HEXAWARE

CASE STUDY
ON
ECOMMERCE APPLICATION

TEAM MEMBERS

MATHEW P

HARSHA K

ABSTRACT:

In the age of digital commerce and growing demand for seamless shopping experiences, ZenCart offers a modular, database-driven e-commerce application built using Python and MySQL. This project mimics the core functionalities of a real-world online retail platform while adhering to best practices in software design, layered architecture, and data integrity. It operates through a clean command-line interface, enabling both admin and customer roles with guided, role-based workflows.

The project structure follows a strict modular design:

- **Entity Layer** (entity): Models essential business objects like Customer, Product, Cart, and Order as pure data holders with private fields and getter/setter methods.
- **DAO Layer** (dao): Implements data access interfaces and logic using parameterized SQL queries with mysql-connector, promoting separation of concerns.
- **Exception Layer** (exception): Defines custom exceptions to handle application-specific errors such as CustomerNotFoundException, OutOfStockException, etc.
- **Utility Layer** (util): Manages configuration and database connection setup through reusable static methods, enhancing portability and maintainability.
- **Main Module** (main): Acts as the user interface, driving all operations via a role-based, menu-driven console interaction.

Key Features Include:

- **Customer Management**: Register, update, and view customer details and their order history.
- **Product Management**: Admins can add, update, view, or delete product listings from the catalog.
- **Cart Management**: Customers can add/remove products, with validation against stock availability.
- **Order Management**: Users can place orders with automatic calculation of totals, manage shipping details, and receive confirmations.

The application also integrates **exception handling, modular code reuse**, and **unit testing with Pytest** to ensure robust functionality across customer flows and database operations. ZenCart serves as a compact proof-of-concept for backend

e-commerce systems, demonstrating practical application of OOP principles, database operations, and clean architecture in Python.

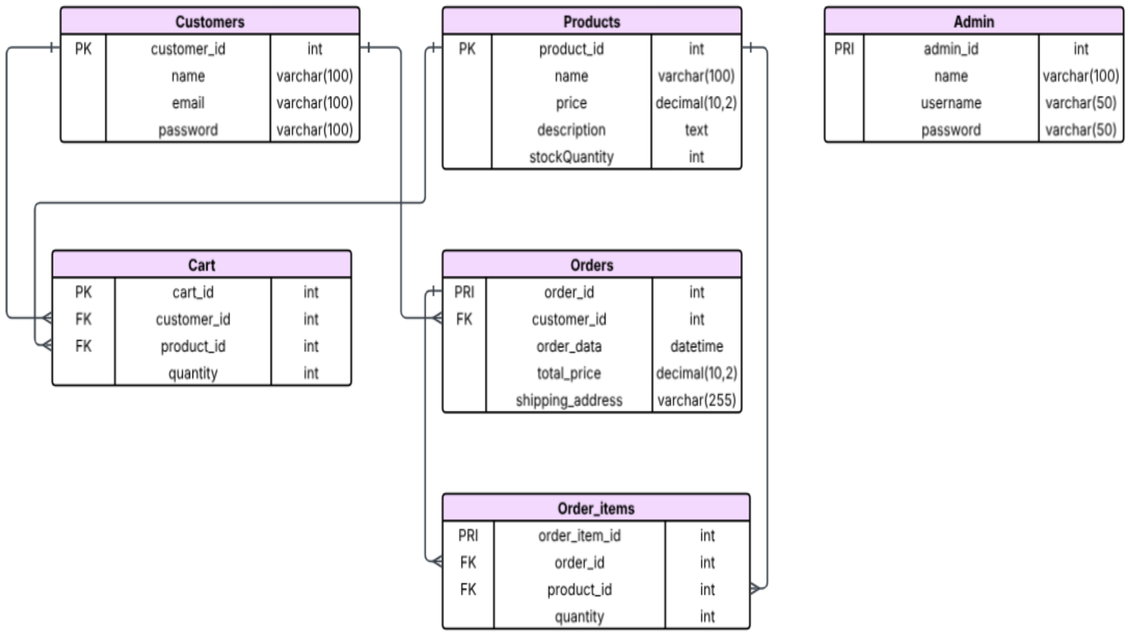
INTRODUCTION:

The ZenCart E-commerce Console Application simulates an online shopping experience with a focus on modular software design, efficient database operations, and a clean user experience through CLI. It was developed to reflect the core operations of an online store, such as managing customers, products, carts, and orders.

The objective of the project is to provide a layered and scalable backend system that can support the essential functions of an e-commerce platform, while following software engineering best practices like interface-based design, exception management, and proper encapsulatioPn.

With a clear separation of concerns and testable modules, ZenCart not only provides a working prototype of a shopping system but also emphasizes backend architectural principles suited for real-world applications.

ENTITY RELATIONSHIP DIAGRAM:

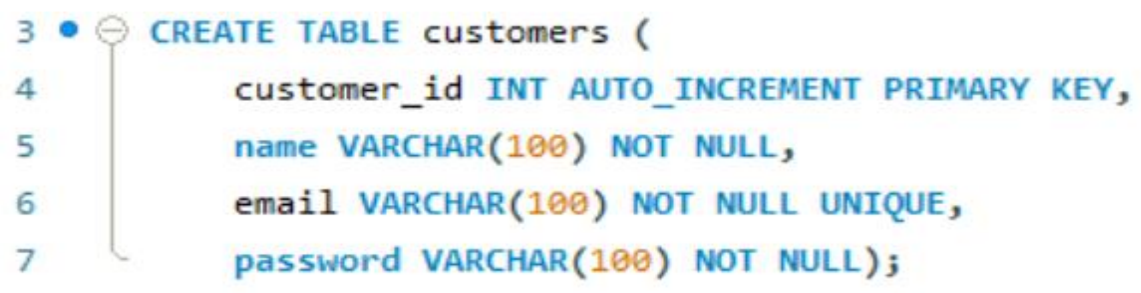


Create following tables in SQL Schema with appropriate class and write the unit test case for the Ecommerce application.

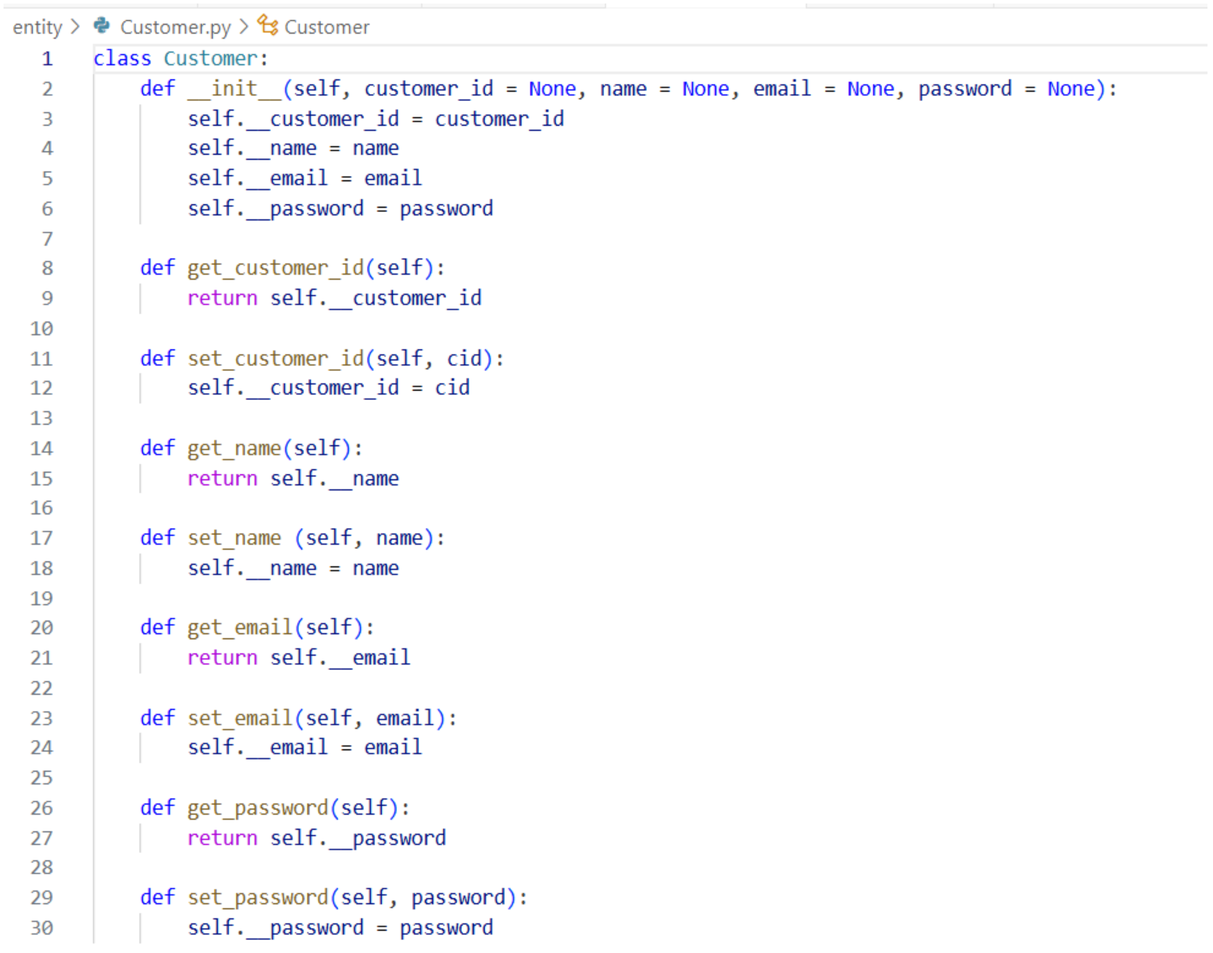
Schema Design:

1. customers table:

- customer_id (Primary Key)
- name
- email
- password



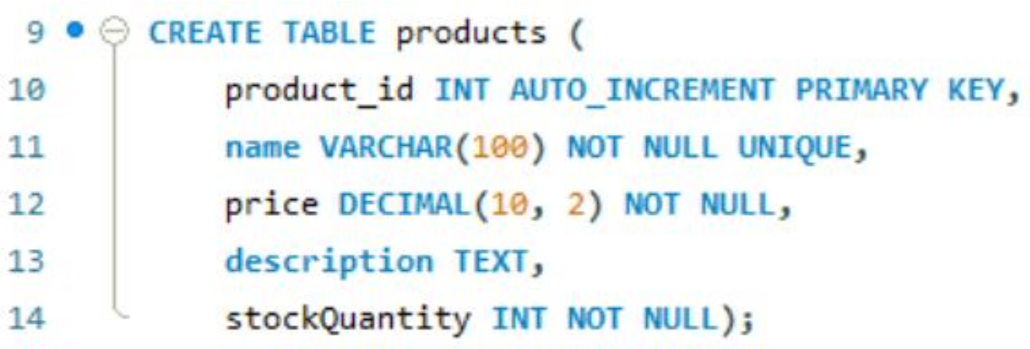
```sql
CREATE TABLE customers (
        customer_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        email VARCHAR(100) NOT NULL UNIQUE,
        password VARCHAR(100) NOT NULL);
```



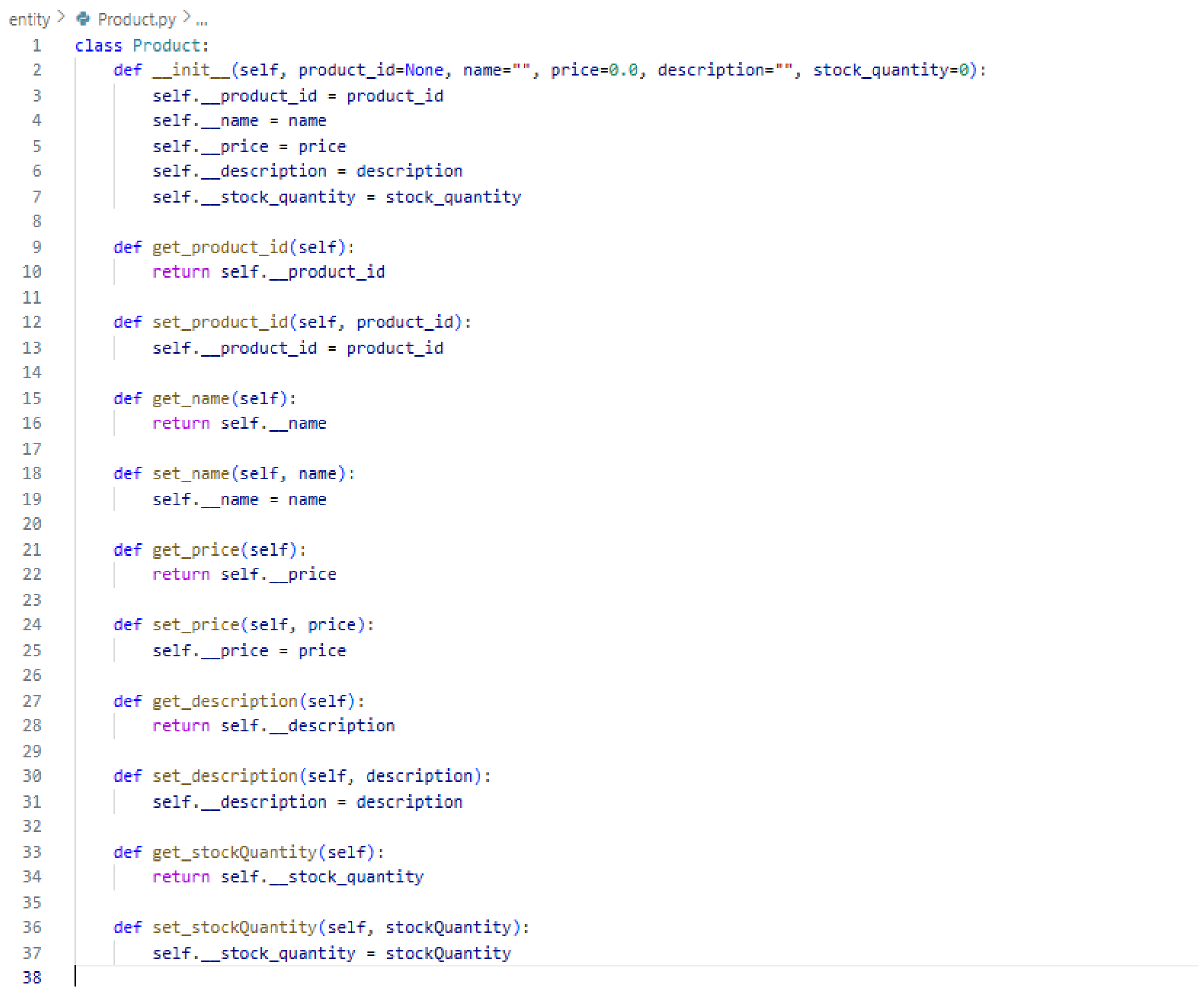
```python
class Customer:
    def __init__(self, customer_id = None, name = None, email = None, password = None):
        self.__customer_id = customer_id
        self.__name = name
        self.__email = email
        self.__password = password

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, cid):
        self.__customer_id = cid

    def get_name(self):
        return self.__name

    def set_name (self, name):
        self.__name = name

    def get_email(self):
        return self.__email

    def set_email(self, email):
        self.__email = email

    def get_password(self):
        return self.__password

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

2. products table:

- product_id (Primary Key)
- name
- price
- description
- stockQuantity



```sql
CREATE TABLE products (
        product_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL UNIQUE,
        price DECIMAL(10, 2) NOT NULL,
        description TEXT,
        stockQuantity INT NOT NULL);
```

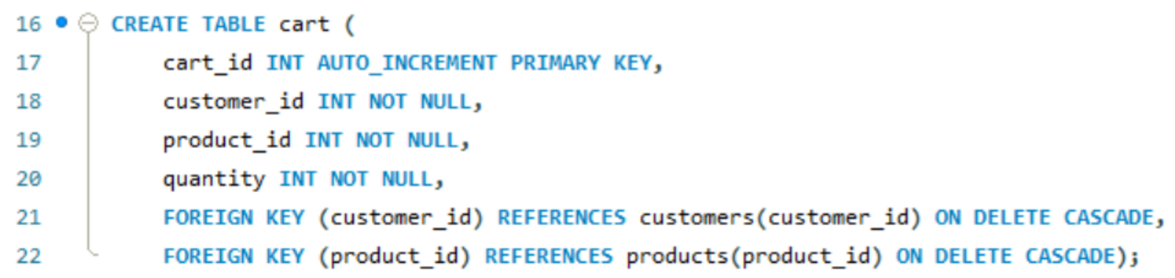


entity > Product.py > ...

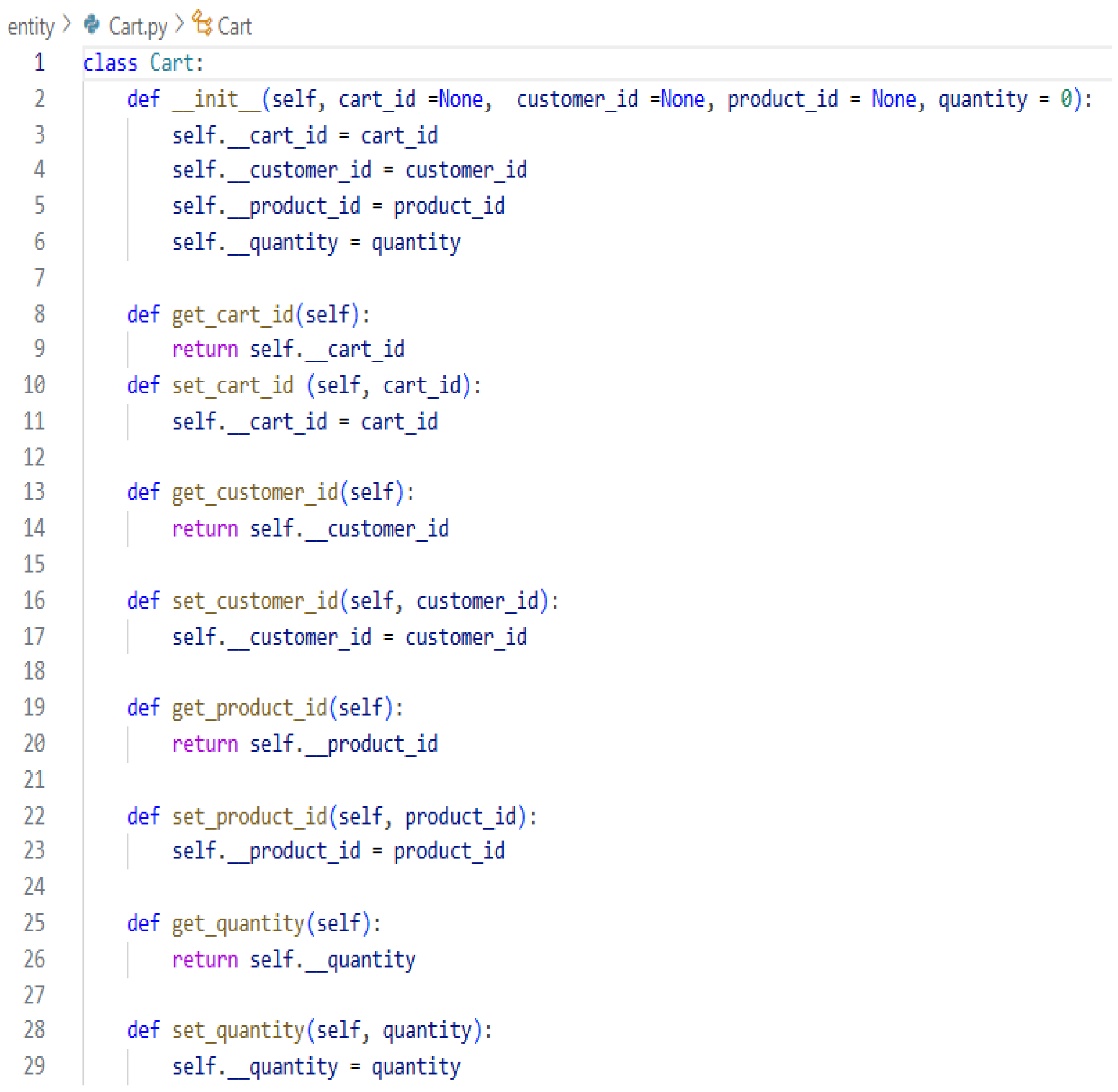
```python
class Product:
    def __init__(self, product_id=None, name="", price=0.0, description="", stock_quantity=0):
        self.__product_id = product_id
        self.__name = name
        self.__price = price
        self.__description = description
        self.__stock_quantity = stock_quantity

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    def get_price(self):
        return self.__price

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

    def get_description(self):
        return self.__description

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

    def get_stockQuantity(self):
        return self.__stock_quantity

    def set_stockQuantity(self, stockQuantity):
        self.__stock_quantity = stockQuantity

```

3. cart table:

- cart_id (Primary Key)
- customer_id (Foreign Key)
- product_id (Foreign Key)
- quantity



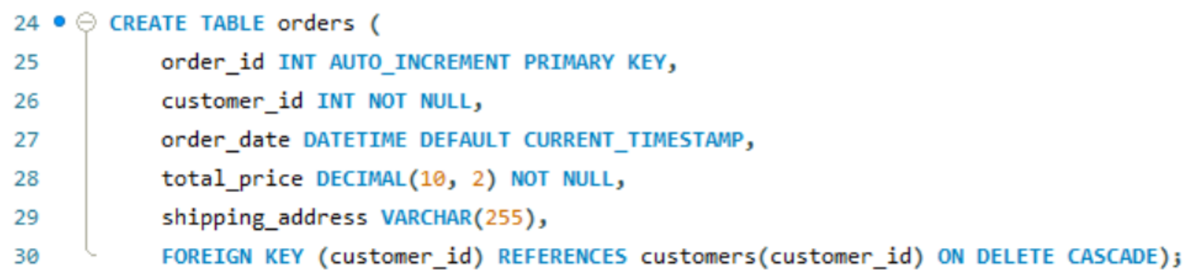
```sql
CREATE TABLE cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(product_id) ON DELETE CASCADE);
```



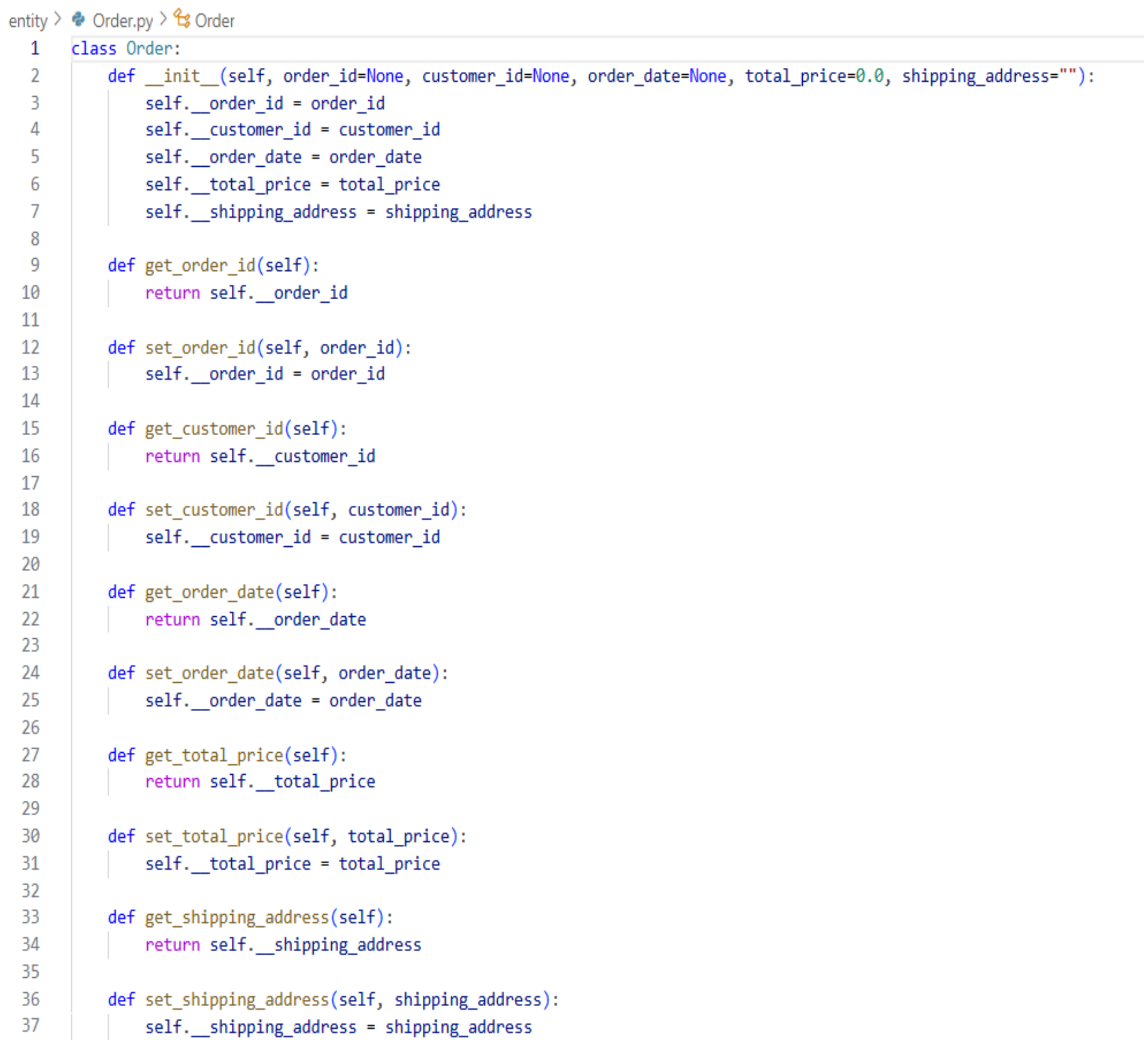
```python
class Cart:
    def __init__(self, cart_id =None,  customer_id =None, product_id = None, quantity = 0):
        self.__cart_id = cart_id
        self.__customer_id = customer_id
        self.__product_id = product_id
        self.__quantity = quantity

    def get_cart_id(self):
        return self.__cart_id
    def set_cart_id (self, cart_id):
        self.__cart_id = cart_id

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_quantity(self):
        return self.__quantity

    def set_quantity(self, quantity):
        self.__quantity = quantity
```

4. orders table:

- order_id (Primary Key)
- customer_id (Foreign Key)
- order_date
- total_price
- shipping_address



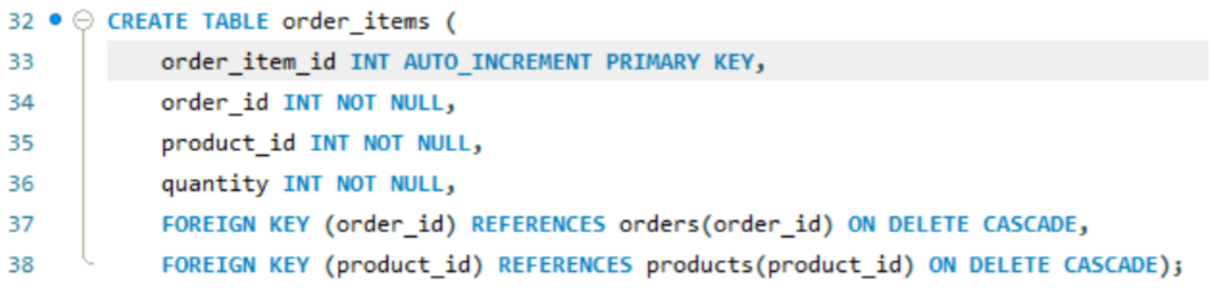
```sql
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    total_price DECIMAL(10, 2) NOT NULL,
    shipping_address VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE);
```



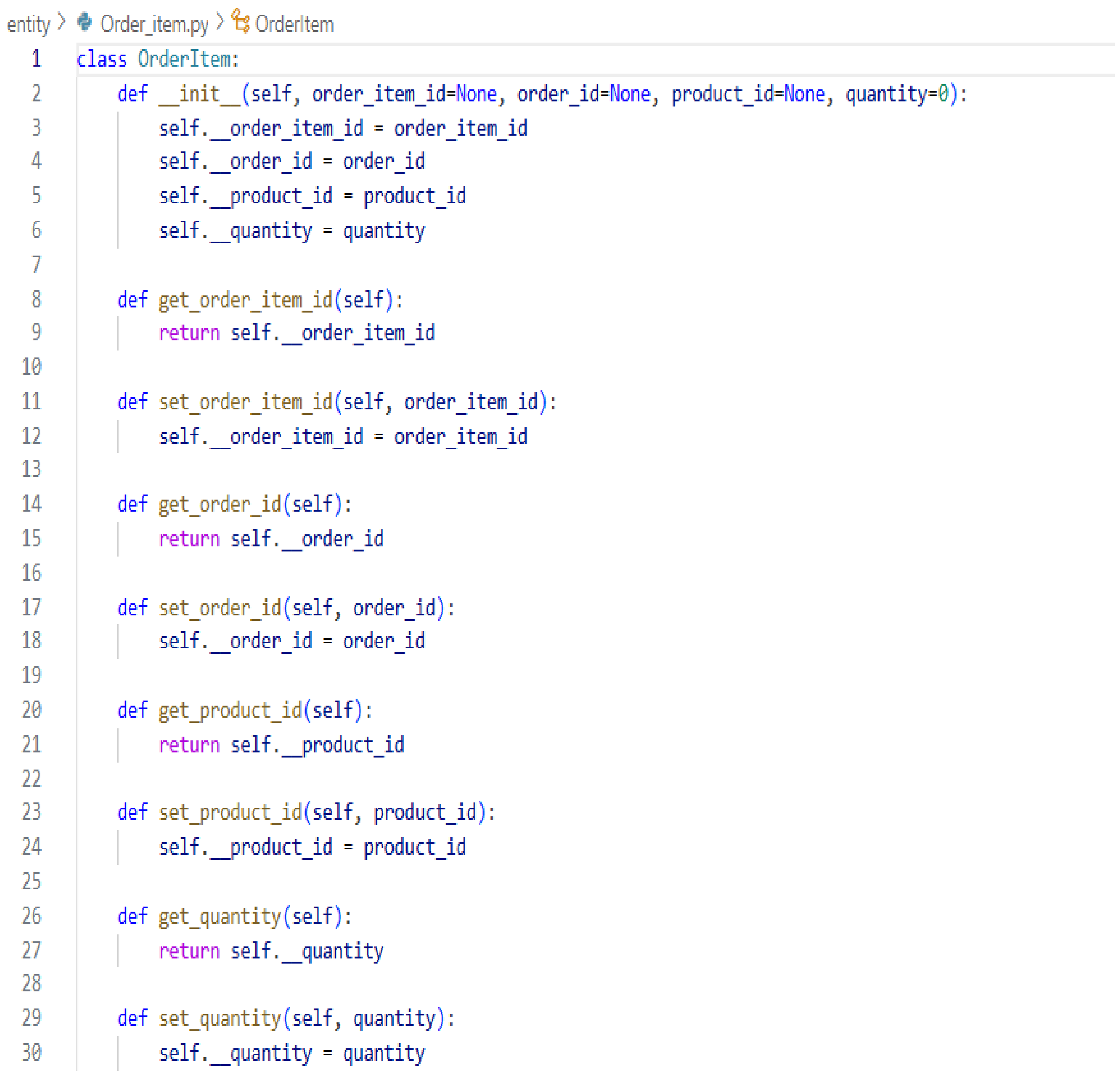
```python
class Order:
    def __init__(self, order_id=None, customer_id=None, order_date=None, total_price=0.0, shipping_address=""):
        self.__order_id = order_id
        self.__customer_id = customer_id
        self.__order_date = order_date
        self.__total_price = total_price
        self.__shipping_address = shipping_address

    def get_order_id(self):
        return self.__order_id

    def set_order_id(self, order_id):
        self.__order_id = order_id

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_order_date(self):
        return self.__order_date

    def set_order_date(self, order_date):
        self.__order_date = order_date

    def get_total_price(self):
        return self.__total_price

    def set_total_price(self, total_price):
        self.__total_price = total_price

    def get_shipping_address(self):
        return self.__shipping_address

    def set_shipping_address(self, shipping_address):
        self.__shipping_address = shipping_address
```

5. order_items table:

- order_item_id (Primary Key)
- order_id (Foreign Key)
- product_id (Foreign Key)
- quantity



```sql
CREATE TABLE order_items (
    order_item_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(product_id) ON DELETE CASCADE);
```



```python
class OrderItem:
    def __init__(self, order_item_id=None, order_id=None, product_id=None, quantity=0):
        self.__order_item_id = order_item_id
        self.__order_id = order_id
        self.__product_id = product_id
        self.__quantity = quantity

    def get_order_item_id(self):
        return self.__order_item_id

    def set_order_item_id(self, order_item_id):
        self.__order_item_id = order_item_id

    def get_order_id(self):
        return self.__order_id

    def set_order_id(self, order_id):
        self.__order_id = order_id

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_quantity(self):
        return self.__quantity

    def set_quantity(self, quantity):
        self.__quantity = quantity
```

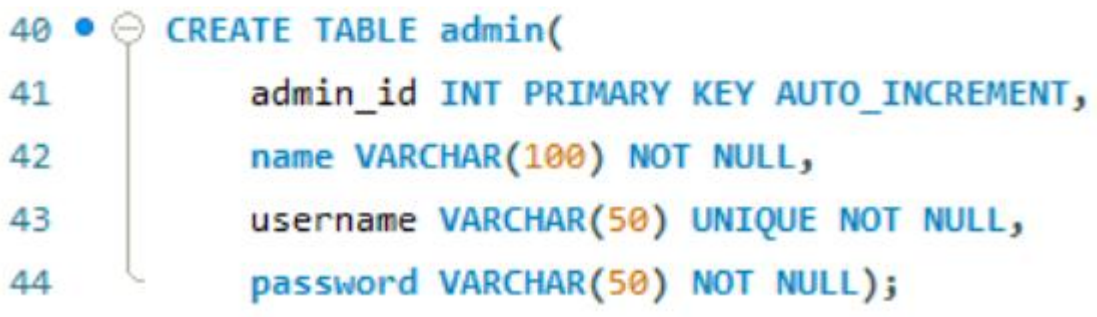
Additional **admin table** added for improved security with the following design:

• admin_id (Primary Key)

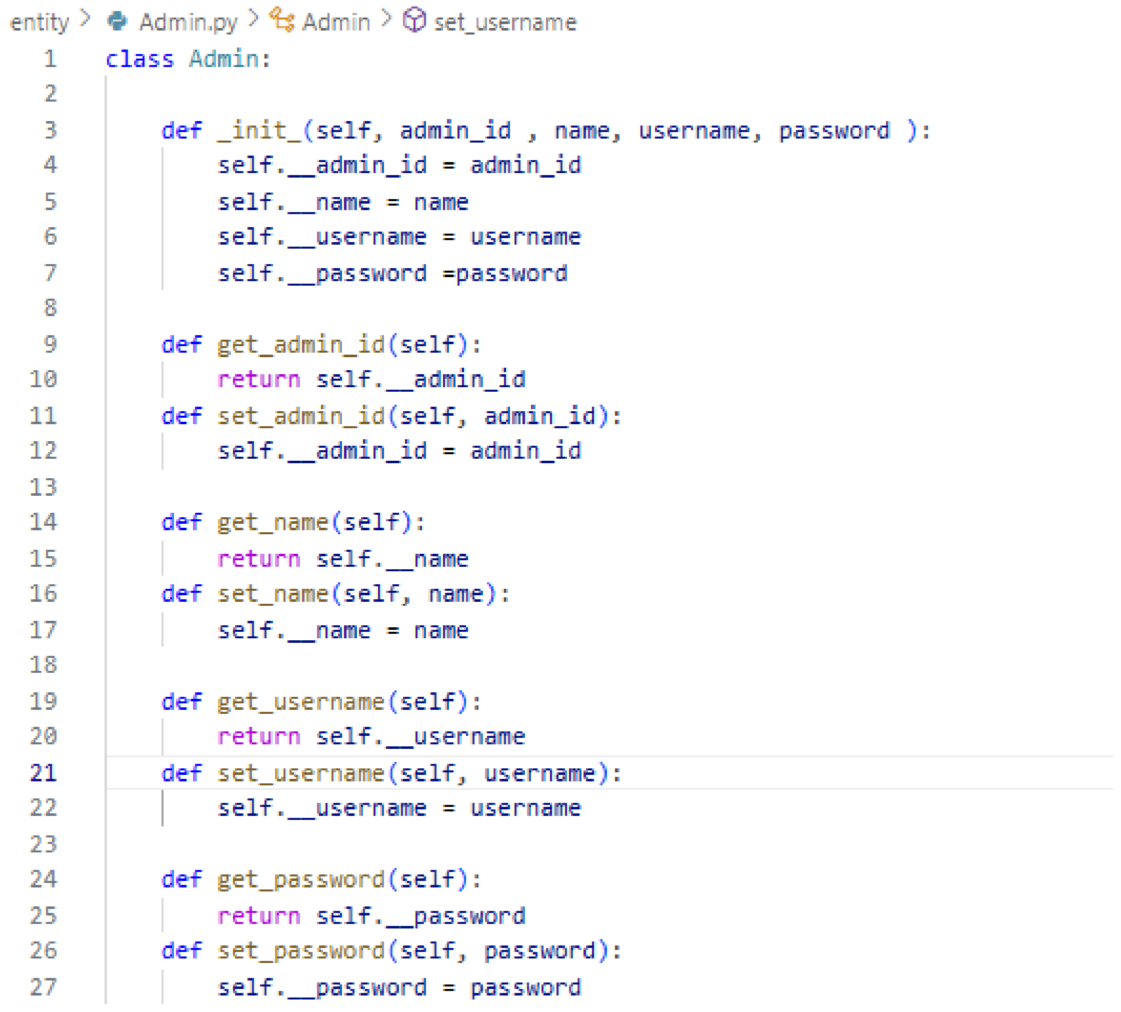
• name

• username

• password



```sql
CREATE TABLE admin(
        admin_id INT PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(100) NOT NULL,
        username VARCHAR(50) UNIQUE NOT NULL,
        password VARCHAR(50) NOT NULL);
```



entity > Admin.py > Admin > set_username

```python
class Admin:

    def _init_(self, admin_id , name, username, password ):
        self.__admin_id = admin_id
        self.__name = name
        self.__username = username
        self.__password =password

    def get_admin_id(self):
        return self.__admin_id
    def set_admin_id(self, admin_id):
        self.__admin_id = admin_id

    def get_name(self):
        return self.__name
    def set_name(self, name):
        self.__name = name

    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
```

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)

6. Service Provider Interface/Abstract class:

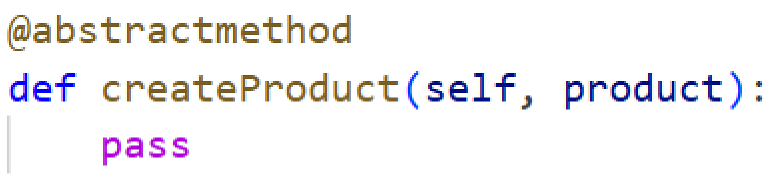
Keep the interfaces and implementation classes in package dao

• Define an OrderProcessorRepository interface/abstract class with methods for adding/removing products to/from the cart and placing orders. The following methods will interact with database.

1. createProduct()

parameter: Product product

return type: Boolean

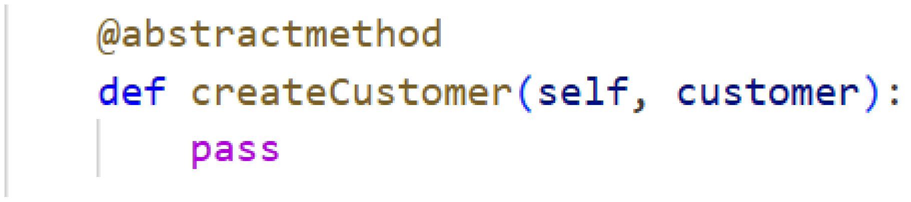


```python
@abstractmethod
def createProduct(self, product):
    pass
```

2. createCustomer()

parameter: Customer customer

return type: boolean

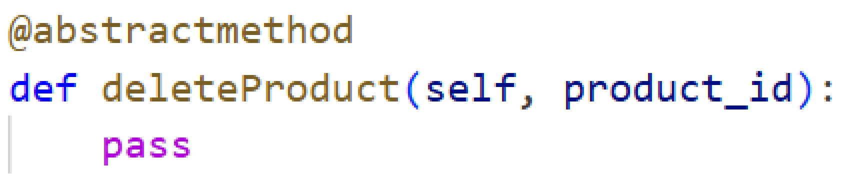


```python
@abstractmethod
def createCustomer(self, customer):
    pass
```

3. deleteProduct()

parameter: productid

return type: boolean

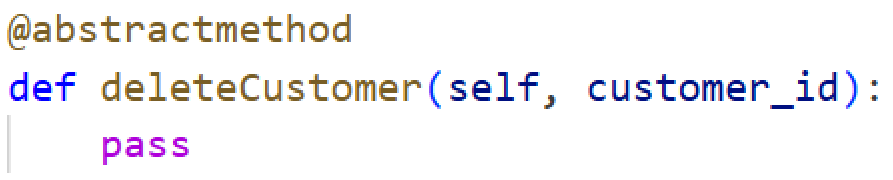


```python
@abstractmethod
def deleteProduct(self, product_id):
    pass
```

4. deleteCustomer(customerId)

parameter: customerId

return type: Boolean

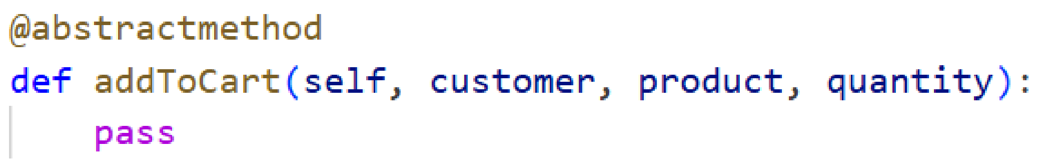


```python
@abstractmethod
def deleteCustomer(self, customer_id):
    pass
```

5. addToCart(): insert the product in cart.

parameter: Customer customer, Product product, int quantity

return type: boolean

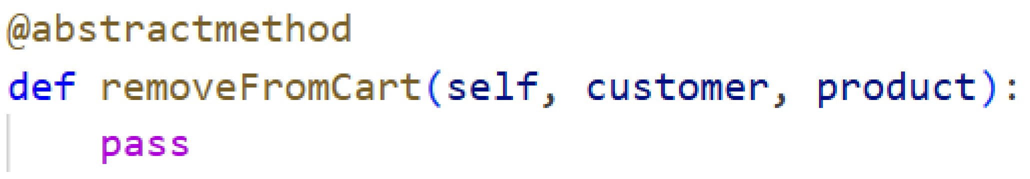


```python
@abstractmethod
def addToCart(self, customer, product, quantity):
    pass
```

6. removeFromCart(): delete the product in cart.

parameter: Customer customer, Product product

return type: boolean

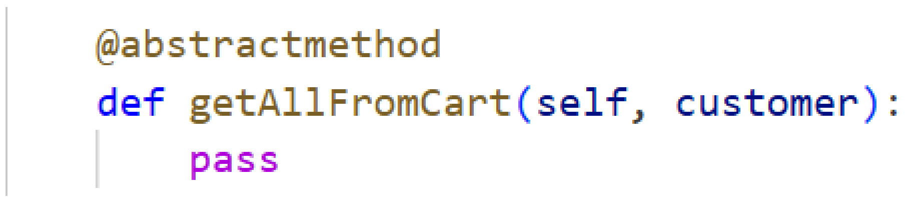


```python
@abstractmethod
def removeFromCart(self, customer, product):
    pass
```

7. getAllFromCart(Customer customer): list the product in cart for a customer.

parameter: Customer customer

return type: list of product

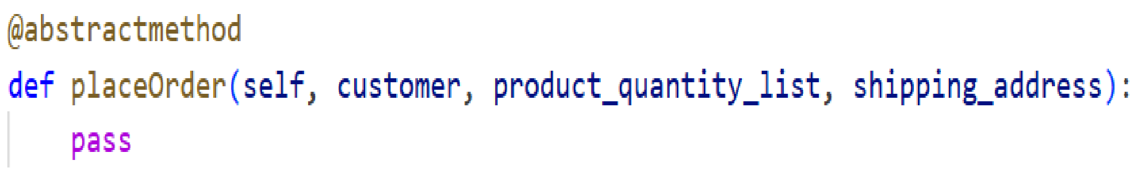


```python
@abstractmethod
def getAllFromCart(self, customer):
    pass
```

8. placeOrder(Customer customer, List<Map>, string shippingAddress): should update order table and orderItems table.

parameter: Customer customer, list of product and quantity

return type: boolean

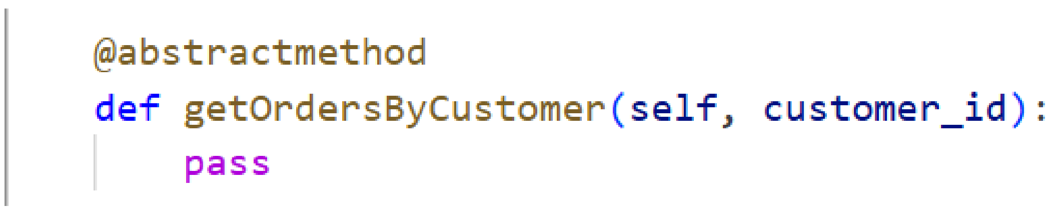


```python
@abstractmethod
def placeOrder(self, customer, product_quantity_list, shipping_address):
    pass
```

9. getOrdersByCustomer()

parameter: customerid

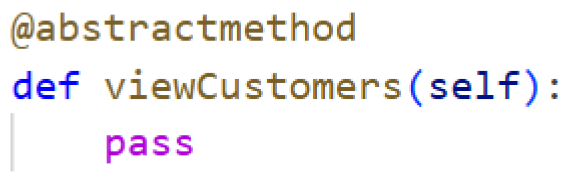
return type: list of product and quantity


```
@abstractmethod
def getOrdersByCustomer(self, customer_id):
    pass
```

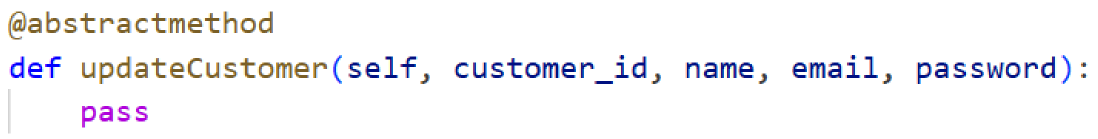
Additionally added abstract classes:

1.viewCustomers(): Can be done only by the Admin

Return type : list of all customers

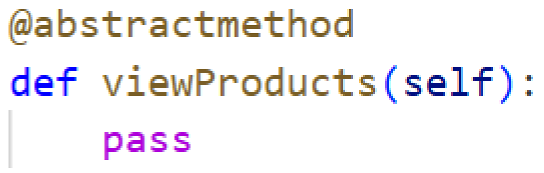

```
@abstractmethod
def viewCustomers(self):
    pass
```

2. updateCustomer() : For updation/changes incase of any error

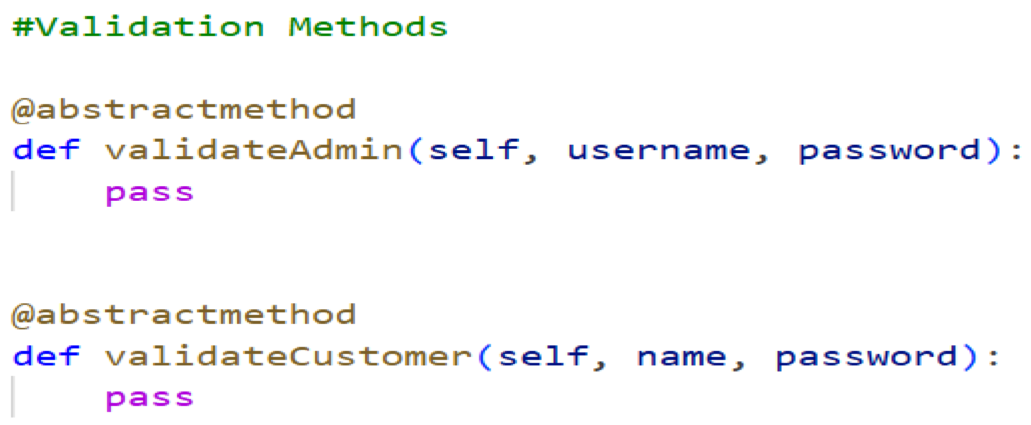

```
@abstractmethod
def updateCustomer(self, customer_id, name, email, password):
    pass
```

3. viewProducts():

Return type: List of all products

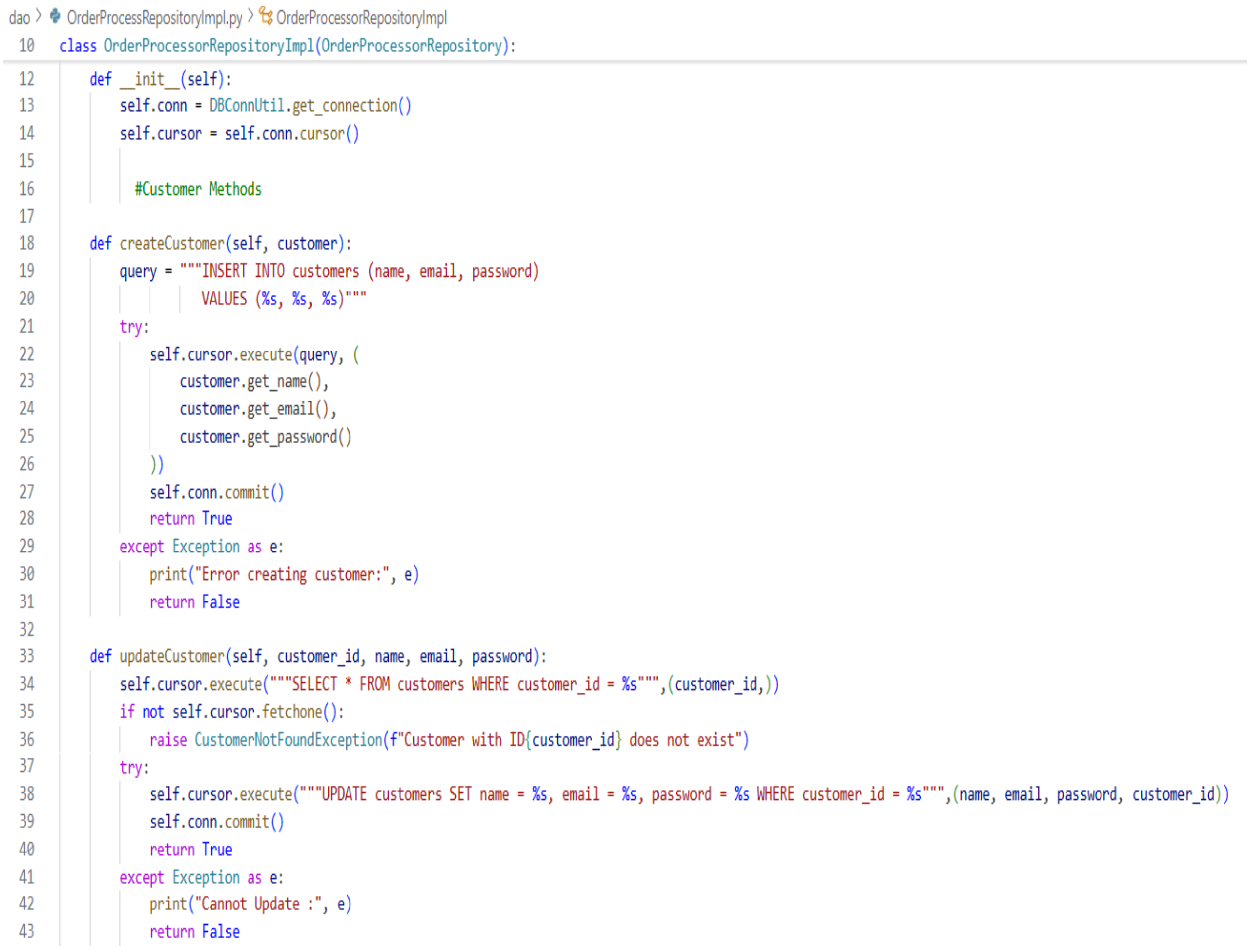

```
@abstractmethod
def viewProducts(self):
    pass
```

4. validataeAdmin() and **validateCustomer():** For improved security



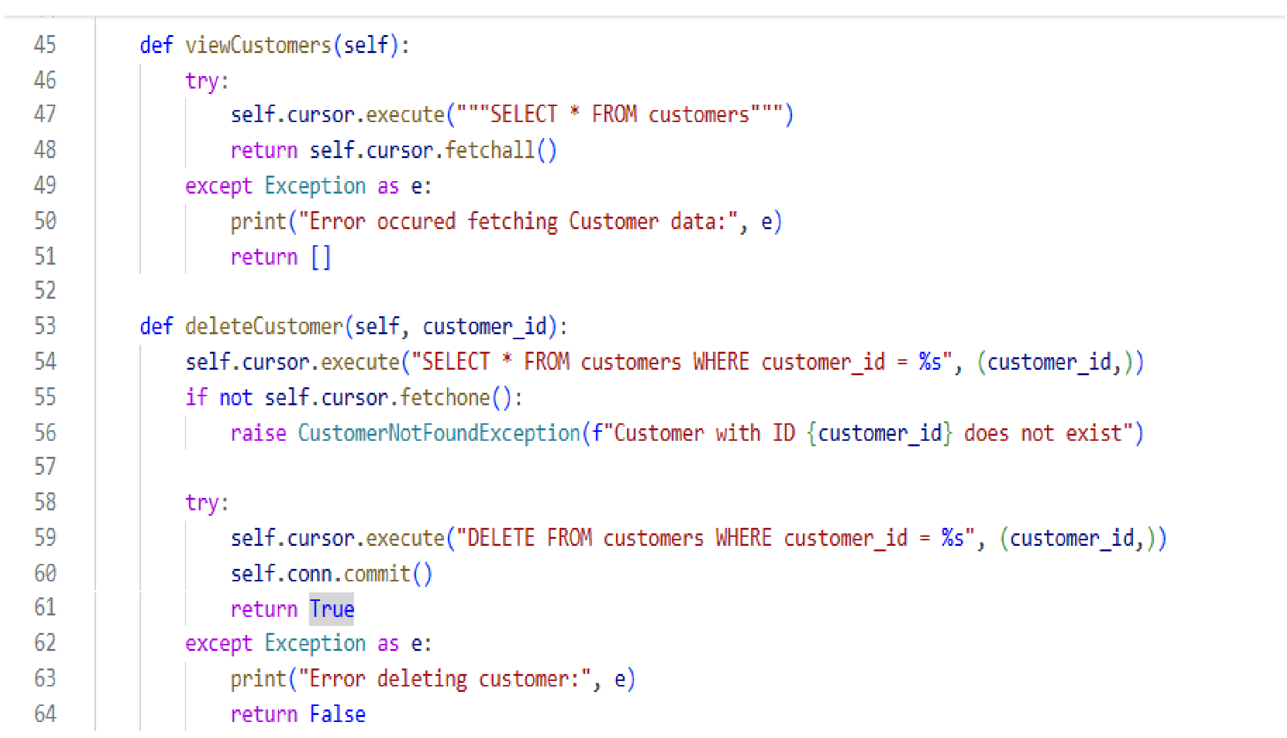
```
#Validation Methods

@abstractmethod
def validateAdmin(self, username, password):
    pass


@abstractmethod
def validateCustomer(self, name, password):
    pass
```

7. Implement the above interface in a class called OrderProcessorRepositoryImpl in package dao.

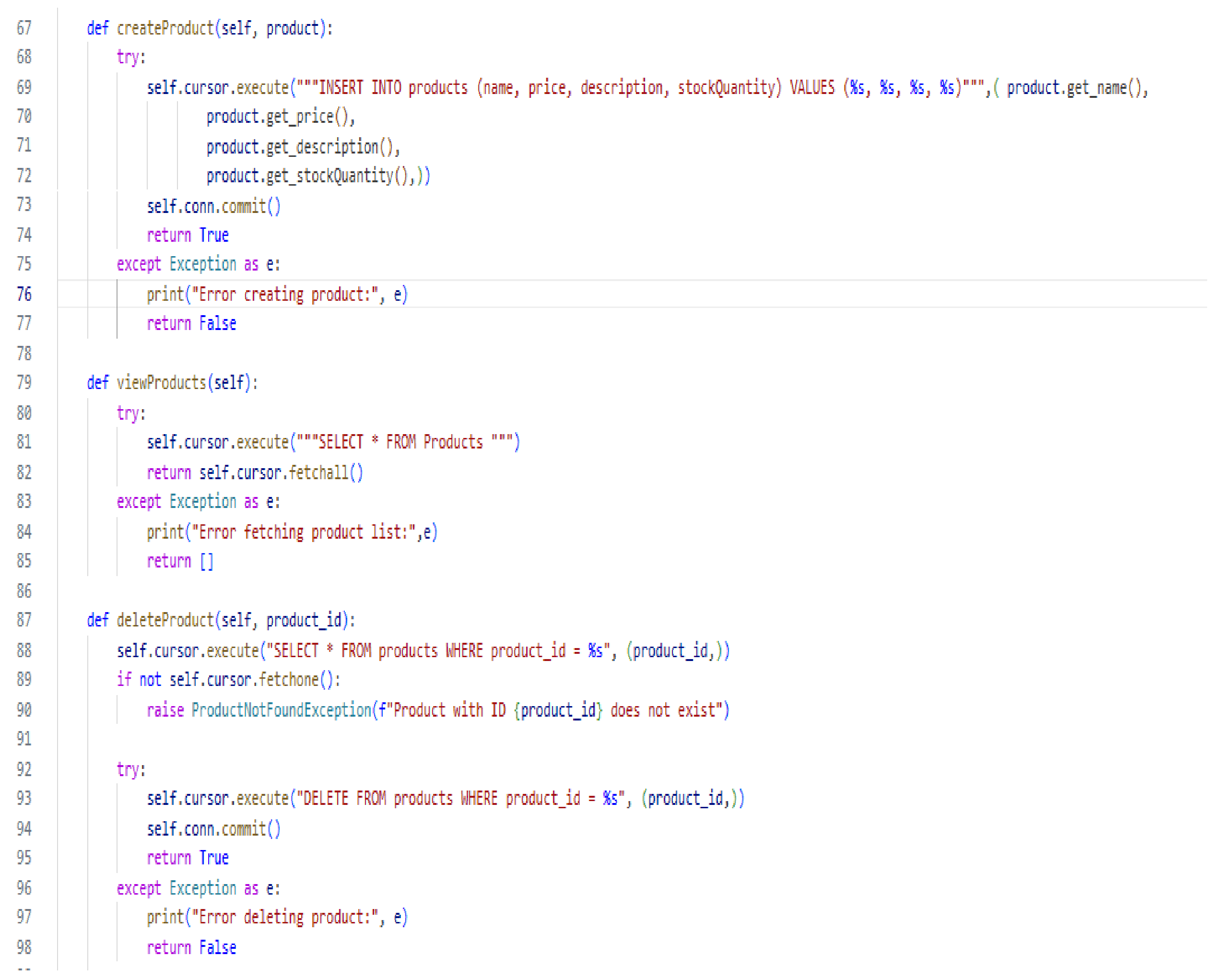


dao > OrderProcessRepositoryImpl.py > OrderProcessorRepositoryImpl

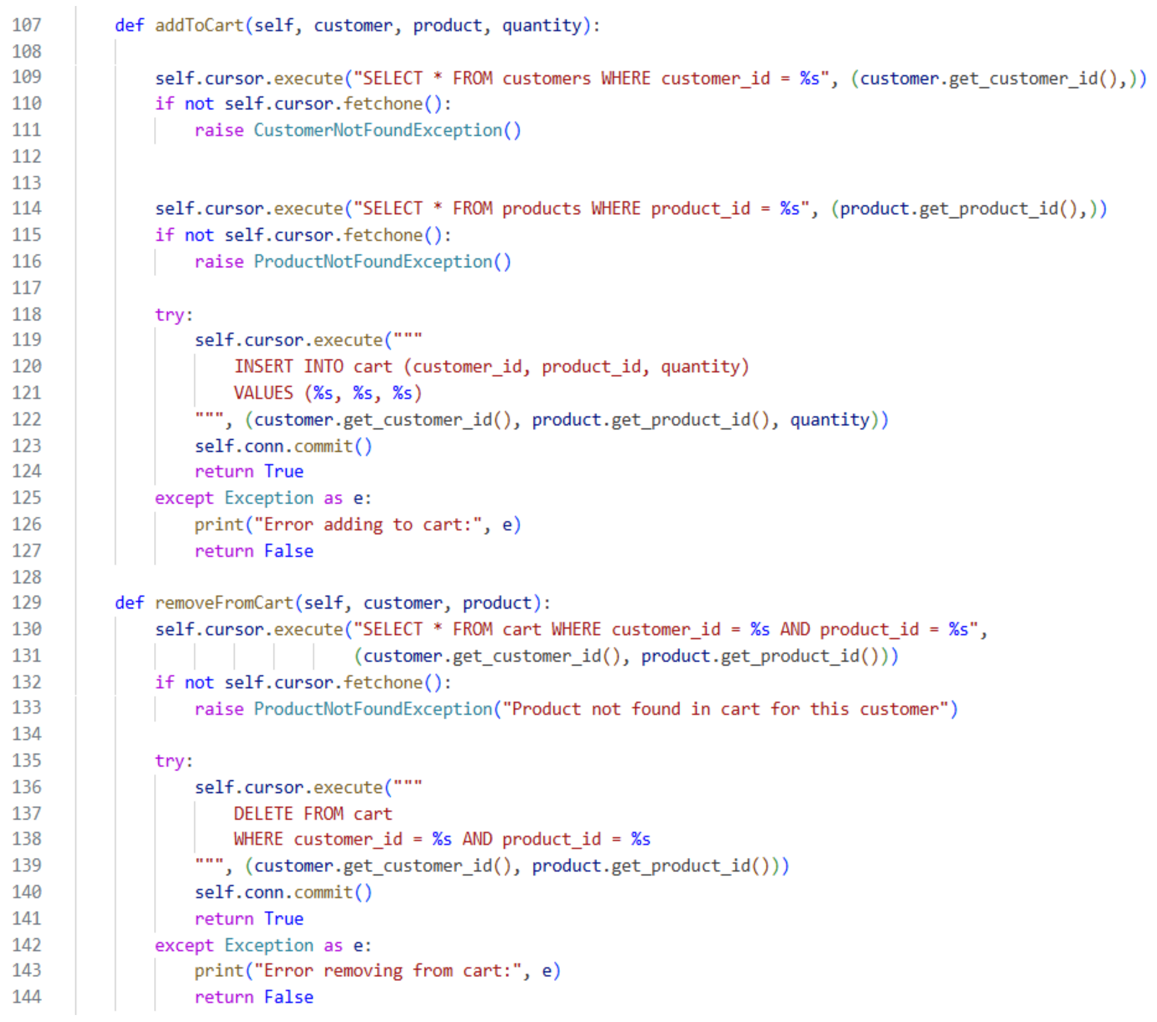
```
class OrderProcessorRepositoryImpl(OrderProcessorRepository):

    def __init__(self):
        self.conn = DBConnUtil.get_connection()
        self.cursor = self.conn.cursor()

         #Customer Methods

    def createCustomer(self, customer):
        query = """INSERT INTO customers (name, email, password)
                   VALUES (%s, %s, %s)"""
        try:
            self.cursor.execute(query, (
                customer.get_name(),
                customer.get_email(),
                customer.get_password()
            ))
            self.conn.commit()
            return True
        except Exception as e:
            print("Error creating customer:", e)
            return False

    def updateCustomer(self, customer_id, name, email, password):
        self.cursor.execute("""SELECT * FROM customers WHERE customer_id = %s""",(customer_id,))
        if not self.cursor.fetchone():
            raise CustomerNotFoundException(f"Customer with ID{customer_id} does not exist")
        try:
            self.cursor.execute("""UPDATE customers SET name = %s, email = %s, password = %s WHERE customer_id = %s""",(name, email, password, customer_id))
            self.conn.commit()
            return True
        except Exception as e:
            print("Cannot Update :", e)
            return False
```



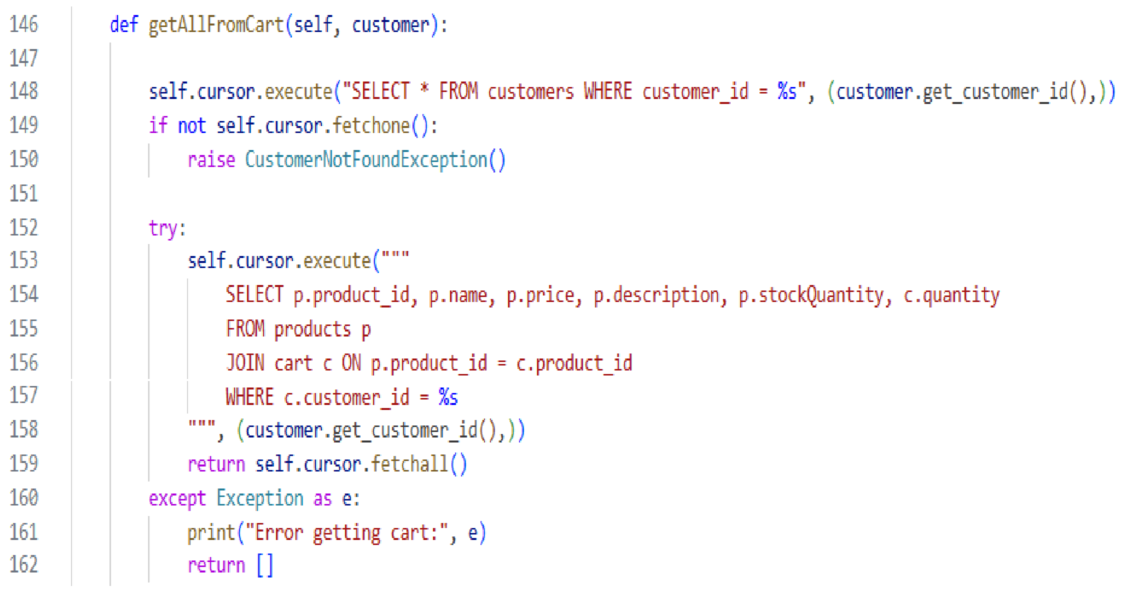
```python
    def viewCustomers(self):
        try:
            self.cursor.execute("""SELECT * FROM customers""")
            return self.cursor.fetchall()
        except Exception as e:
            print("Error occured fetching Customer data:", e)
            return []

    def deleteCustomer(self, customer_id):
        self.cursor.execute("SELECT * FROM customers WHERE customer_id = %s", (customer_id,))
        if not self.cursor.fetchone():
            raise CustomerNotFoundException(f"Customer with ID {customer_id} does not exist")

        try:
            self.cursor.execute("DELETE FROM customers WHERE customer_id = %s", (customer_id,))
            self.conn.commit()
            return True
        except Exception as e:
            print("Error deleting customer:", e)
            return False
```



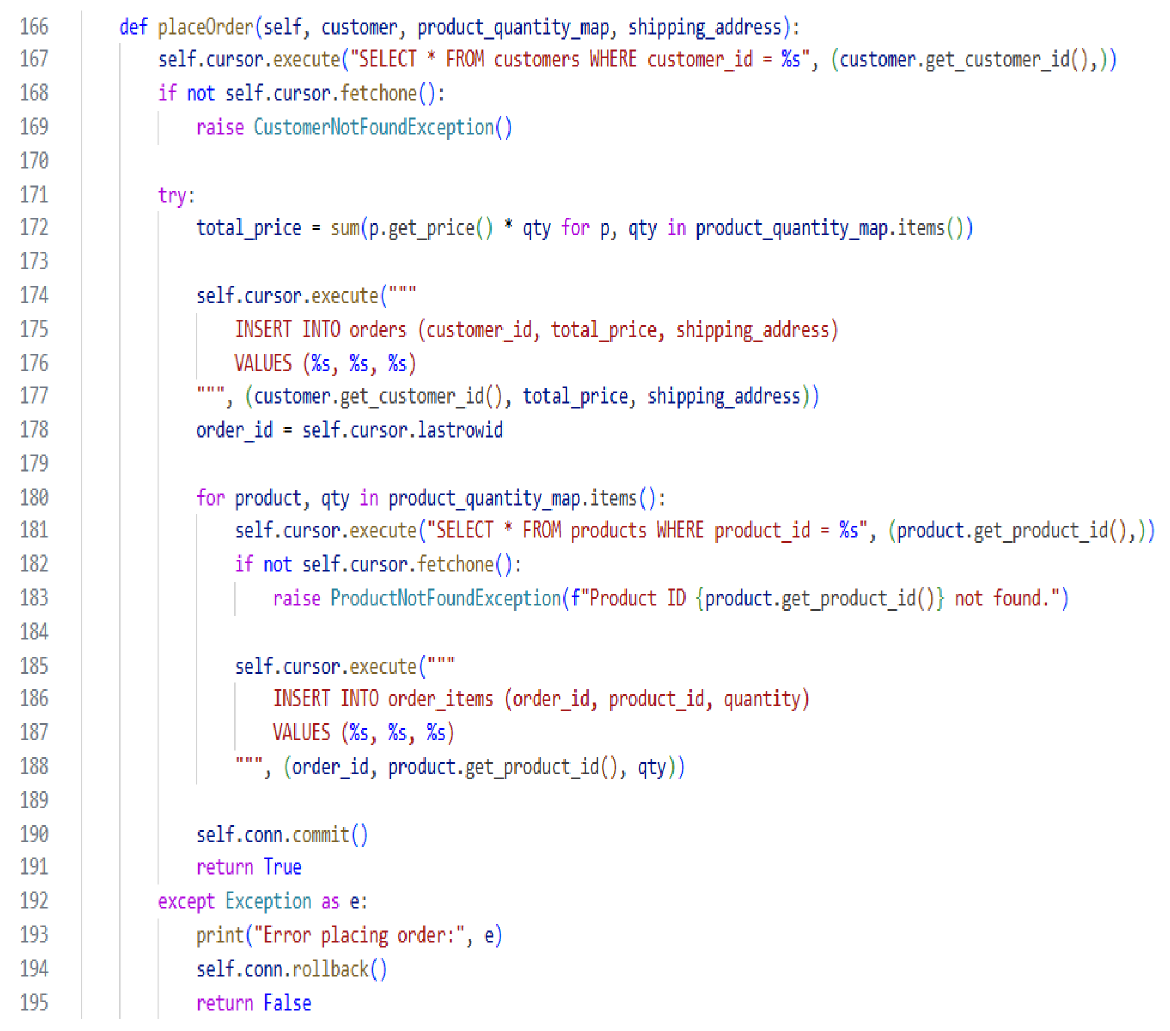
```python
    def createProduct(self, product):
        try:
            self.cursor.execute("""INSERT INTO products (name, price, description, stockQuantity) VALUES (%s, %s, %s, %s)""",( product.get_name(),
                    product.get_price(),
                    product.get_description(),
                    product.get_stockQuantity(),))
            self.conn.commit()
            return True
        except Exception as e:
            print("Error creating product:", e)
            return False

    def viewProducts(self):
        try:
            self.cursor.execute("""SELECT * FROM Products """)
            return self.cursor.fetchall()
        except Exception as e:
            print("Error fetching product list:",e)
            return []

    def deleteProduct(self, product_id):
        self.cursor.execute("SELECT * FROM products WHERE product_id = %s", (product_id,))
        if not self.cursor.fetchone():
            raise ProductNotFoundException(f"Product with ID {product_id} does not exist")

        try:
            self.cursor.execute("DELETE FROM products WHERE product_id = %s", (product_id,))
            self.conn.commit()
            return True
        except Exception as e:
            print("Error deleting product:", e)
            return False
```



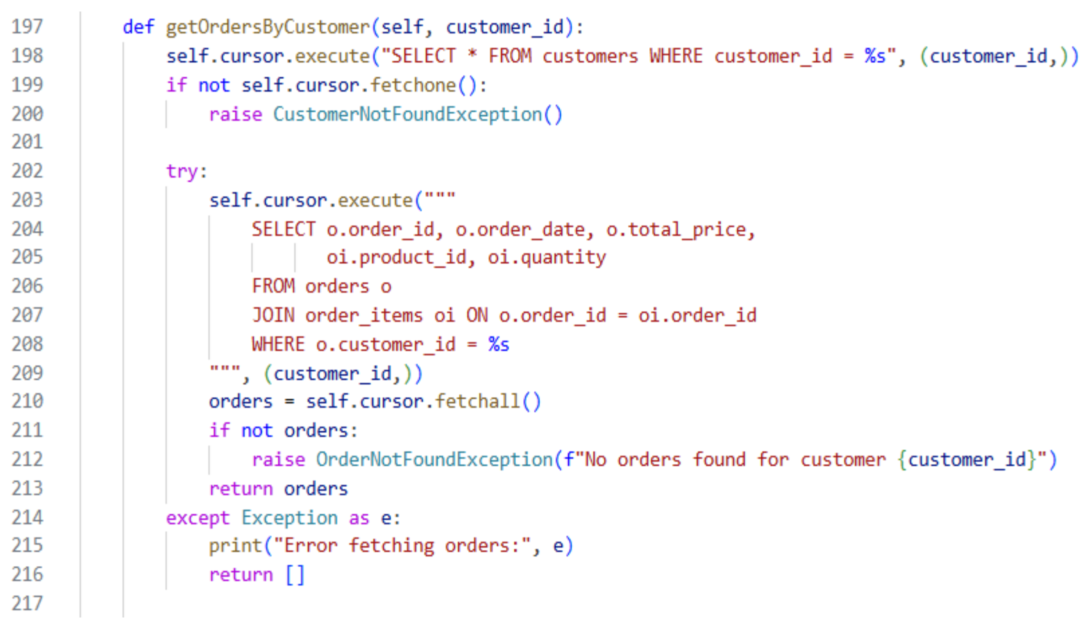
```python
    def addToCart(self, customer, product, quantity):

        self.cursor.execute("SELECT * FROM customers WHERE customer_id = %s", (customer.get_customer_id(),))
        if not self.cursor.fetchone():
            raise CustomerNotFoundException()


        self.cursor.execute("SELECT * FROM products WHERE product_id = %s", (product.get_product_id(),))
        if not self.cursor.fetchone():
            raise ProductNotFoundException()

        try:
            self.cursor.execute("""
                INSERT INTO cart (customer_id, product_id, quantity)
                VALUES (%s, %s, %s)
            """, (customer.get_customer_id(), product.get_product_id(), quantity))
            self.conn.commit()
            return True
        except Exception as e:
            print("Error adding to cart:", e)
            return False

    def removeFromCart(self, customer, product):
        self.cursor.execute("SELECT * FROM cart WHERE customer_id = %s AND product_id = %s",
                            (customer.get_customer_id(), product.get_product_id()))
        if not self.cursor.fetchone():
            raise ProductNotFoundException("Product not found in cart for this customer")

        try:
            self.cursor.execute("""
                DELETE FROM cart
                WHERE customer_id = %s AND product_id = %s
            """, (customer.get_customer_id(), product.get_product_id()))
            self.conn.commit()
            return True
        except Exception as e:
            print("Error removing from cart:", e)
            return False
```



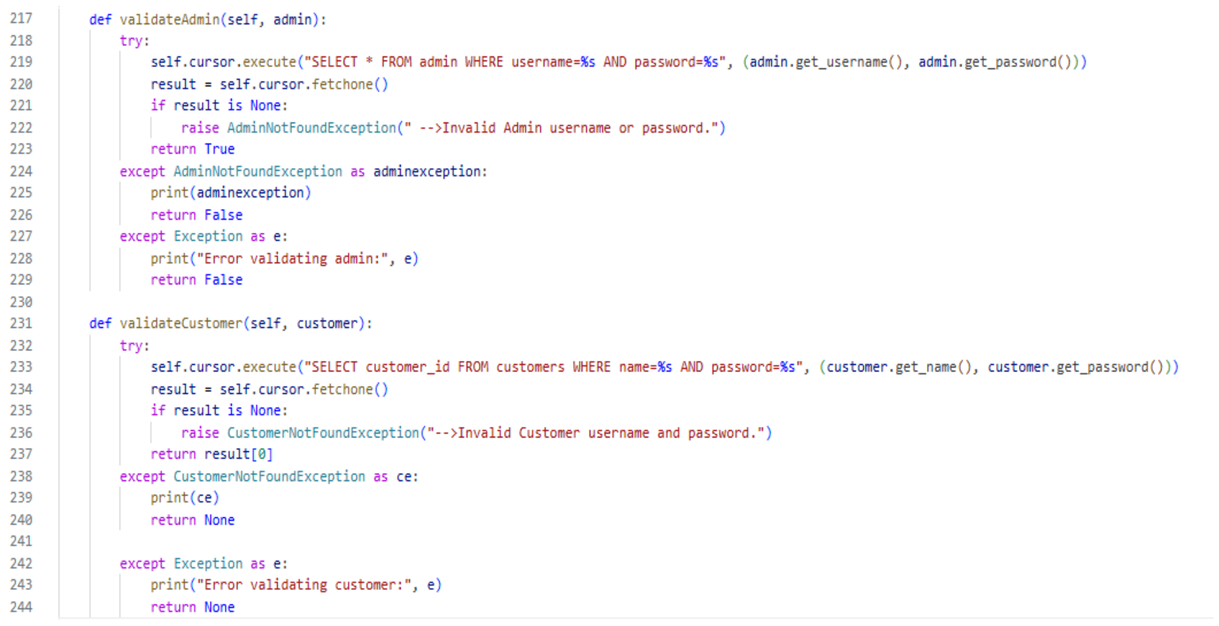
```python
    def getAllFromCart(self, customer):

        self.cursor.execute("SELECT * FROM customers WHERE customer_id = %s", (customer.get_customer_id(),))
        if not self.cursor.fetchone():
            raise CustomerNotFoundException()

        try:
            self.cursor.execute("""
                SELECT p.product_id, p.name, p.price, p.description, p.stockQuantity, c.quantity
                FROM products p
                JOIN cart c ON p.product_id = c.product_id
                WHERE c.customer_id = %s
            """, (customer.get_customer_id(),))
            return self.cursor.fetchall()
        except Exception as e:
            print("Error getting cart:", e)
            return []
```



```python
def placeOrder(self, customer, product_quantity_map, shipping_address):
    self.cursor.execute("SELECT * FROM customers WHERE customer_id = %s", (customer.get_customer_id(),))
    if not self.cursor.fetchone():
        raise CustomerNotFoundException()

    try:
        total_price = sum(p.get_price() * qty for p, qty in product_quantity_map.items())

        self.cursor.execute("""
            INSERT INTO orders (customer_id, total_price, shipping_address)
            VALUES (%s, %s, %s)
        """, (customer.get_customer_id(), total_price, shipping_address))
        order_id = self.cursor.lastrowid

        for product, qty in product_quantity_map.items():
            self.cursor.execute("SELECT * FROM products WHERE product_id = %s", (product.get_product_id(),))
            if not self.cursor.fetchone():
                raise ProductNotFoundException(f"Product ID {product.get_product_id()} not found.")

            self.cursor.execute("""
                INSERT INTO order_items (order_id, product_id, quantity)
                VALUES (%s, %s, %s)
            """, (order_id, product.get_product_id(), qty))

        self.conn.commit()
        return True
    except Exception as e:
        print("Error placing order:", e)
        self.conn.rollback()
        return False
```



```python
def getOrdersByCustomer(self, customer_id):
    self.cursor.execute("SELECT * FROM customers WHERE customer_id = %s", (customer_id,))
    if not self.cursor.fetchone():
        raise CustomerNotFoundException()

    try:
        self.cursor.execute("""
            SELECT o.order_id, o.order_date, o.total_price,
                   oi.product_id, oi.quantity
            FROM orders o
            JOIN order_items oi ON o.order_id = oi.order_id
            WHERE o.customer_id = %s
        """, (customer_id,))
        orders = self.cursor.fetchall()
        if not orders:
            raise OrderNotFoundException(f"No orders found for customer {customer_id}")
        return orders
    except Exception as e:
        print("Error fetching orders:", e)
        return []

```

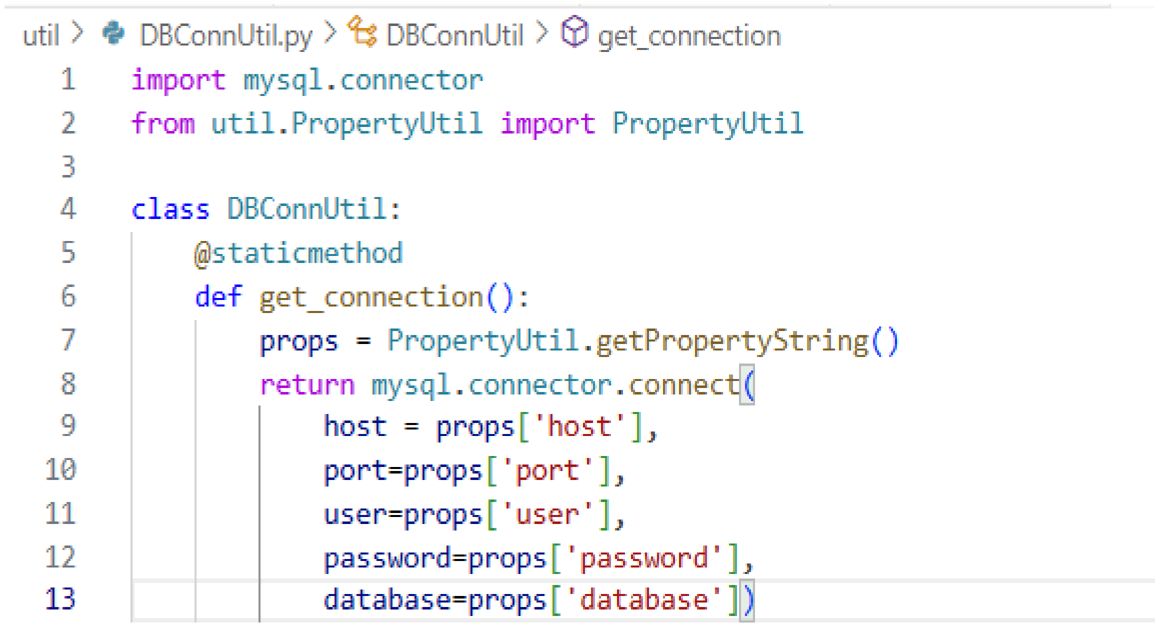


```python
    def validateAdmin(self, admin):
        try:
            self.cursor.execute("SELECT * FROM admin WHERE username=%s AND password=%s", (admin.get_username(), admin.get_password()))
            result = self.cursor.fetchone()
            if result is None:
                raise AdminNotFoundException(" -->Invalid Admin username or password.")
            return True
        except AdminNotFoundException as adminexception:
            print(adminexception)
            return False
        except Exception as e:
            print("Error validating admin:", e)
            return False

    def validateCustomer(self, customer):
        try:
            self.cursor.execute("SELECT customer_id FROM customers WHERE name=%s AND password=%s", (customer.get_name(), customer.get_password()))
            result = self.cursor.fetchone()
            if result is None:
                raise CustomerNotFoundException("-->Invalid Customer username and password.")
            return result[0]
        except CustomerNotFoundException as ce:
            print(ce)
            return None

        except Exception as e:
            print("Error validating customer:", e)
            return None
```

Connect your application to the SQL database:

8. Write code to establish a connection to your SQL database.

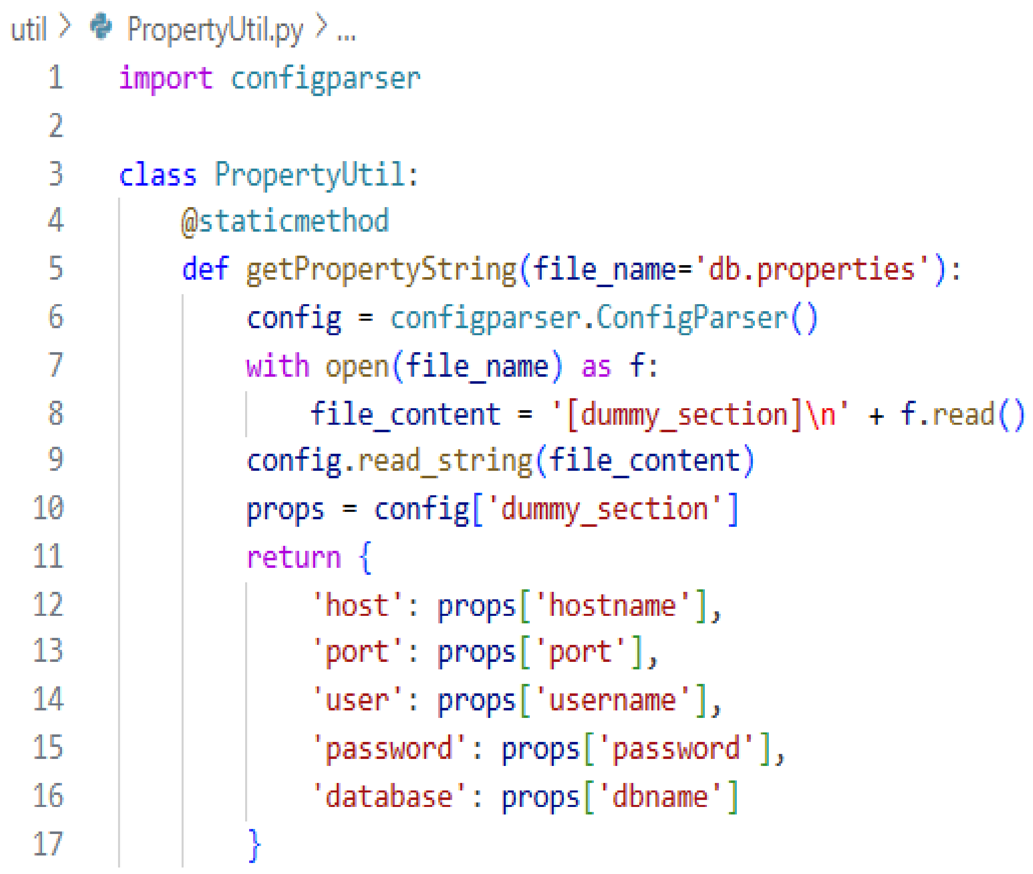
• Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.



```python
import mysql.connector
from util.PropertyUtil import PropertyUtil

class DBConnUtil:
    @staticmethod
    def get_connection():
        props = PropertyUtil.getPropertyString()
        return mysql.connector.connect(
            host = props['host'],
            port=props['port'],
            user=props['user'],
            password=props['password'],
            database=props['database'])
```

• Connection properties supplied in the connection string should be read from a property file.

• Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.



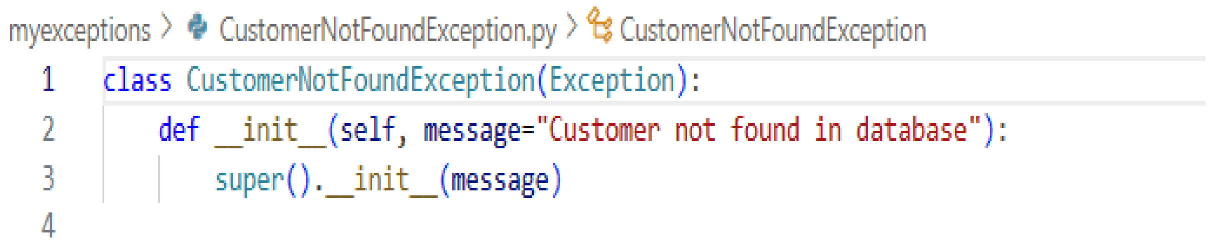
util > PropertyUtil.py > ...

```python
import configparser

class PropertyUtil:
    @staticmethod
    def getPropertyString(file_name='db.properties'):
        config = configparser.ConfigParser()
        with open(file_name) as f:
            file_content = '[dummy_section]\n' + f.read()
        config.read_string(file_content)
        props = config['dummy_section']
        return {
            'host': props['hostname'],
            'port': props['port'],
            'user': props['username'],
            'password': props['password'],
            'database': props['dbname']
        }
```

9. Create the exceptions

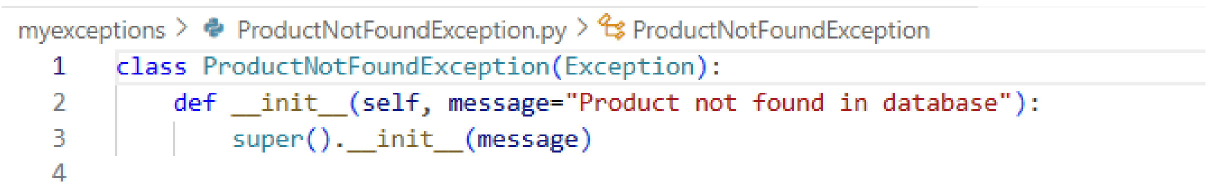
In package myexceptions and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

• **CustomerNotFoundException:** throw this exception when user enters an invalid customer id which doesn't exist in db


myexceptions > CustomerNotFoundException.py > CustomerNotFoundException

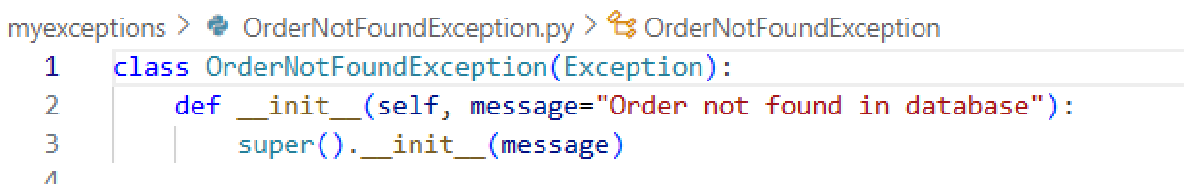
```python
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found in database"):
        super().__init__(message)

```

• **ProductNotFoundException:** throw this exception when user enters an invalid product id which doesn't exist in db


myexceptions > ProductNotFoundException.py > ProductNotFoundException

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

```

• **OrderNotFoundException:** throw this exception when user enters an invalid order id which doesn't exist in db


myexceptions > OrderNotFoundException.py > OrderNotFoundException

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

```

10. Create class with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.

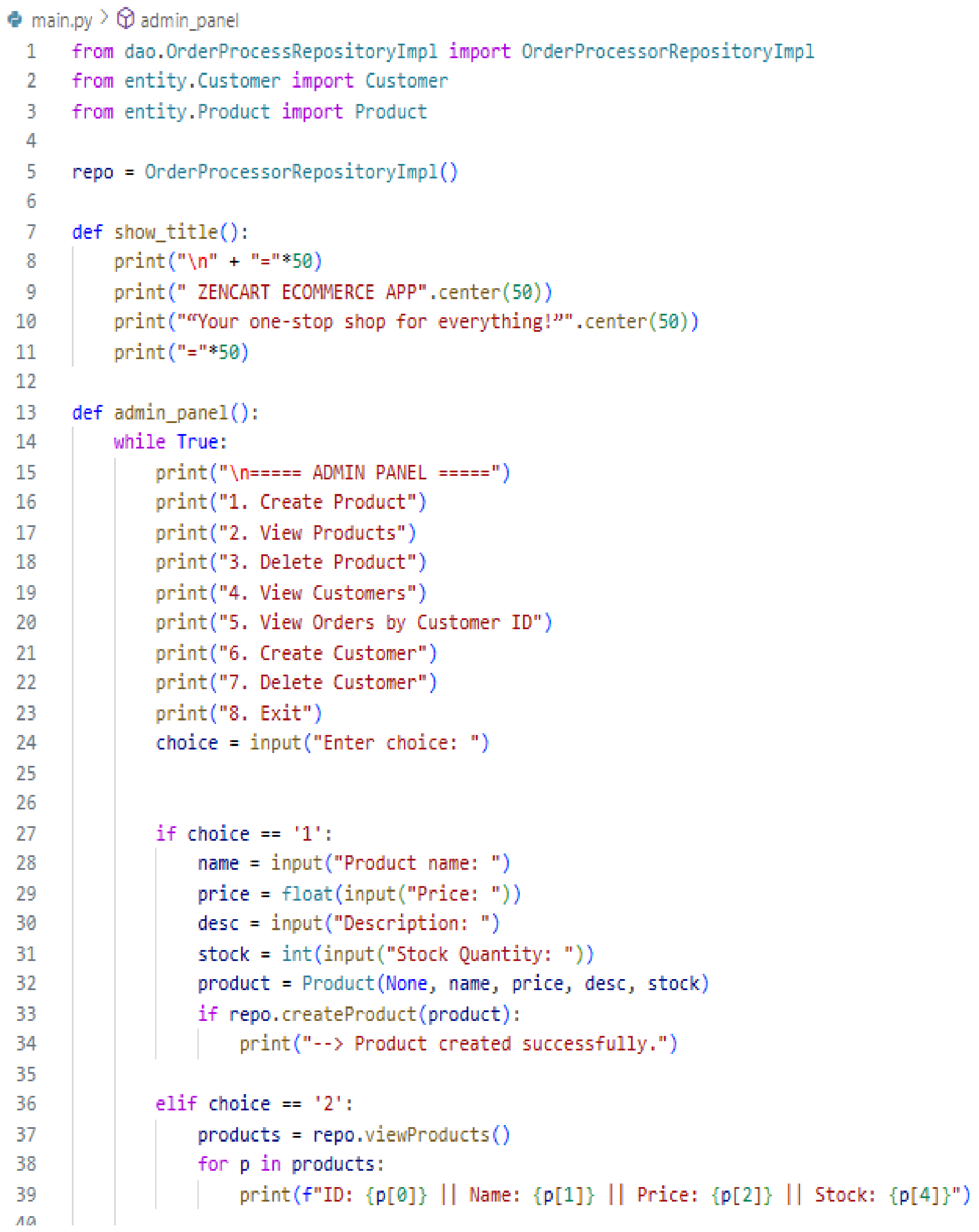
Existing given actions

1. Register Customer.
2. Create Product.
3. Delete Product.
4. Add to cart.
5. View cart.
6. Place order.
7. View Customer Order

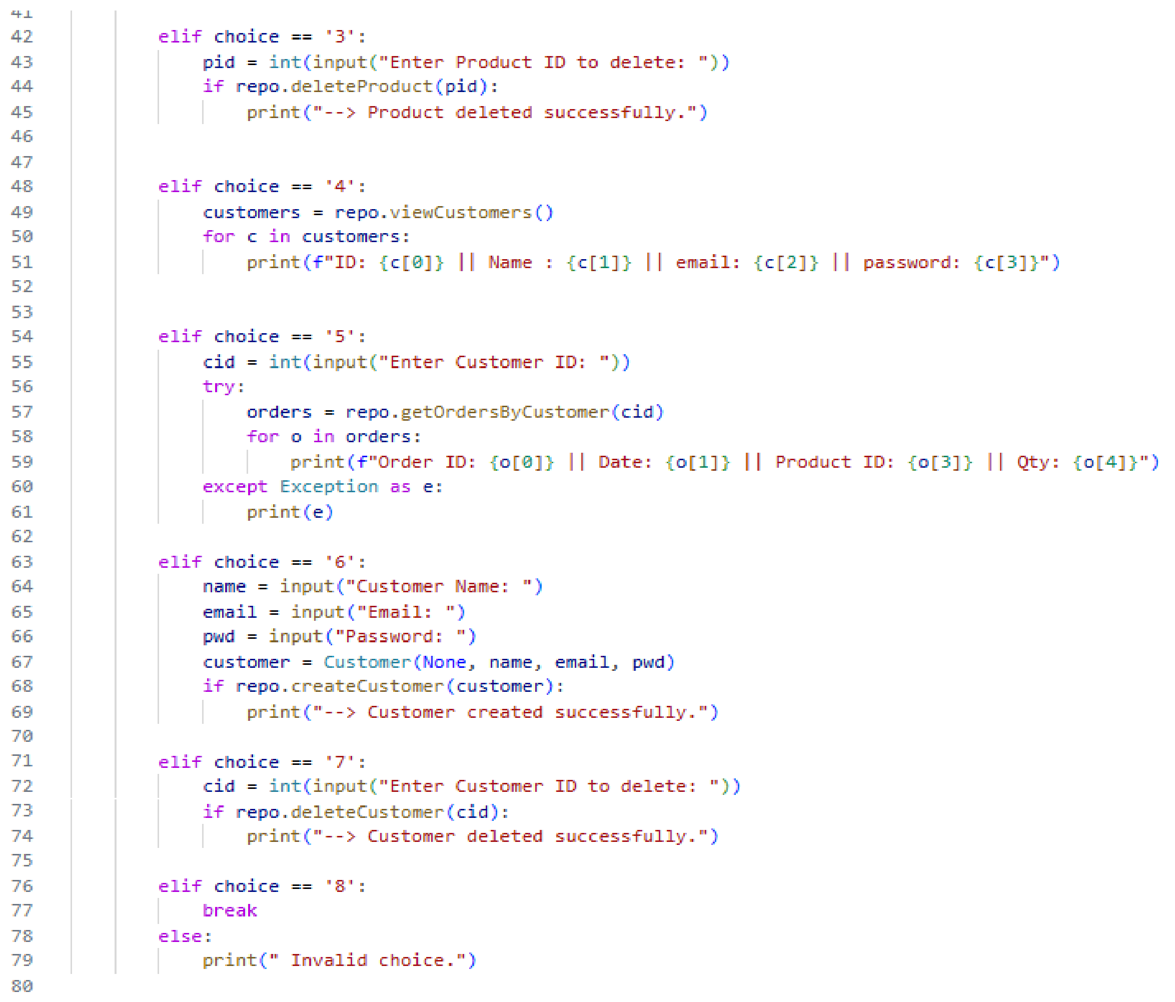
VS How we framed actions

CUSTOMERS	ADMIN
1. Signup/Login	1. Create Product
2. View Products	2. View Product
3. Add to Cart	3. Delete Product
4. Remove from Cart	4. View Customers
5. View Cart	5. View Orders by Customer ID
6. Place Orders	6. Create Customer
7. View Orders	7. Delete Customer
8. Update Details	

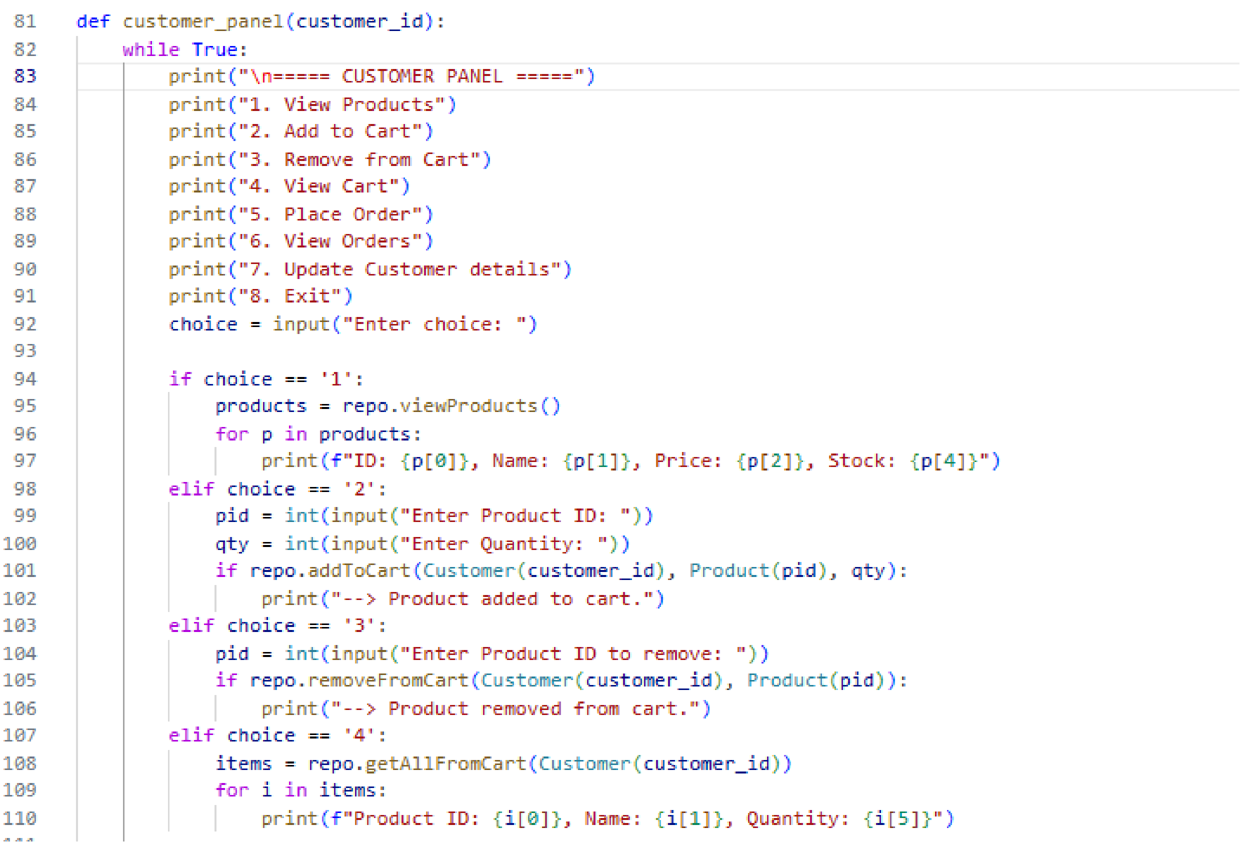
main.py



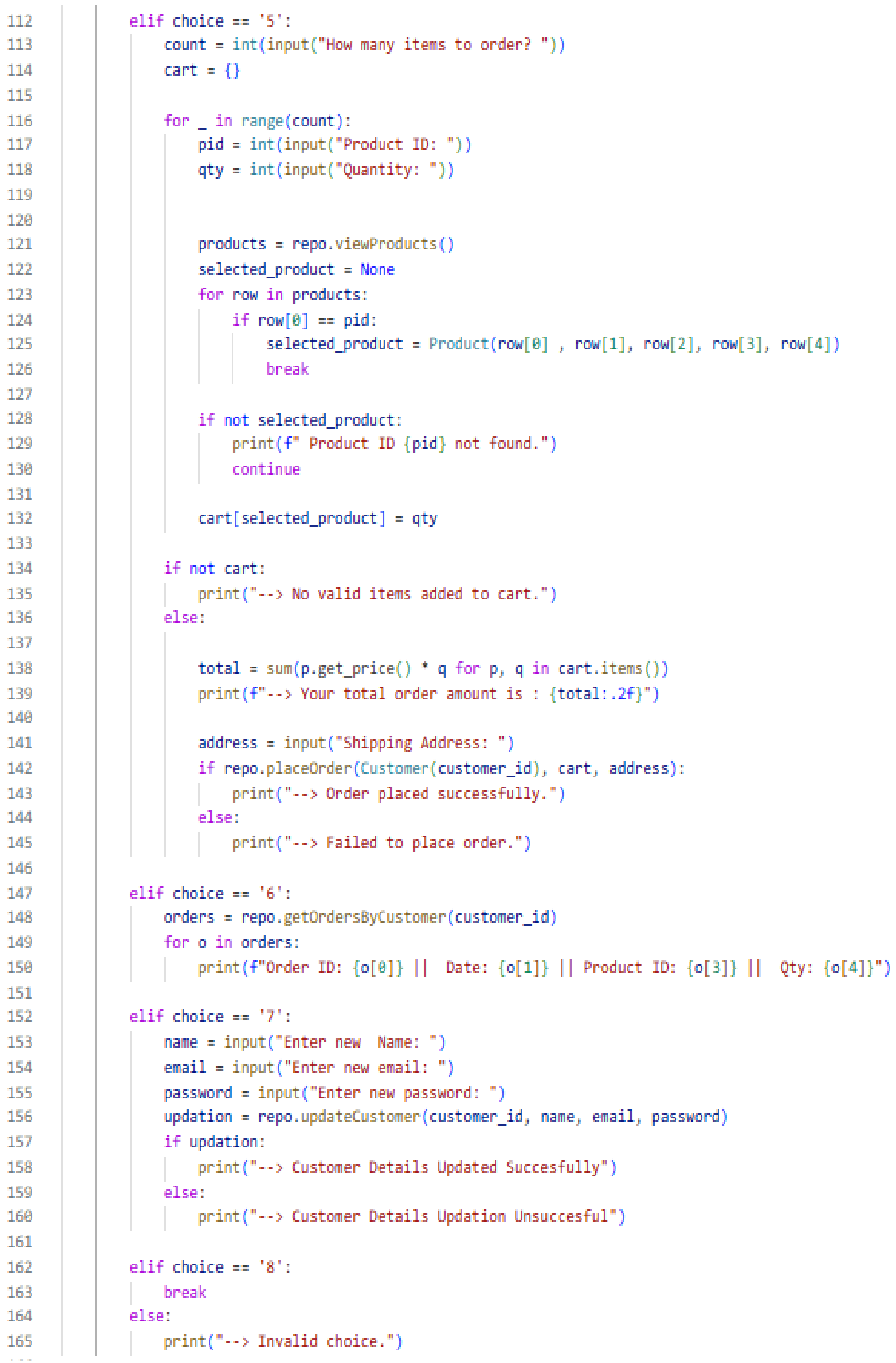
```python
from dao.OrderProcessRepositoryImpl import OrderProcessorRepositoryImpl
from entity.Customer import Customer
from entity.Product import Product

repo = OrderProcessorRepositoryImpl()

def show_title():
    print("\n" + "="*50)
    print(" ZENCART ECOMMERCE APP".center(50))
    print("“Your one-stop shop for everything!”".center(50))
    print("="*50)

def admin_panel():
    while True:
        print("\n===== ADMIN PANEL =====")
        print("1. Create Product")
        print("2. View Products")
        print("3. Delete Product")
        print("4. View Customers")
        print("5. View Orders by Customer ID")
        print("6. Create Customer")
        print("7. Delete Customer")
        print("8. Exit")
        choice = input("Enter choice: ")


        if choice == '1':
            name = input("Product name: ")
            price = float(input("Price: "))
            desc = input("Description: ")
            stock = int(input("Stock Quantity: "))
            product = Product(None, name, price, desc, stock)
            if repo.createProduct(product):
                print("--> Product created successfully.")

        elif choice == '2':
            products = repo.viewProducts()
            for p in products:
                print(f"ID: {p[0]} || Name: {p[1]} || Price: {p[2]} || Stock: {p[4]}")
```



```python
        elif choice == '3':
            pid = int(input("Enter Product ID to delete: "))
            if repo.deleteProduct(pid):
                print("--> Product deleted successfully.")


        elif choice == '4':
            customers = repo.viewCustomers()
            for c in customers:
                print(f"ID: {c[0]} || Name : {c[1]} || email: {c[2]} || password: {c[3]}")


        elif choice == '5':
            cid = int(input("Enter Customer ID: "))
            try:
                orders = repo.getOrdersByCustomer(cid)
                for o in orders:
                    print(f"Order ID: {o[0]} || Date: {o[1]} || Product ID: {o[3]} || Qty: {o[4]}")
            except Exception as e:
                print(e)

        elif choice == '6':
            name = input("Customer Name: ")
            email = input("Email: ")
            pwd = input("Password: ")
            customer = Customer(None, name, email, pwd)
            if repo.createCustomer(customer):
                print("--> Customer created successfully.")

        elif choice == '7':
            cid = int(input("Enter Customer ID to delete: "))
            if repo.deleteCustomer(cid):
                print("--> Customer deleted successfully.")

        elif choice == '8':
            break
        else:
            print(" Invalid choice.")

```



```python
def customer_panel(customer_id):
    while True:
        print("\n===== CUSTOMER PANEL =====")
        print("1. View Products")
        print("2. Add to Cart")
        print("3. Remove from Cart")
        print("4. View Cart")
        print("5. Place Order")
        print("6. View Orders")
        print("7. Update Customer details")
        print("8. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            products = repo.viewProducts()
            for p in products:
                print(f"ID: {p[0]}, Name: {p[1]}, Price: {p[2]}, Stock: {p[4]}")
        elif choice == '2':
            pid = int(input("Enter Product ID: "))
            qty = int(input("Enter Quantity: "))
            if repo.addToCart(Customer(customer_id), Product(pid), qty):
                print("--> Product added to cart.")
        elif choice == '3':
            pid = int(input("Enter Product ID to remove: "))
            if repo.removeFromCart(Customer(customer_id), Product(pid)):
                print("--> Product removed from cart.")
        elif choice == '4':
            items = repo.getAllFromCart(Customer(customer_id))
            for i in items:
                print(f"Product ID: {i[0]}, Name: {i[1]}, Quantity: {i[5]}")
```



```python
        elif choice == '5':
            count = int(input("How many items to order? "))
            cart = {}

            for _ in range(count):
                pid = int(input("Product ID: "))
                qty = int(input("Quantity: "))


                products = repo.viewProducts()
                selected_product = None
                for row in products:
                    if row[0] == pid:
                        selected_product = Product(row[0] , row[1], row[2], row[3], row[4])
                        break

                if not selected_product:
                    print(f" Product ID {pid} not found.")
                    continue

                cart[selected_product] = qty

            if not cart:
                print("--> No valid items added to cart.")
            else:

                total = sum(p.get_price() * q for p, q in cart.items())
                print(f"--> Your total order amount is : {total:.2f}")

                address = input("Shipping Address: ")
                if repo.placeOrder(Customer(customer_id), cart, address):
                    print("--> Order placed successfully.")
                else:
                    print("--> Failed to place order.")

        elif choice == '6':
            orders = repo.getOrdersByCustomer(customer_id)
            for o in orders:
                print(f"Order ID: {o[0]} ||  Date: {o[1]} || Product ID: {o[3]} ||  Qty: {o[4]}")

        elif choice == '7':
            name = input("Enter new  Name: ")
            email = input("Enter new email: ")
            password = input("Enter new password: ")
            updation = repo.updateCustomer(customer_id, name, email, password)
            if updation:
                print("--> Customer Details Updated Succesfully")
            else:
                print("--> Customer Details Updation Unsuccesful")

        elif choice == '8':
            break
        else:
            print("--> Invalid choice.")
```

```python
def customer_login_flow():
    while True:
        print("\n=== CUSTOMER SECTION ===")
        print("1. New Registration")
        print("2. Login")
        print("3. Back")
        choice = input("Enter choice: ")

        if choice == '1':
            name = input("Enter Name: ")
            email = input("Enter Email: ")
            pwd = input("Enter Password: ")
            c = Customer(None, name, email, pwd)
            if repo.createCustomer(c):
                print("--> Registered successfully.")
        elif choice == '2':
            name = input("Enter Name: ")
            pwd = input("Enter Password: ")
            customer_id = repo.validateCustomer(name, pwd)
            if customer_id:
                print("--> Login successful.")
                customer_panel(customer_id)
            else:
                print("--> Invalid credentials.")
        elif choice == '3':
            break
        else:
            print("--> Invalid choice.")

def admin_login_flow():
    print("\n=== ADMIN LOGIN ===")
    uname = input("Enter Username: ")
    pwd = input("Enter Password: ")
    if repo.validateAdmin(uname, pwd):
        print("--> Admin login successful.")
        admin_panel()
    else:
        print("--> Invalid credentials.")

def main():
    show_title()
    while True:
        print("\n=== MAIN MENU ===")
        print("1. Customer")
        print("2. Admin")
        print("3. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            customer_login_flow()
        elif choice == '2':
            admin_login_flow()
        elif choice == '3':
            print(" Exiting ZENCART App. Thank you!")
            break
        else:
            print("--> Invalid choice.")

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

Unit Testing

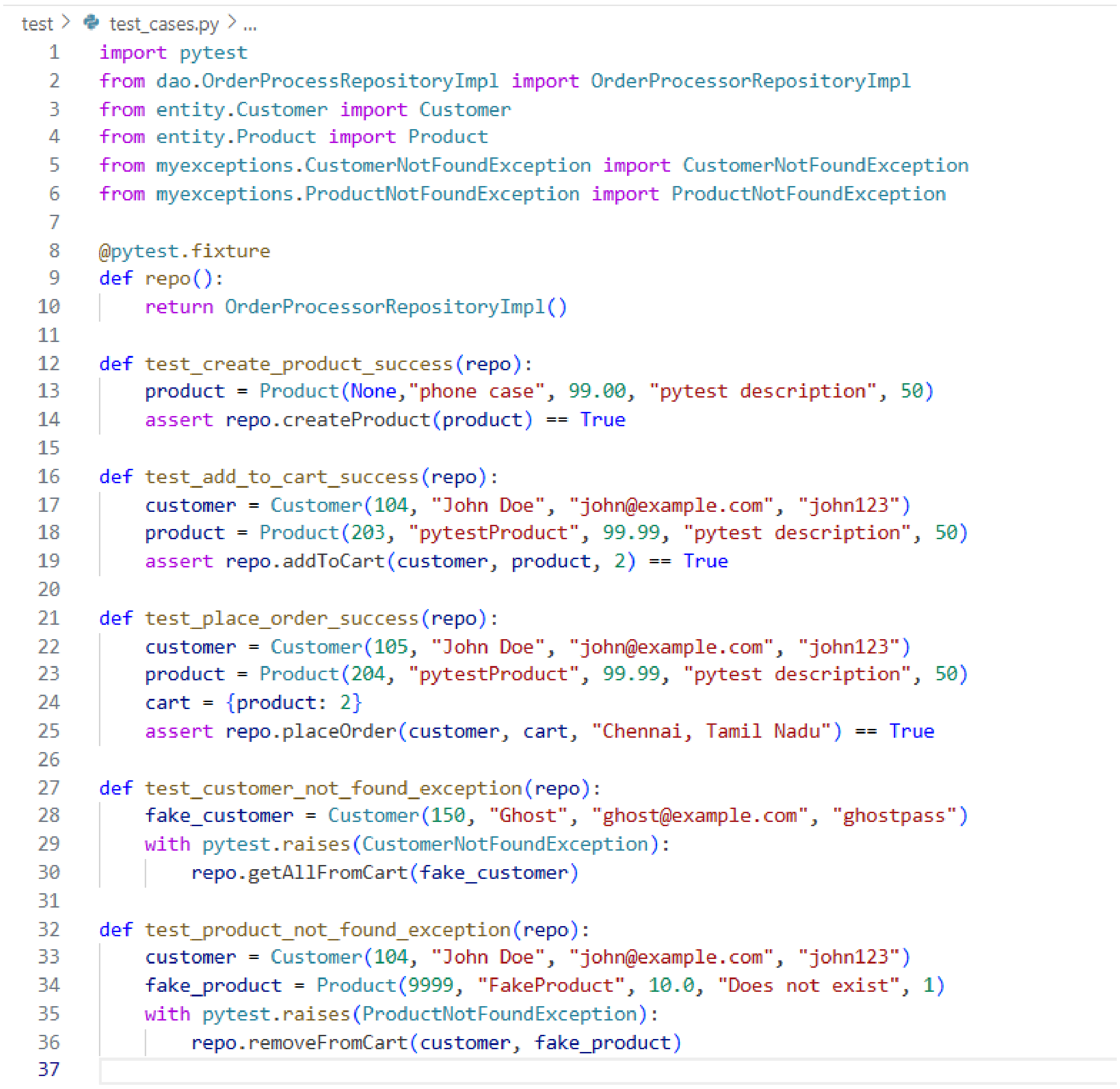
11. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

• Write test case to test Product created successfully or not.

• Write test case to test product is added to cart successfully or not.

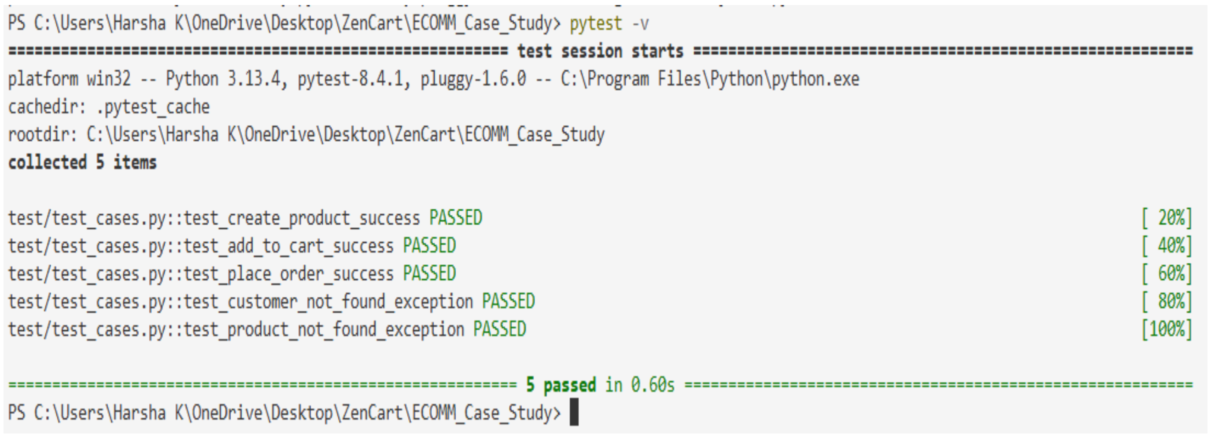
• Write test case to test product is ordered successfully or not.

• write test case to test exception is thrown correctly or not when customer id or product id not found in database.



```python
import pytest
from dao.OrderProcessRepositoryImpl import OrderProcessorRepositoryImpl
from entity.Customer import Customer
from entity.Product import Product
from myexceptions.CustomerNotFoundException import CustomerNotFoundException
from myexceptions.ProductNotFoundException import ProductNotFoundException

@pytest.fixture
def repo():
    return OrderProcessorRepositoryImpl()

def test_create_product_success(repo):
    product = Product(None,"phone case", 99.00, "pytest description", 50)
    assert repo.createProduct(product) == True

def test_add_to_cart_success(repo):
    customer = Customer(104, "John Doe", "john@example.com", "john123")
    product = Product(203, "pytestProduct", 99.99, "pytest description", 50)
    assert repo.addToCart(customer, product, 2) == True

def test_place_order_success(repo):
    customer = Customer(105, "John Doe", "john@example.com", "john123")
    product = Product(204, "pytestProduct", 99.99, "pytest description", 50)
    cart = {product: 2}
    assert repo.placeOrder(customer, cart, "Chennai, Tamil Nadu") == True

def test_customer_not_found_exception(repo):
    fake_customer = Customer(150, "Ghost", "ghost@example.com", "ghostpass")
    with pytest.raises(CustomerNotFoundException):
        repo.getAllFromCart(fake_customer)

def test_product_not_found_exception(repo):
    customer = Customer(104, "John Doe", "john@example.com", "john123")
    fake_product = Product(9999, "FakeProduct", 10.0, "Does not exist", 1)
    with pytest.raises(ProductNotFoundException):
        repo.removeFromCart(customer, fake_product)
```

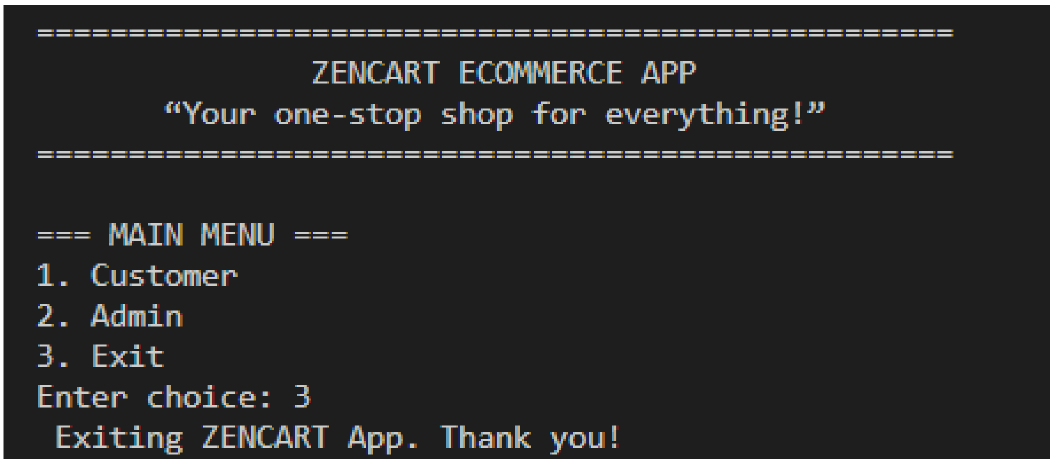
Test case output: 5 passed



```
PS C:\Users\Harsha K\OneDrive\Desktop\ZenCart\ECOMM_Case_Study> pytest -v
========================================================== test session starts ==========================================================
platform win32 -- Python 3.13.4, pytest-8.4.1, pluggy-1.6.0 -- C:\Program Files\Python\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Harsha K\OneDrive\Desktop\ZenCart\ECOMM_Case_Study
collected 5 items

test/test_cases.py::test_create_product_success PASSED                                                                          [ 20%]
test/test_cases.py::test_add_to_cart_success PASSED                                                                             [ 40%]
test/test_cases.py::test_place_order_success PASSED                                                                             [ 60%]
test/test_cases.py::test_customer_not_found_exception PASSED                                                                    [ 80%]
test/test_cases.py::test_product_not_found_exception PASSED                                                                     [100%]

=========================================================== 5 passed in 0.60s ===========================================================
PS C:\Users\Harsha K\OneDrive\Desktop\ZenCart\ECOMM_Case_Study>
```

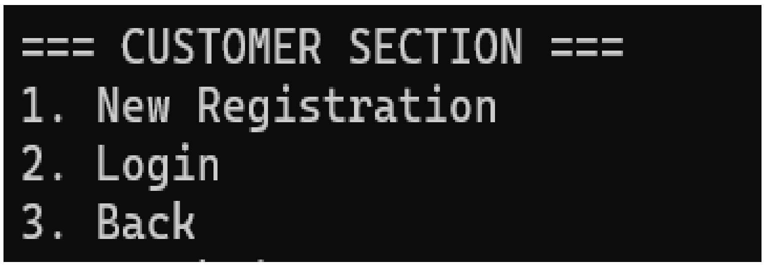
OUTPUT:

1.Main menu



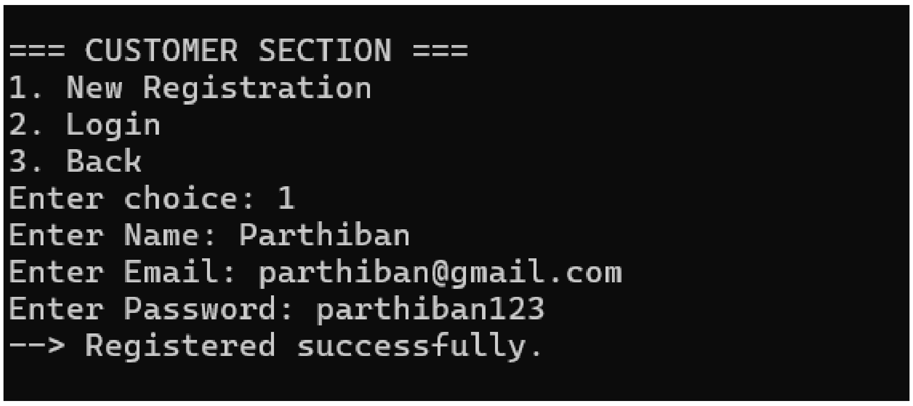
```
==================================================
                ZENCART ECOMMERCE APP
        “Your one-stop shop for everything!”
==================================================

=== MAIN MENU ===
1. Customer
2. Admin
3. Exit
Enter choice: 3
 Exiting ZENCART App. Thank you!
```

2.Customer Login Panel:



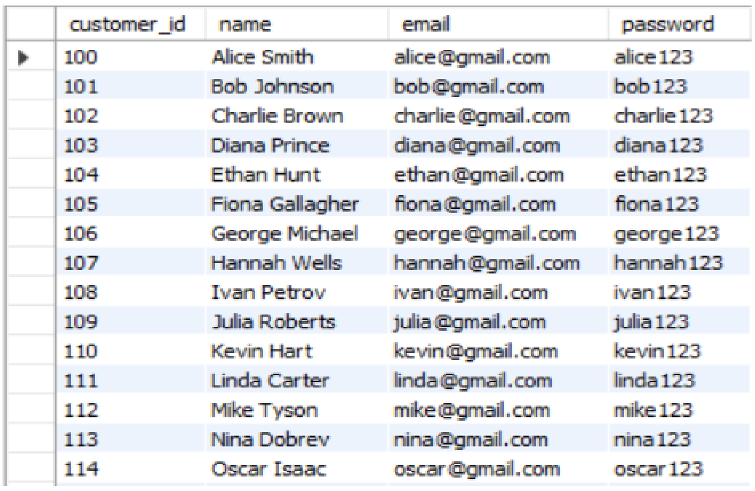
```
=== CUSTOMER SECTION ===
1. New Registration
2. Login
3. Back
```

3.New Registration:



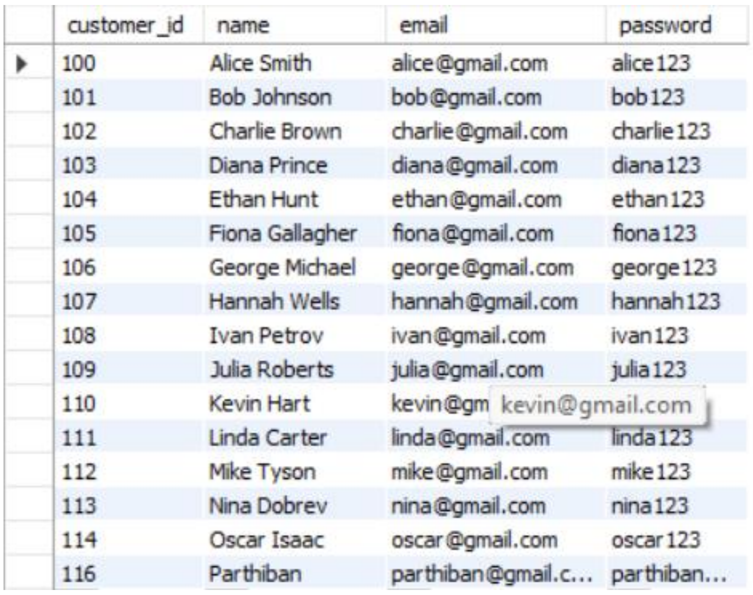
```
=== CUSTOMER SECTION ===
1. New Registration
2. Login
3. Back
Enter choice: 1
Enter Name: Parthiban
Enter Email: parthiban@gmail.com
Enter Password: parthiban123
--> Registered successfully.
```

Customers Table Before:



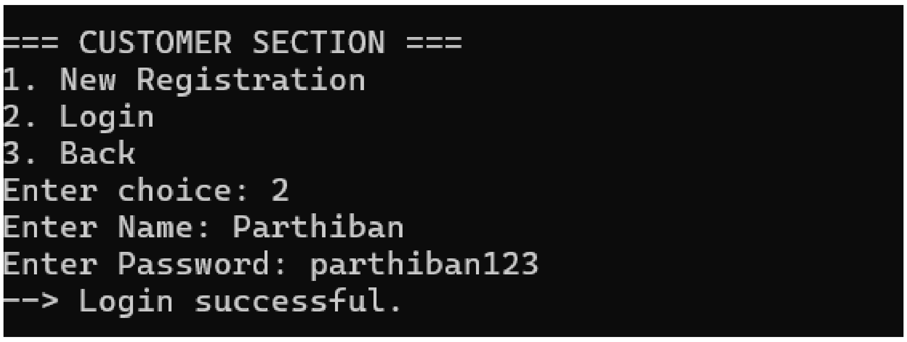
	customer_id	name	email	password
▶	100	Alice Smith	alice@gmail.com	alice123
	101	Bob Johnson	bob@gmail.com	bob123
	102	Charlie Brown	charlie@gmail.com	charlie123
	103	Diana Prince	diana@gmail.com	diana123
	104	Ethan Hunt	ethan@gmail.com	ethan123
	105	Fiona Gallagher	fiona@gmail.com	fiona123
	106	George Michael	george@gmail.com	george123
	107	Hannah Wells	hannah@gmail.com	hannah123
	108	Ivan Petrov	ivan@gmail.com	ivan123
	109	Julia Roberts	julia@gmail.com	julia123
	110	Kevin Hart	kevin@gmail.com	kevin123
	111	Linda Carter	linda@gmail.com	linda123
	112	Mike Tyson	mike@gmail.com	mike123
	113	Nina Dobrev	nina@gmail.com	nina123
	114	Oscar Isaac	oscar@gmail.com	oscar123

Customer Table After:



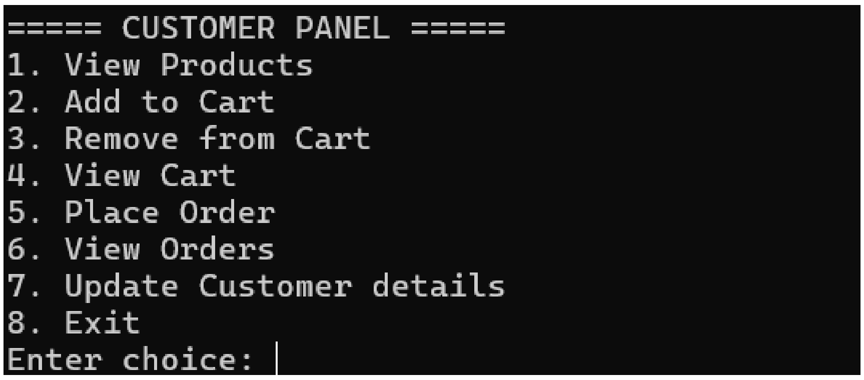
	customer_id	name	email	password
▶	100	Alice Smith	alice@gmail.com	alice123
	101	Bob Johnson	bob@gmail.com	bob123
	102	Charlie Brown	charlie@gmail.com	charlie123
	103	Diana Prince	diana@gmail.com	diana123
	104	Ethan Hunt	ethan@gmail.com	ethan123
	105	Fiona Gallagher	fiona@gmail.com	fiona123
	106	George Michael	george@gmail.com	george123
	107	Hannah Wells	hannah@gmail.com	hannah123
	108	Ivan Petrov	ivan@gmail.com	ivan123
	109	Julia Roberts	julia@gmail.com	julia123
	110	Kevin Hart	kevin@gm kevin@gmail.com	
	111	Linda Carter	linda@gmail.com	linda123
	112	Mike Tyson	mike@gmail.com	mike123
	113	Nina Dobrev	nina@gmail.com	nina123
	114	Oscar Isaac	oscar@gmail.com	oscar123
	116	Parthiban	parthiban@gmail.c...	parthiban...

4.Login:



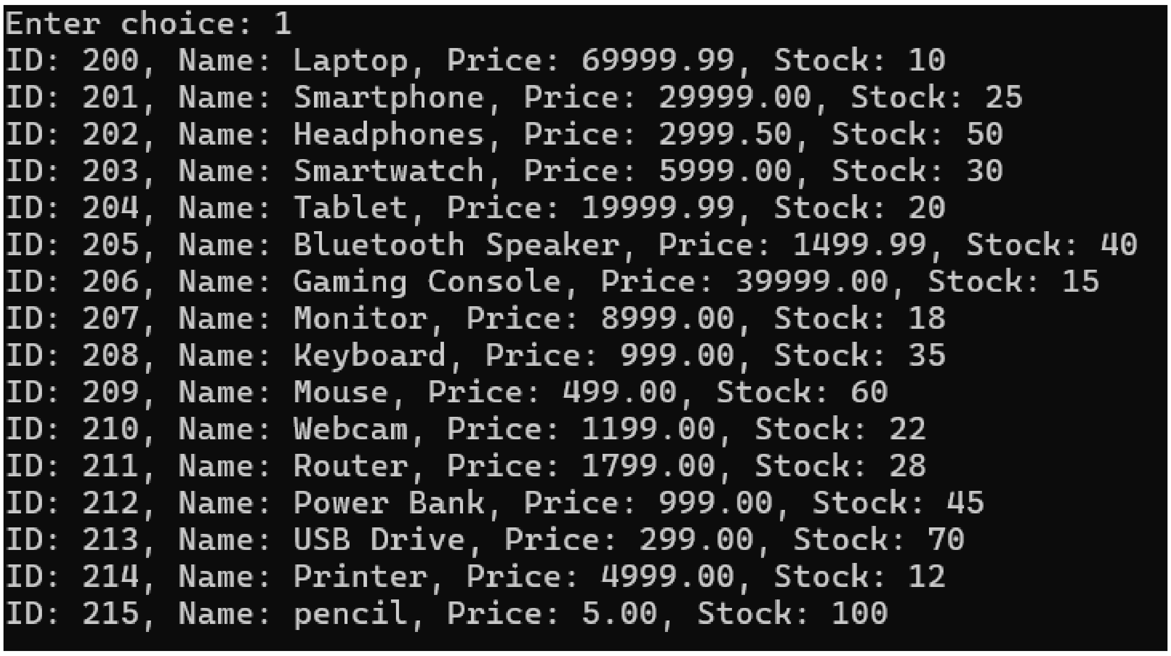
```
=== CUSTOMER SECTION ===
1. New Registration
2. Login
3. Back
Enter choice: 2
Enter Name: Parthiban
Enter Password: parthiban123
--> Login successful.
```

5. Customer Panel:



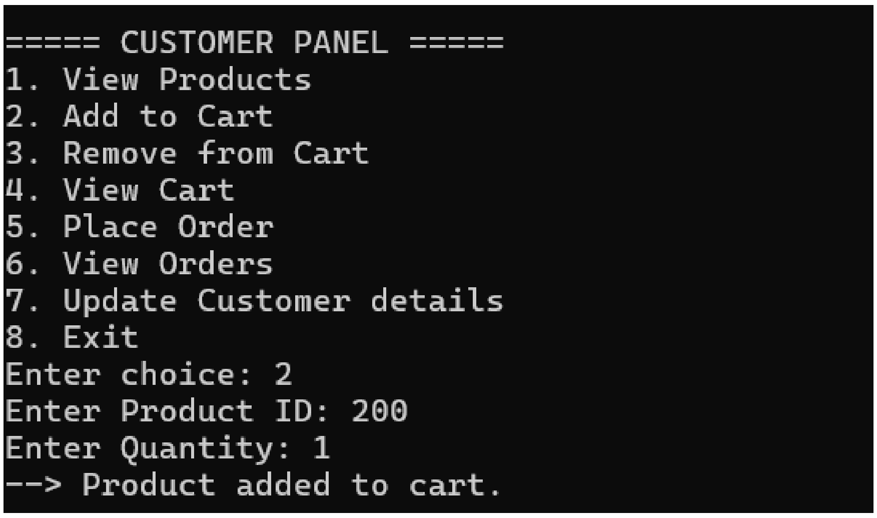
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: |
```

6.View Products:



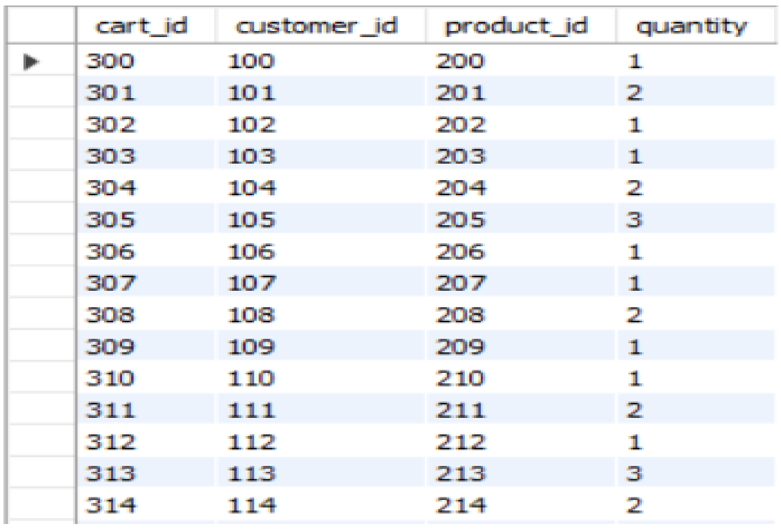
```
Enter choice: 1
ID: 200, Name: Laptop, Price: 69999.99, Stock: 10
ID: 201, Name: Smartphone, Price: 29999.00, Stock: 25
ID: 202, Name: Headphones, Price: 2999.50, Stock: 50
ID: 203, Name: Smartwatch, Price: 5999.00, Stock: 30
ID: 204, Name: Tablet, Price: 19999.99, Stock: 20
ID: 205, Name: Bluetooth Speaker, Price: 1499.99, Stock: 40
ID: 206, Name: Gaming Console, Price: 39999.00, Stock: 15
ID: 207, Name: Monitor, Price: 8999.00, Stock: 18
ID: 208, Name: Keyboard, Price: 999.00, Stock: 35
ID: 209, Name: Mouse, Price: 499.00, Stock: 60
ID: 210, Name: Webcam, Price: 1199.00, Stock: 22
ID: 211, Name: Router, Price: 1799.00, Stock: 28
ID: 212, Name: Power Bank, Price: 999.00, Stock: 45
ID: 213, Name: USB Drive, Price: 299.00, Stock: 70
ID: 214, Name: Printer, Price: 4999.00, Stock: 12
ID: 215, Name: pencil, Price: 5.00, Stock: 100
```

7.Add to Cart:



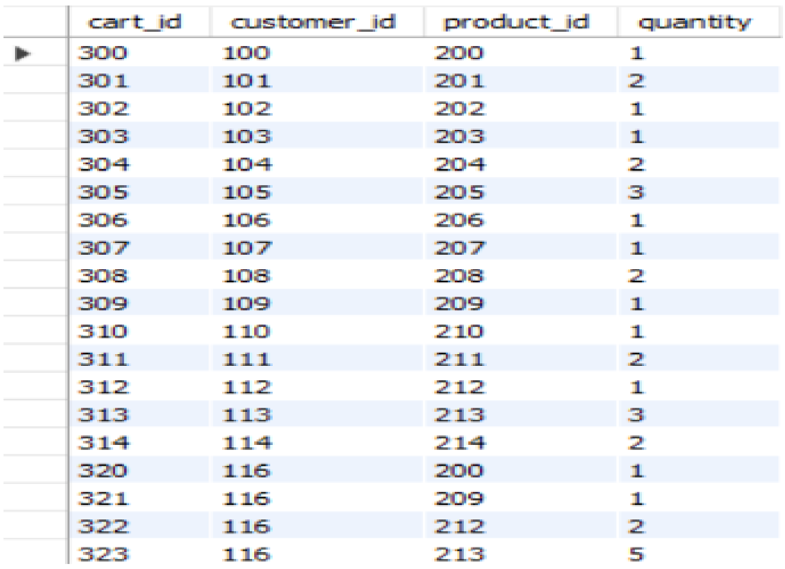
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 2
Enter Product ID: 200
Enter Quantity: 1
--> Product added to cart.
```

Cart Table Before:



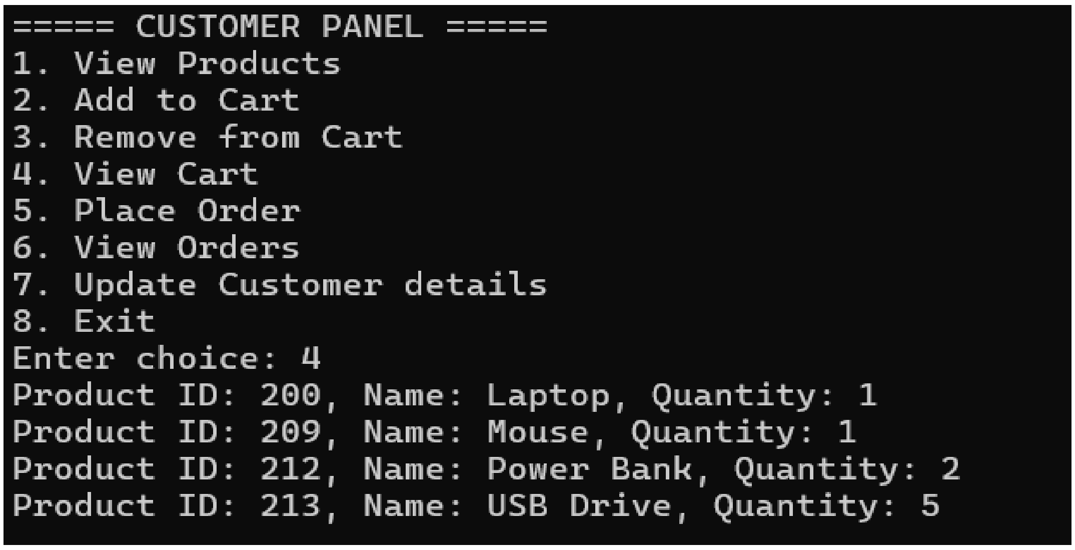
cart_id	customer_id	product_id	quantity
300	100	200	1
301	101	201	2
302	102	202	1
303	103	203	1
304	104	204	2
305	105	205	3
306	106	206	1
307	107	207	1
308	108	208	2
309	109	209	1
310	110	210	1
311	111	211	2
312	112	212	1
313	113	213	3
314	114	214	2

Cart Table After:



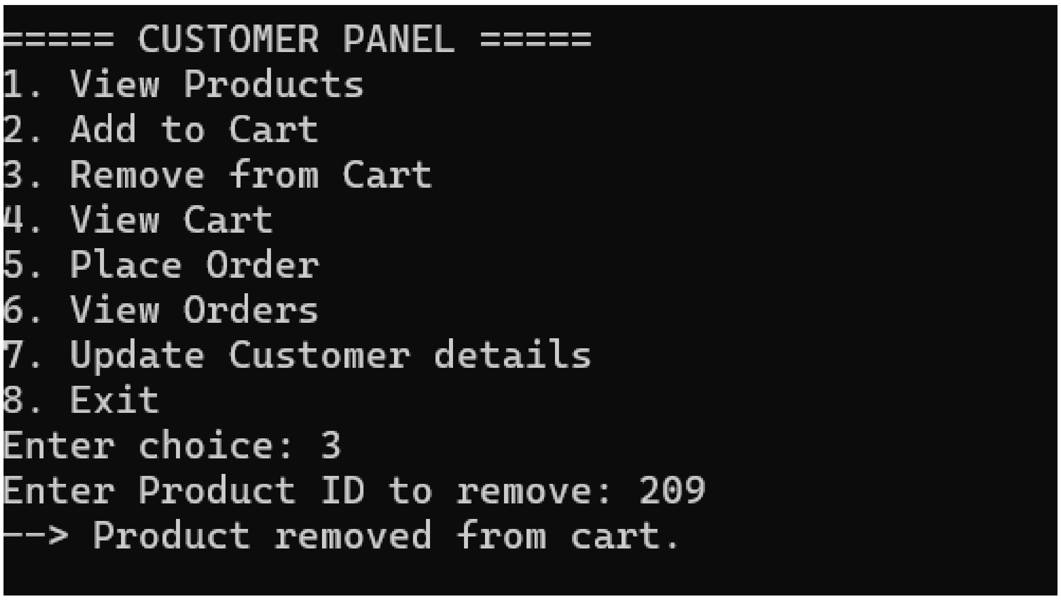
cart_id	customer_id	product_id	quantity
300	100	200	1
301	101	201	2
302	102	202	1
303	103	203	1
304	104	204	2
305	105	205	3
306	106	206	1
307	107	207	1
308	108	208	2
309	109	209	1
310	110	210	1
311	111	211	2
312	112	212	1
313	113	213	3
314	114	214	2
320	116	200	1
321	116	209	1
322	116	212	2
323	116	213	5

8. View Cart:



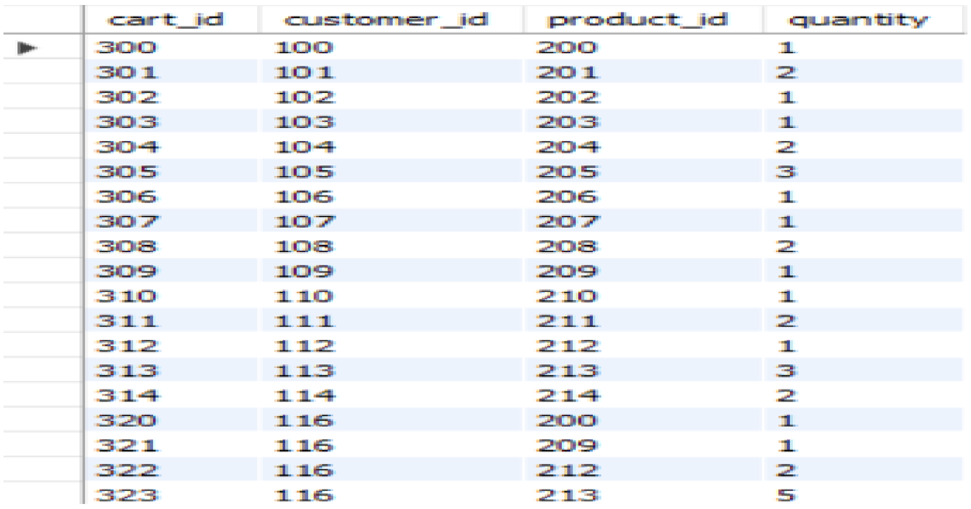
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 4
Product ID: 200, Name: Laptop, Quantity: 1
Product ID: 209, Name: Mouse, Quantity: 1
Product ID: 212, Name: Power Bank, Quantity: 2
Product ID: 213, Name: USB Drive, Quantity: 5
```

9.Remove from Cart:



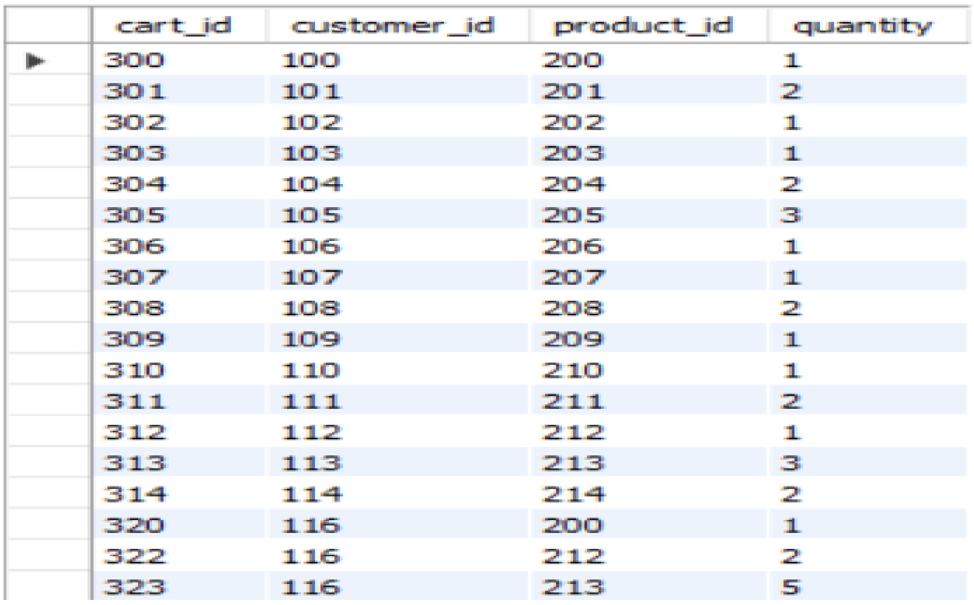
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 3
Enter Product ID to remove: 209
--> Product removed from cart.
```

Cart table Before:



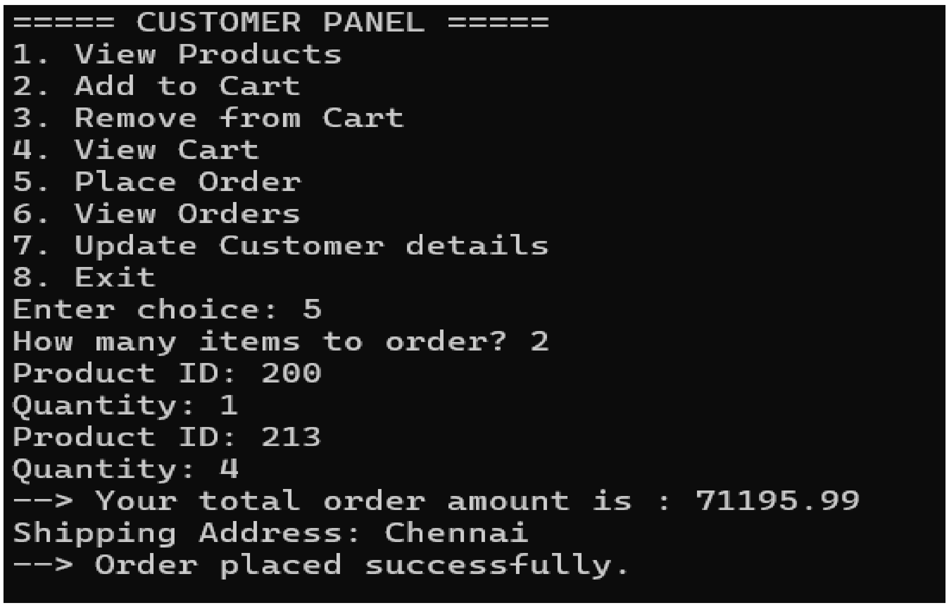
	cart_id	customer_id	product_id	quantity
▶	300	100	200	1
	301	101	201	2
	302	102	202	1
	303	103	203	1
	304	104	204	2
	305	105	205	3
	306	106	206	1
	307	107	207	1
	308	108	208	2
	309	109	209	1
	310	110	210	1
	311	111	211	2
	312	112	212	1
	313	113	213	3
	314	114	214	2
	320	116	200	1
	321	116	209	1
	322	116	212	2
	323	116	213	5

Cart table After:



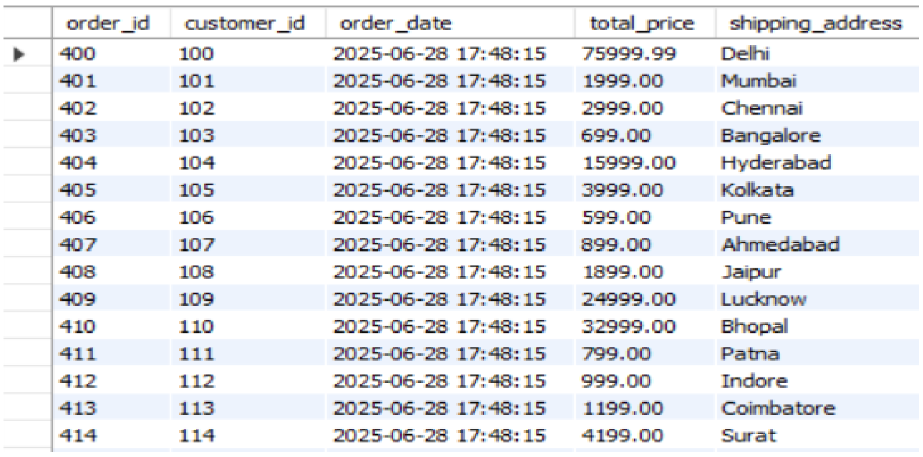
	cart_id	customer_id	product_id	quantity
▶	300	100	200	1
	301	101	201	2
	302	102	202	1
	303	103	203	1
	304	104	204	2
	305	105	205	3
	306	106	206	1
	307	107	207	1
	308	108	208	2
	309	109	209	1
	310	110	210	1
	311	111	211	2
	312	112	212	1
	313	113	213	3
	314	114	214	2
	320	116	200	1
	322	116	212	2
	323	116	213	5

10. Place Order:



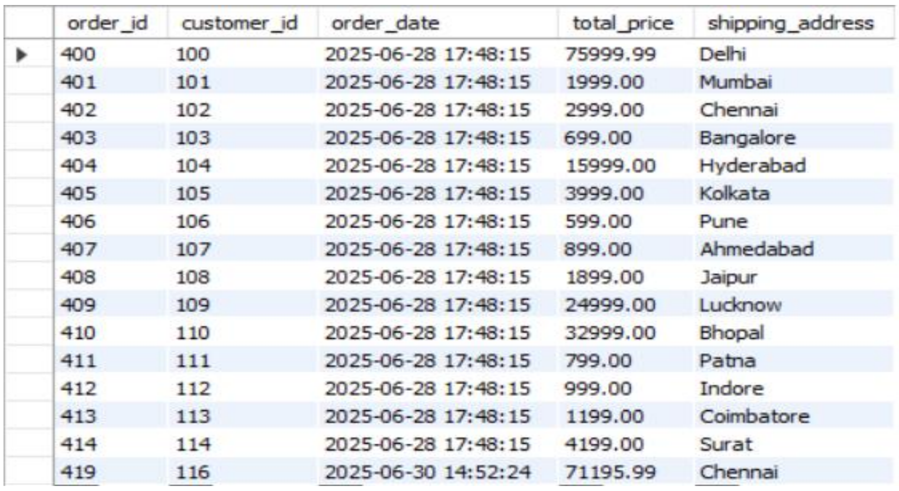
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 5
How many items to order? 2
Product ID: 200
Quantity: 1
Product ID: 213
Quantity: 4
--> Your total order amount is : 71195.99
Shipping Address: Chennai
--> Order placed successfully.
```

Orders Table Before:



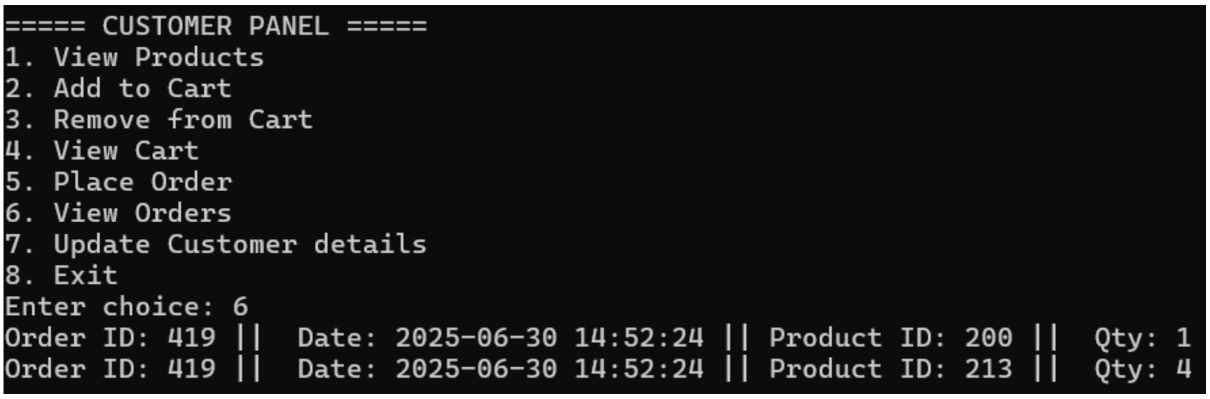
order_id	customer_id	order_date	total_price	shipping_address
400	100	2025-06-28 17:48:15	75999.99	Delhi
401	101	2025-06-28 17:48:15	1999.00	Mumbai
402	102	2025-06-28 17:48:15	2999.00	Chennai
403	103	2025-06-28 17:48:15	699.00	Bangalore
404	104	2025-06-28 17:48:15	15999.00	Hyderabad
405	105	2025-06-28 17:48:15	3999.00	Kolkata
406	106	2025-06-28 17:48:15	599.00	Pune
407	107	2025-06-28 17:48:15	899.00	Ahmedabad
408	108	2025-06-28 17:48:15	1899.00	Jaipur
409	109	2025-06-28 17:48:15	24999.00	Lucknow
410	110	2025-06-28 17:48:15	32999.00	Bhopal
411	111	2025-06-28 17:48:15	799.00	Patna
412	112	2025-06-28 17:48:15	999.00	Indore
413	113	2025-06-28 17:48:15	1199.00	Coimbatore
414	114	2025-06-28 17:48:15	4199.00	Surat

Orders Table After:



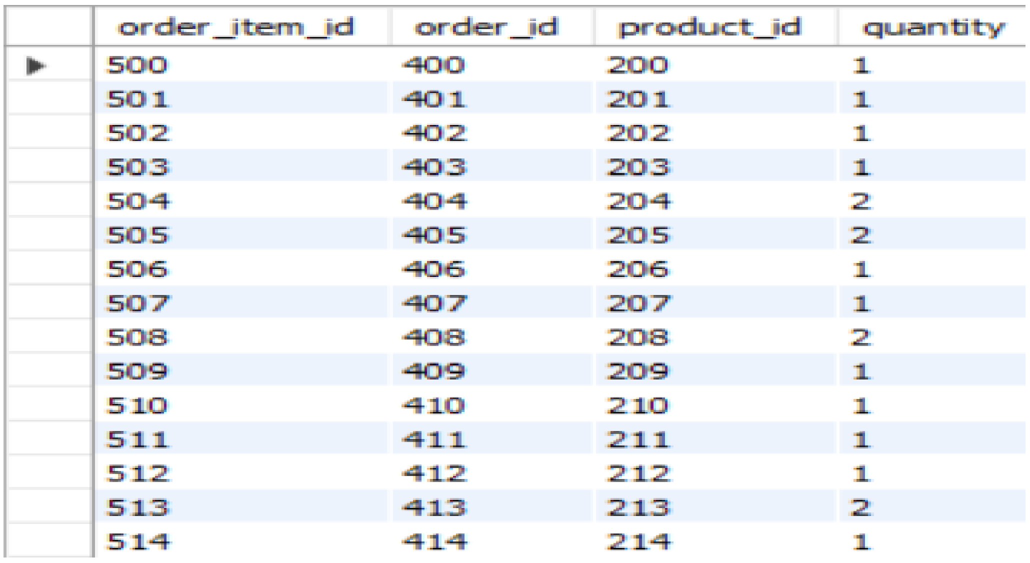
order_id	customer_id	order_date	total_price	shipping_address
400	100	2025-06-28 17:48:15	75999.99	Delhi
401	101	2025-06-28 17:48:15	1999.00	Mumbai
402	102	2025-06-28 17:48:15	2999.00	Chennai
403	103	2025-06-28 17:48:15	699.00	Bangalore
404	104	2025-06-28 17:48:15	15999.00	Hyderabad
405	105	2025-06-28 17:48:15	3999.00	Kolkata
406	106	2025-06-28 17:48:15	599.00	Pune
407	107	2025-06-28 17:48:15	899.00	Ahmedabad
408	108	2025-06-28 17:48:15	1899.00	Jaipur
409	109	2025-06-28 17:48:15	24999.00	Lucknow
410	110	2025-06-28 17:48:15	32999.00	Bhopal
411	111	2025-06-28 17:48:15	799.00	Patna
412	112	2025-06-28 17:48:15	999.00	Indore
413	113	2025-06-28 17:48:15	1199.00	Coimbatore
414	114	2025-06-28 17:48:15	4199.00	Surat
419	116	2025-06-30 14:52:24	71195.99	Chennai

11.View Orders:



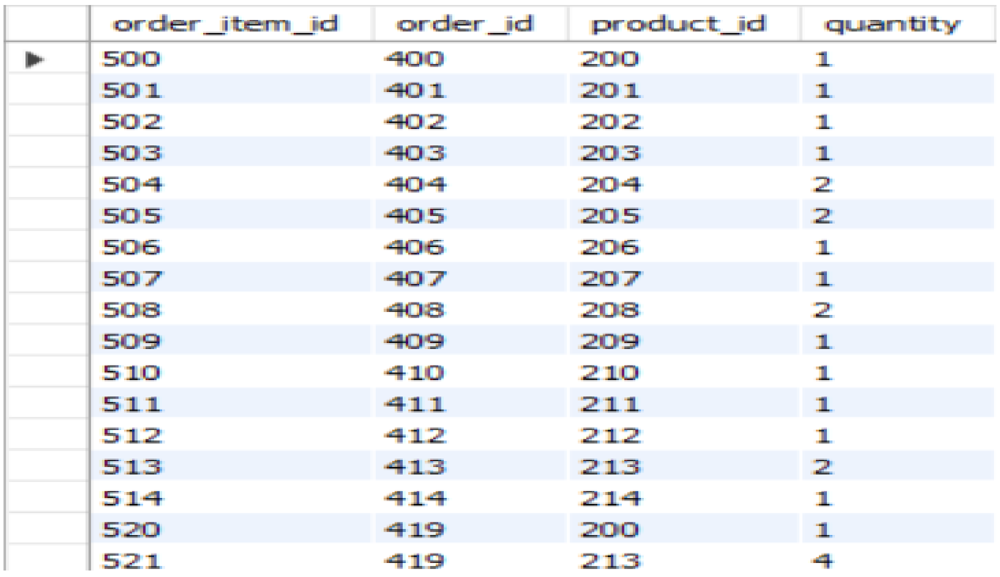
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 6
Order ID: 419 ||  Date: 2025-06-30 14:52:24 || Product ID: 200 ||  Qty: 1
Order ID: 419 ||  Date: 2025-06-30 14:52:24 || Product ID: 213 ||  Qty: 4
```

Order Items Table Before:



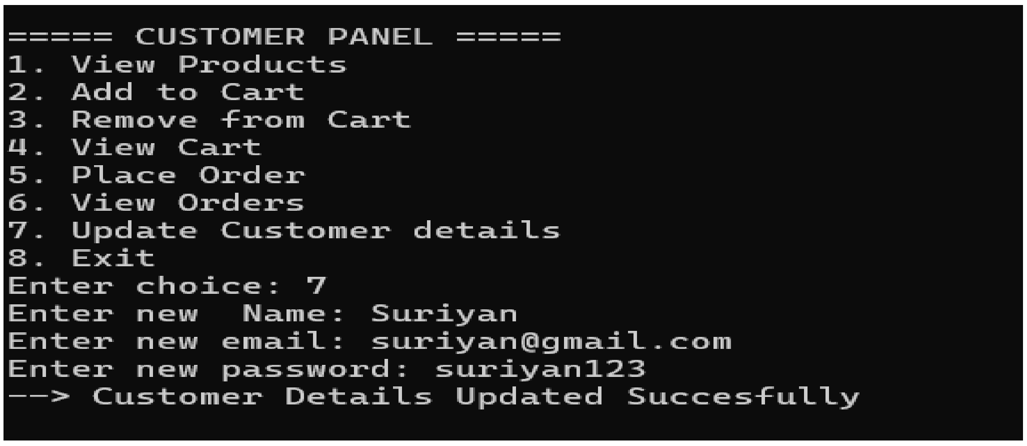
order_item_id	order_id	product_id	quantity
500	400	200	1
501	401	201	1
502	402	202	1
503	403	203	1
504	404	204	2
505	405	205	2
506	406	206	1
507	407	207	1
508	408	208	2
509	409	209	1
510	410	210	1
511	411	211	1
512	412	212	1
513	413	213	2
514	414	214	1

Order Items Table After:



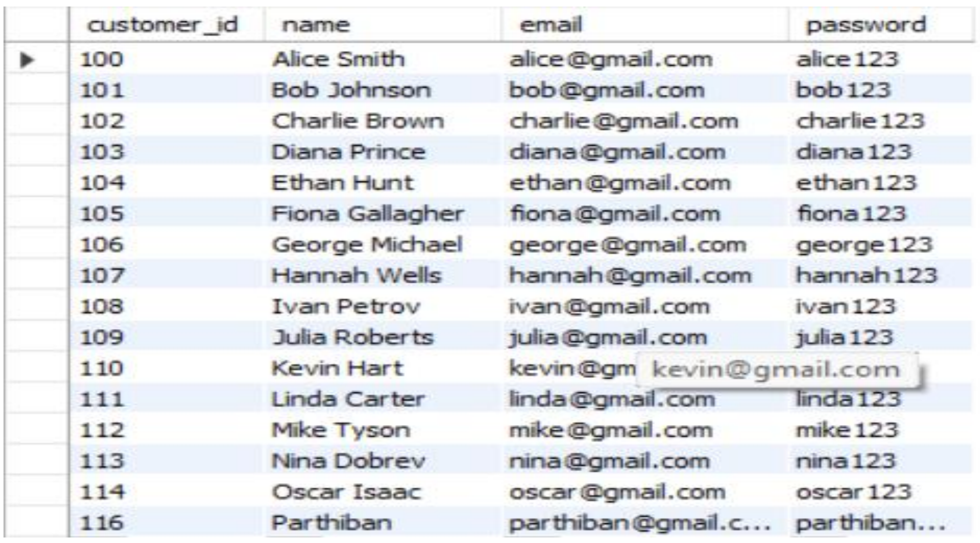
order_item_id	order_id	product_id	quantity
500	400	200	1
501	401	201	1
502	402	202	1
503	403	203	1
504	404	204	2
505	405	205	2
506	406	206	1
507	407	207	1
508	408	208	2
509	409	209	1
510	410	210	1
511	411	211	1
512	412	212	1
513	413	213	2
514	414	214	1
520	419	200	1
521	419	213	4

12. Update Customer details:



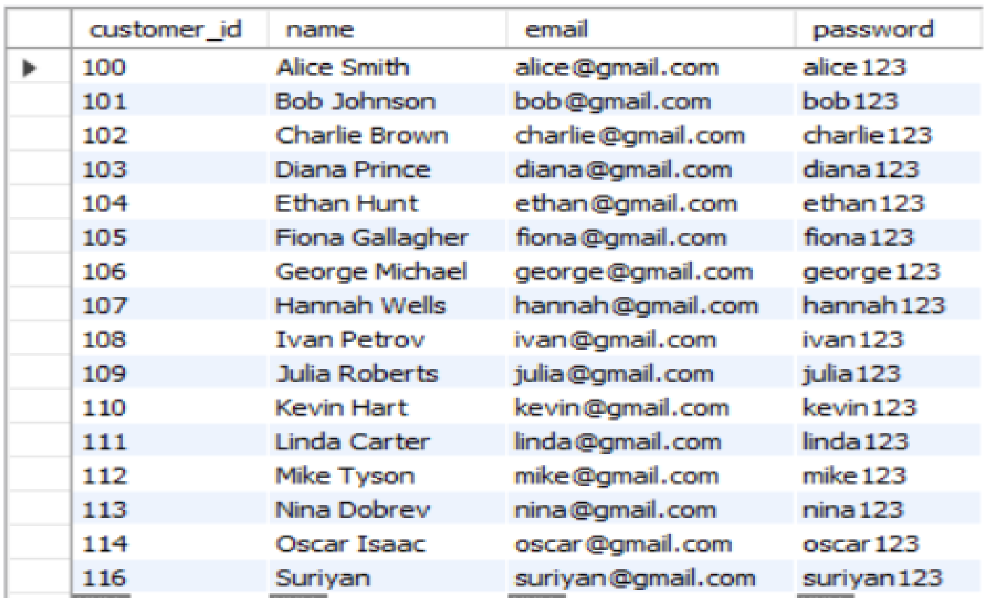
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 7
Enter new  Name: Suriyan
Enter new email: suriyan@gmail.com
Enter new password: suriyan123
--> Customer Details Updated Succesfully
```

Customers table Before:



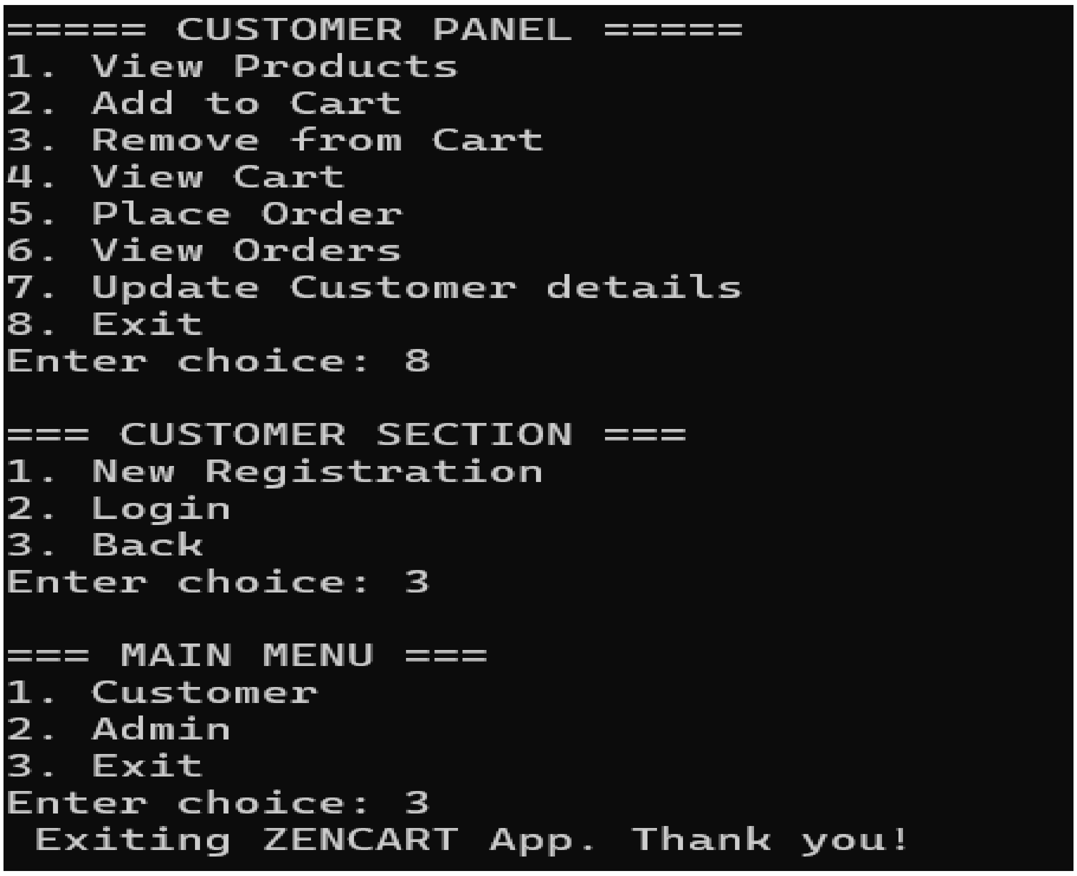
customer_id	name	email	password
100	Alice Smith	alice@gmail.com	alice123
101	Bob Johnson	bob@gmail.com	bob123
102	Charlie Brown	charlie@gmail.com	charlie123
103	Diana Prince	diana@gmail.com	diana123
104	Ethan Hunt	ethan@gmail.com	ethan123
105	Fiona Gallagher	fiona@gmail.com	fiona123
106	George Michael	george@gmail.com	george123
107	Hannah Wells	hannah@gmail.com	hannah123
108	Ivan Petrov	ivan@gmail.com	ivan123
109	Julia Roberts	julia@gmail.com	julia123
110	Kevin Hart	kevin@gm kevin@gmail.com	
111	Linda Carter	linda@gmail.com	linda123
112	Mike Tyson	mike@gmail.com	mike123
113	Nina Dobrev	nina@gmail.com	nina123
114	Oscar Isaac	oscar@gmail.com	oscar123
116	Parthiban	parthiban@gmail.c...	parthiban...

Customers table After:



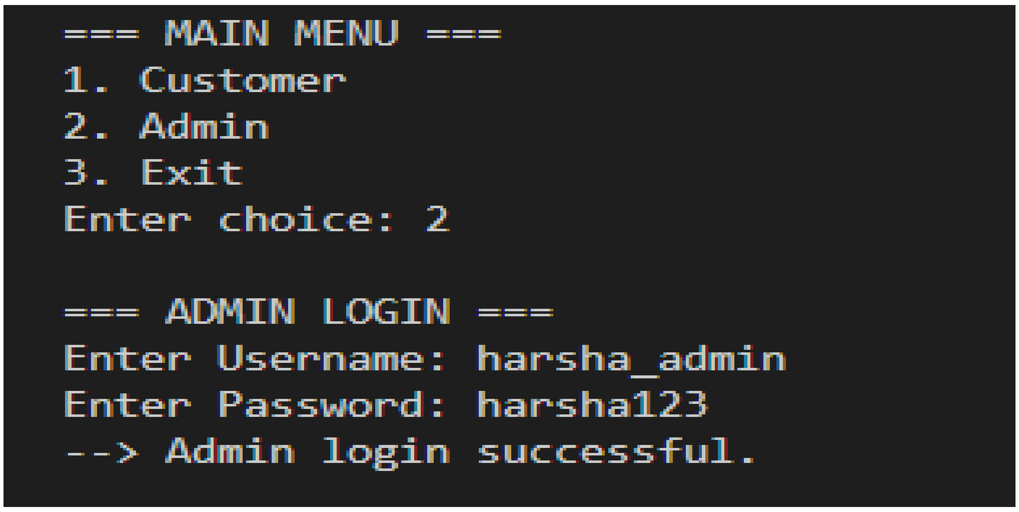
customer_id	name	email	password
100	Alice Smith	alice@gmail.com	alice123
101	Bob Johnson	bob@gmail.com	bob123
102	Charlie Brown	charlie@gmail.com	charlie123
103	Diana Prince	diana@gmail.com	diana123
104	Ethan Hunt	ethan@gmail.com	ethan123
105	Fiona Gallagher	fiona@gmail.com	fiona123
106	George Michael	george@gmail.com	george123
107	Hannah Wells	hannah@gmail.com	hannah123
108	Ivan Petrov	ivan@gmail.com	ivan123
109	Julia Roberts	julia@gmail.com	julia123
110	Kevin Hart	kevin@gmail.com	kevin123
111	Linda Carter	linda@gmail.com	linda123
112	Mike Tyson	mike@gmail.com	mike123
113	Nina Dobrev	nina@gmail.com	nina123
114	Oscar Isaac	oscar@gmail.com	oscar123
116	Suriyan	suriyan@gmail.com	suriyan123

13.Exit:



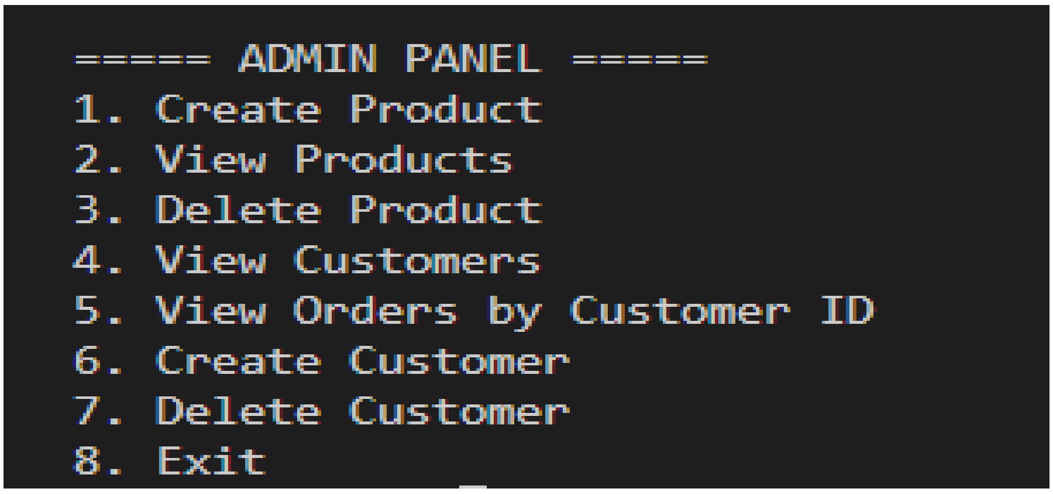
```
===== CUSTOMER PANEL =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Orders
7. Update Customer details
8. Exit
Enter choice: 8

=== CUSTOMER SECTION ===
1. New Registration
2. Login
3. Back
Enter choice: 3

=== MAIN MENU ===
1. Customer
2. Admin
3. Exit
Enter choice: 3
 Exiting ZENCART App. Thank you!
```

14.Admin Login verification



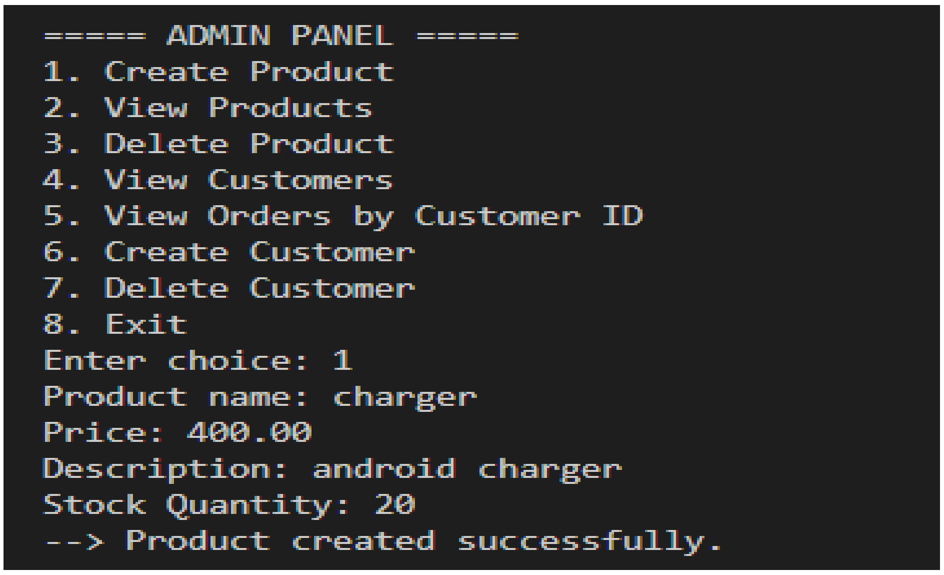
```
=== MAIN MENU ===
1. Customer
2. Admin
3. Exit
Enter choice: 2

=== ADMIN LOGIN ===
Enter Username: harsha_admin
Enter Password: harsha123
--> Admin login successful.
```

15.Admin specific Activities



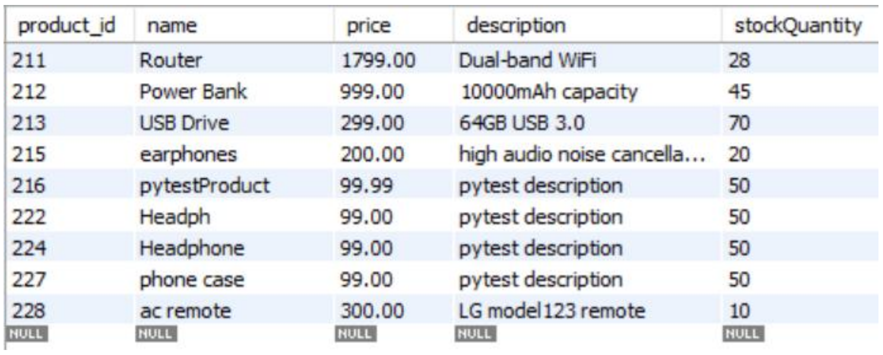
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
```

16.Create Products



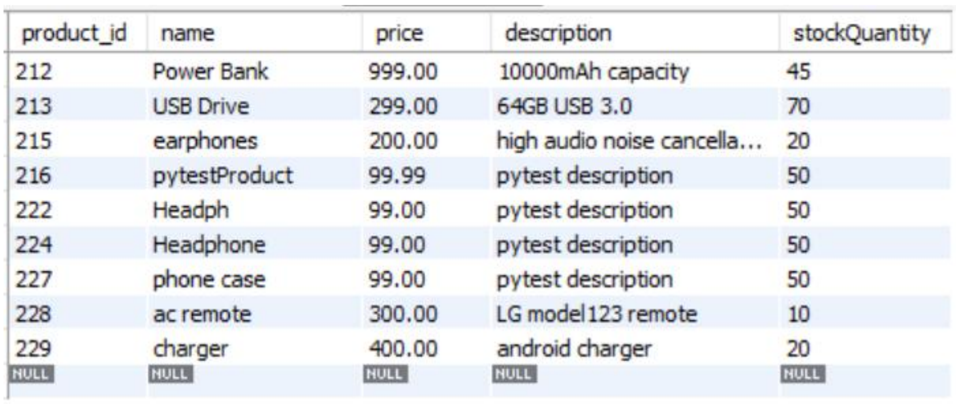
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 1
Product name: charger
Price: 400.00
Description: android charger
Stock Quantity: 20
--> Product created successfully.
```

Before:



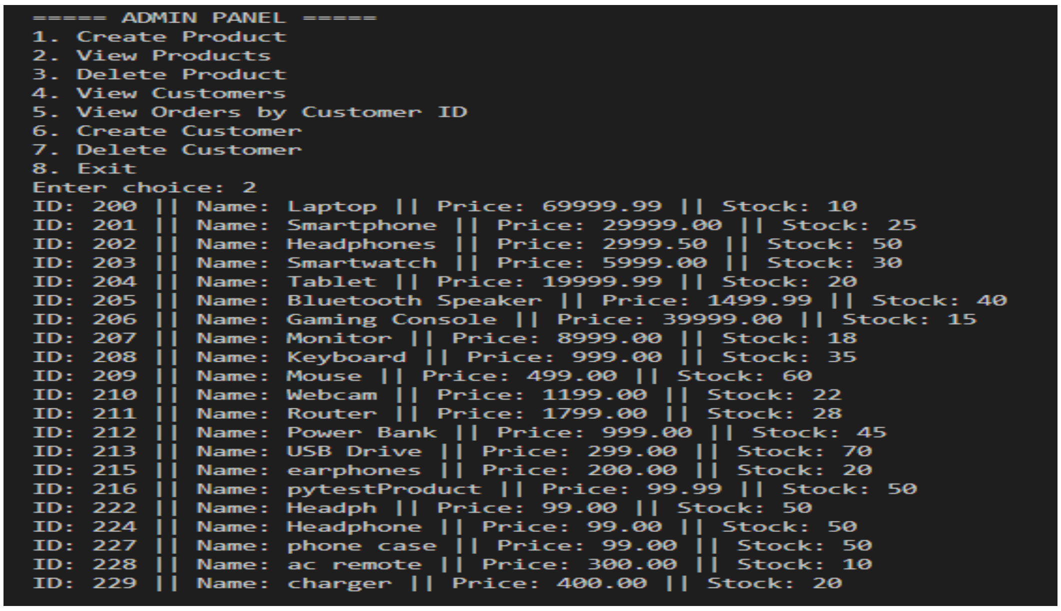
product_id	name	price	description	stockQuantity
211	Router	1799.00	Dual-band WiFi	28
212	Power Bank	999.00	10000mAh capacity	45
213	USB Drive	299.00	64GB USB 3.0	70
215	earphones	200.00	high audio noise cancella...	20
216	pytestProduct	99.99	pytest description	50
222	Headph	99.00	pytest description	50
224	Headphone	99.00	pytest description	50
227	phone case	99.00	pytest description	50
228	ac remote	300.00	LG model123 remote	10
NULL	NULL	NULL	NULL	NULL

After:



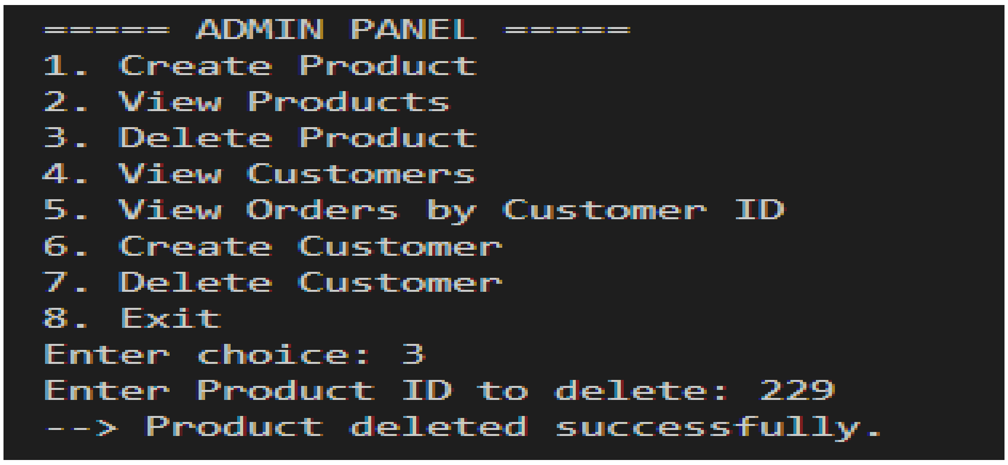
product_id	name	price	description	stockQuantity
212	Power Bank	999.00	10000mAh capacity	45
213	USB Drive	299.00	64GB USB 3.0	70
215	earphones	200.00	high audio noise cancella...	20
216	pytestProduct	99.99	pytest description	50
222	Headph	99.00	pytest description	50
224	Headphone	99.00	pytest description	50
227	phone case	99.00	pytest description	50
228	ac remote	300.00	LG model123 remote	10
229	charger	400.00	android charger	20
NULL	NULL	NULL	NULL	NULL

17.View Products



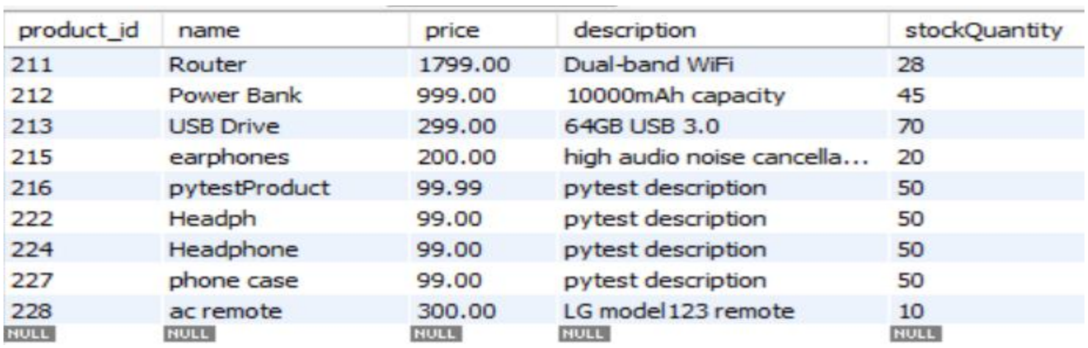
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 2
ID: 200 || Name: Laptop || Price: 69999.99 || Stock: 10
ID: 201 || Name: Smartphone || Price: 29999.00 || Stock: 25
ID: 202 || Name: Headphones || Price: 2999.50 || Stock: 50
ID: 203 || Name: Smartwatch || Price: 5999.00 || Stock: 30
ID: 204 || Name: Tablet || Price: 19999.99 || Stock: 20
ID: 205 || Name: Bluetooth Speaker || Price: 1499.99 || Stock: 40
ID: 206 || Name: Gaming Console || Price: 39999.00 || Stock: 15
ID: 207 || Name: Monitor || Price: 8999.00 || Stock: 18
ID: 208 || Name: Keyboard || Price: 999.00 || Stock: 35
ID: 209 || Name: Mouse || Price: 499.00 || Stock: 60
ID: 210 || Name: Webcam || Price: 1199.00 || Stock: 22
ID: 211 || Name: Router || Price: 1799.00 || Stock: 28
ID: 212 || Name: Power Bank || Price: 999.00 || Stock: 45
ID: 213 || Name: USB Drive || Price: 299.00 || Stock: 70
ID: 215 || Name: earphones || Price: 200.00 || Stock: 20
ID: 216 || Name: pytestProduct || Price: 99.99 || Stock: 50
ID: 222 || Name: Headph || Price: 99.00 || Stock: 50
ID: 224 || Name: Headphone || Price: 99.00 || Stock: 50
ID: 227 || Name: phone case || Price: 99.00 || Stock: 50
ID: 228 || Name: ac remote || Price: 300.00 || Stock: 10
ID: 229 || Name: charger || Price: 400.00 || Stock: 20
```

18.Delete Products



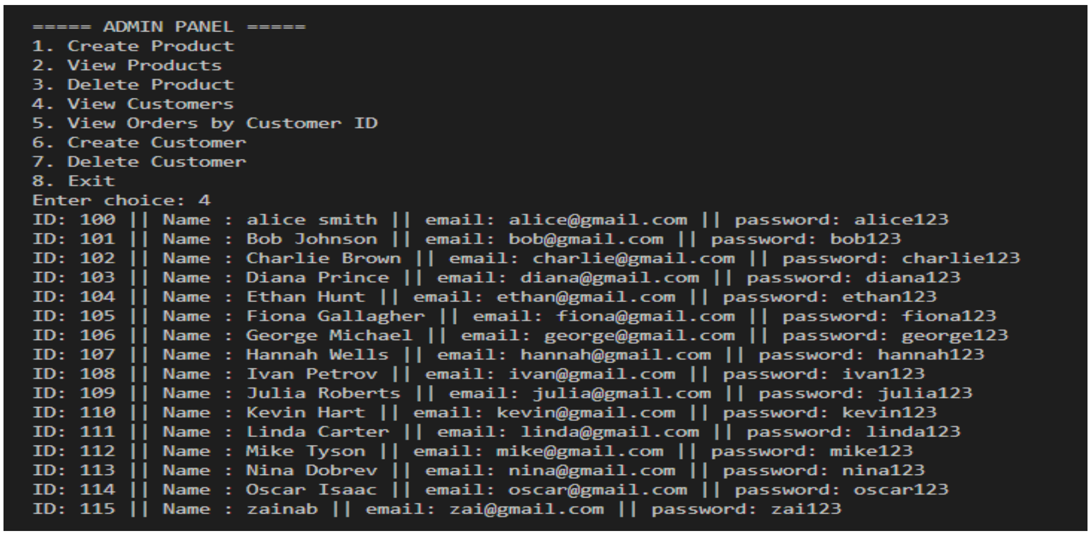
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 3
Enter Product ID to delete: 229
--> Product deleted successfully.
```

Product with id 229 deleted



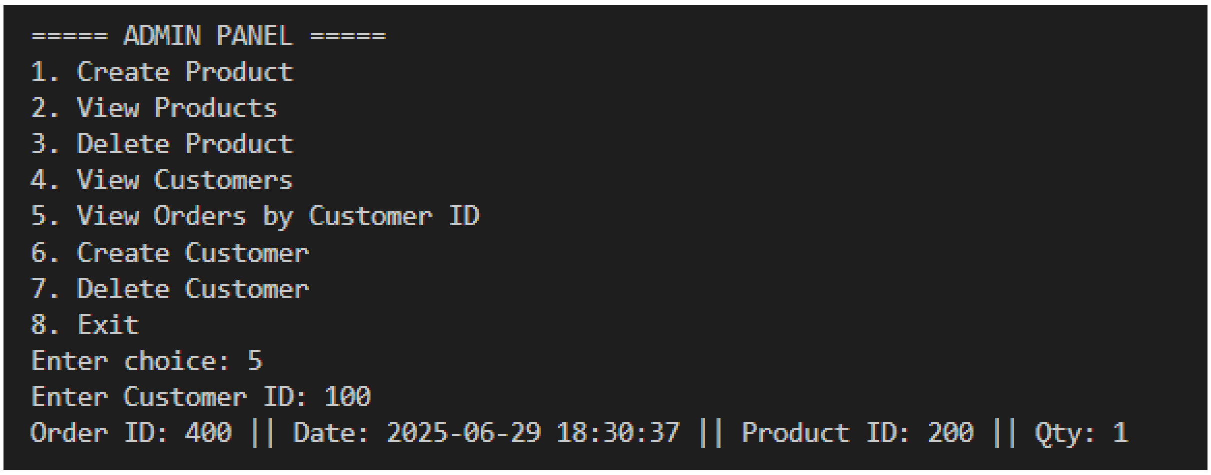
product_id	name	price	description	stockQuantity
211	Router	1799.00	Dual-band WiFi	28
212	Power Bank	999.00	10000mAh capacity	45
213	USB Drive	299.00	64GB USB 3.0	70
215	earphones	200.00	high audio noise cancella...	20
216	pytestProduct	99.99	pytest description	50
222	Headph	99.00	pytest description	50
224	Headphone	99.00	pytest description	50
227	phone case	99.00	pytest description	50
228	ac remote	300.00	LG model123 remote	10
NULL	NULL	NULL	NULL	NULL

19.View Customers



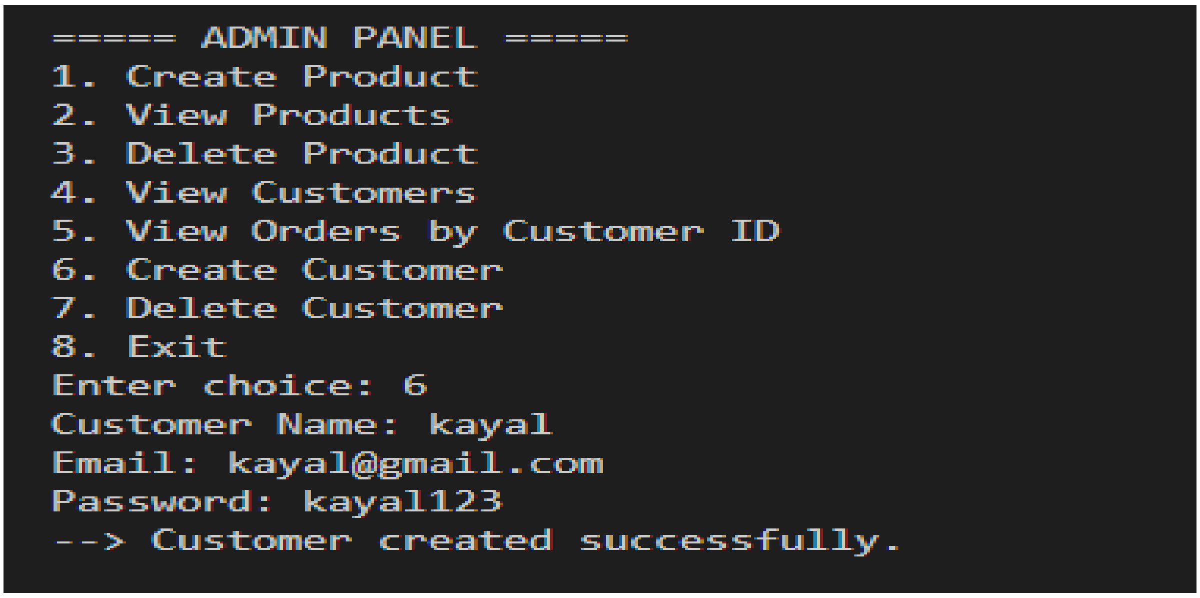
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 4
ID: 100 || Name : alice smith || email: alice@gmail.com || password: alice123
ID: 101 || Name : Bob Johnson || email: bob@gmail.com || password: bob123
ID: 102 || Name : Charlie Brown || email: charlie@gmail.com || password: charlie123
ID: 103 || Name : Diana Prince || email: diana@gmail.com || password: diana123
ID: 104 || Name : Ethan Hunt || email: ethan@gmail.com || password: ethan123
ID: 105 || Name : Fiona Gallagher || email: fiona@gmail.com || password: fiona123
ID: 106 || Name : George Michael || email: george@gmail.com || password: george123
ID: 107 || Name : Hannah Wells || email: hannah@gmail.com || password: hannah123
ID: 108 || Name : Ivan Petrov || email: ivan@gmail.com || password: ivan123
ID: 109 || Name : Julia Roberts || email: julia@gmail.com || password: julia123
ID: 110 || Name : Kevin Hart || email: kevin@gmail.com || password: kevin123
ID: 111 || Name : Linda Carter || email: linda@gmail.com || password: linda123
ID: 112 || Name : Mike Tyson || email: mike@gmail.com || password: mike123
ID: 113 || Name : Nina Dobrev || email: nina@gmail.com || password: nina123
ID: 114 || Name : Oscar Isaac || email: oscar@gmail.com || password: oscar123
ID: 115 || Name : zainab || email: zai@gmail.com || password: zai123
```

20.View Orders by Customer ID



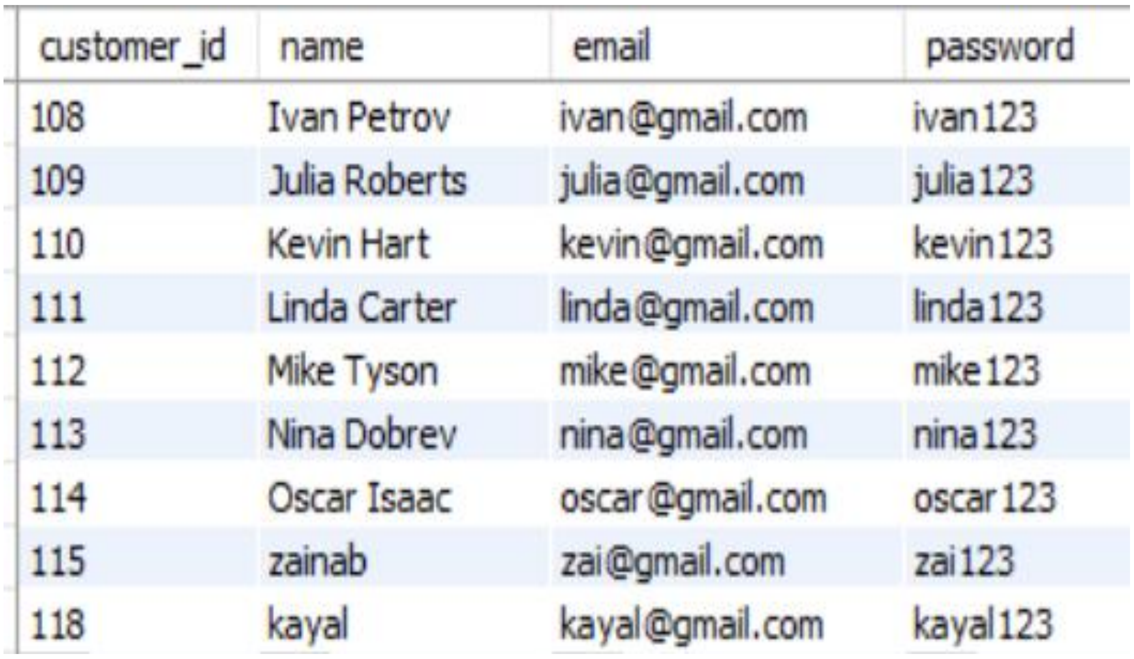
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 5
Enter Customer ID: 100
Order ID: 400 || Date: 2025-06-29 18:30:37 || Product ID: 200 || Qty: 1
```

21.Create Customer



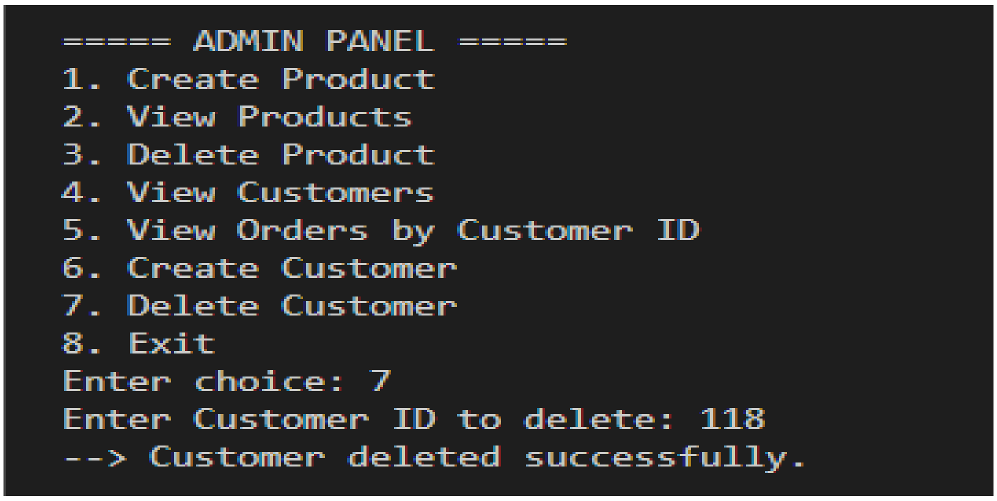
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 6
Customer Name: kayal
Email: kayal@gmail.com
Password: kayal123
--> Customer created successfully.
```

New customer “kayal” added



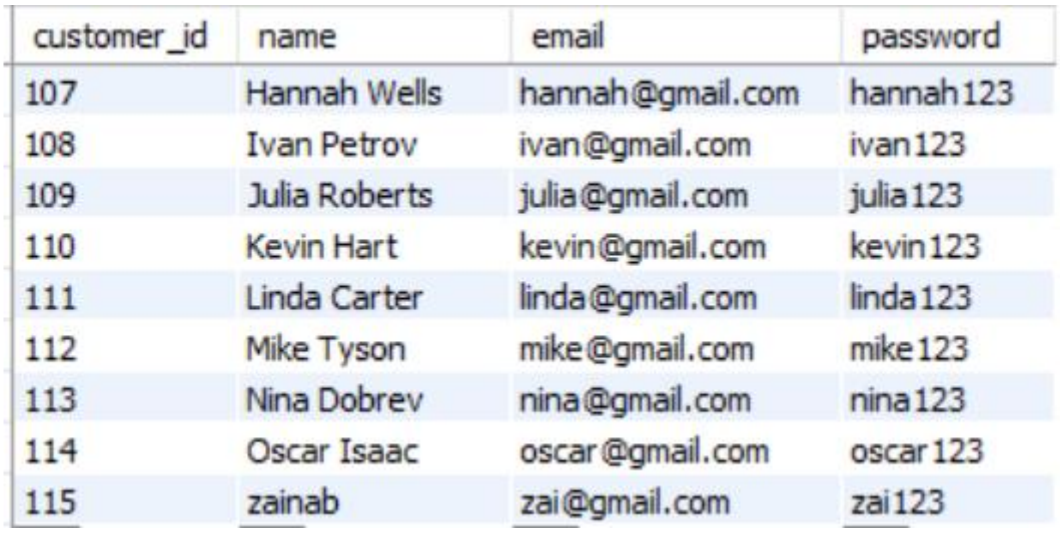
customer_id	name	email	password
108	Ivan Petrov	ivan@gmail.com	ivan123
109	Julia Roberts	julia@gmail.com	julia123
110	Kevin Hart	kevin@gmail.com	kevin123
111	Linda Carter	linda@gmail.com	linda123
112	Mike Tyson	mike@gmail.com	mike123
113	Nina Dobrev	nina@gmail.com	nina123
114	Oscar Isaac	oscar@gmail.com	oscar123
115	zainab	zai@gmail.com	zai123
118	kayal	kayal@gmail.com	kayal123

22.Delete Customer



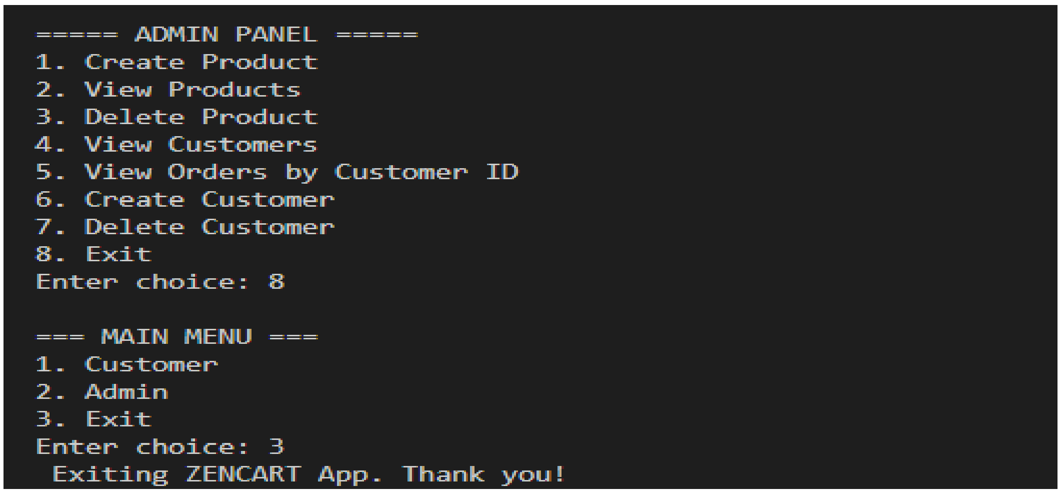
```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 7
Enter Customer ID to delete: 118
--> Customer deleted successfully.
```

Customer “kayal” deleted



customer_id	name	email	password
107	Hannah Wells	hannah@gmail.com	hannah123
108	Ivan Petrov	ivan@gmail.com	ivan123
109	Julia Roberts	julia@gmail.com	julia123
110	Kevin Hart	kevin@gmail.com	kevin123
111	Linda Carter	linda@gmail.com	linda123
112	Mike Tyson	mike@gmail.com	mike123
113	Nina Dobrev	nina@gmail.com	nina123
114	Oscar Isaac	oscar@gmail.com	oscar123
115	zainab	zai@gmail.com	zai123

23.Exit



```
===== ADMIN PANEL =====
1. Create Product
2. View Products
3. Delete Product
4. View Customers
5. View Orders by Customer ID
6. Create Customer
7. Delete Customer
8. Exit
Enter choice: 8

=== MAIN MENU ===
1. Customer
2. Admin
3. Exit
Enter choice: 3
 Exiting ZENCART App. Thank you!
```

CONCLUSION:

The ZenCart project successfully demonstrates the development of a fully functional, menu-driven e-commerce backend system using Python and MySQL. By following a layered architecture and modular design, the application ensures clarity, maintainability, and scalability. Each component—from entity classes to DAO implementations, exception handling, and utility functions—was developed with adherence to best software development practices.

Key operations such as customer registration, product management, cart handling, and order placement were implemented and tested against a live database. Role-based access control through the command-line interface makes the system intuitive for both administrators and customers. Moreover, the integration of custom exceptions, unit testing using Pytest, and parameterized SQL queries ensures that the application is both robust and secure.

Overall, this project showcases a solid understanding of object-oriented programming, database integration, and backend application development, making it a strong foundation.