**OPEN ENDED LAB**

**Title: E Commerce Order Management System**

|  |  |
| --- | --- |
| **COURSE:** | DATABASE MANAGEMENT SYSTEM |
| **GROUP MEMBERS:** | **Ali Huzaifa** |
| **SUBMITTED TO:** | ----------------------------- |

# ****Motivation:****

In today’s digital age, e-commerce has transformed the way goods and services are traded. Behind every click-to-order action lies a sophisticated order management system that handles inventory, orders, users, and payments. This lab allows students to apply SQL and database design knowledge to implement such a system in a real-world context.

# ****Problem Statement:****

Design and implement a relational database for an **E-Commerce Order Management System** using SQL. The system must manage customers, products, orders, order details, and payment information. All constraints (primary keys, foreign keys, check, not null, default, and unique constraints) must be applied appropriately.

**Deliverables**

# Background/Theory:

An **Order Management System (OMS)** in e-commerce manages inventory, customer orders, payments, and order processing. Using **relational database design**, we ensure consistency, avoid redundancy through **normalization**, and enforce data rules via **constraints**, **triggers**, and **transactions**.

Key DBMS concepts used:

* **Normalization (up to 3NF)** for minimizing redundancy.
* **Constraints** for integrity (PK, FK, CHECK, UNIQUE, NOT NULL).
* **CRUD operations** for interacting with data.
* **Stored Procedures** for modular logic.
* **Joins** for retrieving relational data.
* **Triggers** for automatic actions.
* **Indexing** for performance.
* **Transactions & Rollbacks** for atomicity and reliability.

# Procedure / Methodology:

 Requirements gathered: customers, products, orders, payments.

 ERD & EERD designed and normalized up to 3NF.

 SQL schema implemented with integrity constraints.

 Triggers and stored procedures created.

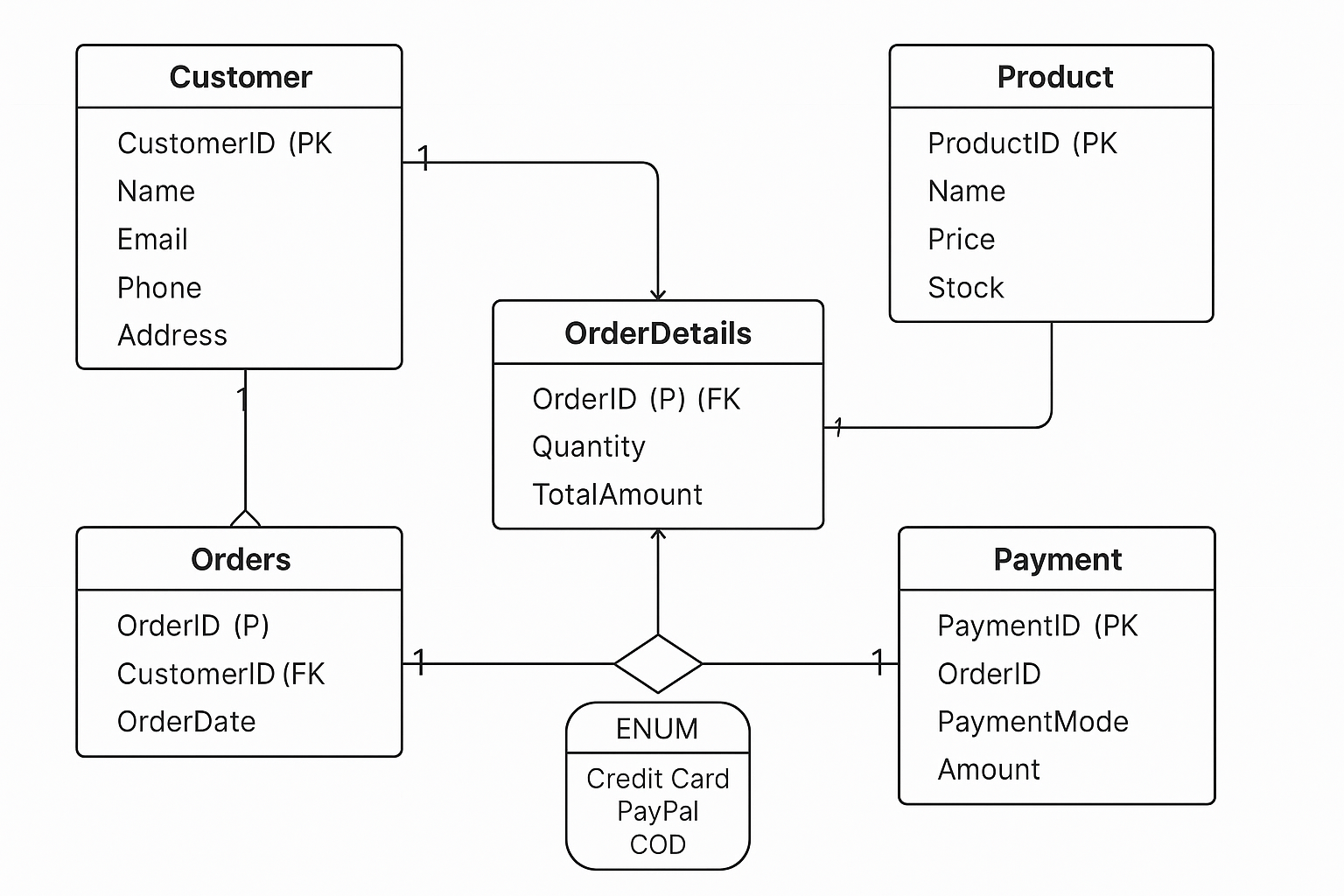
 Sample data inserted for demonstration.

 CRUD queries written.

 Output verified using SELECT JOINs.

 Transactions tested for rollback on failures.

# Flowchart/ERD :



# Code of UI:

import streamlit as st

import pyodbc

import pandas as pd

import os

# Admin credentials

ADMIN\_USERNAME = "admin"

ADMIN\_PASSWORD = "admin123"

# Session state initialization

if "is\_admin" not in st.session\_state:

    st.session\_state.is\_admin = False

if "menu" not in st.session\_state:

    st.session\_state.menu = "🏠 Home"

if "selected\_product\_id" not in st.session\_state:

    st.session\_state.selected\_product\_id = None

# Ensure images folder exists

if not os.path.exists("product\_images"):

    os.makedirs("product\_images")

# DB Connection

def get\_connection():

    return pyodbc.connect(

        r'DRIVER={SQL Server};'

        r'SERVER=DESKTOP-I1VB32G\SQLEXPRESS;'

        r'DATABASE=ecommerce\_db;'

        r'Trusted\_Connection=yes;'

    )

# Fetch all products

def fetch\_all\_products():

    with get\_connection() as conn:

        query = "SELECT ProductID, Name, Price, Stock, ImagePath FROM Product"

        return pd.read\_sql(query, conn)

# Fetch all customers

def fetch\_customers():

    with get\_connection() as conn:

        query = "SELECT \* FROM Customer"

        return pd.read\_sql(query, conn)

# Fetch all orders

def fetch\_orders():

    with get\_connection() as conn:

        query = """

        SELECT o.OrderID, c.Name AS CustomerName, p.Name AS Product,

               od.Quantity, od.TotalAmount

        FROM Orders o

        JOIN Customer c ON o.CustomerID = c.CustomerID

        JOIN OrderDetails od ON o.OrderID = od.OrderID

        JOIN Product p ON od.ProductID = p.ProductID;

        """

        return pd.read\_sql(query, conn)

# Stored procedure call

def get\_customer\_orders(customer\_id):

    with get\_connection() as conn:

        query = f"EXEC GetCustomerOrders @cust\_id = {customer\_id}"

        return pd.read\_sql(query, conn)

# Insert a new order

def insert\_order(customer\_id, product\_id, quantity):

    with get\_connection() as conn:

        cursor = conn.cursor()

        cursor.execute("SELECT Price, Stock FROM Product WHERE ProductID = ?", product\_id)

        row = cursor.fetchone()

        if not row:

            return "❌ Product not found."

        price, stock = row

        if stock < quantity:

            return "❌ Insufficient stock."

        total = price \* quantity

        try:

            cursor.execute("BEGIN TRANSACTION")

            cursor.execute("INSERT INTO Orders (CustomerID) VALUES (?)", customer\_id)

            order\_id = cursor.execute("SELECT @@IDENTITY").fetchval()

            cursor.execute("""

                INSERT INTO OrderDetails (OrderID, ProductID, Quantity, TotalAmount)

                VALUES (?, ?, ?, ?)

            """, order\_id, product\_id, quantity, total)

            cursor.execute("UPDATE Product SET Stock = Stock - ? WHERE ProductID = ?", quantity, product\_id)

            cursor.execute("COMMIT")

            return f"✅ Order {order\_id} placed successfully!"

        except Exception as e:

            cursor.execute("ROLLBACK")

            return f"❌ Transaction failed: {e}"

# Add customer

def add\_customer(name, email, phone, address):

    with get\_connection() as conn:

        cursor = conn.cursor()

        try:

            cursor.execute("""

                INSERT INTO Customer (Name, Email, Phone, Address)

                VALUES (?, ?, ?, ?)

            """, name, email, phone, address)

            conn.commit()

            return "✅ Customer added successfully!"

        except Exception as e:

            return f"❌ Error: {str(e)}"

# Add product with image

def add\_product(name, price, stock, image\_path):

    with get\_connection() as conn:

        cursor = conn.cursor()

        try:

            cursor.execute("""

                INSERT INTO Product (Name, Price, Stock, ImagePath)

                VALUES (?, ?, ?, ?)

            """, name, price, stock, image\_path)

            conn.commit()

            return "✅ Product added successfully!"

        except Exception as e:

            return f"❌ Error: {str(e)}"

# Admin login function

def admin\_login():

    st.subheader("🔐 Admin Login")

    if not st.session\_state.is\_admin:

        username = st.text\_input("Username")

        password = st.text\_input("Password", type="password")

        if st.button("Login"):

            if username == ADMIN\_USERNAME and password == ADMIN\_PASSWORD:

                st.session\_state.is\_admin = True

                st.success("Logged in as admin")

            else:

                st.error("Invalid credentials")

    else:

        st.success("You are logged in as admin.")

        if st.button("Logout"):

            st.session\_state.is\_admin = False

            st.success("Logged out successfully.")

# Main UI

st.title("🛒 E-Commerce Management")

menu\_options = ["🏠 Home", "Add Customer", "View Customers", "View Orders", "Place Order",

                "Get Customer Orders", "Admin Login", "Add Product"]

menu = st.sidebar.selectbox("Menu", menu\_options, index=menu\_options.index(st.session\_state.menu))

# Menu routing

if menu == "🏠 Home":

    st.subheader("📋 Available Products")

    products = fetch\_all\_products()

    cols = st.columns(4)

    for idx, row in products.iterrows():

        with cols[idx % 4]:

            if row['ImagePath'] and os.path.exists(row['ImagePath']):

                st.image(row['ImagePath'], use\_container\_width=True)

            else:

                st.image("https://via.placeholder.com/150", use\_container\_width=True)

            st.markdown(f"\*\*{row['Name']}\*\*")

            st.markdown(f"💰 \*\*Rs {row['Price']:,}\*\*")

            st.markdown(f"📦 Stock: {row['Stock']}")

            if st.button("🛒 Buy Now", key=f"buy\_{row['ProductID']}"):

                st.session\_state.selected\_product\_id = row['ProductID']

                st.session\_state.menu = "Place Order"

                st.rerun()

elif menu == "Add Customer":

    st.subheader("🧍 New Customer Registration")

    name = st.text\_input("Name")

    email = st.text\_input("Email")

    phone = st.text\_input("Phone")

    address = st.text\_area("Address")

    if st.button("Register"):

        msg = add\_customer(name, email, phone, address)

        st.success(msg) if msg.startswith("✅") else st.error(msg)

elif menu == "View Customers":

    st.subheader("👥 All Customers")

    st.dataframe(fetch\_customers())

elif menu == "View Orders":

    st.subheader("📦 All Orders")

    st.dataframe(fetch\_orders())

elif menu == "Place Order":

    st.subheader("🛒 Place New Order")

    if st.session\_state.selected\_product\_id:

        prod\_id = st.number\_input("Product ID", min\_value=1, value=st.session\_state.selected\_product\_id)

    else:

        prod\_id = st.number\_input("Product ID", min\_value=1)

    cust\_id = st.number\_input("Customer ID", min\_value=1)

    qty = st.number\_input("Quantity", min\_value=1)

    if st.button("Place Order"):

        msg = insert\_order(cust\_id, prod\_id, qty)

        if msg.startswith("✅"):

            st.success(msg)

            st.session\_state.selected\_product\_id = None

        else:

            st.error(msg)

elif menu == "Get Customer Orders":

    st.subheader("🔍 Customer Order Lookup")

    cust\_id = st.number\_input("Enter Customer ID", min\_value=1)

    if st.button("Get Orders"):

        st.dataframe(get\_customer\_orders(cust\_id))

elif menu == "Admin Login":

    admin\_login()

elif menu == "Add Product":

    st.subheader("📦 Add Product (Admin Only)")

    if not st.session\_state.is\_admin:

        st.warning("Admin login required to add products.")

    else:

        name = st.text\_input("Product Name")

        price = st.number\_input("Price", min\_value=0.01, format="%.2f")

        stock = st.number\_input("Stock", min\_value=0)

        uploaded\_image = st.file\_uploader("Upload Product Image", type=["png", "jpg", "jpeg"])

        image\_path = None

        if uploaded\_image is not None:

            image\_path = f"product\_images/{uploaded\_image.name}"

            with open(image\_path, "wb") as f:

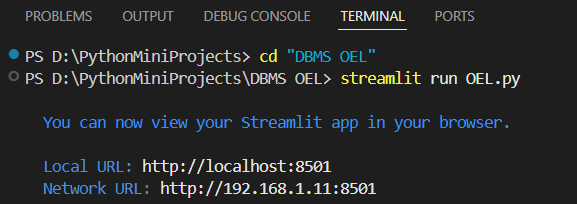
                f.write(uploaded\_image.getbuffer())

        if st.button("Add Product"):

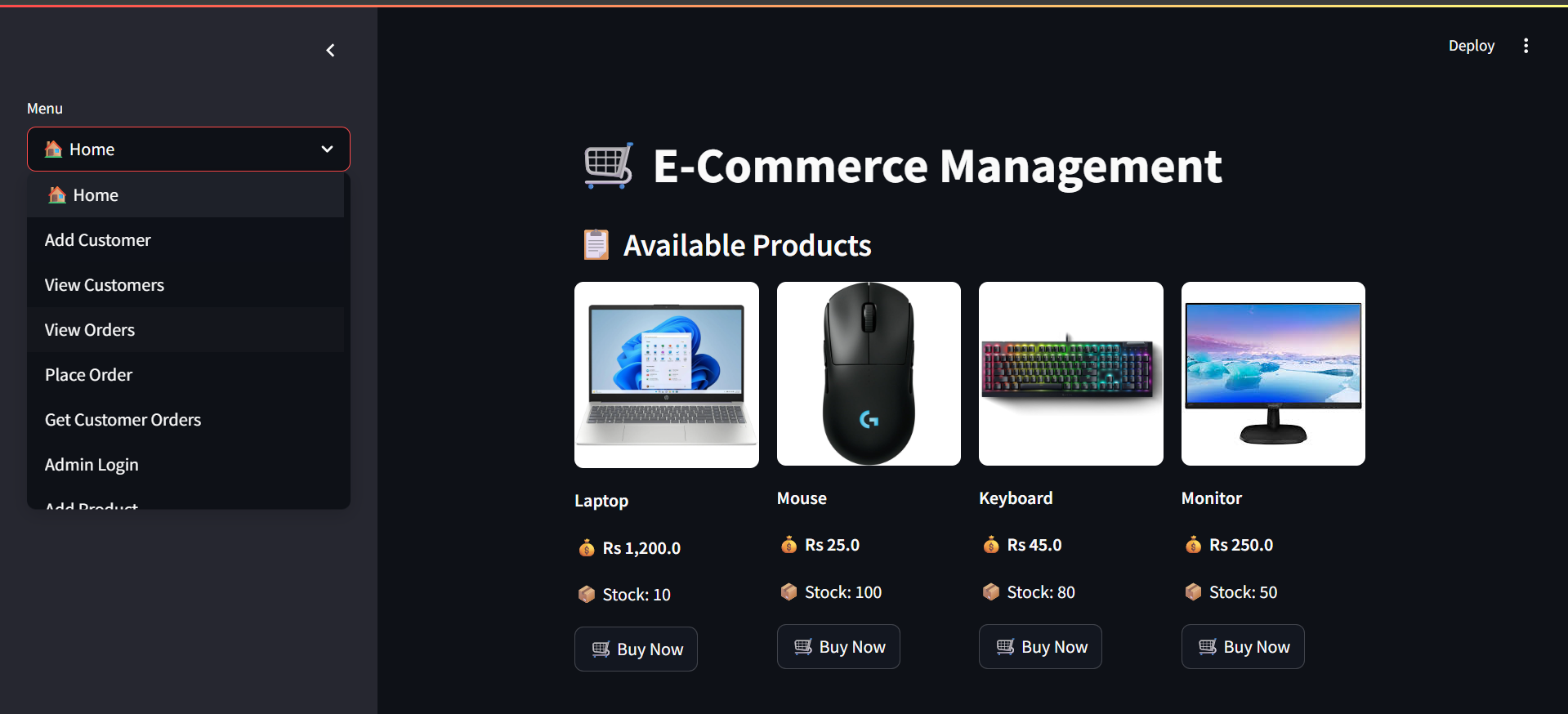
            msg = add\_product(name, price, stock, image\_path)

            st.success(msg) if msg.startswith("✅") else st.error(msg)

**Command to run UI:**



# UI:



# Query:

CREATE DATABASE ecommerce\_db;

USE ecommerce\_db;

GO

-- Customer Table

CREATE TABLE Customer (

CustomerID INT IDENTITY(1,1) PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

Phone VARCHAR(20) UNIQUE NOT NULL,

Address TEXT NOT NULL

);

GO

-- Product Table

CREATE TABLE Product (

ProductID INT IDENTITY(1,1) PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Price DECIMAL(10, 2) NOT NULL CHECK (Price > 0),

Stock INT NOT NULL CHECK (Stock >= 0),

ImagePath VARCHAR(255) -- <-- updated column name

);

GO

-- Orders Table

CREATE TABLE Orders (

OrderID INT IDENTITY(1,1) PRIMARY KEY,

CustomerID INT NOT NULL,

OrderDate DATETIME DEFAULT GETDATE(),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

GO

-- OrderDetails Table

CREATE TABLE OrderDetails (

OrderID INT,

ProductID INT,

Quantity INT NOT NULL CHECK (Quantity > 0),

TotalAmount DECIMAL(10, 2) NOT NULL,

PRIMARY KEY (OrderID, ProductID),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID)

);

GO

-- Indexes for optimization

CREATE INDEX idx\_customer\_email ON Customer(Email);

CREATE INDEX idx\_product\_name ON Product(Name);

GO

-- Insert sample data

INSERT INTO Customer (Name, Email, Phone, Address) VALUES

('Alice', 'alice@example.com', '1234567890', 'Karachi'),

('Bob', 'bob@example.com', '9876543210', 'Lahore'),

('Charlie', 'charlie@example.com', '1122334455', 'Islamabad');

GO

INSERT INTO Product (Name, Price, Stock, ImagePath) VALUES

('Laptop', 1200.00, 10, 'product\_images/laptop.jpg'),

('Mouse', 25.00, 100, 'product\_images/mouse.jpg'),

('Keyboard', 45.00, 80, 'product\_images/keyboard.jpg'),

('Monitor', 250.00, 50, 'product\_images/monitor.jpeg');

GO

INSERT INTO Orders (CustomerID) VALUES (1), (2), (3);

GO

INSERT INTO OrderDetails (OrderID, ProductID, Quantity, TotalAmount) VALUES

(1, 1, 1, 1200.00),

(1, 2, 2, 50.00),

(2, 2, 3, 75.00),

(2, 3, 1, 45.00),

(3, 4, 1, 250.00);

GO

-- Stored Procedure: GetCustomerOrders

IF OBJECT\_ID('GetCustomerOrders', 'P') IS NOT NULL

DROP PROCEDURE GetCustomerOrders;

GO

CREATE PROCEDURE GetCustomerOrders

@cust\_id INT

AS

BEGIN

SELECT o.OrderID, o.OrderDate, p.Name AS Product, od.Quantity, od.TotalAmount

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Product p ON od.ProductID = p.ProductID

WHERE o.CustomerID = @cust\_id

ORDER BY o.OrderDate DESC

END;

GO

-- Trigger to prevent negative stock (keep it for extra safety)

IF OBJECT\_ID('trg\_PreventNegativeStock', 'TR') IS NOT NULL

DROP TRIGGER trg\_PreventNegativeStock;

GO

CREATE TRIGGER trg\_PreventNegativeStock

ON OrderDetails

AFTER INSERT

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted i

JOIN Product p ON i.ProductID = p.ProductID

WHERE i.Quantity > p.Stock

)

BEGIN

RAISERROR('Not enough stock available.', 16, 1);

ROLLBACK TRANSACTION;

END

END;

GO

### Constraints Justified

| **Table** | **Constraint Type** | **Description** |
| --- | --- | --- |
| Customer | UNIQUE(Email) | Prevents duplicate accounts |
| Product | CHECK(Price > 0) | No free products |
| OrderDetails | CHECK(Quantity > 0) | No negative quantity |
| Payment | ENUM | Limits modes to allowed options |
| All tables | NOT NULL | Essential fields must be filled |
| OrderDetails | PK(OrderID, ProductID) | Each product in an order is unique |

# Result: (Sample Queries):

-- SELECT example to view orders

SELECT

o.OrderID,

c.Name AS CustomerName,

p.Name AS Product,

od.Quantity,

od.TotalAmount

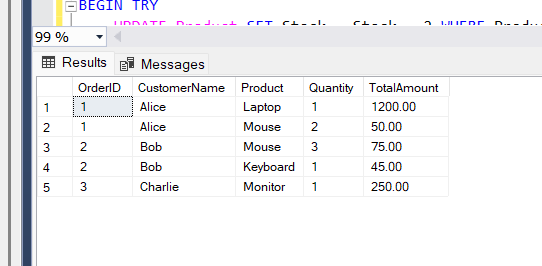
FROM Orders o

JOIN Customer c ON o.CustomerID = c.CustomerID

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Product p ON od.ProductID = p.ProductID;

GO



# Summary:

The developed **E-Commerce Order Management System** demonstrates a real-world DBMS application using:

* Strong schema design
* Normalization
* Procedural programming (stored procs)
* Automation (triggers)
* Optimization (indexing)
* Safety (transactions, rollbacks)

This system handles data reliably and is easily extensible to real-time use cases.

# Reference:

 Oracle SQL Documentation: [https://docs.oracle.com/en/database/oracle/oracle-database/23/cncpt/sql.html#GUID-DA48618A-A6BB-421A-A10A-02859D8ED9AD](https://docs.oracle.com/en/database/oracle/oracle-database/23/cncpt/sql.html%23GUID-DA48618A-A6BB-421A-A10A-02859D8ED9AD)

 Korth, Silberchatz — "Database System Concepts"

 Lab Manual, Iqra University DBMS Course