ASSIGNMENT @DN 4.0 - DotNet FSE

WEEK -> 2

SUPERSET ID: 6363921

This contains the solution of the exercise of week 2 assignment.

Contents:

|  |  |
| --- | --- |
| 1. SQL Exercise - Advanced concepts | Exercise 1: Ranking and Window Functions |
| 2. SQL Exercise - Index | Hands-on in this document |
| 4. SQL Exercise - Stored procedure | Exercise 1: Create a Stored Procedure |
| 4. SQL Exercise - Stored procedure | Exercise 5: Return Data from a Stored Procedure |
| 4. SQL Exercise - Stored procedure | Exercise 4: Execute a Stored Procedure |
| 5. SQL Exercise - Functions | Exercise 7: Return Data from a Scalar Function |

Solution @

|  |  |
| --- | --- |
| 1. SQL Exercise - Advanced concepts | Exercise 1: Ranking and Window Functions |

-- Database Schema

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: Created 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

CREATE NONCLUSTERED INDEX IX\_Products\_ProductName ON Products(ProductName);

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

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

-- Exercise 2: Created 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

-- Note: The Orders table has a PRIMARY KEY on OrderID, which typically creates a clustered index.

-- To create a clustered index on OrderDate, we need to drop the foreign key constraint referencing Orders,

-- drop the existing primary key, create the clustered index, and recreate constraints.

ALTER TABLE OrderDetails DROP CONSTRAINT FK\_\_OrderDeta\_\_Order\_\_4AB81AF0;

ALTER TABLE Orders DROP CONSTRAINT PK\_\_Orders\_\_C3905BAFE2D4859E;

ALTER TABLE Orders ADD CONSTRAINT PK\_Orders PRIMARY KEY NONCLUSTERED (OrderID);

CREATE CLUSTERED INDEX IX\_Orders\_OrderDate ON Orders(OrderDate);

ALTER TABLE OrderDetails ADD CONSTRAINT FK\_OrderDetails\_OrderID FOREIGN KEY (OrderID) REFERENCES Orders(OrderID);

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

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

-- Exercise 3: Created 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

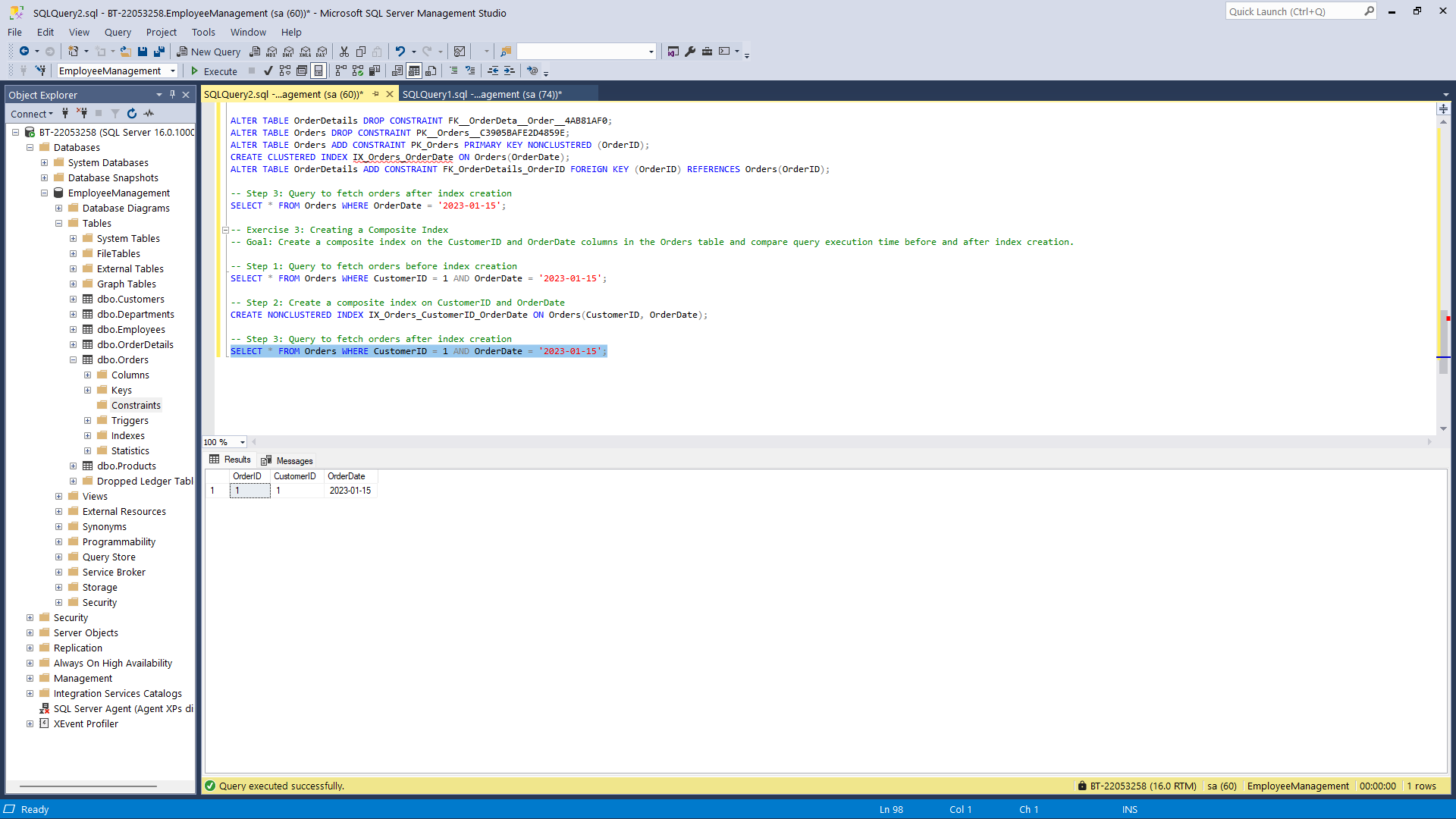
SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

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

CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID\_OrderDate ON Orders(CustomerID, OrderDate);

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

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



Solution @

|  |  |
| --- | --- |
| 4. SQL Exercise - Stored procedure | Exercise 1: Create a Stored Procedure |
| 4. SQL Exercise - Stored procedure | Exercise 5: Return Data from a Stored Procedure |
| 4. SQL Exercise - Stored procedure | Exercise 4: Execute a Stored Procedure |

USE EmployeeManagement;

-- Departments Table:

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

-- Employees Table:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY IDENTITY(1,1),

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 Sample Data:

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

VALUES

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

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

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

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

--Solution @ 1. SQL Exercise - Advanced concepts

--Exercise 1: Ranking and Window Functions

SELECT \* FROM (

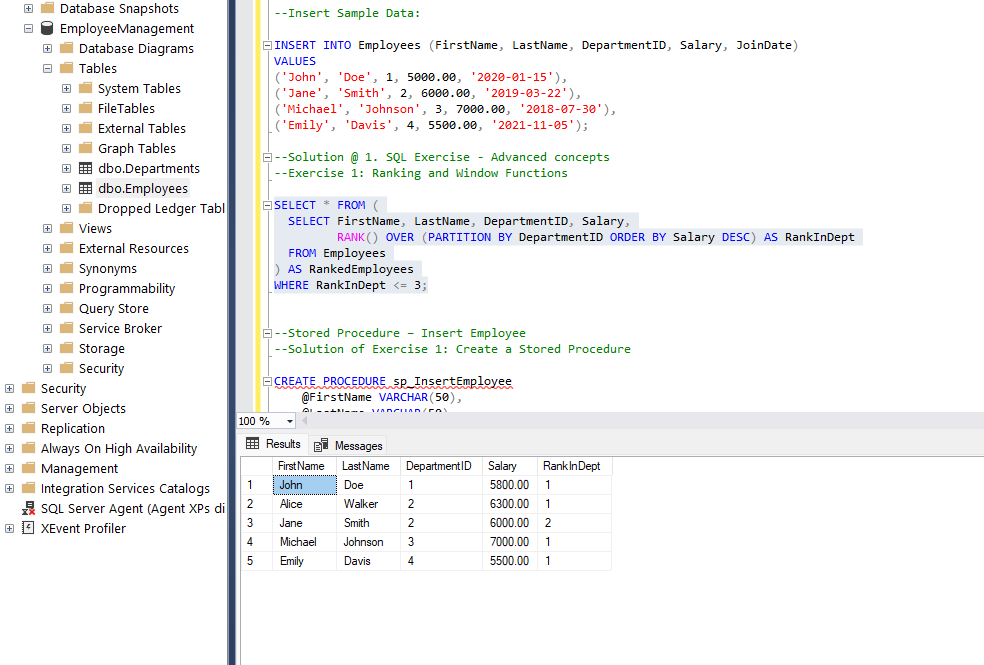
SELECT FirstName, LastName, DepartmentID, Salary,

RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS RankInDept

FROM Employees

) AS RankedEmployees

WHERE RankInDept <= 3;



--Stored Procedure – Insert Employee

--Solution of Exercise 1: Create a Stored Procedure

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

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

VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

EXEC sp\_InsertEmployee 'Alice', 'Walker', 2, 6300.00, '2022-04-01';

--Stored Procedure – Update Salary

--Solution of Exercise 2:

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, 5800.00;

-- Stored Procedure – Count Employees by Department

CREATE PROCEDURE sp\_GetEmployeeCountByDept

@DeptID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DeptID;

END;

EXEC sp\_GetEmployeeCountByDept 2;

-- Execute Procedure to Get Employee Details by Dept

-- Solution of Exercise 4 and Exercise 5:

-- Solution of Exercise 5: Return Data from a Stored Procedure

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DeptID INT

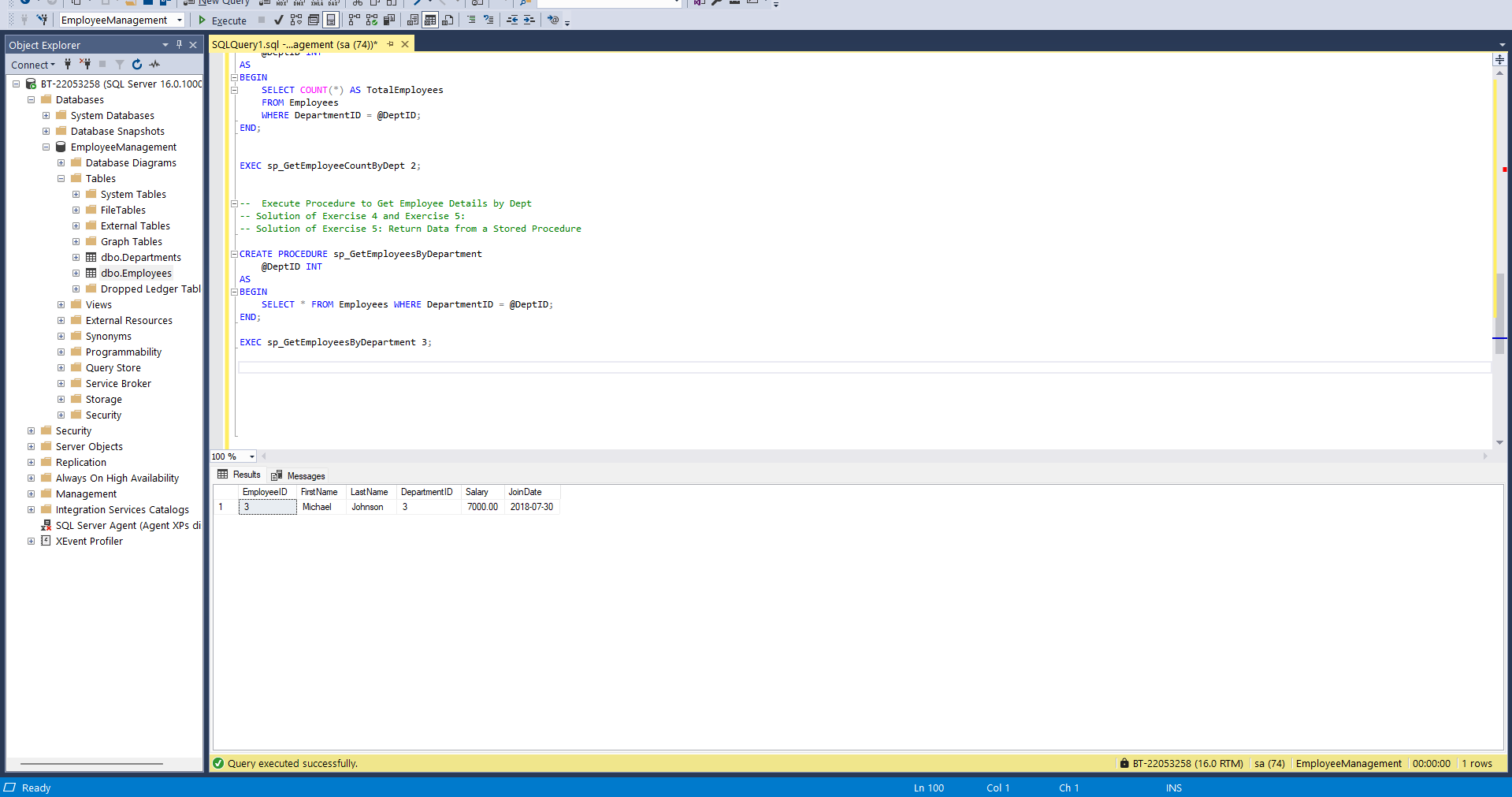
AS

BEGIN

SELECT \* FROM Employees WHERE DepartmentID = @DeptID;

END;

EXEC sp\_GetEmployeesByDepartment 3;



Solution @

|  |  |
| --- | --- |
| 5. SQL Exercise - Functions | Exercise 7: Return Data from a Scalar Function |

-- Created Scalar Function to Calculate Annual Salary

CREATE FUNCTION fn\_CalculateAnnualSalary (

@MonthlySalary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN (@MonthlySalary \* 12);

END;

-- Tested the Scalar Function

SELECT

EmployeeID,

FirstName,

LastName,

Salary AS MonthlySalary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

