Order Management

1.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]

```
CREATE TABLE products (
    productId INT PRIMARY KEY,
    productName VARCHAR(255),
    description TEXT,
    price DOUBLE,
    quantityInStock INT,
    type VARCHAR(20)
);
```

productId	productName	description	price	quantityInStock	type
1	Laptop	High-performance laptop	1200.5	10	Electronics
2	T-Shirt	Cotton T-Shirt	25.99	50	Clothing
3	Smartphone	Latest smartphone model	899.99	20	Electronics
4	Jeans	Denim jeans	39.99	30	Clothing
5	Headphones	Noise-canceling headphones	149.99	15	Electronics
6	Dress Shirt	Formal shirt	49.99	40	Clothing
7	Tablet	Tablet device	299.99	12	Electronics
8	Sneakers	Casual shoes	79.99	25	Clothing
9	Desktop Computer	Powerful desktop computer	1500	8	Electronics
10	Skirt	Women's skirt	29.99	35	Clothing
NULL	NULL	NULL	NULL	NULL	NULL

```
CREATE TABLE users (
userId INT PRIMARY KEY,
username VARCHAR(255),
password VARCHAR(255),
role ENUM('Admin', 'User')
);
```

	userId	username	password	role
▶	1	john_doe	password123	User
	2	jane_doe	password456	Admin
	3	alice_smith	password789	User
	4	bob_johnson	password101	Admin
	5	emma_watson	password202	User
	6	chris_evans	password303	Admin
	7	sarah_parker	password404	User
	8	michael_jordan	password505	Admin
	9	lisa_wong	password606	User
	10	david_jackson	password707	Admin
*	NULL	NULL	NULL	NULL

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

```
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


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):
    if type.lower() in ["clothing", "electronics"]:
        self.type = type.lower()
    else:
        raise ValueError("Invalid product type. Must be 'Clothing' or 'Electronics'.")
```

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

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

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,
size, color):
        super().__init__(productId, productName, description, price, quantityInStock,
Product.Type.CLOTHING)
        self.size = size
        self.color = color
```

5.Create a User class with attributes: • userId (int) • username (String) • password (String) • role (String) // "Admin" or "User"

```
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        if role in ["Admin", "User"]:
            self.role = role
        else:
            raise ValueError("Invalid user role")
```

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.

```
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

class UserNotFound(Exception):
        pass

class OrderNotFound(Exception):
        pass
class InMemoryOrderManagementRepository(IOrderManagementRepository):
    def __init__(self):
        self.users = {}
        self.products = {}
        self.orders = {}

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

    def cancelOrder(self, userId, orderId):
        if userId not in self.users:
            raise UserNotFound("User with ID {} not found.".format(userId))
        if orderId not in self.orders:
            raise OrderNotFound("Order with ID {} not found.".format(orderId))
        del self.orders[orderId]

    def createProduct(self, user, product):
        if user.role != "Admin":
            raise PermissionError("User does not have permission to create products.")
        self.products[product.productId] = product

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

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

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

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

```
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


class OrderProcessor(IOrderManagementRepository):
    def __init__(self):
        self.users = {}
        self.products = {}
        self.orders = {}

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

    def cancelOrder(self, userId, orderId):
        if userId not in self.users:
            raise UserNotFound("User with ID {} not found.".format(userId))
        if orderId not in self.orders:
            raise OrderNotFound("Order with ID {} not found.".format(orderId))
        del self.orders[orderId]

    def createProduct(self, user, product):
        if user.role != "Admin":
            raise PermissionError("User does not have permission to create products.")
        self.products[product.productId] = product

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

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

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


class UserNotFound(Exception):
    pass


class OrderNotFound(Exception):
    pass
```

8.Create DBUtil class and add the following method. • static getDBConn():Connection Establish a connection to the database and return database Connection

```
import mysql.connector

class DBConnection:
    @staticmethod
    def getConnection():
        # property=PropertyUtil.getPropertyString()
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="diana",
            database="ordermanagement"
        )
        return conn
```

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".

```
class OrderManagement:
    def __init__(self):
        pass

    def main(self):
        print("Welcome to Order Management System")

        while True:
            print("\nMenu:")
            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":
                self.create_user()
            elif choice == "2":
                self.create_product()
            elif choice == "3":
                self.cancel_order()
            elif choice == "4":
                self.get_all_products()
            elif choice == "5":
                self.get_orders_by_user()
            elif choice == "6":
                print("Exiting...")
                break
            else:
                print("Invalid choice. Please try again.")

    def create_user(self):
        print("Creating a new user...")

        print("User created successfully.")

    def create_product(self):
        print("Creating a new product...")
        print("Product created successfully.")

    def cancel_order(self):
```

```
        print("Canceling an order...")
        # Placeholder implementation

    def get_all_products(self):
        print("Retrieving all products...")


    def get_orders_by_user(self):
        print("Retrieving orders by user...")


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

```
Welcome to Order Management System


Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
Enter your choice: 3
Canceling an order...

Menu:
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...
```