CODING CHALLENGE

ORDER MANAGEMENT SYSTEM

Problem Statement:

Create SQL Schema from the product and user class, use the class attributes for table column names.

1.Create a base class called Product with the following attributes:

- productId (int)
- productName (String)
- description (String)
- price (double)
- quantityInStock (int)
- type (String) [Electronics/Clothing]

2. Implement constructors, getters, and setters for the Product class.

3. Create a subclass Electronics that inherits from Product. Add attributes specific to electronics products, such as:

- brand (String)
- warrantyPeriod (int)

4. Create a subclass Clothing that also inherits from Product. Add attributes specific to clothing products, such as:

- size (String)
- color (String)

5. Create a User class with attributes:

- userId (int)

• username (String)

• password (String)

• role (String) // "Admin" or "User"

6. Define an interface/abstract class named

IOrderManagementRepository with methods for:

• createOrder(User user, list of products): check the user as already present in database to create order or create user (store in database) and create order.

• cancelOrder(int userId, int orderId): check the userid and orderId already present in database and cancel the order. if any userId or orderId not present in database throw exception corresponding UserNotFound or OrderNotFound exception

• createProduct(User user, Product product): check the admin user as already present in database and create product and store in database.

• createUser(User user): create user and store in database for further development.

• getAllProducts(): return all product list from the database.

• getOrderByUser(User user): return all product ordered by specific user from database.

7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.

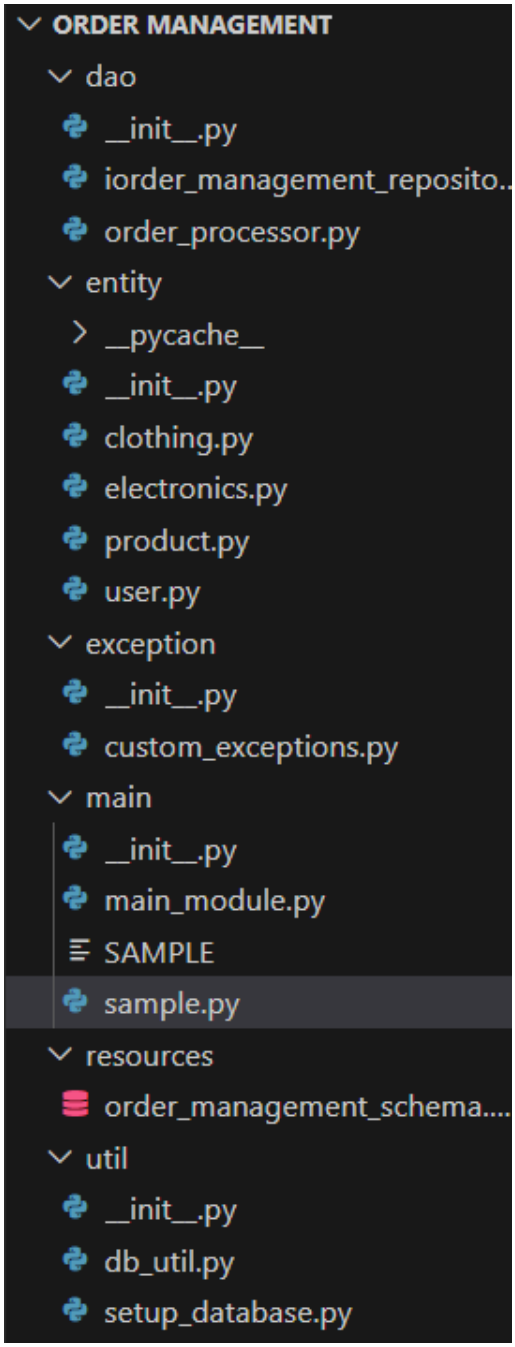
8. Create DBUtil class and add the following method.

• static getDBConn():Connection Establish a connection to the database and return database Connection

9. Create OrderManagement main class and perform following operation:

• main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct", "cancelOrder", "getAllProducts", "getOrderbyUser", "exit".

FOLDERS:



ENTITY:

CLOTHING:

```python
from entity.product import Product

class Clothing(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.size = size
        self.color = color

    def get_size(self):
        return self.size

    def set_size(self, size):
        self.size = size

    def get_color(self):
        return self.color

    def set_color(self, color):
        self.color = color
```

ELECTRONICS:

```python
from entity.product import Product

class Electronics(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, brand, warrantyPeriod):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.brand = brand
        self.warrantyPeriod = warrantyPeriod

    def get_brand(self):
        return self.brand

    def set_brand(self, brand):
        self.brand = brand

    def get_warranty_period(self):
        return self.warrantyPeriod

    def set_warranty_period(self, warrantyPeriod):
        self.warrantyPeriod = warrantyPeriod
```

PRODUCT:

```python
class Product:
    def __init__(self, productId, productName, description, price, quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.type = type

    def get_product_id(self):
        return self.productId

    def set_product_id(self, productId):
        self.productId = productId
```

```python
def get_product_name(self):
    return self.productName

def set_product_name(self, productName):
    self.productName = productName

def get_description(self):
    return self.description

def set_description(self, description):
    self.description = description

def get_price(self):
    return self.price

def set_price(self, price):
    self.price = price

def get_quantity_in_stock(self):
    return self.quantityInStock

def set_quantity_in_stock(self, quantity):
    self.quantityInStock = quantity

def get_type(self):
    return self.type

def set_type(self, type):
    self.type = type
```

USER:

```python
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        self.role = role  # "Admin" or "User"

    def get_user_id(self):
        return self.userId

    def set_user_id(se (parameter) userId: Any
        self.userId = userId

    def get_username(self):
        return self.username

    def set_username(self, username):
        self.username = username

    def get_password(self):
        return self.password

    def set_password(self, password):
        self.password = password

    def get_role(self):
        return self.role

    def set_role(self, role):
        self.role = role
```

EXCEPTIONS:

```python
class UserNotFoundException(Exception):
    def __init__(self, message="User not found in the database."):
        super().__init__(message)

class OrderNotFoundException(Exception):
    def __init__(self, message="Order not found in the database."):
        super().__init__(message)

class UnauthorizedException(Exception):
    def __init__(self, message="User is not authorized to perform this operation."):
        super().__init__(message)

class ProductNotFoundException(Exception):
    def __init__(self, message="Product not found in the database."):
        super().__init__(message)
```

DAO:

IORDER MANAGEMENT:

dao > iorder_management_repository.py > IOrderManagementRepository >

```python
from abc import ABC, abstractmethod

class IOrderManagementRepository(ABC):

    @abstractmethod
    def createUser(self, user):
        pass

    @abstractmethod
    def createProduct(self, user, product):
        pass

    @abstractmethod
    def createOrder(self, user, product_list):
        pass

    @abstractmethod
    def cancelOrder(self, userId, orderId):
        pass

    @abstractmethod
    def getAllProducts(self):
        pass

    @abstractmethod
    def getOrderByUser(self, user):
        pass
```

ORDER PROCESSOR:

dao > order_processor.py > OrderProcessor

```python
import mysql.connector
from dao.iorder_management_repository import IOrderManagementRepository
from util.db_util import DBUtil
from exception.custom_exceptions import UserNotFoundException, OrderNotFoundException
from entity.electronics import Electronics
from entity.clothing import Clothing
from entity.product import Product

class OrderProcessor(IOrderManagementRepository):
    def __init__(self):
        self.connection = DBUtil.getDBConn()

    def createUser(self, user):
        try:
            cursor = self.connection.cursor()
            cursor.execute("INSERT INTO users (userId, username, password, role) VALUES (%s, %s, %s, %s)",
                           (user.userId, user.username, user.password, user.role))
            self.connection.commit()
            print("User created successfully.")
        except mysql.connector.IntegrityError:
            print("User already exists.")
        finally:
            cursor.close()

    def createProduct(self, user, product):
        try:
            cursor = self.connection.cursor()

            cursor.execute("SELECT role FROM users WHERE userId = %s", (user.userId,))
            result = cursor.fetchone()

            if not result:
                raise UserNotFoundException("User not found.")

            role = result[0]
            if role.lower() != "admin":
                print("Only admin users can add products.")
                return

            cursor.execute("""
                INSERT INTO products
                (productId, productName, description, price, quantityInStock, type, brand, warrantyPeriod, size, color)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """, (
                    product.productId,
                    product.productName,
                    product.description,
                    product.price,
                    product.quantityInStock,
                    product.type,
                    getattr(product, 'brand', None),
                    getattr(product, 'warrantyPeriod', None),
                    getattr(product, 'size', None),
                    getattr(product, 'color', None)
                ))

            self.connection.commit()
            print("Product created successfully.")

        except UserNotFoundException as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def createOrder(self, user, product_list):
        try:
            cursor = self.connection.cursor()

            cursor.execute("SELECT * FROM users WHERE userId = %s", (user.userId,))
            existing_user = cursor.fetchone()
```

```
            if not existing_user:
                cursor.execute(
                    "INSERT INTO users (userId, username, password, role) VALUES (%s, %s, %s, %s)",
                    (user.userId, user.username, user.password, user.role)
                )
                print(f"User with ID {user.userId} created.")

            cursor.execute("INSERT INTO orders (userId) VALUES (%s)", (user.userId,))
            self.connection.commit()

            order_id = cursor.lastrowid
            print(f"Order created with ID: {order_id}")

            for product in product_list:
                cursor.execute("SELECT quantityInStock FROM products WHERE productId = %s", (product.productId,))
                result = cursor.fetchone()

                if not result:
                    print(f"Product ID {product.productId} not found. Skipping.")
                    continue

                stock_available = result[0]
                if stock_available < product.quantityInStock:
                    print(f"Insufficient stock for product ID {product.productId}. Skipping.")
                    continue

                cursor.execute(
                    "INSERT INTO order_items (orderId, productId, quantity) VALUES (%s, %s, %s)",
                    (order_id, product.productId, product.quantityInStock)
                )

                cursor.execute(
                    "UPDATE products SET quantityInStock = quantityInStock - %s WHERE productId = %s",
                    (product.quantityInStock, product.productId)
                )

            self.connection.commit()
            print("Order placed successfully.")

        except Exception as e:
            print(f"Error in createOrder: {e}")
            self.connection.rollback()
```

```
        finally:
            cursor.close()

    def cancelOrder(self, userId, orderId):
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM users WHERE userId = %s", (userId,))
            if not cursor.fetchone():
                raise UserNotFoundException(f"User with ID {userId} not found.")

            cursor.execute("SELECT * FROM orders WHERE orderId = %s AND userId = %s", (orderId, userId))
            if not cursor.fetchone():
                raise OrderNotFoundException(f"Order ID {orderId} not found for User ID {userId}.")

            cursor.execute("DELETE FROM order_items WHERE orderId = %s", (orderId,))
            cursor.execute("DELETE FROM orders WHERE orderId = %s", (orderId,))
            self.connection.commit()

            print(f"Order ID {orderId} cancelled successfully.")

        except (UserNotFoundException, OrderNotFoundException) as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def getAllProducts(self):
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM products")
            rows = cursor.fetchall()

            product_list = []
            for row in rows:
                pid, name, desc, price, qty, type, brand, warranty, size, color = row
                if type.lower() == "electronics":
                    product = Electronics(pid, name, desc, price, qty, type, brand, warranty)
                elif type.lower() == "clothing":
                    product = Clothing(pid, name, desc, price, qty, type, size, color)
                else:
                    product = Product(pid, name, desc, price, qty, type)
                product_list.append(product)
```

```
            return product_list

        except Exception as e:
            print(f"Error fetching products: {e}")
            return []
        finally:
            cursor.close()

    def getOrderByUser(self, user):
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT orderId FROM orders WHERE userId = %s", (user.userId,))
            orders = cursor.fetchall()

            if not orders:
                raise OrderNotFoundException("No orders found for the user.")

            result = []
            for (order_id,) in orders:
                cursor.execute("""
                    SELECT p.productName, oi.quantity
                    FROM order_items oi
                    JOIN products p ON oi.productId = p.productId
                    WHERE oi.orderId = %s
                """, (order_id,))
                items = cursor.fetchall()
                result.append({
                    "orderId": order_id,
                    "items": [{"product": i[0], "quantity": i[1]} for i in items]
                })

            return result

        except OrderNotFoundException as e:
            print(f"Error: {e}")
            return []
        finally:
            cursor.close()
```

MAIN:

```python
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from entity.user import User
from entity.electronics import Electronics
from entity.clothing import Clothing
from dao.order_processor import OrderProcessor

def main():
    service = OrderProcessor()

    while True:
        print("\n--- Order Management System ---")
        print("1. Create User")
        print("2. Create Product")
        print("3. Create Order")
        print("4. Cancel Order")
        print("5. Get All Products")
        print("6. Get Orders By User")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")

        if choice == '1':
            userId = int(input("User ID: "))
            username = input("Username: ")
            password = input("Password: ")
            role = input("Role (Admin/User): ")
            user = User(userId, username, password, role)
            service.createUser(user)

        elif choice == '2':
            userId = int(input("Enter Admin User ID: "))
            admin = User(userId, None, None, "Admin")

            productId = int(input("Product ID: "))
            productName = input("Name: ")
            description = input("Description: ")
            price = float(input("Price: "))
            quantity = int(input("Quantity: "))
            productType = input("Type (Electronics/Clothing): ")
```

```python
            if productType.lower() == "electronics":
                brand = input("Brand: ")
                warranty = int(input("Warranty Period (months): "))
                product = Electronics(productId, productName, description, price, quantity, productType, brand, warranty)
            else:
                size = input("Size: ")
                color = input("Color: ")
                product = Clothing(productId, productName, description, price, quantity, productType, size, color)

            service.createProduct(admin, product)

        elif choice == '3':
            userId = int(input("User ID: "))
            username = input("Username: ")
            password = input("Password: ")
            role = "User"
            user = User(userId, username, password, role)

```

```
            num_products = int(input("How many products to order? "))
            product_list = []
            for _ in range(num_products):
                pid = int(input("Product ID: "))
                qty = int(input("Quantity: "))
                # For simplicity, assume Electronics; in real case, you'd fetch full product from DB
                dummy_product = Electronics(pid, "", "", 0.0, qty, "Electronics", "", 0)
                product_list.append(dummy_product)

            service.createOrder(user, product_list)

        elif choice == '4':
            userId = int(input("Enter User ID: "))
            orderId = int(input("Enter Order ID: "))
            service.cancelOrder(userId, orderId)

        elif choice == '5':
            products = service.getAllProducts()
            for p in products:
                print(vars(p))

        elif choice == '6':
            userId = int(input("User ID: "))
            username = input("Username: ")
            password = input("Password: ")
            user = User(userId, username, password, "User")
            orders = service.getOrderByUser(user)
            for order in orders:
                print(order)

        elif choice == '7':
            print("Exiting Order Management System.")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 7.")

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

DB UTIL:

```
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        try:
            connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="your_password",
                database="order_management"
            )
            return connection
        except mysql.connector.Error as err:
            print("Database connection failed:", err)
            return None
```

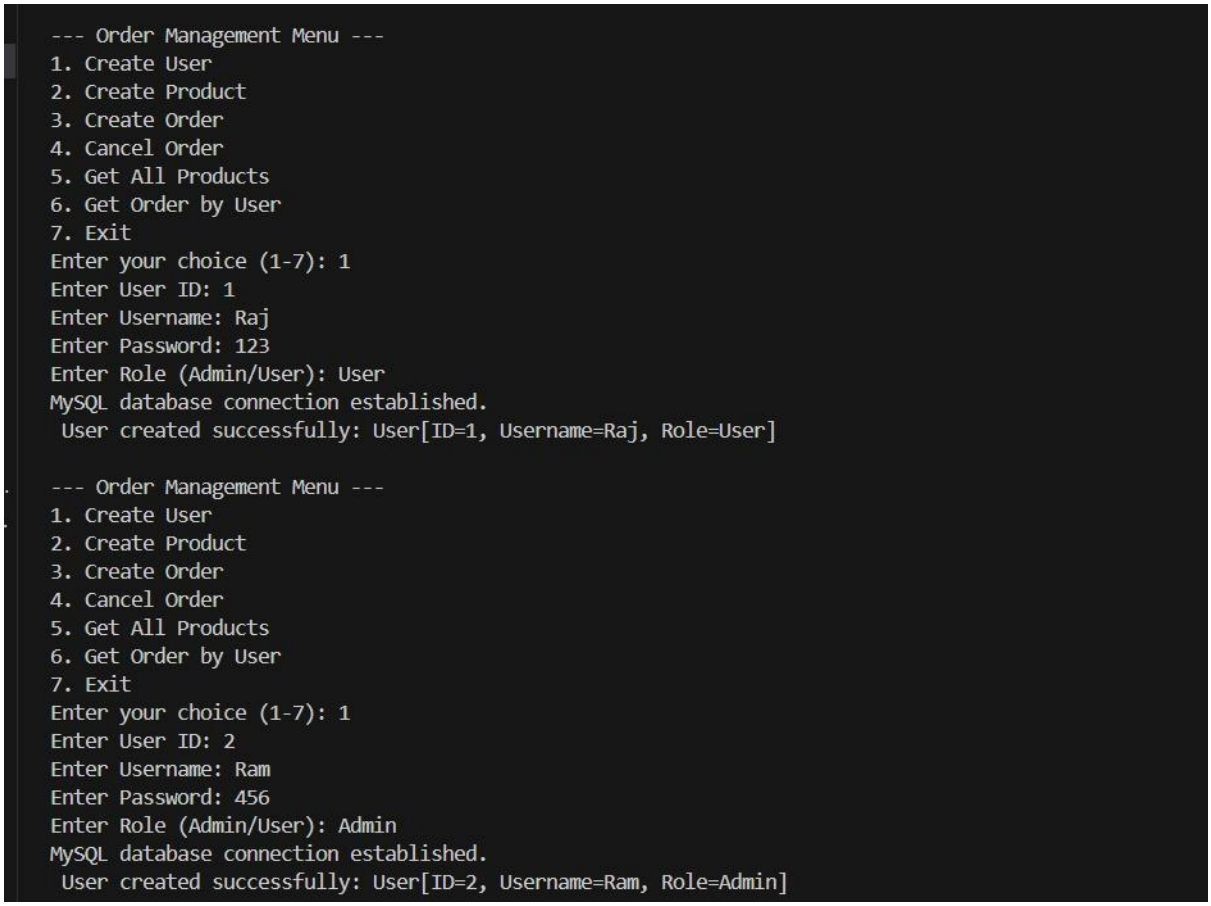
SETUP DATABASE:

```python
import mysql.connector

def setup_database():
    try:
        # Initial connection without specifying database
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="a1a2b1b2"  # Replace with your actual password
        )

        cursor = connection.cursor()

        # Create database
        cursor.execute("CREATE DATABASE IF NOT EXISTS order_management")
        cursor.execute("USE order_management")

        # Create tables
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                userId INT PRIMARY KEY,
                username VARCHAR(50) NOT NULL,
                password VARCHAR(50) NOT NULL,
                role VARCHAR(10) NOT NULL CHECK (role IN ('Admin', 'User'))
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS products (
                productId INT PRIMARY KEY,
                productName VARCHAR(100) NOT NULL,
                description TEXT,
                price DOUBLE NOT NULL,
                quantityInStock INT NOT NULL,
                type VARCHAR(20) NOT NULL,
                brand VARCHAR(50),
                warrantyPeriod INT,
                size VARCHAR(10),
                color VARCHAR(30)
            )
        """)
```

```
42
43          cursor.execute("""
44              CREATE TABLE IF NOT EXISTS orders (
45                  orderId INT AUTO_INCREMENT PRIMARY KEY,
46                  userId INT NOT NULL,
47                  orderDate DATETIME DEFAULT CURRENT_TIMESTAMP,
48                  FOREIGN KEY (userId) REFERENCES users(userId)
49              )
50          """)
51
52          cursor.execute("""
53              CREATE TABLE IF NOT EXISTS order_items (
54                  orderItemId INT AUTO_INCREMENT PRIMARY KEY,
55                  orderId INT NOT NULL,
56                  productId INT NOT NULL,
57                  quantity INT NOT NULL,
58                  FOREIGN KEY (orderId) REFERENCES orders(orderId),
59                  FOREIGN KEY (productId) REFERENCES products(productId)
60              )
61          """)
62
63          connection.commit()
64          print(" Database and tables created successfully.")
65
66      except mysql.connector.Error as err:
67          print(" Error while creating database or tables:", err)
68
69      finally:
70          cursor.close()
71          connection.close()
72
73  # Run the function
74  if __name__ == "__main__":
75      setup_database()
```

OUTPUT SCREENSHOTS

1 CREATE USER:



2.CREATING PRODUCTS:

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 2
Enter User ID: 2
Enter Username: Ram
Enter Password: 456
Enter Role (Admin/User): Admin
Enter Product ID: 11
Enter Product Name: Shirt
Enter Description: Pure Cotton Shirt
Enter Price: 2000
Enter Quantity in Stock: 156
Enter Product Type (Electronics/Clothing): Clothing
Enter Size: 36
Enter Color: Blue
MySQL database connection established.
 Product inserted successfully.
```

3.CREATING ORDERS:

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 3
Enter User ID: 1
Enter Username: Raj
Enter Password: 123
Enter Role (Admin/User): User
Enter Product IDs (comma-separated): 11
MySQL database connection established.
Order created for user Raj with Order ID: 3
```

4.GET ALL PRODUCTS:

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 5
MySQL database connection established.
(11, 'Shirt', 'Pure Cotton Shirt', 2000.0, 156, 'Clothing', None, None, '36', 'Blue')
```

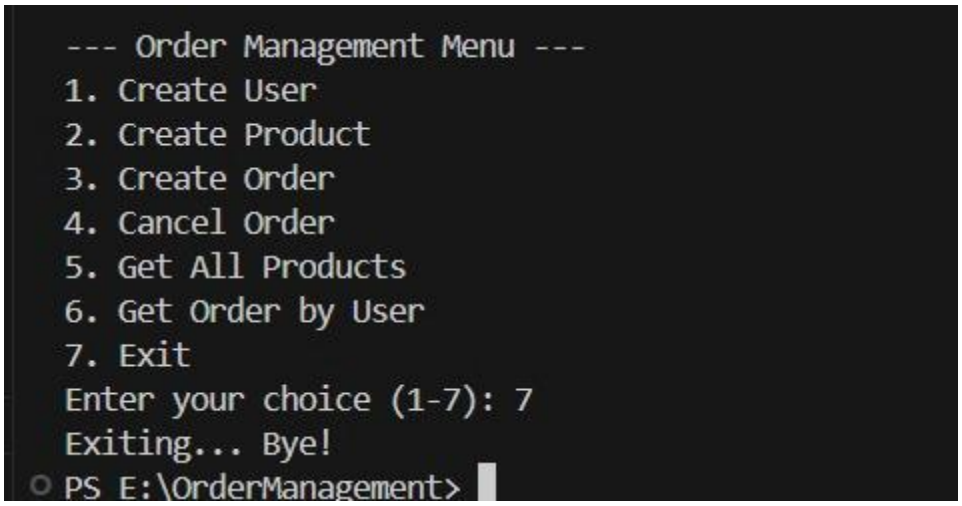
5.GET ORDER BY USER:

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 6
Enter User ID: 1
Enter Username: Raj
Enter Password: 123
Enter Role: User
MySQL database connection established.
(3, 11, 'Shirt', 'Pure Cotton Shirt', 2000.0, 'Clothing')
```

6.CANCEL ORDER:

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 4
Enter User ID: 1
Enter Order ID: 3
MySQL database connection established.
Order 3 cancelled successfully.
```

7.EXIT:


--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 7
Exiting... Bye!
PS E:\OrderManagement>