## CODING CHALLENGE

## ORDER MANAGEMENT SYSTEM

### Problem Statement:

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

- CREATE TABLE User (userId INT PRIMARY KEY, username VARCHAR(100), password VARCHAR(100), role VARCHAR(20));
- CREATE TABLE Product (productId INT PRIMARY KEY,productName VARCHAR(100), description TEXT, price DOUBLE, quantityInStock INT, VARCHAR(20), brand VARCHAR(100), warrantyPeriod INT,size VARCHAR(100), color VARCHAR(20));
- CREATE TABLE Orders (orderId INT PRIMARY KEY AUTO_INCREMENT, userId INT, FOREIGN KEY (userId) REFERENCES User(userId));
- CREATE TABLE OrderItems (orderItemId INT PRIMARY KEY AUTO_INCREMENT, orderId INT, productId INT, FOREIGN KEY (orderId) REFERENCES Orders(orderId), FOREIGN KEY (productId) REFERENCES Product(productId));

### 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]

### Code:

```
class Product:
    def __init__(self, product_id: int, product_name: str, description: str,
                 price: float, quantity_in_stock: int, type_: str):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
        self.quantity_in_stock = quantity_in_stock
        self.type = type_
```

```
    def __str__(self):
        return (f"Product[ID={self.product_id}, Name={self.product_name}, "
                f"Description={self.description}, Price={self.price}, "
                f"Stock={self.quantity_in_stock}, Type={self.type}]")
```

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

**Code:**

```
def get_product_id(self):
        return self.product_id
    def get_product_name(self):
        return self.product_name
    def get_description(self):
        return self.description
    def get_price(self):
        return self.price
    def get_quantity_in_stock(self):
        return self.quantity_in_stock
    def get_type(self):
        return self.type

    def set_product_id(self, product_id):
        self.product_id = product_id
    def set_product_name(self, product_name):
        self.product_name = product_name
    def set_description(self, description):
        self.description = description
    def set_price(self, price):
        self.price = price
    def set_quantity_in_stock(self, quantity):
        self.quantity_in_stock = quantity
    def set_type(self, type_):
        self.type = type_
```

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

• brand (String)

• warrantyPeriod (int)

**Code:**

```
from .product import Product
class Electronics(Product):
    def __init__(self, product_id: int, product_name: str, description: str,
                 price: float, quantity_in_stock: int, type_: str,
                 brand: str, warranty_period: int):
        super().__init__(product_id, product_name, description, price, quantity_in_stock, type_)
        self.brand = brand
        self.warranty_period = warranty_period

    def get_brand(self):
        return self.brand
    def get_warranty_period(self):
        return self.warranty_period

    def set_brand(self, brand):
        self.brand = brand
    def set_warranty_period(self, warranty_period):
        self.warranty_period = warranty_period
    def __str__(self):
        return (super().__str__() +
                f", Brand={self.brand}, Warranty={self.warranty_period} months")
```

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

**• size (String)**

**• color (String)**

**Code:**

```
from .product import Product
class Clothing(Product):
    def __init__(self, product_id: int, product_name: str, description: str,
                 price: float, quantity_in_stock: int, type_: str,
                 size: str, color: str):
        super().__init__(product_id, product_name, description, price, quantity_in_stock, type_)
        self.size = size
        self.color = color
```

```
    def get_size(self):
        return self.size
    def get_color(self):
        return self.color

    def set_size(self, size):
        self.size = size
    def set_color(self, color):
        self.color = color

    def __str__(self):
        return (super().__str__() +
                f", Size={self.size}, Color={self.color}")
```

**5. Create a User class with attributes:**

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

**Code:**

```
class User:
    def __init__(self, user_id: int, username: str, password: str, role: str):
        self.user_id = user_id
        self.username = username
        self.password = password
        self.role = role

    def get_user_id(self):
        return self.user_id
    def get_username(self):
        return self.username
    def get_password(self):
        return self.password
    def get_role(self):
        return self.role

    def set_user_id(self, user_id):
        self.user_id = user_id
```

```
    def set_username(self, username):
        self.username = username
    def set_password(self, password):
        self.password = password
    def set_role(self, role):
        self.role = role
    def __str__(self):
        return f"User[ID={self.user_id}, Username={self.username}, Role={self.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.**

**Code:**

```
from dao.order_repo import IOrderManagementRepository
from entity.user import User
from entity.product import Product
from typing import List
from exception.user_not_found_exception import UserNotFoundException
from exception.order_not_found_exception import OrderNotFoundException
from util.db_conn_util import DBUtil
class OrderProcessor(IOrderManagementRepository):
```

```
    def __init__(self):
        pass
    def create_user(self, user: User):
        conn = DBUtil.get_db_conn()
        cursor = conn.cursor()
        query = "INSERT INTO User (userId, username, password, role) VALUES (%s, %s, %s,
%s)"
        values = (user.get_user_id(), user.get_username(), user.get_password(), user.get_role())
        try:
            cursor.execute(query, values)
            conn.commit()
            print(f" User created successfully: {user}")
        except Exception as e:
            print(f"Error inserting user: {e}")
        finally:
            cursor.close()
            conn.close()
    def create_product(self, user: User, product: Product):
        if user.get_role().lower() != "admin":
            print(" Only admin users can create products.")
            return
        conn = DBUtil.get_db_conn()
        cursor = conn.cursor()
        query = """
            INSERT INTO Product
            (productId, productName, description, price, quantityInStock, type, brand,
warrantyPeriod, size, color)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
```

```
    values = (
        product.get_product_id(), product.get_product_name(), product.get_description(),
        product.get_price(), product.get_quantity_in_stock(), product.get_type(),
        getattr(product, "brand", None),
        getattr(product, "warranty_period", None),
        getattr(product, "size", None),
        getattr(product, "color", None)
    )
    try:
        cursor.execute(query, values)
        conn.commit()
        print(" Product inserted successfully.")
    except Exception as e:
        print(" Error inserting product:", e)
    finally:
        cursor.close()
        conn.close()
def create_order(self, user: User, products: List[Product]):
    conn = DBUtil.get_db_conn()
    cursor = conn.cursor()

    try:
        order_query = "INSERT INTO Orders (userId) VALUES (%s)"
        cursor.execute(order_query, (user.get_user_id(),))
        order_id = cursor.lastrowid
        item_query = "INSERT INTO OrderItems (orderId, productId) VALUES (%s, %s)"
        for product in products:
            cursor.execute(item_query, (order_id, product.get_product_id()))
```

```
        conn.commit()
        print(f"Order created for user {user.get_username()} with Order ID: {order_id}")
    except Exception as e:
        print(" Error creating order:", e)
    finally:
        cursor.close()
        conn.close()
def cancel_order(self, user_id: int, order_id: int):
    conn = DBUtil.get_db_conn()
    cursor = conn.cursor()
    try:
        check_query = "SELECT * FROM Orders WHERE orderId = %s AND userId = %s"
        cursor.execute(check_query, (order_id, user_id))
        if cursor.fetchone() is None:
            raise OrderNotFoundException(f"Order {order_id} for user {user_id} not found.")

        cursor.execute("DELETE FROM OrderItems WHERE orderId = %s", (order_id,))
        cursor.execute("DELETE FROM Orders WHERE orderId = %s", (order_id,))
        conn.commit()
        print(f"Order {order_id} cancelled successfully.")
    except Exception as e:
        print(" Error cancelling order:", e)
    finally:
        cursor.close()
        conn.close()
def get_all_products(self) -> List[Product]:
    conn = DBUtil.get_db_conn()
    cursor = conn.cursor()
```

```
    try:
        cursor.execute("SELECT * FROM Product")
        products = cursor.fetchall()
        for prod in products:
            print(prod)
        return products
    except Exception as e:
        print("Error fetching products:", e)
        return []
    finally:
        cursor.close()
        conn.close()
def get_order_by_user(self, user: User):
    conn = DBUtil.get_db_conn()
    cursor = conn.cursor()
    try:
        query = """
            SELECT o.orderId, p.productId, p.productName, p.description, p.price, p.type
            FROM Orders o
            JOIN OrderItems oi ON o.orderId = oi.orderId
            JOIN Product p ON oi.productId = p.productId
            WHERE o.userId = %s
        """
        cursor.execute(query, (user.get_user_id(),))
        orders = cursor.fetchall()
        for item in orders:
            print(item)
        return orders
```

```
        except Exception as e:
            print(" Error fetching orders for user:", e)
            return []
        finally:
            cursor.close()
            conn.close()
```

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

**Code:**

```
from abc import ABC, abstractmethod
from entity.user import User
from entity.product import Product
from typing import List
class IOrderManagementRepository(ABC):
    @abstractmethod
    def create_order(self, user: User, products: List[Product]):
        pass
    @abstractmethod
    def cancel_order(self, user_id: int, order_id: int):
        pass
    @abstractmethod
    def create_product(self, user: User, product: Product):
        pass
    @abstractmethod
    def create_user(self, user: User):
        pass
    @abstractmethod
```

```
    def get_all_products(self) -> List[Product]:
        pass
    @abstractmethod
    def get_order_by_user(self, user: User):
        pass
```

**8. Create DBUtil class and add the following method.**

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

**Code:**

```
import mysql.connector
class DBUtil:
    @staticmethod
    def get_db_conn():
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",
                password="VINO",
                database="OrderManagement"
            )
            print("MySQL database connection established.")
            return conn
        except mysql.connector.Error as e:
            print("Database connection failed:", e)
            return None
```

**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:**

```
import sys
import os
from typing import List

project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))
if project_root not in sys.path:
    sys.path.insert(0, project_root)

from dao.order_processor import OrderProcessor
from entity.user import User
from entity.product import Product
from entity.electronics import Electronics
from entity.clothing import Clothing

def create_user_input():
    user_id = int(input("Enter User ID: "))
    username = input("Enter Username: ")
    password = input("Enter Password: ")
    role = input("Enter Role (Admin/User): ")
    return User(user_id, username, password, role)
def create_product_input():
    product_id = int(input("Enter Product ID: "))
    name = input("Enter Product Name: ")
    desc = input("Enter Description: ")
```

```
    price = float(input("Enter Price: "))
    stock = int(input("Enter Quantity in Stock: "))
    type_ = input("Enter Product Type (Electronics/Clothing): ")
    if type_.lower() == "electronics":
        brand = input("Enter Brand: ")
        warranty = int(input("Enter Warranty Period (in months): "))
        return Electronics(product_id, name, desc, price, stock, type_, brand, warranty)
    elif type_.lower() == "clothing":
        size = input("Enter Size: ")
        color = input("Enter Color: ")
        return Clothing(product_id, name, desc, price, stock, type_, size, color)
    else:
        print("Invalid product type.")
        return None
def create_order_input():
    product_ids = input("Enter Product IDs (comma-separated): ").split(',')
    products = []
    for pid in product_ids:
        prod = Product(int(pid), "", "", 0, 0, "")
        products.append(prod)
    return products
if __name__ == "__main__":
    processor = OrderProcessor()
    while True:
        print("\n--- Order Management Menu ---")
        print("1. Create User")
        print("2. Create Product")
        print("3. Create Order")
```

```
print("4. Cancel Order")
print("5. Get All Products")
print("6. Get Order by User")
print("7. Exit")
choice = input("Enter your choice (1-7): ")
if choice == "1":
    user = create_user_input()
    processor.create_user(user)
elif choice == "2":
    user = create_user_input()
    product = create_product_input()
    if product:
        processor.create_product(user, product)
elif choice == "3":
    user = create_user_input()
    products = create_order_input()
    processor.create_order(user, products)
elif choice == "4":
    uid = int(input("Enter User ID: "))
    oid = int(input("Enter Order ID: "))
    processor.cancel_order(uid, oid)
elif choice == "5":
    processor.get_all_products()
elif choice == "6":
    uid = int(input("Enter User ID: "))
    username = input("Enter Username: ")
    password = input("Enter Password: ")
    role = input("Enter Role: ")
```

```
        user = User(uid, username, password, role)
        processor.get_order_by_user(user)
    elif choice == "7":
        print("Exiting... Bye!")
        break
    else:
        print("Invalid choice. Please try again.")
```

**SCREENSHOTS:**

**1.Create Users**

```
PS E:\OrderManagement> python main/main_module.py

--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 1
Enter User ID: 230
Enter Username: Manisha
Enter Password: 123
Enter Role (Admin/User): User
MySQL database connection established.
 User created successfully: User[ID=230, Username=Manisha, Role=User]
```

```
PS E:\OrderManagement> python main/main_module.py

--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 1
Enter User ID: 231
Enter Username: Latha
Enter Password: 456
Enter Role (Admin/User): Admin
MySQL database connection established.
 User created successfully: User[ID=231, Username=Latha, Role=Admin]
```

## 2.Create Product

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 2
Enter User ID: 231
Enter Username: Latha
Enter Password: 456
Enter Role (Admin/User): Admin
Enter Product ID: 240
Enter Product Name: Mobile
Enter Description: Samsung galaxy A30
Enter Price: 30000
Enter Quantity in Stock: 9
Enter Product Type (Electronics/Clothing): Electronics
Enter Brand: Samsung
Enter Warranty Period (in months): 8
MySQL database connection established.
 Product inserted successfully.
```

## 3. Create order

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 3
Enter User ID: 230
Enter Username: Manisha
Enter Password: 123
Enter Role (Admin/User): User
Enter Product IDs (comma-separated): 240
MySQL database connection established.
Order created for user Manisha with Order ID: 1
```

## 4.Cancel Order

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 4
Enter User ID: 230
Enter Order ID: 1
MySQL database connection established.
Order 1 cancelled successfully.
```

## 5. Get all products

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 5
MySQL database connection established.
(240, 'Mobile', 'Samsung galaxy A30', 30000.0, 9, 'Electronics', 'Samsung', 8, None, None)
```

## 6. Get Order by User

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 6
Enter User ID: 230
Enter Username: Manisha
Enter Password: 123
Enter Role: User
MySQL database connection established.
(2, 240, 'Mobile', 'Samsung galaxy A30', 30000.0, 'Electronics')
```

## 7.Exit

```
--- Order Management Menu ---
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Order by User
7. Exit
Enter your choice (1-7): 7
Exiting... Bye!
```