Coding Challenge:-6

By SHREYASI REJA

Coding Challenge - Order Management System

Problem Statement:

Create SQL Schema from the product and user class, use the class attributes for table column names.

1. Create a base class called Product with the following attributes:

productId (int)

productName (String)

description (String)

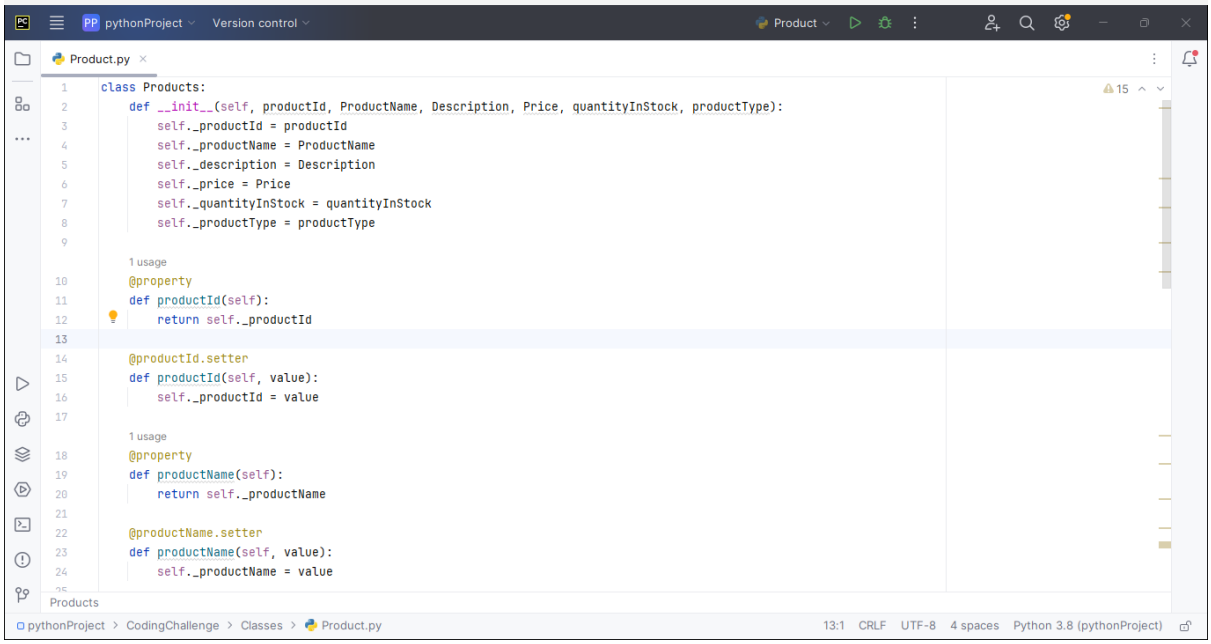
price (double)

quantityInStock (int)

type (String) [Electronics/Clothing]

2. Implement constructors, getters, and setters for the Product class.

CODE:-



```python
class Products:
    def __init__(self, productId, ProductName, Description, Price, quantityInStock, productType):
        self._productId = productId
        self._productName = ProductName
        self._description = Description
        self._price = Price
        self._quantityInStock = quantityInStock
        self._productType = productType

    1 usage
    @property
    def productId(self):
        return self._productId

    @productId.setter
    def productId(self, value):
        self._productId = value

    1 usage
    @property
    def productName(self):
        return self._productName

    @productName.setter
    def productName(self, value):
        self._productName = value
```

```python
    @property
    def description(self):
        return self._description

    @description.setter
    def description(self, value):
        self._description = value

    1 usage
    @property
    def price(self):
        return self._price

    @price.setter
    def price(self, value):
        self._price = value

    1 usage
    @property
    def quantityInStock(self):
        return self._quantityInStock

    @quantityInStock.setter
    def quantityInStock(self, value):
        self._quantityInStock = value
```

```python
        self._price = value

    1 usage
    @property
    def quantityInStock(self):
        return self._quantityInStock

    @quantityInStock.setter
    def quantityInStock(self, value):
        self._quantityInStock = value

    1 usage
    @property
    def productType(self):
        return self._productType

    @productType.setter
    def productType(self, value):
        self._productType = value
```

3. Create a subclass Electronics that inherits from Product. Add attributes specific to electronics products, such as:

brand (String)

warrantyPeriod (int)

code:-

```python
class Electronics(Products):
    def __init__(self, productId, productName, description, price, quantityInStock, productType, brand, warrantyPeriod):
        super().__init__(productId, productName, description, price, quantityInStock, productType)
        self._brand = brand
        self._warrantyPeriod = warrantyPeriod

    @property
    def brand(self):
        return self._brand

    @brand.setter
    def brand(self, value):
        self._brand = value

    @property
    def warrantyPeriod(self):
        return self._warrantyPeriod

    @warrantyPeriod.setter
    def warrantyPeriod(self, value):
        self._warrantyPeriod = value

electronics_product = Electronics(
    productId=1,
```

```python
electronics_product = Electronics(
    productId=1,
    productName="Laptop",
    description="High-performance laptop",
    price=999.99,
    quantityInStock=10,
    productType="Electronics",
    brand="XYZ",
    warrantyPeriod=12
)

print(electronics_product.productId)
print(electronics_product.productName)
print(electronics_product.description)
print(electronics_product.price)
print(electronics_product.quantityInStock)
print(electronics_product.productType)
print(electronics_product.brand)
print(electronics_product.warrantyPeriod)
```

Output:-

```python
electronics_product = Electronics(
    productId=1,
    productName="Laptop",
    description="High-performance laptop",
    price=999.99,
    quantityInStock=10,
    productType="Electronics",
    brand="XYZ",
    warrantyPeriod=12
```

```
"C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical traning code\pythonProject\.venv\Scripts\python.exe" "C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
1
Laptop
High-performance laptop
999.99
10
Electronics
XYZ
12

Process finished with exit code 0
```

4. Create a subclass Clothing that also inherits from Product. Add attributes specific to clothing products, such as:

size (String)

color (String)

```python
class Clothing(Products):
    def __init__(self, productId, productName, description, price, quantityInStock, productType, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, productType)
        self._size = size
        self._color = color

    @property
    def size(self):
        return self._size

    @size.setter
    def size(self, value):
        self._size = value

    @property
    def color(self):
        return self._color

    @color.setter
    def color(self, value):
        self._color = value

clothing_product = Clothing(
    productId=2,
```

```python
        self._color = value

clothing_product = Clothing(
    productId=2,
    productName="T-Shirt",
    description="Comfortable cotton t-shirt",
    price=19.99,
    quantityInStock=50,
    productType="Clothing",
    size="Medium",
    color="Blue"
)

print(clothing_product.productId)
print(clothing_product.productName)
print(clothing_product.description)
print(clothing_product.price)
print(clothing_product.quantityInStock)
print(clothing_product.productType)
print(clothing_product.size)
print(clothing_product.color)
```

Output:-

```python
        self._color = value

clothing_product = Clothing(
    productId=2,
    productName="T-Shirt",
    description="Comfortable cotton t-shirt",
    price=19.99,
    quantityInStock=50,
    productType="Clothing",
```

```
"C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical traning code\pythonProject\.venv\Scripts\python.exe" "C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
2
T-Shirt
Comfortable cotton t-shirt
19.99
50
Clothing
Medium
Blue

Process finished with exit code 0
```

5. Create a User class with attributes:

userId (int)

username (String)

password (String)

role (String) // "Admin" or "User"

```python
class User:
    def __init__(self, userId, username, password, role):
        self._userId = userId
        self._username = username
        self._password = password
        self._role = role

    1 usage
    @property
    def userId(self):
        return self._userId

    @userId.setter
    def userId(self, value):
        self._userId = value

    1 usage
    @property
    def username(self):
        return self._username

    @username.setter
    def username(self, value):
        self._username = value

    1 usage
```

```python
        self._username = value

    1 usage
    @property
    def password(self):
        return self._password

    @password.setter
    def password(self, value):
        self._password = value

    1 usage
    @property
    def role(self):
        return self._role

    @role.setter
    def role(self, value):
        self._role = value
```

6. Define an interface/abstract class named IOrderManagementRepository with methods for:

createOrder(User user, list of products): check the user as already present in database to create order or create user (store in database) and create order.

cancelOrder(int userId, int orderId): check the userid and orderId already present in database and cancel the order. if any userId or orderId not present in database throw exception corresponding UserNotFound or OrderNotFound exception

createProduct(User user, Product product): check the admin user as already present in database and create product and store in database.

createUser(User user): create user and store in database for further development.

getAllProducts(): return all product list from the database.

getOrderByUser(User user): return all product ordered by specific user from database.

Code:-

```python
from abc import ABC, abstractmethod

class IOrderManagementRepository(ABC):
    @abstractmethod
    def createOrder(self, user, products):
        pass

    @abstractmethod
    def cancelOrder(self, userId, orderId):
        pass

    @abstractmethod
    def createProduct(self, adminUser, product):
        pass

    @abstractmethod
    def createUser(self, user):
        pass

    @abstractmethod
    def getAllProducts(self):
        pass

    @abstractmethod
    def getOrderByUser(self, user):
        pass
```

7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.

```python
class UserNotFound(Exception):
    pass

class OrderNotFound(Exception):
    pass

from CodingChallenge.Service.IOrderManagementRepository import IOrderManagementRepository


class OrderProcessor(IOrderManagementRepository):
    def __init__(self):

        self.db_util = {}
        self.users_database = {}
        self.orders_database = {}
        self.products_database = {}
        self.admin_user = None

    def createOrder(self, user, products):
        if user.userId not in self.users_database:
            self.createUser(user)
        order_id = len(self.orders_database) + 1
        self.orders_database[order_id] = {'userId': user.userId, 'products': products}
        return order_id
```

```python
    def cancelOrder(self, userId, orderId):
        if userId not in self.users_database:
            raise UserNotFound("User not found in the database.")
        if orderId not in self.orders_database:
            raise OrderNotFound("Order not found in the database.")
        del self.orders_database[orderId]

    def createProduct(self, adminUser, product):
        if adminUser != self.admin_user:
            raise UserNotFound("Admin user not found in the database.")
        product_id = len(self.products_database) + 1
        self.products_database[product_id] = product
        return product_id

    def createUser(self, user):
        self.users_database[user.userId] = user

    def getAllProducts(self):
        return list(self.products_database.values())

    def getOrderByUser(self, user):
        user_orders = [order for order in self.orders_database.values() if order['userId'] == user.userId]
        return user_orders
```

8. Create DBUtil class and add the following method.

static getDBConn():Connection Establish a connection to the database and return database Connection

```python
from mysql.connector import connect


class DBUtil:
    def __init__(self, host, user, password, port, database):
        self.connection = connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor = self.connection.cursor()

    def execute_query(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            self.connection.commit()
            self.cursor.fetchall()
        except Exception as e:
            print(f"Query Execution Error! {e}")
            self.connection.rollback()

    def fetch_one(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            result = self.cursor.fetchone()
```

```python
            self.cursor.execute(query, values)
            self.connection.commit()
            self.cursor.fetchall()
        except Exception as e:
            print(f"Query Execution Error! {e}")
            self.connection.rollback()

    def fetch_one(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            result = self.cursor.fetchone()
            return result
        except Exception as e:
            print(f"FetchOne Error: {e}")
            self.connection.rollback()
            return None

    def close_connection(self):
        self.cursor.close()
        self.connection.close()
```

7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.

Code:-

```python
1 usage
class OrderNotFound(Exception):
    pass

from CodingChallenge.Service.IOrderManagementRepository import IOrderManagementRepository


class OrderProcessor(IOrderManagementRepository):
    def __init__(self):

        self.db_util = {}
        self.users_database = {}
        self.orders_database = {}
        self.products_database = {}
        self.admin_user = None

    def createOrder(self, user, products):
        if user.userId not in self.users_database:
            self.createUser(user)
        order_id = len(self.orders_database) + 1
        self.orders_database[order_id] = {'userId': user.userId, 'products': products}
        return order_id

    def cancelOrder(self, userId, orderId):
        if userId not in self.users_database:
            raise UserNotFound("User not found in the database.")
```

```python
    def cancelOrder(self, userId, orderId):
        if userId not in self.users_database:
            raise UserNotFound("User not found in the database.")
        if orderId not in self.orders_database:
            raise OrderNotFound("Order not found in the database.")
        del self.orders_database[orderId]

    def createProduct(self, adminUser, product):
        if adminUser != self.admin_user:
            raise UserNotFound("Admin user not found in the database.")
        product_id = len(self.products_database) + 1
        self.products_database[product_id] = product
        return product_id

    1 usage
    def createUser(self, user):
        self.users_database[user.userId] = user

    def getAllProducts(self):
        return list(self.products_database.values())

    def getOrderByUser(self, user):
        user_orders = [order for order in self.orders_database.values() if order['userId'] == user.userId]
        return user_orders
```

9. Create OrderManagement main class and perform following operation:

main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct", "cancelOrder", "getAllProducts", "getOrderbyUser", "exit".

```python
class OrderManagement:
    def __init__(self, order_processor):
        self.order_processor = order_processor

    def main(self):
        db_util = DBUtil(host='localhost', user='root', password='Root', port='3306', database='OrderManagementSystem')

        while True:
            print("\nMenu:")
            print("1. Create User")
            print("2. Create Product")
            print("3. Cancel Order")
            print("4. Get All Products")
            print("5. Get Order by User")
            print("6. Exit")

            choice = input("Enter your choice: ")

            if choice == '1':
                self.createUser()
            elif choice == '2':
                self.createProduct()
            elif choice == '3':
                self.cancelOrder()
            elif choice == '4':
                self.getAllProducts()
```

```python
                self.getAllProducts()
            elif choice == '5':
                self.getOrderByUser()
            elif choice == '6':
                print("Exiting...")
                break
            else:
                print("Invalid choice. Please try again.")

    1 usage
    def createUser(self):
        userId = int(input("Enter userId: "))
        username = input("Enter username: ")
        password = input("Enter password: ")
        role = input("Enter role (Admin/User): ")

        user = {'userId': userId, 'username': username, 'password': password, 'role': role}
        self.order_processor.create_user(user)
        print("User created successfully.")

    1 usage
    def createProduct(self):
        productId = int(input("Enter productId: "))
        productName = input("Enter productName: ")
        description = input("Enter description: ")
        price = float(input("Enter price: "))
```

pythonProject Version control DBUtil (1)

Product.py User.py IOrderManagementRepository.py DBUtil.py Main.py

```python
        quantityInStock = int(input("Enter quantityInStock: "))
        productType = input("Enter type (Electronics/Clothing): ")

        product = {'productId': productId, 'productName': productName, 'description': description,
                   'price': price, 'quantityInStock': quantityInStock, 'productType': productType}
        self.order_processor.create_product(product)
        print("Product created successfully.")

    1 usage
    def cancelOrder(self):
        orderId = int(input("Enter orderId: "))
        userId = int(input("Enter userId: "))

        try:
            self.order_processor.cancel_order(userId, orderId)
            print("Order canceled successfully.")
        except:
            print("Order not found.")

    1 usage
    def getAllProducts(self):
        products = self.order_processor.get_all_products()
        print("All Products:")
        for product in products:
            print(product)
```

OrderProcessor

pythonProject > CodingChallenge > Connection > DBUtil.py 82:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

pythonProject Version control DBUtil (1)

Product.py User.py IOrderManagementRepository.py DBUtil.py Main.py

```python
        except:
            print("Order not found.")

    1 usage
    def getAllProducts(self):
        products = self.order_processor.get_all_products()
        print("All Products:")
        for product in products:
            print(product)

    1 usage
    def getOrderByUser(self):
        userId = int(input("Enter userId: "))

        user_orders = self.order_processor.get_order_by_user(userId)
        if user_orders:
            print("Orders for user:")
            for order in user_orders:
                print(order)
        else:
            print("No orders found for the user.")
```

OrderProcessor

pythonProject > CodingChallenge > Connection > DBUtil.py 82:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)