

Assignment 1

Techshop

Task1&2:

```
275 class Customer:
276     def __init__(self, customer_id, first_name, last_name, email, phone, address):
277         self.customer_id = customer_id
278         self.first_name = first_name
279         self.last_name = last_name
280         self.email = email
281         self.phone = phone
282         self.address = address
283         self.orders = [] # Assuming orders will be stored as a list
284
285     def calculate_total_orders(self):
286         return len(self.orders)
287
288     def get_customer_details(self):
289         print("Customer ID:", self.customer_id)
290         print("Name:", self.first_name, self.last_name)
291         print("Email:", self.email)
292         print("Phone:", self.phone)
293         print("Address:", self.address)
294         print("Total Orders:", self.calculate_total_orders())
295
296     def update_customer_info(self, new_email=None, new_phone=None, new_address=None):
297         if new_email:
298             self.email = new_email
299         if new_phone:
300             self.phone = new_phone
301         if new_address:
302             self.address = new_address
303         print("Customer information updated successfully.")
304
305     @property
306     def customer_id(self):
307         return self.customer_id
308
```

```
Task1_class Implementation.py
Task1_class Implementation.py > Products
209
210 @property
211 def first_name(self):
212     return self._first_name
213
214 @property
215 def last_name(self):
216     return self._last_name
217
218 @property
219 def email(self):
220     return self._email
221
222 @property
223 def phone(self):
224     return self._phone
225
226 @property
227 def address(self):
228     return self._address
229
230 @property
231 def orders(self):
232     return self._orders
233
234 class Product:
235     def __init__(self, product_id, product_name, description, price):
236         self.product_id = product_id
237         self.product_name = product_name
238         self.description = description
239         self._price = 0.0
240         self.price = price
241
```

```
Task1_class_implementation.py
Task1_class_implementation.py
344     def get_product_details(self):
345         print("Product ID:", self._product_id)
346         print("Product Name:", self._product_name)
347         print("Description:", self._description)
348         print("Price: $%.2f" % self._price)
349
350     def update_product_info(self, new_price=None, new_description=None):
351         if new_price is not None:
352             self._price = new_price
353         if new_description:
354             self._description = new_description
355         print("Product information updated successfully.")
356
357     @property
358     def product_id(self):
359         return self._product_id
360
361     @property
362     def product_name(self):
363         return self._product_name
364
365     @property
366     def description(self):
367         return self._description
368
369     @property
370     def price(self):
371         return self._price
372
```

```
Task1_class_implementation.py
Task1_class_implementation.py > Orders > process_order
373 class Orders:
374     def __init__(self, order_id, customer, order_date, total_amount):
375         self._order_id = order_id
376         self._customer = customer # Composition relationship with Customer class
377         self._order_date = order_date
378         self._total_amount = total_amount
379         self._order_status = "Pending"
380         self._order_details = [] # assuming order details will be stored as a list
381
382     def calculate_total_amount(self):
383         return sum(detail.calculate_subtotal() for detail in self._order_details)
384
385     def get_order_details(self):
386         print("Order ID:", self._order_id)
387         print("Customer:", self._customer.first_name, self._customer.last_name)
388         print("Order Date:", self._order_date.strftime("%Y-%m-%d %H:%M:%S"))
389         print("Total Amount: $%.2f" % self.calculate_total_amount())
390         print("Order Status:", self._order_status)
391
392     def update_order_status(self, new_status):
393         self._order_status = new_status
394         print("Order status updated successfully.")
395
396     def cancel_order(self):
397         self.update_order_status("Cancelled")
398         print("Order cancelled.")
399
400     def process_order(self):
401         try:
402             self.validate_order()
403             self.update_inventory()
404             self.process_payment()
405             print("Order processed successfully.")
406         except (InvalidDataException, InsufficientStockException, IncompleteOrderException,
407               PaymentFailedException, Exception) as e:
408             print(f"Error processing order: {e}")
409
```

```
Task1_class_implementation.py
Task1_class_implementation.py > Orders > simulate_payment_processing

411 def validate_order(self):
412     # Validation logic for order details
413     if not self.order_details:
414         raise IncompleteOrderException("Order is incomplete. No order details found.")
415     for detail in self.order_details:
416         if not detail.product:
417             raise IncompleteOrderException("Order is incomplete. Product reference missing.")
418
419 def update_inventory(self):
420     # Inventory management logic
421     for detail in self.order_details:
422         if detail.quantity > detail.product.quantity_in_stock:
423             raise InsufficientStockException("Insufficient stock for product: "
424                                             f"{detail.product.product_name}")
425
426 def process_payment(self):
427     # Payment processing logic
428     try:
429         # Simulate payment processing
430         payment_result = self.simulate_payment_processing()
431
432         if payment_result == "success":
433             print("Payment successful.")
434         else:
435             raise PaymentFailedException("Payment declined.")
436     except PaymentFailedException as e:
437         # Retry or cancel order
438         raise e
439
440 def simulate_payment_processing(self):
441
442     import random
443     if random.random() < 0.8:
444         return "success"
445     else:
446         return "failure"
```

```
Task1_class_implementation.py
Task1_class_implementation.py > Orders > simulate_payment_processing

446
447 @property
448 def order_id(self):
449     return self._order_id
450
451 @property
452 def customer(self):
453     return self._customer
454
455 @property
456 def order_date(self):
457     return self._order_date
458
459 @property
460 def total_amount(self):
461     return self._total_amount
462
463 @property
464 def order_status(self):
465     return self._order_status
466
467 @property
468 def order_details(self):
469     return self._order_details
470
471
472 class OrderDetails:
473     def __init__(self, order_detail_id, order, product, quantity):
474         self.order_detail_id = order_detail_id
475         self.order = order # Composition relationship with Orders class
476         self.product = product # Composition relationship with Products class
477         self.quantity = quantity
478         self.discount = 0.0
479
```

```
Task1_class_implementation.py
Task1_class_implementation.py > Orders > simulate_payment_processing
488     def calculate_subtotal(self):
489         return (self._product.price - self._discount) * self._quantity
490
491     def get_order_detail_info(self):
492         print("Order Detail ID:", self._order_detail_id)
493         print("Product:", self._product.product_name)
494         print("Quantity:", self._quantity)
495         print("Subtotal: $%.2f" % self.calculate_subtotal())
496
497     def update_quantity(self, new_quantity):
498         self._quantity = new_quantity
499         print("Quantity updated successfully.")
500
501     def add_discount(self, discount_amount):
502         self._discount += discount_amount
503         print("Discount applied successfully.")
504
505     @property
506     def order_detail_id(self):
507         return self._order_detail_id
508
509     @property
510     def order(self):
511         return self._order
512
513     @property
514     def product(self):
515         return self._product
516
517     @property
518     def quantity(self):
519         return self._quantity
520
521     @property
522     def discount(self):
523         return self._discount
```

```
Task1_class_implementation.py
Task1_class_implementation.py > _
524
525
526 class Inventory:
527     def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
528         self._inventory_id = inventory_id
529         self._product = product
530         self._quantity_in_stock = 0 # Default to 0 to ensure positivity
531         self._quantity_in_stock = quantity_in_stock # use the setter to apply validation
532         self._last_stock_update = last_stock_update
533
534     @property
535     def inventory_id(self):
536         return self._inventory_id
537
538     @property
539     def product(self):
540         return self._product
541
542     @property
543     def quantity_in_stock(self):
544         return self._quantity_in_stock
545
546     @quantity_in_stock.setter
547     def quantity_in_stock(self, new_quantity):
548         if not isinstance(new_quantity, int):
549             raise ValueError("Quantity must be an integer.")
550         if new_quantity < 0:
551             raise ValueError("Quantity cannot be negative.")
552         self._quantity_in_stock = new_quantity
553
554     @property
555     def last_stock_update(self):
556         return self._last_stock_update
557
```

Implementation of methods:

Task(4,5,6):

```
customers.py •
customers.py > Customers > calculate_total_orders

25 import mysql.connector
26
27 db_config = {
28     "host": "localhost",
29     "user": "root",
30     "password": "root",
31     "database": "techshop"
32 }
33
34 conn = mysql.connector.connect(**db_config)
35 cursor = conn.cursor()
36
37 class InvalidDataException(Exception):
38     def __init__(self, message="Invalid data provided."):
39         self.message = message
40         super().__init__(self.message)
41
42 # Customers Class Queries
43 class Customers:
44     def __init__(self, email):
45         self._validate_email(email)
46
47     def _validate_email(self, email):
48         # Custom email validation logic
49         if "@" not in email or "." not in email:
50             raise InvalidDataException("Invalid email address provided.")
51
52     def calculate_total_orders(self, customer_id):
53         query = """
54             SELECT COUNT(ORDERID) AS TotalOrders
55             FROM Orders
56             WHERE CustomerID = %s;
57         """
58         cursor.execute(query, (customer_id,))
59         result = cursor.fetchone()
60         return result[0] if result else None
```

```
customers.py •
customers.py 2 -

62 def get_customer_details(self, customer_id):
63     query = """
64         SELECT *
65         FROM Customers
66         WHERE CustomerID = %s;
67     """
68     cursor.execute(query, (customer_id,))
69     result = cursor.fetchone()
70     return result
71
72 def update_customer_info(self, customer_id, email, phone, address):
73
74     query = """
75         UPDATE Customers
76         SET Email = %s, Phone = %s, Address = %s
77         WHERE CustomerID = %s;
78     """
79     cursor.execute(query, (email, phone, address, customer_id))
80     conn.commit()
81
82 try:
83     #to create a customer with an invalid email
84     invalid_customer = Customers("invalid-email")
85 except InvalidDataException as e:
86     print(f"Error: {e}")
87
```



```
products.py X
products.py > ...
1 > ...
19 import mysql.connector
20
21 db_config = {
22     "host": "localhost",
23     "user": "root",
24     "password": "root",
25     "database": "techshop"
26 }
27
28 # Connect to MySQL
29 conn = mysql.connector.connect(**db_config)
30 cursor = conn.cursor()
31
32 class DuplicateProductException(Exception):
33     def __init__(self, message="Duplicate product found,"):
34         self.message = message
35         super().__init__(self.message)
36
37
38 class Products:
39     def get_product_details(self, product_id):
40         query = """
41             SELECT *
42             FROM Products
43             WHERE ProductID = %s;
44         """
45         cursor.execute(query, (product_id,))
46         result = cursor.fetchone()
47         return result
48
49     def update_product_info(self, product_id, price, description):
50         query = """
51             UPDATE Products
52             SET Price = %s, Description = %s
53             WHERE ProductID = %s;
54         """
```

```
products.py
products.py > Products > add product
55 cursor.execute(query, (price, description, product_id))
56 conn.commit()
57
58 def is_product_in_stock(self, product_id):
59     query = """
60         SELECT QuantityInStock
61         FROM Inventory
62         WHERE ProductID = %s;
63     """
64     cursor.execute(query, (product_id,))
65     result = cursor.fetchone()
66     return result[0] if result else None
67
68 def search_products(self, search_criteria):
69     try:
70         query = """
71             SELECT *
72             FROM Products
73             WHERE Category LIKE %s;
74         """
75         cursor.execute(query, ("%s(%s)" % (search_criteria, "%")))
76         result = cursor.fetchall()
77         return result
78     except Exception as e:
79         print(f"Error: {e}")
80
81 def add_product(self, product):
82     try:
83         # Check for duplicate products based on name or SKU
84         if any(p['Productname'].lower() == product['Productname'].lower() or
85              p['SKU'].lower() == product['SKU'].lower() for p in self.products):
86             raise DuplicateProductException()
87
88         self.products.append(product)
89         print("Product added successfully.")
90     except DuplicateProductException as e:
91         print(f"Error: {e}")
```

```
orders.py
orders.py > Orders > _init_
1 from datetime import datetime
2 import mysql.connector
3
4 db_config = {
5     "host": "localhost",
6     "user": "root",
7     "password": "root",
8     "database": "techshop"
9 }
10
11 # Connect to MySQL
12 conn = mysql.connector.connect(**db_config)
13 cursor = conn.cursor()
14
15 from datetime import datetime
16 class InvalidDataException(Exception):
17     def __init__(self, message="Invalid data provided."):
18         self.message = message
19         super().__init__(self.message)
20
21 class InsufficientStockException(Exception):
22     def __init__(self, message="Insufficient stock."):
23         self.message = message
24         super().__init__(self.message)
25
26
27 class Orders:
28     def __init__(self, order_id, customer_id, order_date, total_amount, status="Pending"):
29         self.order_id = order_id
30         self.customer_id = customer_id
31         self.order_date = order_date
32         self.total_amount = total_amount
33         self.status = status
34
```

```
orders.py
orders.py > Orders > _init_
35 def calculate_total_amount(self):
36     query = """
37         SELECT SUM(totalAmount) AS totalAmount
38         FROM Orders
39         WHERE CustomerID = %s;
40     """
41     cursor.execute(query, (self.CustomerID,))
42     result = cursor.fetchone()
43     return result[0] if result else None
44
45 def get_order_details(self):
46     query = """
47         SELECT *
48         FROM OrderDetails
49         WHERE OrderID = %s;
50     """
51     cursor.execute(query, (self.OrderID,))
52     result = cursor.fetchall()
53     return result
54
55 def update_order_status(self, new_status):
56     try:
57         self.validate_order_status(new_status)
58         query = """
59             UPDATE Orders
60             SET status = %s
61             WHERE OrderID = %s; (parameter) new_status: Any
62         """
63         cursor.execute(query, (new_status, self.OrderID))
64         conn.commit()
65         print(f"Order {self.order_id} status updated to {new_status}.")
66     except InvalidDataException as e:
67         print(f"Error: {e}")
68
```

```
orders.py •
orders.py > Orders > add_new_order
109 def cancel_order(self):
110     query = """
111     DELETE FROM Orders
112     WHERE OrderID = %s;
113     """
114     cursor.execute(query, (self.OrderID,))
115     conn.commit()
116     print(f"Order {self.OrderID} canceled.")
117
118 def add_new_order(self, product_id, quantity):
119     try:
120         # Check if the product is available in inventory
121         if not self.is_product_available(product_id, quantity):
122             raise InvalidDataException("Product not available in sufficient quantity.")
123
124         # Insert new order
125         oi = self.CustomerID
126         order_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
127         query_insert_order = """
128         INSERT INTO Orders (orderid, customerID, OrderDate, totalAmount)
129         VALUES (%s, %s, %s, %s);
130         """
131         cursor.execute(query_insert_order, (oi, self.CustomerID, order_date, 0))
132         conn.commit()
133
134         # Get the newly created order ID
135         query_get_new_order_id = "SELECT LAST_INSERT_ID();"
136         cursor.execute(query_get_new_order_id)
137         new_order_id = cursor.fetchone()[0]
138
139         # Insert order details
140         query_insert_order_details = """
141         INSERT INTO OrderDetails (orderid, ProductID, Quantity)
142         VALUES (%s, %s, %s);
143         """
144         cursor.execute(query_insert_order_details, (new_order_id, product_id, quantity))
145         conn.commit()
```

```
orders.py •
orders.py > Orders > add_new_order
111 def _validate_order_status(self, status):
112     valid_statuses = ['Pending', 'Processing', 'Shipped', 'Delivered', 'Canceled']
113     if status not in valid_statuses:
114         raise InvalidDataException("Invalid order status provided.")
115
116 def _is_product_available(self, product_id, quantity):
117     query = """
118     SELECT CASE WHEN QuantityInStock >= %s THEN 1 ELSE 0 END AS IsAvailable
119     FROM Inventory
120     WHERE ProductID = %s;
121     """
122     cursor.execute(query, (quantity, product_id))
123     result = cursor.fetchone()
124     return result[0] if result else None
125
126 #collections:handling inventory management
127 def process_order(self):
128     try:
129         # Get order details
130         order_details = self.get_order_details()
131
132         # Update inventory quantities and calculate total amount
133         total_amount = 0
134         for order_detail in order_details:
135             product_id = order_detail['ProductID']
136             quantity = order_detail['Quantity']
137
138             # Check if the product is available in inventory
139             if not self._is_product_available(product_id, quantity):
140                 raise InsufficientStockException(f"Insufficient stock for product {product_id}.")
141
142             # Update inventory quantity
143             self._update_inventory(product_id, quantity)
144
145             # Calculate total amount for the order
146             product_price = order_detail['Price']
147             total_amount += quantity * product_price
```



```

148
149     # Update total amount in the Orders table
150     query_update_total_amount = """
151     UPDATE Orders
152     SET TotalAmount = %s
153     WHERE OrderID = %s;
154     """
155     cursor.execute(query_update_total_amount, (total_amount, self.OrderID))
156     conn.commit()
157
158     print(f"Order {self.OrderID} processed successfully. Total Amount: {total_amount}")
159 except InsufficientStockException as e:
160     print(f"Error: {e}")
161     # Optionally, you can cancel the order or take other actions based on business logic
162
163 def update_inventory(self, product_id, quantity):
164     query_update_inventory = """
165     UPDATE Inventory
166     SET QuantityInStock = QuantityInStock - %s
167     WHERE ProductID = %s;
168     """
169     cursor.execute(query_update_inventory, (quantity, product_id))
170     conn.commit()

```

```

orderdetails.py
orderdetails.py > OrderDetails
1 import mysql.connector
2 db_config = {
3     "host": "localhost",
4     "user": "root",
5     "password": "root",
6     "database": "techshop"
7 }
8
9 # Connect to MySQL
10 conn = mysql.connector.connect(**db_config)
11 cursor = conn.cursor()
12 # OrderDetails Class Queries
13 class OrderDetails:
14     def calculate_subtotal(self, order_detail_id):
15         query = """
16         SELECT Quantity * Price AS Subtotal
17         FROM OrderDetails
18         JOIN Products ON OrderDetails.ProductID = Products.ProductID
19         WHERE OrderDetailID = %s;
20         """
21         cursor.execute(query, (order_detail_id,))
22         result = cursor.fetchone()
23         return result[0] if result else None
24
25     def get_order_detail_info(self, order_detail_id):
26         query = """
27         SELECT *
28         FROM OrderDetails od
29         LEFT JOIN Orders o ON o.orderid = od.orderid
30         LEFT JOIN Customers c ON c.customerid = o.customerid
31         WHERE od.OrderDetailID = %s;
32         """
33         cursor.execute(query, (order_detail_id,))
34         result = cursor.fetchone()
35         return result
36

```

```

36
37 def update_quantity(self, order_detail_id, quantity):
38     query = """
39     UPDATE OrderDetails
40     SET Quantity = %s
41     WHERE OrderDetailID = %s;
42     """
43     cursor.execute(query, (quantity, order_detail_id))
44     conn.commit()
45
46 def add_discount(self, order_detail_id, discount):
47     query = """
48     UPDATE Orders o
49     LEFT JOIN orderdetails od ON od.orderid=o.orderid
50     SET TotalAmount = %s
51     WHERE OrderDetailID = %s;
52     """
53     cursor.execute(query, (discount, order_detail_id))
54     conn.commit()

```

```
inventory.py X
inventory.py > Inventory > get_product
1 import datetime
2 import mysql.connector
3
4 db_config = {
5     "host": "localhost",
6     "user": "root",
7     "password": "root",
8     "database": "techshop"
9 }
10
11 # Connect to MySQL
12 conn = mysql.connector.connect(**db_config)
13 cursor = conn.cursor()
14 # Inventory class Queries
15 class Inventory:
16     def get_product(self, product_id):
17         query = """
18             SELECT *
19             FROM Products
20             WHERE ProductID = %s;
21         """
22         cursor.execute(query, (product_id,))
23         result = cursor.fetchone()
24         return result
25
26     def get_quantity_in_stock(self, product_id):
27         query = """
28             SELECT QuantityInStock
29             FROM Inventory
30             WHERE ProductID = %s;
31         """
32         cursor.execute(query, (product_id,))
33         result = cursor.fetchone()
34         return result[0] if result else None
35
```

```
inventory.py X
inventory.py > Inventory > get_product
36
37     def add_to_inventory(self, product_id, quantity):
38         query = """
39             UPDATE Inventory
40             SET QuantityInStock = QuantityInStock + %s
41             WHERE ProductID = %s;
42         """
43         cursor.execute(query, (quantity, product_id))
44         conn.commit()
45
46     def remove_from_inventory(self, product_id, quantity):
47         query = """
48             UPDATE Inventory
49             SET QuantityInStock = QuantityInStock - %s
50             WHERE ProductID = %s;
51         """
52         cursor.execute(query, (quantity, product_id))
53         conn.commit()
54
55     def update_stock_quantity(self, product_id, new_quantity):
56         query = """
57             UPDATE Inventory
58             SET QuantityInStock = %s
59             WHERE ProductID = %s;
60         """
61         cursor.execute(query, (new_quantity, product_id))
62         conn.commit()
63
64     def is_product_available(self, product_id, quantity_to_check):
65         query = """
66             SELECT CASE WHEN QuantityInStock >= %s THEN 1 ELSE 0 END AS IsAvailable
67             FROM Inventory
68             WHERE ProductID = %s;
69         """
70         cursor.execute(query, (quantity_to_check, product_id))
71         result = cursor.fetchone()
72         return result[0] if result else None

```

```
inventory.py X
inventory.py > inventory > get_product
73 def get_inventory_value(self):
74     query = """
75         SELECT SUM(QuantityInStock * Price) AS InventoryValue
76         FROM Inventory
77         JOIN Products ON Inventory.ProductID = Products.ProductID;
78     """
79     cursor.execute(query)
80     result = cursor.fetchone()
81     return result[0] if result else None
82
83 def list_low_stock_products(self, threshold):
84     query = """
85         SELECT *
86         FROM Inventory
87         JOIN Products ON Inventory.ProductID = Products.ProductID
88         WHERE QuantityInStock < %s;
89     """
90     cursor.execute(query, (threshold,))
91     result = cursor.fetchall()
92     return result
93
94 def list_out_of_stock_products(self):
95     query = """
96         SELECT *
97         FROM Inventory
98         JOIN Products ON Inventory.ProductID = Products.ProductID
99         WHERE QuantityInStock = 0;
100     """
101     cursor.execute(query)
102     result = cursor.fetchall()
103     return result
104
105 def list_all_products(self):
106     query = """
107         SELECT *
108         FROM Inventory
109         JOIN Products ON Inventory.ProductID = Products.ProductID;
```

```
104
105 def list_all_products(self):
106     query = """
107         SELECT *
108         FROM Inventory
109         JOIN Products ON Inventory.ProductID = Products.ProductID;
110     """
111     cursor.execute(query)
112     result = cursor.fetchall()
113     return result
```

Input Methods:

```

65
66 customer_handler = Customers('john@example.com')
67 total_orders = customer_handler.calculate_total_orders(1)
68 print("Total orders:",total_orders)
69
70 customer_details = customer_handler.get_customer_details(1)
71 print("Customer detail",customer_details)
72
73 customer_handler.update_customer_info(1, 'newemail@gmail.com', 'newphone101', 'newaddress101')
74 print('information updated')
75
76 # Commit the changes
77 conn.commit()
78

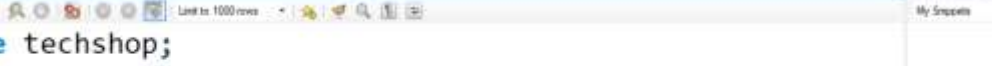
```

```

PS D:\Hexaware\Assignment\Assignment 1\Python > "C:/Program Files/Python311/python.exe" "d:\Hexaware\Assignment\Assignment 1\Python\customers.py"
Error: Invalid email address provided.
Total order: 7
Customer detail (1, 'Rohan', 'Chaudhary', 'newemail@gmail.com', 'newphone11', 'newaddress11', '9')
Information updated
PS D:\Hexaware\Assignment\Assignment 1\Python >

```

DB changes:



The screenshot shows a SQL client window titled "SQL file 2". The query editor contains the following SQL commands:

```
1. use techshop;
2. show tables;
3. select * from customers;
4.
```

The results grid at the bottom displays the output of the last query, showing a table with columns: CustomerID, FirstName, LastName, Email, Phone, Address, and OrderNumbers. The first row of data is:

CustomerID	FirstName	LastName	Email	Phone	Address	OrderNumbers
1	Robert	Chauhan	robert41@gmail.com	newyork101	newyork101	0

INPUT METHODS:

```
30 product_handler = ProductHandler()
31 product_details = product_handler.get_product_details(1)
32 print(product_details)
33 product_handler.update_product_info(1, 1199.99, 'Updated new description')
34 print('updated')
35 stock_status = product_handler.is_product_in_stock(1)
36 print(stock_status)
37
38 # Search for products with a specific criteria
39 search_result = product_handler.search_products("Electronic Gadget")
40
41 # Display search results
42 for product in search_result:
43     print(product)
44
45 conn.commit()
46
47 # Close the database connection
48 cursor.close()
49 conn.close()
```

```
PS D:\Hexaware\Assignment\Assignment 1\Python & "C:\Program Files\Python112\python.exe" "d:\Hexaware\Assignment\Assignment 1\Python\products.py"
(1, 'External Hard Drive', 'Updated new description', Decimal('1199.99'), 'Electronic Gadget')
updated
20
(1, 'External Hard Drive', 'Updated new description', Decimal('1199.99'), 'Electronic Gadget')
(2, 'Wireless Router', 'dual-band router', Decimal('133.89'), 'Electronic Gadget')
(3, 'Tablet', '10-inch tablet with long battery', Decimal('219.99'), 'Electronic Gadget')
(4, 'Printer', 'Wireless all-in-one printer', Decimal('225.59'), 'Electronic Gadget')
(5, 'Smartwatch', 'Fitness tracker having heart rate monitor', Decimal('164.99'), 'Electronic Gadget')
(6, 'Laptop', 'High-performance laptop', Decimal('1099.99'), 'Electronic Gadget')
(7, 'Headphones', 'headphones with bluetooth', Decimal('187.99'), 'Electronic Gadget')
(8, 'Camera', 'Digital camera with 20MP', Decimal('552.19'), 'Electronic Gadget')
(9, 'Desktop Computer', 'Powerful desktop', Decimal('1539.99'), 'Electronic Gadget')
(10, 'Smartphone', 'Latest model with dual cameras', Decimal('879.99'), 'Electronic Gadget')
(11, 'New Electronic Gadget', 'Description of the new gadget', Decimal('399.99'), 'Electronic Gadget')
PS D:\Hexaware\Assignment\Assignment 1\Python
```

DB Changes:

SQL File 2

1. use techshop;

2. show tables;

3. select * from products;

4

ProductID	ProductName	Description	Price	Category
1	External Hard Drive	Updated new description	1299.99	Electronic Gadget
2	Wireless Router	dual-band router	133.89	Electronic Gadget
3	Tablet	10-inch tablet with long battery	219.99	Electronic Gadget
4	Printer	Wireless all-in-one printer	225.59	Electronic Gadget
5	Smartwatch	Fitness tracker having heart rate monitor	164.99	Electronic Gadget
6	Laptop	High-performance laptop	1099.99	Electronic Gadget
7	Headphones	headphones with bluetooth	187.99	Electronic Gadget
8	Camera	Digital camera with 20MP	552.19	Electronic Gadget
9	Desktop Computer	Powerful desktop	1539.99	Electronic Gadget
10	Smartphone	Latest model with dual cameras	879.99	Electronic Gadget
11	New Electronic Ga...	Description of the new gadget	399.99	Electronic Gadget

Input Methods:

```
171
172 order_handler = Orders(181, 1, datetime.now(), 0)
173
174 # Add a new order
175 order_handler.add_new_order(product_id=11, quantity=1)
176
177 # Get order details
178 order_details = order_handler.get_order_details()
179
180 # Update order status
181 order_handler.update_order_status(new_status='Shipped')
182
183 # Cancel order
184 order_handler.cancel_order()
185 print('Order Cancelled')
186
187 order_handler.process_order()
188 print('Order Processed')
189 conn.commit()
190
191 cursor.close()
192 conn.close()
```

```
PS D:\Hexaware\Assignment\Assignment 1\Python> "C:\Program Files\Python112\python.exe" "D:\Hexaware\Assignment\Assignment 1\Python/orders.py"
Error: Product not available in sufficient quantity.
Order 181 status updated to Shipped.
Order 181 canceled.
Order Cancelled
Order 181 processed successfully. Total Amount: 0
Order Processed
PS D:\Hexaware\Assignment\Assignment 1\Python> |
```

DB changes:



The screenshot shows a SQL File 2 window with the following queries:

1. use techshop;
2. show tables;
3. select * from Orders;
- 4.

The result grid displays the following data:

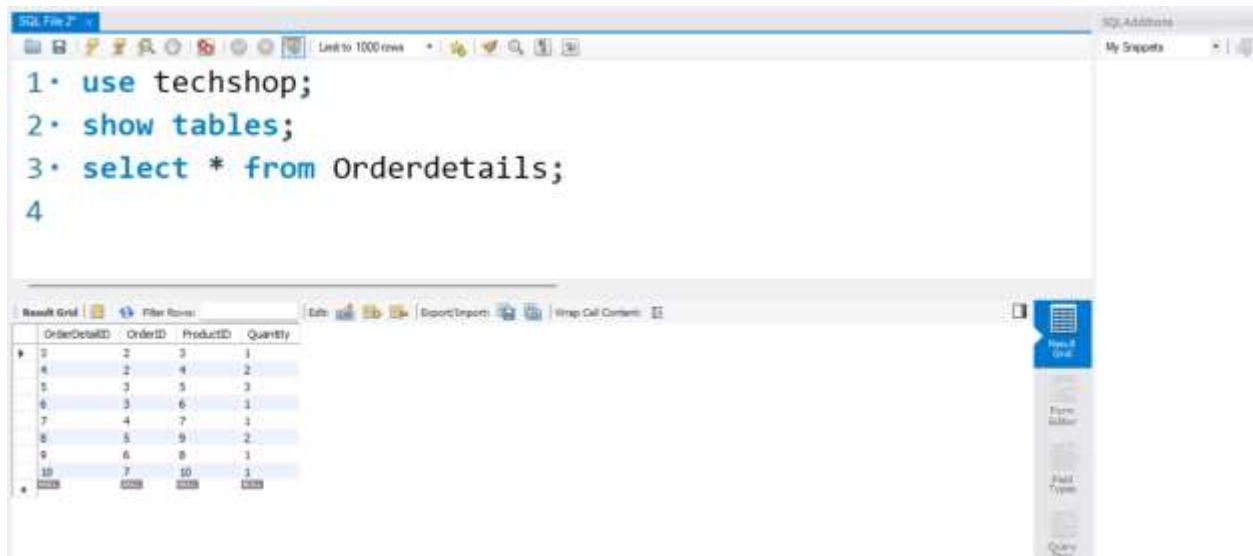
OrderID	CustomerID	OrderDate	TotalAmount	Status
1	1	2024-01-01	2000.00	Shipped
2	2	2024-02-01	621.27	Shipped
3	3	2024-03-01	1594.96	Shipped

Input methods:

```
55
56 order_details_handler = OrderDetails()
57 subtotal_amount = order_details_handler.calculate_subtotal(1)
58 print(subtotal_amount)
59
60 order_detail_info = order_details_handler.get_order_detail_info(1)
61 print(order_detail_info)
62
63 order_details_handler.update_quantity(1, 5)
64 print('updated')
65
66 order_details_handler.add_discount(1, 10)
67 print('Information updated')
68
69 conn.commit()
70
```

```
PS D:\Vivekware\Assignment\Assignment 1\Python> & "C://Program Files/Python32/python.exe" "d:/vivekware/Assignment/Assignment 1/Python/orderdetails.py"
219.99
(1, 2, 1, 1, 2, 2, datetime.date(2024, 2, 1), Decimal('651.17'), 'Shipped', 2, 'Rohit', 'Rajput', 'rohit@gmail.com', '1234567890', '102 aundhal', '1')
updated
Information updated
PS D:\Vivekware\Assignment\Assignment 1\Python>
```

DB changes:



The screenshot shows a SQL Enterprise Manager window with the following SQL queries:

1. `use techshop;`
2. `show tables;`
3. `select * from Orderdetails;`
- 4.

The result grid displays the following data:

OrderDetailID	OrderID	ProductID	Quantity
1	2	3	1
4	2	4	2
5	3	3	3
6	3	6	1
7	4	7	1
8	5	9	2
9	6	8	1
10	7	10	1
11	8	10	1

Input methods:

```

115 inventory_handler = Inventory()
116
117 product_info = inventory_handler.get_product(1)
118 print(product_info)
119 quantity_in_stock = inventory_handler.get_quantity_in_stock(1)
120 print(quantity_in_stock)
121 inventory_handler.add_to_inventory(1, 10)
122 print('added to inventory')
123 inventory_handler.remove_from_inventory(1, 5)
124 print('removed from inventory')
125 inventory_handler.update_stock_quantity(1, 20)
126 print('updated')
127 availability_status = inventory_handler.is_product_available(1, 15)
128 print(availability_status)
129 inventory_value = inventory_handler.get_inventory_value()
130 print(inventory_value)
131 low_stock_products = inventory_handler.list_low_stock_products(10)
132 print(low_stock_products)
133 out_of_stock_products = inventory_handler.list_out_of_stock_products()
134 print(out_of_stock_products)
135 all_products = inventory_handler.list_all_products()
136 print(all_products)
137
138 conn.commit()
139
140 # Close the database connection.
141 cursor.close()
142 conn.close()

```

```

PS D:\VivaWare\Assignment\Assignment 1\Python & C:\Program Files\Python312\python.exe "D:\VivaWare\Assignment\Assignment 1\Python\inventory.py"
(1, 'External Hard Drive', 'Updated new description', Decimal('1199.99'), 'Electronic Gadget')
28
added to inventory
removed from inventory
updated
2
226676.35
[]
[(1, 1, 28, datetime.datetime(2024, 1, 1, 9, 0), 1, 'External Hard Drive', 'Updated new description', Decimal('1199.99'), 'Electronic Gadget'), (2, 2, 380, datetime.datetime(2024, 2, 1, 11, 30), 2, 'Wireless Router', 'dual-band router', Decimal('133.89'), 'Electronic Gadget'), (3, 3, 75, datetime.datetime(2024, 3, 1, 13, 45), 3, 'Tablet', '10-inch tablet with long battery', Decimal('219.99'), 'Electronic Gadget'), (4, 4, 30, datetime.datetime(2024, 4, 1, 12, 15), 4, 'Printer', 'wireless all-in-one printer', Decimal('215.99'), 'Electronic Gadget'), (5, 5, 80, datetime.datetime(2024, 5, 1, 15, 45), 5, 'Smartwatch', 'Fitness tracker having heart rate monitor', Decimal('164.99'), 'Electronic Gadget'), (6, 6, 20, datetime.datetime(2024, 6, 1, 19, 0), 6, 'Laptop', 'High-performance laptop', Decimal('1099.99'), 'Electronic Gadget'), (7, 7, 40, datetime.datetime(2024, 7, 1, 22, 15), 7, 'Headphones', 'headphones with Bluetooth', Decimal('197.99'), 'Electronic Gadget'), (8, 8, 15, datetime.datetime(2024, 8, 1, 21, 30), 8, 'Camera', 'Digital camera with 20MP', Decimal('552.19'), 'Electronic Gadget'), (9, 9, 60, datetime.datetime(2024, 9, 1, 1, 25), 9, 'Desktop Computer', 'Powerful desktop', Decimal('1539.99'), 'Electronic Gadget'), (10, 10, 25, datetime.datetime(2024, 10, 1, 1, 0), 10, 'Smartphone', 'latest model with dual camera', Decimal('879.99'), 'Electronic Gadget')]
PS D:\VivaWare\Assignment\Assignment 1\Python

```

DB changes:

The screenshot shows a SQL client window with a query editor at the top and a results grid at the bottom. The query editor contains the following SQL code:

```
1. use techshop;
2. show tables;
3. select * from Inventory;
4
```

The results grid displays the output of the query, showing 10 rows of data. The columns are InventoryID, ProductID, QuantityInStock, and LastModifiedDate. The data is as follows:

InventoryID	ProductID	QuantityInStock	LastModifiedDate
1	2	20	2024-01-01 09:00:00
2	2	100	2024-02-01 11:30:00
3	3	75	2024-03-01 13:40:00
4	4	30	2024-04-01 12:15:00
5	5	80	2024-05-01 10:45:00
6	6	20	2024-06-01 08:00:00
7	7	40	2024-07-01 20:15:00
8	8	15	2024-08-01 21:30:00
9	9	60	2024-09-01 01:25:00
10	10	25	2024-10-01 03:00:00

Task 7:

```
database.py •
database.py > ...
1 import mysql.connector
2 from datetime import datetime
3
4 # Database Connection
5 conn = mysql.connector.connect(
6     host="localhost",
7     user="root",
8     password="root",
9     database="techshop"
10 )
11 cursor = conn.cursor()
12
13 # Task 1: Customer Registration
14 def register_customer(customerid, first_name, last_name, email, phone):
15     try:
16         # Check for duplicate email
17         cursor.execute("SELECT * FROM customers WHERE Email = %s", (email,))
18         existing_customer = cursor.fetchone()
19
20         if existing_customer:
21             print("Error: Duplicate email address.")
22             return
23
24         # Insert new customer
25         query = "INSERT INTO customers (customerid, first_name, last_name, email, phone) VALUES (%s, %s, %s, %s, %s)"
26         cursor.execute(query, (customerid, first_name, last_name, email, phone))
27         conn.commit()
28         print("Customer registered successfully.")
29     except Exception as e:
30         print(f"Error: {e}")
31
```

```
database.py •
database.py > ...
32 # Task 2: Product Catalog Management
33 def update_product(product_id, new_price, new_description):
34     try:
35         # Update product information
36         query = "UPDATE Products SET Price = %s, Description = %s WHERE ProductID = %s"
37         cursor.execute(query, (new_price, new_description, product_id))
38         conn.commit()
39         print("Product information updated successfully.")
40     except Exception as e:
41         print(f"Error: {e}")
42
43 # Task 3: Placing Customer Orders
44 def place_order(orderid, customer_id, product_id, quantity):
45     try:
46         # Check product availability in inventory
47         cursor.execute("SELECT quantityinstock FROM inventory WHERE ProductID = %s", (product_id,))
48         available_quantity = cursor.fetchone()[0]
49
50         if available_quantity < quantity:
51             print("Error: Insufficient stock.")
52             return
53
54         # Insert new order
55         order_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
56         query_insert_order = "INSERT INTO orders (orderid, customerid, orderdate, totalamount) VALUES (%s, %s, %s, %s)"
57         cursor.execute(query_insert_order, (orderid, customer_id, order_date, 0))
58         conn.commit()
59
60         # Get the newly created order ID
61         query_get_new_order_id = "SELECT LAST_INSERT_ID();"
62         cursor.execute(query_get_new_order_id)
63         new_order_id = cursor.fetchone()[0]
64
65     #
66
```

```
database.py •
database.py >...
65 #
66     orderdetailid=orderid
67     # Insert order details
68     query_insert_order_details = "INSERT INTO OrderDetails (orderdetailid,OrderID, ProductID, Quantity) VALUES (%s,%s, %s, %s)"
69     cursor.execute(query_insert_order_details, (orderdetailid,new_order_id, product_id, quantity))
70     conn.commit()
71
72     # update inventory quantity
73     query_update_inventory = "UPDATE Inventory SET QuantityInStock = QuantityInStock - %s WHERE ProductID = %s"
74     cursor.execute(query_update_inventory, (quantity, product_id))
75     conn.commit()
76 #
77
78     print("Order placed successfully.")
79 except Exception as e:
80     print(f"Error: {e}")
81
82 # Task 4: Tracking Order Status
83 def track_order_status(order_id):
84     try:
85         # Retrieve order status
86         query = "SELECT status FROM Orders WHERE OrderID = %s"
87         cursor.execute(query, (order_id,))
88         status = cursor.fetchone()
89
90         if status:
91             print(f"Order {order_id} status: {status[0]}")
92         else:
93             print("Error: Order not found.")
94     except Exception as e:
95         print(f"Error: {e}")
96
```

```
database.py •
database.py >...
98 def add_new_product(productid,product_name, description, price, quantity_in_stock):
99     try:
100         # Insert new product
101         query_insert_product = "INSERT INTO Products (productid,Productname, Description, Price) VALUES (%s,%s, %s, %s)"
102         cursor.execute(query_insert_product, (productid,product_name, description, price))
103         conn.commit()
104
105         # Get the newly created product ID
106         query_get_new_product_id = "SELECT LAST_INSERT_ID();"
107         cursor.execute(query_get_new_product_id)
108         new_product_id = cursor.fetchone()[0]
109         ***
110         # Add product to inventory
111         inventoryid=productid
112         query_add_to_inventory = "INSERT INTO Inventory (inventoryid,ProductID, QuantityInStock) VALUES (%s,%s, %s)"
113         cursor.execute(query_add_to_inventory, (inventoryid,new_product_id, quantity_in_stock))
114         conn.commit()
115         ***
116         print("New product added to inventory successfully.")
117     except Exception as e:
118         print(f"Error: {e}")
119
120 # Task 6: Sales Reporting
121 def generate_sales_report(start_date, end_date):
122     try:
123         # Retrieve sales data based on specified criteria
124         query = """
125             SELECT Orders.OrderID, Customers.FirstName, Customers.LastName, Orders.OrderDate, Orders.TotalAmount
126             FROM Orders
127             JOIN Customers ON Orders.CustomerID = Customers.CustomerID
128             WHERE Orders.OrderDate BETWEEN %s AND %s
129             """
130         cursor.execute(query, (start_date, end_date))
131         sales_data = cursor.fetchall()
132     except Exception as e:
133         print(f"Error: {e}")
134
```


DB Changes:

```
3. select * from products;  
4
```

ProductID	ProductName	Description	Price	Category
1	Smartwatch	Fitness tracker having heart rate monitor	164.99	Electronic Gadget
2	Laptop	High performance laptop	1099.99	Electronic Gadget
3	Headphones	Headphones with Bluetooth	157.99	Electronic Gadget
4	Camera	Digital camera with 20MP	552.18	Electronic Gadget
5	Desktop Computer	Powerful desktop	1339.99	Electronic Gadget
6	Smartphone	Latest model with dual camera	876.99	Electronic Gadget
7	New Electronic Ge...	Description of the new gadget	299.99	Electronic Gadget
8	Tablet	High-resolution display	800.00	Electronic Gadget
9	Tablet	High-resolution display	800.00	Electronic Gadget
10	AC	High-resolution display	1000.00	Electronic Gadget
11	AC	High-resolution display	1000.00	Electronic Gadget
12	AC	High-resolution display	1000.00	Electronic Gadget

```
3. select * from customers;  
4
```

CustomerID	FirstName	LastName	Email	Phone	Address	OrderNumbers
1	Rohit	Ragavi	rohit@gmail.com	1234567890	101 Mumbai	1
2	Rishi	Shela	rishi@gmail.com	1234567890	102 Pune	1
3	Tejal	Chaudhari	tejal@gmail.com	1234567890	103 Nagpur	1
4	Pritya	Chaudhari	pritya@gmail.com	1234567890	104 Jaipur	1
5	Pranjal	Wani	pranjal@gmail.com	1234567890	105 Navi Mumbai	1
6	Shwetha	Madhwa	shwetha@gmail.com	1234567890	107 Kalyan	1
7	Shubham	Maria	shubham@gmail.com	1234567890	108 Dombivli	1
8	Hiru	Mahajan	hiru@gmail.com	1234567890	109 Anand	1
9	Hanshal	Rane	hanshal@gmail.com	1234567890	110 Jaipur	1
10	Shravya	Zope	shravya@example.com	1234567890	111 Nashik	0
11	John	Doe	john.doe@example.com	1234567890	112	0000
12	Rohani	Chaudhari	Rohani@example.com	1234567890	113	0000

```
3. select * from orders;  
4
```

OrderID	CustomerID	OrderDate	TotalAmount	Status
1	1	2024-08-01	0.00	0000
2	1	2024-09-01	0.00	0000
3	10	2024-10-01	0.00	0000
4	1	2023-12-19	0.00	0000
5	1	2023-12-19	0.00	0000
6	1	2023-12-19	0.00	0000
7	1	2023-12-19	0.00	0000
8	1	2023-12-19	0.00	0000
9	1	2023-12-19	0.00	0000
10	1	2023-12-19	0.00	0000
11	1	2023-12-19	0.00	0000
12	1	2023-12-19	0.00	0000
13	1	2023-12-19	0.00	0000
14	1	2023-12-19	0.00	0000
15	1	2023-12-19	0.00	0000
16	1	2023-12-19	0.00	0000
17	1	2023-12-19	0.00	0000
18	1	2023-12-19	0.00	0000
19	1	2023-12-19	0.00	0000
20	1	2023-12-19	0.00	0000

```
3. select * from orderdetails;  
4
```

OrderDetailID	OrderID	ProductID	Quantity
1	2	3	1
2	2	4	2
3	3	5	3
4	3	6	1
5	4	7	1
6	5	9	2
7	6	8	1
8	7	10	1

```
3. select * from inventory;  
4
```

InventoryID	ProductID	QuantityInStock	LastStockUpdate
1	1	20	2024-01-01 09:00:00
2	2	100	2024-02-01 11:30:00
3	3	75	2024-03-01 13:45:00
4	4	30	2024-04-01 12:15:00
5	5	80	2024-05-01 15:45:00
6	6	20	2024-06-01 19:00:00
7	7	40	2024-07-01 22:15:00
8	8	15	2024-08-01 21:30:00
9	9	60	2024-09-01 01:25:00
10	10	25	2024-10-01 03:00:00