Assignment

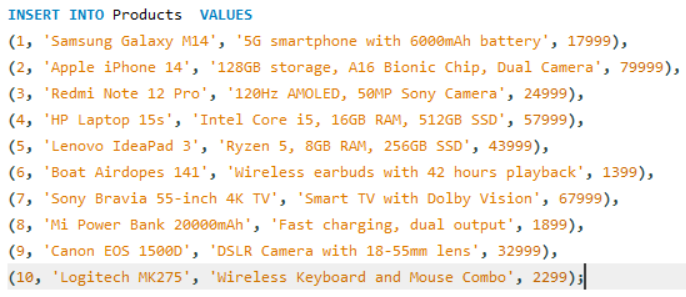
Task-1

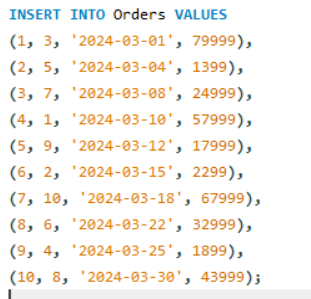
1. Create the database named "TechShop"

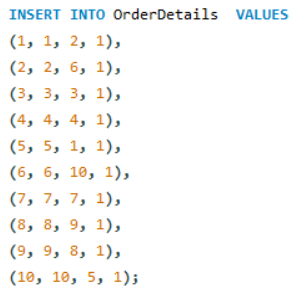
Create database TechShop;

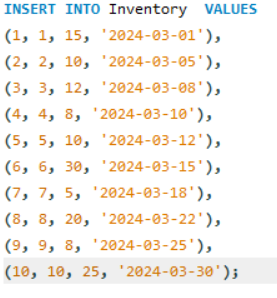
2. Insert at least 10 sample records into each of the following tables. a. Customers b. Products c. Orders d. OrderDetails, e. Inventory











3. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

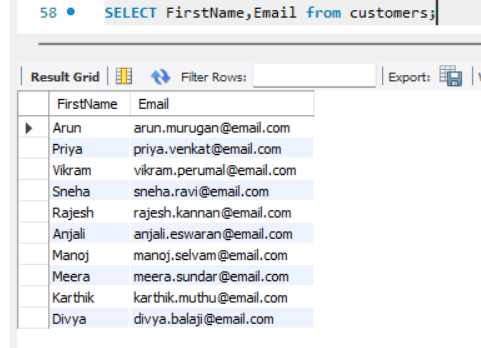
**Table:customers**  
  
**Columns:**

|  |  |
| --- | --- |
| **CustomerId** | int AI PK |
| FirstName | varchar(45) |
| LastName | varchar(45) |
| **Email** | varchar(50) |
| **Phone** | varchar(15) |
| Address | varchar(255) |
| **Table:orders**  **Columns:**   |  |  | | --- | --- | | **OrderId** | int AI PK | | **CustomerId** | int | | OrderDate | date | | TotalAmount | decimal(10,2) | | Status | varchar(20) |   **Table:products**  **Columns:**   |  |  | | --- | --- | | **ProductID** | int AI PK | | ProductName | varchar(100) | | Description | text | | Price | decimal(10,2) |   **Table:orderdetails**  **Columns:**   |  |  | | --- | --- | | **OrderDetailID** | int AI PK | | **OrderID** | int | | **ProductID** | int | | Quantity | int |   **Table:inventory**  **Columns:**   |  |  | | --- | --- | | **InventoryID** | int AI PK | | **ProductID** | int | | QuantityInStock | int | | LastStockUpdate | date | |  |

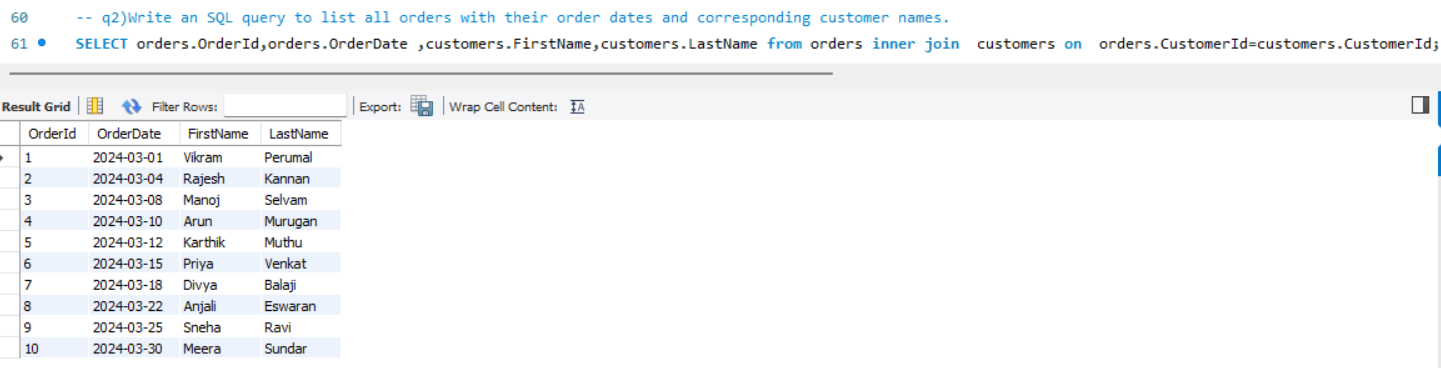
Task -2 :Select, Where, Between, AND, LIKE:

1. Write an SQL query to retrieve the names and emails of all customers.

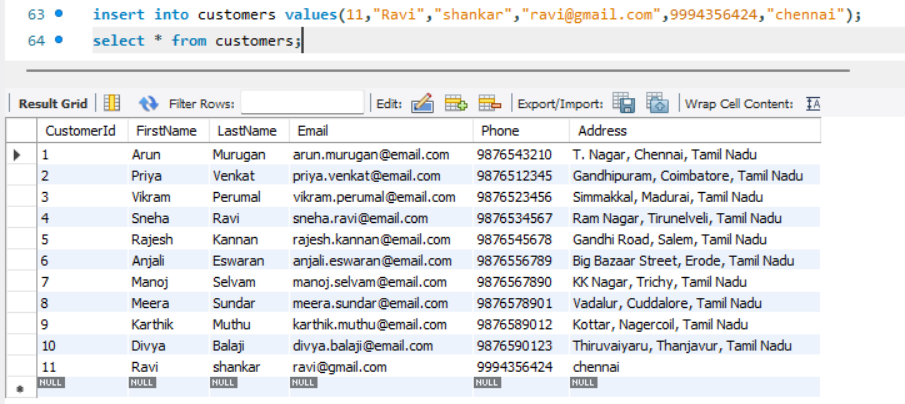
Query : SELECT firstname,email from customers;



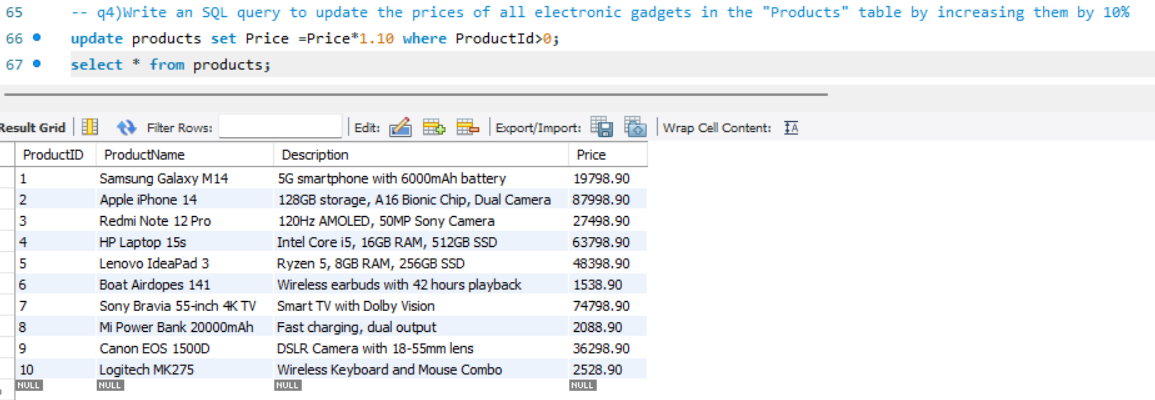
2. Write an SQL query to list all orders with their order dates and corresponding customer names.



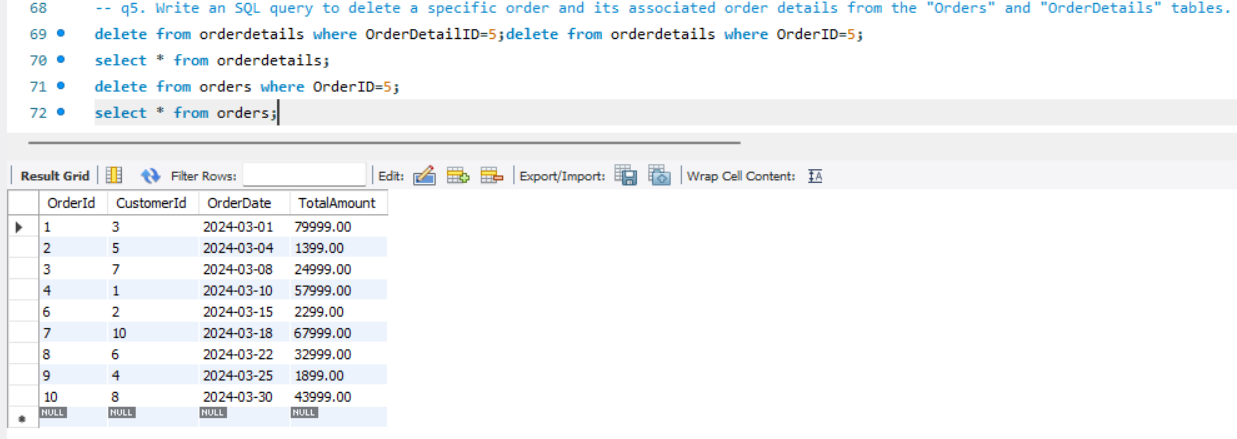
3.Write an SQL query to insert a new customer record into the "Customers" table. Include customer information such as name, email, and address



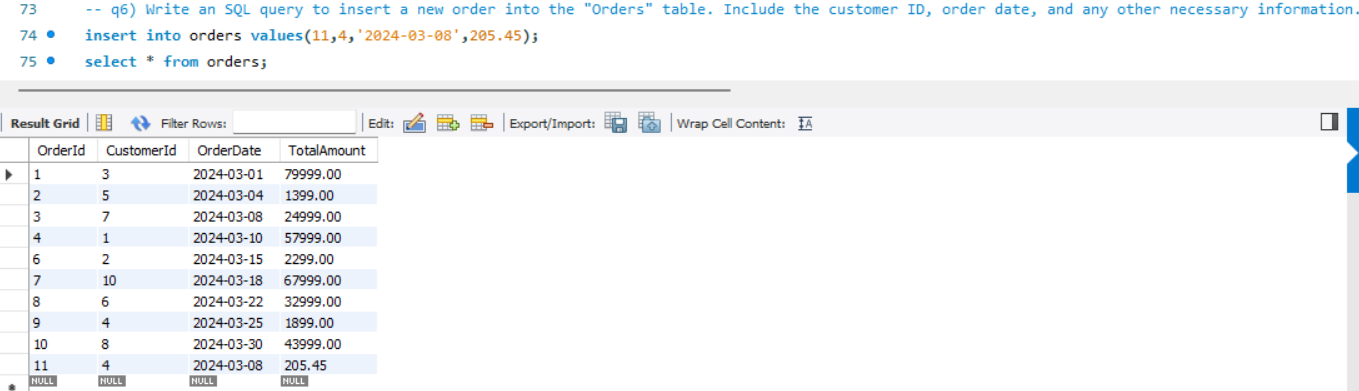
4) Write an SQL query to update the prices of all electronic gadgets in the "Products" table by increasing them by 10%.



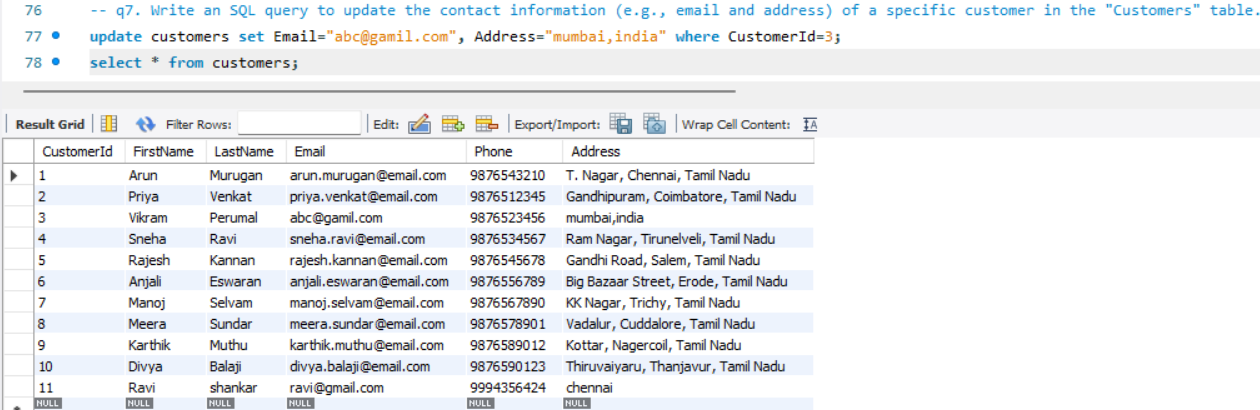
5) Write an SQL query to delete a specific order and its associated order details from the "Orders" and "OrderDetails" tables. Allow users to input the order ID as a parameter.



6. Write an SQL query to insert a new order into the "Orders" table. Include the customer ID, order date, and any other necessary information.

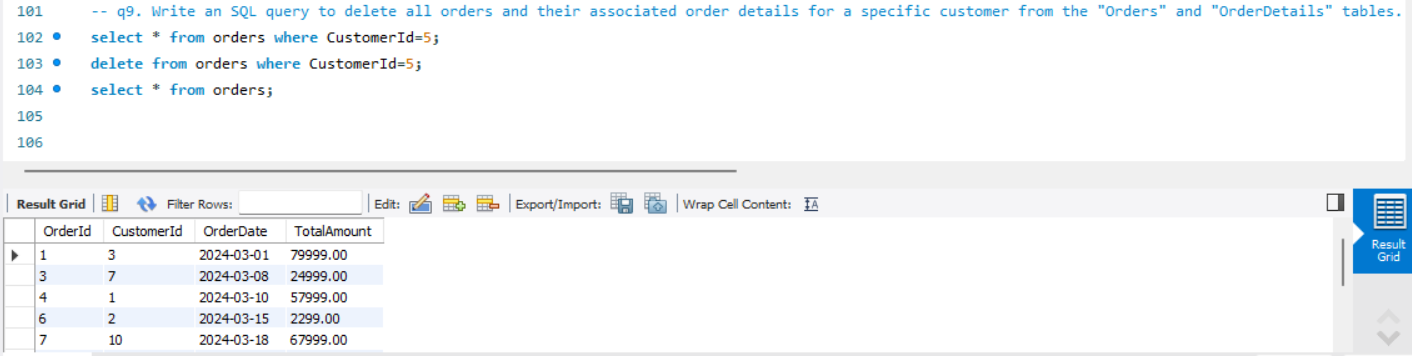


7. Write an SQL query to update the contact information (e.g., email and address) of a specific customer in the "Customers" table. Allow users to input the customer ID and new contact information.

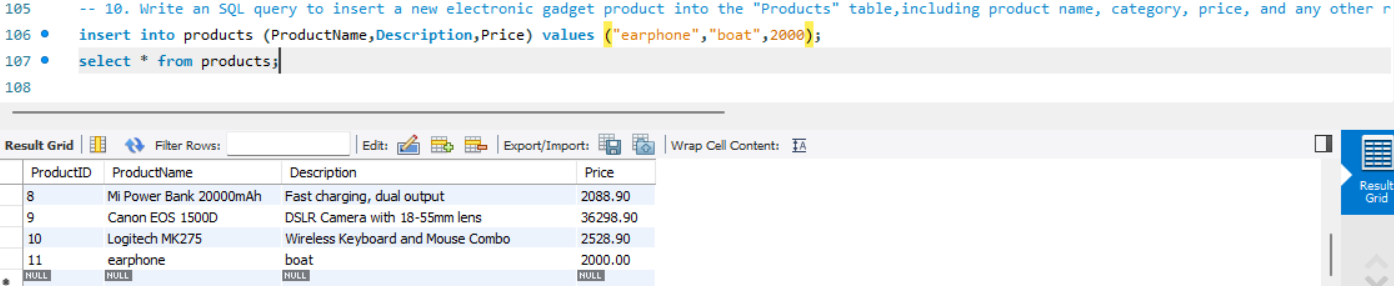


8. Write an SQL query to recalculate and update the total cost of each order in the "Orders" table based on the prices and quantities in the "OrderDetails" table.

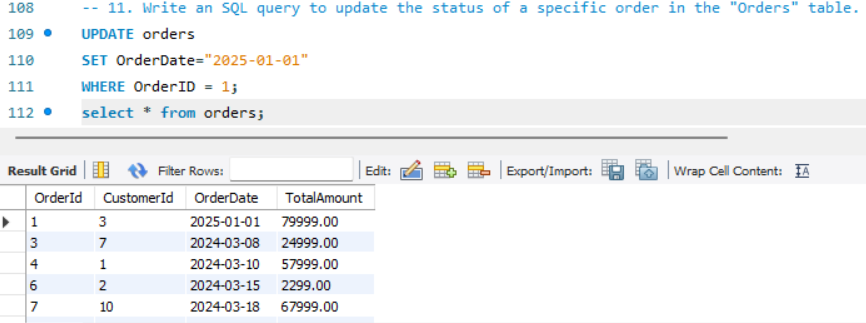
9. Write an SQL query to delete all orders and their associated order details for a specific customer from the "Orders" and "OrderDetails" tables.



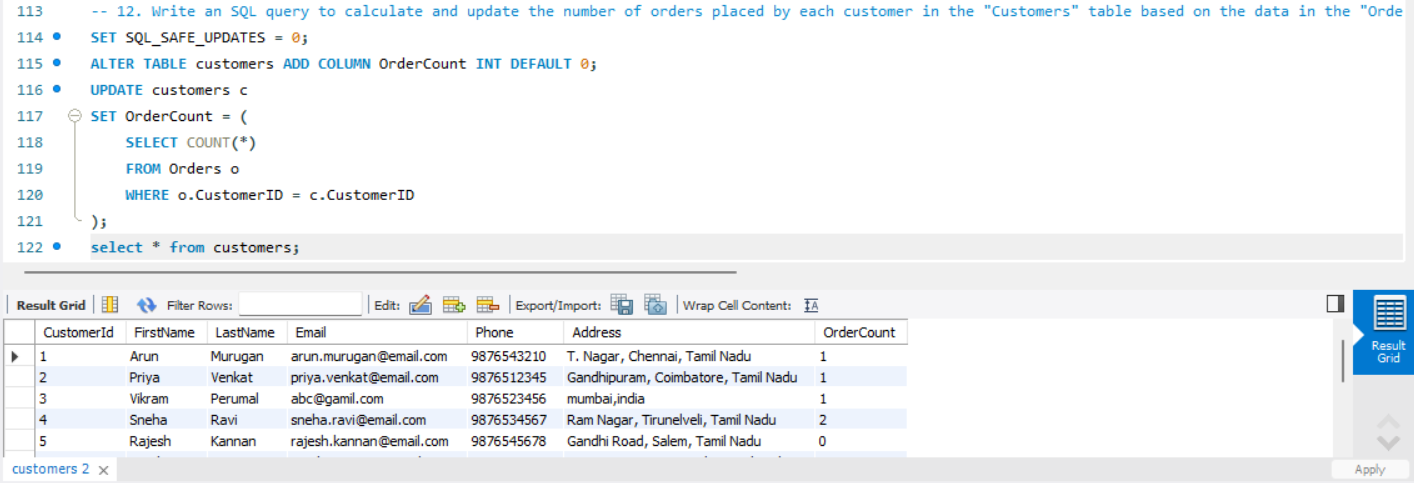
10. Write an SQL query to insert a new electronic gadget product into the "Products" table, including product name, category, price, and any other relevant details.



11. Write an SQL query to update the status of a specific order in the "Orders" table (e.g., from "Pending" to "Shipped"). Allow users to input the order ID and the new status.

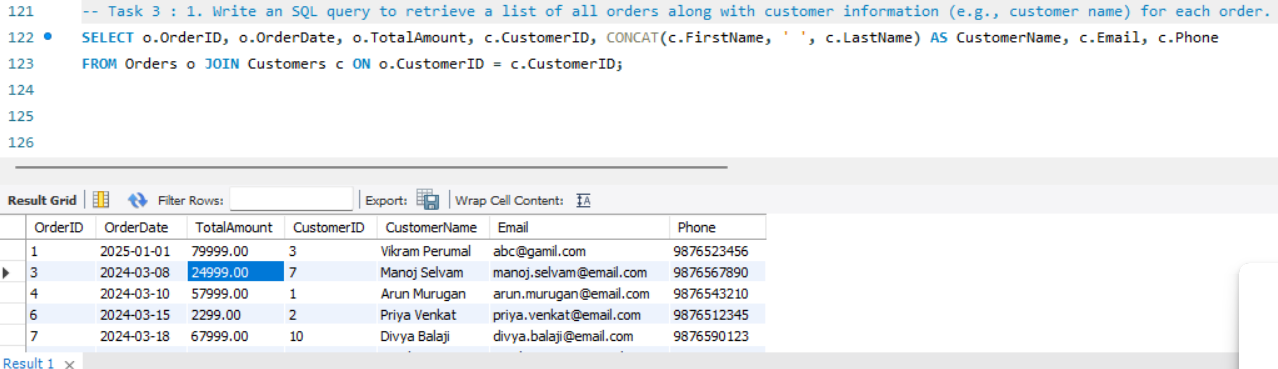


12. Write an SQL query to calculate and update the number of orders placed by each customer in the "Customers" table based on the data in the "Orders" table.

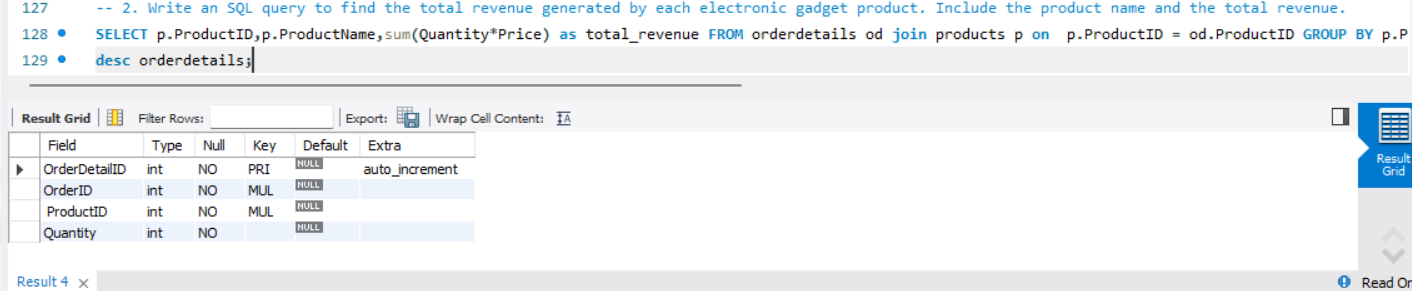


Task -3

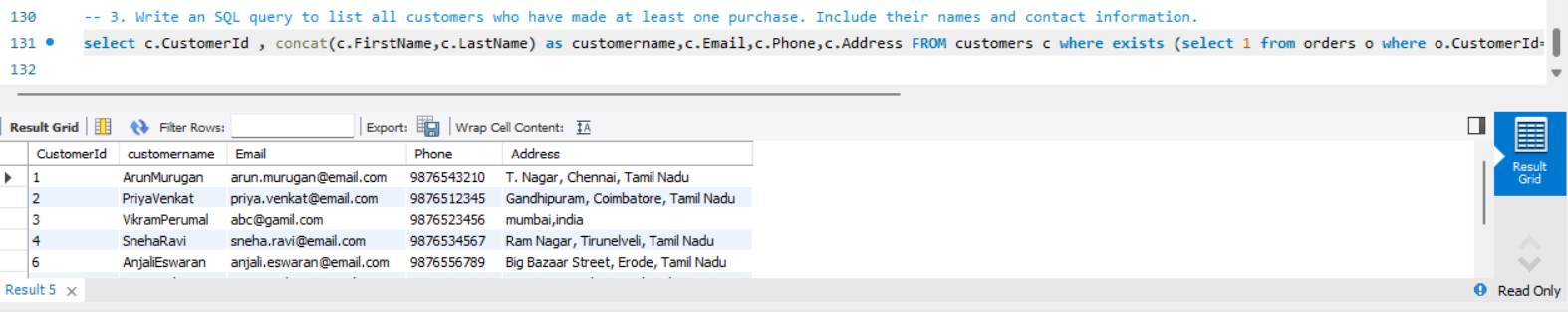
1. Write an SQL query to retrieve a list of all orders along with customer information (e.g., customer name) for each order.



1. Write an SQL query to find the total revenue generated by each electronic gadget product. Include the product name and the total revenue.



3. Write an SQL query to list all customers who have made at least one purchase. Include their names and contact information.



4.Write an SQL query to find the most popular electronic gadget, which is the one with the highest total quantity ordered. Include the product name and the total quantity ordered.

SELECT p.ProductName, SUM(od.Quantity) AS TotalQuantityOrdered

FROM orderdetails od

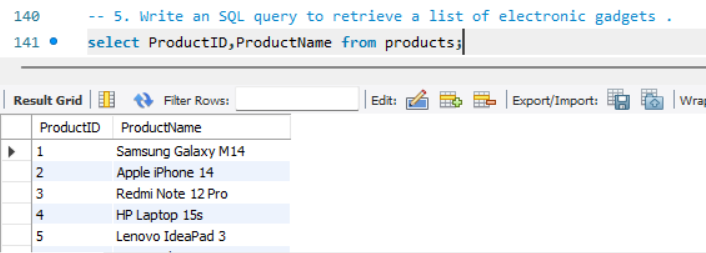
JOIN products p ON od.ProductID = p.ProductID

GROUP BY p.ProductID, p.ProductName

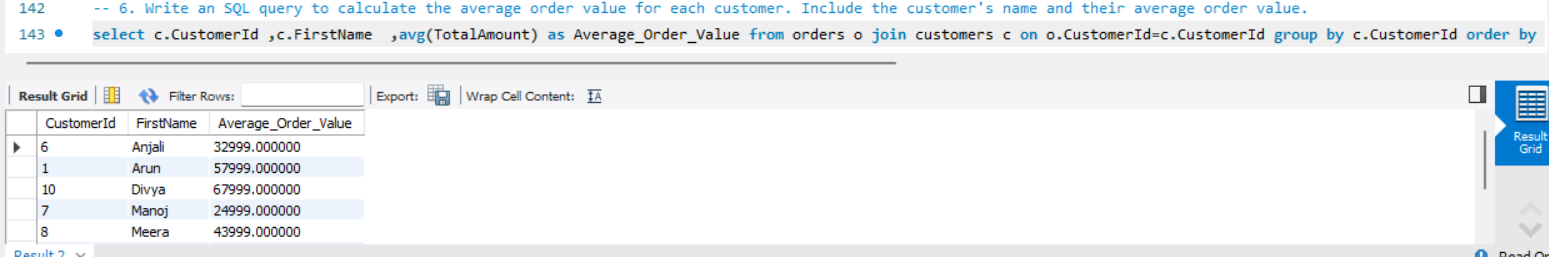
ORDER BY TotalQuantityOrdered DESC

LIMIT 1;

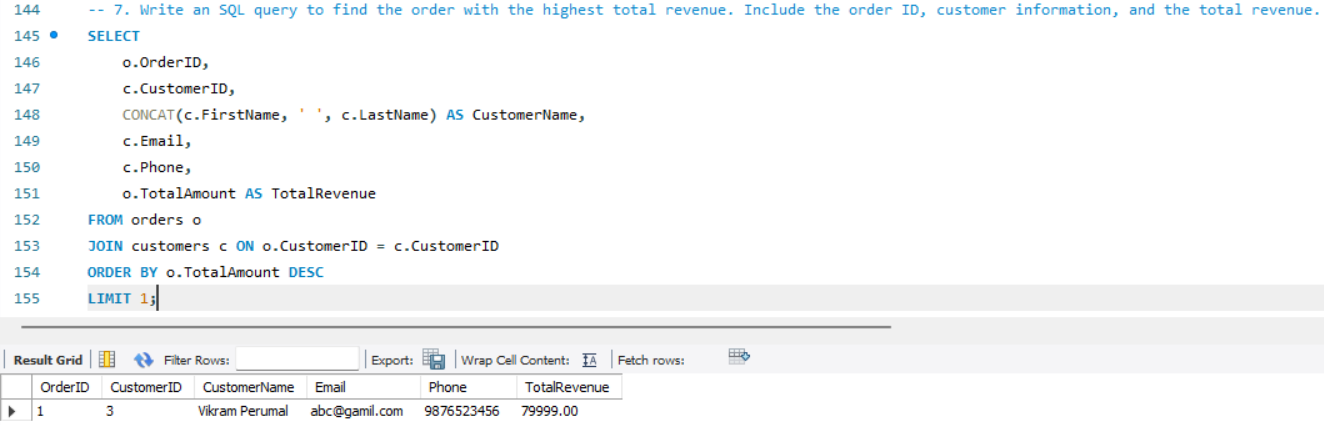
5. Write an SQL query to retrieve a list of electronic gadgets



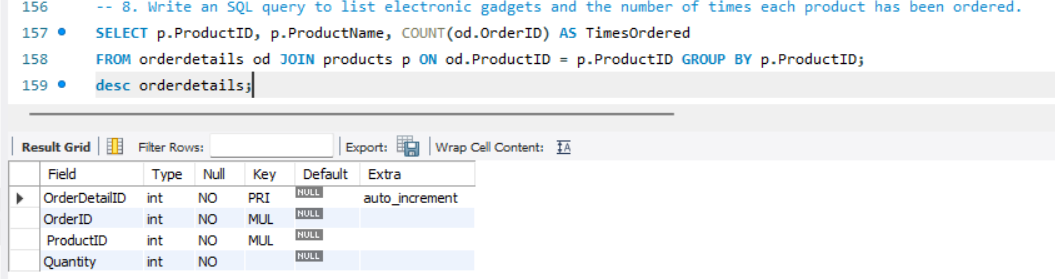
6. Write an SQL query to calculate the average order value for each customer. Include the customer's name and their average order value.



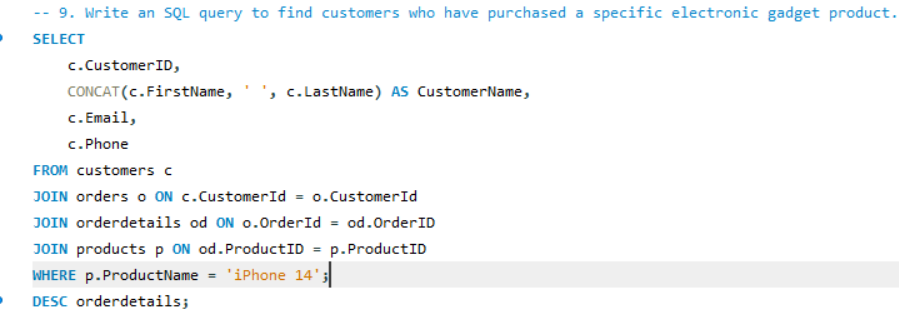
7. Write an SQL query to find the order with the highest total revenue. Include the order ID, customer information, and the total revenue.



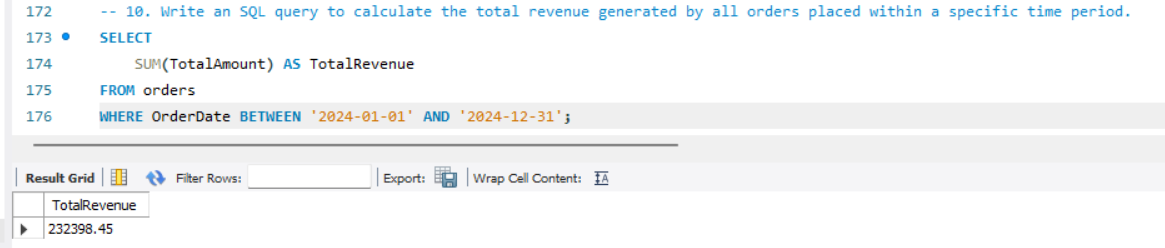
8. Write an SQL query to list electronic gadgets and the number of times each product has been ordered.



9. Write an SQL query to find customers who have purchased a specific electronic gadget product.

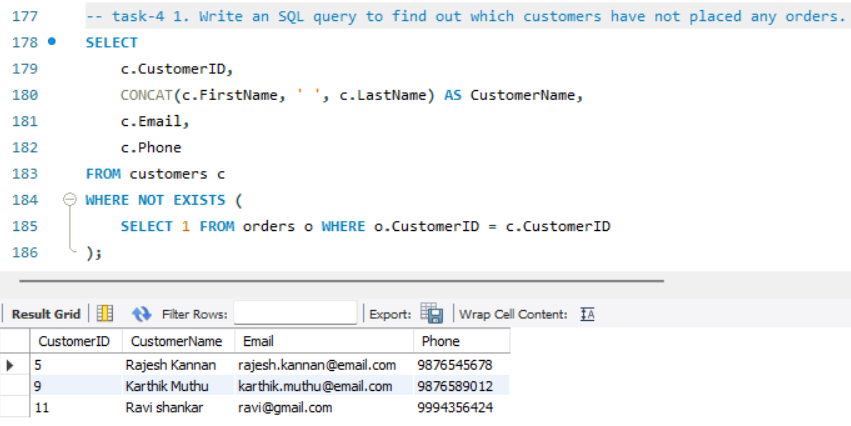


10. Write an SQL query to calculate the total revenue generated by all orders placed within a specific time period.

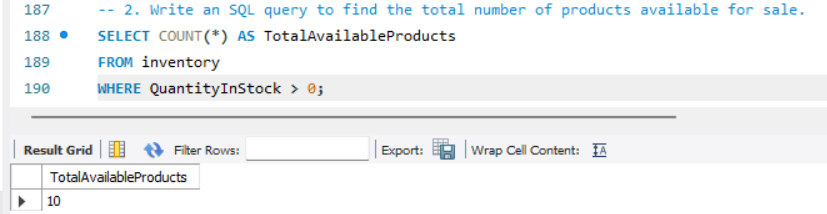


TASK -4

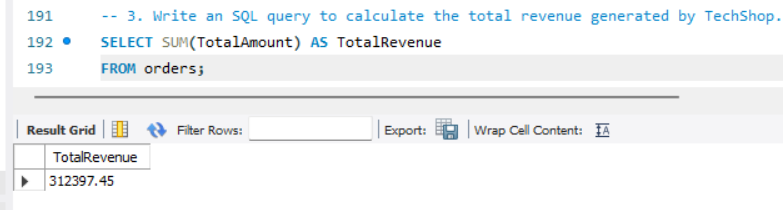
1.Write an SQL query to find out which customers have not placed any orders.



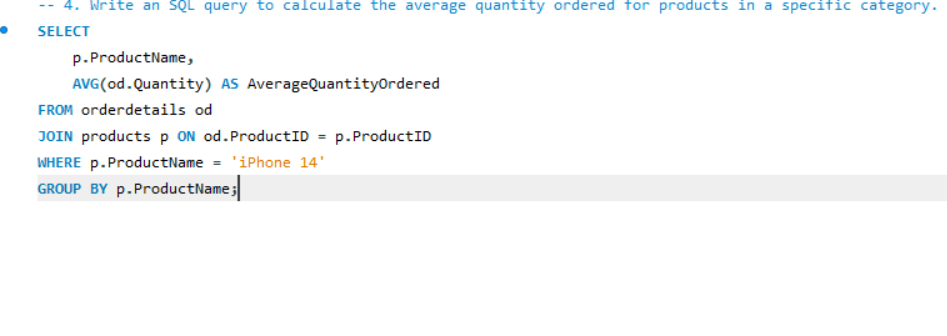
2.Write an SQL query to find the total number of products available for sale.



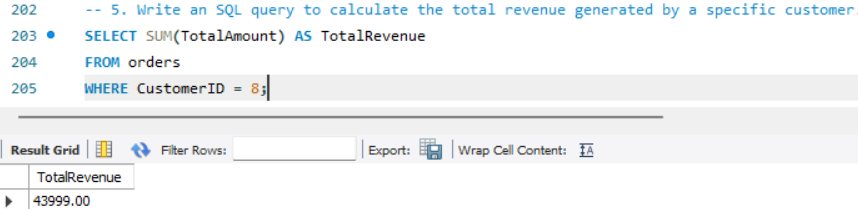
3.Write an SQL query to calculate the total revenue generated by TechShop.



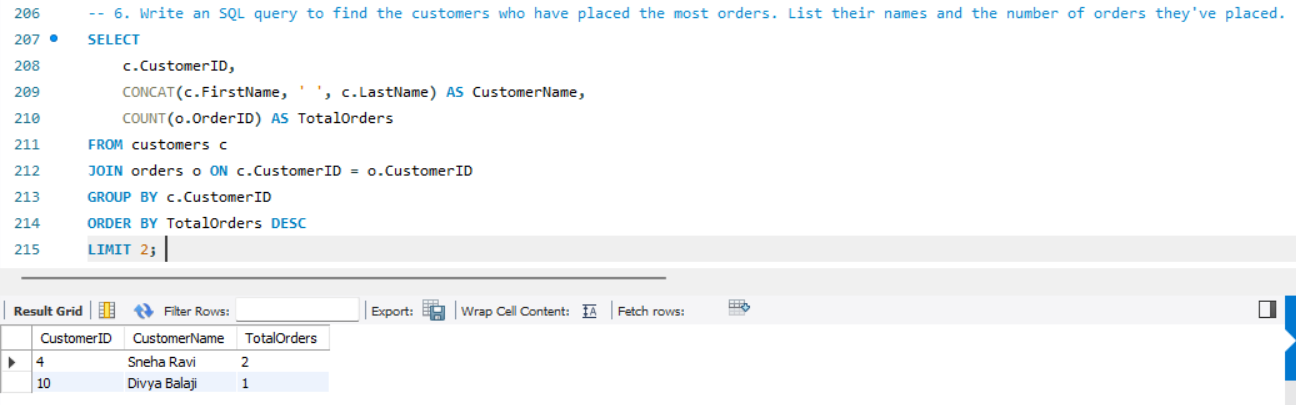
4. Write an SQL query to calculate the average quantity ordered for products in a specific category. Allow users to input the category name as a parameter.



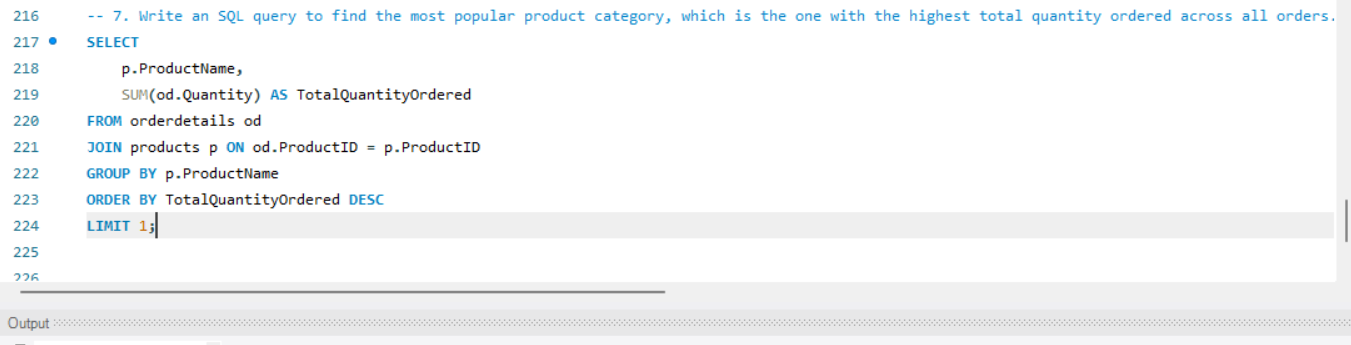
5. Write an SQL query to calculate the total revenue generated by a specific customer. Allow users to input the customer ID as a parameter.



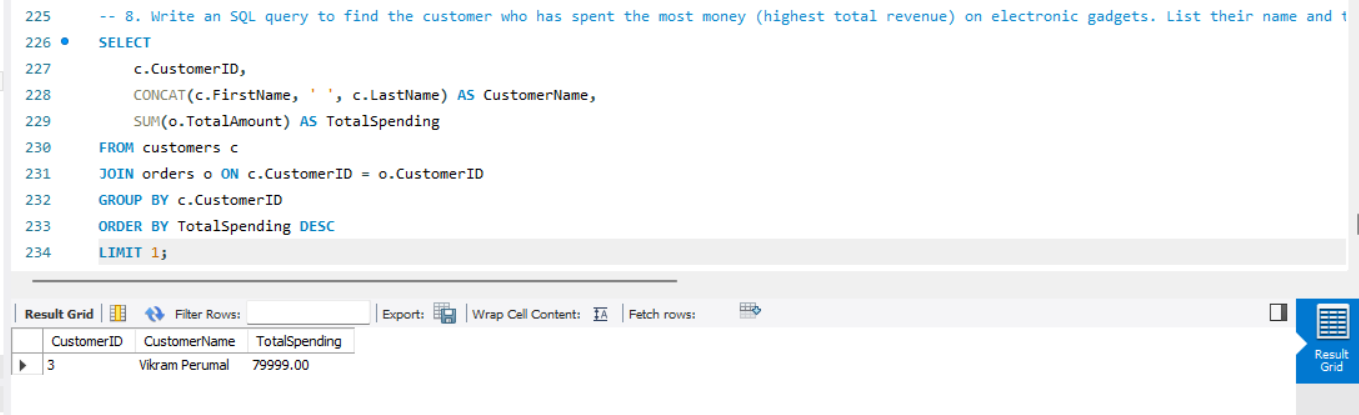
6. Write an SQL query to find the customers who have placed the most orders. List their names and the number of orders they've placed.



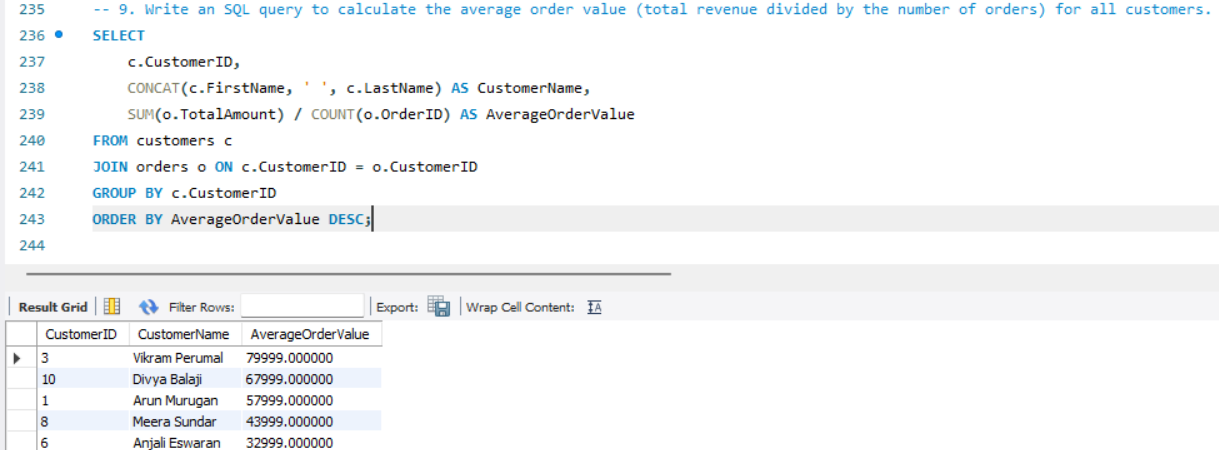
7. Write an SQL query to find the most popular product category, which is the one with the highest total quantity ordered across all orders.



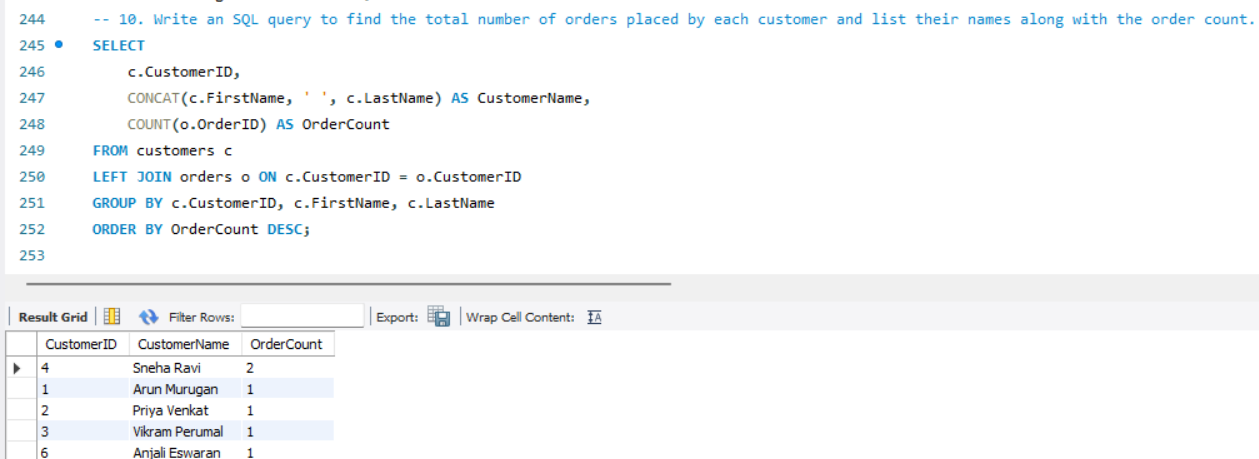
8. Write an SQL query to find the customer who has spent the most money (highest total revenue) on electronic gadgets. List their name and total spending.



9. Write an SQL query to calculate the average order value (total revenue divided by the number of orders) for all customers.



10. Write an SQL query to find the total number of orders placed by each customer and list their names along with the order count.



**Implement OOPs**

**Customer entity**

package entity;

public class Customer {

private int customerId;

private String firstName;

private String lastName;

private String email;

private String phone;

private String address;

public Customer(int customerId, String firstName, String lastName, String email, String phone, String address) {

super();

this.customerId = customerId;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phone = phone;

this.address = address;

}

public int getCustomerId() {

return customerId;

}

public void setCustomerId(int customerId) {

this.customerId = customerId;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

@Override

public String toString() {

return "Customer {" +

"ID=" + customerId +

", Name='" + firstName + " " + lastName + '\'' +

", Email='" + email + '\'' +

", Phone='" + phone + '\'' +

", Address='" + address + '\'' +

'}';

}

}

**Orders entity**

package entity;

import java.time.LocalDate;

public class Orders {

private int orderId;

private Customer customer;

private LocalDate orderDate;

private double totalAmount;

private String status;

public Orders() {

}

public Orders(int orderId, Customer customer, LocalDate orderDate, double totalAmount, String status) {

super();

this.orderId = orderId;

this.customer = customer;

this.orderDate = orderDate;

this.totalAmount = totalAmount;

this.status = status;

}

public int getOrderId() {

return orderId;

}

public void setOrderId(int orderId) {

this.orderId = orderId;

}

public Customer getCustomer() {

return customer;

}

public void setCustomer(Customer customer) {

this.customer = customer;

}

public LocalDate getOrderDate() {

return orderDate;

}

public void setOrderDate(LocalDate orderDate) {

this.orderDate = orderDate;

}

public double getTotalAmount() {

return totalAmount;

}

public void setTotalAmount(double totalAmount) {

this.totalAmount = totalAmount;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

@Override

public String toString() {

return "Order {" +

"ID=" + orderId +

", Customer=" + (customer != null ? customer.getFirstName() + " " + customer.getLastName() : "N/A") +

", Order Date=" + orderDate +

", Total Amount=" + totalAmount +

'}';

}

}

**Orderdetails entity**

package entity;

public class OrderDetails {

private int orderDetailID;

private Orders order;

private Product product;

private int quantity;

public OrderDetails() {

}

public int getOrderDetailID() {

return orderDetailID;

}

public void setOrderDetailID(int orderDetailID) {

this.orderDetailID = orderDetailID;

}

public Orders getOrder() {

return order;

}

public void setOrder(Orders order) {

this.order = order;

}

public Product getProduct() {

return product;

}

public void setProduct(Product product) {

this.product = product;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public OrderDetails(int orderDetailID, Orders order, Product product, int quantity) {

super();

this.orderDetailID = orderDetailID;

this.order = order;

this.product = product;

this.quantity = quantity;

}

@Override

public String toString() {

return "OrderDetailID: " + orderDetailID +

", Order: " + order +

", Product: " + product +

", Quantity: " + quantity;

}

}

**Product entity**

**package entity;**

public class Product {

private int productId;

private String productName;

private String description;

private double price;

public Product() {

}

public Product(int productId, String productName, String description, double price) {

super();

this.productId = productId;

this.productName = productName;

this.description = description;

this.price = price;

}

public Product(int productId, String productName, double price) {

this.productId = productId;

this.productName = productName;

this.price = price;

}

public int getProductId() {

return productId;

}

public void setProductId(int productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

@Override

public String toString() {

return "Product {" +

"ID=" + productId +

", Name='" + productName + '\'' +

", Description='" + description + '\'' +

", Price=" + price +

'}';

}

}

**Inventory entity**

package entity;

import java.util.Date;

public class Inventory {

private int inventoryID;

private Product product;

private int quantityInStock;

private Date lastStockUpdate;

public Inventory() {

}

public int getInventoryID() {

return inventoryID;

}

public void setInventoryID(int inventoryID) {

this.inventoryID = inventoryID;

}

public Product getProduct() {

return product;

}

public void setProduct(Product product) {

this.product = product;

}

public int getQuantityInStock() {

return quantityInStock;

}

public void setQuantityInStock(int quantityInStock) {

this.quantityInStock = quantityInStock;

}

public Date getLastStockUpdate() {

return lastStockUpdate;

}

public void setLastStockUpdate(Date lastStockUpdate) {

this.lastStockUpdate = lastStockUpdate;

}

public Inventory(int inventoryID, Product product, int quantityInStock, Date lastStockUpdate) {

super();

this.inventoryID = inventoryID;

this.product = product;

this.quantityInStock = quantityInStock;

this.lastStockUpdate = lastStockUpdate;

}

@Override

public String toString() {

return "InventoryID: " + inventoryID +

", Product: " + product +

", Quantity In Stock: " + quantityInStock +

", Last Stock Update: " + lastStockUpdate;

}

}

**UTIL PACKAGE**

**DBConnUtil**

package util;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import exception.DatabaseConnectionException;

public class DBConnUtil {

private static final String fileName = "db.properties";

public static Connection getDbConnection() throws DatabaseConnectionException {

Connection con = null;

String connString = null;

try {

connString = DBPropertyUti.getConnectionString(fileName);

} catch (IOException e) {

throw new DatabaseConnectionException("Connection String Creation Failed", e);

}

if (connString != null) {

//System.out.println("Connection String: " + connString);

try {

con = DriverManager.getConnection(connString);

} catch (SQLException e) {

throw new DatabaseConnectionException("Error While Establishing DBConnection........", e);

}

}

return con;

}

}

**DBPropertyUti**

package util;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

public class DBPropertyUti {

public static String getConnectionString(String fileName)throws IOException {

//fileName="db.properties"

String connStr=null;

Properties props=new Properties();

FileInputStream fis=new FileInputStream(fileName);

props.load(fis);

String user=props.getProperty("user");

String password=props.getProperty("password");

String protocol=props.getProperty("protocol");

String system=props.getProperty("system");

String database=props.getProperty("database");

String port=props.getProperty("port");

connStr = protocol + "//" + system + ":" + port + "/" + database + "?user=" + user + "&password=" + password;

//connStr=protocol+"//"+system+":"+port+"/"+database+"?user="+user+"&password="+password;

return connStr;

}

}

**DAO PACKAGE**

**ICustomer service**

package dao;

import java.util.List;

import entity.Customer;

import exception.CustomerNotFoundException;

import exception.InvalidDataException;

import entity.Product;

public interface ICustomerService {

boolean insertCustomer(Customer customer) throws InvalidDataException;

boolean updateCustomerInfo(int customerId, int fieldToUpdate, String newValue)

throws CustomerNotFoundException, InvalidDataException;

int calculateTotalOrders(int customerId) throws CustomerNotFoundException;

Customer getCustomerById(int customerId) throws CustomerNotFoundException;

List<Product> getAvailableProducts();

}

**Customer service**

**package** dao;

**import** entity.Customer;

**import** entity.Product;

**import** exception.CustomerNotFoundException;

**import** exception.DatabaseConnectionException;

**import** exception.InvalidDataException;

**import** util.DBConnUtil;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** CustomerService **implements** ICustomerService {

**private** Connection con;

**public** CustomerService() **throws** DatabaseConnectionException {

**super**();

con = DBConnUtil.*getDbConnection*();

}

@Override

**public** **boolean** insertCustomer(Customer customer) **throws** InvalidDataException {

validateCustomer(customer);

**boolean** inserted = **false**;

**try** {

PreparedStatement pstmt = con.prepareStatement(

"INSERT INTO customers (FirstName, LastName, Email, Phone, Address) VALUES (?, ?, ?, ?, ?)"

);

pstmt.setString(1, customer.getFirstName());

pstmt.setString(2, customer.getLastName());

pstmt.setString(3, customer.getEmail());

pstmt.setString(4, customer.getPhone());

pstmt.setString(5, customer.getAddress());

**int** rows = pstmt.executeUpdate();

inserted = rows > 0;

} **catch** (SQLException e) {

System.***out***.println("Error inserting customer: " + e.getMessage());

}

**return** inserted;

}

@Override

**public** **boolean** updateCustomerInfo(**int** customerId, **int** fieldToUpdate, String newValue)

**throws** CustomerNotFoundException, InvalidDataException {

String sql = "";

**switch** (fieldToUpdate) {

**case** 1:

**if** (!newValue.matches("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,6}$")) {

**throw** **new** InvalidDataException("Invalid email format.");

}

sql = "UPDATE customers SET Email = ? WHERE CustomerId= ?";

**break**;

**case** 2:

**if** (!newValue.matches("^[0-9]{10}$")) {

**throw** **new** InvalidDataException("Phone number must be 10 digits.");

}

sql = "UPDATE customers SET Phone = ? WHERE CustomerId = ?";

**break**;

**case** 3:

sql = "UPDATE customers SET Address = ? WHERE CustomerId = ?";

**break**;

**default**:

**throw** **new** InvalidDataException("Invalid field. Choose only from: email, phone, or address.");

}

**boolean** updated = **false**;

**try** {

PreparedStatement pstmt = con.prepareStatement(sql);

pstmt.setString(1, newValue);

pstmt.setInt(2, customerId);

**int** rows = pstmt.executeUpdate();

**if** (rows == 0) {

**throw** **new** CustomerNotFoundException("Customer with ID " + customerId + " not found.");

}

updated = **true**;

} **catch** (SQLException e) {

System.***out***.println("Error updating customer: " + e.getMessage());

}

**return** updated;

}

@Override

**public** **int** calculateTotalOrders(**int** customerId) **throws** CustomerNotFoundException {

**int** totalOrders = 0;

**try** {

PreparedStatement pstmt = con.prepareStatement("SELECT COUNT(\*) FROM orders WHERE CustomerId = ?");

pstmt.setInt(1, customerId);

ResultSet rs = pstmt.executeQuery();

**if** (rs.next()) {

totalOrders = rs.getInt(1);

}

} **catch** (SQLException e) {

System.***out***.println("Error fetching total orders: " + e.getMessage());

}

**if** (totalOrders == 0) {

**throw** **new** CustomerNotFoundException("No orders found for customer ID: " + customerId);

}

**return** totalOrders;

}

@Override

**public** Customer getCustomerById(**int** customerId) **throws** CustomerNotFoundException {

Customer customer = **null**;

**try** {

PreparedStatement pstmt = con.prepareStatement("SELECT \* FROM customers WHERE CustomerId = ?");

pstmt.setInt(1, customerId);

ResultSet rs = pstmt.executeQuery();

**if** (rs.next()) {

customer = **new** Customer(

rs.getInt("CustomerId"),

rs.getString("FirstName"),

rs.getString("LastName"),

rs.getString("Email"),

rs.getString("Phone"),

rs.getString("Address")

);

} **else** {

**throw** **new** CustomerNotFoundException("Customer with ID " + customerId + " not found.");

}

} **catch** (SQLException e) {

System.***out***.println("Error fetching customer: " + e.getMessage());

}

**return** customer;

}

**private** **void** validateCustomer(Customer customer) **throws** InvalidDataException {

**if** (!customer.getEmail().matches("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,6}$")) {

**throw** **new** InvalidDataException("Invalid email format.");

}

**if** (!customer.getPhone().matches("^[0-9]{10}$")) {

**throw** **new** InvalidDataException("Phone number must be 10 digits.");

}

}

@Override

**public** List<Product> getAvailableProducts() {

List<Product> availableProducts = **new** ArrayList<>();

**try** {

String sql = "SELECT \* FROM products WHERE quantityAvailable > 0";

PreparedStatement pstmt = con.prepareStatement(sql);

ResultSet rs = pstmt.executeQuery();

**while** (rs.next()) {

Product product = **new** Product(

rs.getInt("ProductID"),

rs.getString("ProductName"),

rs.getString("Description"),

rs.getDouble("Price")

);

availableProducts.add(product);

}

} **catch** (SQLException e) {

System.***out***.println("Error fetching available products: " + e.getMessage());

}

**return** availableProducts;

}

}

**IProducts service**

package dao;

import entity.Product;

import exception.ProductNotFoundException;

public interface IProductsService {

Product getProductDetails(int productId) throws ProductNotFoundException;

boolean updateProductInfo(int productId, int option, String newValue) throws ProductNotFoundException;

boolean isProductInStock(int productId) throws ProductNotFoundException;

}

**Products service**

package dao;

import entity.Product;

import exception.DatabaseConnectionException;

import exception.ProductNotFoundException;

import util.DBConnUtil;

import java.sql.\*;

public class ProductsService implements IProductsService {

private Connection con;

public ProductsService() throws DatabaseConnectionException {

super();

con = DBConnUtil.getDbConnection();

}

@Override

public Product getProductDetails(int productId) throws ProductNotFoundException {

Product product = null;

try {

PreparedStatement ps = con.prepareStatement("SELECT \* FROM products WHERE ProductID = ?");

ps.setInt(1, productId);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

product = new Product(

rs.getInt("ProductID"),

rs.getString("ProductName"),

rs.getString("Description"),

rs.getDouble("Price")

);

} else {

throw new ProductNotFoundException("Product with ID " + productId + " not found.");

}

} catch (SQLException e) {

e.printStackTrace();

}

return product;

}

@Override

public boolean updateProductInfo(int productId, int option, String newValue) throws ProductNotFoundException {

boolean updated = false;

String sql;

switch (option) {

case 1 -> sql = "UPDATE products SET Price = ? WHERE ProductID = ?";

case 2 -> sql = "UPDATE products SET Description = ? WHERE ProductID = ?";

default -> throw new IllegalArgumentException("Invalid option selected.");

}

try (PreparedStatement ps = con.prepareStatement(sql)) {

if (option == 1) {

ps.setDouble(1, Double.parseDouble(newValue));

} else {

ps.setString(1, newValue);

}

ps.setInt(2, productId);

int rows = ps.executeUpdate();

updated = rows > 0;

if (!updated) {

throw new ProductNotFoundException("Product with ID " + productId + " not found for update.");

}

} catch (SQLException e) {

e.printStackTrace();

}

return updated;

}

@Override

public boolean isProductInStock(int productId) throws ProductNotFoundException {

try {

PreparedStatement ps = con.prepareStatement("SELECT QuantityInStock FROM inventory WHERE ProductID = ?");

ps.setInt(1, productId);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

return rs.getInt("QuantityInStock") > 0;

} else {

throw new ProductNotFoundException("Product with ID " + productId + " not found in inventory.");

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

}

**IOrder service**

**package** dao;

**import** exception.OrderNotFoundException;

**public** **interface** IOrderService {

**double** calculateTotalAmount(**int** orderId) **throws** OrderNotFoundException;

**void** getOrderDetails(**int** orderId) **throws** OrderNotFoundException;

**boolean** updateOrderStatus(**int** orderId, String status) **throws** OrderNotFoundException;

**boolean** cancelOrder(**int** orderId) **throws** OrderNotFoundException;

}

**Order service**

**package** dao;

**import** exception.OrderNotFoundException;

**import** exception.DatabaseConnectionException;

**import** util.DBConnUtil;

**import** java.sql.\*;

**public** **class** OrderService **implements** IOrderService {

**private** Connection con;

**public** OrderService() **throws** DatabaseConnectionException {

con = DBConnUtil.*getDbConnection*();

}

@Override

**public** **double** calculateTotalAmount(**int** orderId) **throws** OrderNotFoundException {

**double** totalAmount = 0;

String query = "SELECT od.Quantity, p.Price " +

"FROM orderdetails od " +

"JOIN products p ON od.ProductID = p.ProductID " +

"WHERE od.OrderID = ?";

**try** (PreparedStatement ps = con.prepareStatement(query)) {

ps.setInt(1, orderId);

ResultSet rs = ps.executeQuery();

**boolean** found = **false**;

**while** (rs.next()) {

found = **true**;

**int** quantity = rs.getInt("Quantity");

**double** price = rs.getDouble("Price");

totalAmount += quantity \* price;

}

**if** (!found) {

**throw** **new** OrderNotFoundException("Order with ID " + orderId + " not found.");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** totalAmount;

}

@Override

**public** **void** getOrderDetails(**int** orderId) **throws** OrderNotFoundException {

String query = "SELECT p.ProductName, od.Quantity " +

"FROM orderdetails od " +

"JOIN products p ON od.ProductID = p.ProductID " +

"WHERE od.OrderID = ?";

**try** (PreparedStatement ps = con.prepareStatement(query)) {

ps.setInt(1, orderId);

ResultSet rs = ps.executeQuery();

**boolean** found = **false**;

System.***out***.println("Order Details for Order ID " + orderId + ":");

**while** (rs.next()) {

found = **true**;

String productName = rs.getString("ProductName");

**int** qty = rs.getInt("Quantity");

System.***out***.println("Product: " + productName + " | Quantity: " + qty);

}

**if** (!found) {

**throw** **new** OrderNotFoundException("Order with ID " + orderId + " not found.");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

@Override

**public** **boolean** updateOrderStatus(**int** orderId, String status) **throws** OrderNotFoundException {

**boolean** updated = **false**;

String query = "UPDATE orders SET Status = ? WHERE OrderId = ?";

**try** (PreparedStatement ps = con.prepareStatement(query)) {

ps.setString(1, status);

ps.setInt(2, orderId);

**int** rows = ps.executeUpdate();

**if** (rows > 0) {

updated = **true**;

} **else** {

**throw** **new** OrderNotFoundException("Order with ID " + orderId + " not found.");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** updated;

}

@Override

**public** **boolean** cancelOrder(**int** orderId) **throws** OrderNotFoundException {

**boolean** cancelled = **false**;

**try** {

// Step 1: Get productId and quantity for the order

String getDetails = "SELECT ProductID, Quantity FROM orderdetails WHERE OrderId = ?";

**try** (PreparedStatement ps = con.prepareStatement(getDetails)) {

ps.setInt(1, orderId);

ResultSet rs = ps.executeQuery();

**boolean** found = **false**;

**while** (rs.next()) {

found = **true**;

**int** productId = rs.getInt("ProductID");

**int** qty = rs.getInt("Quantity");

// Step 2: Update inventory stock

String updateStock = "UPDATE inventory SET QuantityInStock = QuantityInStock + ? WHERE ProductID = ?";

**try** (PreparedStatement updatePs = con.prepareStatement(updateStock)) {

updatePs.setInt(1, qty);

updatePs.setInt(2, productId);

updatePs.executeUpdate();

}

}

**if** (!found) {

**throw** **new** OrderNotFoundException("Order with ID " + orderId + " not found.");

}

}

// Step 3: Delete from orderdetails

String deleteDetails = "DELETE FROM orderdetails WHERE OrderId = ?";

**try** (PreparedStatement ps = con.prepareStatement(deleteDetails)) {

ps.setInt(1, orderId);

ps.executeUpdate();

}

// Step 4: Delete from orders

String deleteOrder = "DELETE FROM orders WHERE OrderId = ?";

**try** (PreparedStatement ps = con.prepareStatement(deleteOrder)) {

ps.setInt(1, orderId);

**int** rows = ps.executeUpdate();

cancelled = rows > 0;

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** cancelled;

}

}

**IOrderDetails service**

**package** dao;

**import** entity.OrderDetails;

**import** exception.OrderDetailNotFoundException;

**public** **interface** IOrderDetails {

**double** calculateSubtotal(OrderDetails orderDetails);

**void** getOrderDetailInfo(OrderDetails orderDetails);

**void** updateQuantity(OrderDetails orderDetails, **int** newQuantity);

**void** addDiscount(OrderDetails orderDetails, **double** discountPercentage);

OrderDetails fetchOrderDetailById(**int** orderDetailId) **throws** OrderDetailNotFoundException;

}

**OrderDetails service**

**package** dao;

**import** entity.Orders;

**import** entity.Product;

**import** entity.OrderDetails;

**import** exception.DatabaseConnectionException;

**import** exception.OrderDetailNotFoundException;

**import** util.DBConnUtil;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **class** OrderDetailsImplementation **implements** IOrderDetails {

**private** Connection con;

**public** OrderDetailsImplementation() **throws** DatabaseConnectionException {

con = DBConnUtil.*getDbConnection*();

}

@Override

**public** **double** calculateSubtotal(OrderDetails orderDetails) {

**return** orderDetails.getQuantity() \* orderDetails.getProduct().getPrice();

}

@Override

**public** **void** getOrderDetailInfo(OrderDetails orderDetails) {

System.***out***.println("Order Detail ID: " + orderDetails.getOrderDetailID());

System.***out***.println("Order ID: " + orderDetails.getOrder().getOrderId());

System.***out***.println("Product: " + orderDetails.getProduct().getProductName());

System.***out***.println("Quantity: " + orderDetails.getQuantity());

System.***out***.println("Subtotal: ₹" + calculateSubtotal(orderDetails));

}

@Override

**public** **void** updateQuantity(OrderDetails orderDetails, **int** newQuantity) {

String updateQuery = "UPDATE orderdetails SET Quantity = ? WHERE OrderDetailID = ?";

**try** (PreparedStatement ps = con.prepareStatement(updateQuery)) {

ps.setInt(1, newQuantity);

ps.setInt(2, orderDetails.getOrderDetailID());

**int** rows = ps.executeUpdate();

**if** (rows > 0) {

orderDetails.setQuantity(newQuantity);

System.***out***.println("Quantity updated successfully.");

} **else** {

System.***out***.println("Order detail not found.");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

@Override

**public** **void** addDiscount(OrderDetails orderDetails, **double** discountPercentage) {

**double** price = orderDetails.getProduct().getPrice();

**double** discountedPrice = price - (price \* discountPercentage / 100);

orderDetails.getProduct().setPrice(discountedPrice);

System.***out***.println("Discount applied: " + discountPercentage + "%");

System.***out***.println("New Price: ₹" + discountedPrice);

}

@Override

**public** OrderDetails fetchOrderDetailById(**int** orderDetailId) **throws** OrderDetailNotFoundException {

String query = "SELECT od.OrderDetailID, od.Quantity, " +

"o.OrderId, o.OrderDate, o.Status, " +

"p.ProductID, p.ProductName, p.Description, p.Price " +

"FROM orderdetails od " +

"JOIN orders o ON od.OrderID = o.OrderId " +

"JOIN products p ON od.ProductID = p.ProductID " +

"WHERE od.OrderDetailID = ?";

**try** (PreparedStatement ps = con.prepareStatement(query)) {

ps.setInt(1, orderDetailId);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

Orders order = **new** Orders();

order.setOrderId(rs.getInt("OrderId"));

order.setOrderDate(rs.getDate("OrderDate").toLocalDate());

order.setStatus(rs.getString("Status"));

Product product = **new** Product();

product.setProductId(rs.getInt("ProductID"));

product.setProductName(rs.getString("ProductName"));

product.setDescription(rs.getString("Description"));

product.setPrice(rs.getDouble("Price"));

OrderDetails details = **new** OrderDetails();

details.setOrderDetailID(rs.getInt("OrderDetailID"));

details.setOrder(order);

details.setProduct(product);

details.setQuantity(rs.getInt("Quantity"));

**return** details;

} **else** {

**throw** **new** OrderDetailNotFoundException("OrderDetail with ID " + orderDetailId + " not found.");

}

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** OrderDetailNotFoundException("Database error occurred while fetching order detail.");

}

}

}

**IInventory service**

**package** dao;

**import** entity.Inventory;

**import** java.util.List;

**public** **interface** IInventoryService {

**void** addToInventory(**int** inventoryID, **int** quantity);

**void** removeFromInventory(**int** inventoryID, **int** quantity);

**void** updateStockQuantity(**int** inventoryID, **int** newQuantity);

**boolean** isProductAvailable(**int** inventoryID, **int** quantityToCheck);

**double** getInventoryValue(**int** inventoryID);

List<Inventory> listLowStockProducts(**int** threshold);

List<Inventory> listOutOfStockProducts();

**void** listAllProducts();

}

**Inventory service**

**package** dao;

**import** entity.Inventory;

**import** entity.Product;

**import** util.DBConnUtil;

**import** exception.DatabaseConnectionException;

**import** java.sql.\*;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** InventoryService **implements** IInventoryService {

**private** Connection con;

**public** InventoryService() **throws** DatabaseConnectionException {

con = DBConnUtil.*getDbConnection*();

}

@Override

**public** **void** addToInventory(**int** inventoryID, **int** quantity) {

**try** {

String updateQuery = "UPDATE inventory SET QuantityInStock = QuantityInStock + ? WHERE InventoryID = ?";

**try** (PreparedStatement stmt = con.prepareStatement(updateQuery)) {

stmt.setInt(1, quantity);

stmt.setInt(2, inventoryID);

**int** rowsUpdated = stmt.executeUpdate();

**if** (rowsUpdated > 0) {

System.***out***.println("Added " + quantity + " to Inventory ID: " + inventoryID);

} **else** {

System.***out***.println("Inventory ID not found!");

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

@Override

**public** **void** removeFromInventory(**int** inventoryID, **int** quantity) {

**try** {

String updateQuery = "UPDATE inventory SET QuantityInStock = QuantityInStock - ? WHERE InventoryID = ?";

**try** (PreparedStatement stmt = con.prepareStatement(updateQuery)) {

stmt.setInt(1, quantity);

stmt.setInt(2, inventoryID);

**int** rowsUpdated = stmt.executeUpdate();

**if** (rowsUpdated > 0) {

System.***out***.println("Removed " + quantity + " from Inventory ID: " + inventoryID);

} **else** {

System.***out***.println("Inventory ID not found!");

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

@Override

**public** **void** updateStockQuantity(**int** inventoryID, **int** newQuantity) {

**try** {

String updateQuery = "UPDATE inventory SET QuantityInStock = ? WHERE InventoryID = ?";

**try** (PreparedStatement stmt = con.prepareStatement(updateQuery)) {

stmt.setInt(1, newQuantity);

stmt.setInt(2, inventoryID);

**int** rowsUpdated = stmt.executeUpdate();

**if** (rowsUpdated > 0) {

System.***out***.println("Updated stock quantity for Inventory ID: " + inventoryID);

} **else** {

System.***out***.println("Inventory ID not found!");

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

@Override

**public** **boolean** isProductAvailable(**int** inventoryID, **int** quantityToCheck) {

**try** {

String query = "SELECT QuantityInStock FROM inventory WHERE InventoryID = ?";

**try** (PreparedStatement stmt = con.prepareStatement(query)) {

stmt.setInt(1, inventoryID);

ResultSet rs = stmt.executeQuery();

**if** (rs.next()) {

**int** currentStock = rs.getInt("QuantityInStock");

**return** currentStock >= quantityToCheck;

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **false**;

}

@Override

**public** **double** getInventoryValue(**int** inventoryID) {

**try** {

String query = "SELECT QuantityInStock, products.Price FROM inventory " +

"JOIN products ON inventory.ProductID = products.ProductID WHERE InventoryID = ?";

**try** (PreparedStatement stmt = con.prepareStatement(query)) {

stmt.setInt(1, inventoryID);

ResultSet rs = stmt.executeQuery();

**if** (rs.next()) {

**int** quantityInStock = rs.getInt("QuantityInStock");

**double** productPrice = rs.getDouble("Price");

**return** quantityInStock \* productPrice;

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** 0;

}

@Override

**public** List<Inventory> listLowStockProducts(**int** threshold) {

List<Inventory> lowStockProducts = **new** ArrayList<>();

**try** {

String query = "SELECT \* FROM inventory WHERE QuantityInStock < ?";

**try** (PreparedStatement stmt = con.prepareStatement(query)) {

stmt.setInt(1, threshold);

ResultSet rs = stmt.executeQuery();

**while** (rs.next()) {

**int** inventoryID = rs.getInt("InventoryID");

**int** productID = rs.getInt("ProductID");

**int** quantityInStock = rs.getInt("QuantityInStock");

Date lastStockUpdate = rs.getDate("LastStockUpdate");

Product product = getProductById(productID);

Inventory inventory = **new** Inventory(inventoryID, product, quantityInStock, lastStockUpdate);

lowStockProducts.add(inventory);

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** lowStockProducts;

}

@Override

**public** List<Inventory> listOutOfStockProducts() {

List<Inventory> outOfStockProducts = **new** ArrayList<>();

**try** {

String query = "SELECT \* FROM inventory WHERE QuantityInStock = 0";

**try** (PreparedStatement stmt = con.prepareStatement(query)) {

ResultSet rs = stmt.executeQuery();

**while** (rs.next()) {

**int** inventoryID = rs.getInt("InventoryID");

**int** productID = rs.getInt("ProductID");

**int** quantityInStock = rs.getInt("QuantityInStock");

Date lastStockUpdate = rs.getDate("LastStockUpdate");

Product product = getProductById(productID);

Inventory inventory = **new** Inventory(inventoryID, product, quantityInStock, lastStockUpdate);

outOfStockProducts.add(inventory);

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** outOfStockProducts;

}

@Override

**public** **void** listAllProducts() {

**try** {

String query = "SELECT \* FROM inventory";

**try** (PreparedStatement stmt = con.prepareStatement(query)) {

ResultSet rs = stmt.executeQuery();

System.***out***.println("All Products in Inventory:");

System.***out***.println("---------------------------------------------------");

**while** (rs.next()) {

**int** inventoryID = rs.getInt("InventoryID");

**int** productID = rs.getInt("ProductID");

**int** quantityInStock = rs.getInt("QuantityInStock");

Date lastStockUpdate = rs.getDate("LastStockUpdate");

Product product = getProductById(productID);

System.***out***.println(String.*format*("InventoryID: %-5d Product: %-20s Quantity: %-5d Last Update: %-15s",

inventoryID, product.getProductName(), quantityInStock, lastStockUpdate));

}

System.***out***.println("---------------------------------------------------");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**private** Product getProductById(**int** productID) {

**try** {

String query = "SELECT \* FROM products WHERE ProductID = ?";

**try** (PreparedStatement stmt = con.prepareStatement(query)) {

stmt.setInt(1, productID);

ResultSet rs = stmt.executeQuery();

**if** (rs.next()) {

String productName = rs.getString("ProductName");

**double** price = rs.getDouble("Price");

**return** **new** Product(productID, productName, price);

}

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **null**;

}

}

**EXCEPTION PACKAGE**

**AuthenticationException.java**

**package** exception;

**public** **class** AuthenticationException **extends** Exception{

**public** AuthenticationException() {

**super**("Authentication failed.");

}

**public** AuthenticationException(String message) {

**super**(message);

}

**public** AuthenticationException(String message, Throwable cause) {

**super**(message, cause);

}

}

**ConcurrencyException.java**

**package** exception;

**public** **class** ConcurrencyException **extends** Exception{

**public** ConcurrencyException() {

**super**("Concurrency conflict detected.");

}

**public** ConcurrencyException(String message) {

**super**(message);

}

**public** ConcurrencyException(String message, Throwable cause) {

**super**(message, cause);

}

}

**CustomerNotFoundException.java**

**package** exception;

**public** **class** CustomerNotFoundException **extends** Exception{

**public** CustomerNotFoundException() {

**super**("Customer not found.");

}

**public** CustomerNotFoundException(String message) {

**super**(message);

}

}

**DatabaseConnectionException.java**

**package** exception;

**public** **class** DatabaseConnectionException **extends** Exception {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** DatabaseConnectionException(String message) {

**super**(message);

}

**public** DatabaseConnectionException(String message, Throwable cause) {

**super**(message, cause);

}

}

**FileWriteException.java**

**package** exception;

**public** **class** FileWriteException **extends** Exception {

**public** FileWriteException() {

**super**("Error writing to file.");

}

**public** FileWriteException(String message) {

**super**(message);

}

**public** FileWriteException(String message, Throwable cause) {

**super**(message, cause);

}

}

**IncompleteOrderException.java**

**package** exception;

**public** **class** IncompleteOrderException **extends** Exception{

**public** IncompleteOrderException() {

**super**("Order is incomplete or inconsistent.");

}

**public** IncompleteOrderException(String message) {

**super**(message);

}

**public** IncompleteOrderException(String message, Throwable cause) {

**super**(message, cause);

}

}

**InsufficientStockException.java**

**package** exception;

**public** **class** InsufficientStockException **extends** Exception{

**public** InsufficientStockException() {

**super**("Not enough stock available.");

}

**public** InsufficientStockException(String message) {

**super**(message);

}

**public** InsufficientStockException(String message, Throwable cause) {

**super**(message, cause);

}

}

**InvalidDataException.java**

**package** exception;

**public** **class** InvalidDataException **extends** Exception {

**public** InvalidDataException() {

**super**("Invalid data provided.");

}

**public** InvalidDataException(String message) {

**super**(message);

}

}

**OrderDetailNotFoundException.java**

**package** exception;

**public** **class** OrderDetailNotFoundException **extends** Exception {

**public** OrderDetailNotFoundException(String message) {

**super**(message);

}

}

**OrderNotFoundException.java**

**package** exception;

**public** **class** OrderNotFoundException **extends** Exception {

**public** OrderNotFoundException(String message) {

**super**(message);

}

}

**PaymentFailedException.java**

**package** exception;

**public** **class** PaymentFailedException **extends** Exception {

**public** PaymentFailedException() {

**super**("Payment failed or was declined.");

}

**public** PaymentFailedException(String message) {

**super**(message);

}

**public** PaymentFailedException(String message, Throwable cause) {

**super**(message, cause);

}

}

**ProductNotFoundException.java**

**package** exception;

**public** **class** ProductNotFoundException **extends** Exception {

**public** ProductNotFoundException(String message) {

**super**(message);

}

}

**Main package**

**package** main;

**import** dao.\*;

**import** entity.\*;

**import** exception.\*;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** MainModule {

**public** **static** **void** main(String[] args) **throws** DatabaseConnectionException, OrderNotFoundException, OrderDetailNotFoundException {

Scanner scan = **new** Scanner(System.***in***);

CustomerService customerService = **new** CustomerService();

IProductsService productService = **new** ProductsService();

OrderService orderService = **new** OrderService();

OrderDetailsImplementation orderDetailsService = **new** OrderDetailsImplementation();

InventoryService inventoryService = **new** InventoryService();

**while** (**true**) {

System.***out***.println("\n--- TECHSHOP MANAGEMENT SYSTEM ---");

System.***out***.println("1. Customer Management");

System.***out***.println("2. Product Management");

System.***out***.println("3. Order Management");

System.***out***.println("4. Order Details Management");

System.***out***.println("5. Inventory Management");

System.***out***.println("6. Exit");

System.***out***.print("Choose a module (1-6): ");

**int** module = scan.nextInt();

scan.nextLine();

**switch** (module) {

**case** 1 -> *customerMenu*(scan, customerService);

**case** 2 -> *productMenu*(scan, productService);

**case** 3 -> *orderMenu*(scan, orderService);

**case** 4 -> *orderDetailsMenu*(scan, orderDetailsService);

**case** 5 -> *inventoryMenu*(scan, inventoryService);

**case** 6 -> {

System.***out***.println("Exiting application. Goodbye!");

scan.close();

**return**;

}

**default** -> System.***out***.println("Invalid option. Please try again.");

}

}

}

// -------------------- Customer Menu --------------------

**private** **static** **void** customerMenu(Scanner scan, CustomerService service) {

System.***out***.println("\n--- Customer Management ---");

System.***out***.println("1. Register Customer");

System.***out***.println("2. Get Customer by ID");

System.***out***.println("3. Update Customer Info");

System.***out***.println("4. Calculate Total Orders");

System.***out***.print("Choose an option (1-4): ");

**int** key = scan.nextInt();

scan.nextLine(); // consume newline

**switch** (key) {

**case** 1 -> *registerCustomer*(scan, service);

**case** 2 -> *getCustomerById*(scan, service);

**case** 3 -> *updateCustomerInfo*(scan, service);

**case** 4 -> *calculateTotalOrders*(scan, service);

**default** -> System.***out***.println("Invalid option. Returning to main menu.");

}

}

**private** **static** **void** registerCustomer(Scanner scan, CustomerService service) {

System.***out***.println("Enter customer details:");

System.***out***.print("First Name: ");

String firstName = scan.nextLine();

System.***out***.print("Last Name: ");

String lastName = scan.nextLine();

System.***out***.print("Email: ");

String email = scan.nextLine();

System.***out***.print("Phone: ");

String phone = scan.nextLine();

System.***out***.print("Address: ");

String address = scan.nextLine();

Customer customer = **new** Customer(0, firstName, lastName, email, phone, address);

**try** {

**boolean** result = service.insertCustomer(customer);

System.***out***.println(result ? "Customer registered successfully." : "Failed to register customer.");

} **catch** (InvalidDataException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

**private** **static** **void** getCustomerById(Scanner scan, CustomerService service) {

System.***out***.print("Enter Customer ID: ");

**int** id = scan.nextInt();

**try** {

Customer customer = service.getCustomerById(id);

System.***out***.println("Customer Details: " + customer);

} **catch** (CustomerNotFoundException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

**private** **static** **void** updateCustomerInfo(Scanner scan, CustomerService service) {

System.***out***.print("Enter Customer ID to update: ");

**int** id = scan.nextInt();

scan.nextLine(); // consume newline

System.***out***.println("Select field to update:");

System.***out***.println("1. Email");

System.***out***.println("2. Phone");

System.***out***.println("3. Address");

System.***out***.print("Option: ");

**int** field = scan.nextInt();

scan.nextLine(); // consume newline

System.***out***.print("Enter new value: ");

String newValue = scan.nextLine();

**try** {

**boolean** result = service.updateCustomerInfo(id, field, newValue);

System.***out***.println(result ? "Customer info updated." : "Update failed.");

} **catch** (CustomerNotFoundException | InvalidDataException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

**private** **static** **void** calculateTotalOrders(Scanner scan, CustomerService service) {

System.***out***.print("Enter Customer ID: ");

**int** id = scan.nextInt();

**try** {

**int** totalOrders = service.calculateTotalOrders(id);

System.***out***.println("Total Orders: " + totalOrders);

} **catch** (CustomerNotFoundException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

// -------------------- Product Menu --------------------

**private** **static** **void** productMenu(Scanner scan, IProductsService service) {

System.***out***.println("\n--- Product Management ---");

System.***out***.println("1. Get Product Details");

System.***out***.println("2. Update Product Info");

System.***out***.println("3. Check Product Stock");

System.***out***.print("Choose an option (1-3): ");

**int** choice = scan.nextInt();

scan.nextLine();

**switch** (choice) {

**case** 1 -> {

System.***out***.print("Enter Product ID: ");

**int** id = scan.nextInt();

**try** {

Product product = service.getProductDetails(id);

System.***out***.println("Product Details: " + product);

} **catch** (ProductNotFoundException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

**case** 2 -> {

System.***out***.print("Enter Product ID to update: ");

**int** id = scan.nextInt();

System.***out***.println("Select field to update:");

System.***out***.println("1. Price");

System.***out***.println("2. Description");

System.***out***.print("Option: ");

**int** field = scan.nextInt();

scan.nextLine();

System.***out***.print("Enter new value: ");

String newValue = scan.nextLine();

**try** {

**boolean** updated = service.updateProductInfo(id, field, newValue);

System.***out***.println(updated ? "Product updated successfully." : "Update failed.");

} **catch** (ProductNotFoundException e) {

System.***out***.println("Error: " + e.getMessage());

} **catch** (NumberFormatException e) {

System.***out***.println("Invalid price format.");

}

}

**case** 3 -> {

System.***out***.print("Enter Product ID: ");

**int** id = scan.nextInt();

**try** {

**boolean** inStock = service.isProductInStock(id);

System.***out***.println(inStock ? "In stock." : "Out of stock.");

} **catch** (ProductNotFoundException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

**default** -> System.***out***.println("Invalid option. Returning to main menu.");

}

}

// -------------------- Order Menu --------------------

**private** **static** **void** orderMenu(Scanner scan, OrderService service) **throws** OrderNotFoundException {

System.***out***.println("\n--- Order Management ---");

System.***out***.println("1. Calculate Total Amount");

System.***out***.println("2. Get Order Details");

System.***out***.println("3. Update Order Status");

System.***out***.println("4. Cancel Order");

System.***out***.print("Choose an option (1-4): ");

**int** choice = scan.nextInt();

scan.nextLine();

**switch** (choice) {

**case** 1 -> {

System.***out***.print("Enter Order ID: ");

**int** id = scan.nextInt();

**double** total = service.calculateTotalAmount(id);

System.***out***.println("Total Amount: ₹" + total);

}

**case** 2 -> {

System.***out***.print("Enter Order ID: ");

**int** id = scan.nextInt();

service.getOrderDetails(id);

}

**case** 3 -> {

System.***out***.print("Enter Order ID: ");

**int** id = scan.nextInt();

scan.nextLine(); // consume newline

System.***out***.print("Enter new status: ");

String status = scan.nextLine();

**boolean** updated = service.updateOrderStatus(id, status);

System.***out***.println(updated ? "Order updated!" : "Update failed.");

}

**case** 4 -> {

System.***out***.print("Enter Order ID to cancel: ");

**int** id = scan.nextInt();

**boolean** cancelled = service.cancelOrder(id);

System.***out***.println(cancelled ? "Order cancelled!" : "Cancellation failed.");

}

**default** -> System.***out***.println("Invalid option. Returning to main menu.");

}

}

// -----------------------------------------------------order details menu--------------------

**private** **static** **void** orderDetailsMenu(Scanner scan, OrderDetailsImplementation service) **throws** OrderDetailNotFoundException {

System.***out***.println("\n--- Order Details Management ---");

System.***out***.println("1. Calculate Subtotal");

System.***out***.println("2. Get Order Detail Info");

System.***out***.println("3. Update Quantity");

System.***out***.println("4. Add Discount");

System.***out***.print("Choose an option (1-4): ");

**int** choice = scan.nextInt();

scan.nextLine(); // consume newline

**switch** (choice) {

**case** 1 -> {

System.***out***.print("Enter Order Detail ID: ");

**int** id = scan.nextInt();

OrderDetails detail = service.fetchOrderDetailById(id);

**double** subtotal = service.calculateSubtotal(detail);

System.***out***.println("Subtotal: ₹" + subtotal);

}

**case** 2 -> {

System.***out***.print("Enter Order Detail ID: ");

**int** id = scan.nextInt();

OrderDetails detail = service.fetchOrderDetailById(id);

service.getOrderDetailInfo(detail);

}

**case** 3 -> {

System.***out***.print("Enter Order Detail ID: ");

**int** id = scan.nextInt();

OrderDetails detail = service.fetchOrderDetailById(id);

System.***out***.print("Enter new quantity: ");

**int** quantity = scan.nextInt();

service.updateQuantity(detail, quantity);

}

**case** 4 -> {

System.***out***.print("Enter Order Detail ID: ");

**int** id = scan.nextInt();

OrderDetails detail = service.fetchOrderDetailById(id);

System.***out***.print("Enter discount percentage: ");

**double** discount = scan.nextDouble();

service.addDiscount(detail, discount);

}

**default** -> System.***out***.println("Invalid option. Returning to main menu.");

}

}

// -------------------- Inventory Menu --------------------

**private** **static** **void** inventoryMenu(Scanner scan, InventoryService inventoryService) {

System.***out***.println("\n--- Inventory Management ---");

System.***out***.println("1. Add to Inventory");

System.***out***.println("2. Remove from Inventory");

System.***out***.println("3. Update Stock Quantity");

System.***out***.println("4. Check Product Availability");

System.***out***.println("5. Get Inventory Value");

System.***out***.println("6. List Low Stock Products");

System.***out***.println("7. List Out of Stock Products");

System.***out***.println("8. List All Products in Inventory");

System.***out***.print("Choose an option (1-8): ");

**int** key = scan.nextInt();

scan.nextLine(); // consume newline

**switch** (key) {

**case** 1 -> {

System.***out***.print("Enter Inventory ID: ");

**int** id = scan.nextInt();

System.***out***.print("Enter quantity to add: ");

**int** qty = scan.nextInt();

inventoryService.addToInventory(id, qty);

}

**case** 2 -> {

System.***out***.print("Enter Inventory ID: ");

**int** id = scan.nextInt();

System.***out***.print("Enter quantity to remove: ");

**int** qty = scan.nextInt();

inventoryService.removeFromInventory(id, qty);

}

**case** 3 -> {

System.***out***.print("Enter Inventory ID: ");

**int** id = scan.nextInt();

System.***out***.print("Enter new stock quantity: ");

**int** qty = scan.nextInt();

inventoryService.updateStockQuantity(id, qty);

}

**case** 4 -> {

System.***out***.print("Enter Inventory ID: ");

**int** id = scan.nextInt();

System.***out***.print("Enter quantity to check: ");

**int** qty = scan.nextInt();

**boolean** available = inventoryService.isProductAvailable(id, qty);

System.***out***.println(available ? "Product is available." : "Product is NOT available.");

}

**case** 5 -> {

System.***out***.print("Enter Inventory ID: ");

**int** id = scan.nextInt();

**double** value = inventoryService.getInventoryValue(id);

System.***out***.println("Total Inventory Value: ₹" + value);

}

**case** 6 -> {

System.***out***.print("Enter threshold quantity: ");

**int** threshold = scan.nextInt();

List<Inventory> lowStock = inventoryService.listLowStockProducts(threshold);

**if** (lowStock.isEmpty()) {

System.***out***.println("No low stock products.");

} **else** {

lowStock.forEach(System.***out***::println);

}

}

**case** 7 -> {

List<Inventory> outOfStock = inventoryService.listOutOfStockProducts();

**if** (outOfStock.isEmpty()) {

System.***out***.println("No out of stock products.");

} **else** {

outOfStock.forEach(System.***out***::println);

}

}

**case** 8 -> inventoryService.listAllProducts();

**default** -> System.***out***.println("Invalid option.");

}

}}