VIJAY R

Superset Id : 5371616

TechShop, an electronic gadgets shop

Oops Implementation

models/customers.py:

```
from models.exceptions import InvalidDataException

class Customers:
    def __init__(self, customer_id, first_name, last_name, email, phone, address):
        self.__customer_id = customer_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email
        self.__phone = phone
        self.__address = address

    def get_customer_details(self):
        return f"ID: {self.__customer_id}, Name: {self.__first_name} {self.__last_name},
Email: {self.__email}, Phone: {self.__phone}, Address: {self.__address}"

    def update_customer_info(self, email=None, phone=None, address=None):
        if email and "@" not in email:
            raise InvalidDataException("Invalid email address.")
        if email:
            self.__email = email
        if phone:
            self.__phone = phone
        if address:
            self.__address = address
```

models/products.py:

```
import mysql.connector

class Products:
    def __init__(self, product_id, product_name, description, price):
        if price < 0:
            raise ValueError("Price cannot be negative.")

        self.__product_id = product_id
```

```
        self.__product_name = product_name
        self.__description = description
        self.__price = price

    def get_product_details(self):
        return f"Product ID: {self.__product_id}, Name: {self.__product_name}, Price:
{self.__price}"

    @staticmethod
    def add_product(db, product_name, description, price):
        """Insert a new product into the database"""
        try:
            cursor = db.connection.cursor()
            query = "INSERT INTO products (ProductName, Description, Price) VALUES (%s,
%s, %s)"
            cursor.execute(query, (product_name, description, price))
            db.connection.commit()
            print("Product added successfully!")
        except mysql.connector.Error as e:
            print(f"Error adding product: {e}")
        finally:
            cursor.close()

    @staticmethod
    def update_product(db, product_id, new_price=None, new_description=None):
        """Update product details in the database"""
        try:
            cursor = db.connection.cursor()
            update_fields = []
            values = []

            if new_price is not None:
                if new_price < 0:
                    raise ValueError("Price cannot be negative.")
                update_fields.append("Price = %s")
                values.append(new_price)

            if new_description:
                update_fields.append("Description = %s")
                values.append(new_description)

            values.append(product_id)

            if update_fields:
                query = f"UPDATE products SET {', '.join(update_fields)} WHERE ProductID =
%s"
```

```
            cursor.execute(query, values)
            db.connection.commit()
            print("Product updated successfully!")
        else:
            print("No changes made.")

    except mysql.connector.Error as e:
        print(f"Error updating product: {e}")
    finally:
        cursor.close()
```

models/orders.py:

```
from datetime import datetime

class Orders:
    def __init__(self, order_id, customer, order_date, total_price):
        self.order_id = order_id
        self.customer = customer
        self.order_date = order_date
        self.total_price = total_price

    def __str__(self):
        return f"Order ID: {self.order_id}, Customer: {self.customer.name}, Date:
{self.order_date}, Total: {self.total_price}"
```

models/order_details.py:

```
from exceptions.customer_exceptions import InvalidDataException

class OrderDetails:
    def __init__(self, order_detail_id, order, product, quantity):
        if quantity <= 0:
            raise InvalidDataException("Quantity must be positive.")
        self.__order_detail_id = order_detail_id
        self.__order = order
        self.__product = product
        self.__quantity = quantity

    def calculate_subtotal(self):
        return self.__quantity * self.__product.get_price()  # Assuming get_price() exists in
Products

    def update_quantity(self, new_quantity):
        if new_quantity <= 0:
```

```
            raise InvalidDataException("Quantity must be positive.")
        self.__quantity = new_quantity
```

models/payment.py:

```
from models.exceptions import PaymentFailedException

class Payment:
    def __init__(self, payment_id, order, amount, status="Pending"):
        self.__payment_id = payment_id
        self.__order = order
        self.__amount = amount
        self.__status = status

    def process_payment(self, success=True):
        if success:
            self.__status = "Completed"
        else:
            raise PaymentFailedException("Payment was declined.")
```

models/inventory.py:

```
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock):
        self.__inventory_id = inventory_id
        self.__product = product
        self.__quantity_in_stock = quantity_in_stock

    def remove_from_inventory(self, quantity):
        if quantity > self.__quantity_in_stock:
            raise InsufficientStockException("Not enough stock available.")
        self.__quantity_in_stock -= quantity

    def is_product_available(self, quantity_to_check):
        return self.__quantity_in_stock >= quantity_to_check
```

models/exceptions.py:

```
class InvalidDataException(Exception):
    pass

class InsufficientStockException(Exception):
    pass

class IncompleteOrderException(Exception):
```

```
    pass

class PaymentFailedException(Exception):
    pass
```

services/customer_service.py:

```
from database.db_connector import DatabaseConnector
from models.exceptions import InvalidDataException

class CustomerService:
    def __init__(self):
        self.db = DatabaseConnector()
        self.db.open_connection()

    def register_customer(self, first_name, last_name, email, phone, address):
        cursor = self.db.connection.cursor()

        # Check if email already exists
        cursor.execute("SELECT * FROM customers WHERE Email = %s", (email,))
        if cursor.fetchone():
            raise InvalidDataException("Email already registered.")

        # Insert new customer record
        query = "INSERT INTO customers (FirstName, LastName, Email, Phone, Address)
VALUES (%s, %s, %s, %s, %s)"
        values = (first_name, last_name, email, phone, address)

        try:
            cursor.execute(query, values)
            self.db.connection.commit()
            print("Customer registered successfully!")
        except Exception as e:
            print(f"Error registering customer: {e}")
        finally:
            cursor.close()

    def close(self):
        self.db.close_connection()
```

services/order_management.py:

```
from database.db_connector import DatabaseConnector
from models.orders import Orders
from models.products import Products
```

```
from exceptions.customer_exceptions import InsufficientStockException
from datetime import datetime


class OrderManagementService:
    def __init__(self):
        self.db = DatabaseConnector()

    def place_order(self, customer, product, quantity):
        conn = self.db.open_connection()
        cursor = conn.cursor()

        # Check product stock
        cursor.execute("SELECT stock, price FROM products WHERE id = %s",
(product.product_id,))
        result = cursor.fetchone()
        if not result:
            raise ValueError("Product not found.")

        stock, price = result
        if stock < quantity:
            raise InsufficientStockException("Not enough stock available.")

        # Calculate total price
        total_price = price * quantity

        # Insert order into the database
        cursor.execute(
            "INSERT INTO orders (customer_id, order_date, total_price) VALUES (%s, %s,
%s)",
            (customer.customer_id, datetime.now(), total_price)
        )
        order_id = cursor.lastrowid  # Get the generated order ID

        # Update product stock
        new_stock = stock - quantity
        cursor.execute("UPDATE products SET stock = %s WHERE id = %s", (new_stock,
product.product_id))

        conn.commit()
        self.db.close_connection()

        return f"Order placed successfully! Order ID: {order_id}, Total: {total_price}"
```

services/order_processing.py:

```
from database.db_connector import DatabaseConnector
from models.exceptions import InsufficientStockException, InvalidDataException


class OrderProcessingService:
    def __init__(self):
        self.db = DatabaseConnector()
        self.db.open_connection()

    def place_order(self, customer_id, product_id, quantity):
        cursor = self.db.connection.cursor()

        # Check product stock
        cursor.execute("SELECT QuantityInStock FROM inventory WHERE ProductID = %s",
(product_id,))
        stock = cursor.fetchone()

        if not stock or stock[0] < quantity:
            raise InsufficientStockException("Not enough stock available.")

        # Get product price
        cursor.execute("SELECT Price FROM products WHERE ProductID = %s",
(product_id,))
        product_price = cursor.fetchone()[0]
        total_price = product_price * quantity

        # Insert into orders table
        cursor.execute("INSERT INTO orders (CustomerID, OrderDate, TotalAmount)
VALUES (%s, NOW(), %s)",
                       (customer_id, total_price))
        self.db.connection.commit()
        order_id = cursor.lastrowid

        # Insert into order details
        cursor.execute("INSERT INTO order_details (OrderID, ProductID, Quantity) VALUES
(%s, %s, %s)",
                       (order_id, product_id, quantity))
        self.db.connection.commit()

        # Update inventory
        cursor.execute("UPDATE inventory SET QuantityInStock = QuantityInStock - %s
WHERE ProductID = %s",
                       (quantity, product_id))
        self.db.connection.commit()
```

```
        print("Order placed successfully!")
        cursor.close()

    def close(self):
        self.db.close_connection()
```

services/product_management.py:

```
from database.db_connector import DatabaseConnector
from models.products import Products

class ProductManagementService:
    def __init__(self):
        self.db = DatabaseConnector()
        self.db.open_connection()

    def add_product(self):
        """Get product details from user and add to the database"""
        name = input("Enter Product Name: ")
        description = input("Enter Description: ")
        price = float(input("Enter Price: "))
        Products.add_product(self.db, name, description, price)

    def update_product(self):
        """Update product details"""
        product_id = int(input("Enter Product ID to update: "))
        new_price = input("Enter new price (press enter to skip): ")
        new_description = input("Enter new description (press enter to skip): ")

        new_price = float(new_price) if new_price else None
        new_description = new_description if new_description else None

        Products.update_product(self.db, product_id, new_price, new_description)

    def close_service(self):
        self.db.close_connection()
```

exceptions/customer_exceptions.py:

```
class InsufficientStockException(Exception):
    def __init__(self, message="Not enough stock available."):
        super().__init__(message)

class InvalidDataException(Exception):
```

```
    def __init__(self, message="Invalid data provided."):
        super().__init__(message)
```

Database Connection:

database/db_connector.py:

```
import mysql.connector

class DatabaseConnector:
    def __init__(self):
        self.host = "localhost"
        self.user = "root"
        self.password = "#vijaysql**"
        self.database = "TechGadgetShop"
    def open_connection(self):
        try:
            conn = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            if conn.is_connected():
                print("✅ Database connected successfully!")
                return conn
            else:
                print("❌ Database connection failed!")
                return None
        except mysql.connector.Error as e:
            print(f"❌ Error connecting to database: {e}")
            return None

    def close_connection(self, conn):
        if conn:
            conn.close()
            print("🔒 Database connection closed.")
```

1: Customer Registration Description:

When a new customer registers on the TechShop website, their information (e.g., name, email, phone) needs to be stored in the database. Task: Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.

Main.py :

```
from services.customer_service import CustomerService

if __name__ == "__main__":
    customer_service = CustomerService()

    try:
        # Simulating user input
        first_name = input("Enter First Name: ")
        last_name = input("Enter Last Name: ")
        email = input("Enter Email: ")
        phone = input("Enter Phone Number: ")
        address = input("Enter Address: ")

        customer_service.register_customer(first_name, last_name, email, phone, address)

    except Exception as e:
        print(f"Error: {e}")

    finally:
        customer_service.close()
```

```
C:\Users\VIJAY\PycharmProjects\TechShop\.venv\Scripts\python.exe C:\Users\VIJAY\PycharmProjects\TechShop\.venv\main.py
Enter First Name: Harish
Enter Last Name: Rao
Enter Email: harishrao@gmail.com
Enter Phone Number: 1234567890
Enter Address: 123 street, Bangalore
Customer registered successfully!

Process finished with exit code 0
```

CustomerID	FirstName	LastName	Email	Phone	Address	OrderCount
1	Steve	John	steve.john@email.com	9876543210	123 Main St, NY	1
2	Mark	Smith	mark.smith@email.com	9876543211	456 Last St, LA	1
3	Kelly	Brown	kelly.brown@email.com	9876543212	789 Pine St, SF	2
4	Mohamad	Sherif	sherif.m345@email.com	9876543213	101 Broadway St, TX	1
5	David	Raj	david.rr45@email.com	9876543214	202 Paris St, FL	0
6	David	Wilson	david.wilson@email.com	9876543215	303 Birch St, NV	1
7	Emma	Watson	emma.wat6070@email.com	9876543216	404 Cedar St, IL	1
8	Joseph	Vijay	jos.vj123@email.com	9876543217	505 Panayur St, CH	1
9	Joyshy	Grace	grace.Joy@email.com	9876543218	606 New St, WA	1
10	Mark	Henry	henry.mark@gmail.com	9876543219	706 kads St, NY	0
11	Sunil	Roa	sunil.roa@email.com	9876543220	1725 5th Ave, London	0
12	Ravi	Sharma	ravi.sharma@example.com	9876543210	12 MG Road, Bengal...	0
13	Harish	Rao	harishrao@gmail.com	1234567890	123 street, Bangalore	0

2: Product Catalog Management

Description: TechShop regularly updates its product catalog with new items and changes in product details (e.g., price, description). These changes need to be reflected in the database.

Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency.

Main.py:

```
from services.product_management import ProductManagementService

def main():
    product_service = ProductManagementService()

    while True:
        print("\n--- Product Catalog Management ---")
        print("1. Add Product")
        print("2. Update Product")
        print("3. Exit")
        choice = input("Enter choice: ")

        if choice == "1":
            product_service.add_product()
        elif choice == "2":
            product_service.update_product()
        elif choice == "3":
            product_service.close_service()
            break
        else:
            print("Invalid choice. Try again.")

if __name__ == "__main__":
    main()
```

```
C:\Users\VIJAY\PycharmProjects\TechShop\.venv\Scripts\python.exe C:\Users\VIJAY\PycharmProjects\TechShop\.venv\main.py

--- Product Catalog Management ---
1. Add Product
2. Update Product
3. Exit
Enter choice: 1
Enter Product Name: Airpods
Enter Description: It has Bluetooth 5.3
Enter Price: 2999
Product added successfully!

--- Product Catalog Management ---
1. Add Product
2. Update Product
3. Exit
Enter choice: 2
Enter Product ID to update: 2
Enter new price (press enter to skip): 1299
Enter new description (press enter to skip): Has Intel I9
Product updated successfully!

--- Product Catalog Management ---
1. Add Product
2. Update Product
3. Exit
Enter choice: 3

Process finished with exit code 0
```

ProductID	ProductName	Description	Price
1	Smartphone	With 120hz refresh rate	30000.00
2	Laptop	Has Intel I9	1299.00
3	Tablet	10-inch display tablet	439.99
4	Smartwatch	Water-resistant smartwatch	219.99
5	Bluetooth Speaker	Portable speaker	109.99
6	Headphones	Noise-cancelling headphones	164.99
7	Gaming Console	Next-gen gaming console	549.99
8	Wireless Mouse	Ergonomic wireless mouse	32.99
9	Keyboard	Mechanical keyboard	87.99
10	Monitor	27-inch 4K monitor	329.99
11	Power Bank	20,000 mah Battery	738.99
12	Airpods	It has Bluetooth 5.3	2999.00
NULL	NULL	NULL	NULL

3: Placing Customer Orders

Description: Customers browse the product catalog and place orders for products they want to purchase. The orders need to be stored in the database.

Task: Implement an order processing system. Use database connectivity to record customer orders, update product quantities in inventory, and calculate order totals.

Main.py:

```
import mysql.connector
from datetime import datetime

# Database connection
```

```
def connect_db():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",  # Change if needed
            password="#vijaysql**",  # Change if needed
            database="TechGadgetShop"  # Change if needed
        )
        return conn
    except mysql.connector.Error as err:
        print("Database connection error:", err)
        return None

# Display available products
def show_products(cursor):
    cursor.execute("SELECT ProductID, ProductName, Price FROM products")
    products = cursor.fetchall()
    print("\nAvailable Products:")
    for product in products:
        print(f"{product[0]}. {product[1]} - ₹{product[2]}")
    return products

# Get customer ID from email
def get_customer_id(cursor, email):
    cursor.execute("SELECT CustomerID FROM customers WHERE Email = %s", (email,))
    result = cursor.fetchone()
    return result[0] if result else None

# Place an order
def place_order(cursor, conn, customer_id, product_id, quantity):
    # Check stock availability
    cursor.execute("SELECT ProductName, Price, stock FROM products WHERE ProductID
= %s", (product_id,))
    product = cursor.fetchone()
    if not product:
        print("Invalid product selection.")
        return
    product_name, price, stock = product

    if stock < quantity:
        print("Insufficient stock available!")
        return

    # Calculate total amount
    total_amount = price * quantity
```

```
    # Insert order
    order_date = datetime.now().date()
    cursor.execute("INSERT INTO orders (CustomerID, OrderDate, TotalAmount, Status)
VALUES (%s, %s, %s, %s)",
                   (customer_id, order_date, total_amount, "Pending"))
    order_id = cursor.lastrowid

    # Update stock
    new_stock = stock - quantity
    cursor.execute("UPDATE products SET stock = %s WHERE ProductID = %s",
(new_stock, product_id))

    conn.commit()
    print(f"\nOrder placed successfully! Order ID: {order_id}")
    print(f"{quantity} {product_name}(s) ordered for ₹{total_amount}")

# Main function
def main():
    conn = connect_db()
    if conn is None:
        return
    cursor = conn.cursor()

    # Display products
    products = show_products(cursor)

    # Get customer email
    email = input("\nEnter your registered email: ")
    customer_id = get_customer_id(cursor, email)

    if not customer_id:
        print("Customer not found! Please register first.")
        return

    # Select product
    product_id = int(input("Enter Product ID to order: "))
    quantity = int(input("Enter quantity: "))

    # Place order
    place_order(cursor, conn, customer_id, product_id, quantity)

    # Close connection
    cursor.close()
    conn.close()
```

```
if __name__ == "__main__":
    main()
```

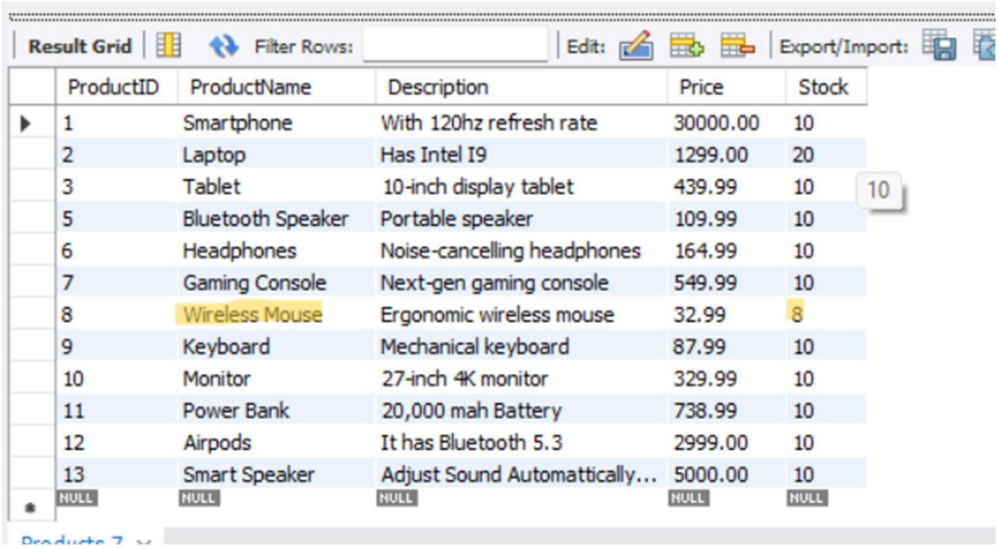
```
C:\Users\VIJAY\PycharmProjects\TechShop\.venv\Scripts\python.exe C:\Users\VIJAY\PycharmProjects\TechShop\.venv\main.py

Available Products:
1. Smartphone - ₹30000.00
2. Laptop - ₹1299.00
3. Tablet - ₹439.99
4. Smartwatch - ₹219.99
5. Bluetooth Speaker - ₹109.99
6. Headphones - ₹164.99
7. Gaming Console - ₹549.99
8. Wireless Mouse - ₹32.99
9. Keyboard - ₹87.99
10. Monitor - ₹329.99
11. Power Bank - ₹738.99
12. Airpods - ₹2999.00

Enter your registered email: emma.wat6070@email.com
Enter Product ID to order: 8
Enter quantity: 2

Order placed successfully! Order ID: 12
2 Wireless Mouse(s) ordered for ₹65.98

Process finished with exit code 0
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	Email	Phone	Address	OrderCount
1	Steve	John	steve.john@email.com	9876543210	123 Main St, NY	1
2	Mark	Smith	mark.smith@email.com	9876543211	456 Last St, LA	1
3	Kelly	Brown	kelly.brown@email.com	9876543212	789 Pine St, SF	2
4	Mohamad	Sherif	sherif.m345@email.com	9876543213	101 Broadway St, TX	1
5	David	Raj	david.rr45@email.com	9876543214	202 Paris St, FL	0
6	David	Wilson	david.wilson@email.com	9876543215	303 Birch St, NV	1
7	Emma	Watson	emma.wat6070@email.com	9876543216	404 Cedar St, IL	1
8	Joseph	Vijay	jos.vj123@email.com	9876543217	505 Panayur St, CH	1
9	Joyshy	Grace	grace.Joy@email.com	9876543218	606 New St, WA	1
10	Mark	Henry	henry.mark@gmail.com	9876543219	706 kads St, NY	0
11	Sunil	Roa	sunil.roa@email.com	9876543220	1725 5th Ave, London	0
12	Ravi	Sharma	ravi.sharma@example.com	9876543210	12 MG Road, Bengal...	0
13	Harish	Rao	harishrao@gmail.com	1234567890	123 street, Bangalore	0

Result Grid | Filter Rows: | Edit: | Export

OrderID	CustomerID	OrderDate	TotalAmount	Status
1	1	2024-01-01	769.99	Pending
2	2	2024-01-05	1429.44	Pending
3	3	2024-02-10	2199.95	Pending
4	4	2024-02-12	219.99	Pending
6	6	2025-02-18	164.99	Pending
7	7	2025-01-20	3849.93	Pending
8	8	2024-05-22	197.94	Pending
9	9	2024-08-24	175.98	Shipped
11	3	2025-02-12	0.00	Pending
12	7	2025-03-28	65.98	Pending
NULL	NULL	NULL	NULL	NULL



Result Grid | Filter Rows: | Edit: | Export/Import:

ProductID	ProductName	Description	Price	Stock
1	Smartphone	With 120hz refresh rate	30000.00	10
2	Laptop	Has Intel I9	1299.00	20
3	Tablet	10-inch display tablet	439.99	10
5	Bluetooth Speaker	Portable speaker	109.99	10
6	Headphones	Noise-cancelling headphones	164.99	10
7	Gaming Console	Next-gen gaming console	549.99	10
8	Wireless Mouse	Ergonomic wireless mouse	32.99	8
9	Keyboard	Mechanical keyboard	87.99	10
10	Monitor	27-inch 4K monitor	329.99	10
11	Power Bank	20,000 mah Battery	738.99	10
12	Airpods	It has Bluetooth 5.3	2999.00	10
13	Smart Speaker	Adjust Sound Automattically...	5000.00	10
NULL	NULL	NULL	NULL	NULL

4: Tracking Order Status

Description: Customers and employees need to track the status of their orders. The order status information is stored in the database.

Task: Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

Main.py:

```
import mysql.connector


def track_order_status():
    """Retrieve and display the order status for a given customer."""
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="#vijaysql**",
        database="TechGadgetShop"
    )
    cursor = conn.cursor()

    email = input("Enter your email to track orders: ")

    # Check if the customer exists
    cursor.execute("SELECT CustomerID FROM customers WHERE Email = %s", (email,))
    customer = cursor.fetchone()

    if not customer:
        print("No customer found with this email.")
        return
```

```
    customer_id = customer[0]

    # Retrieve order details
    cursor.execute("SELECT OrderID, OrderDate, TotalAmount, Status FROM orders
WHERE CustomerID = %s", (customer_id,))
    orders = cursor.fetchall()

    if not orders:
        print("No orders found for this customer.")
    else:
        print("\nYour Orders:")
        print("{:<10} {:<15} {:<10} {:<10}".format("OrderID", "OrderDate", "TotalAmount",
"Status"))
        print("-" * 50)
        for order in orders:
            print("{:<10} {:<15} {:<10} {:<10}".format(order[0], order[1], order[2], order[3]))

    cursor.close()
    conn.close()


def main():
    while True:
        print("\nTechShop Order Management")
        print("1. Place Order")
        print("2. Track Order Status")
        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            print("Order placement feature is under development.")
        elif choice == "2":
            track_order_status()
        elif choice == "3":
            print("Exiting... Thank you!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")


if __name__ == "__main__":
    main()
```

5: Inventory Management

Description: TechShop needs to manage product inventory, including adding new products, updating stock levels, and removing discontinued items.

Task: Create an inventory management system with database connectivity. Implement features for adding new products, updating quantities, and handling discontinued products.

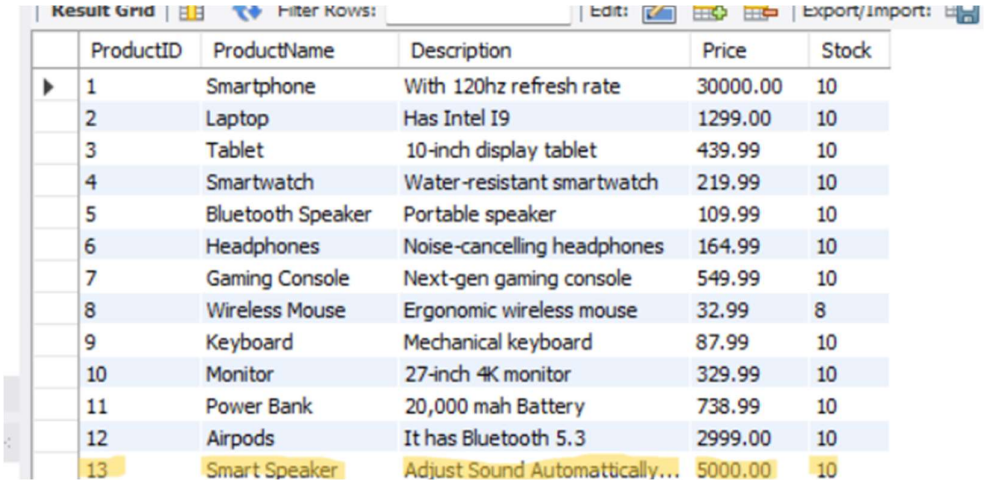
Main.py:

```
import mysql.connector

def add_product():
    """Add a new product to the inventory."""
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="#vijaysql**",
        database="TechGadgetShop"
    )
    cursor = conn.cursor()

    name = input("Enter product name: ")
    description = input("Enter product description: ")
    price = float(input("Enter product price: "))

    query = "INSERT INTO products (ProductName, Description, Price) VALUES (%s, %s, %s)"
    values = (name, description, price)
    cursor.execute(query, values)
```

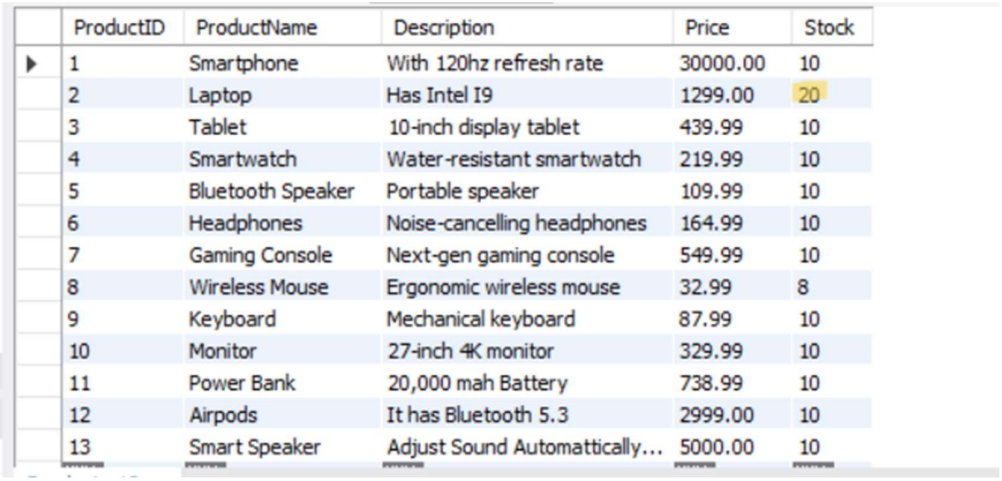
```
    conn.commit()

    print("Product added successfully!")

    cursor.close()
    conn.close()


def update_stock():
    """Update stock level of an existing product."""
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="#vijaysql**",
        database="TechGadgetShop"
    )
    cursor = conn.cursor()

    product_id = int(input("Enter ProductID to update stock: "))
    new_stock = int(input("Enter new stock quantity: "))

    query = "UPDATE products SET Stock = %s WHERE ProductID = %s"
    cursor.execute(query, (new_stock, product_id))
    conn.commit()

    print("Stock updated successfully!")

    cursor.close()
    conn.close()


def remove_product():
    """Remove a discontinued product from inventory."""
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="#vijaysql**",
        database="TechGadgetShop"
    )
    cursor = conn.cursor()

    product_id = int(input("Enter ProductID to remove: "))

    query = "DELETE FROM products WHERE ProductID = %s"
    cursor.execute(query, (product_id,))
    conn.commit()
```

```
    print("Product removed successfully!")

    cursor.close()
    conn.close()


def main():
    while True:
        print("\nTechShop Inventory Management")
        print("1. Add New Product")
        print("2. Update Stock Level")
        print("3. Remove Discontinued Product")
        print("4. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            add_product()
        elif choice == "2":
            update_stock()
        elif choice == "3":
            remove_product()
        elif choice == "4":
            print("Exiting... Thank you!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()
```

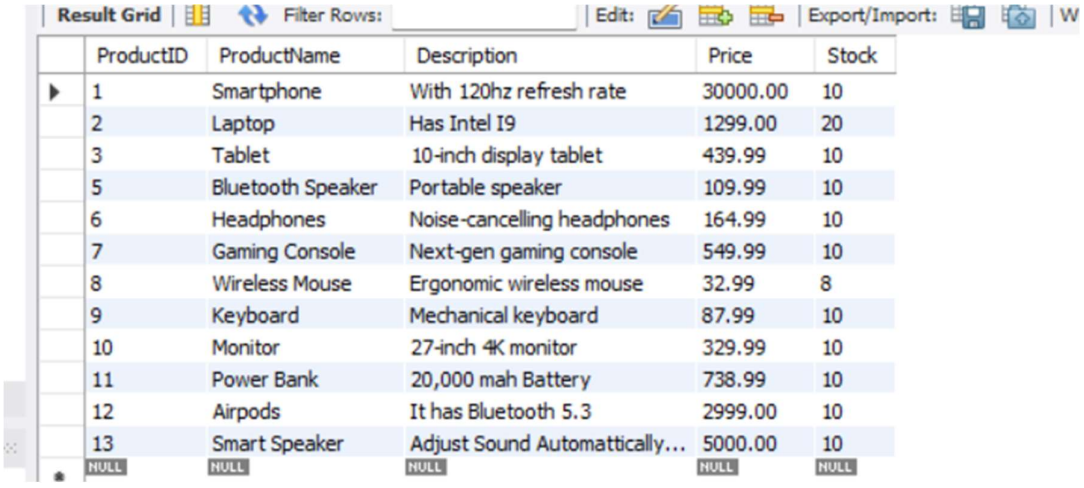
```
C:\Users\VIJAY\PycharmProjects\TechShop\.venv\Scripts\python.exe C:\Users\VIJAY\PycharmProjects\TechShop\.venv\main.py

TechShop Inventory Management
1. Add New Product
2. Update Stock Level
3. Remove Discontinued Product
4. Exit
Enter your choice: 1
Enter product name: Smart Speaker
Enter product description: Adjust Sound Automattically According to the Background.
Enter product price: 5000
Product added successfully!

TechShop Inventory Management
1. Add New Product
2. Update Stock Level
3. Remove Discontinued Product
4. Exit
Enter your choice: 2
Enter ProductID to update stock: 2
Enter new stock quantity: 20
Stock updated successfully!

TechShop Inventory Management
1. Add New Product
2. Update Stock Level
3. Remove Discontinued Product
4. Exit
Enter your choice: 3
Enter ProductID to remove: 4
Product removed successfully!
```



ProductID	ProductName	Description	Price	Stock
1	Smartphone	With 120hz refresh rate	30000.00	10
2	Laptop	Has Intel I9	1299.00	10
3	Tablet	10-inch display tablet	439.99	10
4	Smartwatch	Water-resistant smartwatch	219.99	10
5	Bluetooth Speaker	Portable speaker	109.99	10
6	Headphones	Noise-cancelling headphones	164.99	10
7	Gaming Console	Next-gen gaming console	549.99	10
8	Wireless Mouse	Ergonomic wireless mouse	32.99	8
9	Keyboard	Mechanical keyboard	87.99	10
10	Monitor	27-inch 4K monitor	329.99	10
11	Power Bank	20,000 mah Battery	738.99	10
12	Airpods	It has Bluetooth 5.3	2999.00	10
13	Smart Speaker	Adjust Sound Automattically...	5000.00	10



ProductID	ProductName	Description	Price	Stock
1	Smartphone	With 120hz refresh rate	30000.00	10
2	Laptop	Has Intel I9	1299.00	20
3	Tablet	10-inch display tablet	439.99	10
4	Smartwatch	Water-resistant smartwatch	219.99	10
5	Bluetooth Speaker	Portable speaker	109.99	10
6	Headphones	Noise-cancelling headphones	164.99	10
7	Gaming Console	Next-gen gaming console	549.99	10
8	Wireless Mouse	Ergonomic wireless mouse	32.99	8
9	Keyboard	Mechanical keyboard	87.99	10
10	Monitor	27-inch 4K monitor	329.99	10
11	Power Bank	20,000 mah Battery	738.99	10
12	Airpods	It has Bluetooth 5.3	2999.00	10
13	Smart Speaker	Adjust Sound Automattically...	5000.00	10



ProductID	ProductName	Description	Price	Stock
1	Smartphone	With 120hz refresh rate	30000.00	10
2	Laptop	Has Intel I9	1299.00	20
3	Tablet	10-inch display tablet	439.99	10
5	Bluetooth Speaker	Portable speaker	109.99	10
6	Headphones	Noise-cancelling headphones	164.99	10
7	Gaming Console	Next-gen gaming console	549.99	10
8	Wireless Mouse	Ergonomic wireless mouse	32.99	8
9	Keyboard	Mechanical keyboard	87.99	10
10	Monitor	27-inch 4K monitor	329.99	10
11	Power Bank	20,000 mah Battery	738.99	10
12	Airpods	It has Bluetooth 5.3	2999.00	10
13	Smart Speaker	Adjust Sound Automattically...	5000.00	10
NULL	NULL	NULL	NULL	NULL

6: Sales Reporting

Description: TechShop management requires sales reports for business analysis. The sales data is stored in the database.

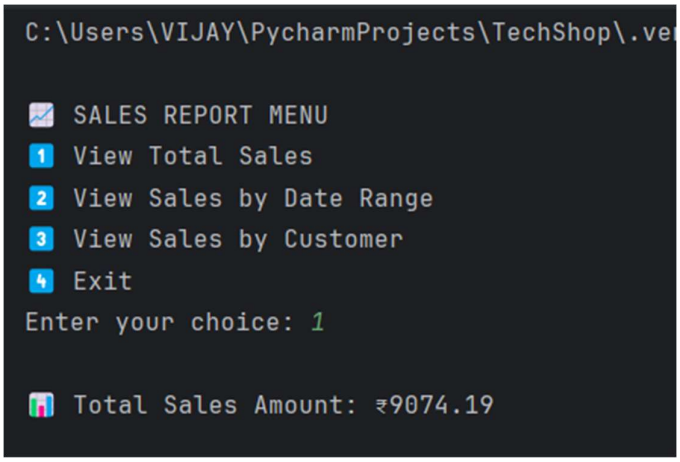
Task: Design and implement a reporting system that retrieves sales data from the database and generates reports based on specified criteria.

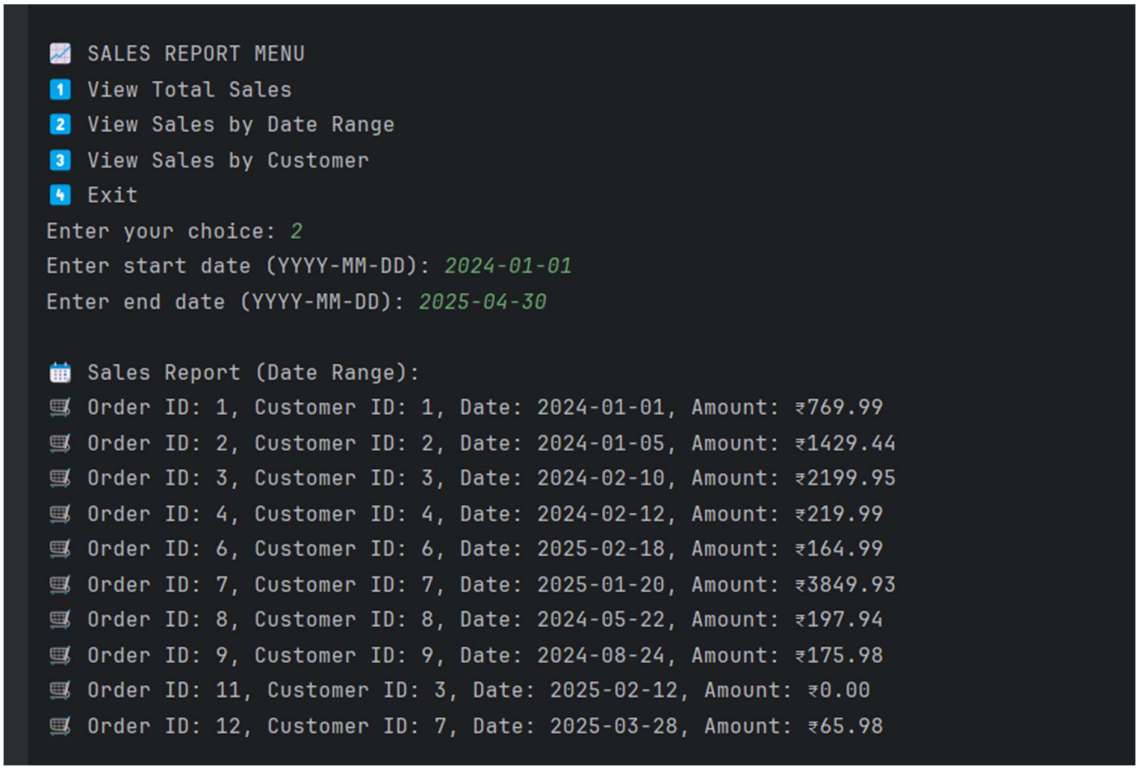
Main.py:

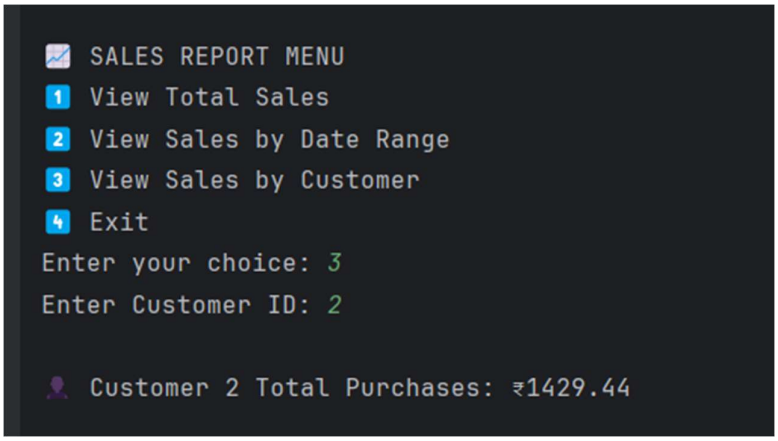
```
import mysql.connector


def get_db_connection():
    """Establish database connection"""
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="#vijaysql**",
            database="TechGadgetShop"
        )
        return conn
    except mysql.connector.Error as e:
        print(f"Error connecting to database: {e}")
        return None


def total_sales_report(cursor):
    """Fetch total sales amount"""
    cursor.execute("SELECT SUM(TotalAmount) FROM orders")
    total_sales = cursor.fetchone()[0]
    print(f"\n📊 Total Sales Amount: ₹{total_sales if total_sales else 0}")


def sales_by_date_range(cursor):
    """Fetch sales data between a specific date range"""
    start_date = input("Enter start date (YYYY-MM-DD): ")
    end_date = input("Enter end date (YYYY-MM-DD): ")

    cursor.execute(
        "SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM orders WHERE
OrderDate BETWEEN %s AND %s",
        (start_date, end_date)
    )
    orders = cursor.fetchall()

    print("\n🗓️ Sales Report (Date Range):")
    for order in orders:
        print(f"🛒 Order ID: {order[0]}, Customer ID: {order[1]}, Date: {order[2]}, Amount:
₹{order[3]}")
```

```
def sales_by_customer(cursor):
    """Fetch total sales made by a specific customer"""
    customer_id = input("Enter Customer ID: ")

    cursor.execute(
        "SELECT SUM(TotalAmount) FROM orders WHERE CustomerID = %s",
        (customer_id,)
    )
    total_sales = cursor.fetchone()[0]

    print(f"\n👤 Customer {customer_id} Total Purchases: ₹{total_sales if total_sales else
0}")


def main():
    """Main function to run the sales report system"""
    conn = get_db_connection()
    if not conn:
        return

    cursor = conn.cursor()

    while True:
        print("\n📈 SALES REPORT MENU")
        print("1️⃣ View Total Sales")
        print("2️⃣ View Sales by Date Range")
        print("3️⃣ View Sales by Customer")
        print("4️⃣ Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            total_sales_report(cursor)
        elif choice == "2":
            sales_by_date_range(cursor)
        elif choice == "3":
            sales_by_customer(cursor)
        elif choice == "4":
            print("Exiting Sales Report...")
            break
        else:
            print("❌ Invalid choice. Please try again.")

    cursor.close()
```

```
    conn.close()


if __name__ == "__main__":
    main()
```







7: Customer Account Updates

Description: Customers may need to update their account information, such as changing their email address or phone number.

Task: Implement a user profile management feature with database connectivity to allow customers to update their account details. Ensure data validation and integrity.

Main.py:

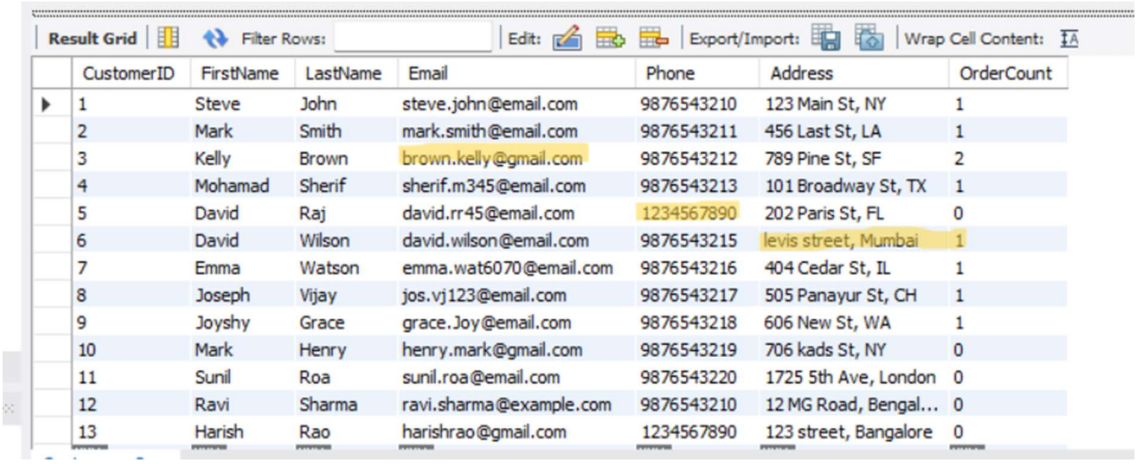
```
import mysql.connector
import re


def get_db_connection():
    """Establish database connection"""
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="#vijaysql**",
            database="TechGadgetShop"
        )
        return conn
    except mysql.connector.Error as e:
        print(f"Error connecting to database: {e}")
        return None


def validate_email(email):
    """Validate email format"""
    pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
    return re.match(pattern, email)
```

```
def validate_phone(phone):
    """Validate phone number (only digits, length 10-15)"""
    return phone.isdigit() and 10 <= len(phone) <= 15


def update_customer_profile(cursor, conn):
    """Update customer details"""
    customer_id = input("Enter your Customer ID: ")

    # Check if customer exists
    cursor.execute("SELECT * FROM customers WHERE CustomerID = %s",
(customer_id,))
    customer = cursor.fetchone()
    if not customer:
        print("❌ Customer not found.")
        return

    print("\n📌 Update Options:")
    print("1️⃣ Update Email")
    print("2️⃣ Update Phone")
    print("3️⃣ Update Address")
    choice = input("Enter choice: ")

    if choice == "1":
        new_email = input("Enter new Email: ")
        if not validate_email(new_email):
            print("❌ Invalid email format.")
            return
        cursor.execute("UPDATE customers SET Email = %s WHERE CustomerID = %s",
(new_email, customer_id))

    elif choice == "2":
        new_phone = input("Enter new Phone Number: ")
        if not validate_phone(new_phone):
            print("❌ Invalid phone number. Must be 10-15 digits.")
            return
        cursor.execute("UPDATE customers SET Phone = %s WHERE CustomerID = %s",
(new_phone, customer_id))

    elif choice == "3":
        new_address = input("Enter new Address: ")
        cursor.execute("UPDATE customers SET Address = %s WHERE CustomerID = %s",
(new_address, customer_id))
```

```
    else:
        print("❌ Invalid choice.")
        return

    conn.commit()
    print("✅ Customer details updated successfully.")


def main():
    """Main function to manage customer updates"""
    conn = get_db_connection()
    if not conn:
        return

    cursor = conn.cursor()

    while True:
        print("\n👤 CUSTOMER ACCOUNT MANAGEMENT")
        print("1️⃣ Update Account Information")
        print("2️⃣ Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            update_customer_profile(cursor, conn)
        elif choice == "2":
            print("Exiting Customer Account Management...")
            break
        else:
            print("❌ Invalid choice. Please try again.")

    cursor.close()
    conn.close()


if __name__ == "__main__":
    main()
```

```
👤 CUSTOMER ACCOUNT MANAGEMENT
1️⃣ Update Account Information
2️⃣ Exit
Enter your choice: 1
Enter your Customer ID: 3

📌 Update Options:
1️⃣ Update Email
2️⃣ Update Phone
3️⃣ Update Address
Enter choice: 1
Enter new Email: brown.kelly@gmail.com
✅ Customer details updated successfully.

👤 CUSTOMER ACCOUNT MANAGEMENT
1️⃣ Update Account Information
2️⃣ Exit
Enter your choice: 1
Enter your Customer ID: 5

📌 Update Options:
1️⃣ Update Email
2️⃣ Update Phone
3️⃣ Update Address
Enter choice: 2
Enter new Phone Number: 1234567890
✅ Customer details updated successfully.
```

```
👤 CUSTOMER ACCOUNT MANAGEMENT
1️⃣ Update Account Information
2️⃣ Exit
Enter your choice: 1
Enter your Customer ID: 6

📌 Update Options:
1️⃣ Update Email
2️⃣ Update Phone
3️⃣ Update Address
Enter choice: 3
Enter new Address: levis street, Mumbai
✅ Customer details updated successfully.
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	Email	Phone	Address	OrderCount
1	Steve	John	steve.john@email.com	9876543210	123 Main St, NY	1
2	Mark	Smith	mark.smith@email.com	9876543211	456 Last St, LA	1
3	Kelly	Brown	brown.kelly@gmail.com	9876543212	789 Pine St, SF	2
4	Mohamad	Sherif	sherif.m345@email.com	9876543213	101 Broadway St, TX	1
5	David	Raj	david.rr45@email.com	1234567890	202 Paris St, FL	0
6	David	Wilson	david.wilson@email.com	9876543215	levis street, Mumbai	1
7	Emma	Watson	emma.wat6070@email.com	9876543216	404 Cedar St, IL	1
8	Joseph	Vijay	jos.vj123@email.com	9876543217	505 Panayur St, CH	1
9	Joyshy	Grace	grace.Joy@email.com	9876543218	606 New St, WA	1
10	Mark	Henry	henry.mark@gmail.com	9876543219	706 kads St, NY	0
11	Sunil	Roa	sunil.roa@email.com	9876543220	1725 5th Ave, London	0
12	Ravi	Sharma	ravi.sharma@example.com	9876543210	12 MG Road, Bengal...	0
13	Harish	Rao	harishrao@gmail.com	1234567890	123 street, Bangalore	0

8: Payment Processing

Description: When customers make payments for their orders, the payment details (e.g., payment method, amount) must be recorded in the database.

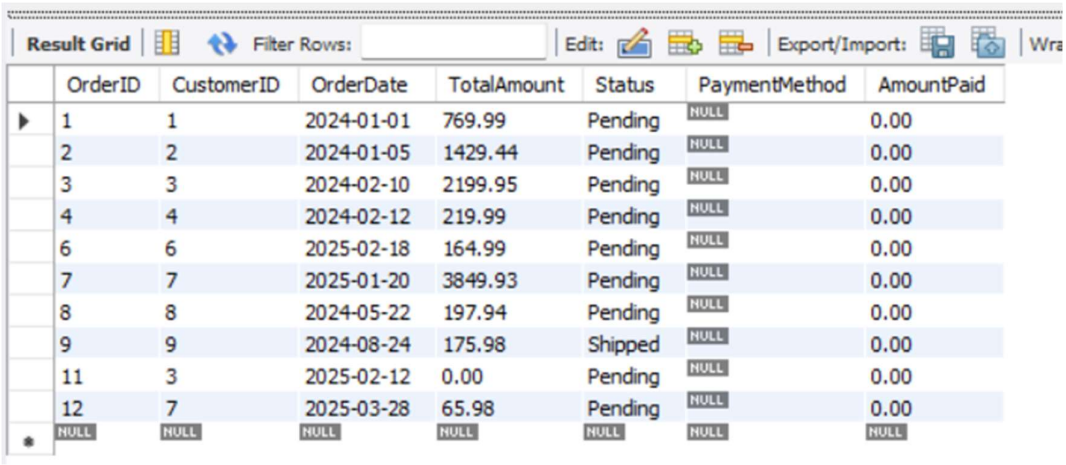
Task: Develop a payment processing system that interacts with the database to record payment transactions, validate payment information, and handle errors.

Main.py:

```
import mysql.connector


def get_db_connection():
    """Establish database connection"""
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="#vijaysql**",
            database="TechGadgetShop"
        )
        return conn
    except mysql.connector.Error as e:
        print(f"Error connecting to database: {e}")
        return None


def process_payment(cursor, conn):
    """Process a payment for an order"""
    order_id = input("Enter Order ID: ")

    # Check if order exists
    cursor.execute("SELECT TotalAmount, Status FROM orders WHERE OrderID = %s",
```

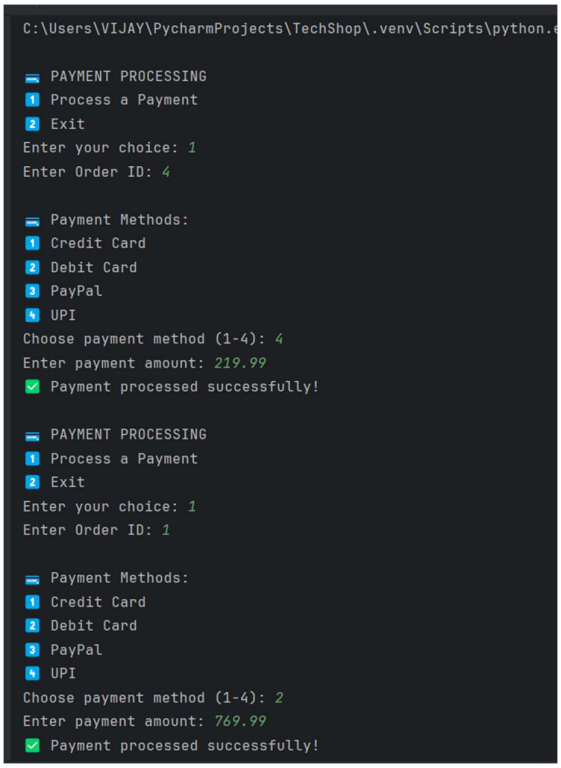
```
(order_id,))
    order = cursor.fetchone()

    if not order:
        print("❌ Order not found.")
        return

    total_amount, status = order
    if status == "Paid":
        print("✅ This order is already paid.")
        return

    print("\n💳 Payment Methods:")
    print("1️⃣ Credit Card")
    print("2️⃣ Debit Card")
    print("3️⃣ PayPal")
    print("4️⃣ UPI")

    method_choice = input("Choose payment method (1-4): ")
    payment_methods = {"1": "Credit Card", "2": "Debit Card", "3": "PayPal", "4": "UPI"}

    if method_choice not in payment_methods:
        print("❌ Invalid payment method.")
        return

    payment_method = payment_methods[method_choice]
    amount_paid = float(input("Enter payment amount: "))

    if amount_paid != float(total_amount):
        print(f"❌ Payment amount must match TotalAmount: {total_amount}")
        return

    # Update orders table with payment details
    cursor.execute("UPDATE orders SET PaymentMethod = %s, AmountPaid = %s, Status =
'Paid' WHERE OrderID = %s",
                   (payment_method, amount_paid, order_id))

    conn.commit()
    print("✅ Payment processed successfully!")


def main():
    """Main function for payment processing"""
    conn = get_db_connection()
    if not conn:
```

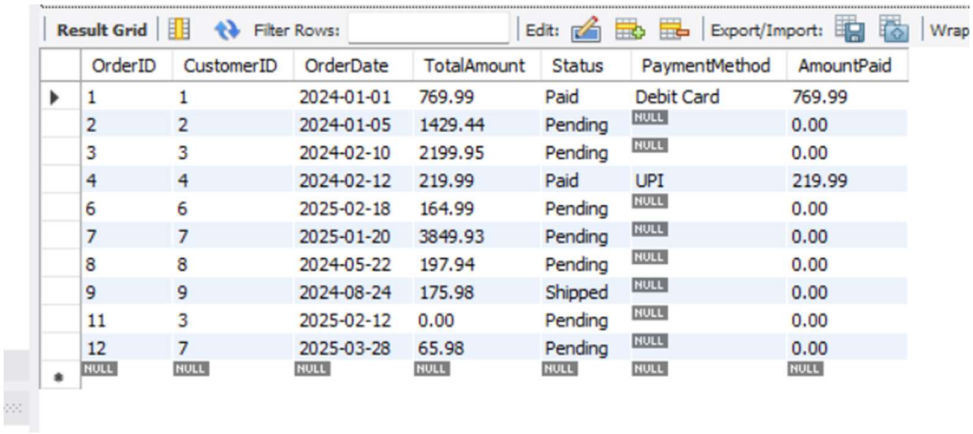
```
        return

    cursor = conn.cursor()

    while True:
        print("\n💳 PAYMENT PROCESSING")
        print("1️⃣ Process a Payment")
        print("2️⃣ Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            process_payment(cursor, conn)
        elif choice == "2":
            print("Exiting Payment Processing...")
            break
        else:
            print("❌ Invalid choice. Try again.")

    cursor.close()
    conn.close()


if __name__ == "__main__":
    main()
```



Result Grid | Filter Rows: | Edit: | Export/Import:

OrderID	CustomerID	OrderDate	TotalAmount	Status	PaymentMethod	AmountPaid
1	1	2024-01-01	769.99	Pending	NULL	0.00
2	2	2024-01-05	1429.44	Pending	NULL	0.00
3	3	2024-02-10	2199.95	Pending	NULL	0.00
4	4	2024-02-12	219.99	Pending	NULL	0.00
6	6	2025-02-18	164.99	Pending	NULL	0.00
7	7	2025-01-20	3849.93	Pending	NULL	0.00
8	8	2024-05-22	197.94	Pending	NULL	0.00
9	9	2024-08-24	175.98	Shipped	NULL	0.00
11	3	2025-02-12	0.00	Pending	NULL	0.00
12	7	2025-03-28	65.98	Pending	NULL	0.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL





Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

OrderID	CustomerID	OrderDate	TotalAmount	Status	PaymentMethod	AmountPaid
1	1	2024-01-01	769.99	Paid	Debit Card	769.99
2	2	2024-01-05	1429.44	Pending	NULL	0.00
3	3	2024-02-10	2199.95	Pending	NULL	0.00
4	4	2024-02-12	219.99	Paid	UPI	219.99
6	6	2025-02-18	164.99	Pending	NULL	0.00
7	7	2025-01-20	3849.93	Pending	NULL	0.00
8	8	2024-05-22	197.94	Pending	NULL	0.00
9	9	2024-08-24	175.98	Shipped	NULL	0.00
11	3	2025-02-12	0.00	Pending	NULL	0.00
12	7	2025-03-28	65.98	Pending	NULL	0.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

9: Product Search and Recommendations

Description: Customers should be able to search for products based on various criteria (e.g., name, category) and receive product recommendations.

Task: Implement a product search and recommendation engine that uses database connectivity to retrieve relevant product information.

Main.py:

```
import mysql.connector


# Database Connection
def connect_db():
```

```
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="#vijaysql**",
            database="TechGadgetShop"
        )
        return conn
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None


# Function to Search Products
def search_products():
    conn = connect_db()
    if not conn:
        return
    cursor = conn.cursor()

    search_query = input("Enter product name or keyword to search: ")
    query = """SELECT ProductID, ProductName, Price, Description
               FROM products
               WHERE ProductName LIKE %s OR Description LIKE %s"""

    cursor.execute(query, (f"%{search_query}%", f"%{search_query}%"))
    results = cursor.fetchall()

    if not results:
        print("No matching products found.")
    else:
        print("\n🔍 Search Results:")
        for row in results:
            print(f"ID: {row[0]}, Name: {row[1]}, Price: ₹{row[2]}, Description: {row[3]}")

        # Fetch recommendations
        recommend_products(search_query, cursor)

    cursor.close()
    conn.close()


# Function to Recommend Products Based on Description
def recommend_products(search_query, cursor):
    query = """SELECT ProductID, ProductName, Price, Description
               FROM products
```

```
            WHERE Description LIKE %s
            LIMIT 3"""
    cursor.execute(query, (f"%{search_query}%",))
    recommendations = cursor.fetchall()

    if recommendations:
        print("\n⭐ Recommended Products:")
        for row in recommendations:
            print(f"ID: {row[0]}, Name: {row[1]}, Price: ₹{row[2]}, Description: {row[3]}")


# Main Menu
def main():
    while True:
        print("\n1. Search Products\n2. Exit")
        choice = input("Enter choice: ")

        if choice == "1":
            search_products()
        elif choice == "2":
            print("Exiting...")
            break
        else:
            print("Invalid choice. Try again.")


if __name__ == "__main__":
    main()
```

```
C:\Users\VIJAY\PycharmProjects\TechShop\.venv\Scripts\python.exe C:\Users\VIJAY\PycharmProjects\TechShop\.venv\main.py

1. Search Products
2. Exit
Enter choice: 1
Enter product name or keyword to search: Airpods

🔍 Search Results:
ID: 12, Name: Airpods, Price: ₹2999.00, Description: It has Bluetooth 5.3

1. Search Products
2. Exit
Enter choice: 1
Enter product name or keyword to search: bluetooth

🔍 Search Results:
ID: 5, Name: Bluetooth Speaker, Price: ₹109.99, Description: Portable speaker
ID: 12, Name: Airpods, Price: ₹2999.00, Description: It has Bluetooth 5.3

⭐ Recommended Products:
ID: 12, Name: Airpods, Price: ₹2999.00, Description: It has Bluetooth 5.3
```