CREATE TABLE Users (

UserID INT IDENTITY(1,1) PRIMARY KEY,

Username VARCHAR(255) NOT NULL,

Email VARCHAR(255) UNIQUE NOT NULL,

PasswordHash VARCHAR(255) NOT NULL,

UserRole VARCHAR(50) NOT NULL,

AddressID INT,

--FOREIGN KEY (AddressID) REFERENCES UserAddresses(AddressID)

);

CREATE TABLE UserAddresses (

AddressID INT IDENTITY(1,1) PRIMARY KEY,

UserID INT,

Street VARCHAR(255) NOT NULL,

City VARCHAR(100) NOT NULL,

State VARCHAR(100),

ZipCode VARCHAR(20) NOT NULL,

Country VARCHAR(100) NOT NULL,

--FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

CREATE TABLE Products (

ProductID INT IDENTITY(1,1) PRIMARY KEY,

ProductName VARCHAR(255) NOT NULL,

Description TEXT,

Price DECIMAL(10, 2) NOT NULL,

StockQuantity INT NOT NULL,

CategoryID INT,

--FOREIGN KEY (CategoryID) REFERENCES ProductCategories(CategoryID)

);

CREATE TABLE ProductCategories (

CategoryID INT IDENTITY(1,1) PRIMARY KEY,

CategoryName VARCHAR(255) NOT NULL

);

CREATE TABLE Orders (

OrderID INT IDENTITY(1,1) PRIMARY KEY,

UserID INT NOT NULL,

OrderDate DATETIME NOT NULL,

ShippingAddressID INT,

TotalAmount DECIMAL(10, 2) NOT NULL,

Status VARCHAR(50) NOT NULL,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

--FOREIGN KEY (ShippingAddressID) REFERENCES UserAddresses(AddressID)

);

select \* from Orders;

-- Order Details Table

CREATE TABLE OrderDetails (

OrderDetailID INT IDENTITY(1,1) PRIMARY KEY,

OrderID INT NOT NULL,

ProductID INT NOT NULL,

Quantity INT NOT NULL,

PriceAtPurchase DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Payment Information Table (Optional)

CREATE TABLE PaymentInformation (

PaymentID INT identity(1,1) PRIMARY KEY,

OrderID INT NOT NULL,

Amount DECIMAL(10, 2) NOT NULL,

PaymentMethod VARCHAR(100) NOT NULL,

PaymentStatus VARCHAR(50) NOT NULL,

--FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

INSERT INTO Users (Username, Email, PasswordHash, UserRole, AddressID) VALUES

('JohnDoe', 'johndoe@example.com', 'hashedpassword1', 'customer', NULL),

('JaneDoe', 'janedoe@example.com', 'hashedpassword2', 'customer', NULL),

('MikeSmith', 'mikesmith@example.com', 'hashedpassword3', 'customer', NULL),

('SarahBrown', 'sarahbrown@example.com', 'hashedpassword4', 'customer', NULL),

('AlexJohnson', 'alexjohnson@example.com', 'hashedpassword5', 'customer', NULL),

('EmilyDavis', 'emilydavis@example.com', 'hashedpassword6', 'customer', NULL),

('DavidWilson', 'davidwilson@example.com', 'hashedpassword7', 'admin', NULL),

('SophiaMartinez', 'sophiamartinez@example.com', 'hashedpassword8', 'customer', NULL),

('JamesTaylor', 'jamestaylor@example.com', 'hashedpassword9', 'customer', NULL),

('OliviaAnderson', 'oliviaanderson@example.com', 'hashedpassword10', 'customer', NULL);

INSERT INTO UserAddresses (UserID, Street, City, State, ZipCode, Country) VALUES

(1, '123 Main St', 'Anytown', 'CA', '12345', 'USA'),

(2, '456 Maple Ave', 'Springfield', 'IL', '23456', 'USA'),

(3, '789 Pine St', 'Metropolis', 'NY', '34567', 'USA'),

(4, '101 River Rd', 'Gotham', 'NJ', '45678', 'USA'),

(5, '202 Lakeview Dr', 'Star City', 'WA', '56789', 'USA'),

(6, '303 Ocean Ave', 'Central City', 'OH', '67890', 'USA'),

(7, '404 Mountain Pass', 'Smallville', 'KS', '78901', 'USA'),

(8, '505 Desert Blvd', 'Sunnydale', 'NM', '89012', 'USA'),

(9, '606 Forest Ln', 'Hill Valley', 'CA', '90123', 'USA'),

(10, '707 Vine St', 'Twin Peaks', 'WA', '01234', 'USA');

INSERT INTO Products (ProductName, Description, Price, StockQuantity, CategoryID) VALUES

('T-Shirt', 'Cool merch T-Shirt', 19.99, 100, 1),

('Hoodie', 'Warm and cozy hoodie', 39.99, 50, 1),

('Mug', 'Coffee mug with logo', 12.99, 150, 2),

('Sticker Pack', 'Assorted stickers', 4.99, 200, 3),

('Poster', 'High-quality glossy poster', 9.99, 75, 3),

('Hat', 'Baseball cap with logo', 22.99, 100, 1),

('Notebook', 'Lined notebook for your thoughts', 6.99, 120, 4),

('Water Bottle', 'Reusable water bottle', 14.99, 80, 2),

('Keychain', 'Metal keychain with logo', 5.99, 150, 3),

('Tote Bag', 'Eco-friendly shopping bag', 11.99, 100, 4);

INSERT INTO ProductCategories (CategoryName) VALUES

('Apparel'),

('Accessories'),

('Stationery'),

('Drinkware');

INSERT INTO Orders (UserID, OrderDate, ShippingAddressID, TotalAmount, Status) VALUES

(1, '2024-02-01 10:00:00', 1, 59.97, 'Shipped'),

(2, '2024-02-02 11:30:00', 2, 19.99, 'Processing'),

(3, '2024-02-03 15:45:00', 3, 22.99, 'Delivered'),

(4, '2024-02-04 09:20:00', 4, 12.99, 'Shipped'),

(5, '2024-02-05 16:15:00', 5, 39.99, 'Processing'),

(6, '2024-02-06 14:05:00', 6, 9.99, 'Delivered'),

(7, '2024-02-07 18:00:00', 7, 4.99, 'Shipped'),

(8, '2024-02-08 20:00:00', 8, 6.99, 'Processing'),

(9, '2024-02-09 13:00:00', 9, 5.99, 'Delivered'),

(10, '2024-02-10 11:00:00', 10, 11.99, 'Shipped');

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

(1, 1, 3, 19.99),

(2, 2, 1, 39.99),

(3, 3, 2, 12.99),

(4, 4, 1, 4.99),

(5, 5, 1, 9.99),

(6, 6, 1, 22.99),

(7, 7, 3, 6.99),

(8, 8, 2, 14.99),

(9, 9, 5, 5.99),

(10, 10, 1, 11.99);

INSERT INTO PaymentInformation (OrderID, Amount, PaymentMethod, PaymentStatus) VALUES

(1, 59.97, 'Credit Card', 'Completed'),

(2, 19.99, 'PayPal', 'Completed'),

(3, 45.98, 'Credit Card', 'Completed'),

(4, 4.99, 'Debit Card', 'Completed'),

(5, 39.99, 'Credit Card', 'Completed'),

(6, 9.99, 'PayPal', 'Completed'),

(7, 20.97, 'Debit Card', 'Completed'),

(8, 29.98, 'Credit Card', 'Completed'),

(9, 29.95, 'PayPal', 'Completed'),

(10, 11.99, 'Debit Card', 'Completed');

--### Query 1: List of Users with Total Spending. Retrieve users and their total spending on orders.

SELECT

u.Username,

SUM(o.TotalAmount) AS TotalSpending

FROM

Users u

JOIN

Orders o ON u.UserID = o.UserID

GROUP BY

u.Username;

--### Query 2: Products Never Ordered. Find products that have never been ordered.

SELECT

np.ProductID,

np.ProductName

FROM

Products np

LEFT JOIN

OrderDetails od ON np.ProductID = od.ProductID

WHERE

od.ProductID IS NULL;

--### Query 3: Top Selling Products. List the top 5 selling products by quantity.

SELECT TOP 5

np.ProductID,

np.ProductName,

SUM(od.Quantity) AS TotalQuantitySold

FROM

Products np

JOIN

OrderDetails od ON np.ProductID = od.ProductID

GROUP BY

np.ProductID,

np.ProductName

ORDER BY

TotalQuantitySold DESC;

--### Query 4: Users with No Orders. Retrieve a list of users who have never placed an order.

SELECT

pb.UserID,

pb.Username

FROM

Users pb

LEFT JOIN

Orders o ON pb.UserID = o.UserID

WHERE

o.UserID IS NULL;

--### Query 5: Month-over-Month Growth in Sales. Calculate the month-over-month percentage growth in sales.

WITH sales\_per\_month AS (

SELECT

YEAR(OrderDate) AS OrderYear,

MONTH(OrderDate) AS OrderMonth,

SUM(TotalAmount) AS TotalSales

FROM

Orders

GROUP BY

YEAR(OrderDate),

MONTH(OrderDate)

)

SELECT

t1.OrderYear,

t1.OrderMonth,

t1.TotalSales,

t2.TotalSales AS PreviousMonthSales,

(t1.TotalSales - t2.TotalSales) \* 100.0 / t2.TotalSales AS MoM\_Growth

FROM

sales\_per\_month t1

LEFT JOIN

sales\_per\_month t2 ON t1.OrderYear = t2.OrderYear AND t1.OrderMonth = t2.OrderMonth + 1

ORDER BY

t1.OrderYear, t1.OrderMonth;

--### Query 6: Most Popular Product Category. Identify the most popular product category based on quantity sold.

SELECT TOP 1

pc.CategoryName,

SUM(od.Quantity) AS TotalQuantitySold

FROM

Products pb

JOIN

ProductCategories pc ON pb.CategoryID = pc.CategoryID

JOIN

OrderDetails od ON pb.ProductID = od.ProductID

GROUP BY

pc.CategoryName

ORDER BY

TotalQuantitySold DESC;

--### Query 7: Users with Orders Above Average Value. Find users who have placed orders above the average order value.

WITH AverageOrderValue AS (

SELECT AVG(TotalAmount) AS AvgOrderValue

FROM Orders

),

UsersAboveAverage AS (

SELECT

o.UserID,

u.Username,

u.Email,

o.TotalAmount

FROM

Orders o

JOIN

Users u ON o.UserID = u.UserID

CROSS JOIN

AverageOrderValue avg

WHERE

o.TotalAmount > avg.AvgOrderValue

)

SELECT

UserID,

Username,

Email,

TotalAmount

FROM

UsersAboveAverage;

--### Query 8: Product Stock Status. List products along with a status indicating if their stock is Low, Medium, or High based on quantity.

SELECT

ProductID,

ProductName,

Description,

Price,

StockQuantity,

CASE

WHEN StockQuantity <= 50 THEN 'Low'

WHEN StockQuantity <= 100 THEN 'Medium'

ELSE 'High'

END AS StockStatus

FROM

Products;

--### Query 9: Orders with Multiple Products. Retrieve orders that contain more than one type of product.

SELECT o.OrderID, o.UserID, o.OrderDate, o.TotalAmount

FROM Orders o

JOIN (

SELECT OrderID

FROM OrderDetails

GROUP BY OrderID

HAVING COUNT(DISTINCT ProductID) > 1

) od ON o.OrderID = od.OrderID