# Assignment

Task 1

You are tasked with designing a database for a basic online e-commerce platform.

The system should manage products, customers, orders, and shipments. Design the

necessary tables with appropriate relationships and attributes. Provide a script to

create these tables along with any necessary constraints.

Script:

-- Create Customer Table

CREATE TABLE Customers(

CustomerID INT IDENTITY(1,1) PRIMARY KEY,

FirstName NVARCHAR(100) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

Email NVARCHAR(255) NOT NULL UNIQUE,

PhoneNumber NVARCHAR(15),

Address NVARCHAR(255),

City NVARCHAR(100),

State NVARCHAR(100),

ZipCode NVARCHAR(10),

Country NVARCHAR(100),

CreatedAt DATETIME DEFAULT GETDATE()

);

-- Create Product Table

CREATE TABLE Products (

ProductID INT IDENTITY(1,1) PRIMARY KEY,

ProductName NVARCHAR(255) NOT NULL,

Description NVARCHAR(MAX),

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

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

CreatedAt DATETIME DEFAULT GETDATE()

);

-- Create Orders Table

CREATE TABLE Orders (

OrderID INT IDENTITY(1,1) PRIMARY KEY,

CustomerID INT NOT NULL,

OrderDate DATETIME DEFAULT GETDATE(),

Status NVARCHAR(50) DEFAULT 'Pending',

TotalAmount DECIMAL(10, 2) CHECK (TotalAmount >= 0),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE

);

-- Create OrderDetails Table

CREATE TABLE OrderDetails (

OrderDetailID INT IDENTITY(1,1) PRIMARY KEY,

OrderID INT NOT NULL,

ProductID INT NOT NULL,

Quantity INT NOT NULL CHECK (Quantity > 0),

UnitPrice DECIMAL(10, 2) NOT NULL CHECK (UnitPrice >= 0),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE,

FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE CASCADE

);

-- Create Shipments Table

CREATE TABLE Shipments (

ShipmentID INT IDENTITY(1,1) PRIMARY KEY,

OrderID INT NOT NULL,

ShipmentDate DATETIME DEFAULT GETDATE(),

Carrier NVARCHAR(100),

TrackingNumber NVARCHAR(100),

Status NVARCHAR(50) DEFAULT 'Shipped',

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE

);

Task 2: Data Manipulation

1. Populate the tables created in Task 1 with some sample data.

-- Insert Random Data into Customers

INSERT INTO Customers (FirstName, LastName, Email, PhoneNumber, Address, City, State, ZipCode, Country)

VALUES ('Sam', 'Jain', 'Sam.Jain@example.com', '123-456-7890', 'Indore', 'Indore', 'MP', '451551', 'India'),

('Rohit', 'Jain', 'Rohit.Jain@example.com', '987-654-3210', 'HSR layout', 'Bangalore', 'karnataka', '560102', 'India'),

('Astha', 'Jain', 'Astha.Jain@example.com', '555-123-4567', 'BTM Layout', 'Bangalore', 'karnataka', '560102', 'India');

-- Insert Random Data into Products

INSERT INTO Products (ProductName, Description, Price, Stock)

VALUES ('Laptop', 'A high-end gaming laptop', 1200.00, 10),

('Smartphone', 'Latest model smartphone with great features', 800.00, 25),

('Headphones', 'Noise-cancelling wireless headphones', 150.00, 50);

-- Insert Random Data into Orders

INSERT INTO Orders (CustomerID, Status, TotalAmount)

VALUES (1, 'Completed', 1950.00),

(2, 'Pending', 800.00),

(3, 'Shipped', 1200.00);

-- Insert Random Data into OrderDetails

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

VALUES (1, 1, 1, 1200.00), -- 1 Laptop

(1, 3, 5, 150.00), -- 5 Headphones

(2, 2, 1, 800.00), -- 1 Smartphone

(3, 1, 1, 1200.00); -- 1 Laptop

-- Insert Random Data into Shipments

INSERT INTO Shipments (OrderID, Carrier, TrackingNumber, Status)

VALUES (1, 'FedEx', '1234567890', 'Delivered'),

(3, 'UPS', '0987654321', 'In Transit');

1. Write SQL queries to perform the following tasks:

Retrieve the top 5 customers who have made the most orders.

SELECT TOP 5

c.CustomerID,

c.FirstName,

c.LastName,

COUNT(o.OrderID) AS TotalOrders

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.CustomerID, c.FirstName, c.LastName

ORDER BY

TotalOrders DESC;

Display the total revenue generated by each product category.

SELECT

p.CategoryID,

cat.CategoryName,

SUM(od.Quantity \* od.UnitPrice) AS TotalRevenue

FROM

Products p

JOIN

OrderDetails od ON p.ProductID = od.ProductID

JOIN

Orders o ON od.OrderID = o.OrderID

JOIN

Categories cat ON p.CategoryID = cat.CategoryID

GROUP BY

p.CategoryID, cat.CategoryName

ORDER BY

TotalRevenue DESC;

Find the number of orders placed in the last month, broken down by day.

SELECT

CAST(o.OrderDate AS DATE) AS OrderDay,

COUNT(o.OrderID) AS TotalOrders

FROM

Orders o

WHERE

o.OrderDate >= DATEADD(MONTH, -1, GETDATE()) -- Orders from the last month

GROUP BY

CAST(o.OrderDate AS DATE)

ORDER BY

OrderDay ASC;

Calculate the average order value for each customer.

SELECT

OrderValues.CustomerID,

AVG(OrderTotal) AS AvgOrderValue

FROM (

SELECT

o.OrderID,

o.CustomerID,

SUM(od.Quantity \* od.UnitPrice) AS OrderTotal

FROM

Orders o

JOIN

OrderDetails od ON o.OrderID = od.OrderID

GROUP BY

o.OrderID, o.CustomerID

) AS OrderValues

GROUP BY

OrderValues.CustomerID

ORDER BY

AvgOrderValue DESC;

Task 3: Database Administration

You have a SQL Server instance running with multiple databases. Perform the

following tasks:

a) Write SQL commands to make a FULL backup of the "ShopDB" database and store

a compressed backup file in a designated location on disk.

Script:

BACKUP DATABASE ShopDB

TO DISK = 'C:\Backups\ShopDB\_FullBackup.bak'

WITH COMPRESSION, STATS = 10;

b) Give example of what could be used to find the most long-running queries.

SELECT

TOP 5

SUBSTRING(qt.text, (qs.statement\_start\_offset / 2) + 1,

((CASE qs.statement\_end\_offset

WHEN -1 THEN DATALENGTH(qt.text)

ELSE qs.statement\_end\_offset

END - qs.statement\_start\_offset) / 2) + 1) AS QueryText,

qs.execution\_count AS ExecutionCount,

qs.total\_elapsed\_time / qs.execution\_count AS AvgElapsedTime\_ms,

qs.total\_elapsed\_time AS TotalElapsedTime\_ms,

qs.total\_logical\_reads AS TotalLogicalReads,

qs.total\_logical\_writes AS TotalLogicalWrites,

qs.creation\_time AS CreationTime,

qs.last\_execution\_time AS LastExecutionTime,

qs.total\_worker\_time / qs.execution\_count AS AvgCPUTime\_ms

FROM

sys.dm\_exec\_query\_stats AS qs

CROSS APPLY

sys.dm\_exec\_sql\_text(qs.sql\_handle) AS qt

ORDER BY

AvgElapsedTime\_ms DESC;

c) Discuss some common performance tuning techniques in SQL Server. How would

you identify and address performance bottlenecks?

1. **Indexing**
2. **Rewrite Queries**
3. **Avoid Cursors**
4. **Use Query Hints**
5. **Normalization**
6. **Partitioning**

**Identifying Performance Bottlenecks**

1. **Monitor Performance Metrics**
2. **Check Execution Plans**
3. **Analyze Wait Statistics**
4. **Assess Index Usage**

Task 4: Performance Tuning

Identify what could be done to optimize the following stored procedure:

CREATE PROCEDURE GetOrderDetails

@OrderID INT

AS

BEGIN

DECLARE @TotalAmount MONEY;

SELECT

OD.ProductID,

P.ProductName,

OD.UnitPrice,

OD.Quantity,

OD.UnitPrice \* OD.Quantity AS LineTotal

INTO #TempOrderDetails

FROM

OrderDetails OD

JOIN

Products P ON OD.ProductID = P.ProductID

WHERE

OD.OrderID = @OrderID;

SELECT @TotalAmount = SUM(LineTotal) FROM #TempOrderDetails;

SELECT

ProductID,

ProductName,

UnitPrice,

Quantity,

LineTotal,

@TotalAmount AS TotalAmount

FROM

#TempOrderDetails;

DROP TABLE #TempOrderDetails;

END

SCRIPT:

CREATE PROCEDURE GetOrderDetails

@OrderID INT

AS

BEGIN

-- Declare a variable to store the total amount

DECLARE @TotalAmount MONEY;

-- Calculate the total amount and select the order details

WITH OrderDetailsCTE AS (

SELECT

OD.ProductID,

P.ProductName,

OD.UnitPrice,

OD.Quantity,

OD.UnitPrice \* OD.Quantity AS LineTotal

FROM

OrderDetails OD

JOIN

Products P ON OD.ProductID = P.ProductID

WHERE

OD.OrderID = @OrderID

)

SELECT

@TotalAmount = SUM(LineTotal)

FROM

OrderDetailsCTE;

-- Return the order details along with the total amount

SELECT

ProductID,

ProductName,

UnitPrice,

Quantity,

LineTotal,

@TotalAmount AS TotalAmount

FROM

OrderDetailsCTE;

END;