WEEK TWO :6363695

Advanced SQL:Index:

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Region VARCHAR(50)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Sample Data

INSERT INTO Customers (CustomerID, Name, Region) VALUES

(1, 'Alice', 'North'),

(2, 'Bob', 'South'),

(3, 'Charlie', 'East'),

(4, 'David', 'West');

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

(1, 'Laptop', 'Electronics', 1200.00),

(2, 'Smartphone', 'Electronics', 800.00),

(3, 'Tablet', 'Electronics', 600.00),

(4, 'Headphones', 'Accessories', 150.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(1, 1, '2023-01-15'),

(2, 2, '2023-02-20'),

(3, 3, '2023-03-25'),

(4, 4, '2023-04-30');

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

(1, 1, 1, 1),

(2, 2, 2, 2),

(3, 3, 3, 1),

(4, 4, 4, 3);

-- Exercise 1: Creating a Non-Clustered Index

-- Goal: Create a non-clustered index on the ProductName column in the Products table and compare query execution time before and after index creation.

-- Step 1: Query to fetch product details before index creation

SELECT \* FROM Products WHERE ProductName = 'Laptop';

-- Step 2: Create a non-clustered index on ProductName

-- Step 3: Query to fetch product details after index creation

SELECT \* FROM Products WHERE ProductName = 'Laptop';

-- Exercise 2: Creating a Clustered Index

-- Goal: Create a clustered index on the OrderDate column in the Orders table and compare query execution time before and after index creation.

-- Step 1: Query to fetch orders before index creation

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

-- Step 2: Create a clustered index on OrderDate

-- Step 3: Query to fetch orders after index creation

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

-- Exercise 3: Creating a Composite Index

-- Goal: Create a composite index on the CustomerID and OrderDate columns in the Orders table and compare query execution time before and after index creation.

-- Step 1: Query to fetch orders before index creation

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

-- Step 2: Create a composite index on CustomerID and OrderDate

-- Step 3: Query to fetch orders after index creation

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

Exercise 1: Ranking and Window Functions:

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

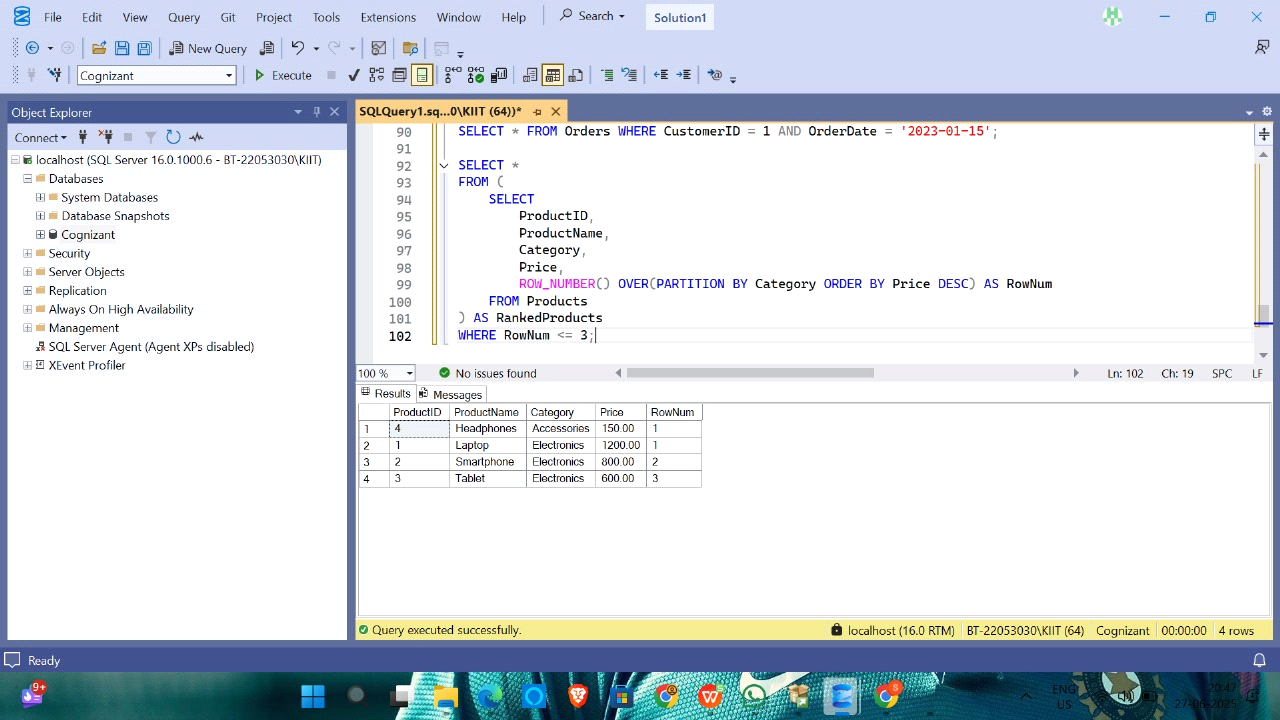
ROW\_NUMBER() OVER(PARTITION BY Category ORDER BY Price DESC) AS RowNum

FROM Products

) AS RankedProducts

WHERE RowNum <= 3;

OUTPUT:



Exercise 1: Create a Stored Procedure:

CREATE TABLE Departments ( DepartmentID INT PRIMARY KEY, DepartmentName VARCHAR(100) );

//Employees Table

CREATE TABLE Employees ( EmployeeID INT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),Salary DECIMAL(10,2), JoinDate DATE );

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

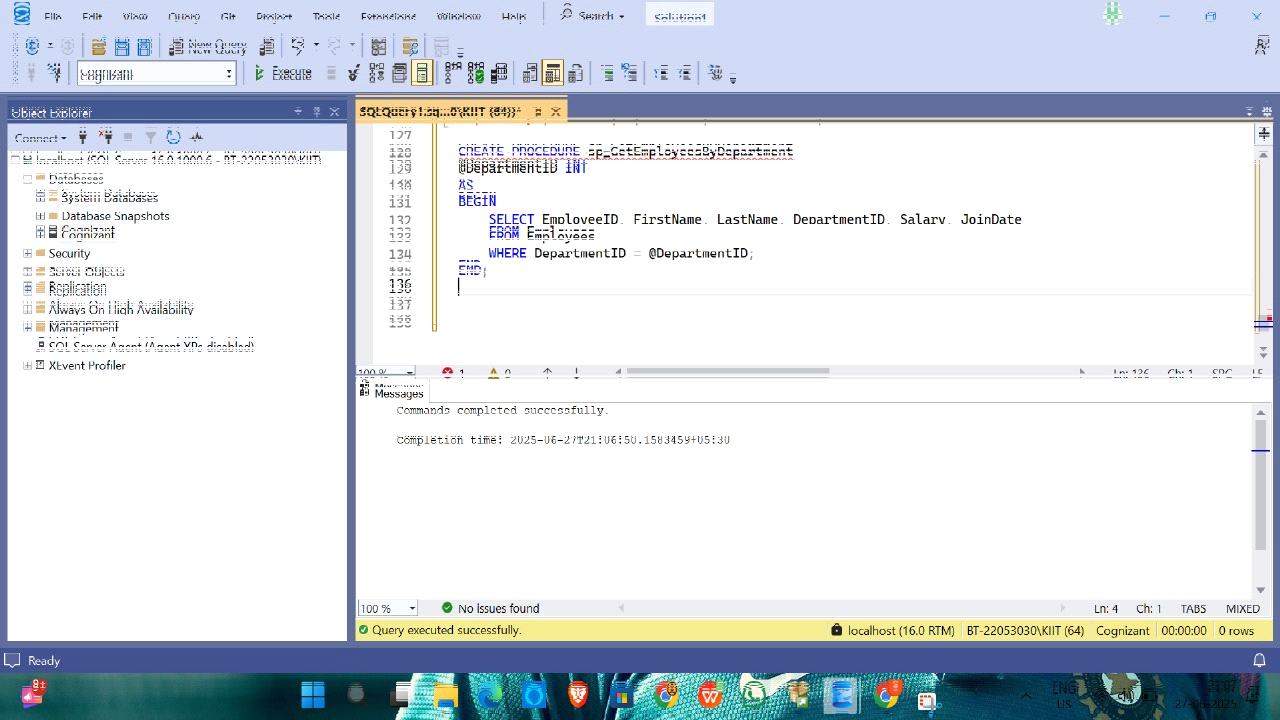
INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES

(1, 'John', 'Doe', 1, 5000.00, '2020-01-15'),

(2, 'Jane', 'Smith', 2, 6000.00, '2019-03-22'),

(3, 'Michael', 'Johnson', 3, 7000.00, '2018-07-30'),

(4, 'Emily', 'Davis', 4, 5500.00, '2021-11-05');



//Exercise 1: Stored Procedure to retrieve employees by DepartmentID

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

//Exercise 7: Stored Procedure to update employee salary

CREATE PROCEDURE sp\_UpdateEmployeeSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

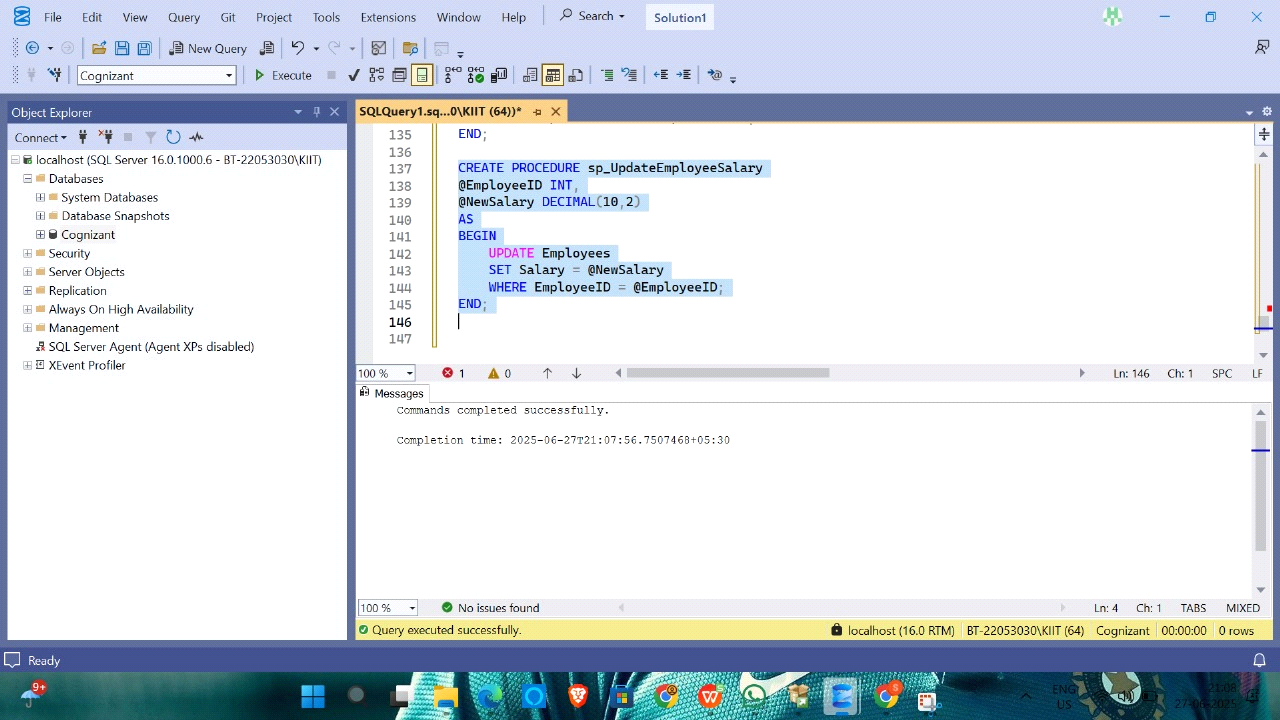
UPDATE Employees

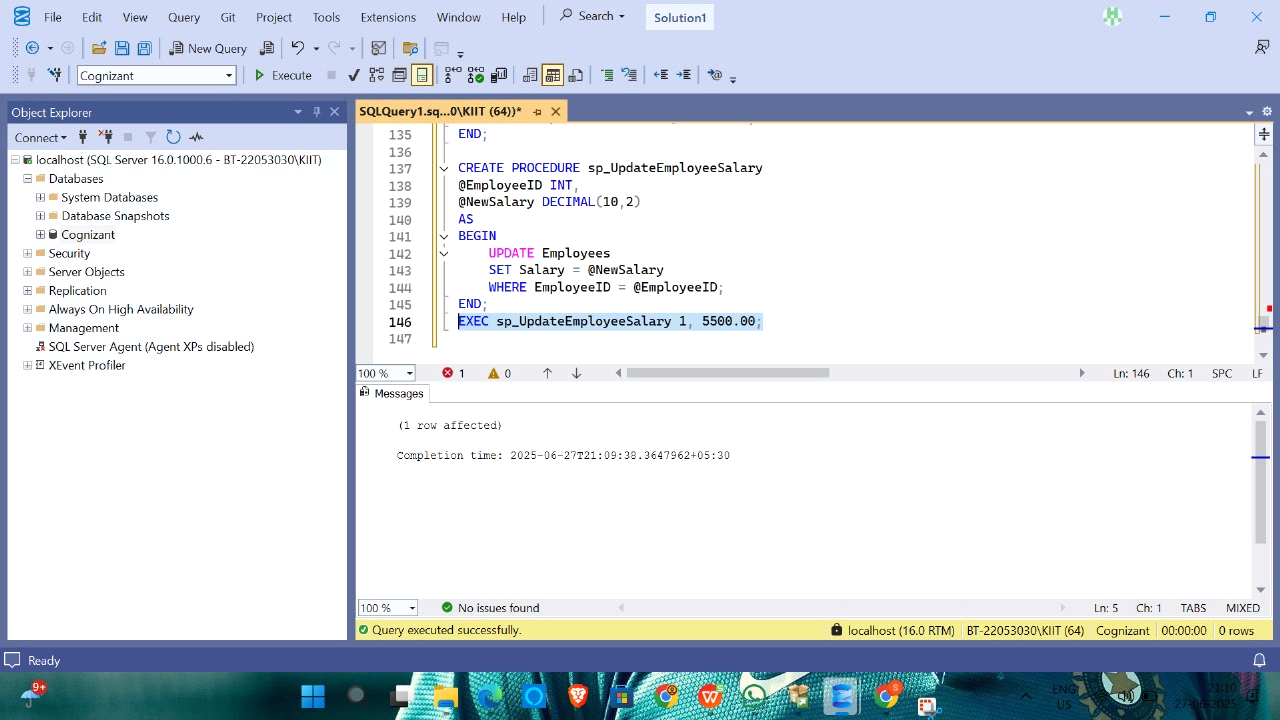
SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END;

EXEC sp\_UpdateEmployeeSalary 1, 5500.00;





//Exercise 5: Stored Procedure to count employees in a department

CREATE PROCEDURE sp\_CountEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

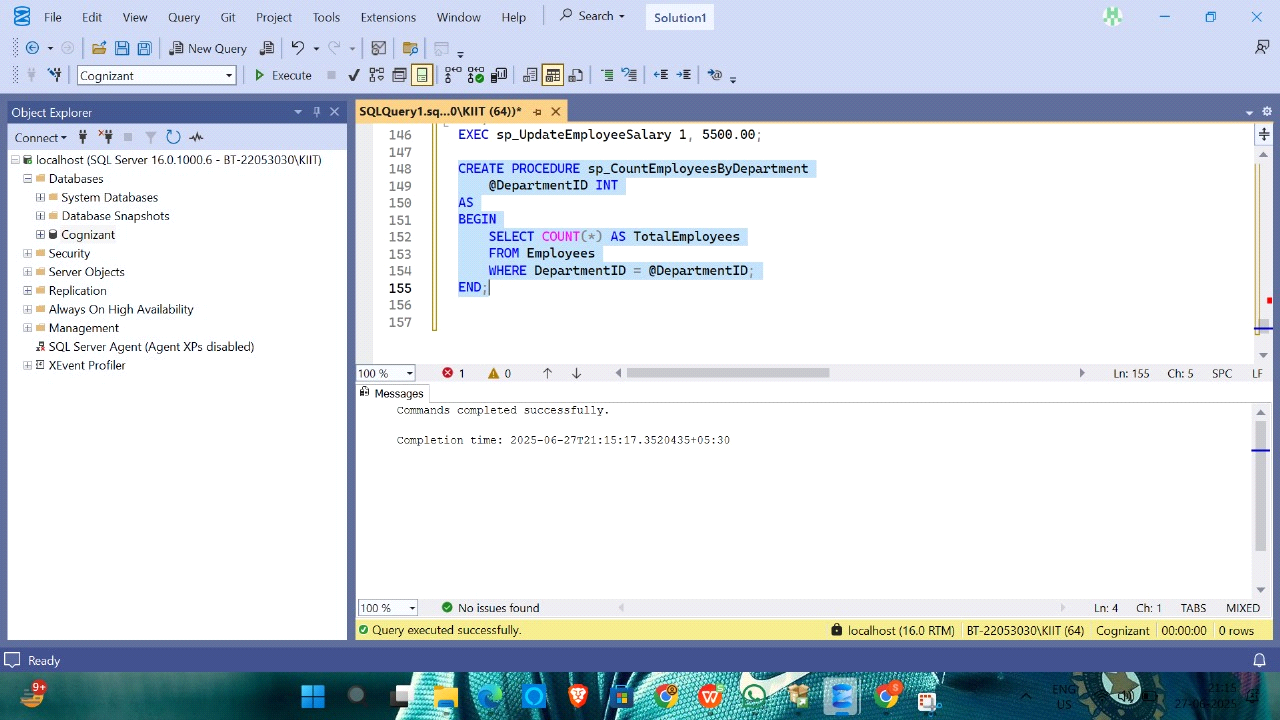
END;

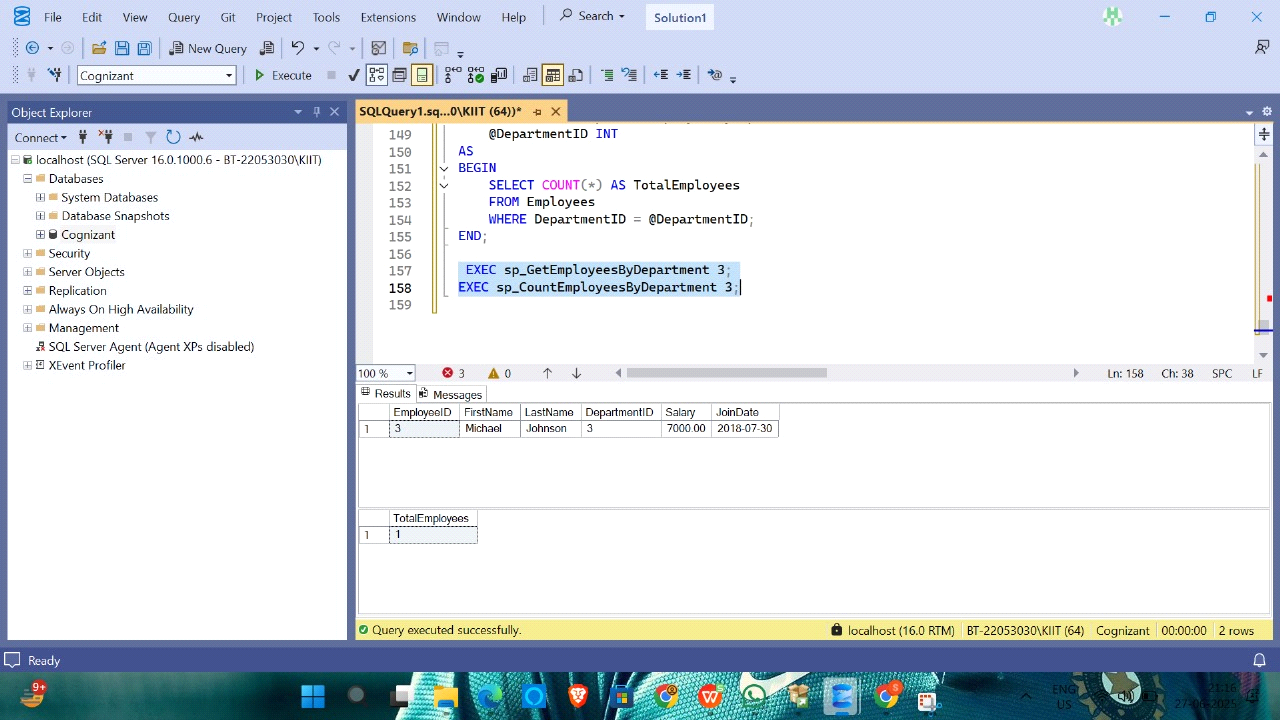
GO

// Exercise 4: Execute stored procedure to get employee details

EXEC sp\_GetEmployeesByDepartment 3;

EXEC sp\_CountEmployeesByDepartment 3;





MOQ:1. Write Testable Code with Moq:

using System;

using Moq;

using NUnit.Framework;

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

return \_mailSender.SendMail("cust123@abc.com", "Some Message");

}

}

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender = null!;

private CustomerComm \_customerComm = null!;

[OneTimeSetUp]

public void Init()

{

\_mockMailSender = new Mock<IMailSender>();

\_mockMailSender

.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

\_customerComm = new CustomerComm(\_mockMailSender.Object);

}

[Test]

public void SendMailToCustomer\_ShouldReturnTrue()

{

var result = \_customerComm.SendMailToCustomer();

Assert.That(result, Is.True);

}

}

