearchQueries.QueryText,
   COUNT(SearchQueries.SearchQueryID) AS QueryCount
FROM SearchQueries
GROUP BY SearchQueries.QueryText
```

6. UserOrderSummary:

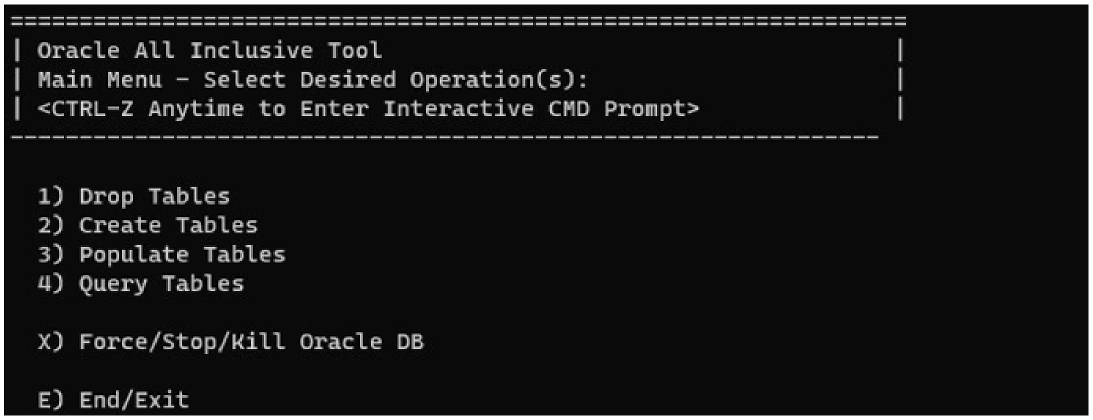
```
CREATE VIEW UserOrderSummary AS
SELECT
   Users.UserID,
   Users.Username,
   COUNT(Orders.OrderID) AS OrderCount,
   SUM(Products.ProductPrice) AS TotalSpending
FROM Users
LEFT JOIN Orders ON Users.UserID = Orders.UserID
LEFT JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
LEFT JOIN Products ON OrderItems.ProductID = Products.ProductID
GROUP BY Users.UserID, Users.Username
```

Bash Script (A5)



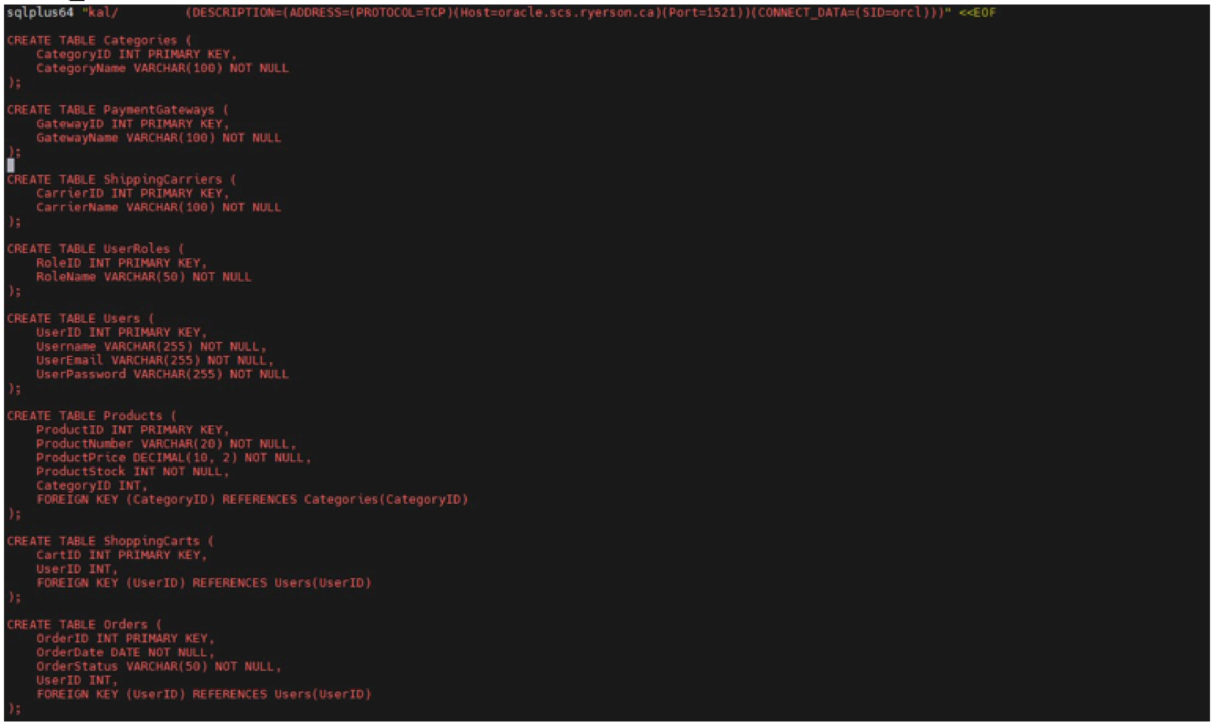
```
#!/bin/sh
MainMenu()
{
while [ "$CHOICE" != "START" ]
do

echo "=================================================================="
echo "| Oracle All Inclusive Tool                                        |"
echo "| Main Menu - Select Desired Operation(s):                         |"
echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt>                 |"
echo "----------------------------------------------------------------"
echo " $IS_SELECTEDM M) View Manual"
echo " "
echo " $IS_SELECTED1 1) Drop Tables"
echo " $IS_SELECTED2 2) Create Tables"
echo " $IS_SELECTED3 3) Populate Tables"
echo " $IS_SELECTED4 4) Query Tables"
echo " "
echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
echo " "
echo " $IS_SELECTEDE E) End/Exit"
echo -n "Choose: "
read CHOICE
if [ "$CHOICE" = "0" ]
then
echo "Nothing Here"
elif [ "$CHOICE" = "1" ]
then
bash drop_tables.sh
elif [ "$CHOICE" = "2" ]
then
bash create_tables.sh
elif [ "$CHOICE" = "3" ]
then
bash populate_tables.sh
elif [ "$CHOICE" = "4" ]
then
bash queries.sh
elif [ "$CHOICE" = "X" ]
then
exit

elif [ "$CHOICE" = "E" ]
then
exit
fi
done
}
#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart()
{
StartMessage
while [ 1 ]
do
MainMenu
done
}
```

result:

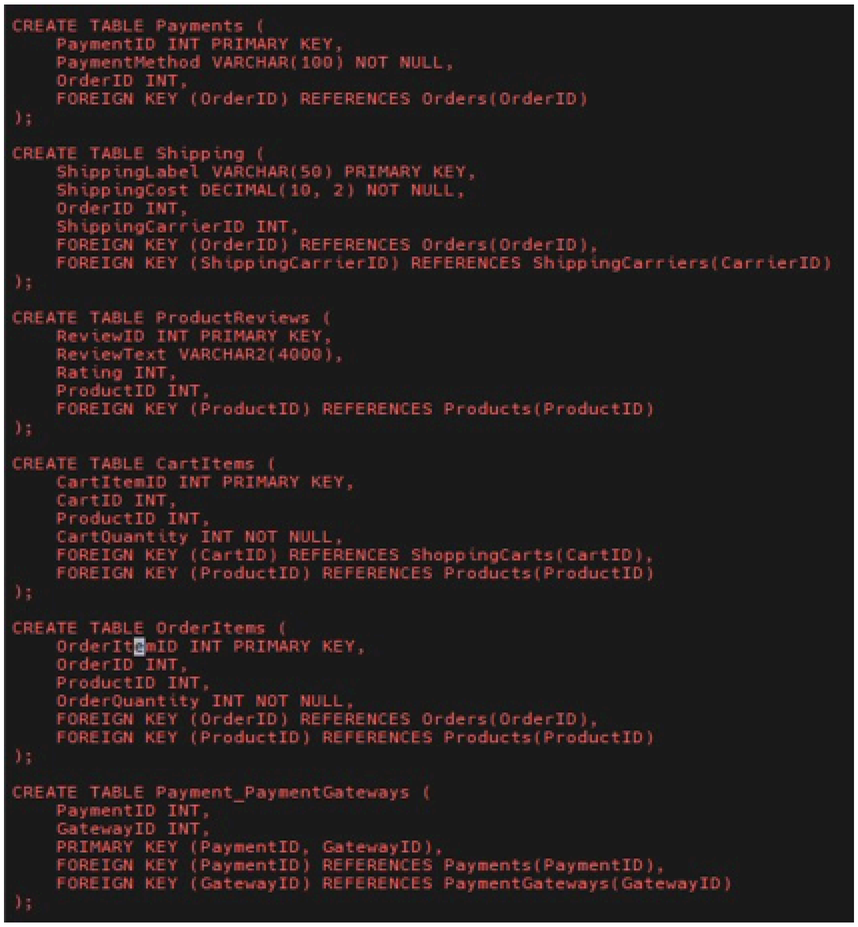


```
==================================================================
| Oracle All Inclusive Tool                                      |
| Main Menu - Select Desired Operation(s):                       |
| <CTRL-Z Anytime to Enter Interactive CMD Prompt>               |
----------------------------------------------------------------

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
```

create_tables.sh:

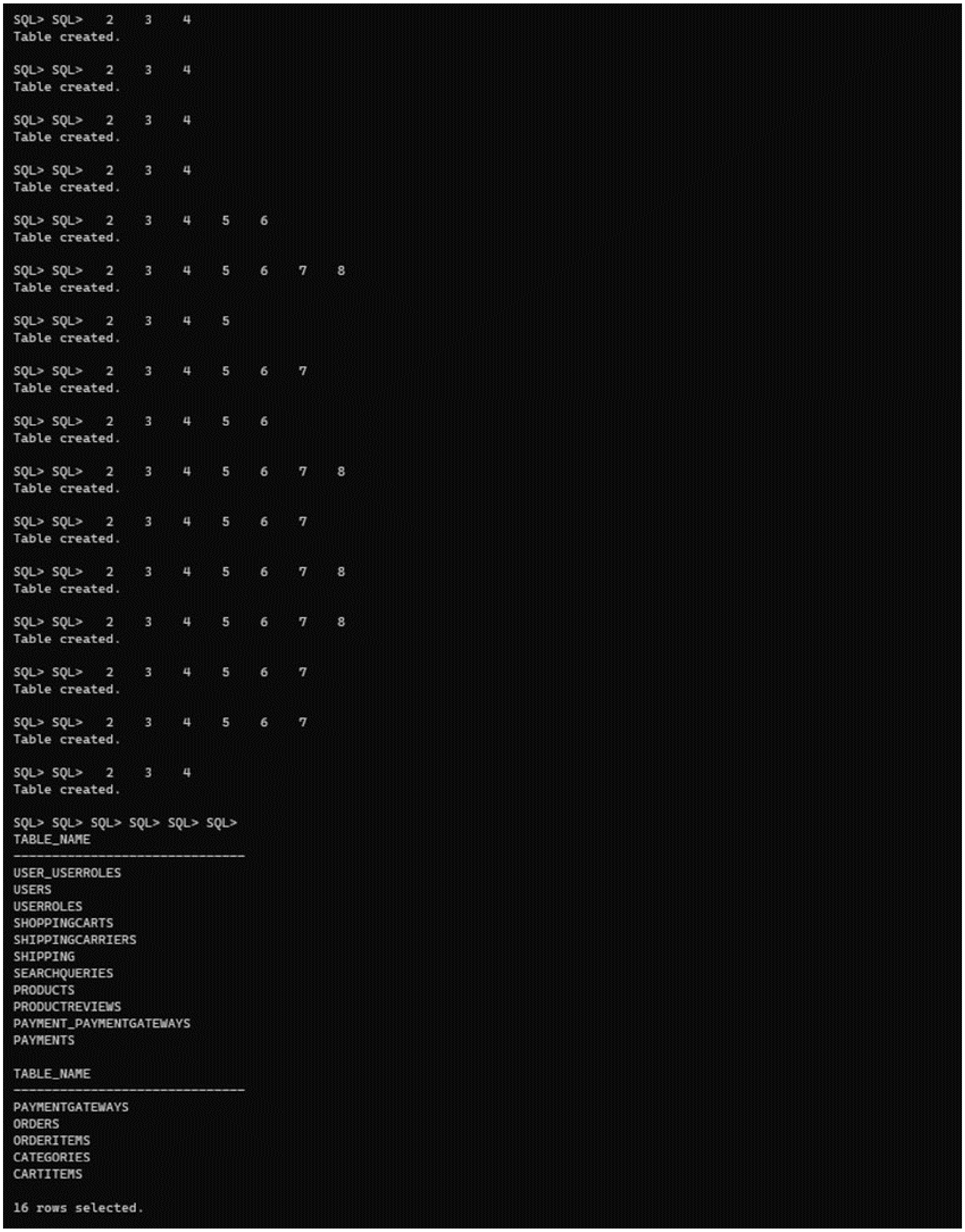


```
sqlplus64 "kal/        (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(100) NOT NULL
);

CREATE TABLE PaymentGateways (
    GatewayID INT PRIMARY KEY,
    GatewayName VARCHAR(100) NOT NULL
);

CREATE TABLE ShippingCarriers (
    CarrierID INT PRIMARY KEY,
    CarrierName VARCHAR(100) NOT NULL
);

CREATE TABLE UserRoles (
    RoleID INT PRIMARY KEY,
    RoleName VARCHAR(50) NOT NULL
);

CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(255) NOT NULL,
    UserEmail VARCHAR(255) NOT NULL,
    UserPassword VARCHAR(255) NOT NULL
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductNumber VARCHAR(20) NOT NULL,
    ProductPrice DECIMAL(10, 2) NOT NULL,
    ProductStock INT NOT NULL,
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

CREATE TABLE ShoppingCarts (
    CartID INT PRIMARY KEY,
    UserID INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE NOT NULL,
    OrderStatus VARCHAR(50) NOT NULL,
    UserID INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

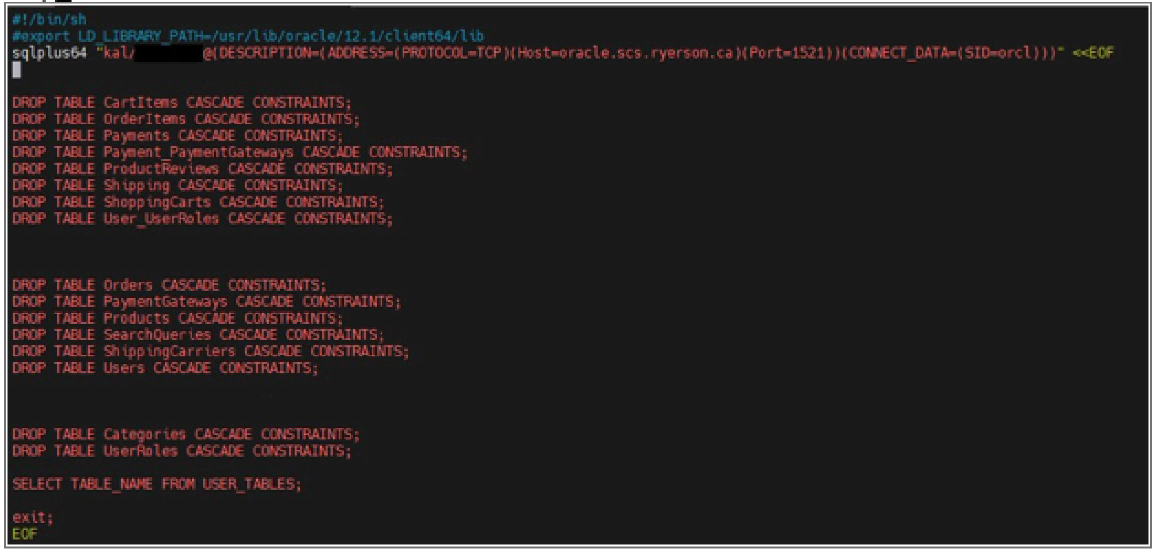


```
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY,
    PaymentMethod VARCHAR(100) NOT NULL,
    OrderID INT,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);

CREATE TABLE Shipping (
    ShippingLabel VARCHAR(50) PRIMARY KEY,
    ShippingCost DECIMAL(10, 2) NOT NULL,
    OrderID INT,
    ShippingCarrierID INT,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ShippingCarrierID) REFERENCES ShippingCarriers(CarrierID)
);

CREATE TABLE ProductReviews (
    ReviewID INT PRIMARY KEY,
    ReviewText VARCHAR2(4000),
    Rating INT,
    ProductID INT,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE CartItems (
    CartItemID INT PRIMARY KEY,
    CartID INT,
    ProductID INT,
    CartQuantity INT NOT NULL,
    FOREIGN KEY (CartID) REFERENCES ShoppingCarts(CartID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE OrderItems (
    OrderItemID INT PRIMARY KEY,
    OrderID INT,
    ProductID INT,
    OrderQuantity INT NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE Payment_PaymentGateways (
    PaymentID INT,
    GatewayID INT,
    PRIMARY KEY (PaymentID, GatewayID),
    FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
    FOREIGN KEY (GatewayID) REFERENCES PaymentGateways(GatewayID)
);
```

result:

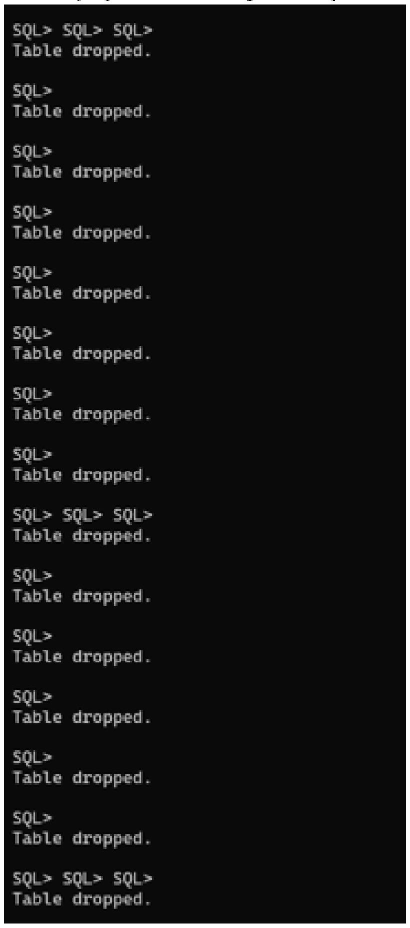

```
SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL> SQL> SQL> SQL> SQL>
TABLE_NAME
------------------------------
USER_USERROLES
USERS
USERROLES
SHOPPINGCARTS
SHIPPINGCARRIERS
SHIPPING
SEARCHQUERIES
PRODUCTS
PRODUCTREVIEWS
PAYMENT_PAYMENTGATEWAYS
PAYMENTS

TABLE_NAME
------------------------------
PAYMENTGATEWAYS
ORDERS
ORDERITEMS
CATEGORIES
CARTITEMS

16 rows selected.
```

drop_tables.sh:



```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "kal/[redacted]@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF


DROP TABLE CartItems CASCADE CONSTRAINTS;
DROP TABLE OrderItems CASCADE CONSTRAINTS;
DROP TABLE Payments CASCADE CONSTRAINTS;
DROP TABLE Payment_PaymentGateways CASCADE CONSTRAINTS;
DROP TABLE ProductReviews CASCADE CONSTRAINTS;
DROP TABLE Shipping CASCADE CONSTRAINTS;
DROP TABLE ShoppingCarts CASCADE CONSTRAINTS;
DROP TABLE User_UserRoles CASCADE CONSTRAINTS;



DROP TABLE Orders CASCADE CONSTRAINTS;
DROP TABLE PaymentGateways CASCADE CONSTRAINTS;
DROP TABLE Products CASCADE CONSTRAINTS;
DROP TABLE SearchQueries CASCADE CONSTRAINTS;
DROP TABLE ShippingCarriers CASCADE CONSTRAINTS;
DROP TABLE Users CASCADE CONSTRAINTS;



DROP TABLE Categories CASCADE CONSTRAINTS;
DROP TABLE UserRoles CASCADE CONSTRAINTS;

SELECT TABLE_NAME FROM USER_TABLES;

exit;
EOF
```

result (repeated many times):



```
SQL> SQL> SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL> SQL> SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL> SQL> SQL>
Table dropped.
```

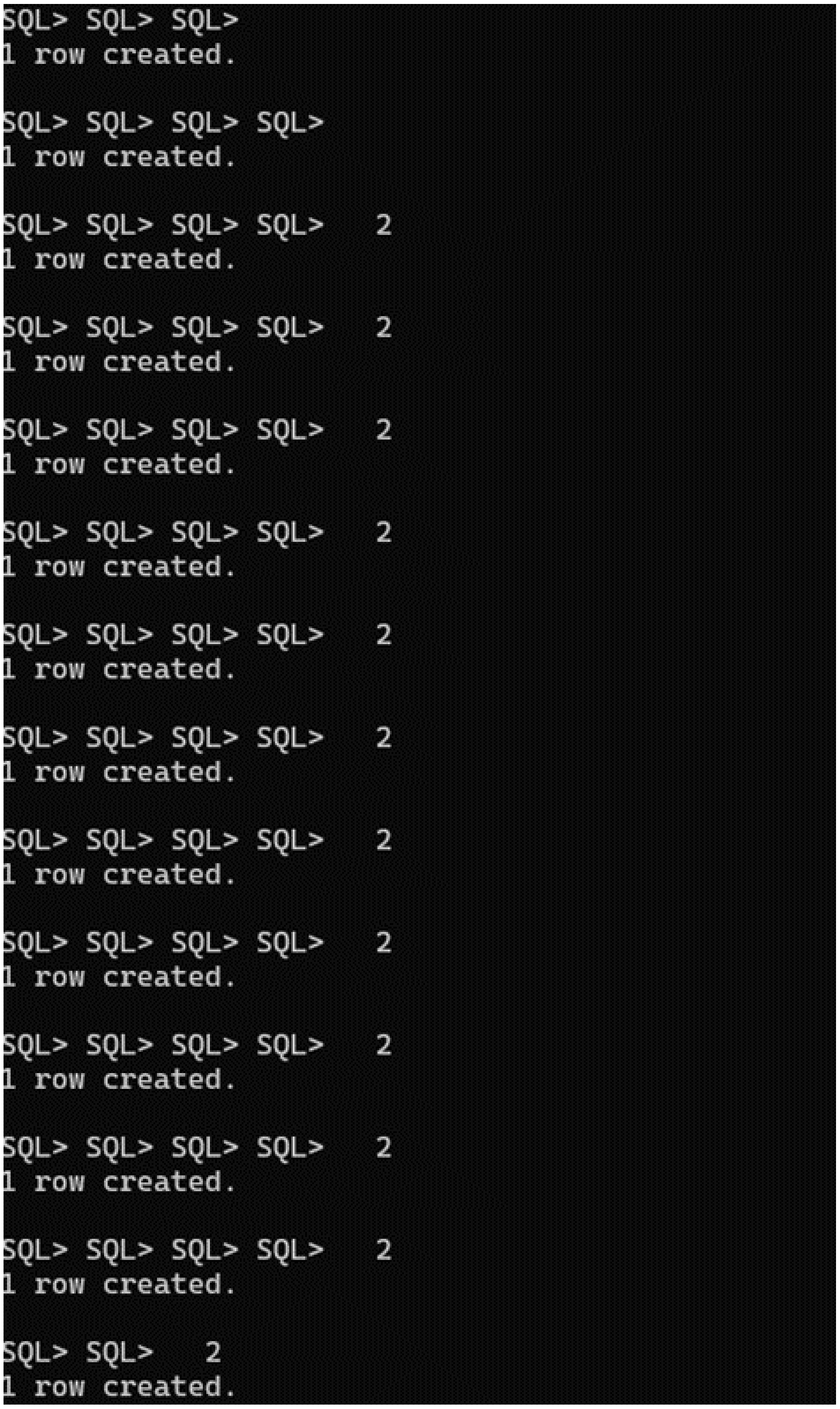
populate_tables.sh:
#!/bin/sh
sqlplus64

```
"kal/REDACTED@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerso
n. ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

INSERT INTO Users (UserID, Username, UserEmail, UserPassword) VALUES (1,
'john_doe',

'john@example.com', 'password123');


INSERT INTO Users (UserID, Username, UserEmail, UserPassword) VALUES ('2',
'admin_user', 'admin@example.com', 'adminpass');


INSERT INTO UserRoles (RoleID, RoleName)
VALUES ('1', 'Customer');

INSERT INTO UserRoles (RoleID, RoleName)
VALUES ('2', 'Admin');

INSERT INTO User_UserRoles (UserID, RoleID)
VALUES ('1', '1');

INSERT INTO User_UserRoles (UserID, RoleID)
VALUES ('2', '2');

INSERT INTO Categories (CategoryID, CategoryName)
VALUES ('1', 'Tops');

INSERT INTO Categories (CategoryID, CategoryName)

VALUES (2, 'Bottoms');

INSERT INTO Products (ProductID, ProductNumber, ProductPrice, ProductStock,
CategoryID) VALUES ('1', 'P12345', '49.99', '100', '1');

INSERT INTO Products (ProductID, ProductNumber, ProductPrice, ProductStock,
CategoryID)

VALUES ('2', 'C56789', '29.99', '100', '2');


INSERT INTO Orders (OrderID, OrderDate, OrderStatus, UserID)

VALUES ('1', TO_DATE('2023-09-15', 'YYYY-MM-DD'), 'Processing', '1');
```

```
INSERT INTO Orders (OrderID, OrderDate, OrderStatus, UserID)

VALUES ('2',TO_DATE('2023-09-13', 'YYYY-MM-DD'), 'Shipped', '2');


INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, OrderQuantity)
VALUES (3, 1, 1, 5);

INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, OrderQuantity)

VALUES (4, 2, 2, 3);


INSERT INTO Payments (PaymentID, PaymentMethod, OrderID)
VALUES ('1', 'Credit Card', '1');


INSERT INTO Payments (PaymentID, PaymentMethod, OrderID)
VALUES ('2', 'PayPal', '2');


INSERT INTO PaymentGateways (GatewayID, GatewayName)

VALUES ('1', 'Stripe');


INSERT INTO PaymentGateways (GatewayID, GatewayName)
VALUES ('2', 'PayPal');


INSERT INTO Payment_PaymentGateways (PaymentID, GatewayID)
VALUES ('1', '1');


INSERT INTO Payment_PaymentGateways (PaymentID, GatewayID)

VALUES ('2', '2');


INSERT INTO ShoppingCarts (CartID, UserID)
VALUES ('1', '1');

INSERT INTO ShoppingCarts (CartID, UserID)
VALUES ('2', '2');
```

```
INSERT INTO CartItems (CartItemID, CartID, ProductID, CartQuantity)

VALUES ('1', '1', '1', '2');

INSERT INTO CartItems (CartItemID, CartID, ProductID, CartQuantity)

VALUES ('2', '2', '2', '1');


INSERT INTO ShippingCarriers (CarrierID, CarrierName)
VALUES ('1', 'Canada Post');


INSERT INTO ShippingCarriers (CarrierID, CarrierName)
VALUES ('2', 'Fedex');


INSERT INTO ShippingCarriers (CarrierID, CarrierName)

VALUES ('3', 'UPS');


INSERT INTO Shipping (ShippingLabel, ShippingCost, OrderID, ShippingCarrierID)
VALUES ('UPS-123456', '10.99', '1', '3');


INSERT INTO Shipping (ShippingLabel, ShippingCost, OrderID, ShippingCarrierID)

VALUES ('Canada Post-789012', '7.99', '2', '1');


INSERT INTO SearchQueries (SearchQueryID, QueryText)
VALUES ('1', 'Satin Top');


INSERT INTO SearchQueries (SearchQueryID, QueryText)
VALUES ('2', 'Denim Skirt');


INSERT INTO ProductReviews (ReviewID, ReviewText, Rating, ProductID)
VALUES ('1', 'Great product!', '5', '1');
INSERT INTO ProductReviews (ReviewID, ReviewText, Rating,
ProductID) VALUES ('2', 'Excellent quality for the price', '4', '2'); exit;
EOF
```

result (this is repeated many times):



```
SQL> SQL> SQL>
1 row created.

SQL> SQL> SQL> SQL>
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL> SQL> SQL>   2
1 row created.

SQL> SQL>   2
1 row created.
```

FOR THE QUERIES FILE SEE ORIGINAL A5, IT IS TOO LONG TO PUT HERE!

Database Normalization

Users

- Functional Dependency: UserID → Username, UserEmail, UserPassword
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since UserID is the only candidate key.

UserRoles

- Functional Dependency: RoleID → RoleName
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since RoleID is the only candidate key.

User_UserRoles

- Functional Dependencies: (UserID, RoleID) → None
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since (UserID, RoleID) is the only candidate key.

Categories

- Functional Dependency: CategoryID → CategoryName
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since CategoryID is the only candidate key.

Products

- Functional Dependency: ProductID → ProductNumber, ProductPrice, ProductStock, CategoryID
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.

- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since ProductID is the only candidate key.

Orders
- Functional Dependency: OrderID → OrderDate, OrderStatus, UserID
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF because OrderID is the only candidate key.

OrderItems
- Functional Dependency: (OrderID, ProductID) → OrderQuantity
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF because (OrderID, ProductID) is the only candidate key.

Payments
- Functional Dependency: PaymentID → PaymentMethod, OrderID
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since PaymentID is the only candidate key.

PaymentGateways
- Functional Dependency: GatewayID → GatewayName
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since GatewayID is the only candidate key.

Payment_PaymentGateways
- Functional Dependencies: (PaymentID, GatewayID) → None
- This table is in 1NF because all values are atomic.

- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since (PaymentID, GatewayID) is the only candidate key.

ShoppingCarts

- Functional Dependency: CartID → UserID
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since CartID is the only candidate key.

CartItems

- Functional Dependency: (CartID, ProductID) → CartQuantity
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF because (CartID, ProductID) is the only candidate key.

ShippingCarriers

- Functional Dependency: CarrierID → CarrierName
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since CarrierID is the only candidate key.

Shipping

- Functional Dependency: ShippingLabel → ShippingCost, OrderID, ShippingCarrierID
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF because ShippingLabel is the only candidate key..

SearchQueries

- Functional Dependency: SearchQueryID → QueryText

- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF since SearchQueryID is the only candidate key.

ProductReviews

- Functional Dependency: ReviewID → ReviewText, Rating, ProductID
- This table is in 1NF because all values are atomic.
- This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key.
- This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key.
- This table is in BCNF because ReviewID is the only candidate key.

Bernstein's Algorithm:

```
CREATE TABLE ProductReviews_NonBCNF (
   ReviewID INT PRIMARY KEY,
   ReviewText VARCHAR2(4000),
   Rating INT,
   ProductID INT,
   ProductNumber VARCHAR(20), -- Introduce a non-prime attribute
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

Step 1: List of Attributes and FDs
Attributes: {ReviewID, ReviewText, Rating, ProductID, ProductName}
FDs: {ReviewID → ReviewText, Rating, ProductID; ProductID → ProductNumber}

Step 2: Reduce the list of FDs (Minimal Cover)
ReviewID → ReviewText, Rating;

Step 3: Find the Keys
Candidate key: {ReviewID}

Step 4: Derive the Final Schema

```
CREATE TABLE ProductReviews (
   ReviewID INT PRIMARY KEY,
   ReviewText VARCHAR2(4000),
   Rating INT,
   ProductID INT,
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductNumber VARCHAR(20) NOT NULL,
   ProductPrice DECIMAL(10, 2) NOT NULL,
   ProductStock INT NOT NULL,
   CategoryID INT,
   FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

GUI

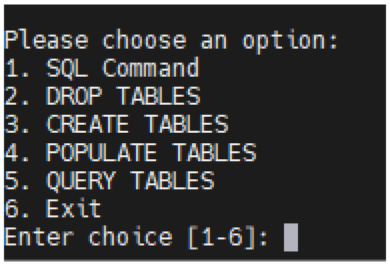
Our DB application is in 3NF/BCNF.

NOTE: IN THE JAVA CODE YOU NEED TO REPLACE THE USERNAME AND PASSWORD FIELDS WITH YOUR OWN USERNAME/PASSWORD OR ELSE IT WONT WORK!

How to run the code: Put assignment9.java and ojdbc6.jar in the same folder and run the following commands:

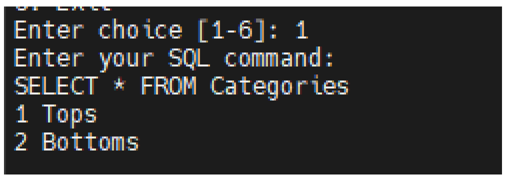
```
javac -cp .:ojdbc6.jar assignment9.java
java -cp .:ojdbc6.jar assignment9
```

The full code is submitted, it's far too long to include here. Additionally, my username and password have been redacted from the file, to make the program work you must add your own.



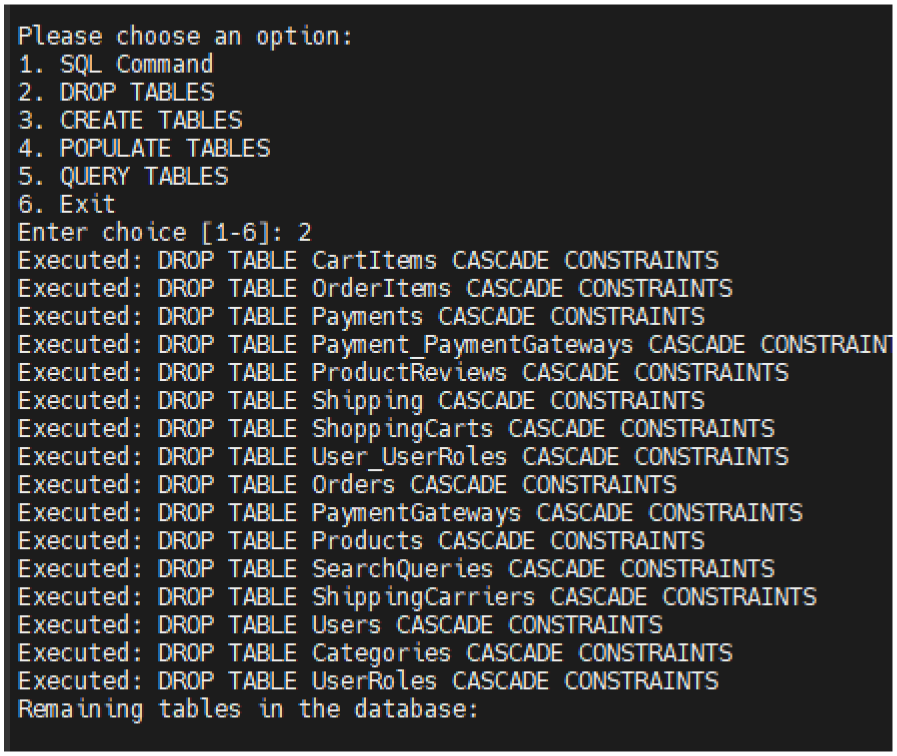
```
Please choose an option:
1. SQL Command
2. DROP TABLES
3. CREATE TABLES
4. POPULATE TABLES
5. QUERY TABLES
6. Exit
Enter choice [1-6]:
```

Option 1 allows the user to enter any SQL command to the database.



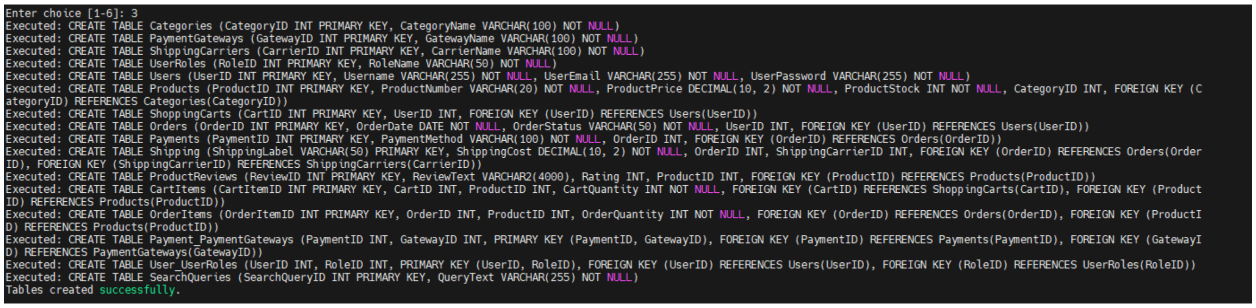
```
Enter choice [1-6]: 1
Enter your SQL command:
SELECT * FROM Categories
1 Tops
2 Bottoms
```

Option 2 drops all the tables in appropriate order.



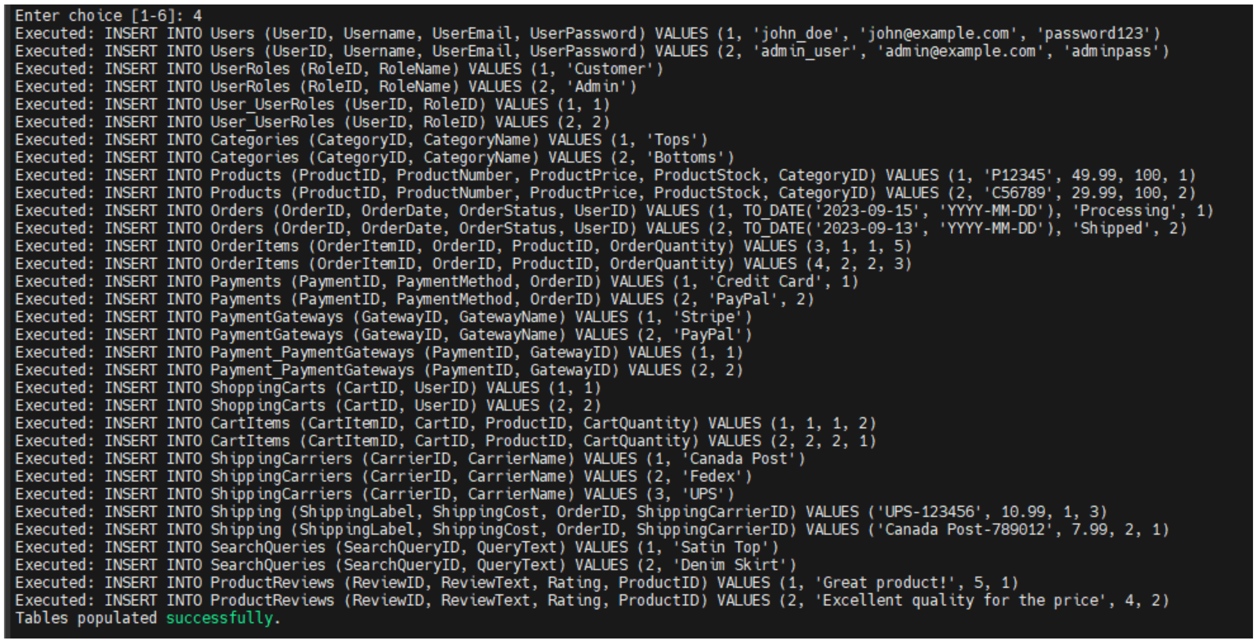
```
Please choose an option:
1. SQL Command
2. DROP TABLES
3. CREATE TABLES
4. POPULATE TABLES
5. QUERY TABLES
6. Exit
Enter choice [1-6]: 2
Executed: DROP TABLE CartItems CASCADE CONSTRAINTS
Executed: DROP TABLE OrderItems CASCADE CONSTRAINTS
Executed: DROP TABLE Payments CASCADE CONSTRAINTS
Executed: DROP TABLE Payment_PaymentGateways CASCADE CONSTRAINT
Executed: DROP TABLE ProductReviews CASCADE CONSTRAINTS
Executed: DROP TABLE Shipping CASCADE CONSTRAINTS
Executed: DROP TABLE ShoppingCarts CASCADE CONSTRAINTS
Executed: DROP TABLE User_UserRoles CASCADE CONSTRAINTS
Executed: DROP TABLE Orders CASCADE CONSTRAINTS
Executed: DROP TABLE PaymentGateways CASCADE CONSTRAINTS
Executed: DROP TABLE Products CASCADE CONSTRAINTS
Executed: DROP TABLE SearchQueries CASCADE CONSTRAINTS
Executed: DROP TABLE ShippingCarriers CASCADE CONSTRAINTS
Executed: DROP TABLE Users CASCADE CONSTRAINTS
Executed: DROP TABLE Categories CASCADE CONSTRAINTS
Executed: DROP TABLE UserRoles CASCADE CONSTRAINTS
Remaining tables in the database:
```

Option 3 creates all the tables.



```
Enter choice [1-6]: 3
Executed: CREATE TABLE Categories (CategoryID INT PRIMARY KEY, CategoryName VARCHAR(100) NOT NULL)
Executed: CREATE TABLE PaymentGateways (GatewayID INT PRIMARY KEY, GatewayName VARCHAR(100) NOT NULL)
Executed: CREATE TABLE ShippingCarriers (CarrierID INT PRIMARY KEY, CarrierName VARCHAR(100) NOT NULL)
Executed: CREATE TABLE UserRoles (RoleID INT PRIMARY KEY, RoleName VARCHAR(50) NOT NULL)
Executed: CREATE TABLE Users (UserID INT PRIMARY KEY, Username VARCHAR(255) NOT NULL, UserEmail VARCHAR(255) NOT NULL, UserPassword VARCHAR(255) NOT NULL)
Executed: CREATE TABLE Products (ProductID INT PRIMARY KEY, ProductNumber VARCHAR(20) NOT NULL, ProductPrice DECIMAL(10, 2) NOT NULL, ProductStock INT NOT NULL, CategoryID INT, FOREIGN KEY (C
ategoryID) REFERENCES Categories(CategoryID))
Executed: CREATE TABLE ShoppingCarts (CartID INT PRIMARY KEY, UserID INT, FOREIGN KEY (UserID) REFERENCES Users(UserID))
Executed: CREATE TABLE Orders (OrderID INT PRIMARY KEY, OrderDate DATE NOT NULL, OrderStatus VARCHAR(50) NOT NULL, UserID INT, FOREIGN KEY (UserID) REFERENCES Users(UserID))
Executed: CREATE TABLE Payments (PaymentID INT PRIMARY KEY, PaymentMethod VARCHAR(100) NOT NULL, OrderID INT, FOREIGN KEY (OrderID) REFERENCES Orders(OrderID))
Executed: CREATE TABLE Shipping (ShippingLabel VARCHAR(50) PRIMARY KEY, ShippingCost DECIMAL(10, 2) NOT NULL, OrderID INT, ShippingCarrierID INT, FOREIGN KEY (OrderID) REFERENCES Orders(Order
ID), FOREIGN KEY (ShippingCarrierID) REFERENCES ShippingCarriers(CarrierID))
Executed: CREATE TABLE ProductReviews (ReviewID INT PRIMARY KEY, ReviewText VARCHAR2(4000), Rating INT, ProductID INT, FOREIGN KEY (ProductID) REFERENCES Products(ProductID))
Executed: CREATE TABLE CartItems (CartItemID INT PRIMARY KEY, CartID INT, ProductID INT, CartQuantity INT NOT NULL, FOREIGN KEY (CartID) REFERENCES ShoppingCarts(CartID), FOREIGN KEY (Product
ID) REFERENCES Products(ProductID))
Executed: CREATE TABLE OrderItems (OrderItemID INT PRIMARY KEY, OrderID INT, ProductID INT, OrderQuantity INT NOT NULL, FOREIGN KEY (OrderID) REFERENCES Orders(OrderID), FOREIGN KEY (ProductI
D) REFERENCES Products(ProductID))
Executed: CREATE TABLE Payment_PaymentGateways (PaymentID INT, GatewayID INT, PRIMARY KEY (PaymentID, GatewayID), FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID), FOREIGN KEY (GatewayI
D) REFERENCES PaymentGateways(GatewayID))
Executed: CREATE TABLE User_UserRoles (UserID INT, RoleID INT, PRIMARY KEY (UserID, RoleID), FOREIGN KEY (UserID) REFERENCES Users(UserID), FOREIGN KEY (RoleID) REFERENCES UserRoles(RoleID))
Executed: CREATE TABLE SearchQueries (SearchQueryID INT PRIMARY KEY, QueryText VARCHAR(255) NOT NULL)
Tables created successfully.
```

Option 4 populates all the tables.



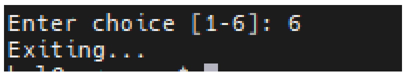
```
Enter choice [1-6]: 4
Executed: INSERT INTO Users (UserID, Username, UserEmail, UserPassword) VALUES (1, 'john_doe', 'john@example.com', 'password123')
Executed: INSERT INTO Users (UserID, Username, UserEmail, UserPassword) VALUES (2, 'admin_user', 'admin@example.com', 'adminpass')
Executed: INSERT INTO UserRoles (RoleID, RoleName) VALUES (1, 'Customer')
Executed: INSERT INTO UserRoles (RoleID, RoleName) VALUES (2, 'Admin')
Executed: INSERT INTO User_UserRoles (UserID, RoleID) VALUES (1, 1)
Executed: INSERT INTO User_UserRoles (UserID, RoleID) VALUES (2, 2)
Executed: INSERT INTO Categories (CategoryID, CategoryName) VALUES (1, 'Tops')
Executed: INSERT INTO Categories (CategoryID, CategoryName) VALUES (2, 'Bottoms')
Executed: INSERT INTO Products (ProductID, ProductNumber, ProductPrice, ProductStock, CategoryID) VALUES (1, 'P12345', 49.99, 100, 1)
Executed: INSERT INTO Products (ProductID, ProductNumber, ProductPrice, ProductStock, CategoryID) VALUES (2, 'C56789', 29.99, 100, 2)
Executed: INSERT INTO Orders (OrderID, OrderDate, OrderStatus, UserID) VALUES (1, TO_DATE('2023-09-15', 'YYYY-MM-DD'), 'Processing', 1)
Executed: INSERT INTO Orders (OrderID, OrderDate, OrderStatus, UserID) VALUES (2, TO_DATE('2023-09-13', 'YYYY-MM-DD'), 'Shipped', 2)
Executed: INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, OrderQuantity) VALUES (3, 1, 1, 5)
Executed: INSERT INTO OrderItems (OrderItemID, OrderID, ProductID, OrderQuantity) VALUES (4, 2, 2, 3)
Executed: INSERT INTO Payments (PaymentID, PaymentMethod, OrderID) VALUES (1, 'Credit Card', 1)
Executed: INSERT INTO Payments (PaymentID, PaymentMethod, OrderID) VALUES (2, 'PayPal', 2)
Executed: INSERT INTO PaymentGateways (GatewayID, GatewayName) VALUES (1, 'Stripe')
Executed: INSERT INTO PaymentGateways (GatewayID, GatewayName) VALUES (2, 'PayPal')
Executed: INSERT INTO Payment_PaymentGateways (PaymentID, GatewayID) VALUES (1, 1)
Executed: INSERT INTO Payment_PaymentGateways (PaymentID, GatewayID) VALUES (2, 2)
Executed: INSERT INTO ShoppingCarts (CartID, UserID) VALUES (1, 1)
Executed: INSERT INTO ShoppingCarts (CartID, UserID) VALUES (2, 2)
Executed: INSERT INTO CartItems (CartItemID, CartID, ProductID, CartQuantity) VALUES (1, 1, 1, 2)
Executed: INSERT INTO CartItems (CartItemID, CartID, ProductID, CartQuantity) VALUES (2, 2, 2, 1)
Executed: INSERT INTO ShippingCarriers (CarrierID, CarrierName) VALUES (1, 'Canada Post')
Executed: INSERT INTO ShippingCarriers (CarrierID, CarrierName) VALUES (2, 'Fedex')
Executed: INSERT INTO ShippingCarriers (CarrierID, CarrierName) VALUES (3, 'UPS')
Executed: INSERT INTO Shipping (ShippingLabel, ShippingCost, OrderID, ShippingCarrierID) VALUES ('UPS-123456', 10.99, 1, 3)
Executed: INSERT INTO Shipping (ShippingLabel, ShippingCost, OrderID, ShippingCarrierID) VALUES ('Canada Post-789012', 7.99, 2, 1)
Executed: INSERT INTO SearchQueries (SearchQueryID, QueryText) VALUES (1, 'Satin Top')
Executed: INSERT INTO SearchQueries (SearchQueryID, QueryText) VALUES (2, 'Denim Skirt')
Executed: INSERT INTO ProductReviews (ReviewID, ReviewText, Rating, ProductID) VALUES (1, 'Great product!', 5, 1)
Executed: INSERT INTO ProductReviews (ReviewID, ReviewText, Rating, ProductID) VALUES (2, 'Excellent quality for the price', 4, 2)
Tables populated successfully.
```

Option 5 queries the tables with the queries we made previously (Too much to screenshot):

```
Enter choice [1-6]: 5

Executing query: SELECT 'User Information for john_doe:' as Description, Users.* FROM Users WHERE Username = 'john_doe'
User Information for john_doe: 1 john_doe john@example.com password123

Executing query: SELECT 'Products in Category 1:' as Description, Products.* FROM Products WHERE CategoryID = 1
Products in Category 1: 1 P12345 49.99 100 1

Executing query: SELECT 'Orders Shipped on 2023-09-13:' as Description, Orders.* FROM Orders WHERE OrderDate = TO_DATE('2023-09-13', 'YYYY-MM-DD')
Orders Shipped on 2023-09-13: 2 2023-09-13 00:00:00.0 Shipped 2

Executing query: SELECT 'Payment Methods for OrderID 1:' as Description, PaymentMethod FROM Payments WHERE OrderID = 1
Payment Methods for OrderID 1: Credit Card

Executing query: SELECT 'Product and Their Categories:' as Description, Products.ProductID, Products.ProductNumber, Products.ProductPrice, Categories.CategoryName FROM Products INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID
Product and Their Categories: 1 P12345 49.99 Tops
Product and Their Categories: 2 C56789 29.99 Bottoms
```

```
Executing query: SELECT 'Roles for UserID 1:' as Description, UserRoles.RoleName FROM
UserRoles INNER JOIN User_UserRoles ON UserRoles.RoleID = User_UserRoles.RoleID
WHERE User_UserRoles.UserID = 1
Roles for UserID 1: Customer

Executing query: SELECT 'Total Cost for CartID 1:' as Description, SUM(Products.ProductPrice
* CartItems.CartQuantity) AS TotalCost FROM CartItems INNER JOIN Products ON
CartItems.ProductID = Products.ProductID WHERE CartItems.CartID = 1
Total Cost for CartID 1: 99.98

Executing query: SELECT 'Shipping Details for OrderID 1:' as Description,
Shipping.ShippingLabel, Shipping.ShippingCost, ShippingCarriers.CarrierName FROM
Shipping INNER JOIN ShippingCarriers ON Shipping.ShippingCarrierID =
ShippingCarriers.CarrierID WHERE Shipping.OrderID = 1
Shipping Details for OrderID 1: UPS-123456 10.99 UPS

Executing query: SELECT 'Reviews for ProductID 1:' as Description, ReviewText, Rating FROM
ProductReviews WHERE ProductID = 1
Reviews for ProductID 1: Great product! 5

Executing query: SELECT DISTINCT 'Users who have placed orders:' as Description,
Users.Username FROM Users INNER JOIN Orders ON Users.UserID = Orders.UserID
Users who have placed orders: admin_user
Users who have placed orders: john_doe

Executing query: SELECT 'Orders with Status Shipped:' as Description, OrderID, OrderDate
FROM Orders WHERE OrderStatus = 'Shipped'
Orders with Status Shipped: 2 2023-09-13 00:00:00.0

Executing query: SELECT 'Total Revenue for Category 1:' as Description,
SUM(Products.ProductPrice * OrderItems.OrderQuantity) AS TotalRevenue FROM Products
INNER JOIN OrderItems ON Products.ProductID = OrderItems.ProductID WHERE
Products.CategoryID = 1
Total Revenue for Category 1: 249.95

Executing query: SELECT 'Users with active carts:' as Description, Users.Username,
ShoppingCarts.CartID FROM Users LEFT JOIN ShoppingCarts ON Users.UserID =
ShoppingCarts.UserID
Users with active carts: john_doe 1
Users with active carts: admin_user 2

Executing query: SELECT 'Average Rating for each product:' as Description, ProductID,
AVG(Rating) AS AvgRating FROM ProductReviews GROUP BY ProductID
Average Rating for each product: 1 5
Average Rating for each product: 2 4

Executing query: SELECT 'Orders placed by UserID 1:' as Description, OrderID, OrderDate,
OrderStatus FROM Orders WHERE UserID = 1
Orders placed by UserID 1: 1 2023-09-15 00:00:00.0 Processing

Executing query: SELECT DISTINCT 'Unique search queries:' as Description, QueryText FROM
SearchQueries
Unique search queries: Satin Top
Unique search queries: Denim Skirt

Executing query: SELECT DISTINCT 'Users who reviewed products they ordered:' as
Description, u.Username FROM Users u WHERE EXISTS (SELECT 1 FROM Orders o WHERE
o.UserID = u.UserID AND EXISTS (SELECT 1 FROM ProductReviews pr WHERE pr.ProductID
IN (SELECT ProductID FROM OrderItems oi WHERE oi.OrderID = o.OrderID)))
Users who reviewed products they ordered: admin_user
Users who reviewed products they ordered: john_doe

Executing query: SELECT 'Users without Admin Role:' as Description, Username FROM Users
u WHERE NOT EXISTS (SELECT 1 FROM User_UserRoles ur WHERE ur.UserID = u.UserID
AND ur.RoleID = 2)
Users without Admin Role: john_doe

Executing query: SELECT 'Distinct Emails from Users:' as Description, UserEmail FROM Users
UNION SELECT 'Distinct UserIDs from User_UserRoles:' as Description, TO_CHAR(UserID)
FROM User_UserRoles
Distinct Emails from Users: admin@example.com
Distinct Emails from Users: john@example.com
Distinct UserIDs from User_UserRoles: 1
Distinct UserIDs from User_UserRoles: 2

Executing query: SELECT 'Total products in each order:' as Description, Orders.OrderID,
COUNT(OrderItems.ProductID) AS TotalProducts FROM Orders LEFT JOIN OrderItems ON
Orders.OrderID = OrderItems.OrderID GROUP BY Orders.OrderID
Total products in each order: 1 1
Total products in each order: 2 1

Queries executed successfully.
```

Option 6 just exits the program.



```
Enter choice [1-6]: 6
Exiting...
```

Relational Algebra for Queries

1. Retrieve User Information by Username:

SELECT * FROM Users WHERE Username = 'john_doe';

$\pi_{UserID,Username,UserEmail,UserPassword}(\sigma_{Username='john\_doe'}(Users))$

2. List Products in a Specific Category:

SELECT * FROM Products WHERE CategoryID = 1;

$\sigma_{CategoryID=1}(Products)$

3. Find Orders Shipped on a Specific Date:

SELECT * FROM Orders WHERE OrderDate = '2023-09-13';

$\sigma_{OrderDate='2023-09-13'}(Orders)$

4. Retrieve All Payment Methods for an Order:

SELECT PaymentMethod FROM Payments WHERE OrderID = 1;

$\pi_{PaymentMethod}(\sigma_{OrderID=1}(Payments))$

5. List Products and Their Categories:

SELECT Products.ProductID, Products.ProductNumber, Products.ProductPrice, Categories.CategoryName
FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;

$\pi_{Products.ProductID,Products.ProductNumber,Products.ProductPrice,Categories.CategoryName}(\sigma_{Products.CategoryID=Categories.CategoryID}(Products \bowtie Categories))$

6. Find All User Roles for a Specific User:

SELECT UserRoles.RoleName
FROM UserRoles
INNER JOIN User_UserRoles ON UserRoles.RoleID = User_UserRoles.RoleID
WHERE User_UserRoles.UserID = 1;

$\pi_{UserRoles.RoleName}(\sigma_{User_UserRoles.UserID=1}(UserRoles \bowtie User\_UserRoles))$

7. Calculate Total Cost of Items in a Shopping Cart:

SELECT SUM(Products.ProductPrice * CartItems.CartQuantity) AS TotalCost
FROM CartItems
INNER JOIN Products ON CartItems.ProductID = Products.ProductID
WHERE CartItems.CartID = 1;

$\rho_{TotalCost}(\gamma_{SUM(Products.ProductPrice \times CartItems.CartQuantity)}(\sigma_{CartItems.CartID=1}(CartItems \bowtie Products)), TotalCost)$

8. List Shipping Details for an Order:

SELECT Shipping.ShippingLabel, Shipping.ShippingCost, ShippingCarriers.CarrierName
FROM Shipping
INNER JOIN ShippingCarriers ON Shipping.ShippingCarrierID = ShippingCarriers.CarrierID
WHERE Shipping.OrderID = 1;

$\pi_{Shipping.ShippingLabel, Shipping.ShippingCost, ShippingCarriers.CarrierName}(\sigma_{Shipping.OrderID=1}(Shipping \bowtie ShippingCarriers))$

9. Retrieve All Reviews for a Product:

SELECT ReviewText, Rating
FROM ProductReviews
WHERE ProductID = 1;

$\pi_{ReviewText, Rating}(\sigma_{ProductID=1}(\text{ProductReviews}))$

10. List All Users Who Have Placed Orders:

SELECT DISTINCT Users.Username
FROM Users
INNER JOIN Orders ON Users.UserID = Orders.UserID;

$\pi_{Users.Username}(\sigma_{Users.UserID=Orders.UserID}(Users \bowtie Orders))$

11. Find Orders with a Specific Status:

SELECT OrderID, OrderDate
FROM Orders
WHERE OrderStatus = 'Shipped';

$\pi_{OrderID, OrderDate}(\sigma_{OrderStatus='Shipped'}(\text{Orders}))$

12. Calculate Total Revenue for a Specific Category:

SELECT SUM(Products.ProductPrice * OrderItems.OrderQuantity) AS TotalRevenue
FROM Products
INNER JOIN OrderItems ON Products.ProductID = OrderItems.ProductID
WHERE Products.CategoryID = 1;

$\rho_{TotalRevenue}(\gamma_{SUM(Products.ProductPrice \times OrderItems.OrderQuantity)}(\sigma_{Products.CategoryID=1}(Products \bowtie OrderItems)), TotalRevenue)$

13. List Users with Their Active Shopping Carts:

SELECT Users.Username, ShoppingCarts.CartID
FROM Users
LEFT JOIN ShoppingCarts ON Users.UserID = ShoppingCarts.UserID;

$\pi_{Users.Username, ShoppingCarts.CartID}(\rho_{ActiveCarts}(Users \bowtie (\pi_{CartID}(ShoppingCarts))), ActiveCarts)$

14. Find Average Rating of Each Product:

```
SELECT ProductID, AVG(Rating) AS AvgRating
FROM ProductReviews
GROUP BY ProductID;
```

$\gamma_{AVG(Rating)\rightarrow AvgRating}(ProductReviews)$

15. Find Orders Placed by a Specific User:

```
SELECT OrderID, OrderDate, OrderStatus
FROM Orders
WHERE UserID = 1;
```

$\pi_{OrderID,OrderDate,OrderStatus}(\sigma_{UserID=1}(Orders))$

16. List All Unique Search Queries Used:

```
SELECT DISTINCT QueryText
FROM SearchQueries;
```

$\gamma_{DISTINCT(QueryText)}(SearchQueries)$

17. Find Users that Reviewed a Products they Ordered:

```
SELECT DISTINCT u.Username
FROM Users u
WHERE EXISTS (
SELECT 1
FROM Orders o
WHERE o.UserID = u.UserID
AND EXISTS (
SELECT 1
FROM ProductReviews pr
WHERE pr.ProductID IN (
SELECT ProductID
FROM OrderItems oi
WHERE oi.OrderID = o.OrderID
)
)
);
```

$\pi_{u.Username}(Users \bowtie \sigma_{\exists o(\exists pr(\exists oi(o.UserID=u.UserID \wedge o.OrderID=oi.OrderID \wedge oi.ProductID=pr.ProductID)))}(u))$

18. Find Users without Admin Roles:

```
SELECT Username
FROM Users u
WHERE NOT EXISTS (
SELECT 1
FROM User_UserRoles ur
```

WHERE ur.UserID = u.UserID
AND ur.RoleID = 2
);
$\pi_{Username}(\sigma_{\neg\exists ur(ur.RoleID=2)}(Users \bowtie User\_UserRoles))$

19. Get distinct UserEmails from Users and User_UserRoles tables:
SELECT UserEmail FROM Users
UNION
SELECT UserID FROM User_UserRoles;
$\gamma_{DISTINCT(UserEmail)}(Users) \cup \gamma_{DISTINCT(UserID)}(User\_UserRoles)$

20. Calculate the total number of products in each order:
SELECT Orders.OrderID, COUNT(OrderItems.ProductID) AS TotalProducts
FROM Orders
LEFT JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
GROUP BY Orders.OrderID;
$\gamma_{COUNT(OrderItems.ProductID)\rightarrow TotalProducts}(\sigma_{Orders.OrderID=OrderItems.OrderID}(Orders \ltimes OrderItems))$