Case Study On Ecommerce Application

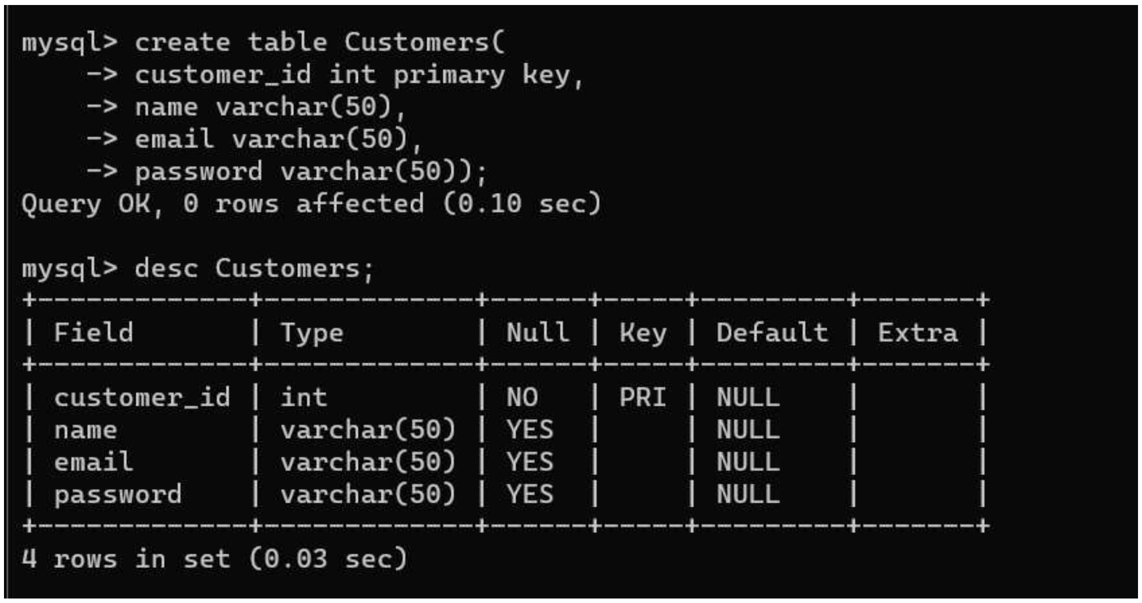
Create following tables in SQL Schema with appropriate class and write the unit test case for the Ecommerce application.

Schema Design:

1. customers table:

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

mysql> create table Customers(
  -> customer_id int primary key,
  -> name varchar(50),
  -> email varchar(50),
  -> password varchar(50));

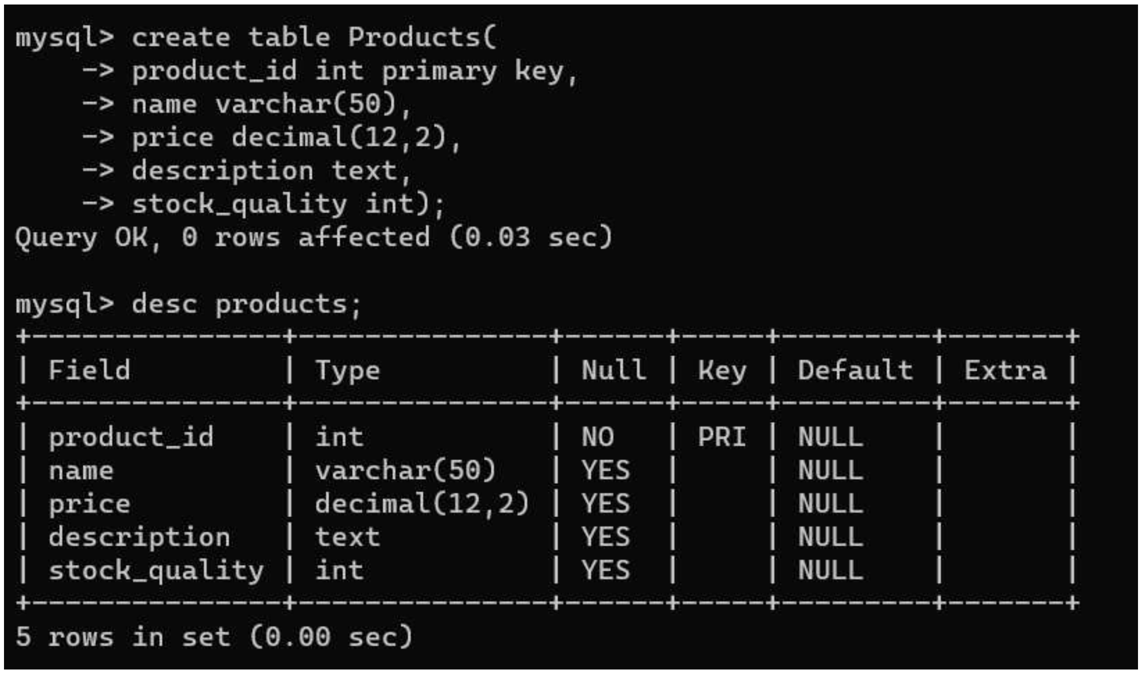


```
mysql> create table Customers(
    -> customer_id int primary key,
    -> name varchar(50),
    -> email varchar(50),
    -> password varchar(50));
Query OK, 0 rows affected (0.10 sec)

mysql> desc Customers;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| customer_id | int         | NO   | PRI | NULL    |       |
| name        | varchar(50) | YES  |     | NULL    |       |
| email       | varchar(50) | YES  |     | NULL    |       |
| password    | varchar(50) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.03 sec)
```

2. products table:

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

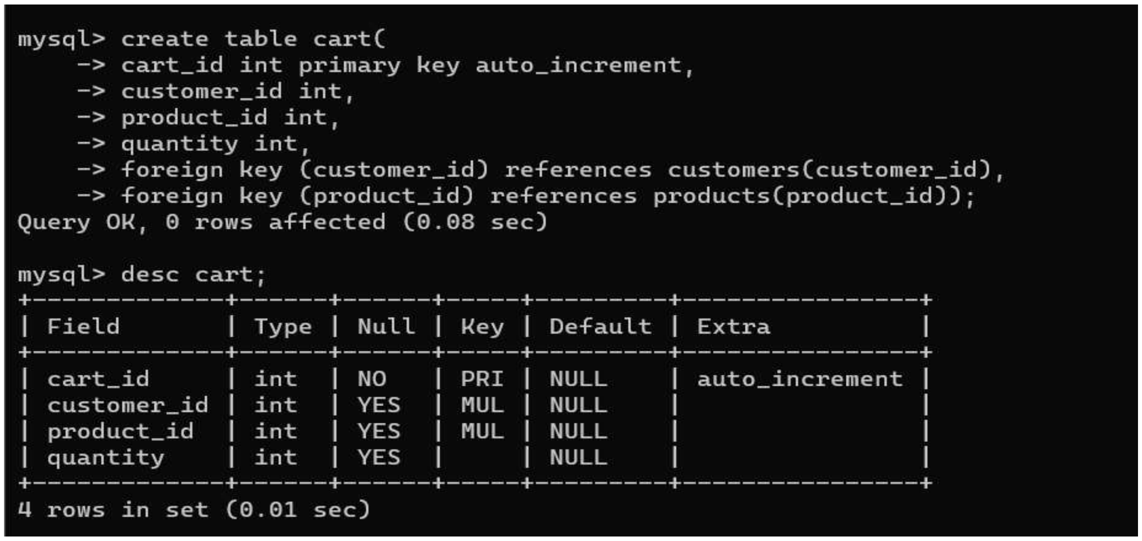
mysql> create table Products(
-> product_id int primary key,
-> name varchar(50),
-> price decimal(12,2),
-> description text,
-> stock_quality int);



```
mysql> create table Products(
    -> product_id int primary key,
    -> name varchar(50),
    -> price decimal(12,2),
    -> description text,
    -> stock_quality int);
Query OK, 0 rows affected (0.03 sec)

mysql> desc products;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| product_id    | int           | NO   | PRI | NULL    |       |
| name          | varchar(50)   | YES  |     | NULL    |       |
| price         | decimal(12,2) | YES  |     | NULL    |       |
| description   | text          | YES  |     | NULL    |       |
| stock_quality | int           | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

3. cart table:
- *cart_id (Primary Key)*
- *customer_id (Foreign Key)*
- *product_id (Foreign Key)*
- *quantity*

mysql> create table cart(
-> cart_id int primary key auto_increment,
-> customer_id int,
-> product_id int,
-> quantity int,
-> foreign key (customer_id) references customers(customer_id),
-> foreign key (product_id) references products(product_id));

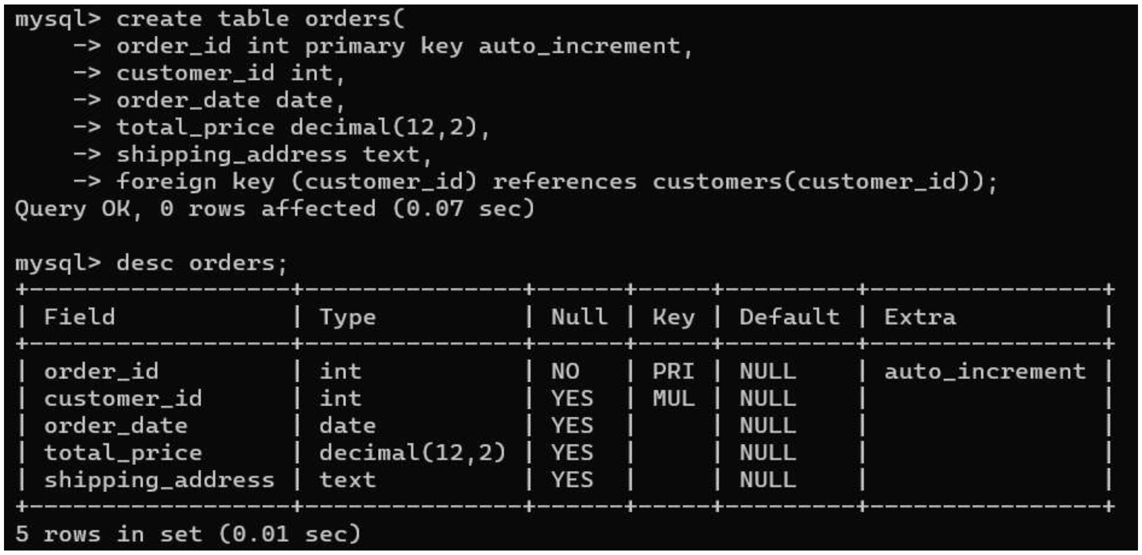


```
mysql> create table cart(
    -> cart_id int primary key auto_increment,
    -> customer_id int,
    -> product_id int,
    -> quantity int,
    -> foreign key (customer_id) references customers(customer_id),
    -> foreign key (product_id) references products(product_id));
Query OK, 0 rows affected (0.08 sec)

mysql> desc cart;
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| cart_id     | int  | NO   | PRI | NULL    | auto_increment |
| customer_id | int  | YES  | MUL | NULL    |                |
| product_id  | int  | YES  | MUL | NULL    |                |
| quantity    | int  | YES  |     | NULL    |                |
+-------------+------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

4. orders table:

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

```
mysql> create table orders(
  -> order_id int primary key auto_increment,
  -> customer_id int,
  -> order_date date,
  -> total_price decimal(12,2),
  -> shipping_address text,
  -> foreign key (customer_id) references customers(customer_id));
```

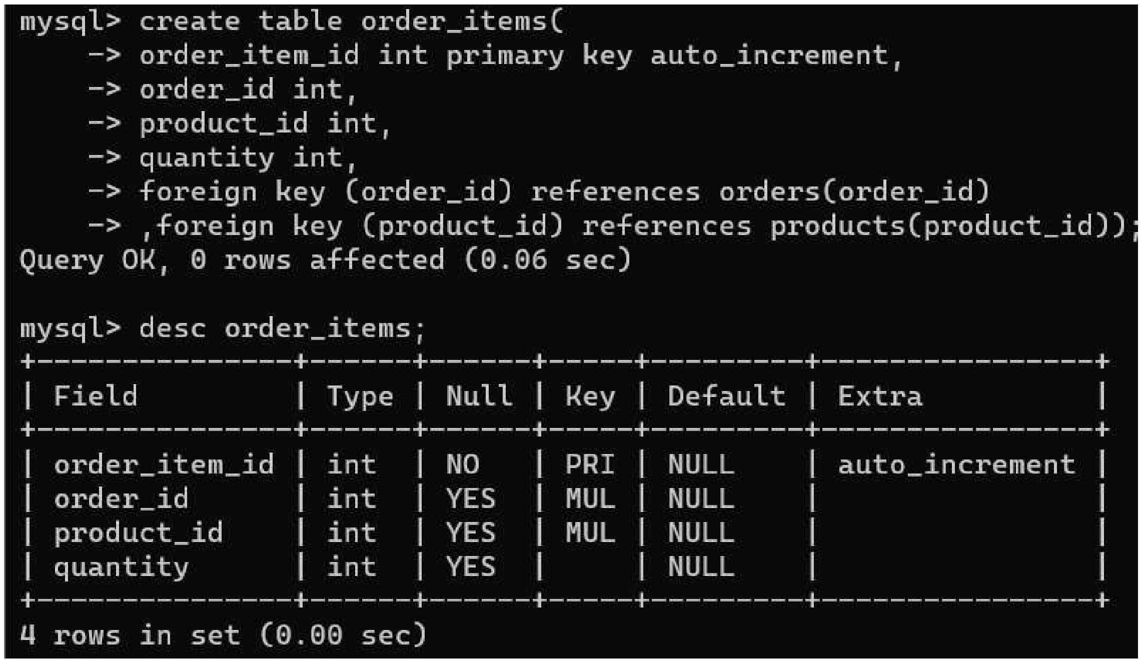


```
mysql> create table orders(
    -> order_id int primary key auto_increment,
    -> customer_id int,
    -> order_date date,
    -> total_price decimal(12,2),
    -> shipping_address text,
    -> foreign key (customer_id) references customers(customer_id));
Query OK, 0 rows affected (0.07 sec)

mysql> desc orders;
+------------------+---------------+------+-----+---------+----------------+
| Field            | Type          | Null | Key | Default | Extra          |
+------------------+---------------+------+-----+---------+----------------+
| order_id         | int           | NO   | PRI | NULL    | auto_increment |
| customer_id      | int           | YES  | MUL | NULL    |                |
| order_date       | date          | YES  |     | NULL    |                |
| total_price      | decimal(12,2) | YES  |     | NULL    |                |
| shipping_address | text          | YES  |     | NULL    |                |
+------------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

5. order_items table (to store order details):

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

```
mysql> create table order_items(
   -> order_item_id int primary key auto_increment,
   -> order_id int,
   -> product_id int,
   -> quantity int,
   -> foreign key (order_id) references orders(order_id)
   -> ,foreign key (product_id) references products(product_id));
```



```
mysql> create table order_items(
    -> order_item_id int primary key auto_increment,
    -> order_id int,
    -> product_id int,
    -> quantity int,
    -> foreign key (order_id) references orders(order_id)
    -> ,foreign key (product_id) references products(product_id))
Query OK, 0 rows affected (0.06 sec)

mysql> desc order_items;
+---------------+------+------+-----+---------+----------------+
| Field         | Type | Null | Key | Default | Extra          |
+---------------+------+------+-----+---------+----------------+
| order_item_id | int  | NO   | PRI | NULL    | auto_increment |
| order_id      | int  | YES  | MUL | NULL    |                |
| product_id    | int  | YES  | MUL | NULL    |                |
| quantity      | int  | YES  |     | NULL    |                |
+---------------+------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

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 the database.

1. createProduct()

parameter: Product product

return type: boolean

```
@abstractmethod
def create_product(self, product):
      query = "insert into products(product_id, name, price, description, 
stock_quantity) values(%s, %s, %s, %s, %s)"
      try:
      self.cursor.execute(query, (product.product_id, product.name, 
product.price, product.description, product.stock_quantity))
      self.connection.commit()
      return True
      except Exception as e:
      print("Error while creating product: ", e)
      return False
```

2. createCustomer()

parameter: Customer

customer return type: boolean

```
@abstractmethod
def create_customer(self, customer):
        query = "insert into customers(customer_id, name, email, password) values(%s, %s, 
%s, %s)"
        try:
        self.cursor.execute(query, customer.customer_id, customer.name, customer.email, 
customer.password)
        self.connection.commit()
        return True
        except Exception as e:
        print("Error while creating customer: ", e)
```

3. deleteProduct()

parameter: productId

return type: boolean

```
@abstractmethod
def delete_product(self, product_id):
   if not self.product_exists(product_id):
       raise ProductNotFound()

   try:
       query = "delete from products where product_id = %s"
       self.cursor.execute(query, (product_id,))
       self.connection.commit()
```

```
        return True

    except Exception as e:
        print("Error while deleting product: ", e)
```

4. deleteCustomer(customerId)
parameter: customerId
return type: boolean

```
@abstractmethod
def delete_customer(self, customer_id):
    if not self.customer_exists(customer_id):
        raise CustomerNotFound()

    try:
        query = "delete from customers where customer_id = %s"
        self.cursor.execute(query, (customer_id,))
        self.connection.commit()
        return True
    except Exception as e:
        print("Exception while deleting customer: ", e)
```

5. addToCart(): *insert the product in the cart.*
parameter: Customer customer, Product product, int quantity
return type: boolean

```
@abstractmethod
def delete_customer(self, customer_id):
    if not self.customer_exists(customer_id):
        raise CustomerNotFound()

    try:
        query = "delete from customers where customer_id = %s"
        self.cursor.execute(query, (customer_id,))
        self.connection.commit()
        return True
    except Exception as e:
        print("Exception while deleting customer: ", e)
```

6. removeFromCart(): *delete the product in cart.*
parameter: Customer customer, Product product
return type: boolean

```
@abstractmethod
def remove_from_cart(self, customer, product):
    if not self.order_in_cart(customer, product):
        raise OrderNotFound()
    try:
        query = "delete from cart where customer_id = %s and product_id =
%s"
        self.cursor.execute(query, (customer.customer_id,
product.product_id))
        self.connection.commit()
        return True
    except Exception as e:
        print("Error while removing from cart: ", e)
```

7. getAllFromCart(Customer customer): *list the product in the cart for a customer.*
parameter: Customer customer
return type: list of product

```
@abstractmethod
def get_all_from_cart(self, customer):
    if not self.customer_exists(customer.customer_id):
        raise CustomerNotFound()

    try:
        query = "select * from cart where customer_id = %s"
        self.cursor.execute(query, (customer.customer_id,))
        return self.cursor

    except Exception as e:
        print("Error while getting orders from cart:", e)
```

8. placeOrder(Customer customer, List<Map<Product,quantity>>, string shippingAddress): *should update order table and orderItems table.*
parameter: Customer customer, list of product and quantity
return type: boolean

```

def place_order(self, customer, products, shipping_address):
    try:
        query = "INSERT INTO orders (customer_id, shipping_address) VALUES
(%s, %s)"
        self.cursor.execute(query, (customer.id, shipping_address))
        order_id = self.cursor.lastrowid

        for product, quantity in products.items():
            product_id = product.id
            query = "INSERT INTO order_items (order_id, product_id,
quantity) VALUES (%s, %s, %s)"
            self.cursor.execute(query, (order_id, product_id, quantity))

        self.connection.commit()
        return True
```

```
   except Exception as e:
       print("Error while placing order:", e)
```

9. getOrdersByCustomer()
1. parameter: customerid
2. return type: list of product and quantity

```
@abstractmethod
def get_orders_by_customer(self, customer_id):
   if not self.customer_exists(customer_id):
       raise CustomerNotFound()

   try:
       query = "select P.name, OI.quantity from Orders O " \
               "join order_items OI on OI.order_id = O.order_id " \
               "join products P on P.product_id = OI.product_id " \
               "where O.customer_id = %s;"
       self.cursor.execute(query, (customer_id,))
       return self.cursor

   except Exception as e:
       print("Error while getting orders: ", e)
```

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

```
from orderProcessorRepository import OrderProcessorRepository


class OrderProcessorRepositoryImpl(OrderProcessorRepository):

   def create_product(self, product):
       if super().create_product(product):
           print("Product Created")

   def create_customer(self, customer):
       if super().create_product(customer):
           print("Customer Created")

   def delete_product(self, product_id):
       if super().delete_product(product_id):
           print(f"Product: {product_id} got deleted")

   def delete_customer(self, customer_id):
       if super().delete_customer(customer_id):
           print(f"Customer: {customer_id} got deleted")

   def add_to_cart(self, customer, product, quantity):
```

```python
        if super().add_to_cart(customer, product, quantity):
            print("Product added to cart")

    def remove_from_cart(self, customer, product):
        if super().remove_from_cart(customer, product):
            print(f"Product: {product.product_id} removed from cart")

    def get_all_from_cart(self, customer):
        for i in super().get_all_from_cart(customer):
            print(i)

    def place_orders(self):
        if super().place_orders():
            print("Order Placed Successfully")

    def get_orders_by_customer(self, customer_id):
        for i in super().get_orders_by_customer(customer_id):
            print(i)
```

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 propertyUtil import PropertyUtil


class DBConnection:
    def __init__(self):
        pass

    def get_connection(self):
        try:
            conn_str = PropertyUtil.get_property()
            conn = mysql.connector.connect(conn_str)
            print(conn_str)
            if not conn.is_connected():
                print("Connected to MySQL database")
                cur = conn.cursor()
                return cur

        except Exception as e:
            print("Error while connecting", e)

db_connect = DBConnection()
db_connect.get_connection()
```

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

```
[data]

host='localhost'
user='root'
password='root'
database='Ecomm'
```

• 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.

```
import configparser

class PropertyUtil:
   @staticmethod
   def get_property_string():
       config = configparser.ConfigParser()
       config.read('data.properties')

       host = config.get('data', 'host')
       user = config.get('data', 'user')
       password = config.get('data', 'password')
       database = config.get('data', 'database')

       return f"host={host}, user={user}, password={password},
database={database}"
```

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

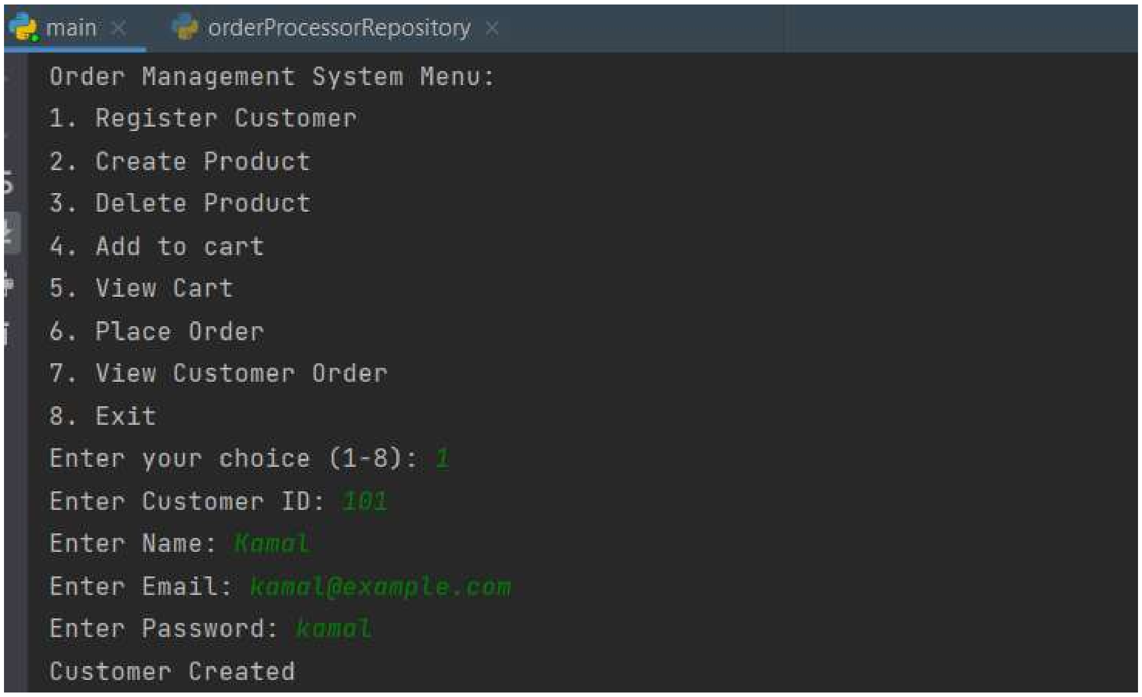
```

class ProductNotFound(Exception):
   def __init__(self, message="Product Not Found"):
       self.message = message
       super().__init__(self.message)


class CustomerNotFound(Exception):
   def __init__(self, message="Customer Not Found"):
        self.message = message
        super().__init__(self.message)


class OrderNotFound(Exception):
    def __init__(self, message="Order Not Found"):
        self.message = message
        super().__init__(self.message)
```

***10.** Create class named **EcomApp** with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.*
1. Register Customer.
2. Create Product.
3. Delete Product.
4. Add to cart.
5. View cart.
6. Place order.
7. View Customer Order

```
from dao import OrderProcessorRepositoryImpl
from customers import Customers
from products import Products


class EcomApp:
    def __init__(self):
        self.order_processor = OrderProcessorRepositoryImpl()

    def display_menu(self):
        print("Order Management System Menu:")
        print("1. Register Customer")
        print("2. Create Product")
        print("3. Delete Product")
        print("4. Add to cart")
        print("5. View Cart")
        print("6. Place Order")
        print("7. View Customer Order")
        print("8. Exit")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice (1-7): ")
            if choice == '1':
                self.create_customer()
            elif choice == '2':
                self.create_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
```

```python
                self.add_to_cart()
            elif choice == '5':
                self.get_all_from_cart()
            elif choice == '6':
                self.place_orders()
            elif choice == '7':
                self.get_order_by_customer()
            elif choice == '8':
                print("Exiting...")
                break
            else:
                print("Invalid choice. Please enter a number from 1 to 6.")

    def create_customer(self):

        customer_id = input("Enter Customer ID: ")
        name = input("Enter Name: ")
        email = input("Enter Email: ")
        password = input("Enter Password: ")

        customer = Customers(customer_id, name, email, password)
        self.order_processor.create_customer(customer)

    def create_product(self):

        product_id = input("Enter Product ID: ")
        name = input("Enter Product Name: ")
        price = input("Enter Price: ")
        description = input("About Product: ")
        stock_quantity = input("Enter Product Quantity: ")

        product = Products(product_id, name, price, description,
stock_quantity)
        self.order_processor.create_product(product)

    def delete_product(self):

        product_id = input("Enter ProductID that should be deleted: ")

        self.order_processor.delete_product(product_id)

    def add_to_cart(self):
        customer_id = input("Enter Customer ID: ")
        name = input("Enter Name: ")
        email = input("Enter Email: ")
        password = input("Enter Password: ")

        customer = Customers(customer_id, name, email, password)

        product_id = input("Enter Product ID: ")
        name = input("Enter Product Name: ")
        price = input("Enter Price: ")
        description = input("About Product: ")
        stock_quantity = input("Enter Product Quantity: ")

        product = Products(product_id, name, price, description,
stock_quantity)
        quantity = input("Enter no.of Products need to be added: ")

        self.add_to_cart(customer, product, quantity)

    def get_all_from_cart(self):
        customer_id = input("Enter Customer ID: ")
        name = input("Enter Name: ")
        email = input("Enter Email: ")
        password = input("Enter Password: ")

        customer = Customers(customer_id, name, email, password)
        self.get_all_from_cart(customer)

    def place_orders(self):
        self.place_orders()

    def get_orders_by_customers(self):
        customer_id = input("Enter Customer ID: ")
        self.get_orders_by_customers(customer_id)


order = EcomApp()
order.run()
```

Output:



```
Order Management System Menu:
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View Cart
6. Place Order
7. View Customer Order
8. Exit
Enter your choice (1-8): 1
Enter Customer ID: 101
Enter Name: Kamal
Enter Email: kamal@example.com
Enter Password: kamal
Customer Created
```

Unit Testing

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

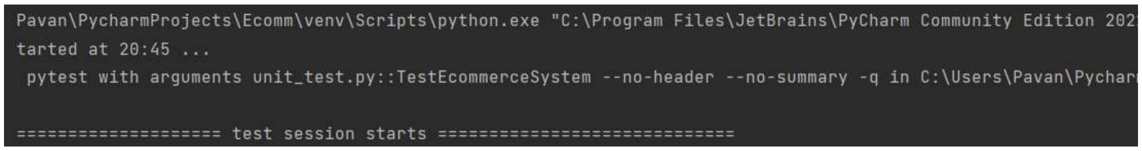
Following questions to guide the creation of Unit test cases:

- *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.*

```
import pytest
from main import EcomApp
from customers import Customers
from products import Products
from cart import Cart
from myExceptions import CustomerNotFound, ProductNotFound


class TestEcommerceSystem(pytest.TestCase):

   def setUp(self):

       self.ecommerce_system = EcomApp()
       self.customer = Customers(id=101, name="Kamal",
email="kamal@example.com", password="kamal")
       self.product = Products(id=201, name="Laptop", price=42000,
description="Brand Lenovo", stock_quantity=25)
       self.cart = Cart()

   def test_product_creation(self):
       self.ecommerce_system.create_product(self.product)
       self.assertIn(self.product, self.ecommerce_system.products)

   def test_add_to_cart(self):
       self.ecommerce_system.add_to_cart(self.customer, self.product,
quantity=2)
       self.assertIn((self.customer, self.product, 2),
self.ecommerce_system.carts)

   def test_place_order(self):
       order_placed = self.ecommerce_system.place_order(self.customer,
[(self.product, 2)], "123 Main St")
       self.assertTrue(order_placed)

   def test_customer_not_found_exception(self):
       with self.assertRaises(CustomerNotFound):
           self.ecommerce_system.get_order_by_customer(500)

   def test_product_not_found_exception(self):
       with self.assertRaises(ProductNotFound):
           self.ecommerce_system.get_product_by_id(999)
```

```
if __name__ == '__main__':
   pytest.main()
```

Output:



```
Pavan\PycharmProjects\Ecomm\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 202
tarted at 20:45 ...
 pytest with arguments unit_test.py::TestEcommerceSystem --no-header --no-summary -q in C:\Users\Pavan\Pychar

==================== test session starts ============================
```

*****