

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:

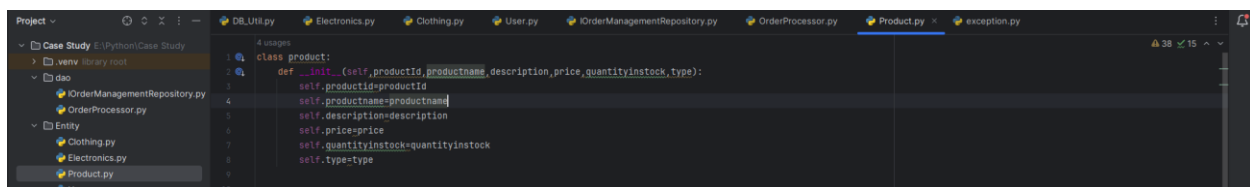
- productId (int)
- productName (String)
- description (String)
- price (double)
- quantityInStock (int)
- type (String) [Electronics/Clothing]

```
mysql> create table Product(  
  -> productID int primary key,  
  -> productName text,  
  -> description text,  
  -> price int,  
  -> QuantityInStock int,  
  -> type enum("Electronics","Clothing"));  
Query OK, 0 rows affected (0.04 sec)
```

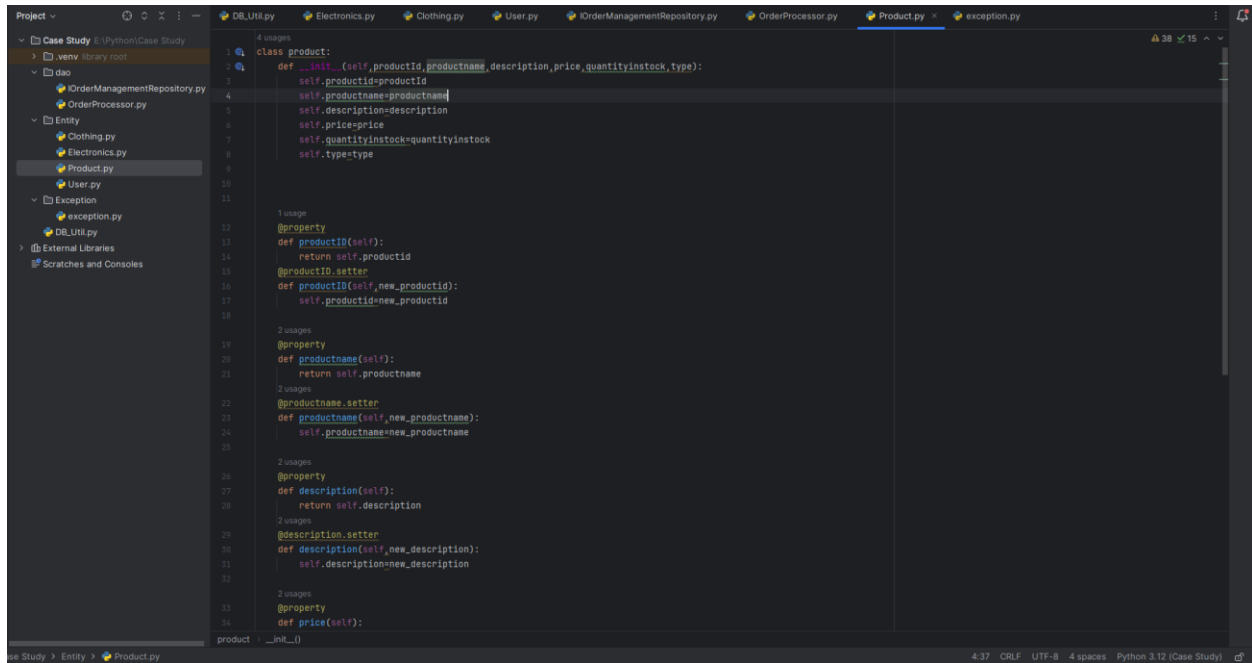
```
mysql> create table user(  
  -> userID int primary key,  
  -> userName text,  
  -> password text,  
  -> role enum("Admin","User"));  
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> create table Orders(  
  -> OrderID int primary key,  
  -> userID int,  
  -> foreign key(userID) references user(userID),  
  -> productID int,  
  -> foreign key(productID) references Product(productID),  
  -> Quantity int,  
  -> TotalAmount int);  
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> create table Orders(  
  -> OrderID int primary key,  
  -> userID int,  
  -> foreign key(userID) references user(userID),  
  -> productID int,  
  -> foreign key(productID) references Product(productID),  
  -> Quantity int,  
  -> TotalAmount int);  
Query OK, 0 rows affected (0.04 sec)
```

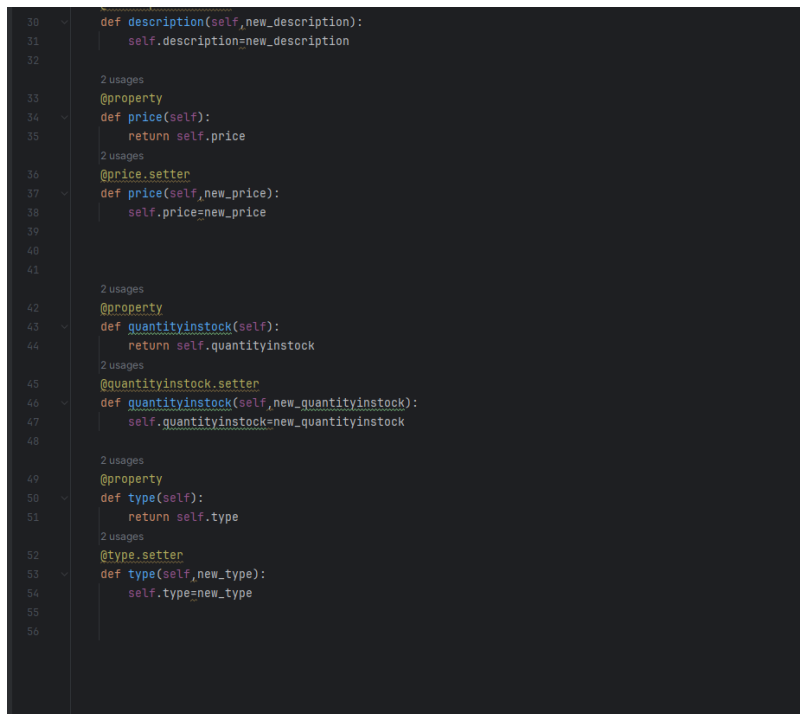


## 2. Implement constructors, getters, and setters for the Product class.



The screenshot shows a code editor with a project structure on the left and a Python file named `Product.py` open. The `Product` class is defined with an `__init__` constructor and several properties for `productid`, `productname`, `description`, `price`, `quantityinstock`, and `type`. Each property has a getter and a setter method. The code is as follows:

```
1 class product:
2     def __init__(self, productid, productname, description, price, quantityinstock, type):
3         self.productid=productid
4         self.productname=productname
5         self.description=description
6         self.price=price
7         self.quantityinstock=quantityinstock
8         self.type=type
9
10
11
12     1 usage
13     @property
14     def productid(self):
15         return self.productid
16     @productid.setter
17     def productid(self, new_productid):
18         self.productid=new_productid
19
20     2 usages
21     @property
22     def productname(self):
23         return self.productname
24     @productname.setter
25     def productname(self, new_productname):
26         self.productname=new_productname
27
28     2 usages
29     @property
30     def description(self):
31         return self.description
32     @description.setter
33     def description(self, new_description):
34         self.description=new_description
35
36     2 usages
37     @property
38     def price(self):
39         return self.price
40     @price.setter
41     def price(self, new_price):
42         self.price=new_price
43
44     2 usages
45     @property
46     def quantityinstock(self):
47         return self.quantityinstock
48     @quantityinstock.setter
49     def quantityinstock(self, new_quantityinstock):
50         self.quantityinstock=new_quantityinstock
51
52     2 usages
53     @property
54     def type(self):
55         return self.type
56     @type.setter
57     def type(self, new_type):
58         self.type=new_type
59
60     def __str__(self):
61         return f'Product(id={self.productid}, name={self.productname}, description={self.description}, price={self.price}, quantityinstock={self.quantityinstock}, type={self.type})'
```



This block continues the implementation of the `Product` class, showing the remaining properties and methods. The code is as follows:

```
30     def description(self, new_description):
31         self.description=new_description
32
33     2 usages
34     @property
35     def price(self):
36         return self.price
37     @price.setter
38     def price(self, new_price):
39         self.price=new_price
40
41
42     2 usages
43     @property
44     def quantityinstock(self):
45         return self.quantityinstock
46     @quantityinstock.setter
47     def quantityinstock(self, new_quantityinstock):
48         self.quantityinstock=new_quantityinstock
49
50     2 usages
51     @property
52     def type(self):
53         return self.type
54     @type.setter
55     def type(self, new_type):
56         self.type=new_type
57
58     def __str__(self):
59         return f'Product(id={self.productid}, name={self.productname}, description={self.description}, price={self.price}, quantityinstock={self.quantityinstock}, type={self.type})'
```

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

- brand (String)
- warrantyPeriod (int)

```
1  from Product import product
2
3  class electronics(product):
4      def __init__(self, productId, brand, warrantyperiod):
5          super().__init__(productId)
6          self.brand=brand
7          self.warrantyperiod=warrantyperiod
8
9
10     2 usages
11     @property
12     def brand(self):
13         return self.brand
14
15     2 usages
16     @brand.setter
17     def brand(self, new_brand):
18         self.brand=new_brand
19
20     2 usages
21     @property
22     def warrantyperiod(self):
23         return self.warrantyperiod
24
25     2 usages
26     @warrantyperiod.setter
27     def warrantyperiod(self, new_warrantyperiod):
28         self.warrantyperiod=new_warrantyperiod
29
30 |
```

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

- size (String)
- color (String)

```
1 from Product import product
2
3
4 class clothing(product):
5     def __init__(self, productId, size, color):
6         super().__init__(productId)
7         self.size = size
8         self.color = color
9
10    2 usages
11    @property
12    def size(self):
13        return self.size
14
15    2 usages
16    @size.setter
17    def size(self, new_size):
18        self.size = new_size
19
20    2 usages
21    @property
22    def color(self):
23        return self.color
24
25    2 usages
26    @color.setter
27    def color(self, new_color):
28        self.color = new_color
29
```

5. Create a User class with attributes:

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

```
1
2 class user:
3     def __init__(self,userid,username,password,role):
4         self.userid=userid
5         self.username=username
6         self.password=password
7         self.role=role
8
9     2 usages
10    @property
11    def userid(self):
12        return self.userid
13
14    2 usages
15    @userid.setter
16    def userid(self, new_userid):
17        self.userid = new_userid
18
19    2 usages
20    @property
21    def username(self):
22        return self.username
23
24    2 usages
25    @username.setter
26    def username(self, new_username):
27        self.username = new_username
28
29    2 usages
30    @property
31    def password(self):
32        return self.password
33
34    2 usages
35    @password.setter
36    def password(self, new_password):
37        self.password = new_password
38
39    2 usages
40    @property
41    def role(self):
```

```
42
43    2 usages
44    @property
45    def role(self):
46        return self.role
47
48    2 usages
49    @role.setter
50    def role(self, new_role):
51        self.role = new_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.

```
1 from OrderProcessor import orderprocessor
2 from DB_Util import DBConnector
3 from Exception.exception import InvalidDataException
4 db_connector = DBConnector(host="localhost", user="root", password="root", port="3306", database="OrderManagementSystem")
5
6
7 1 usage
8 class IOrderManagementRepository():
9     def __init__(self,db_connector):
10         self.db_connector=db_connector
11
12
13 1 usage
14 def create_order(self):
15     userID=input("Enter the User ID : ")
16     productID=input("Enter the Product Id of order you want to purchase : ")
17     Quantity=input("Quantity you want to purchase : ")
18     existing_record=self.get_user_id(userID)
19     if existing_record is None:
20         self.create_user()
21
22     existing_product=self.get_productID(productID)
23     if existing_product is None:
24         self.create_product()
25
26     orderprocess.create_order(userID,productID,Quantity)
27
28
29 1 usage
30 def cancel_order(self):
31     userID=input("Please Enter the user ID : ")
32     orderId=input("Enter the Order ID you want to delete")
33     existing_order=self.get_order_id(userID,orderId)
34     if existing_order:
35         self.delete_order(orderId)
36     else:
37         raise InvalidDataException("This Order ID not associated with this user ID")
38
39
40 2 usages
41 def create_product(self):
42     self.db_connector.open_connection()
43     cur = self.db_connector.connection.cursor()
44     productId=self.get_unique_productId()
```

```
DB_Util.py Electronics.py Clothing.py User.py IOrderManagementRepository.py x OrderProcessor.py Product.py exception.py

36 cur = self.db_connector.connection.cursor()
37 productId=self.get_unique_productId()
38 productName=input("Enter the Product Name : ")
39 description=input("Enter the discription of Product : ")
40 price=input("Enter the price of Product : ")
41 Quantity=int(input("Enter the Quantity in Stock : "))
42 type=input("Enter the type of the Product 1. Electronics 2. Clothing : ")
43 query="Insert into Product values(%,%,%,%,%,%,%)"
44 values=(productId,productName,description,price,Quantity,type,)
45 cur.execute(query,values)
46 self.db_connector.connection.commit()
47
2 usages
48 def create_user(self):
49     self.db_connector.open_connection()
50     cur = self.db_connector.connection.cursor()
51     userID=self.get_unique_userId()
52     username=input("Enter the username : ")
53     password=input("Enter the password : ")
54     roles=input("Enter the Role 1.Admin 2.User")
55     query="Insert into User values(%,%,%,%,%)"
56     values=(userID,username,password,role,)
57     cur.execute(query,values)
58
59
60
1 usage
61 def get_all_Products(self):
62     self.db_connector.open_connection()
63     cur = self.db_connector.connection.cursor()
64     cur.execute("select Productname from Product")
65     record = cur.fetchall()
66     for i in record:
67         print(i[0])
68
69
1 usage
69 def get_allProducts(self):
70     self.db_connector.open_connection()
71     cur = self.db_connector.connection.cursor()
72     cur.execute("select Productname from Product")
73     record = cur.fetchall()
```

```
DB_Util.py Electronics.py Clothing.py User.py IOrderManagementRepository.py x OrderProcessor.py Product.py exception.py

73     record = cur.fetchall()
74     return record[0]
75
1 usage
76 def getOrderByUser(self):
77     self.db_connector.open_connection()
78     cur = self.db_connector.connection.cursor()
79     UserID=input("Enter the user Id For which you want to fetch orders: ")
80     query="Select OrderId from Orders Where UserID=%s"
81     values=(UserID)
82     cur.execute(query,values)
83     record=cur.fetchall()
84     for i in record:
85         print(i[0])
86
87
1 usage
88 def get_user_id(self,userID):
89     self.db_connector.open_connection()
90     cur = self.db_connector.connection.cursor()
91     cur.execute("Select UserID from user Where UserID=%s",(userID,))
92     return cur.fetchone()
93
94
1 usage
95 def get_productID(self,productId):
96     self.db_connector.open_connection()
97     cur = self.db_connector.connection.cursor()
98     cur.execute("Select ProductID from Product Where ProductID=%s",(productId,))
99     return cur.fetchone()
100
101
102 def get_quantity(self,productId):
103     self.db_connector.open_connection()
104     cur = self.db_connector.connection.cursor()
105     cur.execute("Select QuantityInStock From Product Where ProductId =%s",(productId,))
106     record=cur.fetchone()
107     return record[0]
108
109
1 usage
108 def get_order_id(self,UserID,orderId):
109     self.db_connector.open_connection()
```



```

109         self.db_connector.open_connection()
110         cur = self.db_connector.connection.cursor()
111         cur.execute("Select orderId from Orders Where userId=%s and orderId=%s", (UserID, orderId,))
112         return cur.fetchone()
113
114     1 usage
115     def delete_order(self, orderId):
116         self.db_connector.open_connection()
117         cur = self.db_connector.connection.cursor()
118         cur.execute("Delete From Orders Where OrderId=%s", (orderId,))
119         self.db_connector.connection.commit()
120
121     1 usage
122     def get_unique_productId(self):
123         return len(self.get_allProducts())+1
124
125     1 usage
126     def get_unique_userId(self):
127         return len(self.get_allUser())+1
128
129     1 usage
130     def get_alluser(self):
131         self.db_connector.open_connection()
132         cur = self.db_connector.connection.cursor()
133         cur.execute("select userName from User")
134         record=cur.fetchall()
135         return record[0]

```

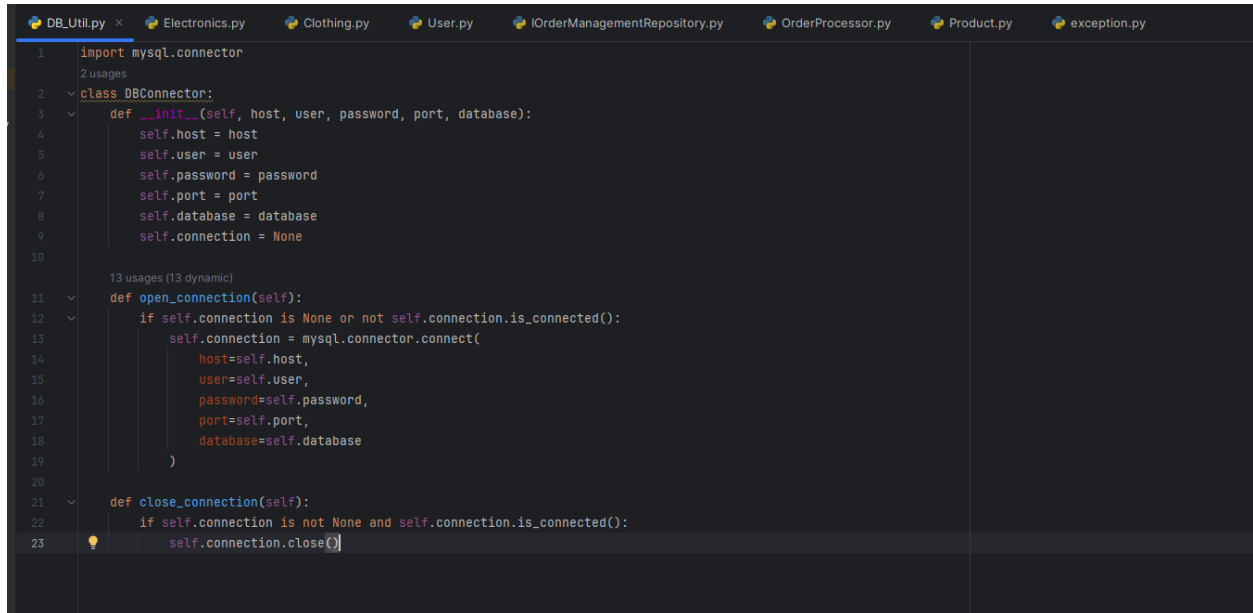
7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.

```
1 from abc import ABC, abstractmethod
2
3 class IOrderManagementRepository(ABC):
4     @abstractmethod
5     def create_order(self, user, products):
6         pass
7
8     @abstractmethod
9     def cancel_order(self, userID, orderId):
10         pass
11
12     @abstractmethod
13     def create_product(self, admin_user, products):
14         pass
15
16     @abstractmethod
17     def create_user(self, user):
18         pass
19
20     @abstractmethod
21     def get_all_products(self):
22         pass
23
24     @abstractmethod
25     def getOrderById(self, user):
26         pass
27
28
29 2 usages
30 class OrderProcessor():
31     def __init__(self, db_connector):
32         self.db_connector=db_connector
33
34     1 usage
35     def create_order(self, userID, productID, Quantity):
36         self.db_connector.open_connection()
37         cur = self.db_connector.connection.cursor()
38         orderId=self.get_unique_orderId()
39         userID=userID
40         productID=productID
41
42 2 usages
43 class OrderProcessor():
44     def __init__(self, db_connector):
45         self.db_connector=db_connector
46
47     1 usage
48     def create_order(self, userID, productID, Quantity):
49         self.db_connector.open_connection()
50         cur = self.db_connector.connection.cursor()
51         orderId=self.get_unique_orderId()
52         userID=userID
53         productID=productID
54         Quantity=Quantity
55         cur.execute("Select Price from Product Where ProductId=%s",(productID,))
56         record=cur.fetchone()
57         TotalAmount=Quantity*record[0]
58         Query="INSERT INTO Orders (orderId, userID, ProductId, TotalAmount, Quantity, OrderDate) VALUES (%s, %s, %s, %s, %s, CURDATE())"
59         values=(orderId, userID, productID, TotalAmount, Quantity,)
60         cur.execute(Query, values)
61
62     1 usage
63     def get_unique_orderId(self):
64         return len(self.get_allorders()) + 1
65
66     1 usage
67     def get_allorders(self):
68         self.db_connector.open_connection()
69         cur = self.db_connector.connection.cursor()
70         cur.execute("Select orderId from Orders")
71         record = cur.fetchall()
72         return record
```

8. Create DBUtil class and add the following method.

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

database Connection



```
1 import mysql.connector
2
3 class DBConnector:
4     def __init__(self, host, user, password, port, database):
5         self.host = host
6         self.user = user
7         self.password = password
8         self.port = port
9         self.database = database
10        self.connection = None
11
12    def open_connection(self):
13        if self.connection is None or not self.connection.is_connected():
14            self.connection = mysql.connector.connect(
15                host=self.host,
16                user=self.user,
17                password=self.password,
18                port=self.port,
19                database=self.database
20            )
21
22    def close_connection(self):
23        if self.connection is not None and self.connection.is_connected():
24            self.connection.close()
```

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"

```
1 usage
def main():
    print("Choose from The below options : ")
    print("1. If you want to create Order")
    print("2. If you want to cancel the order")
    print("3. If you want add a product in the inventory")
    print("4. If you want to Create user ID")
    print("5. If you want to see all the products in the inventory:")
    print("6. If you want to see Completed Orders")
    choice = input("Enter your choice : ")
    if choice == '1':
        order_respositoy.create_order()
    elif choice == '2':
        order_respositoy.cancel_order()
    elif choice == '3':
        order_respositoy.create_product()
    elif choice == '4':
        order_respositoy.create_user()
    elif choice == '5':
        order_respositoy.get_all_Products()
    elif choice == '6':
        order_respositoy.getOrderByUser()
    else:
        print("Enter correct choice")
order_respositoy=IOrderManagementRepository(db_connector)
orderprocessor = orderprocessor(db_connector)

main()
```

OutPut:

```
"E:\Python\Case Study\.venv\Scripts\python.exe" "E:\Python\Case Study\dao\IOrderManagementRepository.py"
Choose from The below options :
1. If you want to create Order
2. If you want to cancel the order
3. If you want add a product in the inventory
4. If you want to Create user ID
5. If you want to see all the products in the inventory:
6. If you want to see Completed Orders
Enter your choice :
```

mysql> select\*from product;

productID	productName	description	price	QuantityInStock	type
1	Smartphone	High-performance smartphone	799	50	Electronics
2	Laptop	Slim and lightweight laptop	1299	30	Electronics
3	T-shirt	Comfortable cotton T-shirt	20	100	Clothing
4	Headphones	Noise-canceling headphones	149	40	Electronics
5	Jeans	Classic blue jeans	50	80	Clothing

5 rows in set (0.00 sec)

mysql>

Project: Case Study E:\Python\Case Study

venv library root

dao

OrderManagementRepository.py

OrderProcessor.py

Entity

Clothing.py

Run: OrderManagementRepository

Process finished with exit code 0

"E:\Python\Case Study\.venv\Scripts\python.exe" "E:\Python\Case Study\dao\IOrderManagementRepository.py"

Choose from The below options :

1. If you want to create Order
2. If you want to cancel the order
3. If you want add a product in the inventory
4. If you want to Create user ID
5. If you want to see all the products in the inventory:
6. If you want to see Completed Orders

Enter your choice : 3

Enter the Product Name : SmartWatch

Enter the discription of Product : Accessories

Enter the price of Product : 950

Enter the Quantity in Stock : 50

Enter the type of the Product 1. Electronics 2. Clothing : Electronics

Process finished with exit code 0

mysql> select\*from product;

productID	productName	description	price	QuantityInStock	type
1	Smartphone	High-performance smartphone	799	50	Electronics
2	Laptop	Slim and lightweight laptop	1299	30	Electronics
3	T-shirt	Comfortable cotton T-shirt	20	100	Clothing
4	Headphones	Noise-canceling headphones	149	40	Electronics
5	Jeans	Classic blue jeans	50	80	Clothing

5 rows in set (0.00 sec)

mysql> select\*from product;

productID	productName	description	price	QuantityInStock	type
1	Smartphone	High-performance smartphone	799	50	Electronics
2	Laptop	Slim and lightweight laptop	1299	30	Electronics
3	T-shirt	Comfortable cotton T-shirt	20	100	Clothing
4	Headphones	Noise-canceling headphones	149	40	Electronics
5	Jeans	Classic blue jeans	50	80	Clothing
6	SmartWatch	Accessories	950	50	Electronics

6 rows in set (0.00 sec)

```
"E:\Python\Case Study\.venv\Scripts\python.exe" "E:\Python\Case Study\dao\IOrderManagementRepository.py"
```

```
Choose from The below options :
```

1. If you want to create Order
2. If you want to cancel the order
3. If you want add a product in the inventory
4. If you want to Create user ID
5. If you want to see all the products in the inventory:
6. If you want to see Completed Orders

```
Enter your choice : 4
```

```
Enter the username : manas
```

```
Enter the password : manas
```

```
Enter the Role 1.Admin 2.UserAdmin
```

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

```
|
```

```
mysql> select*from user;
+----+-----+-----+-----+
| userID | username | password | role |
+----+-----+-----+-----+
| 1 | admin1 | admin_password | Admin |
| 2 | user1 | user_password1 | User |
| 3 | user2 | user_password2 | User |
| 4 | user3 | user_password3 | User |
| 5 | user4 | user_password4 | User |
+----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select*from user;
+----+-----+-----+-----+
| userID | username | password | role |
+----+-----+-----+-----+
| 1 | admin1 | admin_password | Admin |
| 2 | user1 | user_password1 | User |
| 3 | user2 | user_password2 | User |
| 4 | user3 | user_password3 | User |
| 5 | user4 | user_password4 | User |
| 6 | manas | manas | User |
+----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>
```

Project Explorer shows a Python project structure with files: Case Study, dao, OrderProcessor.py, Clothing.py, User.py, and IOrderManagementRepository.py. The IOrderManagementRepository.py file contains the following code:

```
values=(userID,username,password_role,)
cur.execute(query,values)
self.db_connector.connection.commit()
```

The Run console shows the execution of the IOrderManagementRepository.py file, displaying the following output:

```
"E:\Python\Case Study\.venv\Scripts\python.exe" "E:\Python\Case Study\dao\IOrderManagementRepository.py"
Choose from The below options :
1. If you want to create Order
2. If you want to cancel the order
3. If you want add a product in the inventory
4. If you want to Create user ID
5. If you want to see all the products in the inventory:
6. If you want to see Completed Orders
Enter your choice : 4
Enter the username : manas
Enter the password : manas
Enter the Role 1.Admin 2.User : User
Process finished with exit code 0
```

```
"E:\Python\Case Study\.venv\Scripts\python.exe" "E:\Python\Case Study\dao\IOrderManagementRepository.py"
Choose from The below options :
1. If you want to create Order
2. If you want to cancel the order
3. If you want add a product in the inventory
4. If you want to Create user ID
5. If you want to see all the products in the inventory:
6. If you want to see Completed Orders
Enter your choice : 1
Enter the User ID : 2
Enter the Product Id of order you want to purchase : 2
Quantity you want to purchase : 4
```

```
mysql> select* from orders
-> ;
+-----+-----+-----+-----+-----+-----+
| OrderID | userID | productID | Quantity | TotalAmount | OrderDate |
+-----+-----+-----+-----+-----+-----+
| 1 | 2 | 3 | 4 | 80 | 2024-02-06 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```