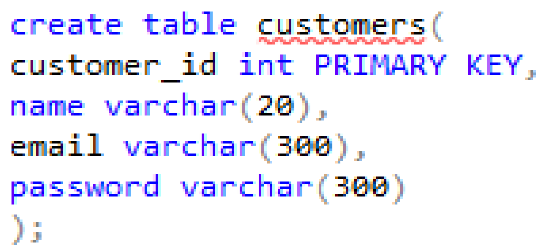
Case Study 6: Ecommerce

-Seemalamudi Nikhil Sai (Maverick)

- nikhilsai16802@gmail.com - (+91) 9441148735

1. **Customers table:**
   - **customer_id (Primary Key)**
   - **name**
   - **email**
   - **password**

→

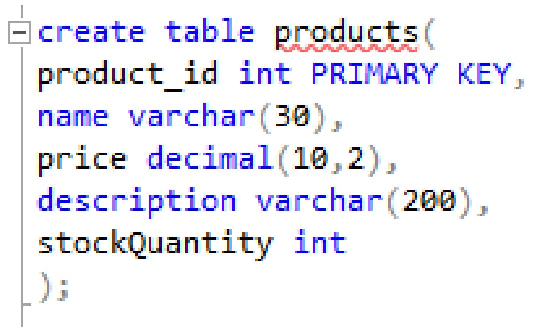


```
create table customers(
customer_id int PRIMARY KEY,
name varchar(20),
email varchar(300),
password varchar(300)
);
```

---

2. **Products table:**
   - **product_id (Primary Key)**
   - **name**
   - **price**
   - **description**
   - **stockQuantity.**

→

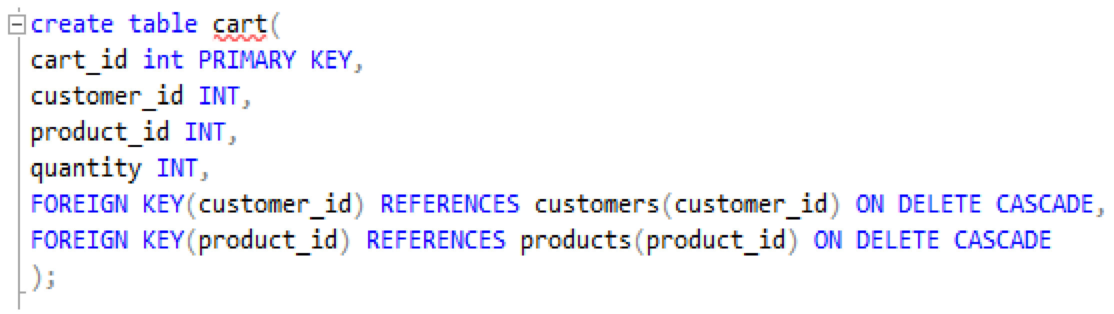


```
create table products(
product_id int PRIMARY KEY,
name varchar(30),
price decimal(10,2),
description varchar(200),
stockQuantity int
);
```

---

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

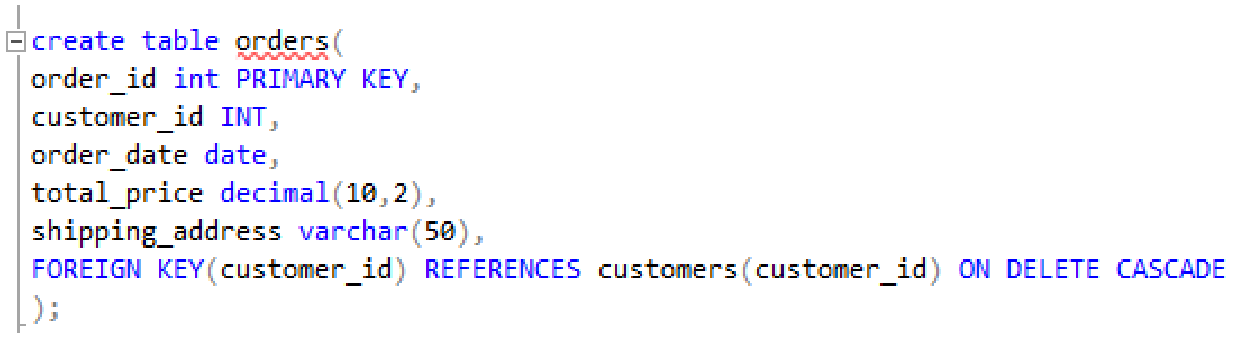
→



```sql
create table cart(
cart_id int PRIMARY KEY,
customer_id INT,
product_id INT,
quantity INT,
FOREIGN KEY(customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE,
FOREIGN KEY(product_id) REFERENCES products(product_id) ON DELETE CASCADE
);
```

4. **Orders table:**
   - **order_id (Primary Key)**
   - **customer_id (Foreign Key)**
   - **order_date**
   - **total_price**
   - **shipping_address**

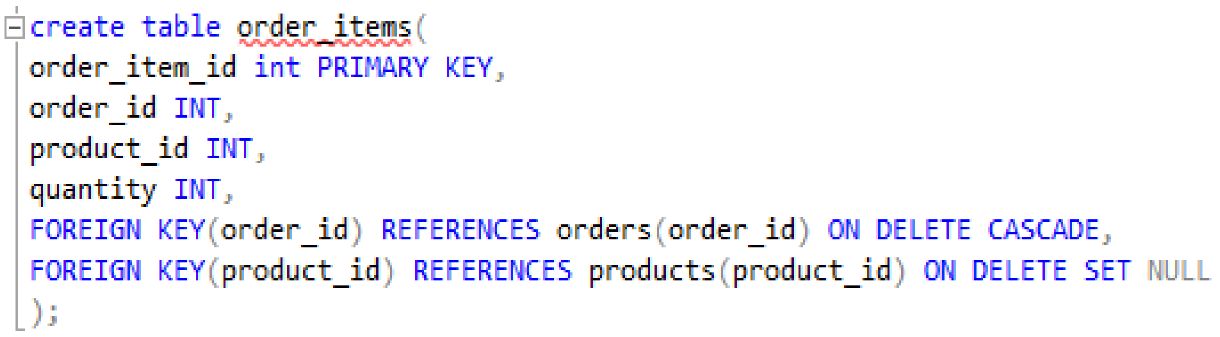
→



```sql
create table orders(
order_id int PRIMARY KEY,
customer_id INT,
order_date date,
total_price decimal(10,2),
shipping_address varchar(50),
FOREIGN KEY(customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE
);
```

5. **Order_items table (to store order details):**
   - **order_item_id (Primary Key)**
   - **order_id (Foreign Key)**
   - **product_id (Foreign Key)**
   - **quantity**

→



```sql
create table order_items(
order_item_id int PRIMARY KEY,
order_id INT,
product_id INT,
quantity INT,
FOREIGN KEY(order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
FOREIGN KEY(product_id) REFERENCES products(product_id) ON DELETE SET NULL
);
```

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**
2. **createCustomer() parameter: Customer customer return type: Boolean**
3. **deleteProduct() parameter: productId return type: boolean**
4. **deleteCustomer(customerId) parameter: customerId return type: Boolean**
5. **addToCart(): insert the product in cart. parameter: Customer customer, Product product, int quantity return type: Boolean**
6. **removeFromCart(): delete the product in cart. parameter: Customer customer, Product product return type: Boolean**
7. **getAllFromCart(Customer customer): list the product in cart for a customer. parameter: Customer customer return type: list of product**
8. **placeOrder(Customer customer, List<Map>, string shippingAddress): should update order table and orderItems table. 1. parameter: Customer customer, list of product and quantity 2. return type: Boolean**
9. **getOrdersByCustomer() 1. parameter: customerid 2. return type: list of product and quantity**

→ #dao/OrderProcessor.py

```
import sys

import os

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from abc import ABC, abstractmethod

from typing import List, Dict

from entity.customer import Customer

from entity.product import Product


class OrderProcessorRepository(ABC):


    @abstractmethod

    def create_product(self, product: Product) -> bool:

        pass


    @abstractmethod
    def create_customer(self, customer: Customer) -> bool:
        pass

    @abstractmethod
    def delete_product(self, product_id: int) -> bool:
        pass

    @abstractmethod
    def delete_customer(self, customer_id: int) -> bool:
        pass

    @abstractmethod
    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
        pass

    @abstractmethod
    def remove_from_cart(self, customer: Customer, product: Product) -> bool:
        pass

    @abstractmethod
    def get_all_from_cart(self, customer: Customer) -> List[Product]:
        pass

    @abstractmethod
    def place_order(self, customer: Customer, product_quantity_map: List[Dict[Product, int]],
shipping_address: str) -> bool:
        pass

    @abstractmethod
    def get_orders_by_customer(self, customer_id: int) -> List[Dict[Product, int]]:
        pass
```

---

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

→

```
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from dao.OrderProcessorRepository import OrderProcessorRepository
from typing import List, Dict, Optional, Tuple
from entity.customer import Customer
from entity.product import Product
from exception.customernotfound import CustomerNotFound
from exception.productnotfound import ProductNotFound
from util.DBConnection import DBConnection

class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    def __init__(self):
        self.connection = DBConnection.get_connection()

    def create_product(self, product: Product) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute(
                "INSERT INTO products (product_id, name, price, description, stockQuantity)
VALUES (?, ?, ?, ?, ?)",
                (product.get_product_id(), product.get_name(), product.get_price(),
product.get_description(), product.get_stockQuantity())
```

```
            )
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error creating product: {e}")
            return False

    def create_customer(self, customer: Customer) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute(
                "INSERT INTO customers (customer_id, name, email, password) VALUES (?, ?, ?,
?)",
                (customer.get_customer_id(), customer.get_name(), customer.get_email(),
customer.get_password())
            )
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error creating customer: {e}")
            return False

    def delete_product(self, product_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM products WHERE product_id = ?", (product_id,))
            if cursor.fetchone() is None:
                raise ProductNotFound(f"Product ID {product_id} does not exist.")
            cursor.execute("DELETE FROM products WHERE product_id = ?", (product_id,))
            self.connection.commit()
            return True
        except ProductNotFound as e:
            print(e)
            return False
        except Exception as e:
            print(f"Error deleting product: {e}")
            return False

    def delete_customer(self, customer_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM customers WHERE customer_id = ?", (customer_id,))
            if cursor.fetchone() is None:
                raise CustomerNotFound(f"Customer ID {customer_id} does not exist.")

            cursor.execute("DELETE FROM customers WHERE customer_id = ?", (customer_id,))
            self.connection.commit()
            return True
        except CustomerNotFound as e:
            print(e)
            return False
        except Exception as e:
            print(f"Error deleting customer: {e}")
            return False

    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT MAX(cart_id) FROM cart")
            max_cart_id = cursor.fetchone()[0]
            if max_cart_id is None:
                new_cart_id = 1  # Start from 1 if no entries exist
            else:
                new_cart_id = max_cart_id + 1  # Increment the max_id for new entry


            cursor.execute(
                "INSERT INTO cart (cart_id, customer_id, product_id, quantity) VALUES (?, ?, ?, ?)",
                (new_cart_id, customer.get_customer_id(), product.get_product_id(), quantity)
            )
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error adding to cart: {e}")
            return False


    def remove_from_cart(self, customer: Customer, product: Product) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute(
                "DELETE FROM cart WHERE customer_id = ? AND product_id = ?",
                (customer.customer_id, product.product_id)
            )
            self.connection.commit()
            return True
    except Exception as e:
        print(f"Error removing from cart: {e}")
        return False

def get_all_from_cart(self, customer):
    cart_items = []
    cursor = self.connection.cursor()

    try:
        cursor.execute("SELECT p.product_id, p.name, p.price, c.quantity "
                       "FROM cart c "
                       "JOIN products p ON c.product_id = p.product_id "
                       "WHERE c.customer_id = ?", (customer.get_customer_id(),))

        rows = cursor.fetchall()

        for row in rows:
            product = Product(
                product_id=row[0],
                name=row[1],
                price=row[2],
                description='',  # Assuming description is not needed in the cart view
                stockQuantity=0  # Not needed here, but you can set it to 0 or leave it
        )
        cart_items.append({'product': product, 'quantity': row[3]})  # Use row[3] for quantity

    except Exception as e:
        print("Error retrieving cart items:", e)
        finally:
            cursor.close()

        return cart_items

    def place_order(self, customer: Customer, product_quantity_map: List[Tuple[Product, int]],
shipping_address: str) -> bool:
        try:
            cursor = self.connection.cursor()
            #Fetch the max order_id from the orders table
            cursor.execute("SELECT MAX(order_id) FROM orders")
            max_order_id=cursor.fetchone()[0]
            if max_order_id is None:
                new_order_id = 1
            else:
                new_order_id = max_order_id + 1
            total_price = self.calculate_total_price(product_quantity_map)
            cursor.execute(
                "INSERT INTO orders (customer_id, order_id, order_date, total_price,
shipping_address) VALUES (?, ?, GETDATE(), ?, ?)",
                (customer.get_customer_id(), new_order_id, total_price, shipping_address)
            )
            # Step 3: Fetch the max order_item_id from the order_items table
            cursor.execute("SELECT MAX(order_item_id) FROM order_items")
            max_order_item_id=cursor.fetchone()[0]
            if max_order_item_id is None:
                new_order_item_id = 1
            else:
                new_order_item_id = max_order_item_id + 1
            for product, quantity in product_quantity_map:
                cursor.execute(
                    "INSERT INTO order_items (order_item_id, order_id, product_id, quantity)
VALUES (?, ?, ?, ?)",
                    (new_order_item_id, new_order_id, product.get_product_id(), quantity)
                )
                new_order_item_id += 1      # Increment for each order item


                cursor.execute("""
                UPDATE products
                SET stockQuantity = stockQuantity - ?
                WHERE product_id = ?
                """, (quantity, product.get_product_id()))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error placing order: {e}")
            self.connection.rollback()  # Rollback in case of any errors
            return False
        finally:
            cursor.close()


    def get_orders_by_customer(self, customer_id: int) -> Dict[Product, int]:
        try:
            cursor = self.connection.cursor()
            cursor.execute(
                "SELECT oi.product_id, oi.quantity, p.price, p.description, p.stockQuantity "
                "FROM orders o "
                "JOIN order_items oi ON o.order_id = oi.order_id "
                "JOIN products p ON oi.product_id = p.product_id "
                "WHERE o.customer_id = ?",
                (customer_id,)
                )
            rows = cursor.fetchall()
            # print("Fetched rows:", rows)
            orders={}
            for row in rows:
                product_id = row[0]          # product_id from order_items
                quantity = row[1]            # quantity from order_items
                price = row[2]               # price from products
                description = row[3]         # description from products
                stock_quantity = row[4]      # stockQuantity from products

                product = Product(
                    product_id=product_id,
                    name=None,
                    price=price,
                    description=description,
                    stockQuantity=stock_quantity
                )
                orders[product] = quantity  # quantity
            return orders
        except Exception as e:
            print(f"Error retrieving orders: {e}")
            return {}

    def calculate_total_price(self, product_quantity_map: List[Tuple[Product, int]]) -> float:
        total_price = 0.0
        for product, quantity in product_quantity_map:
            total_price += float(product.get_price() * quantity)
        return total_price

    def get_customer_by_id(self, customer_id: int) -> Customer:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM customers WHERE customer_id = ?", (customer_id,))
            row = cursor.fetchone()
            if row:
                return Customer(row[0], row[1], row[2], row[3])  # Adjust index based on your
schema
            else:
                return None  # No customer found
        except Exception as e:
            print(f"Error retrieving customer: {e}")
            return None

    def get_product_by_id(self, product_id: int) -> Optional[Tuple[int, str, float, str, int]]:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM products WHERE product_id = ?", (product_id,))
            return cursor.fetchone()  # Return the entire row as a tuple
        except Exception as e:
            print(f"Error retrieving product: {e}")
            return None
```

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.

• 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/DBConnection.py

```
import sys

import os

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

import pyodbc

from util.PropertyUtil import PropertyUtil  # Adjust import based on your package structure


class DBConnection:

    connection = None


    @staticmethod

    def get_connection():

        if DBConnection.connection is None:

            try:

                conn_str = PropertyUtil.get_property_string()

                DBConnection.connection = pyodbc.connect(conn_str)

                print("Connected Successfully")

            except Exception as e:

                print(f"Connection failed: {e}")

        return DBConnection.connection

    @staticmethod

    def test_connection():
        # This method will only test the connection without executing any queries
        connection = DBConnection.get_connection()
        if connection:
            print("Database connection is successful.")
        else:
            print("Failed to connect to the database.")

if __name__ == "__main__":
    DBConnection.test_connection()
```

#util/PropertyUtil.py

```
class PropertyUtil:
    @staticmethod
    def get_property_string():
        hostname = "SARTHAKKULKARNI"  # Your SQL Server instance name
        dbname = "Ecommerce_Application"          # Your database name

        connection_string = (
            f"Driver={{SQL Server}};"
            f"Server={hostname};"
            f"Database={dbname};"
            "Trusted_Connection=yes;"
        )
        return connection_string
```

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

→

#exception/customernotfound.py

```
class CustomerNotFound(Exception):
    def __init__(self, message="Customer not found."):
        self.message = message
        super().__init__(self.message)
```

#exception/productnotfound.py

```
class ProductNotFound(Exception):
    def __init__(self, message="Product not found."):
        self.message = message
        super().__init__(self.message)
```

#exception/ordernotfound.py

```
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

→

```
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from entity.customer import Customer
from entity.product import Product
from dao.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl
from exception.customernotfound import CustomerNotFound
from exception.productnotfound import ProductNotFound

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

    def display_menu(self):
        print("\nE-commerce Application\n")
        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 register_customer(self):
        customerid=int(input("Enter customer Id:"))
        name = input("Enter customer name: ")
        email = input("Enter customer email: ")
        password = input("Enter customer password: ")
        customer = Customer(customer_id=customerid, name=name, email=email,
password=password)
        if self.repository.create_customer(customer):
            print("Customer registered successfully.")
        else:
            print("Failed to register customer.")

    def create_product(self):
        productid = int(input("Enter product Id:"))
        name = input("Enter product name: ")
        price = float(input("Enter product price: "))
        description = input("Enter product description: ")
        stock_quantity = int(input("Enter stock quantity: "))
        product = Product(product_id=productid, name=name, price=price, description=description,
stockQuantity=stock_quantity)
        if self.repository.create_product(product):
            print("Product created successfully.")
        else:
            print("Failed to create product.")

    def delete_product(self):
        product_id = int(input("Enter product ID to delete: "))
        if self.repository.delete_product(product_id):
```

```
        print("Product deleted successfully.")
    else:
        print("Failed to delete product.")

def add_to_cart(self):
    customer_id = int(input("Enter customer ID: "))
    product_id = int(input("Enter product ID: "))
    quantity = int(input("Enter quantity: "))
    customer = self.repository.get_customer_by_id(customer_id)
    if not customer:
        print("Customer not found. Please register first.")
        return
    product_row = self.repository.get_product_by_id(product_id)
    if not product_row:
        print("Product not found.")
        return

    # Create product object with all required attributes
    product = Product(
        product_id=product_row[0],
        name=product_row[1],
        price=product_row[2],
        description=product_row[3],
        stockQuantity=product_row[4]
    )

    if self.repository.add_to_cart(customer, product, quantity):
        print("Product added to cart successfully.")
        else:
            print("Failed to add product to cart.")

    def view_cart(self):
        customer_id = int(input("Enter customer ID: "))
        # Fetch the customer from the repository to get all details
        customer = self.repository.get_customer_by_id(customer_id)
        if customer:
            cart_items = self.repository.get_all_from_cart(customer)
            if cart_items:
                print("Cart items:")
                for item in cart_items:
                    product = item['product']
                    quantity = item['quantity']
                    print(f"- {product.get_name()}, Price: {product.get_price()}, Quantity: {quantity}")
            else:
                print("Cart is empty.")
        else:
            print("Cart is empty or Customer not found.")

    def place_order(self):
        customer_id = int(input("Enter customer ID: "))
        shipping_address = input("Enter shipping address: ")
        product_quantity_map = []  # List of tuples (product, quantity)
        # Fetch the customer from the repository
        customer = self.repository.get_customer_by_id(customer_id)
        if not customer:
            print("Customer not found. Please register first.")
            return
        while True:
            product_id = int(input("Enter product ID to order (0 to finish): "))
            if product_id == 0:
                break
            quantity = int(input("Enter quantity: "))
            product_row = self.repository.get_product_by_id(product_id)
            if not product_row:
                print(f"Product with ID {product_id} not found.")
                continue
            # Create product object with all required attributes
            product = Product(
                product_id=product_row[0],
                name=product_row[1],
                price=product_row[2],
                description=product_row[3],
                stockQuantity=product_row[4]
            )
            if product.get_stockQuantity()< quantity:
                print(f"Not enough stock for {product.get_name()}. Available:
{product.get_stockQuantity()}")
                continue
            product_quantity_map.append((product, quantity))
        if not product_quantity_map:
            print("No products were selected for the order.")
            return

        if self.repository.place_order(customer, product_quantity_map, shipping_address):
            print("Order placed successfully.")
```

```
    else:
        print("Failed to place order.")

def view_customer_order(self):
    customer_id = int(input("Enter customer ID to view orders: "))
    orders = self.repository.get_orders_by_customer(customer_id)
    if orders:
        print("Orders for Customer ID:", customer_id)
        for product, quantity in orders.items():
            print(f"- Product ID: {product.get_product_id()}, Quantity: {quantity}")
    else:
        print("No orders found for this customer.")

def run(self):
    while True:
        self.display_menu()
        choice = input("Choose an operation (1-8): ")
        if choice == '1':
            self.register_customer()
        elif choice == '2':
            self.create_product()
        elif choice == '3':
            self.delete_product()
        elif choice == '4':
            self.add_to_cart()
        elif choice == '5':
            self.view_cart()
        elif choice == '6':
            self.place_order()
        elif choice == '7':
            self.view_customer_order()
        elif choice == '8':
            print("Thank you for visiting...We hope to see you again soon!")
            break
        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    app = EcomApp()
    app.run()
```

---

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.

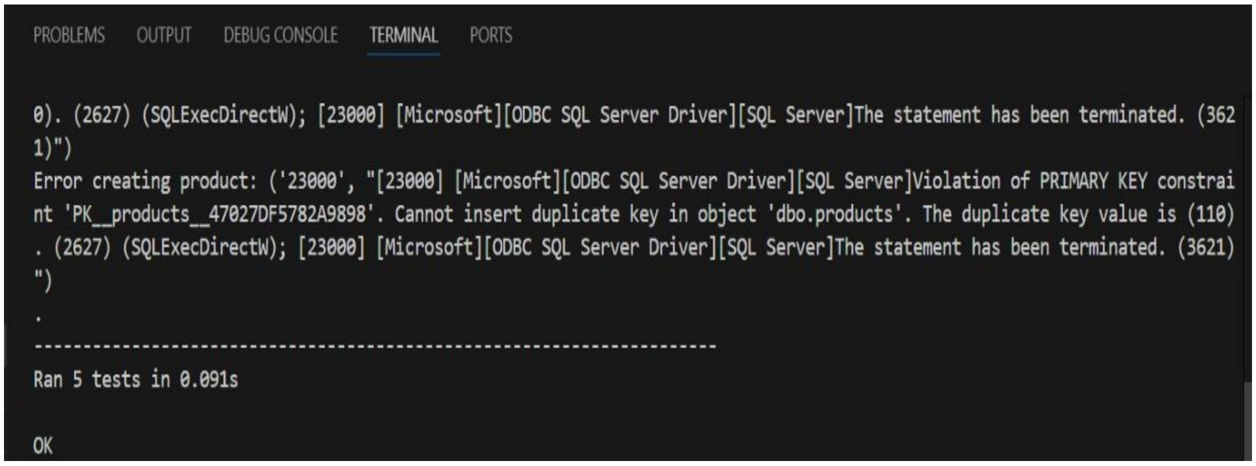
• Write test case to test product is ordered successfully or not.

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

→

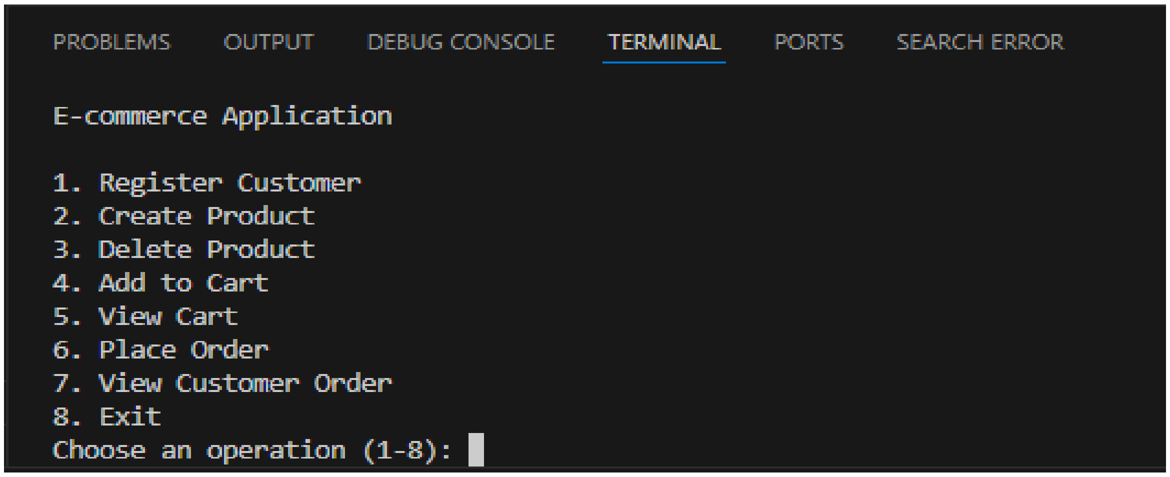
```
import sys
import os
import unittest
from entity.product import Product
from entity.customer import Customer
from dao.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl
from exception.customernotfound import CustomerNotFound
from exception.productnotfound import ProductNotFound
```

```
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(_file_))))


class TestEcommerceSystem(unittest.TestCase):
    def setUp(self):
        self.repository = OrderProcessorRepositoryImpl()



        # Create a sample customer and product for testing
        self.customer = Customer(customer_id=10, name="Aman", email="aman32@gmail.com",
password="pass@45")
        self.product = Product(product_id=110, name="Belt", price=499.99, description="Stylish
belt for students", stockQuantity=20)

        # Add sample customer and product to the repository for testing
        self.repository.create_customer(self.customer)
        self.repository.create_product(self.product)

    def test_product_created_successfully(self):
        """Test case to check if product is created successfully."""
        product = self.repository.get_product_by_id(110)  # Use the new product ID
        self.assertIsNotNone(product)
        self.assertEqual(product.get_name(), "Belt")  # Check product name

    def test_product_added_to_cart_successfully(self):
        """Test case to check if product is added to cart successfully."""
        quantity = 2
        result = self.repository.add_to_cart(self.customer, self.product, quantity)
        self.assertTrue(result)  # Ensure product was added successfully
    def test_product_ordered_successfully(self):
        """Test case to check if product is ordered successfully."""
        quantity = 1
        shipping_address = "456 Oak St"
        result = self.repository.place_order(self.customer, [(self.product, quantity)],
shipping_address)
        self.assertTrue(result)  # Ensure the order was placed successfully

    def test_exception_thrown_when_customer_not_found(self):
        """Test case to check if exception is thrown when customer ID not found."""
        with self.assertRaises(CustomerNotFound):
            self.repository.get_customer_by_id(20)  # Using a non-existent customer ID

    def test_exception_thrown_when_product_not_found(self):
        """Test case to check if exception is thrown when product ID not found."""
        with self.assertRaises(ProductNotFound):
            self.repository.get_product_by_id(200)  # Using a non-existent product ID

if _name_ == "_main_":
    unittest.main()
```

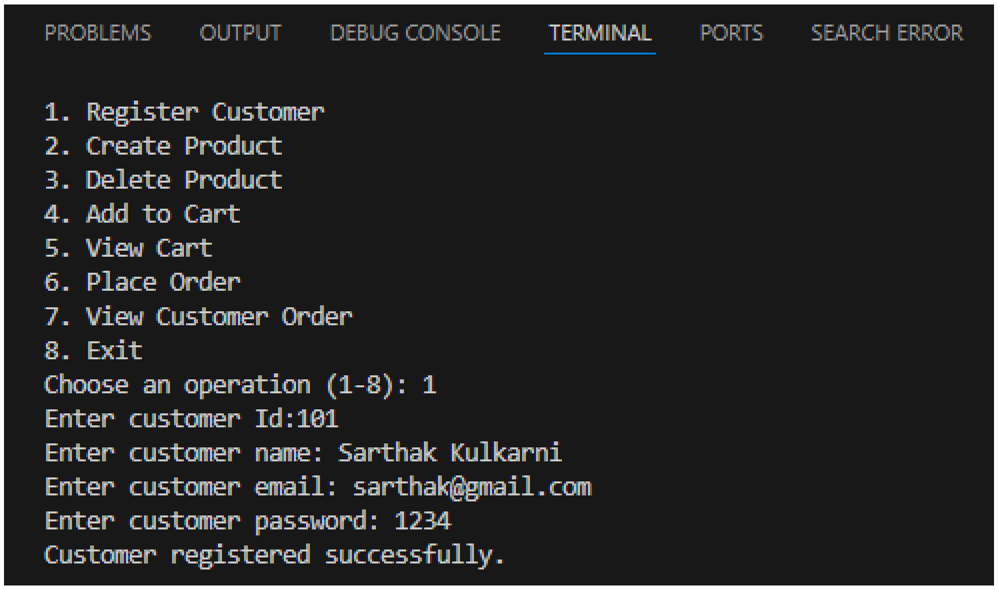


```
0). (2627) (SQLExecDirectW); [23000] [Microsoft][ODBC SQL Server Driver][SQL Server]The statement has been terminated. (362
1)")
Error creating product: ('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constrai
nt 'PK__products__47027DF5782A9898'. Cannot insert duplicate key in object 'dbo.products'. The duplicate key value is (110)
. (2627) (SQLExecDirectW); [23000] [Microsoft][ODBC SQL Server Driver][SQL Server]The statement has been terminated. (3621)
")
.
----------------------------------------------------------------------
Ran 5 tests in 0.091s

OK
```

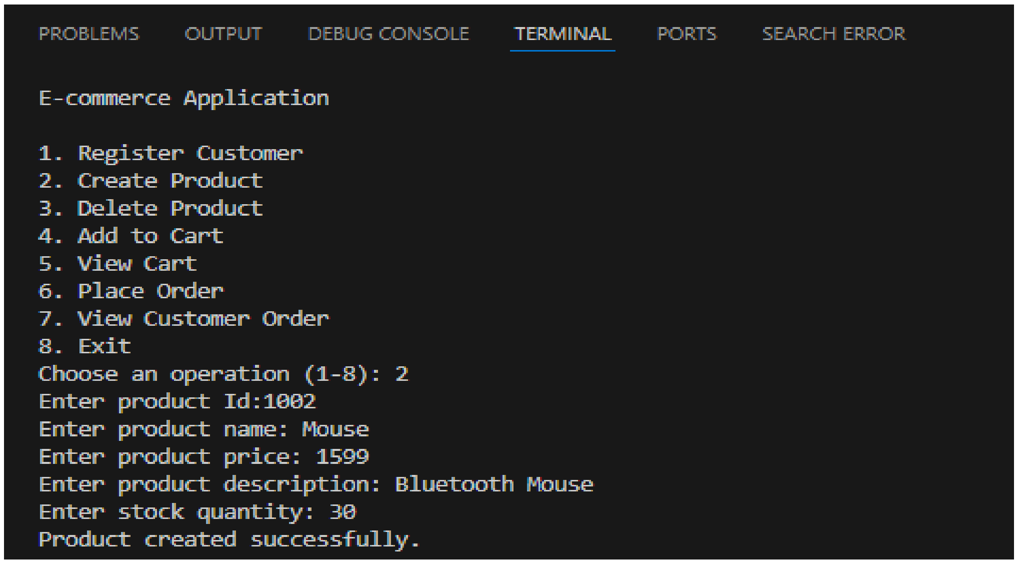
OUTPUTS:-



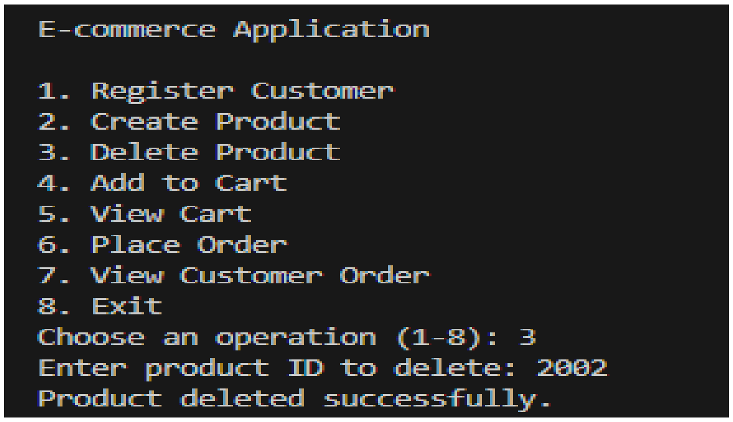
```
E-commerce Application

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
Choose an operation (1-8):
```



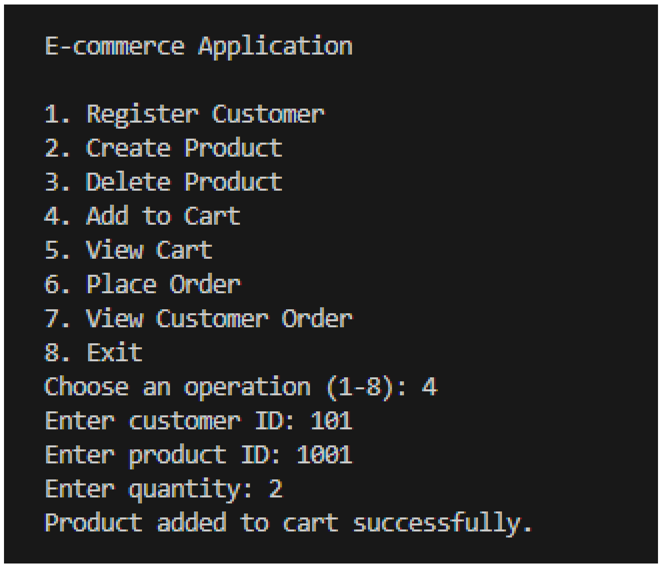
```
1. Register Customer
2. Create Product
3. Delete Product
4. Add to Cart
5. View Cart
6. Place Order
7. View Customer Order
8. Exit
Choose an operation (1-8): 1
Enter customer Id:101
Enter customer name: Sarthak Kulkarni
Enter customer email: sarthak@gmail.com
Enter customer password: 1234
Customer registered successfully.
```



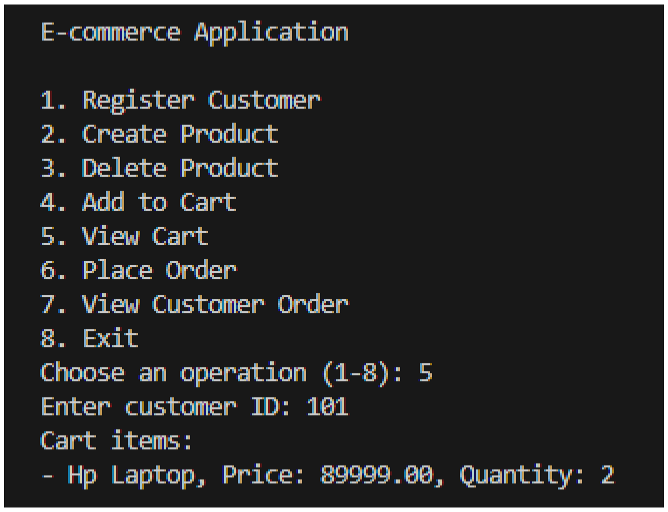
```
E-commerce Application

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
Choose an operation (1-8): 2
Enter product Id:1002
Enter product name: Mouse
Enter product price: 1599
Enter product description: Bluetooth Mouse
Enter stock quantity: 30
Product created successfully.
```



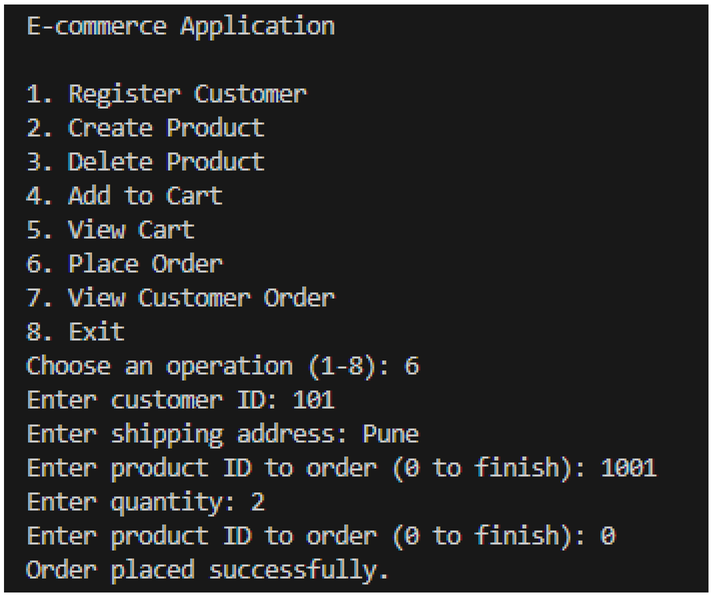
```
E-commerce Application

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
Choose an operation (1-8): 3
Enter product ID to delete: 2002
Product deleted successfully.
```



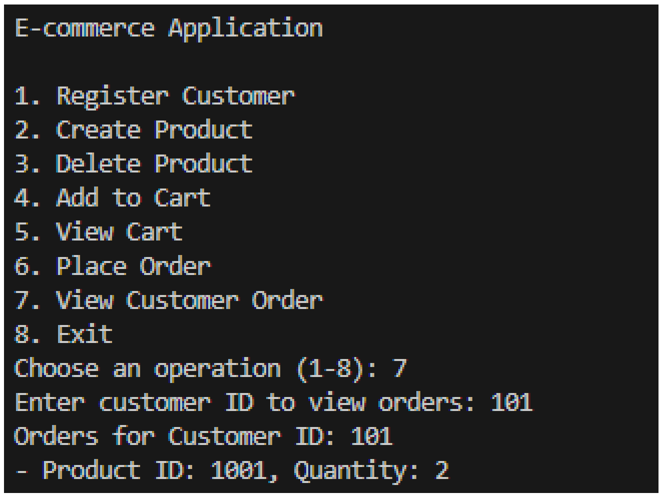
```
E-commerce Application

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
Choose an operation (1-8): 4
Enter customer ID: 101
Enter product ID: 1001
Enter quantity: 2
Product added to cart successfully.
```



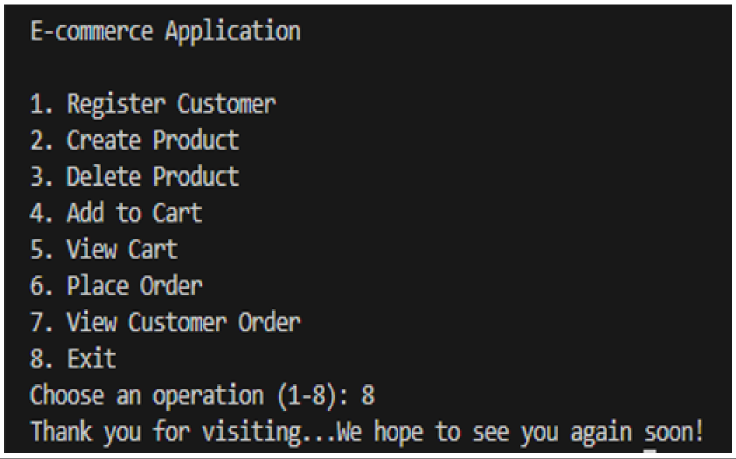
```
E-commerce Application

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
Choose an operation (1-8): 5
Enter customer ID: 101
Cart items:
- Hp Laptop, Price: 89999.00, Quantity: 2
```



```
E-commerce Application

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
Choose an operation (1-8): 6
Enter customer ID: 101
Enter shipping address: Pune
Enter product ID to order (0 to finish): 1001
Enter quantity: 2
Enter product ID to order (0 to finish): 0
Order placed successfully.
```



```
E-commerce Application

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
Choose an operation (1-8): 7
Enter customer ID to view orders: 101
Orders for Customer ID: 101
- Product ID: 1001, Quantity: 2
```



```
E-commerce Application

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
Choose an operation (1-8): 8
Thank you for visiting...We hope to see you again soon!
```