Case Study on Ecommerce Application

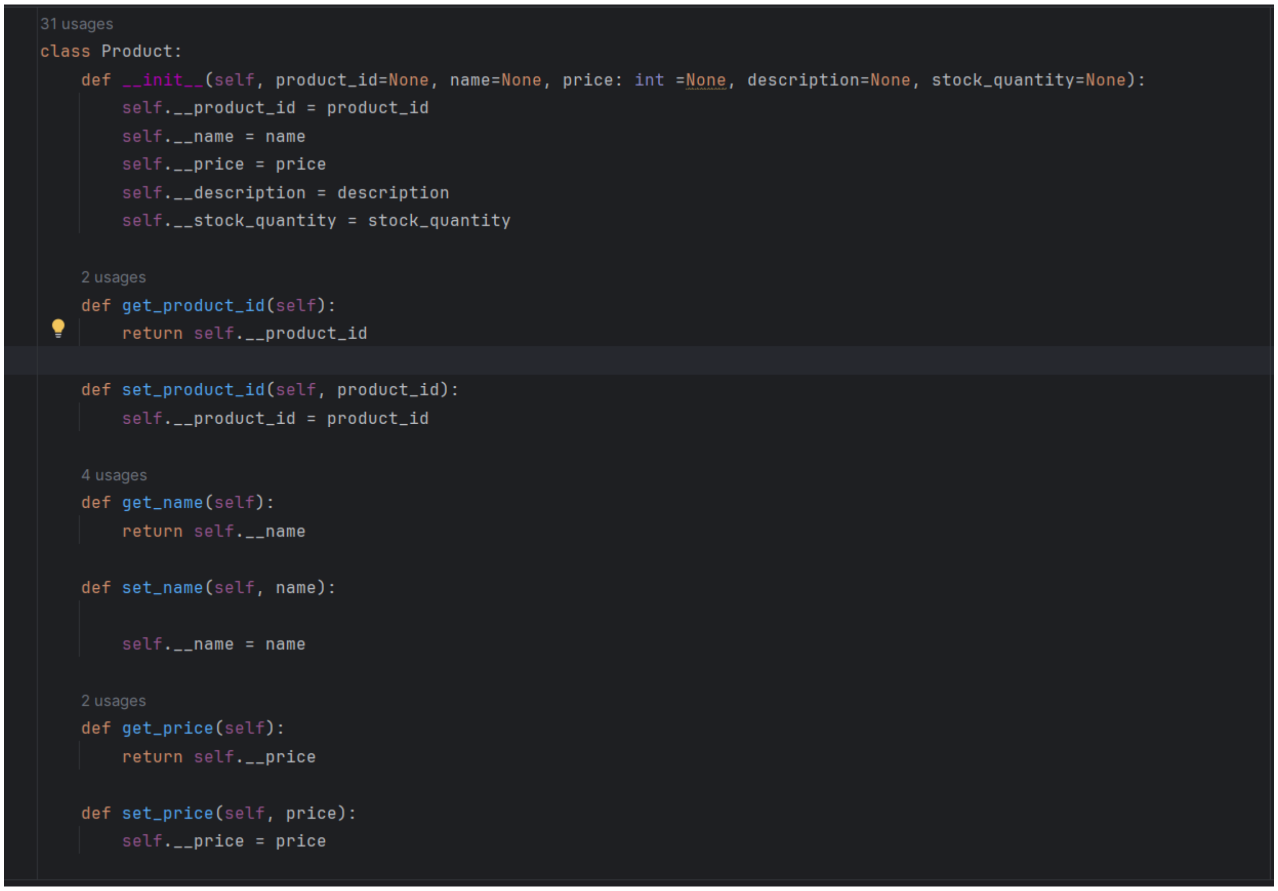
submitted by – Sankar Roy

Creating classes in python :

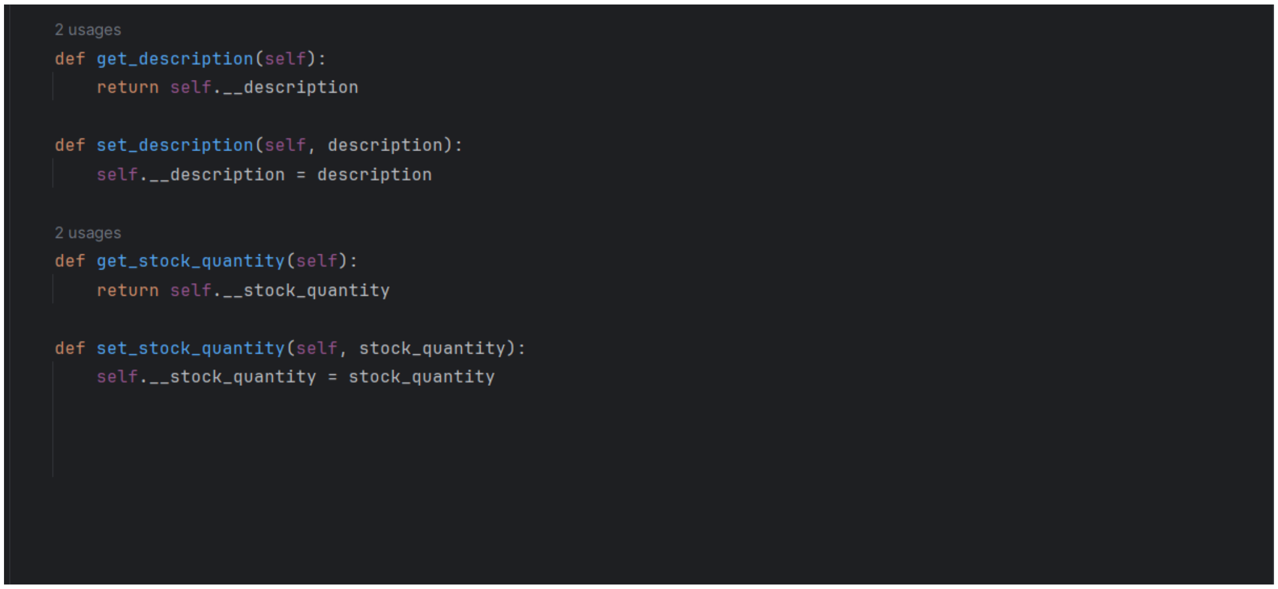
The following classes were made for the application

- Product
- Customer
- Cart
- Order
- Order item

Product –

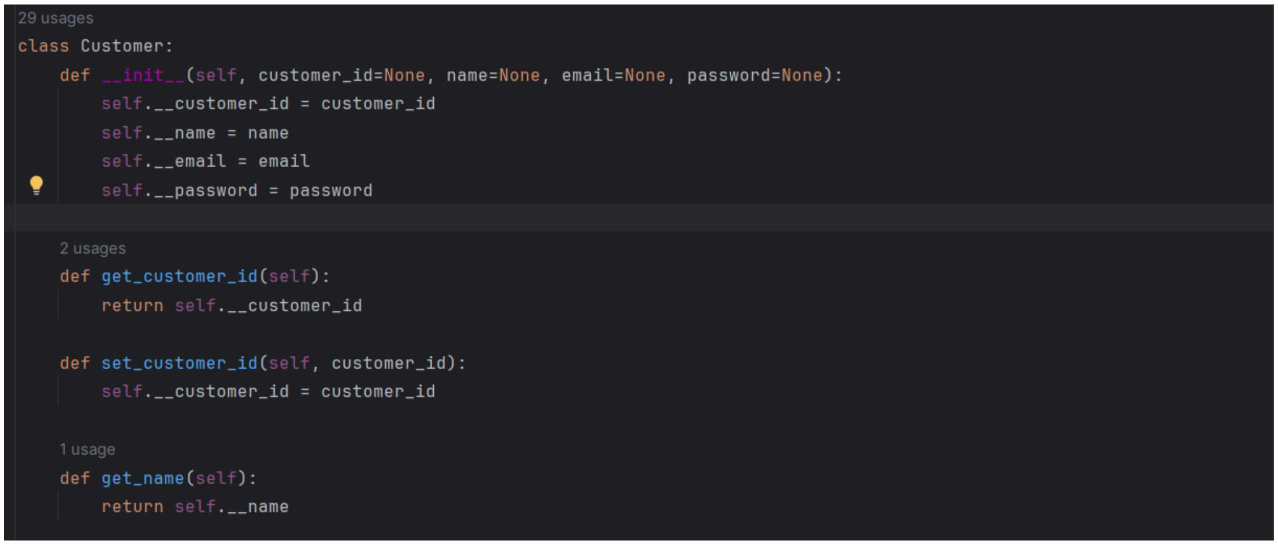


```
31 usages
class Product:
    def __init__(self, product_id=None, name=None, price: int =None, description=None, stock_quantity=None):
        self.__product_id = product_id
        self.__name = name
        self.__price = price
        self.__description = description
        self.__stock_quantity = stock_quantity

    2 usages
    def get_product_id(self):
        return self.__product_id

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

    4 usages
    def get_name(self):
        return self.__name

    def set_name(self, name):

        self.__name = name

    2 usages
    def get_price(self):
        return self.__price

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

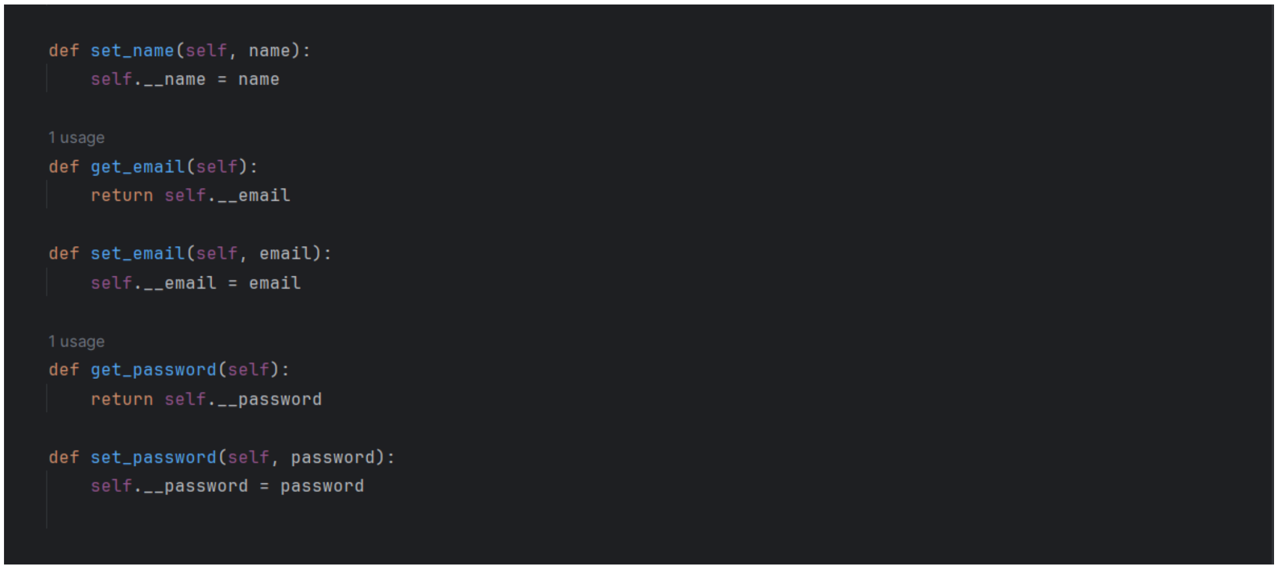


```
2 usages
def get_description(self):
    return self.__description

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

2 usages
def get_stock_quantity(self):
    return self.__stock_quantity

def set_stock_quantity(self, stock_quantity):
    self.__stock_quantity = stock_quantity
```

<u>Customer</u> –

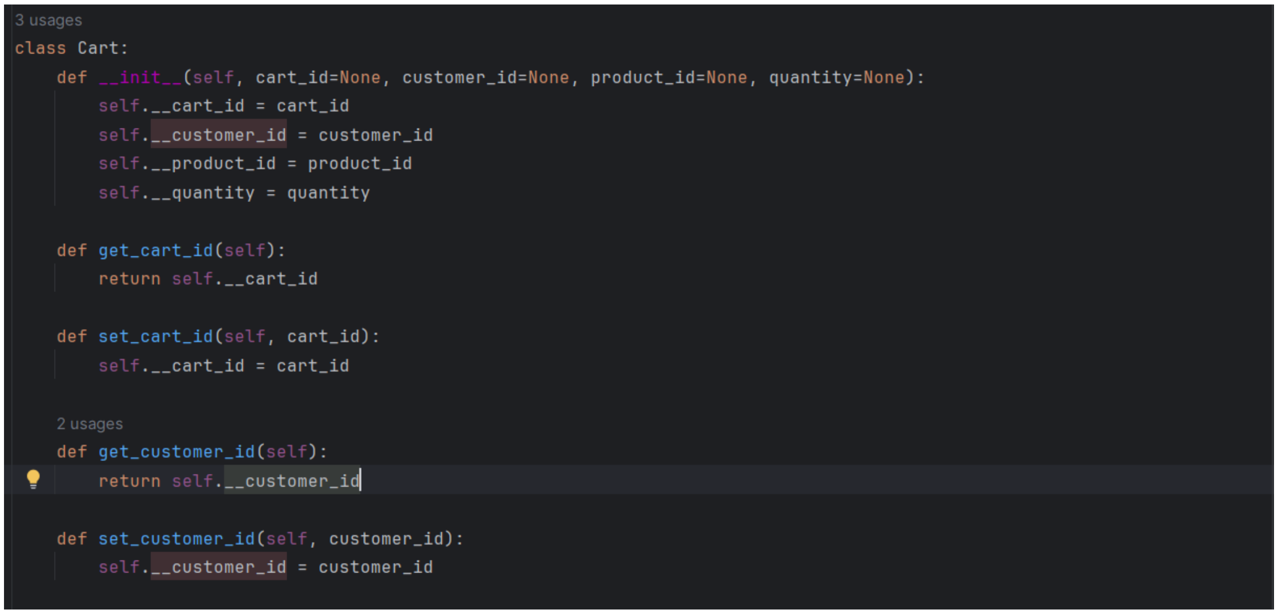


```
29 usages
class Customer:
    def __init__(self, customer_id=None, name=None, email=None, password=None):
        self.__customer_id = customer_id
        self.__name = name
        self.__email = email
        self.__password = password

    2 usages
    def get_customer_id(self):
        return self.__customer_id

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

    1 usage
    def get_name(self):
        return self.__name
```

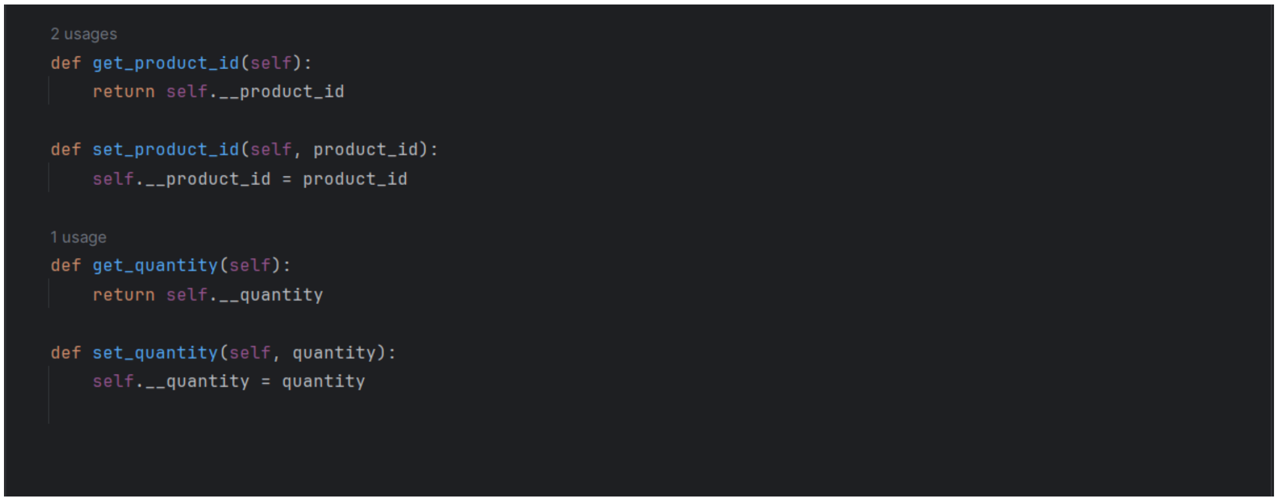


```
def set_name(self, name):
    self.__name = name

1 usage
def get_email(self):
    return self.__email

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

1 usage
def get_password(self):
    return self.__password

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

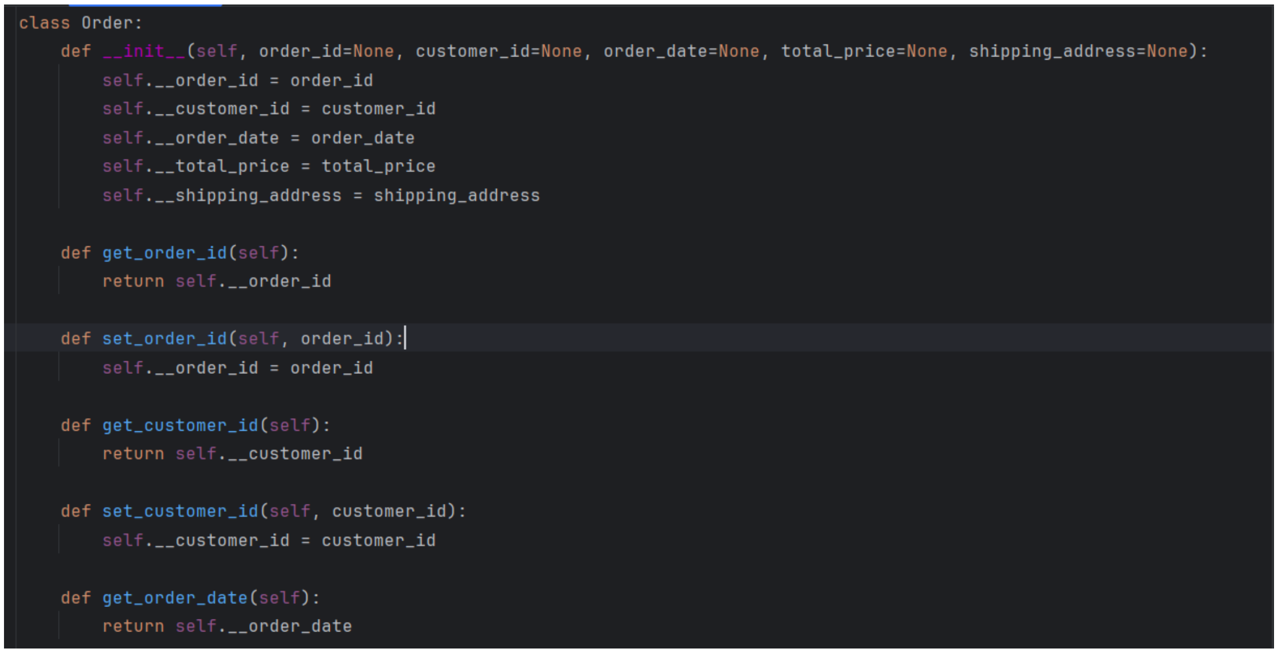
Cart -

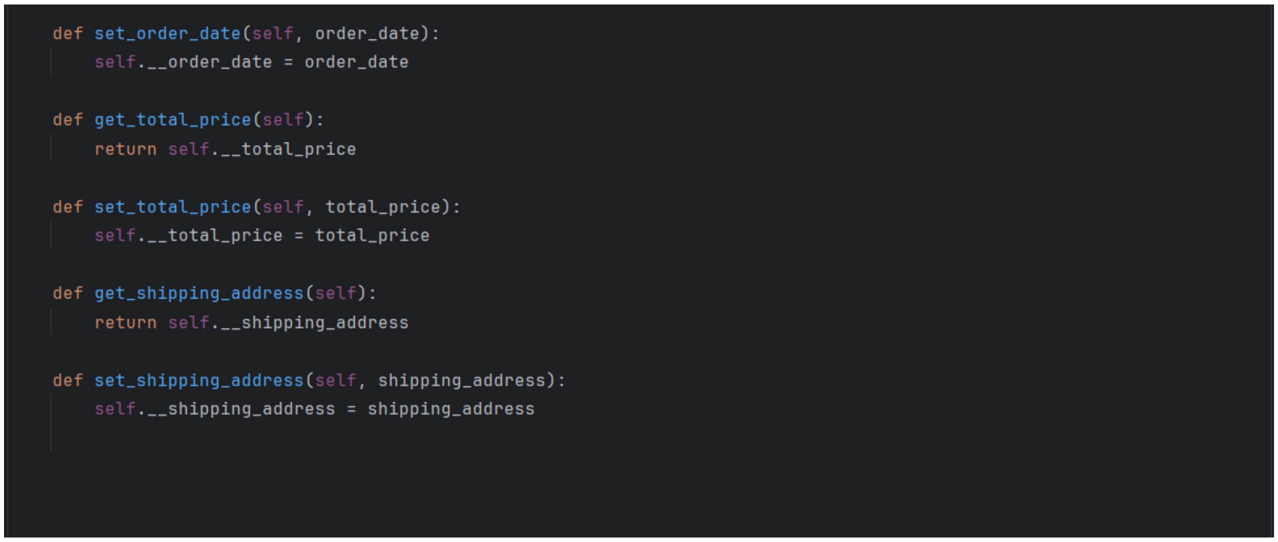


```
3 usages
class Cart:
    def __init__(self, cart_id=None, customer_id=None, product_id=None, quantity=None):
        self.__cart_id = cart_id
        self.__customer_id = customer_id
        self.__product_id = product_id
        self.__quantity = quantity

    def get_cart_id(self):
        return self.__cart_id

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

    2 usages
    def get_customer_id(self):
        return self.__customer_id

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


```
2 usages
def get_product_id(self):
    return self.__product_id

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

1 usage
def get_quantity(self):
    return self.__quantity

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

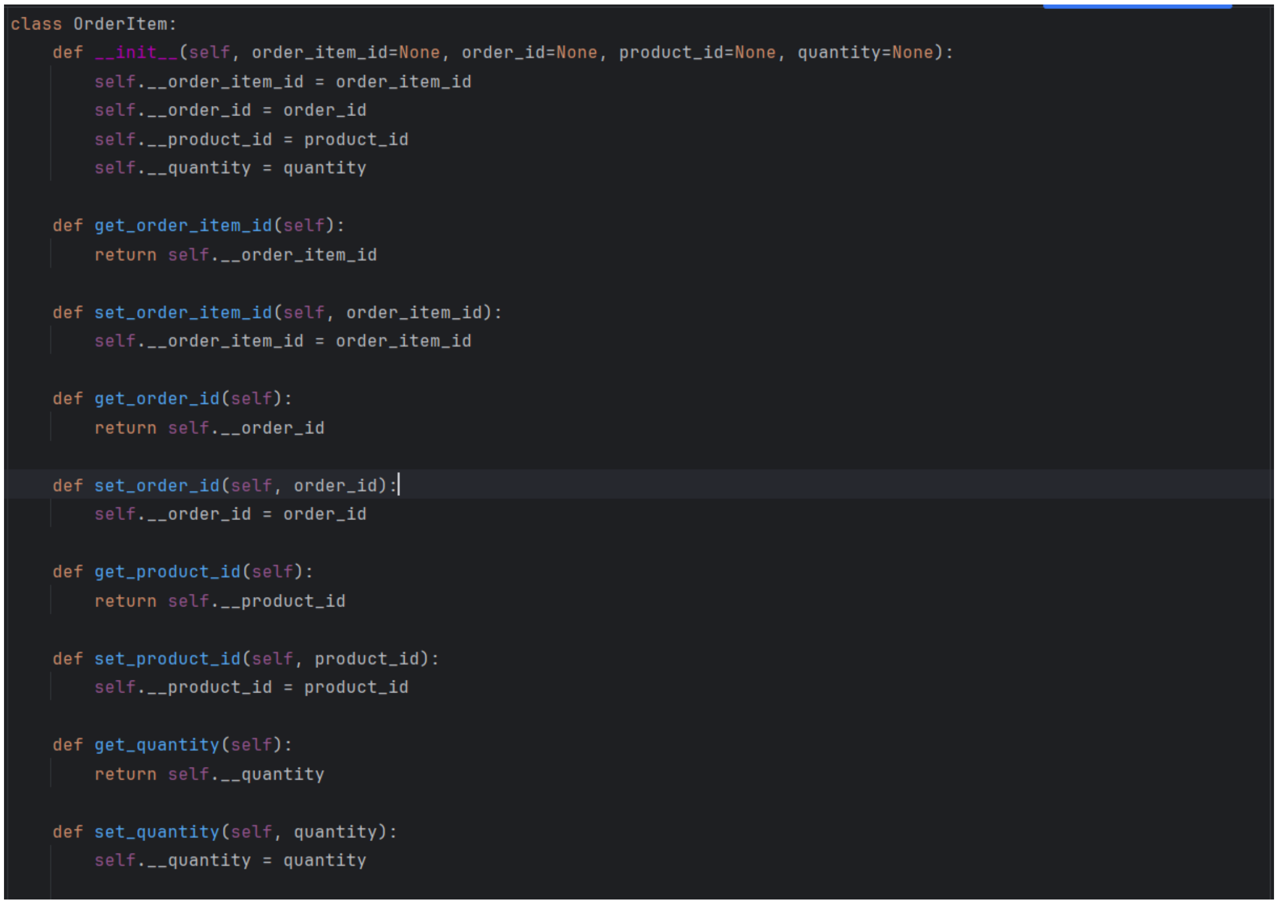
Order –


```
class Order:
    def __init__(self, order_id=None, customer_id=None, order_date=None, total_price=None, shipping_address=None):
        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
```



```python
    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
```

<u>Order Item</u> –



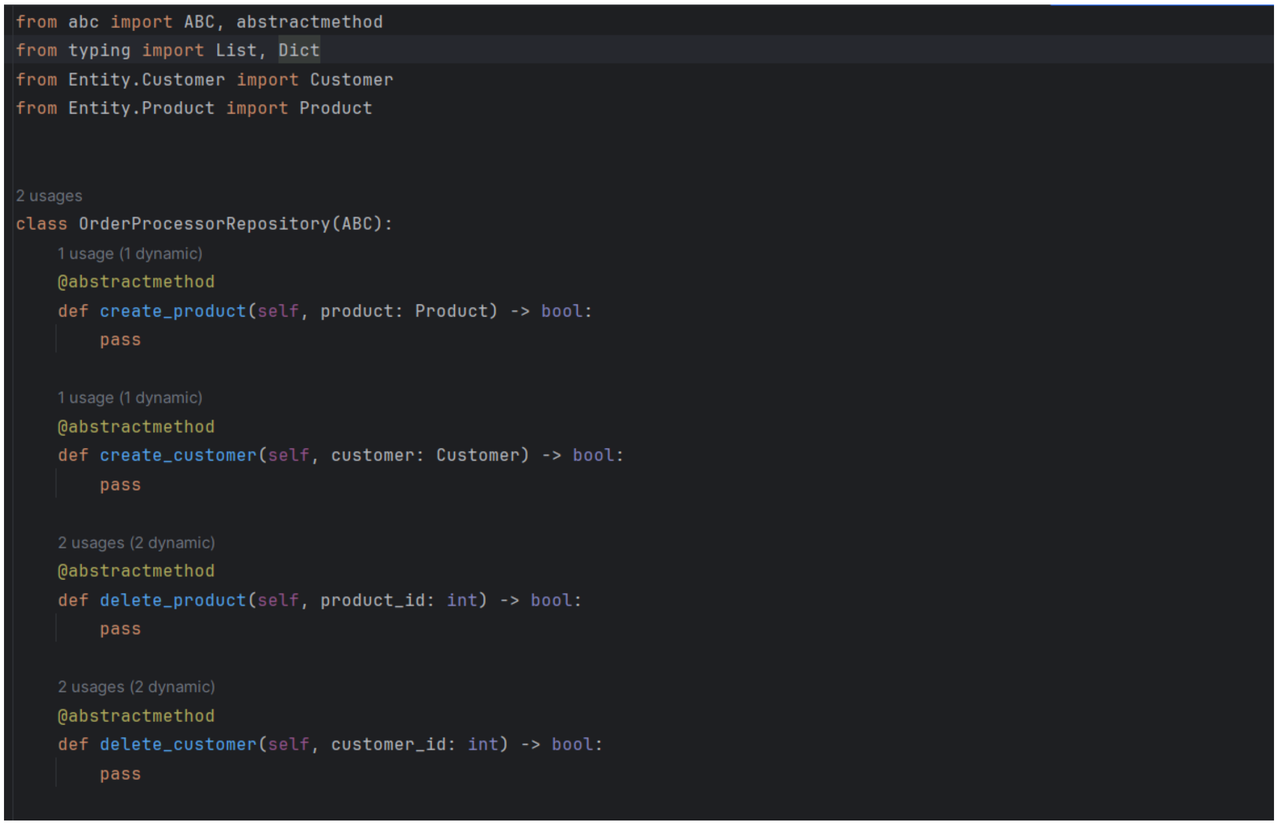
```python
class OrderItem:
    def __init__(self, order_item_id=None, order_id=None, product_id=None, quantity=None):
        self.__order_item_id = order_item_id
        self.__order_id = order_id
        self.__product_id = product_id
        self.__quantity = quantity

    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
```

© Hexaware Technologies Limited. All rights

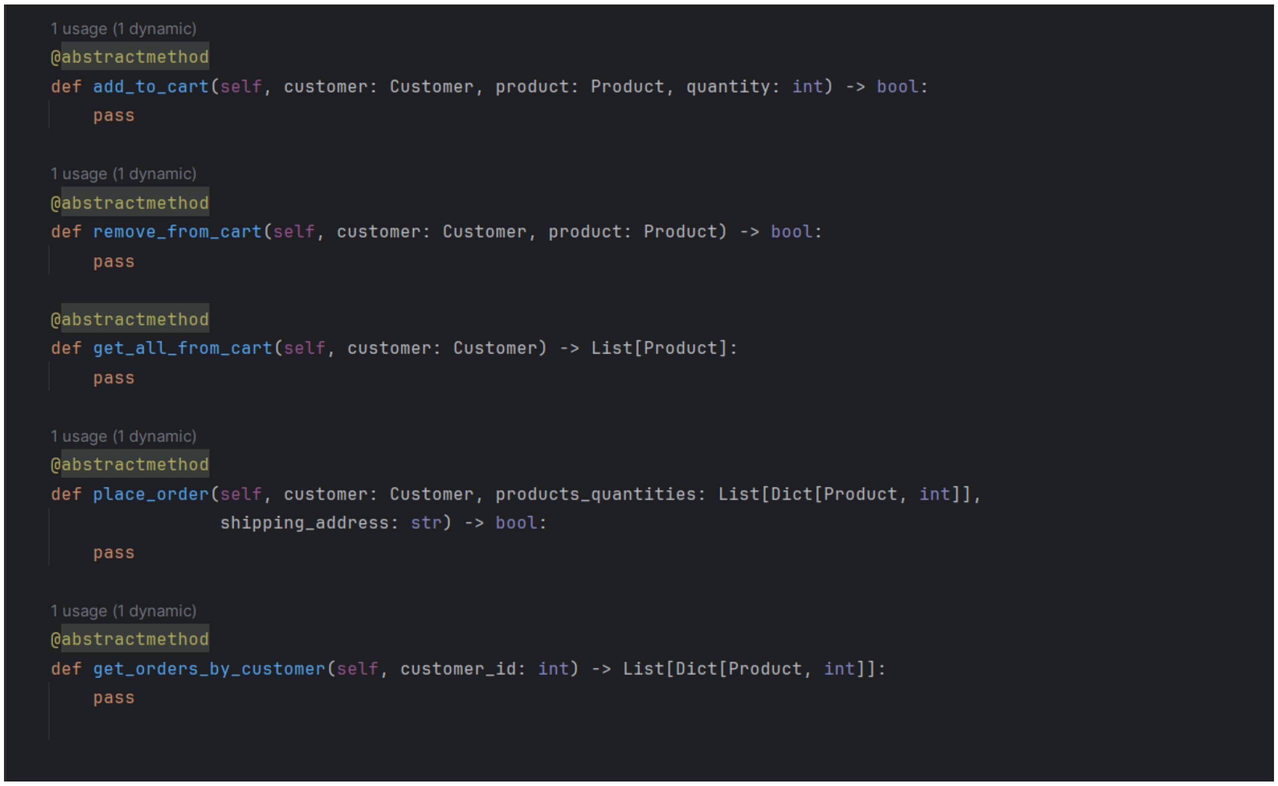
Creating dao package :

In the dao package we will be storing two classes

- OrderProcessorRepository – this is an interface class which will hold all the abstract method that are required for our application to run

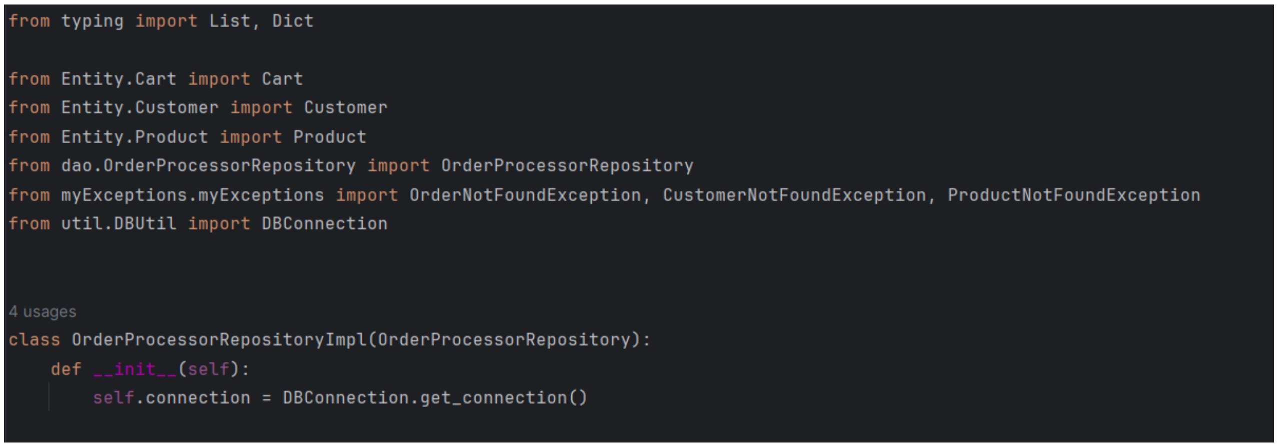


```
from abc import ABC, abstractmethod
from typing import List, Dict
from Entity.Customer import Customer
from Entity.Product import Product


2 usages
class OrderProcessorRepository(ABC):
    1 usage (1 dynamic)
    @abstractmethod
    def create_product(self, product: Product) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def create_customer(self, customer: Customer) -> bool:
        pass

    2 usages (2 dynamic)
    @abstractmethod
    def delete_product(self, product_id: int) -> bool:
        pass

    2 usages (2 dynamic)
    @abstractmethod
    def delete_customer(self, customer_id: int) -> bool:
        pass
```



```
    1 usage (1 dynamic)
    @abstractmethod
    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def remove_from_cart(self, customer: Customer, product: Product) -> bool:
        pass

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

    1 usage (1 dynamic)
    @abstractmethod
    def place_order(self, customer: Customer, products_quantities: List[Dict[Product, int]],
                    shipping_address: str) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def get_orders_by_customer(self, customer_id: int) -> List[Dict[Product, int]]:
        pass
```

- OrderProcessorRepositoryImpl – this is the most important class which will hold the implementation and logic that runs the actual methods created

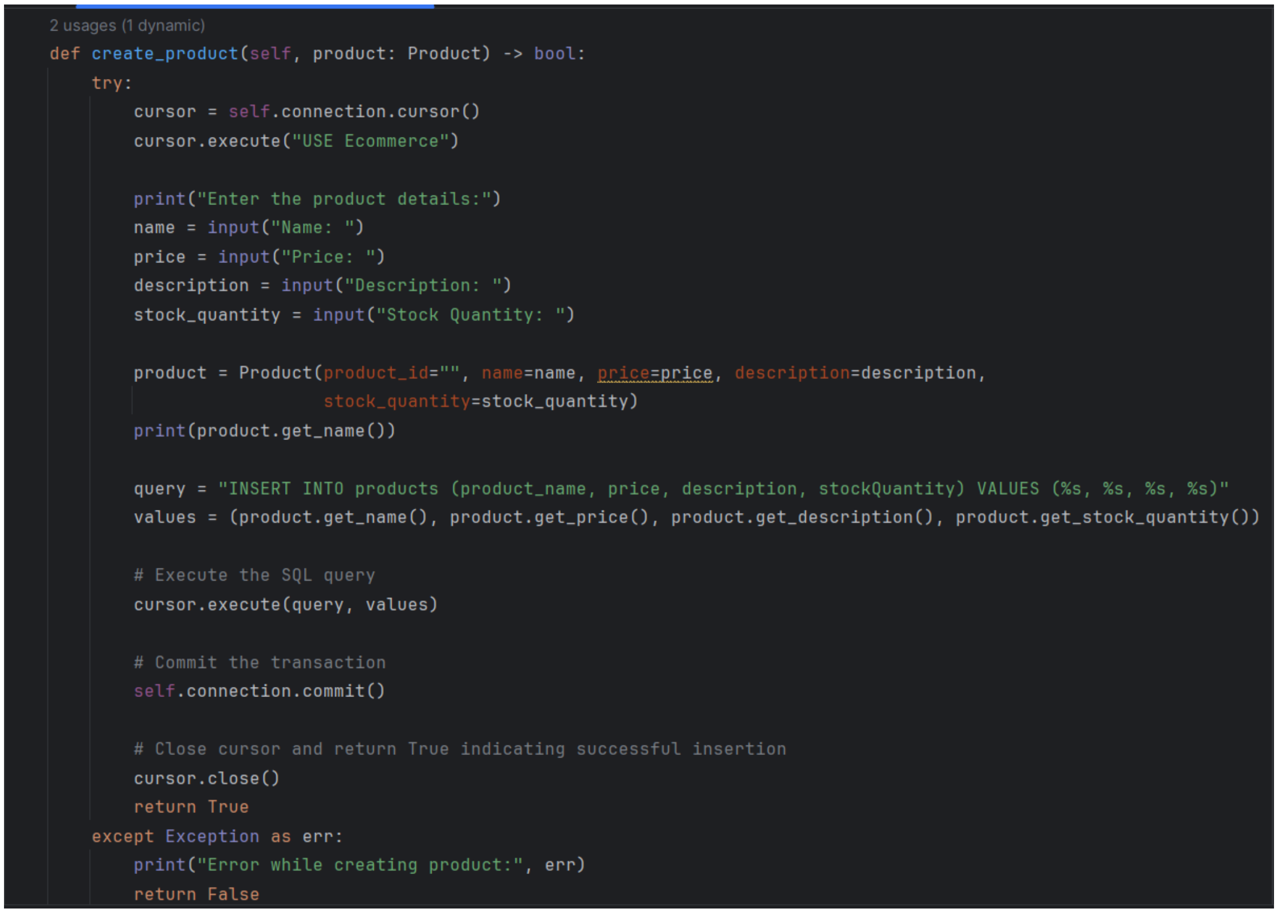


```
from typing import List, Dict

from Entity.Cart import Cart
from Entity.Customer import Customer
from Entity.Product import Product
from dao.OrderProcessorRepository import OrderProcessorRepository
from myExceptions.myExceptions import OrderNotFoundException, CustomerNotFoundException, ProductNotFoundException
from util.DBUtil import DBConnection


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

In the OrderProcessorRepositoryImpl all the classes are implemented that are mentioned in the abstract class and also will be given to the user as a menu to choose from

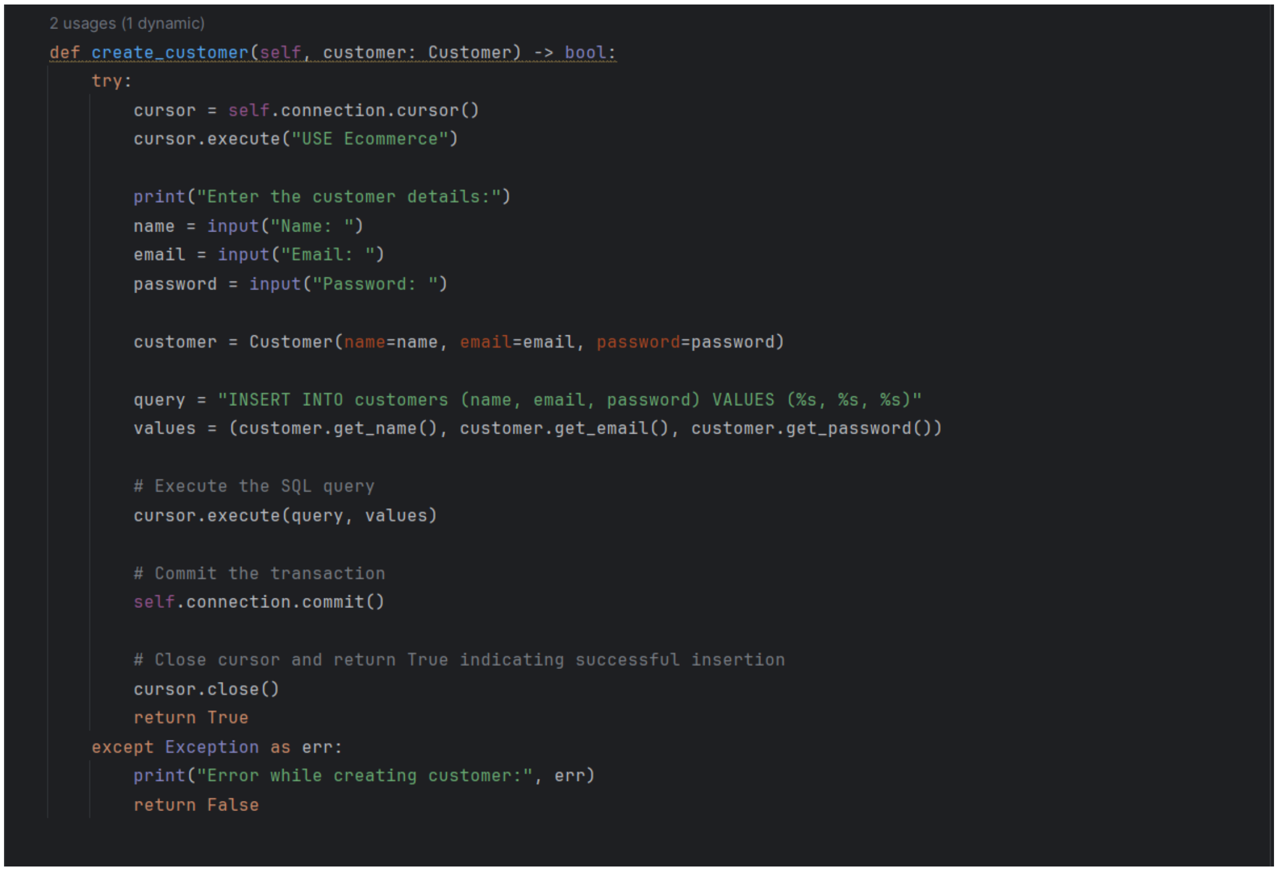
The methods are as follows :

1. create_product



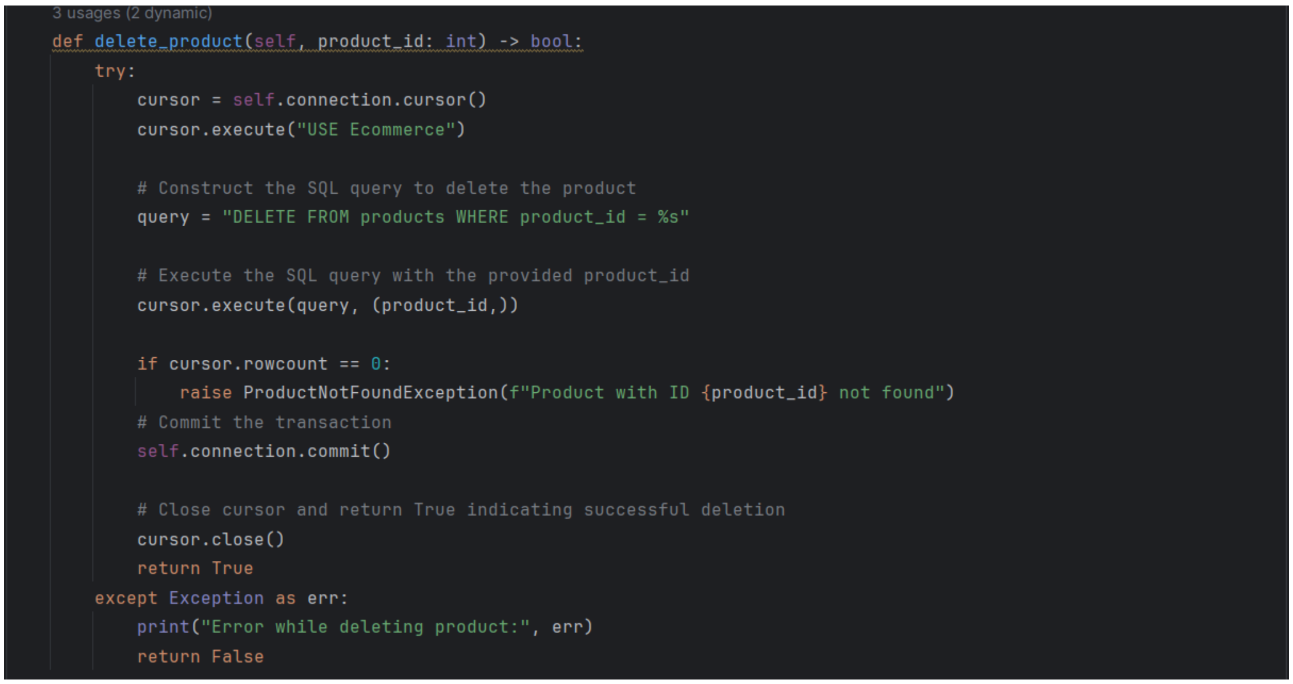
```
    2 usages (1 dynamic)
    def create_product(self, product: Product) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("USE Ecommerce")

            print("Enter the product details:")
            name = input("Name: ")
            price = input("Price: ")
            description = input("Description: ")
            stock_quantity = input("Stock Quantity: ")

            product = Product(product_id="", name=name, price=price, description=description,
                              stock_quantity=stock_quantity)
            print(product.get_name())

            query = "INSERT INTO products (product_name, price, description, stockQuantity) VALUES (%s, %s, %s, %s)"
            values = (product.get_name(), product.get_price(), product.get_description(), product.get_stock_quantity())

            # Execute the SQL query
            cursor.execute(query, values)

            # Commit the transaction
            self.connection.commit()

            # Close cursor and return True indicating successful insertion
            cursor.close()
            return True
        except Exception as err:
            print("Error while creating product:", err)
            return False
```

2. create_customer



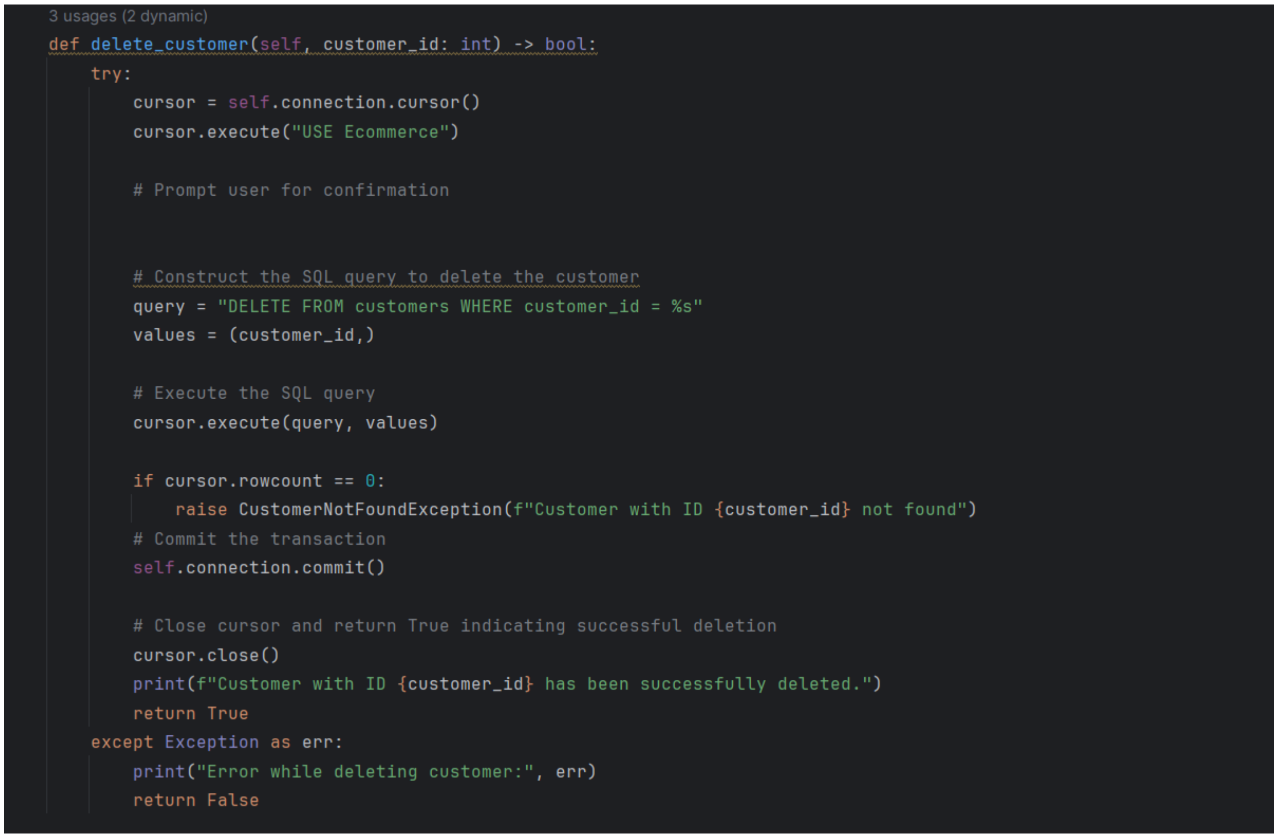
```
2 usages (1 dynamic)
def create_customer(self, customer: Customer) -> bool:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        print("Enter the customer details:")
        name = input("Name: ")
        email = input("Email: ")
        password = input("Password: ")

        customer = Customer(name=name, email=email, password=password)

        query = "INSERT INTO customers (name, email, password) VALUES (%s, %s, %s)"
        values = (customer.get_name(), customer.get_email(), customer.get_password())

        # Execute the SQL query
        cursor.execute(query, values)

        # Commit the transaction
        self.connection.commit()

        # Close cursor and return True indicating successful insertion
        cursor.close()
        return True
    except Exception as err:
        print("Error while creating customer:", err)
        return False
```

3. delete_product



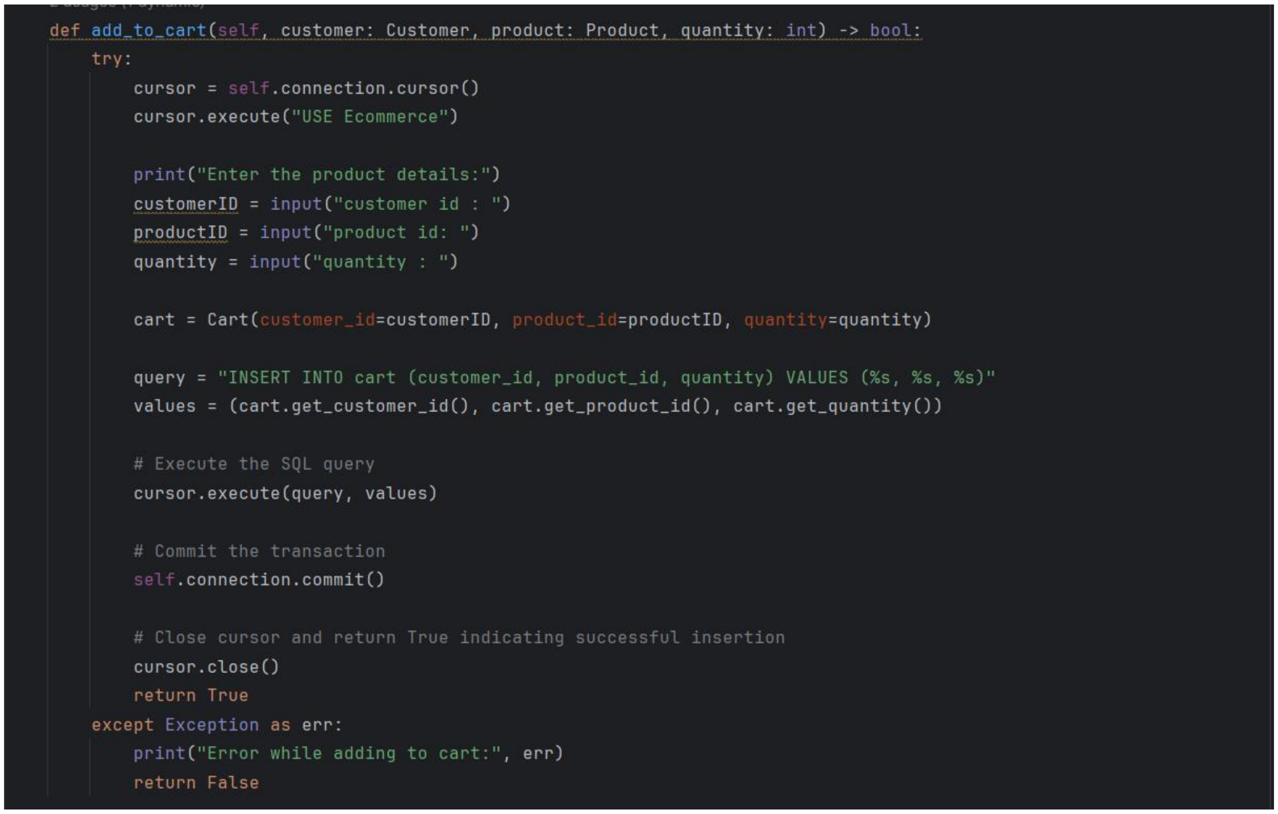
```
3 usages (2 dynamic)
def delete_product(self, product_id: int) -> bool:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        # Construct the SQL query to delete the product
        query = "DELETE FROM products WHERE product_id = %s"

        # Execute the SQL query with the provided product_id
        cursor.execute(query, (product_id,))

        if cursor.rowcount == 0:
            raise ProductNotFoundException(f"Product with ID {product_id} not found")
        # Commit the transaction
        self.connection.commit()

        # Close cursor and return True indicating successful deletion
        cursor.close()
        return True
    except Exception as err:
        print("Error while deleting product:", err)
        return False
```

4. delete_customer



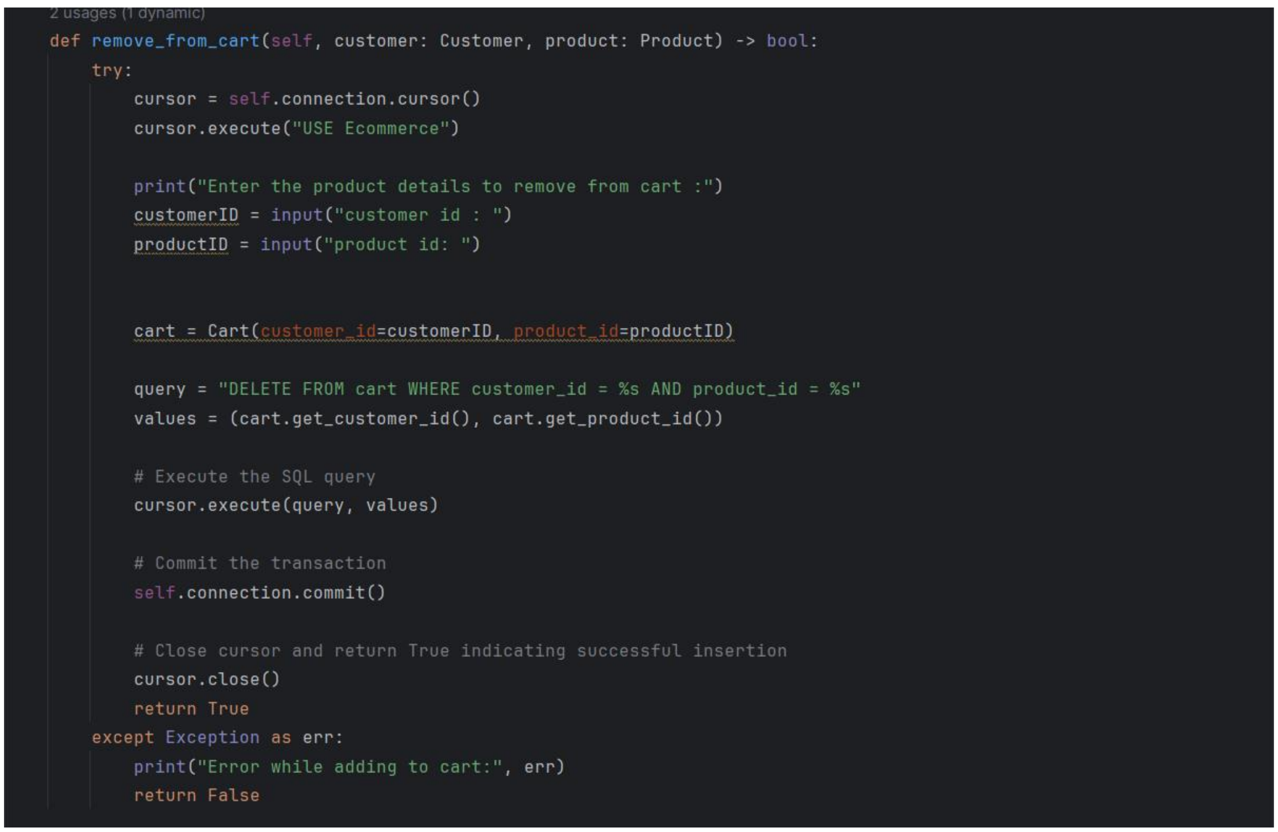
```
3 usages (2 dynamic)
def delete_customer(self, customer_id: int) -> bool:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        # Prompt user for confirmation


        # Construct the SQL query to delete the customer
        query = "DELETE FROM customers WHERE customer_id = %s"
        values = (customer_id,)

        # Execute the SQL query
        cursor.execute(query, values)

        if cursor.rowcount == 0:
            raise CustomerNotFoundException(f"Customer with ID {customer_id} not found")
        # Commit the transaction
        self.connection.commit()

        # Close cursor and return True indicating successful deletion
        cursor.close()
        print(f"Customer with ID {customer_id} has been successfully deleted.")
        return True
    except Exception as err:
        print("Error while deleting customer:", err)
        return False
```

5. add_to_cart



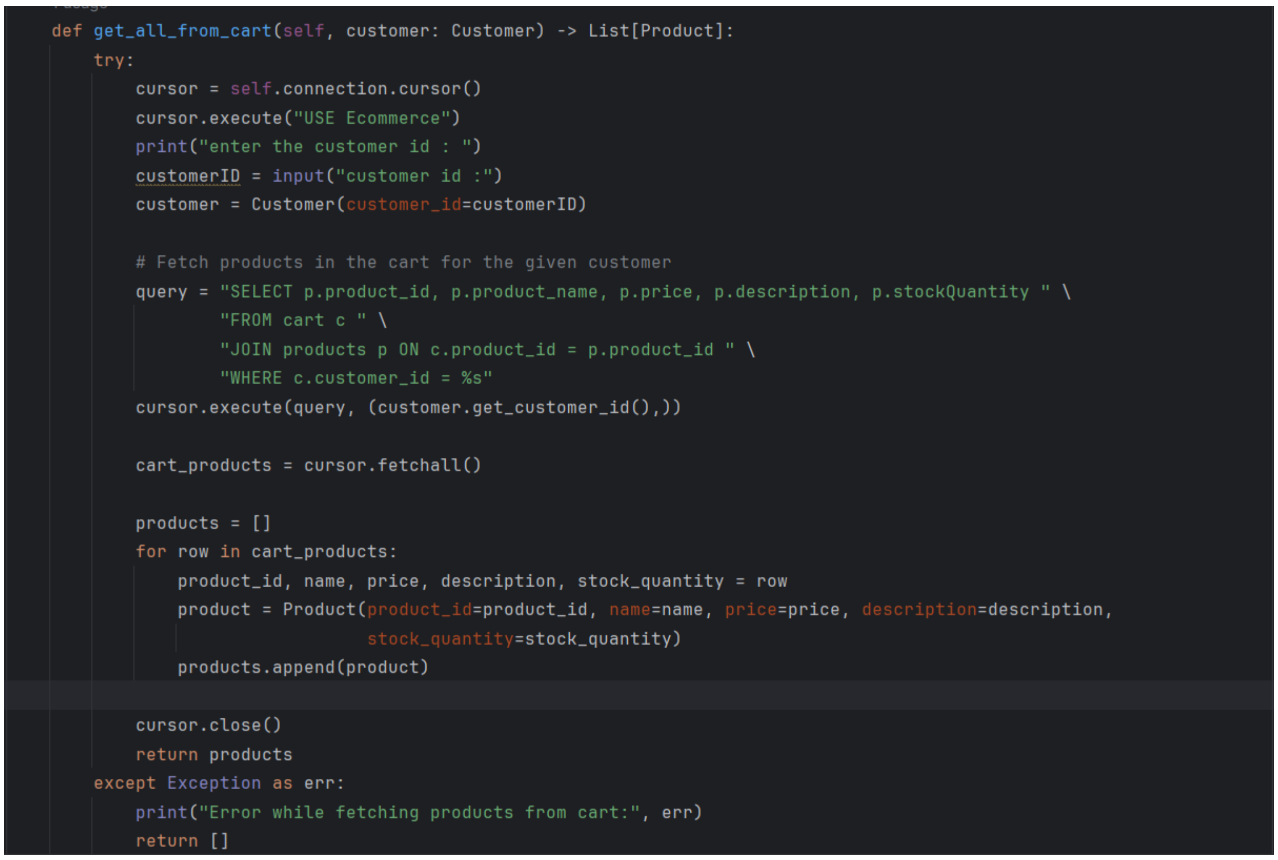
```
def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        print("Enter the product details:")
        customerID = input("customer id : ")
        productID = input("product id: ")
        quantity = input("quantity : ")

        cart = Cart(customer_id=customerID, product_id=productID, quantity=quantity)

        query = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s, %s)"
        values = (cart.get_customer_id(), cart.get_product_id(), cart.get_quantity())

        # Execute the SQL query
        cursor.execute(query, values)

        # Commit the transaction
        self.connection.commit()

        # Close cursor and return True indicating successful insertion
        cursor.close()
        return True
    except Exception as err:
        print("Error while adding to cart:", err)
        return False
```

6. remove_from_cart



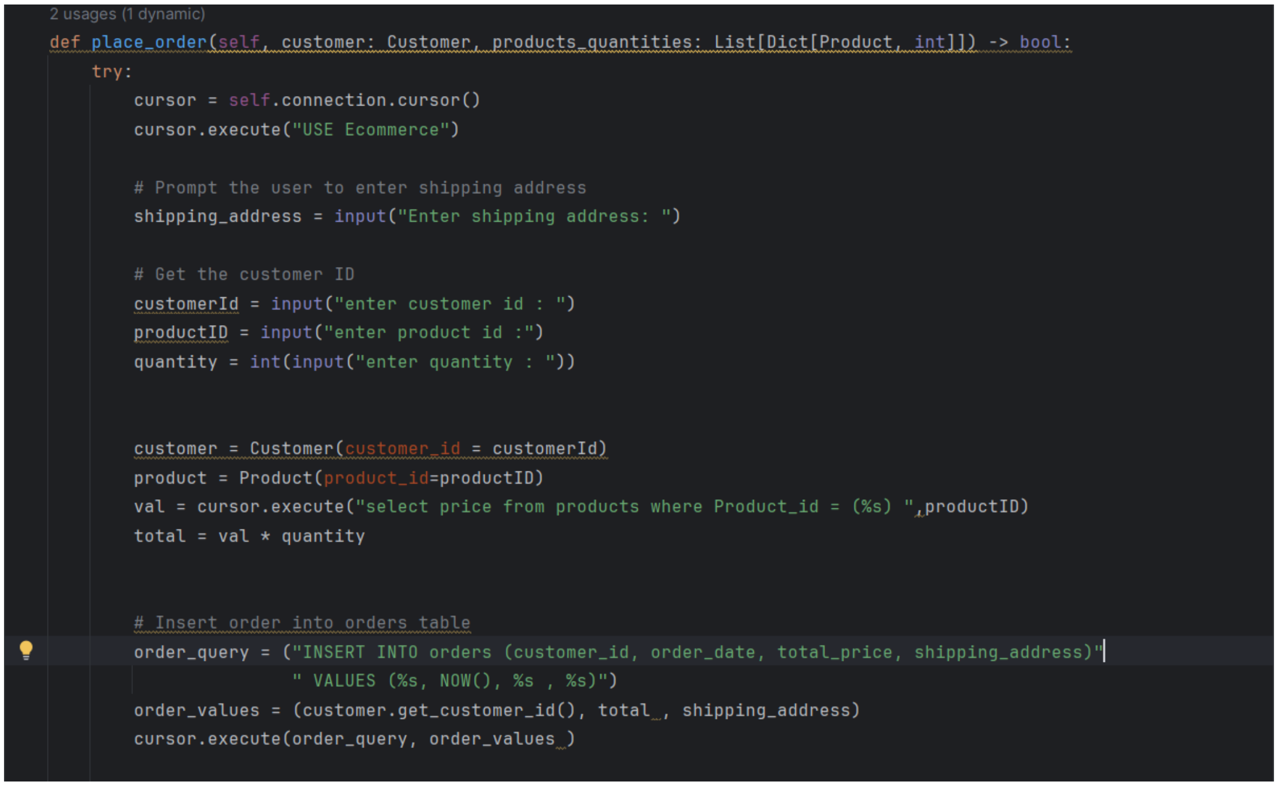
```
2 usages (1 dynamic)
def remove_from_cart(self, customer: Customer, product: Product) -> bool:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        print("Enter the product details to remove from cart :")
        customerID = input("customer id : ")
        productID = input("product id: ")


        cart = Cart(customer_id=customerID, product_id=productID)

        query = "DELETE FROM cart WHERE customer_id = %s AND product_id = %s"
        values = (cart.get_customer_id(), cart.get_product_id())

        # Execute the SQL query
        cursor.execute(query, values)

        # Commit the transaction
        self.connection.commit()

        # Close cursor and return True indicating successful insertion
        cursor.close()
        return True
    except Exception as err:
        print("Error while adding to cart:", err)
        return False
```

7. get_all_from_cart

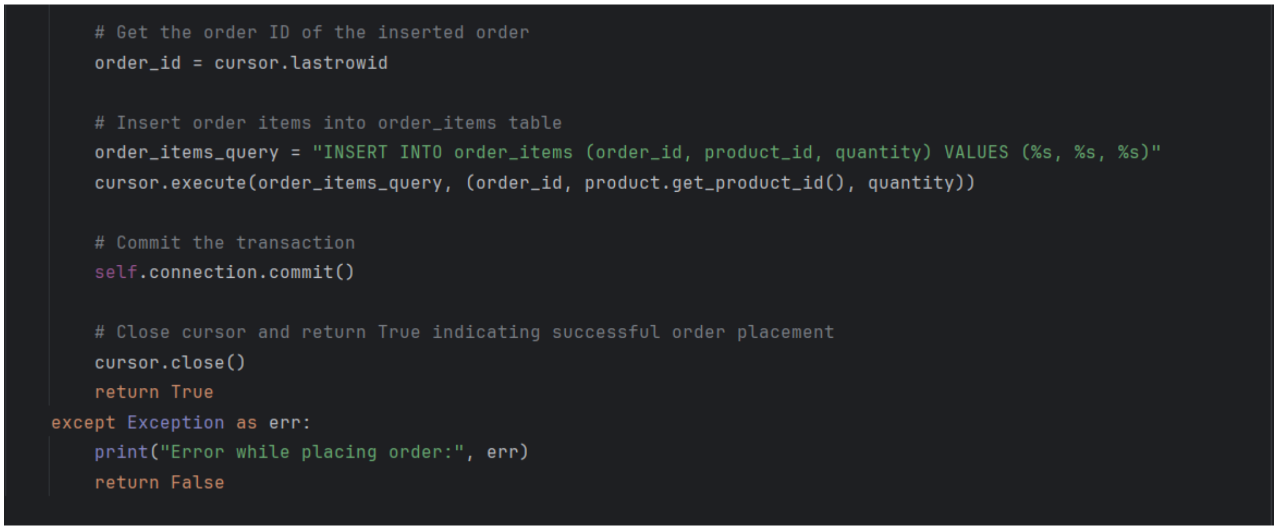


```
def get_all_from_cart(self, customer: Customer) -> List[Product]:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")
        print("enter the customer id : ")
        customerID = input("customer id :")
        customer = Customer(customer_id=customerID)

        # Fetch products in the cart for the given customer
        query = "SELECT p.product_id, p.product_name, p.price, p.description, p.stockQuantity " \
                "FROM cart c " \
                "JOIN products p ON c.product_id = p.product_id " \
                "WHERE c.customer_id = %s"
        cursor.execute(query, (customer.get_customer_id(),))

        cart_products = cursor.fetchall()

        products = []
        for row in cart_products:
            product_id, name, price, description, stock_quantity = row
            product = Product(product_id=product_id, name=name, price=price, description=description,
                              stock_quantity=stock_quantity)
            products.append(product)

        cursor.close()
        return products
    except Exception as err:
        print("Error while fetching products from cart:", err)
        return []
```

8. place_order

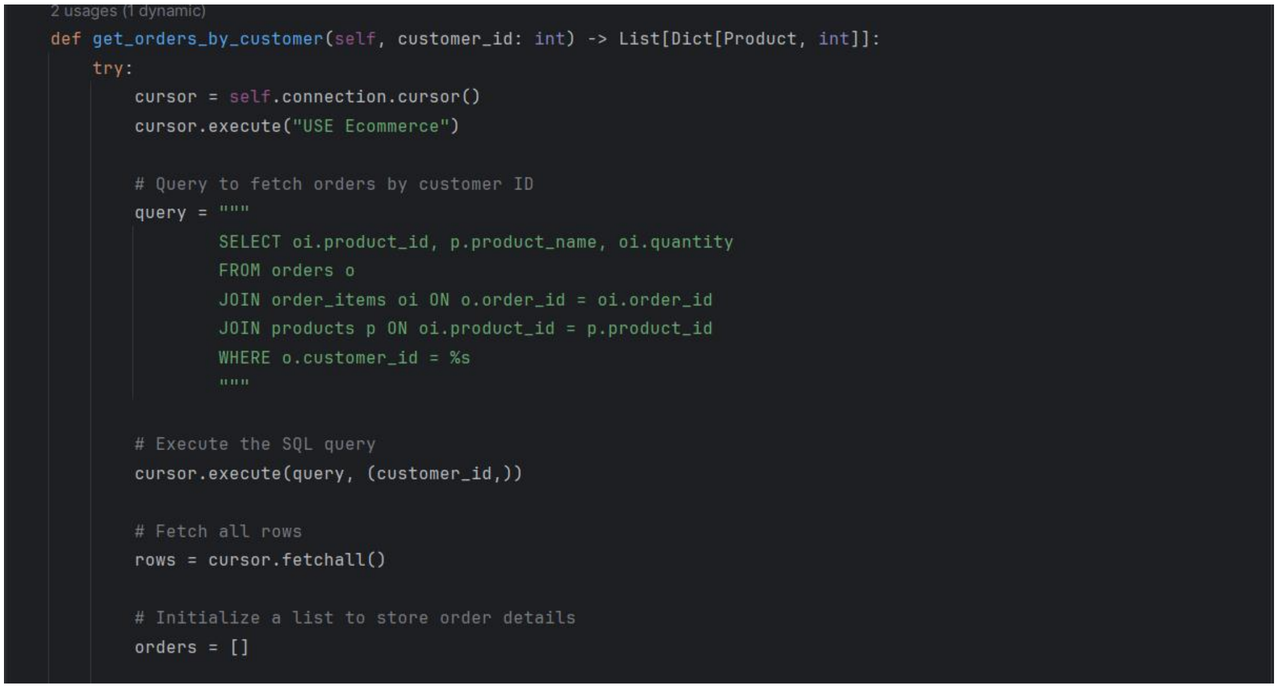


```
2 usages (1 dynamic)
def place_order(self, customer: Customer, products_quantities: List[Dict[Product, int]]) -> bool:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        # Prompt the user to enter shipping address
        shipping_address = input("Enter shipping address: ")

        # Get the customer ID
        customerId = input("enter customer id : ")
        productID = input("enter product id :")
        quantity = int(input("enter quantity : "))


        customer = Customer(customer_id = customerId)
        product = Product(product_id=productID)
        val = cursor.execute("select price from products where Product_id = (%s) ",productID)
        total = val * quantity


        # Insert order into orders table
        order_query = ("INSERT INTO orders (customer_id, order_date, total_price, shipping_address)"
                       " VALUES (%s, NOW(), %s , %s)")
        order_values = (customer.get_customer_id(), total , shipping_address)
        cursor.execute(order_query, order_values )
```

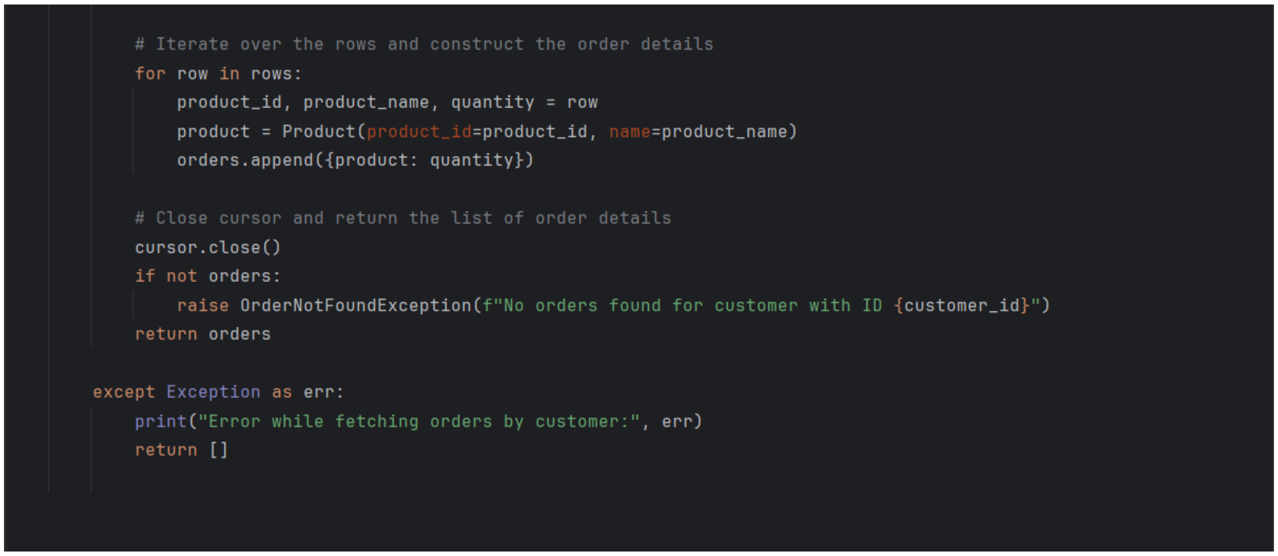


```
        # Get the order ID of the inserted order
        order_id = cursor.lastrowid

        # Insert order items into order_items table
        order_items_query = "INSERT INTO order_items (order_id, product_id, quantity) VALUES (%s, %s, %s)"
        cursor.execute(order_items_query, (order_id, product.get_product_id(), quantity))

        # Commit the transaction
        self.connection.commit()

        # Close cursor and return True indicating successful order placement
        cursor.close()
        return True
    except Exception as err:
        print("Error while placing order:", err)
        return False
```

9. get_orders_by_customer



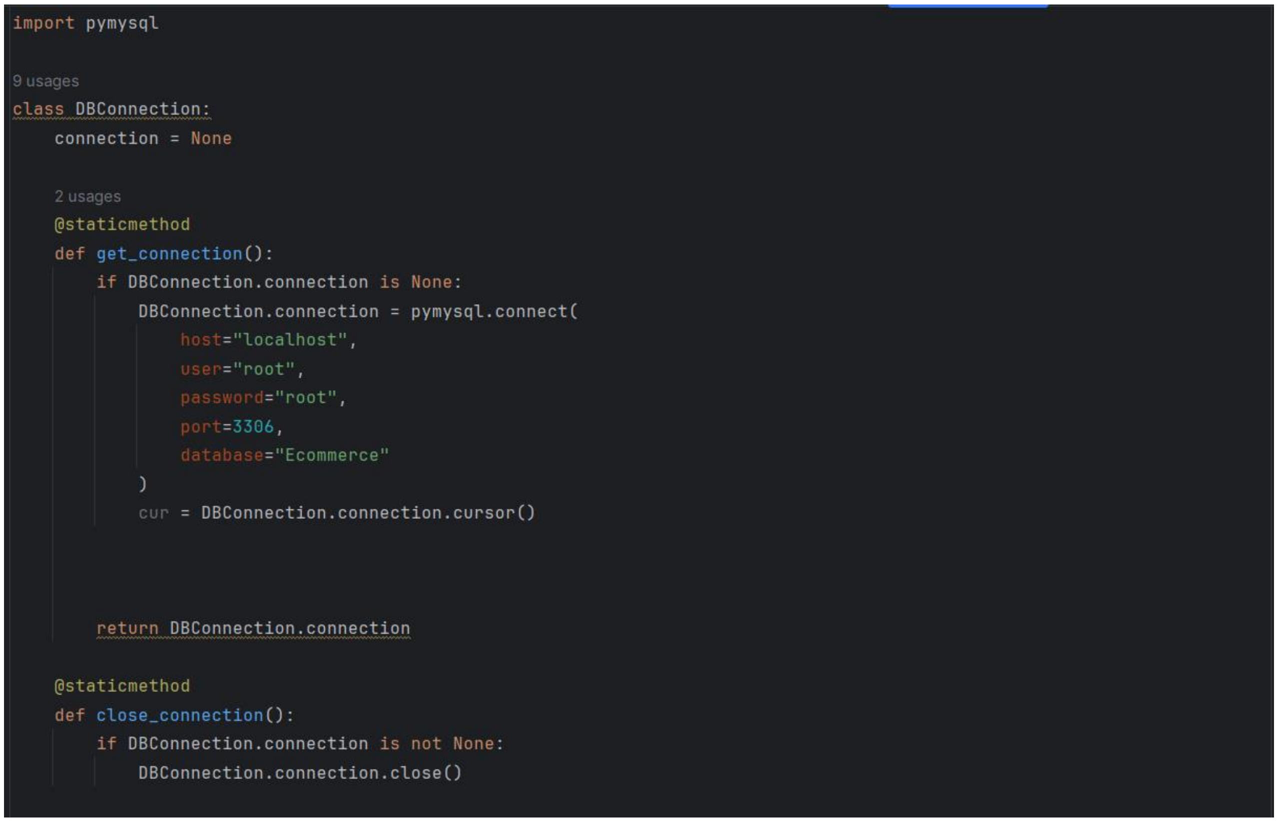
```
2 usages (1 dynamic)
def get_orders_by_customer(self, customer_id: int) -> List[Dict[Product, int]]:
    try:
        cursor = self.connection.cursor()
        cursor.execute("USE Ecommerce")

        # Query to fetch orders by customer ID
        query = """
                SELECT oi.product_id, p.product_name, oi.quantity
                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 = %s
                """

        # Execute the SQL query
        cursor.execute(query, (customer_id,))

        # Fetch all rows
        rows = cursor.fetchall()

        # Initialize a list to store order details
        orders = []
```



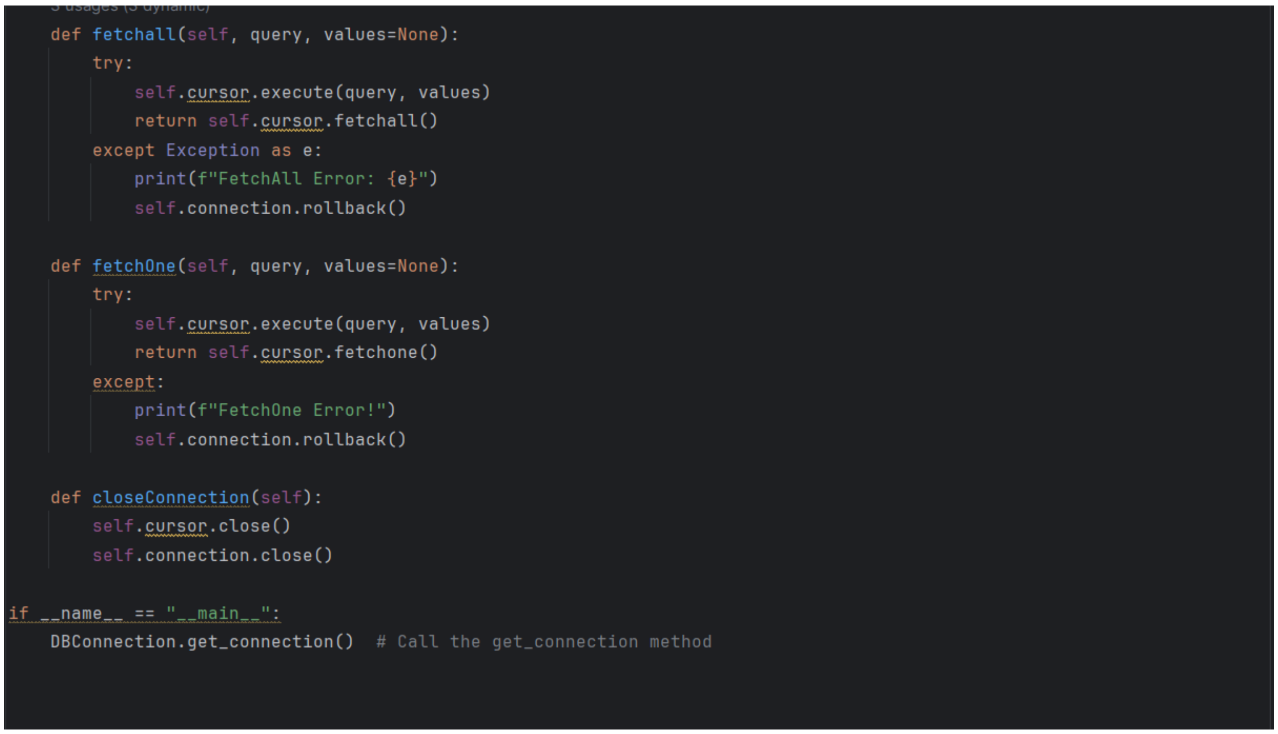
```
        # Iterate over the rows and construct the order details
        for row in rows:
            product_id, product_name, quantity = row
            product = Product(product_id=product_id, name=product_name)
            orders.append({product: quantity})

        # Close cursor and return the list of order details
        cursor.close()
        if not orders:
            raise OrderNotFoundException(f"No orders found for customer with ID {customer_id}")
        return orders

    except Exception as err:
        print("Error while fetching orders by customer:", err)
        return []
```

Connection to database :

- To run the implemented code we need to give a connection to the implementation class so that the class can just call the connection and use it
- For that its necessary to create a util package where all the connection is made



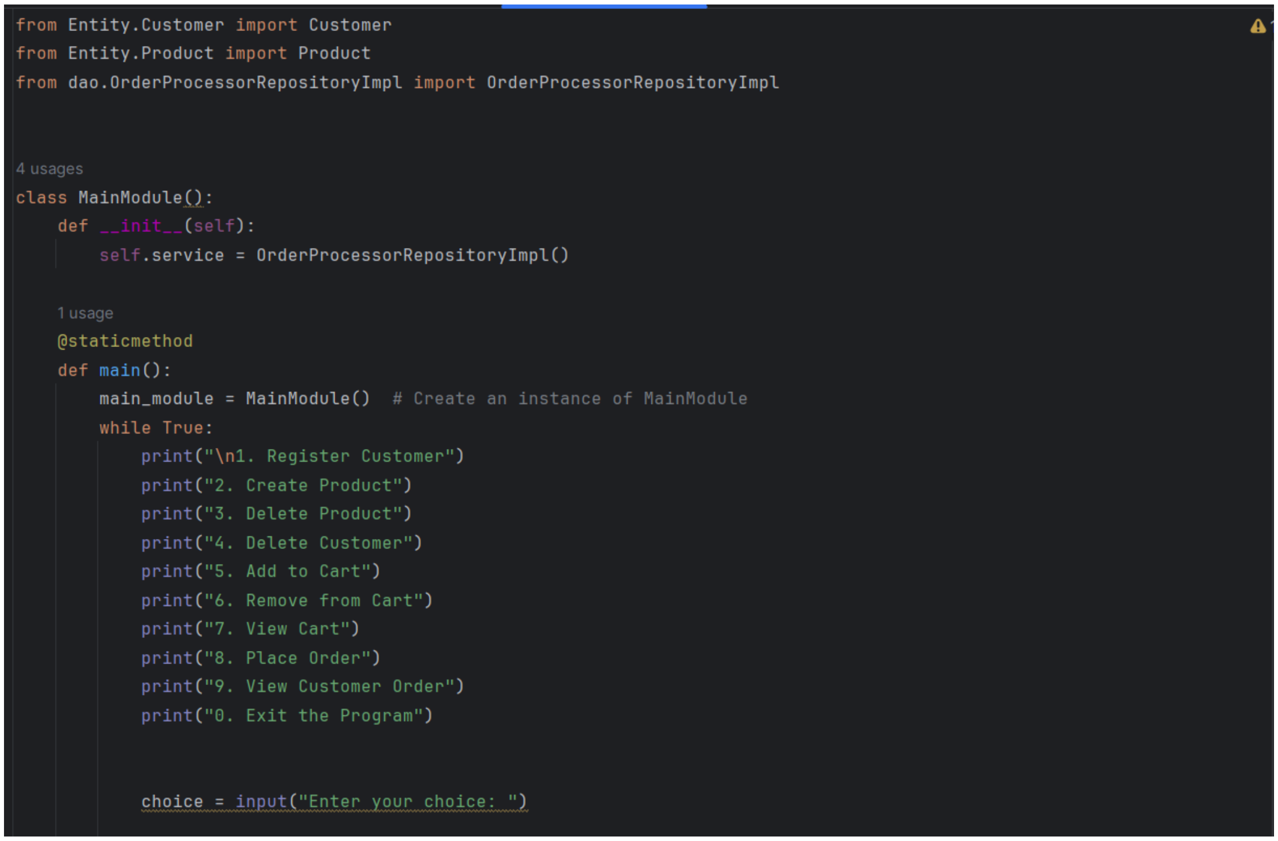
```
import pymysql

9 usages
class DBConnection:
    connection = None

    2 usages
    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            DBConnection.connection = pymysql.connect(
                host="localhost",
                user="root",
                password="root",
                port=3306,
                database="Ecommerce"
            )
            cur = DBConnection.connection.cursor()



        return DBConnection.connection

    @staticmethod
    def close_connection():
        if DBConnection.connection is not None:
            DBConnection.connection.close()
```



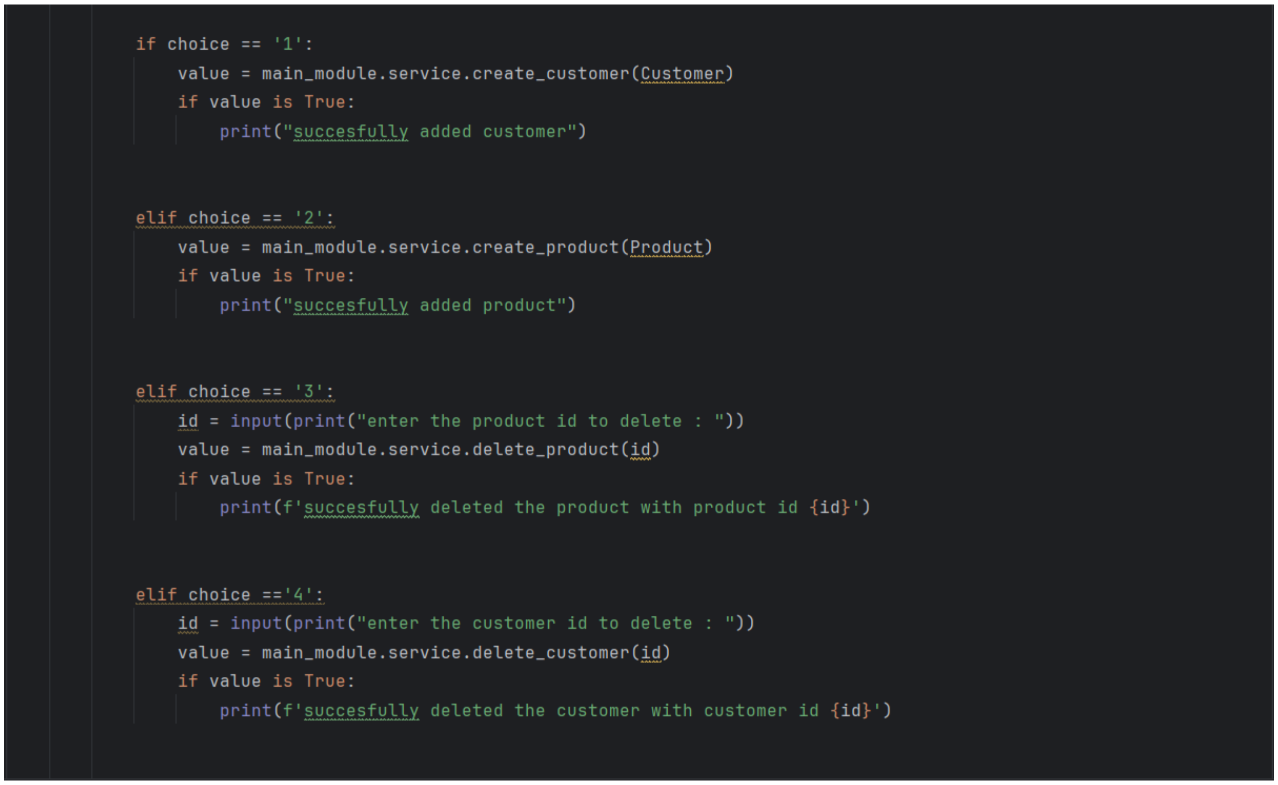
```
    def fetchall(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchall()
        except Exception as e:
            print(f"FetchAll Error: {e}")
            self.connection.rollback()

    def fetchOne(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchone()
        except:
            print(f"FetchOne Error!")
            self.connection.rollback()

    def closeConnection(self):
        self.cursor.close()
        self.connection.close()

if __name__ == "__main__":
    DBConnection.get_connection()  # Call the get_connection method
```

<u>Designing the Main method :</u>

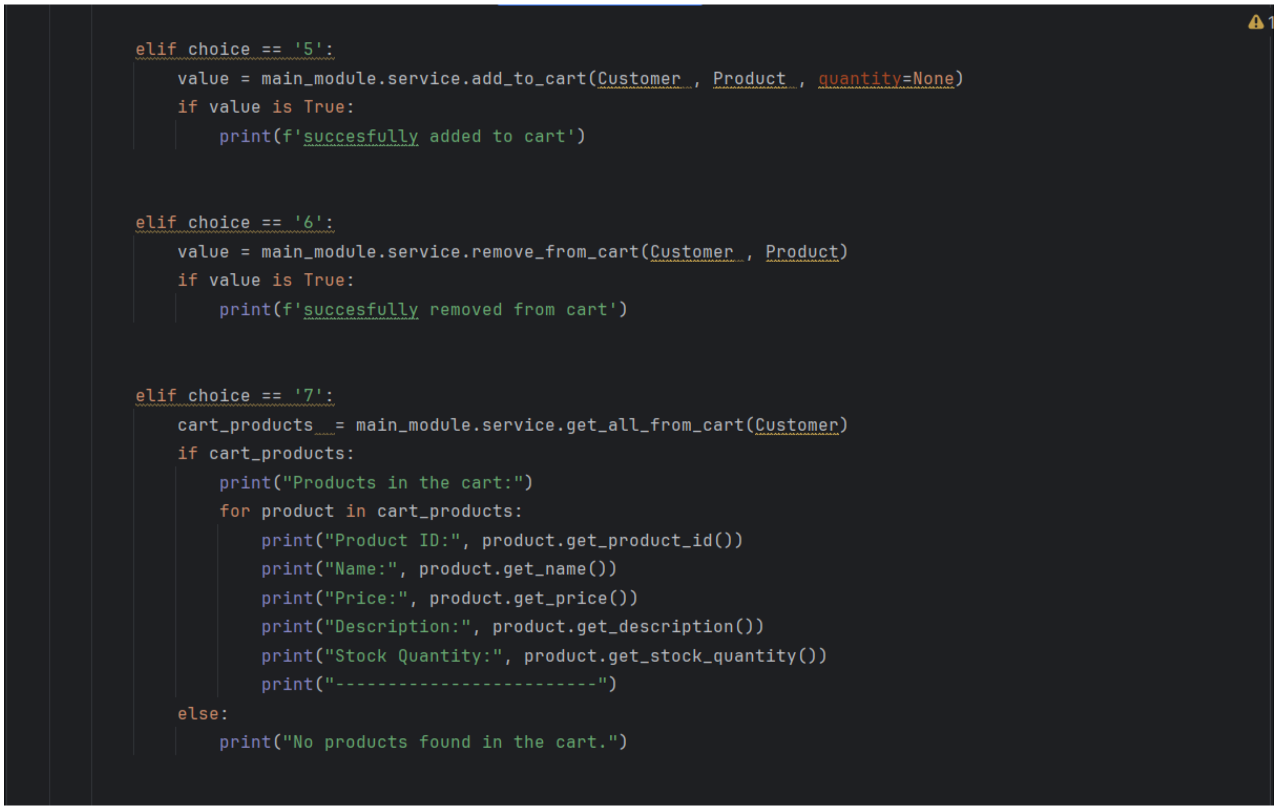
- The main method is the actual app that will be running infront of the user , the user only able to see what is there in the main
- All the rest processing is been done in the backend park as mentioned above



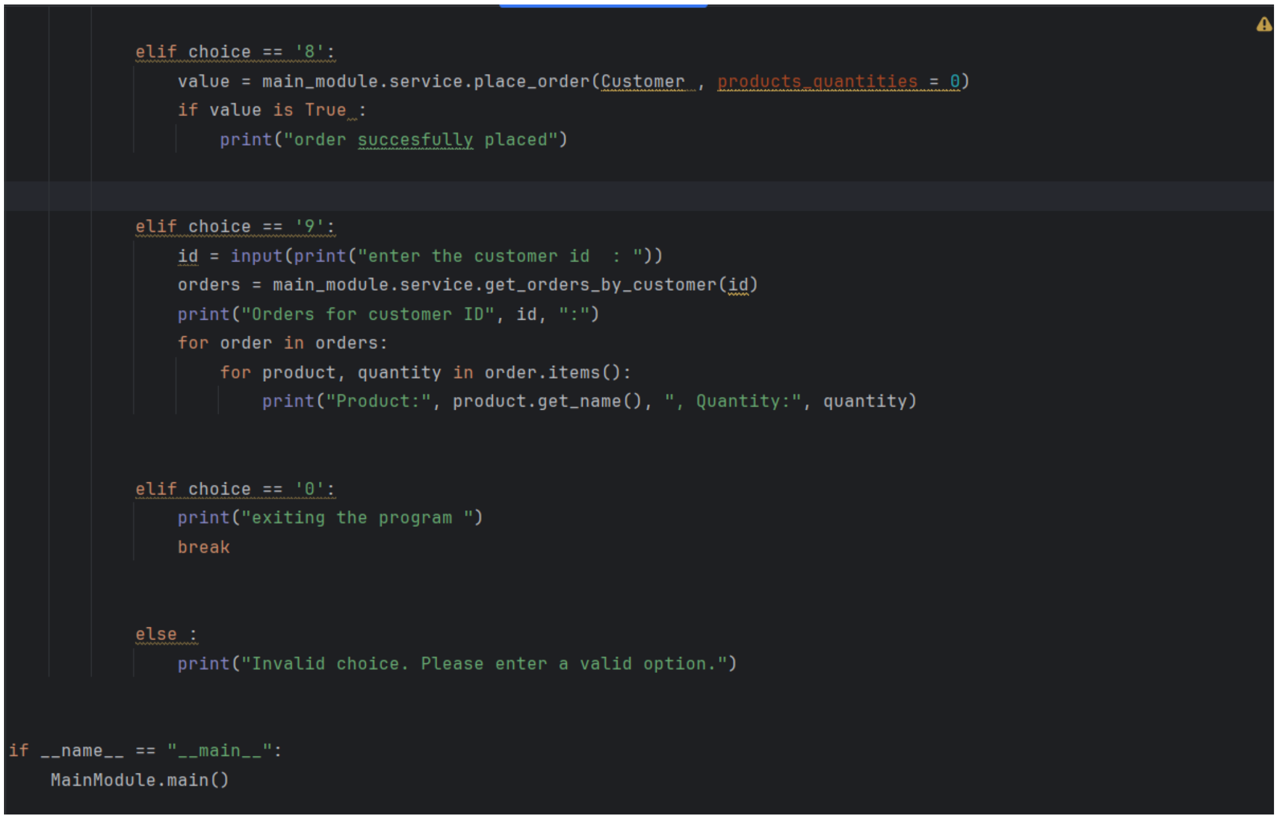
```
from Entity.Customer import Customer
from Entity.Product import Product
from dao.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl


4 usages
class MainModule():
    def __init__(self):
        self.service = OrderProcessorRepositoryImpl()

    1 usage
    @staticmethod
    def main():
        main_module = MainModule()  # Create an instance of MainModule
        while True:
            print("\n1. Register Customer")
            print("2. Create Product")
            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. View Customer Order")
            print("0. Exit the Program")


            choice = input("Enter your choice: ")
```



```
            if choice == '1':
                value = main_module.service.create_customer(Customer)
                if value is True:
                    print("succesfully added customer")


            elif choice == '2':
                value = main_module.service.create_product(Product)
                if value is True:
                    print("succesfully added product")


            elif choice == '3':
                id = input(print("enter the product id to delete : "))
                value = main_module.service.delete_product(id)
                if value is True:
                    print(f'succesfully deleted the product with product id {id}')


            elif choice =='4':
                id = input(print("enter the customer id to delete : "))
                value = main_module.service.delete_customer(id)
                if value is True:
                    print(f'succesfully deleted the customer with customer id {id}')
```



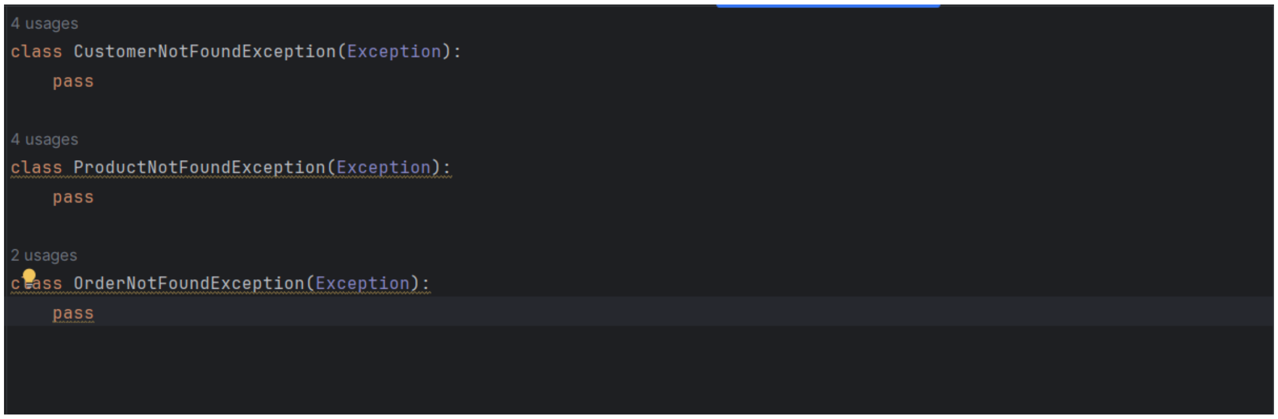
```
            elif choice == '5':
                value = main_module.service.add_to_cart(Customer , Product , quantity=None)
                if value is True:
                    print(f'succesfully added to cart')


            elif choice == '6':
                value = main_module.service.remove_from_cart(Customer , Product)
                if value is True:
                    print(f'succesfully removed from cart')


            elif choice == '7':
                cart_products  = main_module.service.get_all_from_cart(Customer)
                if cart_products:
                    print("Products in the cart:")
                    for product in cart_products:
                        print("Product ID:", product.get_product_id())
                        print("Name:", product.get_name())
                        print("Price:", product.get_price())
                        print("Description:", product.get_description())
                        print("Stock Quantity:", product.get_stock_quantity())
                        print("-------------------------")
                else:
                    print("No products found in the cart.")
```



```
            elif choice == '8':
                value = main_module.service.place_order(Customer , products_quantities = 0)
                if value is True :
                    print("order succesfully placed")


            elif choice == '9':
                id = input(print("enter the customer id  : "))
                orders = main_module.service.get_orders_by_customer(id)
                print("Orders for customer ID", id, ":")
                for order in orders:
                    for product, quantity in order.items():
                        print("Product:", product.get_name(), ", Quantity:", quantity)


            elif choice == '0':
                print("exiting the program ")
                break


            else :
                print("Invalid choice. Please enter a valid option.")


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

Creating Exceptions for the program :

User defined exceptions are made to understand the problems easily

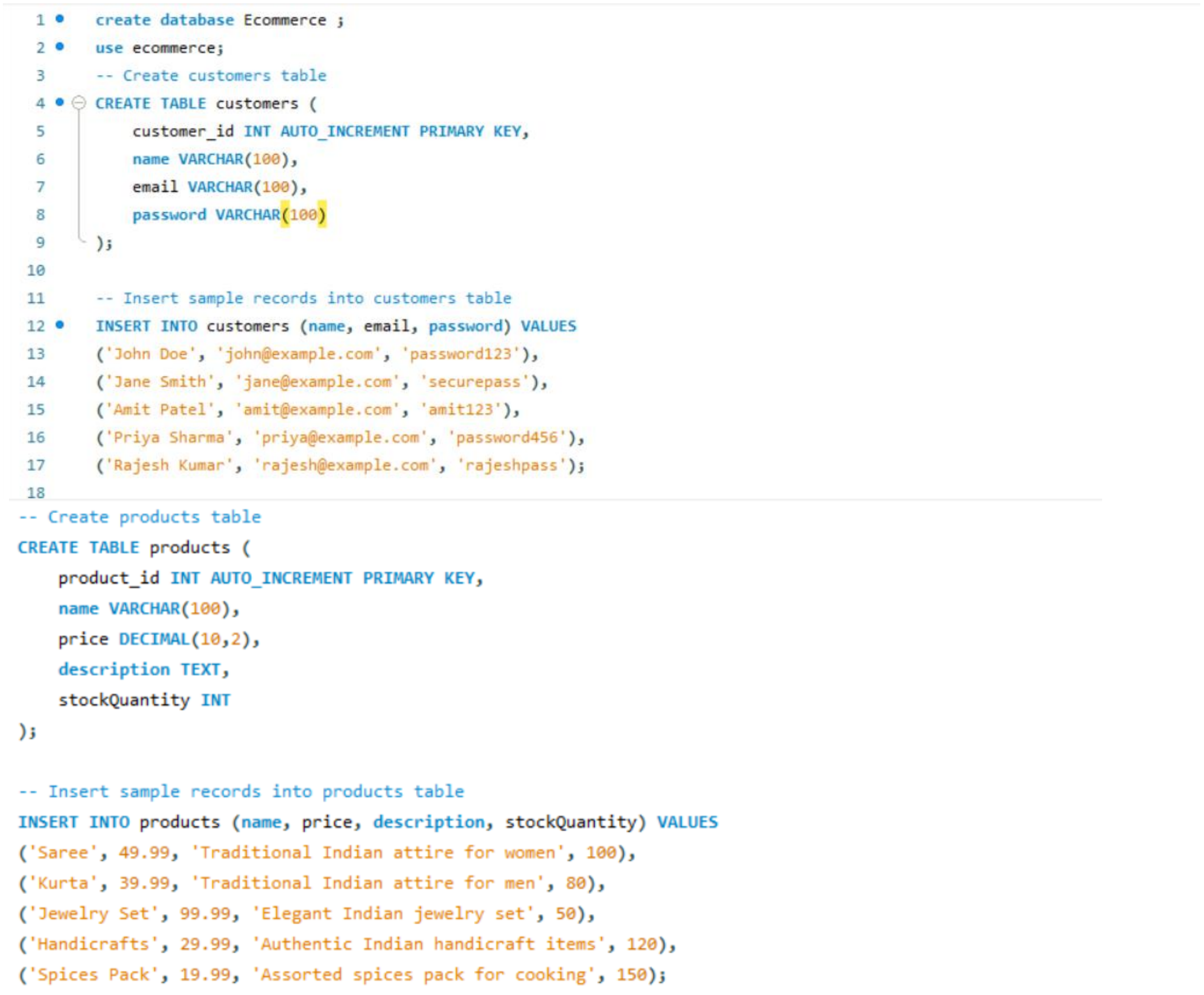


```
4 usages
class CustomerNotFoundException(Exception):
    pass

4 usages
class ProductNotFoundException(Exception):
    pass

2 usages
class OrderNotFoundException(Exception):
    pass
```

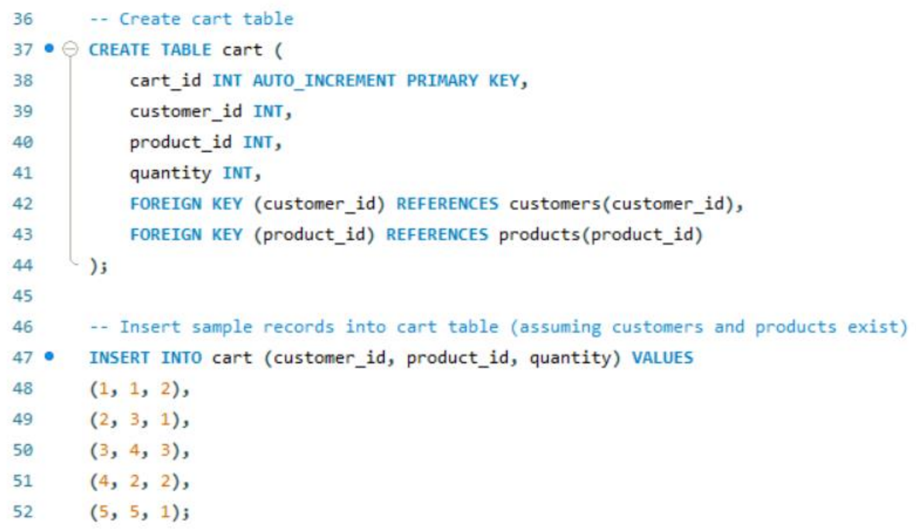
Database Creation :

The database is made to store all the data from the program.

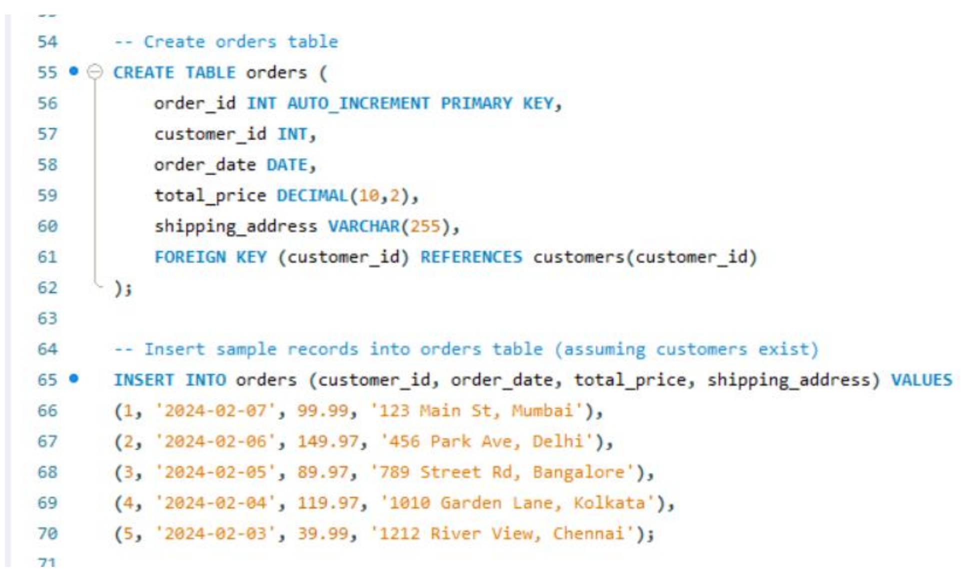
Also some sample records are filled to make database more readable



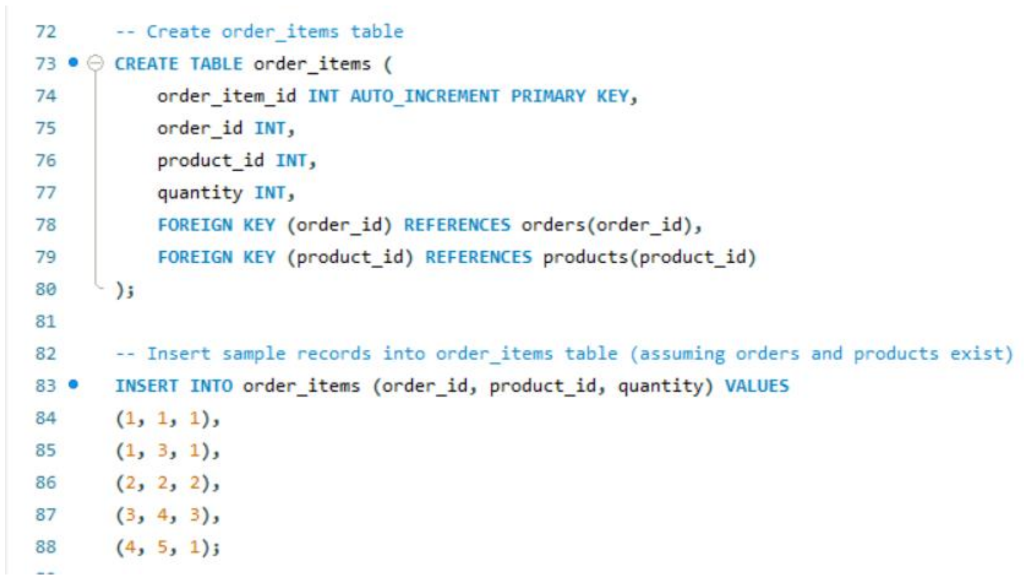
```
create database Ecommerce ;
use ecommerce;
-- Create customers table
CREATE TABLE customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    password VARCHAR(100)
);

-- Insert sample records into customers table
INSERT INTO customers (name, email, password) VALUES
('John Doe', 'john@example.com', 'password123'),
('Jane Smith', 'jane@example.com', 'securepass'),
('Amit Patel', 'amit@example.com', 'amit123'),
('Priya Sharma', 'priya@example.com', 'password456'),
('Rajesh Kumar', 'rajesh@example.com', 'rajeshpass');

-- Create products table
CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL(10,2),
    description TEXT,
    stockQuantity INT
);

-- Insert sample records into products table
INSERT INTO products (name, price, description, stockQuantity) VALUES
('Saree', 49.99, 'Traditional Indian attire for women', 100),
('Kurta', 39.99, 'Traditional Indian attire for men', 80),
('Jewelry Set', 99.99, 'Elegant Indian jewelry set', 50),
('Handicrafts', 29.99, 'Authentic Indian handicraft items', 120),
('Spices Pack', 19.99, 'Assorted spices pack for cooking', 150);
```



```sql
-- Create cart table
CREATE TABLE cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    quantity INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- Insert sample records into cart table (assuming customers and products exist)
INSERT INTO cart (customer_id, product_id, quantity) VALUES
(1, 1, 2),
(2, 3, 1),
(3, 4, 3),
(4, 2, 2),
(5, 5, 1);
```

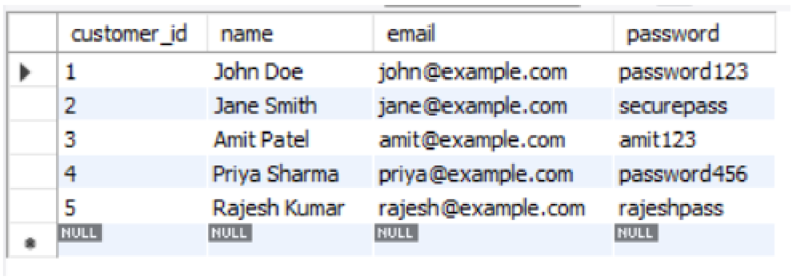


```sql
-- Create orders table
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_price DECIMAL(10,2),
    shipping_address VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- Insert sample records into orders table (assuming customers exist)
INSERT INTO orders (customer_id, order_date, total_price, shipping_address) VALUES
(1, '2024-02-07', 99.99, '123 Main St, Mumbai'),
(2, '2024-02-06', 149.97, '456 Park Ave, Delhi'),
(3, '2024-02-05', 89.97, '789 Street Rd, Bangalore'),
(4, '2024-02-04', 119.97, '1010 Garden Lane, Kolkata'),
(5, '2024-02-03', 39.99, '1212 River View, Chennai');
```

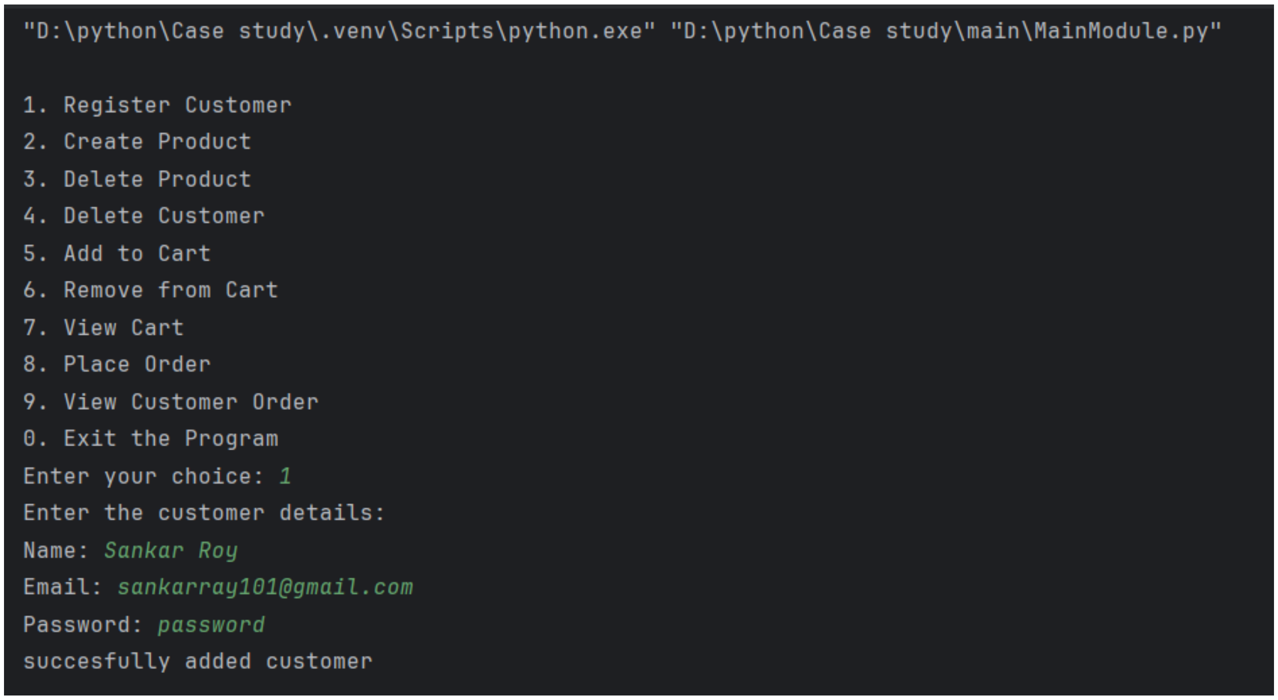


```sql
-- Create order_items table
CREATE TABLE order_items (
    order_item_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- Insert sample records into order_items table (assuming orders and products exist)
INSERT INTO order_items (order_id, product_id, quantity) VALUES
(1, 1, 1),
(1, 3, 1),
(2, 2, 2),
(3, 4, 3),
(4, 5, 1);
```

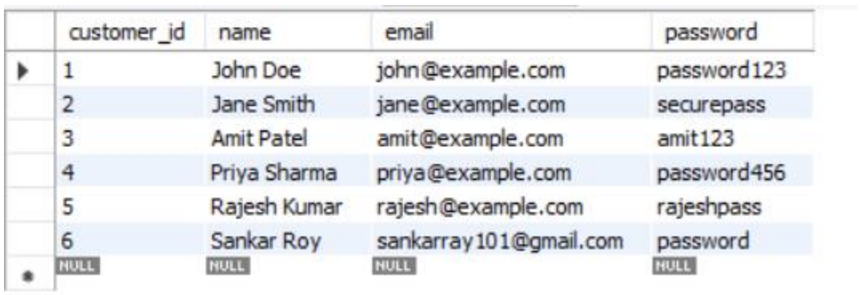
OUTPUT- 1



	customer_id	name	email	password
▶	1	John Doe	john@example.com	password123
	2	Jane Smith	jane@example.com	securepass
	3	Amit Patel	amit@example.com	amit123
	4	Priya Sharma	priya@example.com	password456
	5	Rajesh Kumar	rajesh@example.com	rajeshpass
*	NULL	NULL	NULL	NULL



```
"D:\python\Case study\.venv\Scripts\python.exe" "D:\python\Case study\main\MainModule.py"

1. Register Customer
2. Create Product
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. View Customer Order
0. Exit the Program
Enter your choice: 1
Enter the customer details:
Name: Sankar Roy
Email: sankarray101@gmail.com
Password: password
succesfully added customer
```

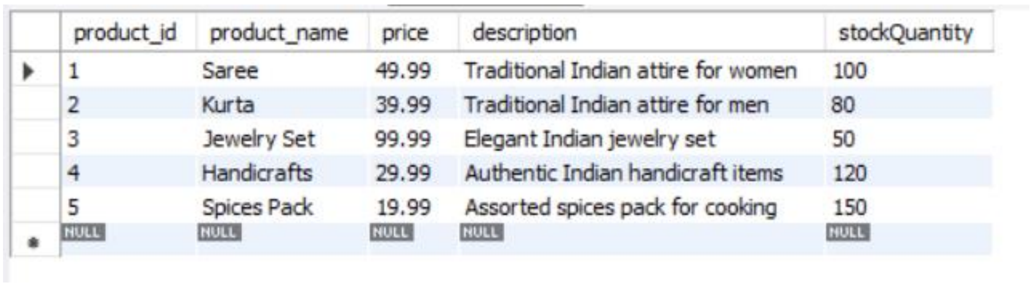


	customer_id	name	email	password
▶	1	John Doe	john@example.com	password123
	2	Jane Smith	jane@example.com	securepass
	3	Amit Patel	amit@example.com	amit123
	4	Priya Sharma	priya@example.com	password456
	5	Rajesh Kumar	rajesh@example.com	rajeshpass
	6	Sankar Roy	sankarray101@gmail.com	password
*	NULL	NULL	NULL	NULL

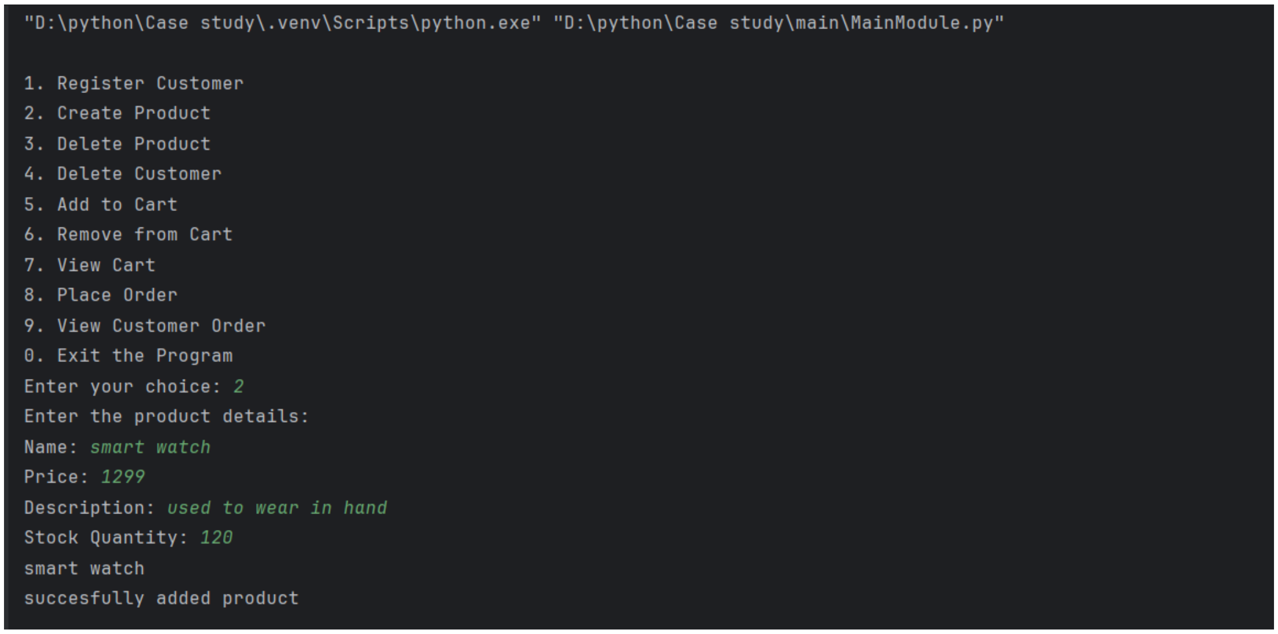
Here the new entry of the customer has been reflected in the database also

© Hexaware Technologies Limited. All rights

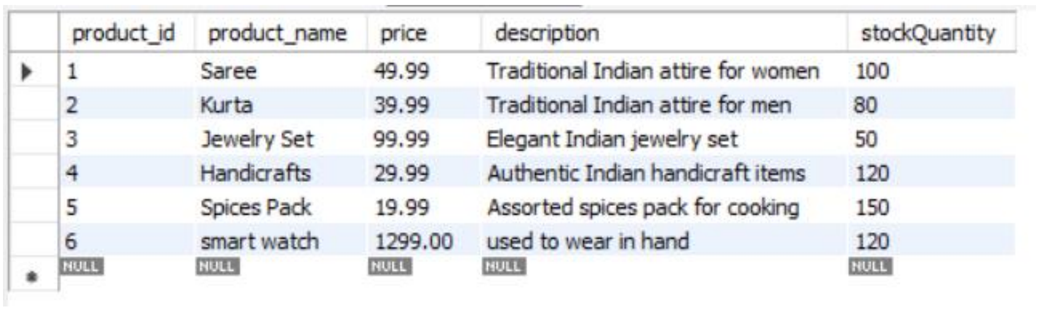
OUTPUT- 2



	product_id	product_name	price	description	stockQuantity
▶	1	Saree	49.99	Traditional Indian attire for women	100
	2	Kurta	39.99	Traditional Indian attire for men	80
	3	Jewelry Set	99.99	Elegant Indian jewelry set	50
	4	Handicrafts	29.99	Authentic Indian handicraft items	120
	5	Spices Pack	19.99	Assorted spices pack for cooking	150
*	NULL	NULL	NULL	NULL	NULL

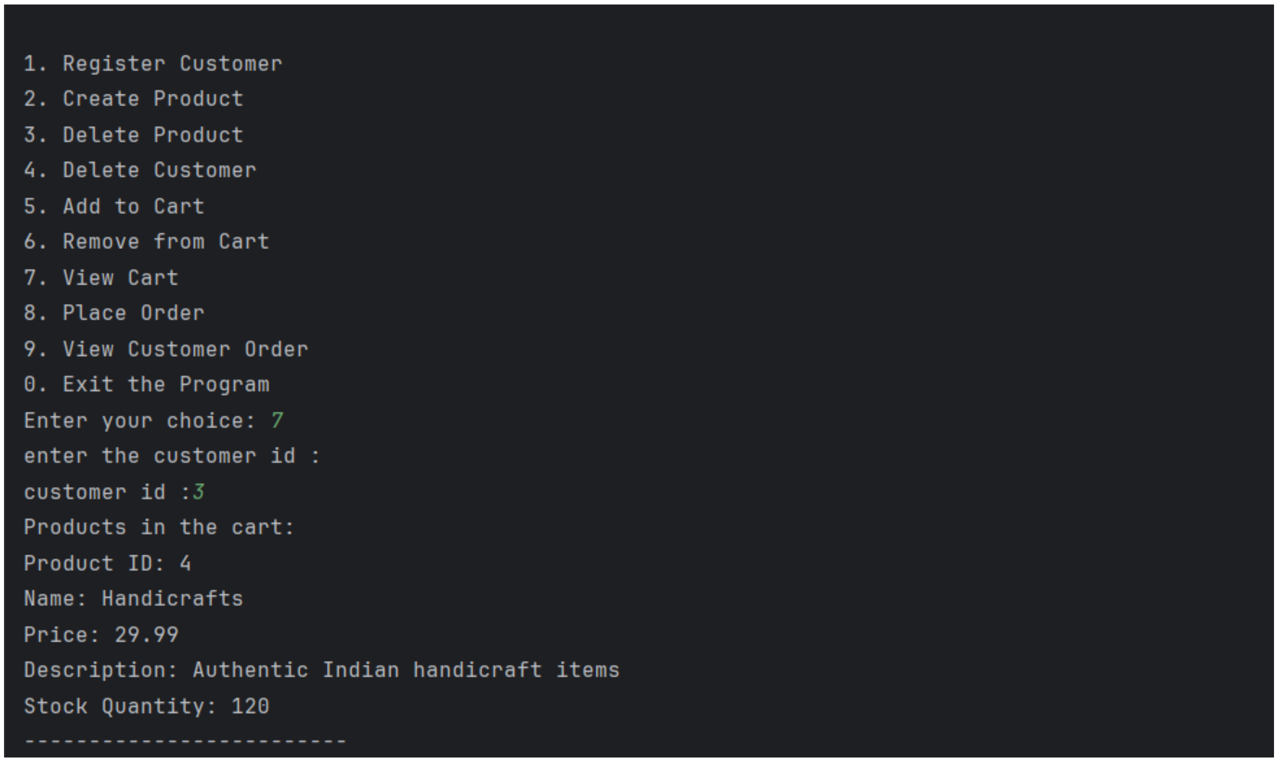


```
"D:\python\Case study\.venv\Scripts\python.exe" "D:\python\Case study\main\MainModule.py"

1. Register Customer
2. Create Product
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. View Customer Order
0. Exit the Program
Enter your choice: 2
Enter the product details:
Name: smart watch
Price: 1299
Description: used to wear in hand
Stock Quantity: 120
smart watch
succesfully added product
```

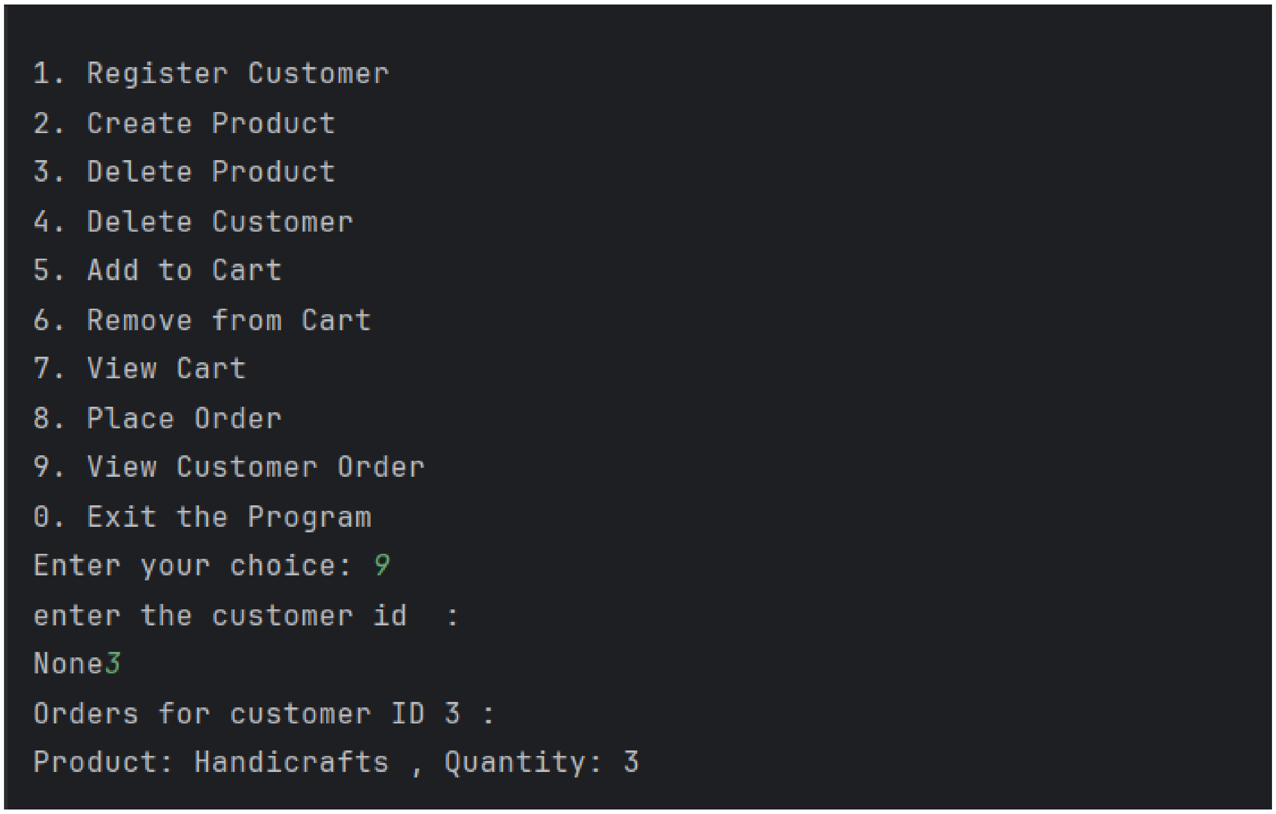


	product_id	product_name	price	description	stockQuantity
▶	1	Saree	49.99	Traditional Indian attire for women	100
	2	Kurta	39.99	Traditional Indian attire for men	80
	3	Jewelry Set	99.99	Elegant Indian jewelry set	50
	4	Handicrafts	29.99	Authentic Indian handicraft items	120
	5	Spices Pack	19.99	Assorted spices pack for cooking	150
	6	smart watch	1299.00	used to wear in hand	120
*	NULL	NULL	NULL	NULL	NULL

OUTPUT- 3

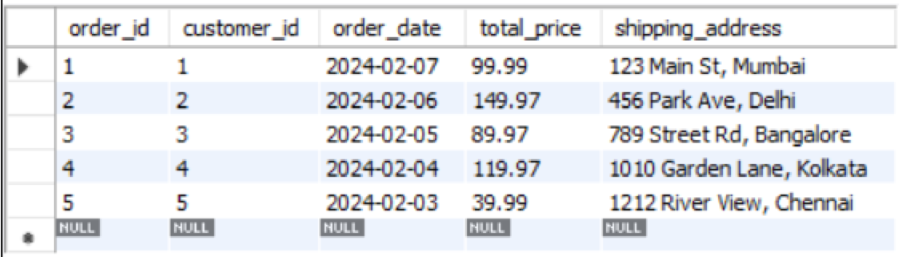


```
1. Register Customer
2. Create Product
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. View Customer Order
0. Exit the Program
Enter your choice: 7
enter the customer id :
customer id :3
Products in the cart:
Product ID: 4
Name: Handicrafts
Price: 29.99
Description: Authentic Indian handicraft items
Stock Quantity: 120
-------------------------
```

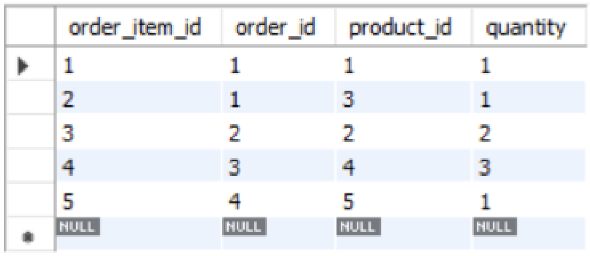


```
1. Register Customer
2. Create Product
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. View Customer Order
0. Exit the Program
Enter your choice: 9
enter the customer id  :
None3
Orders for customer ID 3 :
Product: Handicrafts , Quantity: 3
```

OUTPUT- 4

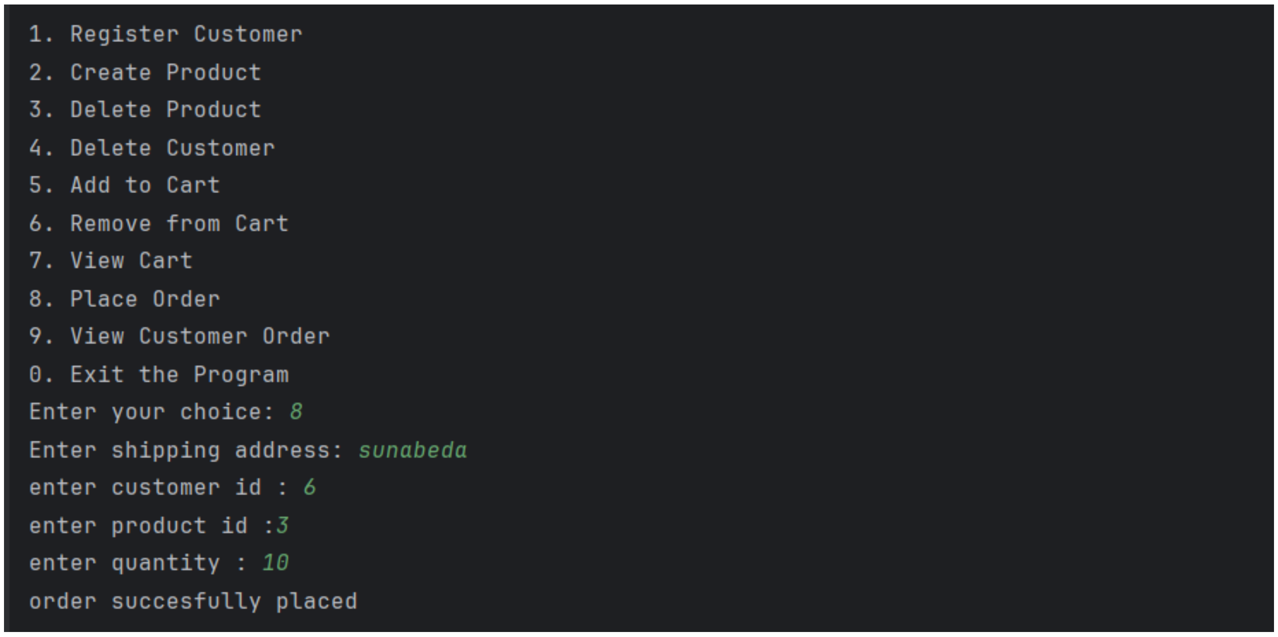


order_id	customer_id	order_date	total_price	shipping_address
1	1	2024-02-07	99.99	123 Main St, Mumbai
2	2	2024-02-06	149.97	456 Park Ave, Delhi
3	3	2024-02-05	89.97	789 Street Rd, Bangalore
4	4	2024-02-04	119.97	1010 Garden Lane, Kolkata
5	5	2024-02-03	39.99	1212 River View, Chennai
NULL	NULL	NULL	NULL	NULL

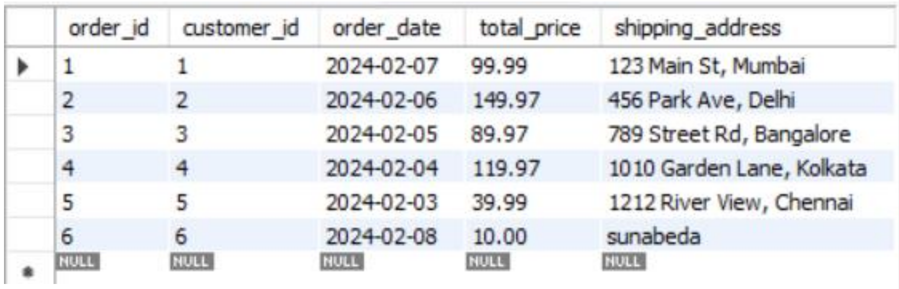


order_item_id	order_id	product_id	quantity
1	1	1	1
2	1	3	1
3	2	2	2
4	3	4	3
5	4	5	1
NULL	NULL	NULL	NULL

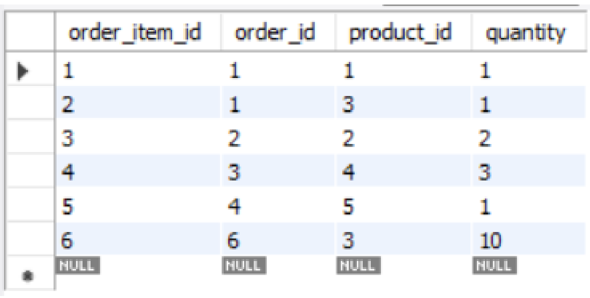
The order and order_item table before placing order



```
1. Register Customer
2. Create Product
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. View Customer Order
0. Exit the Program
Enter your choice: 8
Enter shipping address: sunabeda
enter customer id : 6
enter product id :3
enter quantity : 10
order succesfully placed
```



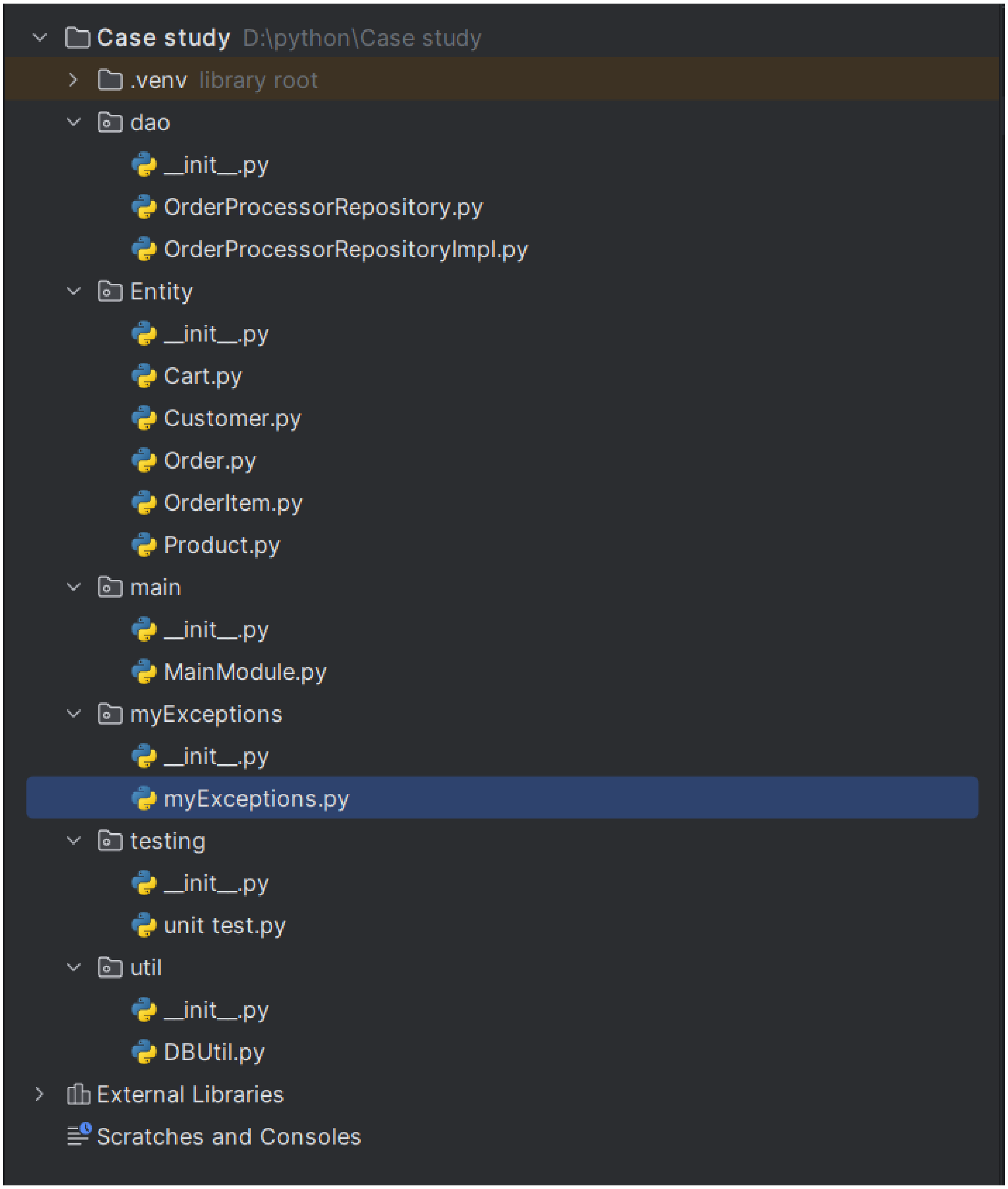
order_id	customer_id	order_date	total_price	shipping_address
1	1	2024-02-07	99.99	123 Main St, Mumbai
2	2	2024-02-06	149.97	456 Park Ave, Delhi
3	3	2024-02-05	89.97	789 Street Rd, Bangalore
4	4	2024-02-04	119.97	1010 Garden Lane, Kolkata
5	5	2024-02-03	39.99	1212 River View, Chennai
6	6	2024-02-08	10.00	sunabeda
NULL	NULL	NULL	NULL	NULL



order_item_id	order_id	product_id	quantity
1	1	1	1
2	1	3	1
3	2	2	2
4	3	4	3
5	4	5	1
6	6	3	10
NULL	NULL	NULL	NULL

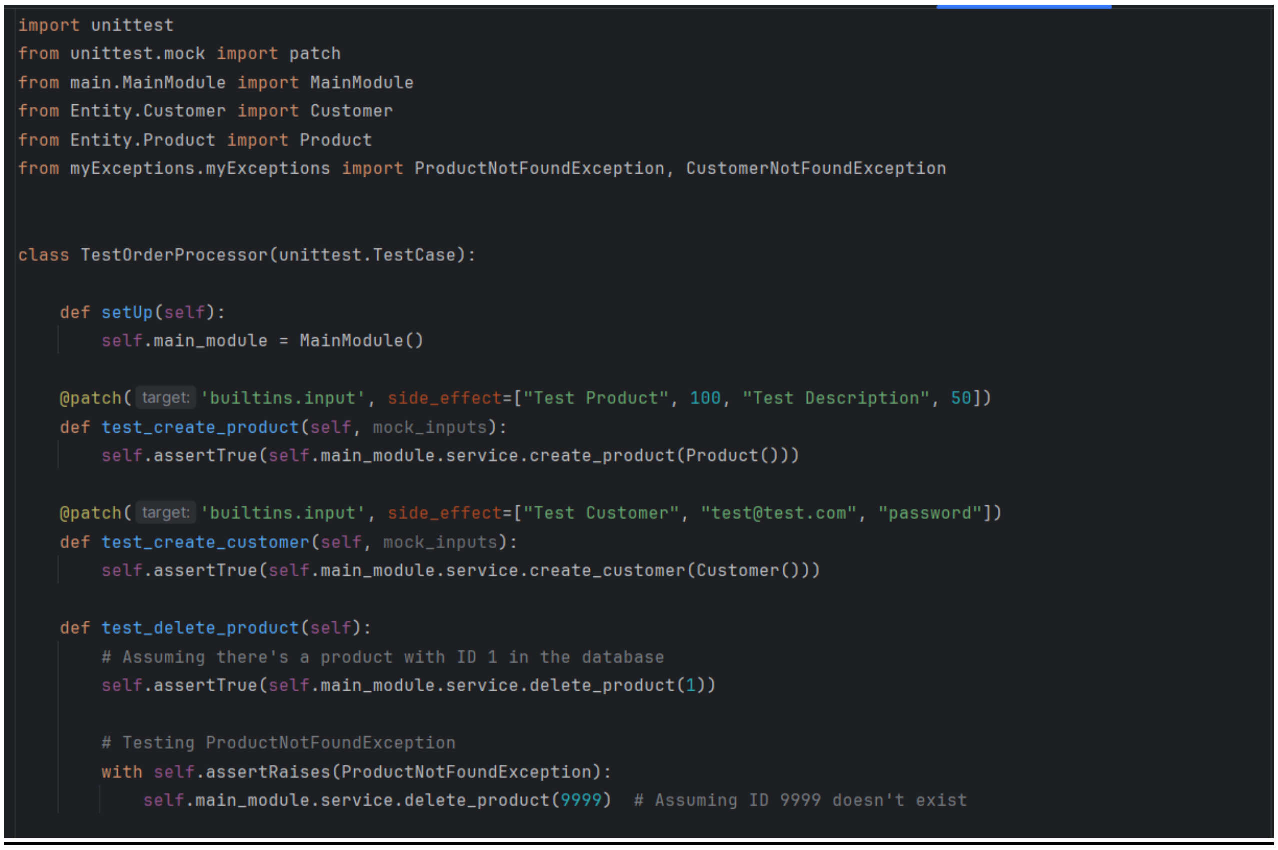
The order and order_item table after placing order

The Final folder after creating the project

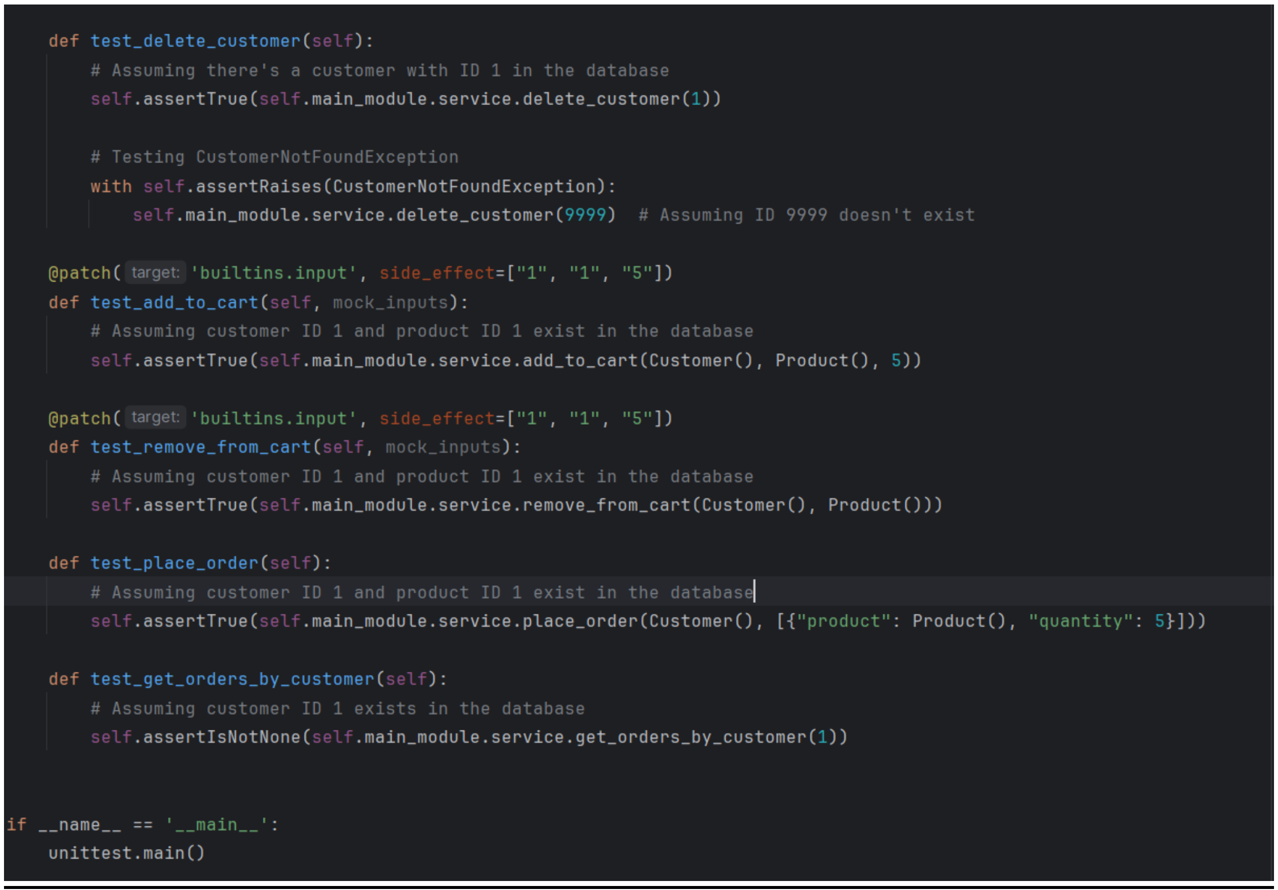


```
Case study  D:\python\Case study
    .venv  library root
    dao
        __init__.py
        OrderProcessorRepository.py
        OrderProcessorRepositoryImpl.py
    Entity
        __init__.py
        Cart.py
        Customer.py
        Order.py
        OrderItem.py
        Product.py
    main
        __init__.py
        MainModule.py
    myExceptions
        __init__.py
        myExceptions.py
    testing
        __init__.py
        unit test.py
    util
        __init__.py
        DBUtil.py
External Libraries
Scratches and Consoles
```

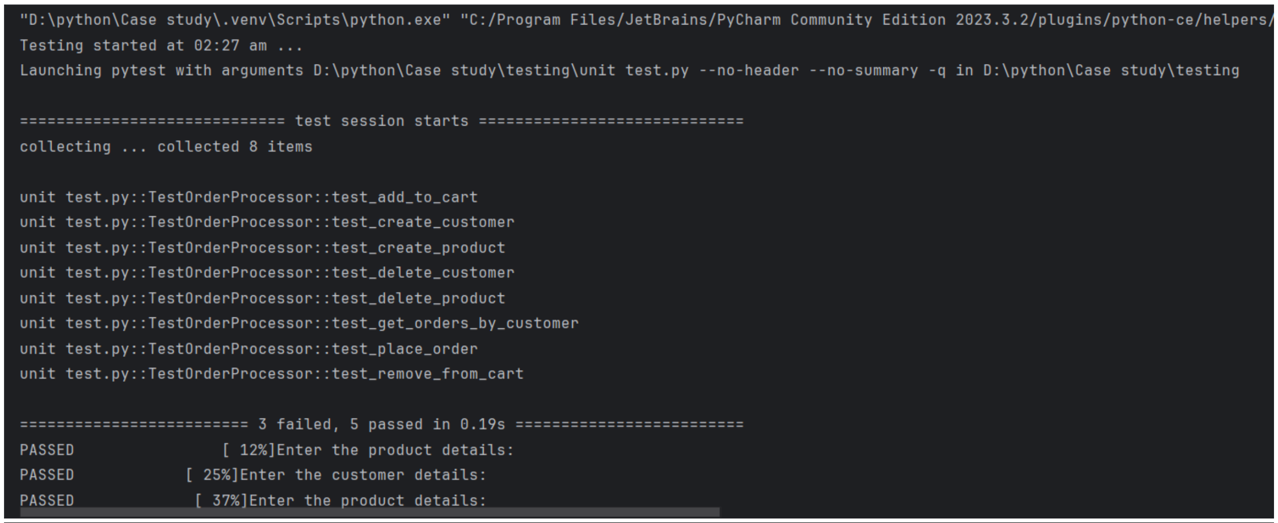
Testing and result



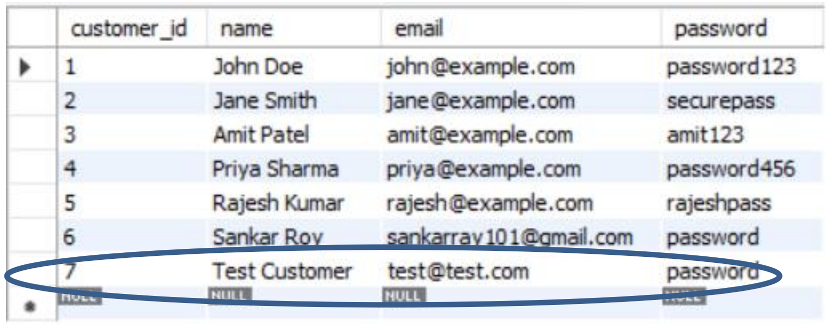
```python
import unittest
from unittest.mock import patch
from main.MainModule import MainModule
from Entity.Customer import Customer
from Entity.Product import Product
from myExceptions.myExceptions import ProductNotFoundException, CustomerNotFoundException


class TestOrderProcessor(unittest.TestCase):

    def setUp(self):
        self.main_module = MainModule()

    @patch( target: 'builtins.input', side_effect=["Test Product", 100, "Test Description", 50])
    def test_create_product(self, mock_inputs):
        self.assertTrue(self.main_module.service.create_product(Product()))

    @patch( target: 'builtins.input', side_effect=["Test Customer", "test@test.com", "password"])
    def test_create_customer(self, mock_inputs):
        self.assertTrue(self.main_module.service.create_customer(Customer()))

    def test_delete_product(self):
        # Assuming there's a product with ID 1 in the database
        self.assertTrue(self.main_module.service.delete_product(1))

        # Testing ProductNotFoundException
        with self.assertRaises(ProductNotFoundException):
            self.main_module.service.delete_product(9999)  # Assuming ID 9999 doesn't exist
```



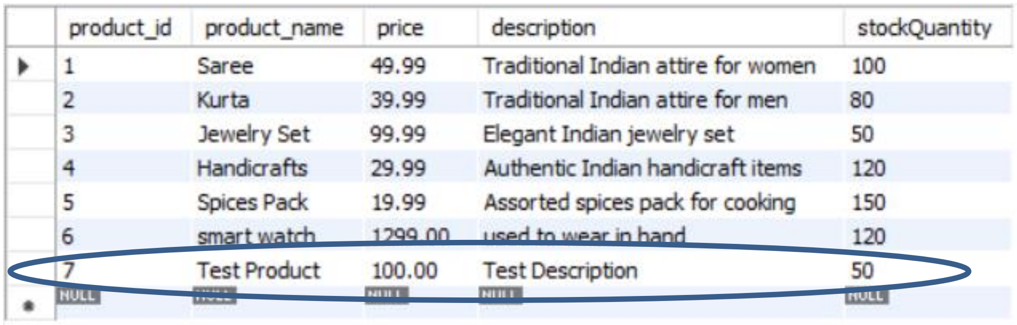
```python
    def test_delete_customer(self):
        # Assuming there's a customer with ID 1 in the database
        self.assertTrue(self.main_module.service.delete_customer(1))

        # Testing CustomerNotFoundException
        with self.assertRaises(CustomerNotFoundException):
            self.main_module.service.delete_customer(9999)  # Assuming ID 9999 doesn't exist

    @patch( target: 'builtins.input', side_effect=["1", "1", "5"])
    def test_add_to_cart(self, mock_inputs):
        # Assuming customer ID 1 and product ID 1 exist in the database
        self.assertTrue(self.main_module.service.add_to_cart(Customer(), Product(), 5))

    @patch( target: 'builtins.input', side_effect=["1", "1", "5"])
    def test_remove_from_cart(self, mock_inputs):
        # Assuming customer ID 1 and product ID 1 exist in the database
        self.assertTrue(self.main_module.service.remove_from_cart(Customer(), Product()))

    def test_place_order(self):
        # Assuming customer ID 1 and product ID 1 exist in the database
        self.assertTrue(self.main_module.service.place_order(Customer(), [{"product": Product(), "quantity": 5}]))

    def test_get_orders_by_customer(self):
        # Assuming customer ID 1 exists in the database
        self.assertIsNotNone(self.main_module.service.get_orders_by_customer(1))


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



```
"D:\python\Case study\.venv\Scripts\python.exe" "C:/Program Files/JetBrains/PyCharm Community Edition 2023.3.2/plugins/python-ce/helpers/
Testing started at 02:27 am ...
Launching pytest with arguments D:\python\Case study\testing\unit test.py --no-header --no-summary -q in D:\python\Case study\testing

============================ test session starts ============================
collecting ... collected 8 items

unit test.py::TestOrderProcessor::test_add_to_cart
unit test.py::TestOrderProcessor::test_create_customer
unit test.py::TestOrderProcessor::test_create_product
unit test.py::TestOrderProcessor::test_delete_customer
unit test.py::TestOrderProcessor::test_delete_product
unit test.py::TestOrderProcessor::test_get_orders_by_customer
unit test.py::TestOrderProcessor::test_place_order
unit test.py::TestOrderProcessor::test_remove_from_cart

======================== 3 failed, 5 passed in 0.19s ========================
PASSED                [ 12%]Enter the product details:
PASSED            [ 25%]Enter the customer details:
PASSED             [ 37%]Enter the product details:
```



	customer_id	name	email	password
▶	1	John Doe	john@example.com	password123
	2	Jane Smith	jane@example.com	securepass
	3	Amit Patel	amit@example.com	amit123
	4	Priya Sharma	priya@example.com	password456
	5	Rajesh Kumar	rajesh@example.com	rajeshpass
	6	Sankar Roy	sankarray101@gmail.com	password
	7	Test Customer	test@test.com	password
*	NULL	NULL	NULL	NULL



	product_id	product_name	price	description	stockQuantity
▶	1	Saree	49.99	Traditional Indian attire for women	100
	2	Kurta	39.99	Traditional Indian attire for men	80
	3	Jewelry Set	99.99	Elegant Indian jewelry set	50
	4	Handicrafts	29.99	Authentic Indian handicraft items	120
	5	Spices Pack	19.99	Assorted spices pack for cooking	150
	6	smart watch	1299.00	used to wear in hand	120
	7	Test Product	100.00	Test Description	50
*	NULL	NULL	NULL	NULL	NULL

© Hexaware Technologies Limited. All rights