

# CASE STUDY

## Ecom

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<, 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 > order_processor_repository.py > OrderProcessorRepository > place_order
1 from abc import ABC, abstractmethod
2 from typing import List, Dict, Tuple
3 from Entity.model import Product, Customer
4
5 class OrderProcessorRepository(ABC):
6     @abstractmethod
7     def create_product(self, product: Product) -> bool:
8         pass
9
10    @abstractmethod
11    def create_customer(self, customer: Customer) -> bool:
12        pass
13
14    @abstractmethod
15    def delete_product(self, product_id: int) -> bool:
16        pass
17
18    @abstractmethod
19    def delete_customer(self, customer_id: int) -> bool:
20        pass
21
22    @abstractmethod
23    def add_to_cart(self, customer_id: int, product_id: int, quantity: int) -> bool:
24        pass
25
26    @abstractmethod
27    def remove_from_cart(self, customer_id: int, product_id: int) -> bool:
28        pass
29
30    @abstractmethod
31    def get_all_from_cart(self, customer_id: int) -> List[Product]:
32        pass
33
34    @abstractmethod
35    def place_order(self, customer_id: int, products_quantities: List[Tuple[int, int]], shipping_address: str) -> bool:
36        pass
37
38    @abstractmethod
39    def get_orders_by_customer(self, customer_id: int) -> List[Tuple[Product, int]]:
40        pass
```

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS

PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom>

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

```

dao > OrderProcessorRepository/impl.py > OrderProcessorRepositoryImpl > get_db_connection
1 import mysql.connector
2 from mysql.connector import Error
3 from typing import List, Dict, Tuple
4 from Entity.model import Customer, Product
5 from order_processor_repository import OrderProcessorRepository
6 class OrderProcessorRepositoryImpl():
7     def __init__(self):
8         self.connection = self.get_db_connection()
9
10    def get_db_connection(self):
11        connection = None
12        try:
13            connection = mysql.connector.connect(
14                host='localhost',
15                user='root',
16                password='Mghv@1725',
17                database='ecom',
18                port="3306"
19            )
20            if connection.is_connected():
21                print("Connected to MySQL database")
22        except Error as e:
23            print(f"Error connecting to MySQL: {e}")
24        return connection
25    def createProduct(self, product: Product) -> bool:
26        try:
27            with self.connection.cursor() as cursor:
28                sql = "INSERT INTO products (product_id, name, price, description, stockQuantity) VALUES (%s, %s, %s, %s, %s)"
29                cursor.execute(sql, (product.product_id, product.name, product.price, product.description, product.stock_quantity))
30            self.connection.commit()
31            print("Product created successfully.")
32            return True
33        except Error as e:
34            print(f"Error creating product: {e}")
35            return False
36    def createCustomer(self, customer: Customer) -> bool:
37        try:
38            with self.connection.cursor() as cursor:
39                sql = "INSERT INTO customers (name, email, password) VALUES (%s, %s, %s)"
40                cursor.execute(sql, (customer.name, customer.email, customer.password))
41            self.connection.commit()
42            print("Customer created successfully.")
43            return True
44        except Error as e:
45            print(f"Error creating customer: {e}")
46            return False
47    def deleteProduct(self, product_id: int) -> bool:
48        try:
49            with self.connection.cursor() as cursor:
50                sql = "DELETE FROM products WHERE product_id = %s"
51                cursor.execute(sql, (product_id,))
52            self.connection.commit()

```

```

dao > OrderProcessorRepositoryImpl.py > OrderProcessorRepositoryImpl > get_db_connection
6 class OrderProcessorRepositoryImpl():
47     def deleteProduct(self, product_id: int) -> bool:
53         print("Product deleted successfully.")
54         return True
55     except Error as e:
56         print(f"Error deleting product: {e}")
57         return False
58
59     def deleteCustomer(self, customer_id: int) -> bool:
60     try:
61         with self.connection.cursor() as cursor:
62             sql = "DELETE FROM customers WHERE customer_id = %s"
63             cursor.execute(sql, (customer_id,))
64             self.connection.commit()
65             print("Customer deleted successfully.")
66             return True
67     except Error as e:
68         print(f"Error deleting customer: {e}")
69         return False
70
71     def addToCart(self, customer_id: int, product_id: int, quantity: int) -> bool:
72     try:
73         with self.connection.cursor() as cursor:
74             sql = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s, %s)"
75             cursor.execute(sql, (customer_id, product_id, quantity))
76             self.connection.commit()
77             print("Product added to cart successfully.")
78             return True
79     except Error as e:
80         print(f"Error adding product to cart: {e}")
81         return False
82
83     def removeFromCart(self, customer_id: int, product_id: int) -> bool:
84     try:
85         with self.connection.cursor() as cursor:
86             sql = "DELETE FROM cart WHERE customer_id = %s AND product_id = %s"
87             cursor.execute(sql, (customer_id, product_id))
88             self.connection.commit()
89             print("Product removed from cart successfully.")
90             return True
91     except Error as e:
92         print(f"Error removing product from cart: {e}")
93         return False
94
95     def getAllFromCart(self, customer_id: int) -> List[Product]:
96     try:
97         with self.connection.cursor() as cursor:
98             sql = "SELECT * FROM products WHERE product_id IN (SELECT product_id FROM cart WHERE customer_id = %s)"
99             cursor.execute(sql, (customer_id,))
100             products = cursor.fetchall()
101             return [Product(**product) for product in products]
102     except Error as e:

```

```

class OrderProcessorRepositoryImpl():
    def getAllFromCart(self, customer_id: int) -> List[Product]:
        print(f"Error retrieving products from cart: {e}")
        return []

    def placeOrder(self, customer_id: int, products_quantities: List[Tuple[int, int]], shipping_address: str) -> bool:
        try:
            with self.connection.cursor() as cursor:
                order_sql = "INSERT INTO orders (customer_id, order_date, shipping_address) VALUES (%s, NOW(), %s)"
                cursor.execute(order_sql, (customer_id, shipping_address))
                order_id = cursor.lastrowid
                order_item_sql = "INSERT INTO order_items (order_id, product_id, quantity) VALUES (%s, %s, %s)"

                for product_id, quantity in products_quantities:
                    cursor.execute(order_item_sql, (order_id, product_id, quantity))

            self.connection.commit()
            print("Order placed successfully.")
            return True
        except Error as e:
            print(f"Error placing order: {e}")
            self.connection.rollback()
            return False

    def getOrdersByCustomer(self, customer_id: int) -> List[Tuple[Product, int]]:
        try:
            with self.connection.cursor() as cursor:
                sql = """SELECT p.*, oi.quantity
                        FROM order_items oi
                        JOIN products p ON oi.product_id = p.product_id
                        JOIN orders o ON oi.order_id = o.order_id
                        WHERE o.customer_id = %s"""
                cursor.execute(sql, (customer_id,))
                order_items = cursor.fetchall()
                return [(Product(**item), item['quantity']) for item in order_items]
        except Error as e:
            print(f"Error retrieving orders by customer: {e}")
            return []

def main():
    while True:
        print("\nChoose an operation:")
        print("1. Create Product")
        print("2. Create Customer")
        print("3. Delete Product")
        print("4. Delete Customer")
        print("5. Add to Cart")
        print("6. Remove from Cart")
        print("7. View Cart")
        print("8. Place Order")
        print("9. Get Orders By Customer")
        print("10. Exit")

```

```

def main():
    print("9. Get Orders By Customer")
    print("10. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        prod_id = int(input("Enter product ID: "))
        product_name = input("Enter product name: ")
        product_price = float(input("Enter product price: "))
        product_description = input("Enter product description: ")
        product_quantity = int(input("Enter product quantity: "))
        product = Product(product_id=prod_id, name=product_name, price=product_price, description=product_description, stock_quantity=product_quantity)
        order_processor.createProduct(product)

    elif choice == '2':
        customer_name = input("Enter customer name: ")
        customer_email = input("Enter customer email: ")
        customer_password = input("Enter customer password: ")
        customer = Customer(name=customer_name, email=customer_email, password=customer_password)
        order_processor.createCustomer(customer)

    elif choice == '3':
        product_id = int(input("Enter product ID to delete: "))
        order_processor.deleteProduct(product_id)

    elif choice == '4':
        customer_id = int(input("Enter customer ID to delete: "))
        order_processor.deleteCustomer(customer_id)

    elif choice == '5':
        customer_id = int(input("Enter customer ID: "))
        product_id = int(input("Enter product ID to add to cart: "))
        quantity = int(input("Enter quantity: "))
        order_processor.addToCart(customer_id, product_id, quantity)

    elif choice == '6':
        customer_id = int(input("Enter customer ID: "))
        product_id = int(input("Enter product ID to remove from cart: "))
        order_processor.removeFromCart(customer_id, product_id)

    elif choice == '7':
        customer_id = int(input("Enter customer ID: "))
        cart_items = order_processor.getAllFromCart(customer_id)
        print("Items in cart:")
        for item in cart_items:
            print(item.name)

    elif choice == '8':
        customer_id = int(input("Enter customer ID: "))
        shipping_address = input("Enter shipping address: ")
        products_quantities = []

```

```

140 def main():
141     shipping_address = input("Enter shipping address: ")
142     products_quantities = []
143     while True:
144         product_id = int(input("Enter product ID (0 to stop): "))
145         if product_id == 0:
146             break
147         quantity = int(input("Enter quantity: "))
148         products_quantities.append((product_id, quantity))
149         order_processor.placeOrder(customer_id, products_quantities, shipping_address)
150
151     elif choice == '9':
152         customer_id = int(input("Enter customer ID: "))
153         orders = order_processor.getOrdersByCustomer(customer_id)
154         print("Orders by customer:")
155         for order in orders:
156             print(order[0].name, "-", order[1], "quantity")
157
158     elif choice == '10':
159         print("Exiting...")
160         break
161
162     else:
163         print("Invalid choice. Please enter a number between 1 and 10.")
164
165 if __name__ == "__main__":
166     order_processor = OrderProcessorRepositoryImpl()
167     main()
168

```

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS

PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom> python -u "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\dao\OrderProcessorRepositoryImpl.py"

Connected to MySQL database

Choose an operation:

1. Create Product
2. Create Customer
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. Get Orders By Customer
10. Exit

Enter your choice: 1

Enter product ID: 13

Enter product name: SAMSUNG

Enter product price: 18000

Enter product description: F23 New Launch - Monster series

Enter product quantity: 100

Product created successfully.

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 > ...
1 import mysql.connector
2 from util.property import PropertyUtil
3
4 class DBConnection:
5     connection = None
6
7     @staticmethod
8     def getConnection():
9         if DBConnection.connection is None:
10             properties = PropertyUtil.getPropertyString('connection.properties')
11             print(properties)
12             try:
13                 DBConnection.connection = mysql.connector.connect(
14                     host=properties['hostname'],
15                     user=properties['username'],
16                     password=properties['password'],
17                     database=properties['dbname'],
18                     port=properties['port']
19                 )
20                 print("Database connected!")
21             except mysql.connector.Error as e:
22                 print("Error connecting to database:", e)
23             return DBConnection.connection
24
```

```
connection.py > ...
1
2 from util.DBConnection import DBConnection
3 connection = DBConnection.getConnection()
4
5 connection.close()
```

```
util > property.py > PropertyUtil > getPropertyString
1 class PropertyUtil:
2     @staticmethod
3     def getPropertyString(file_path):
4         properties = {}
5         with open(r'C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\util\connection_properties', 'r') as file:
6             for line in file:
7                 if '=' in line:
8                     key, value = line.strip().split('=')
9                     properties[key.strip()] = value.strip()
10         return properties
11
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS
● PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom> python -u "C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\connection.py"
{'hostname': 'localhost', 'dbname': 'ecom', 'username': 'root', 'password': 'Mghw@1725', 'port': '3306'}
Database connected!
○ PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom> |
```



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

```
OrderProcessorRepositoryImpl.py  myexception.py X  order_processor_repository.py  model.py
dao > Exception > myexception.py > OrderNotFoundException > __init__
1  class CustomerNotFoundException(Exception):
2      def __init__(self, message="Customer not found."):
3          self.message = message
4          super().__init__(self.message)
5
6  class ProductNotFoundException(Exception):
7      def __init__(self, message="Product not found."):
8          self.message = message
9          super().__init__(self.message)
10
11 class OrderNotFoundException(Exception):
12     def __init__(self, message="Order not found."):
13         self.message = message
14         super().__init__(self.message)
```

```
Choose an operation:
1. Create Product
2. Create Customer
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. Get Orders By Customer
10. Exit
Enter your choice: 9
Enter customer ID: 156
Traceback (most recent call last):
  File "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\dao\OrderProcessorRepositoryImpl.py", line 242, in <module>
    main()
  File "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\dao\OrderProcessorRepositoryImpl.py", line 228, in main
    orders = order_processor.getOrdersByCustomer(customer_id)
              ~~~~~
  File "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\dao\OrderProcessorRepositoryImpl.py", line 151, in getOrdersByCustomer
    raise OrderNotFoundException("Order not found for the given customer.")
OrderNotFoundException: Order not found for the given customer.
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom>
```

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

```
dao > Main > main.py > EcomApp > main
1  from dao import OrderProcessorRepositoryImpl
2  from dao.Entity.model import Customer, Product
3  from dao.myexception import CustomerNotFoundException, ProductNotFoundException
4
5
6  class EcomApp:
7      def __init__(self):
8          self.order_processor = OrderProcessorRepositoryImpl()
9
10     def main(self):
11         while True:
12             print("\nChoose an operation:")
13             print("1. Register Customer")
14             print("2. Create Product")
15             print("3. Delete Product")
16             print("4. Add to Cart")
17             print("5. View Cart")
18             print("6. Place Order")
19             print("7. View Customer Order")
20             print("8. Exit")
21
22             choice = input("Enter your choice: ")
23
24             if choice == '1':
25                 self.register_customer()
26             elif choice == '2':
27                 self.create_product()
28             elif choice == '3':
29                 self.delete_product()
30             elif choice == '4':
31                 self.add_to_cart()
32             elif choice == '5':
33                 self.view_cart()
34             elif choice == '6':
35                 self.place_order()
36             elif choice == '7':
37                 self.view_customer_order()
38             elif choice == '8':
39                 print("Exiting...")
40                 break
41             else:
42                 print("Invalid choice. Please enter a number between 1 and 8.")
43
44     def register_customer(self):
45         name = input("Enter customer name: ")
46         email = input("Enter customer email: ")
47         password = input("Enter customer password: ")
48         customer = Customer(name, email, password)
49         if self.order_processor.create_customer(customer):
50             print("Customer registered successfully.")
51         else:
52             print("Failed to register customer.")
53
54     def create_product(self):
55         name = input("Enter product name: ")
56         price = float(input("Enter product price: "))
57         description = input("Enter product description: ")
58         stock_quantity = int(input("Enter product stock quantity: "))
59         product = Product(name, price, description, stock_quantity)
60         if self.order_processor.create_product(product):
61             print("Product created successfully.")
62         else:
63             print("Failed to create product.")
64
```

```

def delete_product(self):
    product_id = int(input("Enter product ID to delete: "))
    if self.order_processor.delete_product(product_id):
        print("Product deleted successfully.")
    else:
        print("Failed to delete product.")

def add_to_cart(self):
    # Assuming customer and product IDs are known
    customer_id = int(input("Enter customer ID: "))
    product_id = int(input("Enter product ID to add to cart: "))
    quantity = int(input("Enter quantity: "))
    try:
        customer = self.order_processor.get_customer_by_id(customer_id)
        product = self.order_processor.get_product_by_id(product_id)
        if self.order_processor.add_to_cart(customer, product, quantity):
            print("Product added to cart successfully.")
        else:
            print("Failed to add product to cart.")
    except (CustomerNotFoundException, ProductNotFoundException) as e:
        print(e)

def view_cart(self):
    # Assuming customer ID is known
    customer_id = int(input("Enter customer ID to view cart: "))
    try:
        customer = self.order_processor.get_customer_by_id(customer_id)
        cart_items = self.order_processor.get_all_from_cart(customer)
        print("Cart Items:")
        for item in cart_items:
            print(f"{item['product'].name} - Quantity: {item['quantity']}")
    except CustomerNotFoundException as e:
        print(e)

def place_order(self):
    # Assuming customer ID is known
    customer_id = int(input("Enter customer ID to place order: "))
    shipping_address = input("Enter shipping address: ")
    try:
        customer = self.order_processor.get_customer_by_id(customer_id)
        cart_items = self.order_processor.get_all_from_cart(customer)
        if self.order_processor.place_order(customer, cart_items, shipping_address):
            print("Order placed successfully.")
        else:
            print("Failed to place order.")
    except CustomerNotFoundException as e:
        print(e)

def view_customer_order(self):
    # Assuming customer ID is known
    customer_id = int(input("Enter customer ID to view orders: "))
    try:
        orders = self.order_processor.get_orders_by_customer(customer_id)
        print("Customer Orders:")
        for order in orders:
            print(
                f"Order ID: {order['order_id']}, Total Price: {order['total_price']}, Order Date: {order['order_date']}"
            )
    except CustomerNotFoundException as e:
        print(e)

if __name__ == "__main__":
    App = EcomApp()
    App.main()

```

```
01 // ... print("Product deleted successfully");
PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS

7. View Customer Order
8. Exit
Enter your choice: 1
Enter Customer_id : 123
Enter customer name: asada
Enter customer email: sd
Enter customer password: dfg
Enter Customer's Address : adfag
Customer created successfully.
Customer registered successfully.

Choose an operation:
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: █
```

## Unit Testing

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.

```

testing_dummy.py > OrderProcessorRepositoryImpl > addToCart
import mysql.connector
from mysql.connector import Error
from myexception import CustomerNotFoundException, ProductNotFoundException
class OrderProcessorRepositoryImpl:
    def __init__(self):
        self.connection = self.connect_to_database()

    def connect_to_database(self):
        try:
            connection = mysql.connector.connect(
                host='localhost',
                database='ecom',
                user='root',
                password='Mghv@1725'
            )
            if connection.is_connected():
                print("Connected to database successfully.")
                return connection
        except Error as e:
            print(f"Error connecting to database: {e}")
    def customer_exist(self, customer_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            sql = "SELECT COUNT(*) FROM customers WHERE customer_id = %s"
            cursor.execute(sql, (customer_id,))
            count = cursor.fetchone()[0]
            return count > 0
        except Error as e:
            print(f"Error checking customer existence: {e}")
            return False
        finally:
            if cursor:
                cursor.close()
    def product_exist(self, product_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            sql = "SELECT COUNT(*) FROM products WHERE product_id = %s"
            cursor.execute(sql, (product_id,))
            count = cursor.fetchone()[0]
            return count > 0

```

```

class OrderProcessorRepositoryImpl:
    def product_exist(self, product_id: int) -> bool:
        except Error as e:
            print(f"Error checking customer existence: {e}")
            return False
        finally:
            if cursor:
                cursor.close()
    def create_product(self, product_id, name, price, description, stock_quantity):
        try:
            cursor = self.connection.cursor()
            sql = "INSERT INTO products (product_id, name, price, description, stockQuantity) VALUES (%s, %s, %s, %s, %s)"
            cursor.execute(sql, (product_id, name, price, description, stock_quantity))
            self.connection.commit()
            print("Product created successfully.")
            return True
        except Error as e:
            print(f"Error creating product: {e}")
            return False
        finally:
            if cursor:
                cursor.close()
    def addToCart(self, customer_id: int, product_id: int, quantity: int) -> bool:
        try:
            if not self.customer_exist(customer_id):
                raise CustomerNotFoundException(f"Customer with ID {customer_id} not found.")
            elif not self.product_exist(product_id):
                raise ProductNotFoundException(f"Product with ID {product_id} not found.")
            with self.connection.cursor() as cursor:
                sql = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s, %s)"
                cursor.execute(sql, (customer_id, product_id, quantity))
            self.connection.commit()
            print("Product added to cart successfully.")
            return True
        except Error as e:
            print(f"Error adding product to cart: {e}")
            return False
    def orderProduct(self, cart_id: int) -> bool:
        try:

```

```

        cursor = self.connection.cursor()
        sql = "SELECT * FROM cart WHERE cart_id = %s"
        cursor.execute(sql, (cart_id,))
        result = cursor.fetchall()
        if result:
            print("Product ordered successfully.")
            return True
        else:
            print("Order not found in cart.")
            return False
    except Error as e:
        print(f"Error ordering product: {e}")
        return False
    finally:
        if cursor:
            cursor.close()
def __del__(self):
    if self.connection:
        self.connection.close()
        print("Database connection closed.")

```

```

OK
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom> python -u "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom\dao\test_unit.py"
Connected to database successfully.
.Database connection closed.
Connected to database successfully.
Product added to cart successfully.
.Database connection closed.
Connected to database successfully.
Error creating product: 1062 (23000): Duplicate entry '12' for key 'products.PRIMARY'
.Database connection closed.
Connected to database successfully.
.Database connection closed.
Connected to database successfully.
Product ordered successfully.
.Database connection closed.

-----
Ran 5 tests in 0.454s

OK
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\ecom>

```