Python coding challenge – Sivaprakas B M – Order Management System.

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]

SQL :

```
create database orders;
use orders;

CREATE TABLE Product (
   productId INT PRIMARY KEY,
   productName VARCHAR(255),
   description TEXT,
   price DOUBLE,
   quantityInStock INT,
   type VARCHAR(50)
);
```

```
mysql> create database orders;
Query OK, 1 row affected (0.01 sec)

mysql> use orders;
Database changed
mysql> CREATE TABLE Product (
    ->     productId INT PRIMARY KEY,
    ->     productName VARCHAR(255),
    ->     description TEXT,
    ->     price DOUBLE,
    ->     quantityInStock INT,
    ->     type VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Python:

class Product:

```
def __init__(self, productId, productName, description, price, quantityInStock, type):
    self.productId = productId
    self.productName = productName
    self.description = description
    self.price = price
    self.quantityInStock = quantityInStock
    self.type = type
```

```
Process finished with exit code 0
```

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

```python
class Product:
    def __init__(self, productId, productName, description, price, quantityInStock, type):
        self.quantityInStock = quantityInStock
        self.type = type


class Product:
    def __init__(self, productId, productName, description, price, quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.type = type

    # Getters
    def getProductId(self):
        return self.productId

    def getProductName(self):
        return self.productName

    def getDescription(self):
        return self.description

    def getPrice(self):
        return self.price

    def getQuantityInStock(self):
        return self.quantityInStock
```

```
    def getType(self):
        return self.type

    # Setters
    def setProductId(self, productId):
        self.productId = productId

    def setProductName(self, productName):
        self.productName = productName

    def setDescription(self, description):
        self.description = description

    def setPrice(self, price):
        self.price = price

    def setQuantityInStock(self, quantityInStock):
        self.quantityInStock = quantityInStock

    def setType(self, type):
        self.type = type
```

Implementation :

```
#Implementation

product1 = Product( productId: 1, productName: "Laptop", description: "High-performance laptop", price: 999.99, quantityInStock: 10, type: "Electronics")


print(product1.getProductName())
product1.setPrice(899.99)
print(product1.getPrice())
```

```
C:\Users\Siva\PycharmProjects\pythonProject\.venv\Scripts\python.exe (
Laptop
899.99
```

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

- **1)brand (String)**
- **2)warrantyPeriod (int)**

```
class Electronics(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, brand, warrantyPeriod):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.brand = brand
        self.warrantyPeriod = warrantyPeriod

    # Getter and setter for brand
    def getBrand(self):
        return self.brand

    def setBrand(self, brand):
        self.brand = brand

    # Getter and setter for warrantyPeriod
    def getWarrantyPeriod(self):
        return self.warrantyPeriod

    def setWarrantyPeriod(self, warrantyPeriod):
        self.warrantyPeriod = warrantyPeriod
```

Implementation :

```
# implementaion

electronics1 = Electronics( productId: 1,  productName: "Laptop",  description: "High-performance laptop",  price: 999.99,  quantityInStock: 10,  type: "Electronics",  brand: "Branc


print(electronics1.getBrand())
electronics1.setWarrantyPeriod(3)
print(electronics1.getWarrantyPeriod())
```

```
BrandX
3


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)**

```
class Clothing(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.size = size
        self.color = color

    # Getter and setter for size
    def getSize(self):
        return self.size

    def setSize(self, size):
        self.size = size

    # Getter and setter for color
    def getColor(self):
        return self.color

    def setColor(self, color):
        self.color = color
```

Implementation :

```
# Implementaion

clothing1 = Clothing( productId: 1, productName: "T-shirt", description: "Cotton T-shirt", price: 19.99, quantityInStock: 50, type: "Clothing", size: "M", color: "Red")

print(clothing1.getSize())
clothing1.setColor("Blue")
print(clothing1.getColor())
```

```
M
Blue

Process finished with exit code 0
```

5) Create a User class with attributes:

- **userId (int)**
- **username (String)**
- **password (String)**
- **role (String) (either "Admin" or "User")**

```
class User:
    def __init__(self, userId, username, password, role):
        self.username = username
        self.password = password
        self.role = role

    # Getters
    def getUserId(self):
        return self.userId

    def getUsername(self):
        return self.username

    def getPassword(self):
        return self.password

    def getRole(self):
        return self.role

    # Setters
    def setUserId(self, userId):
        self.userId = userId

    def setUsername(self, username):
        self.username = username

    def setPassword(self, password):
        self.password = password

    def setRole(self, role):
        self.role = role
```

Implementation :

```
# Implementation

# Creating a User object
user1 = User( userId: 1,  username: "siva",  password: "password123",  role: "Admin")

# Getting and setting attributes
print(user1.getUsername())
user1.setPassword("new_password")
print(user1.getPassword())
```

```
siva
new_password

Process finished with exit code 0
```

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, user, 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.

CODE:

```
class OrderProcessor(IOrderManagementRepository):
    def createOrder(self, user, products):

        # Check if the user is already present in the database
        if self.isUserPresent(user):
            print("User already exists in the database.")
        else:
            self.createUser(user)
            print("User created and stored in the database.")

        # Create the order
        order_successful = self.storeOrder(user, products)
        if order_successful:
            print("Order created successfully.")
        else:
            print("Failed to create the order.")
        return order_successful

    1 usage
    def isUserPresent(self, user):

        return False

    1 usage
    def createUser(self, user):

        pass

    1 usage
    def storeOrder(self, user, products):
        return True
```

8) Create DBUtil class and add the following method.

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

CODE :

```
class DBUtil:
    @staticmethod
    def getDBConn():
        # Establish connection to the database
        try:
            connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="Siva",
                database="3306"
            )
            print("Database connection established.")
            return connection
        except mysql.connector.Error as err:
            print("Error: ", err)
```

```
<mysql.connector.connection_cext.CMySQLConnection object at 0x000001AA47D5C8F0>

Process finished with exit code 0
```

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".**

CODE :

```
class OrderManagement:
    @staticmethod
    def main():
        order_processor = OrderProcessor()

        while True:
            print("\n=== Order Management System ===")
            print("1. Create User")
            print("2. Create Product")
            print("3. Cancel Order")
```

```
            print("4. Get All Products")
            print("5. Get Orders by User")
            print("6. Exit")
            choice = input("Enter your choice: ")

            if choice == "1":
                OrderManagement.createUser(order_processor)
            elif choice == "2":
                OrderManagement.createProduct(order_processor)
            elif choice == "3":
                OrderManagement.cancelOrder(order_processor)
            elif choice == "4":
                OrderManagement.getAllProducts(order_processor)
            elif choice == "5":
                OrderManagement.getOrdersByUser(order_processor)
            elif choice == "6":
                print("Exiting Order Management System. Goodbye!")
                break
            else:
                print("Invalid choice. Please enter a valid option.")

    @staticmethod
    def createUser(order_processor):
        userId = int(input("Enter User ID: "))
        username = input("Enter Username: ")
        password = input("Enter Password: ")
        role = input("Enter Role (Admin/User): ")

        user = User(userId, username, password, role)
        order_processor.createUser(user)

    @staticmethod
    def createProduct(order_processor):
        productId = int(input("Enter Product ID: "))
        productName = input("Enter Product Name: ")
        description = input("Enter Description: ")
        price = float(input("Enter Price: "))
        quantityInStock = int(input("Enter Quantity in Stock: "))
        type = input("Enter Type (Electronics/Clothing): ")

        product = Product(productId, productName, description, price,
quantityInStock, type)
        order_processor.createProduct(None, product)  # Assuming None
for admin user

    @staticmethod
    def cancelOrder(order_processor):
        userId = int(input("Enter User ID: "))
        orderId = int(input("Enter Order ID: "))

        order_processor.cancelOrder(userId, orderId)

    @staticmethod
    def getAllProducts(order_processor):
        products = order_processor.getAllProducts()
        print("All Products:")
        for product in products:
            print(product.getProductName())

    @staticmethod
    def getOrdersByUser(order_processor):
```

```
            userId = int(input("Enter User ID: "))
            # Assuming getOrderByUser returns a list of orders for the
    given user
            orders = order_processor.getOrderByUser(None)  # Assuming
    None for user
            print("Orders by User:")
            for order in orders:
                print("Order ID:", order.getId())  # Assuming there's a
    getId() method for orders


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

```
=== Order Management System ===
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
Enter your choice: 1
Enter User ID: 1
Enter Username: siva
Enter Password: sia
Enter Role (Admin/User): Admin

=== Order Management System ===
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
Enter your choice: 2
Enter Product ID: 1
Enter Product Name: Phone
Enter Description: Gadget
Enter Price: 10000
Enter Quantity in Stock: 100
Enter Type (Electronics/Clothing): Electronics
```

```
=== Order Management System ===
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
Enter your choice: 6
Exiting Order Management System. Goodbye!

Process finished with exit code 0
```