**CODING CHALLENGE**

Order Management System

Agenda:

Implementation of SQL

Implementation of OOPS

BY

JASWANTH KUAMR S

Batch 4

1. **Create a base class called Product with the following attributes:**
2. **Implement constructors, getters, and setters for the Product class.**

**Code :**

**Product**

class Product:  
 def \_\_init\_\_(self, product\_id, product\_name, description, price, quantity\_in\_stock, product\_type):  
 self.product\_id = product\_id  
 self.product\_name = product\_name  
 self.description = description  
 self.price = price  
 self.quantity\_in\_stock = quantity\_in\_stock  
 self.product\_type = product\_type  
  
 def \_\_str\_\_(self):  
 return f"{self.product\_name} ({self.product\_type}) - ${self.price}"

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

**Electronics**

class Electronics:  
 def \_\_init\_\_(self, product\_id, product\_name, description, price, quantity\_in\_stock, product\_type, brand, warranty\_period):  
 self.product\_id = product\_id  
 self.product\_name = product\_name  
 self.description = description  
 self.price = price  
 self.quantity\_in\_stock = quantity\_in\_stock  
 self.product\_type = product\_type  
 self.brand = brand  
 self.warranty\_period = warranty\_period

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

**clothing**

class Clothing:  
 def \_\_init\_\_(self, product\_id, product\_name, description, price, quantity\_in\_stock, product\_type, size, color):  
 self.product\_id = product\_id  
 self.product\_name = product\_name  
 self.description = description  
 self.price = price  
 self.quantity\_in\_stock = quantity\_in\_stock  
 self.product\_type = product\_type  
 self.size = size  
 self.color = color

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

**user**

class User:  
 def \_\_init\_\_(self, user\_id, username, password, role):  
 self.user\_id = user\_id  
 self.username = username  
 self.password = password  
 self.role = role

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

CREATE DATABASE IF NOT EXISTS order\_management;

USE order\_management;

CREATE TABLE IF NOT EXISTS users (

user\_id INT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

password VARCHAR(50) NOT NULL,

role VARCHAR(20) CHECK (role IN ('Admin', 'User'))

);

CREATE TABLE IF NOT EXISTS products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

description TEXT,

price DOUBLE,

quantity\_in\_stock INT,

product\_type VARCHAR(50)

);

CREATE TABLE IF NOT EXISTS electronics (

product\_id INT PRIMARY KEY,

brand VARCHAR(50),

warranty\_period INT,

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

CREATE TABLE IF NOT EXISTS clothing (

product\_id INT PRIMARY KEY,

size VARCHAR(10),

color VARCHAR(30),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

CREATE TABLE IF NOT EXISTS orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT,

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

CREATE TABLE IF NOT EXISTS order\_items (

order\_id INT,

product\_id INT,

quantity INT,

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

PRIMARY KEY (order\_id, product\_id)

);

**7. Define an interface/abstract class named OrderManagementRepository with methods**

**Implement the OrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.[DAO]**

**OrderManagementRepository**

from abc import ABC, abstractmethod  
  
class OrderManagementRepository(ABC):  
  
 @abstractmethod  
 def create\_user(self, user):  
 pass  
  
 @abstractmethod  
 def create\_product(self, user, product):  
 pass  
  
 @abstractmethod  
 def create\_order(self, user, product\_list):  
 pass  
  
 @abstractmethod  
 def cancel\_order(self, user\_id, order\_id):  
 pass  
  
 @abstractmethod  
 def get\_all\_products(self):  
 pass  
  
 @abstractmethod  
 def get\_order\_by\_user(self, user):  
 pass

**OrderProcessor**

import mysql.connector  
from dao.OrderManagementRepository import OrderManagementRepository  
from util.DBConnUtil import DBConnUtil  
from exception.UserNotFoundException import UserNotFoundException  
from exception.OrderNotFoundException import OrderNotFoundException  
  
class OrderProcessor(OrderManagementRepository):  
  
 def \_\_init\_\_(self):  
 self.conn = DBConnUtil.get\_connection()  
 self.cursor = self.conn.cursor(dictionary=True)  
  
 def create\_user(self, user):  
 self.cursor.execute("SELECT \* FROM users WHERE user\_id = %s", (user.user\_id,))  
 if not self.cursor.fetchone():  
 self.cursor.execute(  
 "INSERT INTO users (user\_id, username, password, role) VALUES (%s, %s, %s, %s)",  
 (user.user\_id, user.username, user.password, user.role)  
 )  
 self.conn.commit()  
  
 def create\_product(self, user, product):  
 self.cursor.execute("SELECT \* FROM users WHERE user\_id = %s AND role = 'Admin'", (user.user\_id,))  
 if not self.cursor.fetchone():  
 raise UserNotFoundException("Admin user not found.")  
  
 self.cursor.execute("SELECT \* FROM products WHERE product\_id = %s", (product.product\_id,))  
 if self.cursor.fetchone():  
 print("Product already exists.")  
 return  
  
 self.cursor.execute(  
 "INSERT INTO products (product\_id, product\_name, description, price, quantity\_in\_stock, product\_type) "  
 "VALUES (%s, %s, %s, %s, %s, %s)",  
 (product.product\_id, product.product\_name, product.description,  
 product.price, product.quantity\_in\_stock, product.product\_type)  
 )  
  
 if product.product\_type == "Electronics":  
 self.cursor.execute(  
 "INSERT INTO electronics (product\_id, brand, warranty\_period) VALUES (%s, %s, %s)",  
 (product.product\_id, product.brand, product.warranty\_period)  
 )  
 elif product.product\_type == "Clothing":  
 self.cursor.execute(  
 "INSERT INTO clothing (product\_id, size, color) VALUES (%s, %s, %s)",  
 (product.product\_id, product.size, product.color)  
 )  
  
 self.conn.commit()  
  
 def create\_order(self, user, product\_list):  
 self.cursor.execute("SELECT \* FROM users WHERE user\_id = %s", (user.user\_id,))  
 if not self.cursor.fetchone():  
 self.create\_user(user)  
  
 self.cursor.execute("INSERT INTO orders (user\_id) VALUES (%s)", (user.user\_id,))  
 order\_id = self.cursor.lastrowid  
  
 for item in product\_list:  
 self.cursor.execute(  
 "INSERT INTO order\_items (order\_id, product\_id, quantity) VALUES (%s, %s, %s)",  
 (order\_id, item['product\_id'], item['quantity'])  
 )  
  
 self.conn.commit()  
 print(f"Order #{order\_id} created for user {user.user\_id}")  
  
 def cancel\_order(self, user\_id, order\_id):  
 self.cursor.execute("SELECT \* FROM users WHERE user\_id = %s", (user\_id,))  
 if not self.cursor.fetchone():  
 raise UserNotFoundException()  
  
 self.cursor.execute("SELECT \* FROM orders WHERE order\_id = %s AND user\_id = %s", (order\_id, user\_id))  
 if not self.cursor.fetchone():  
 raise OrderNotFoundException()  
  
 self.cursor.execute("DELETE FROM order\_items WHERE order\_id = %s", (order\_id,))  
 self.cursor.execute("DELETE FROM orders WHERE order\_id = %s", (order\_id,))  
 self.conn.commit()  
  
 def get\_all\_products(self):  
 self.cursor.execute("SELECT \* FROM products")  
 return self.cursor.fetchall()  
  
 def get\_order\_by\_user(self, user):  
 self.cursor.execute("SELECT \* FROM orders WHERE user\_id = %s", (user.user\_id,))  
 orders = self.cursor.fetchall()  
 result = []  
 for order in orders:  
 self.cursor.execute("SELECT \* FROM order\_items WHERE order\_id = %s", (order['order\_id'],))  
 items = self.cursor.fetchall()  
 result.append({'order\_id': order['order\_id'], 'items': items})  
 return result

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

import os  
import configparser  
  
class DBPropertyUtil:  
 @staticmethod  
 def get\_connection\_string(filename):  
 config = configparser.ConfigParser()full\_path = os.path.join(os.path.dirname(\_\_file\_\_), '..', filename)  
 config.read(full\_path)  
  
 if not config.has\_section("mysql"):  
 raise Exception("Missing [mysql] section in db.properties")  
  
 return {  
 "host": config.get("mysql", "host"),  
 "user": config.get("mysql", "user"),  
 "password": config.get("mysql", "password"),  
 "database": config.get("mysql", "database")  
 }

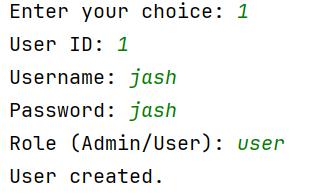
import mysql.connector  
from util.DBPropertyUtil import DBPropertyUtil  
  
class DBConnUtil:  
 @staticmethod  
 def get\_connection():  
 props = DBPropertyUtil.get\_connection\_string("db.properties")  
 conn = mysql.connector.connect(  
 host=props["host"],  
 user=props["user"],  
 password=props["password"],  
 database=props["database"]  
 )  
 return conn

**9. Create OrderManagement main class and perform following operation: • main method . Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct", "cancelOrder", "getAllProducts", "getOrderbyUser", "exit".**

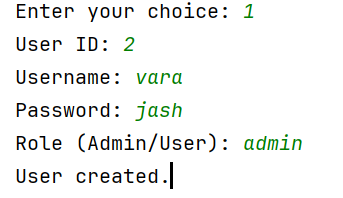
from dao.OrderProcessor import OrderProcessor  
from entity.user import User  
from entity.electronics import Electronics  
from entity.clothing import Clothing  
  
def main():  
 processor = OrderProcessor()  
  
 while True:  
 print("\n===== Order Management System =====")  
 print("1. Create User")  
 print("2. Create Product")  
 print("3. Create Order")  
 print("4. Cancel Order")  
 print("5. Get All Products")  
 print("6. Get Orders by User")  
 print("7. Exit")  
 choice = input("Enter your choice: ")  
  
 if choice == '1':  
 uid = int(input("User ID: "))  
 uname = input("Username: ")  
 pwd = input("Password: ")  
 role = input("Role (Admin/User): ")  
 user = User(uid, uname, pwd, role)  
 processor.create\_user(user)  
 print("User created.")  
  
 elif choice == '2':  
 uid = int(input("Admin User ID: "))  
 user = User(uid, "", "", "Admin")  
  
 pid = int(input("Product ID: "))  
 name = input("Product Name: ")  
 desc = input("Description: ")  
 price = float(input("Price: "))  
 qty = int(input("Quantity: "))  
 ptype = input("Type (Electronics/Clothing): ")  
  
 if ptype == "Electronics":  
 brand = input("Brand: ")  
 warranty = int(input("Warranty (months): "))  
 product = Electronics(pid, name, desc, price, qty, ptype, brand, warranty)  
 elif ptype == "Clothing":  
 size = input("Size: ")  
 color = input("Color: ")  
 product = Clothing(pid, name, desc, price, qty, ptype, size, color)  
 else:  
 print("Invalid product type!")  
 continue  
  
 processor.create\_product(user, product)  
 print("Product created.")  
  
 elif choice == '3':  
 uid = int(input("User ID: "))  
 uname = input("Username: ")  
 pwd = input("Password: ")  
 role = input("Role: ")  
 user = User(uid, uname, pwd, role)  
  
 num\_items = int(input("Number of products in the order: "))  
 products = []  
 for \_ in range(num\_items):  
 pid = int(input("Product ID: "))  
 qty = int(input("Quantity: "))  
 products.append({"product\_id": pid, "quantity": qty})  
  
 processor.create\_order(user, products)  
  
 elif choice == '4':  
 uid = int(input("User ID: "))  
 oid = int(input("Order ID: "))  
 try:  
 processor.cancel\_order(uid, oid)  
 print("Order cancelled.")  
 except Exception as e:  
 print(e)  
  
 elif choice == '5':  
 products = processor.get\_all\_products()  
 for p in products:  
 print(p)  
  
 elif choice == '6':  
 uid = int(input("User ID: "))  
 uname = input("Username: ")  
 pwd = input("Password: ")  
 role = input("Role: ")  
 user = User(uid, uname, pwd, role)  
  
 orders = processor.get\_order\_by\_user(user)  
 for order in orders:  
 print(f"Order ID: {order['order\_id']}")  
 for item in order['items']:  
 print(f" Product ID: {item['product\_id']}, Quantity: {item['quantity']}")  
  
 elif choice == '7':  
 print("Exiting Order Management System.")  
 break  
 else:  
 print("Invalid choice. Try again.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**IMPLEMENTATION OF OPERATIONS:**

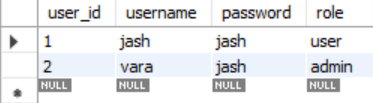
**1.create user /admin**

**User creation :**

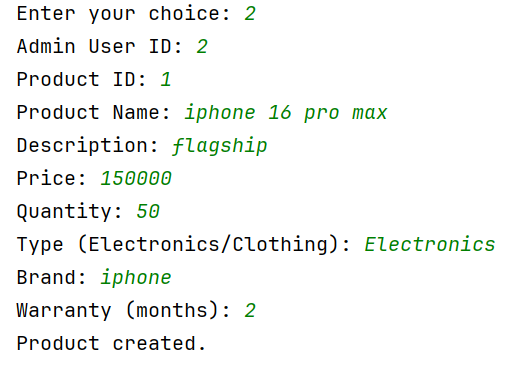
**Admin creation :**

****

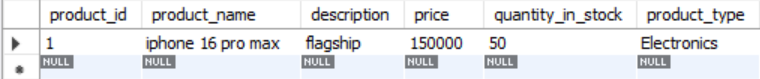
**Output from the database:**

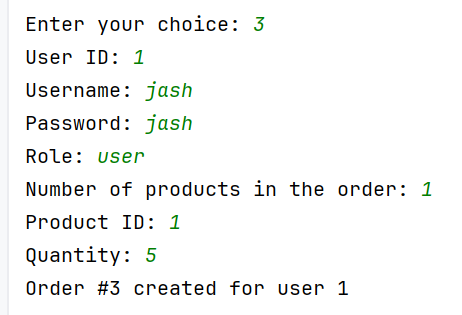
****

**2.create product**

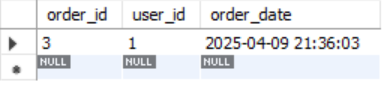
****

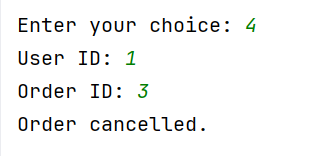
**Output from the database:**

****

**3.create order**

**Output from the database:**

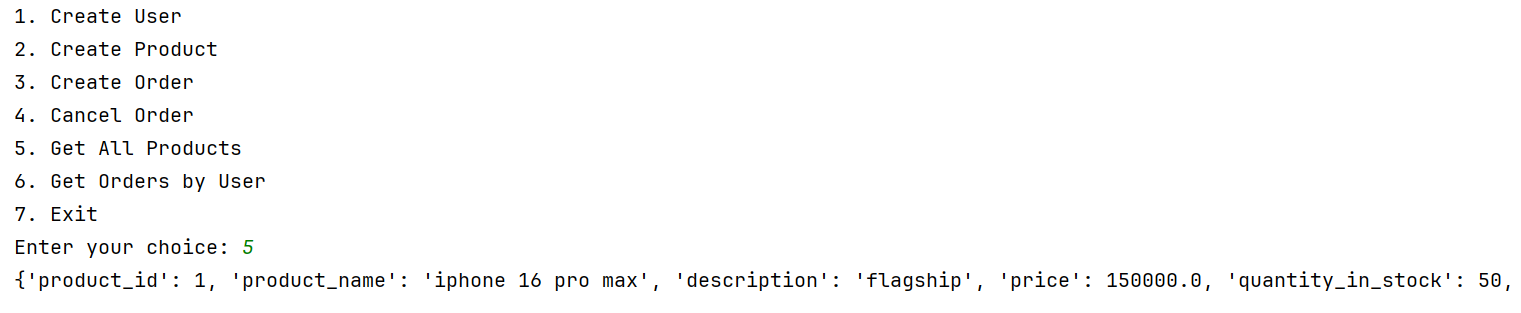
****

**4.Cancel order:**

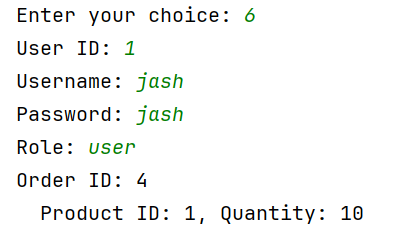
**Output from the database:**

****

**5. get all products**

****

**6. getOrderbyUser**

****

**End of the document**