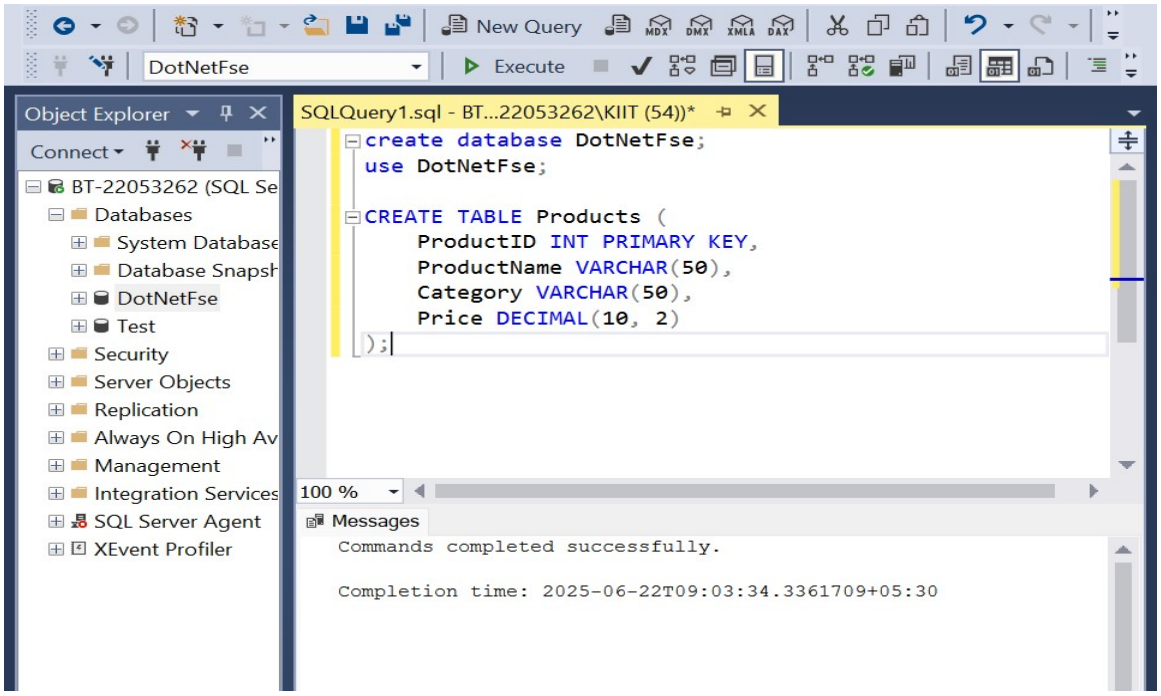
ADVANCE SQL

Exercise1: RANKING AND WINDOW FUNCTIONS

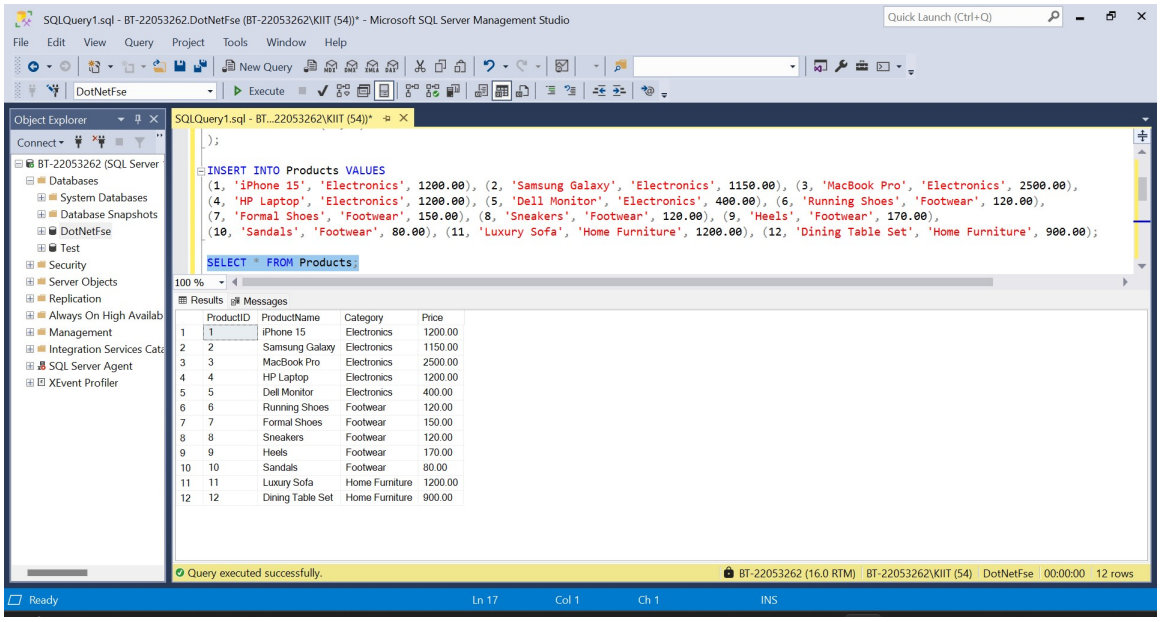
Goal: Use ROW_NUMBER(), RANK(), DENSE_RANK(), OVER(), and PARTITION BY.
Scenario:
Find the top 3 most expensive products in each category using different ranking functions.
Steps:
1. Use ROW_NUMBER() to assign a unique rank within each category.
2. Use RANK() and DENSE_RANK() to compare how ties are handled.
3. Use PARTITION BY Category and ORDER BY Price DESC

```
create database DotNetFse;
use DotNetFse;

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(50),
    Category VARCHAR(50),
    Price DECIMAL(10, 2)
);
```

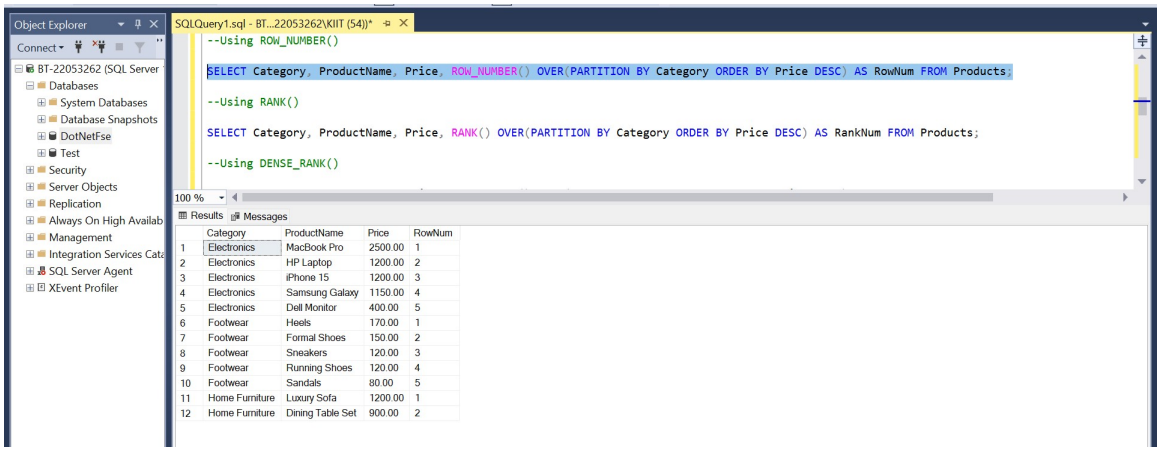



```
INSERT INTO Products VALUES
(1, 'iPhone 15', 'Electronics', 1200.00), (2, 'Samsung Galaxy', 'Electronics',
1150.00), (3, 'MacBook Pro', 'Electronics', 2500.00),
(4, 'HP Laptop', 'Electronics', 1200.00), (5, 'Dell Monitor', 'Electronics',
400.00), (6, 'Running Shoes', 'Footwear', 120.00),
```

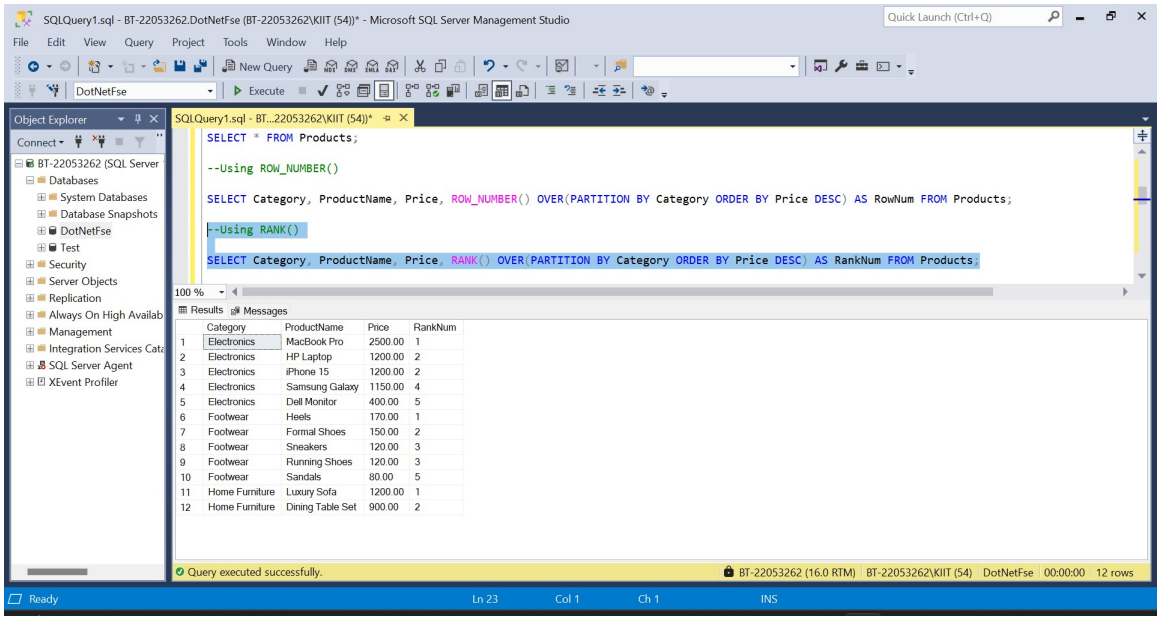
```
(7, 'Formal Shoes', 'Footwear', 150.00), (8, 'Sneakers', 'Footwear', 120.00),
(9, 'Heels', 'Footwear', 170.00),
(10, 'Sandals', 'Footwear', 80.00), (11, 'Luxury Sofa', 'Home Furniture',
1200.00), (12, 'Dining Table Set', 'Home Furniture', 900.00);

SELECT * FROM Products;
```



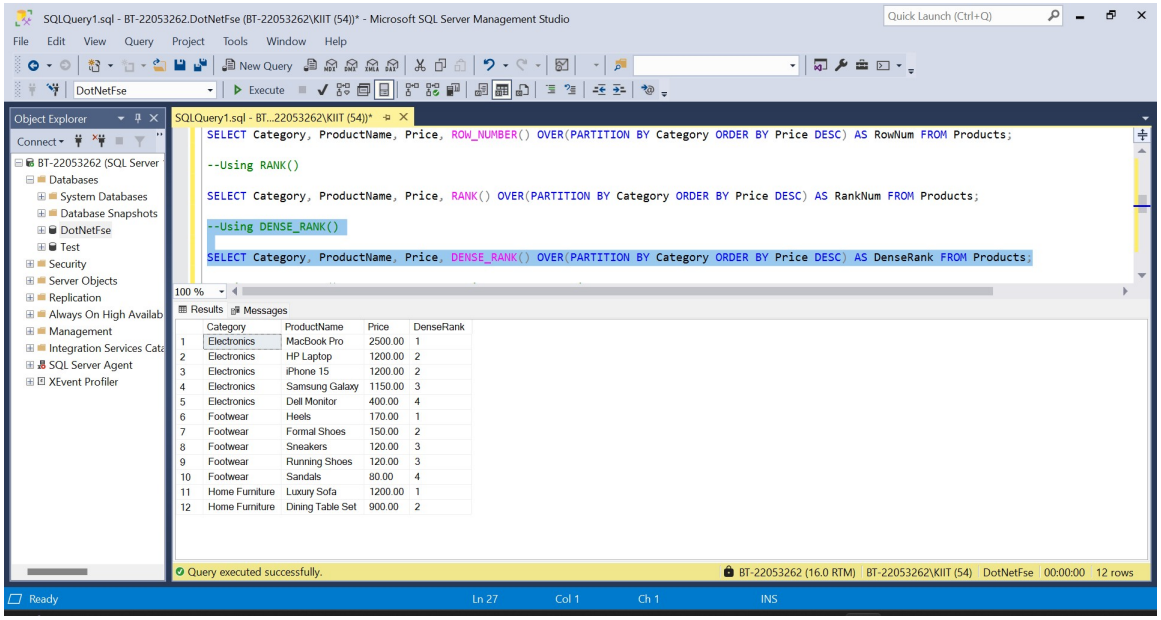

```
--Using ROW_NUMBER()

SELECT Category, ProductName, Price, ROW_NUMBER() OVER(PARTITION BY Category
ORDER BY Price DESC) AS RowNum FROM Products;
```



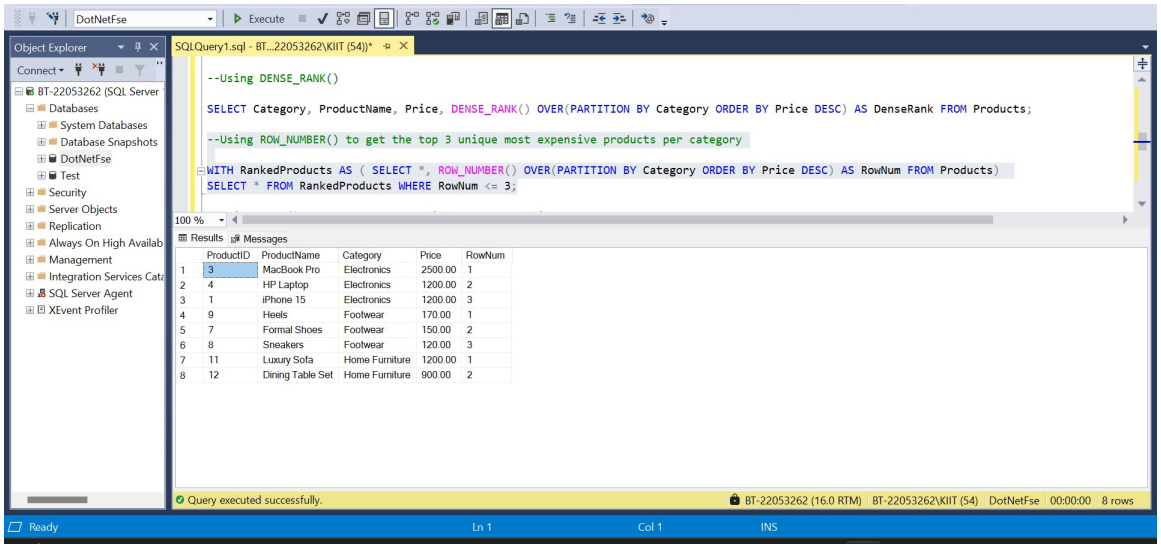

```
--Using RANK()

SELECT Category, ProductName, Price, RANK() OVER(PARTITION BY Category ORDER BY
Price DESC) AS RankNum FROM Products;
```



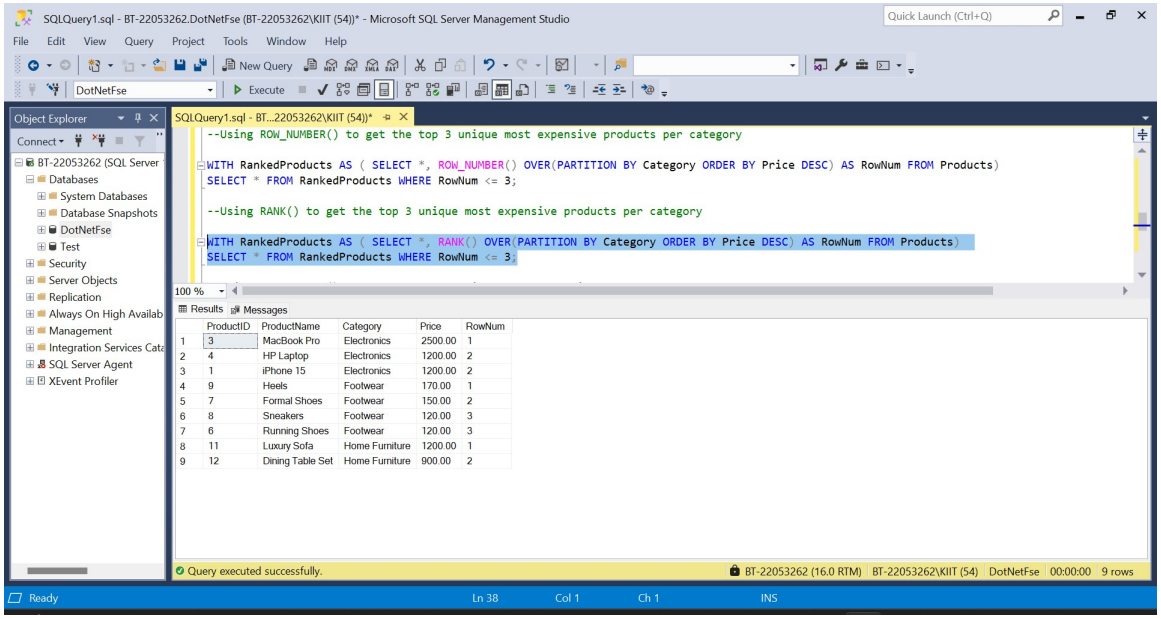

```
--Using DENSE_RANK()

SELECT Category, ProductName, Price, DENSE_RANK() OVER(PARTITION BY Category
ORDER BY Price DESC) AS DenseRank FROM Products;
```



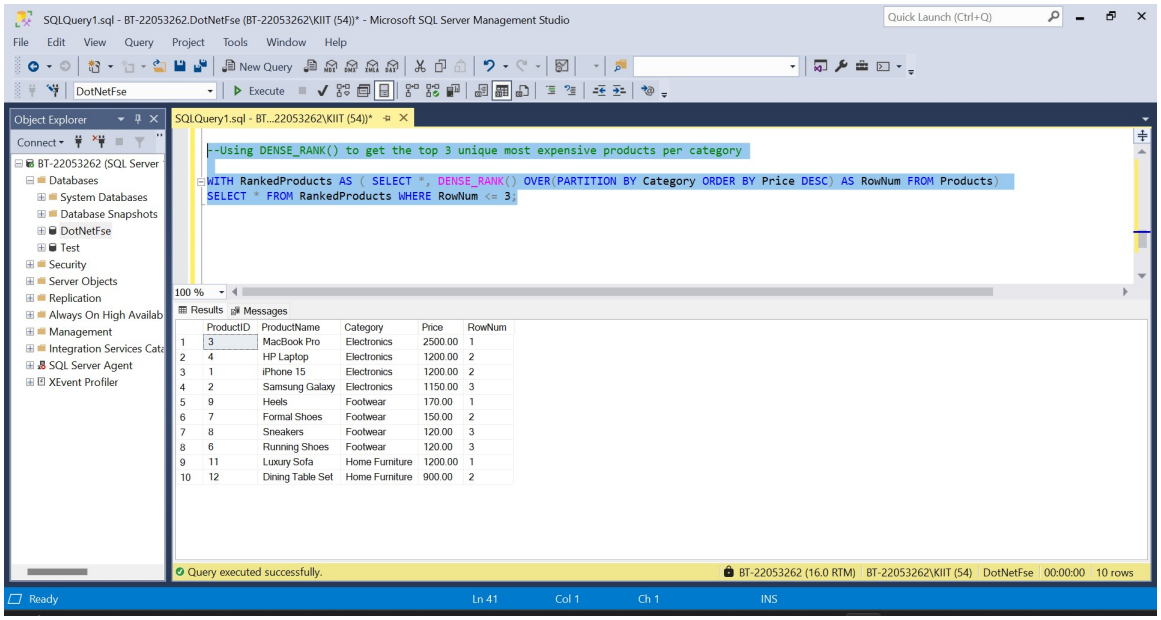

```
--Using ROW_NUMBER() to get the top 3 unique most expensive products per
category

WITH RankedProducts AS ( SELECT *, ROW_NUMBER() OVER(PARTITION BY Category
ORDER BY Price DESC) AS RowNum FROM Products)
SELECT * FROM RankedProducts WHERE RowNum <= 3;
```




```sql
--Using RANK() to get the top 3 unique most expensive products per category

WITH RankedProducts AS ( SELECT *, RANK() OVER(PARTITION BY Category ORDER BY Price DESC) AS RowNum FROM Products)
SELECT * FROM RankedProducts WHERE RowNum <= 3;
```




```sql
--Using DENSE_RANK() to get the top 3 unique most expensive products per category

WITH RankedProducts AS ( SELECT *, DENSE_RANK() OVER(PARTITION BY Category ORDER BY Price DESC) AS RowNum FROM Products)
SELECT * FROM RankedProducts WHERE RowNum <= 3;
```



STORED PROCEDURE

Employee Management System SQL Exercises

Database Schema

The following schema defines the structure for an Employee Management System:

Departments Table

```
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
);
```

Sample Data

The following sample data can be used for testing:

Departments Sample Data

```
INSERT INTO Departments (DepartmentID, DepartmentName) VALUES
(1, 'HR'),
(2, 'Finance'),
(3, 'IT'),
(4, 'Marketing');
```

Employees Sample Data

```
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');
```

Exercises

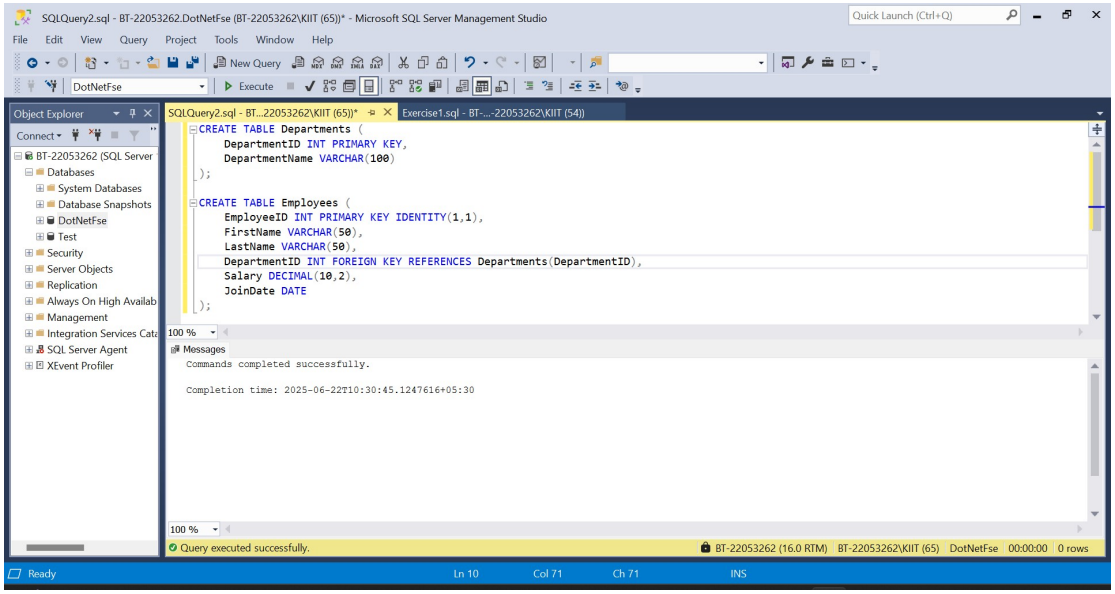
Exercise 1: Create a Stored Procedure

Goal: Create a stored procedure to retrieve employee details by department.

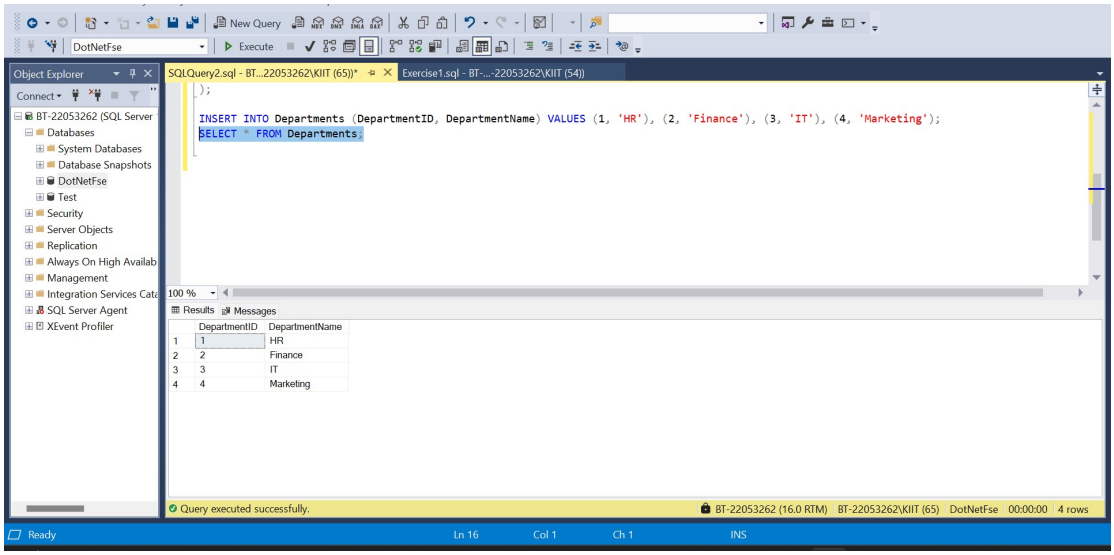
Steps:

1. Define the stored procedure with a parameter for DepartmentID.
2. Write the SQL query to select employee details based on the DepartmentID.
3. Create a stored procedure named `sp_InsertEmployee` with the following code:

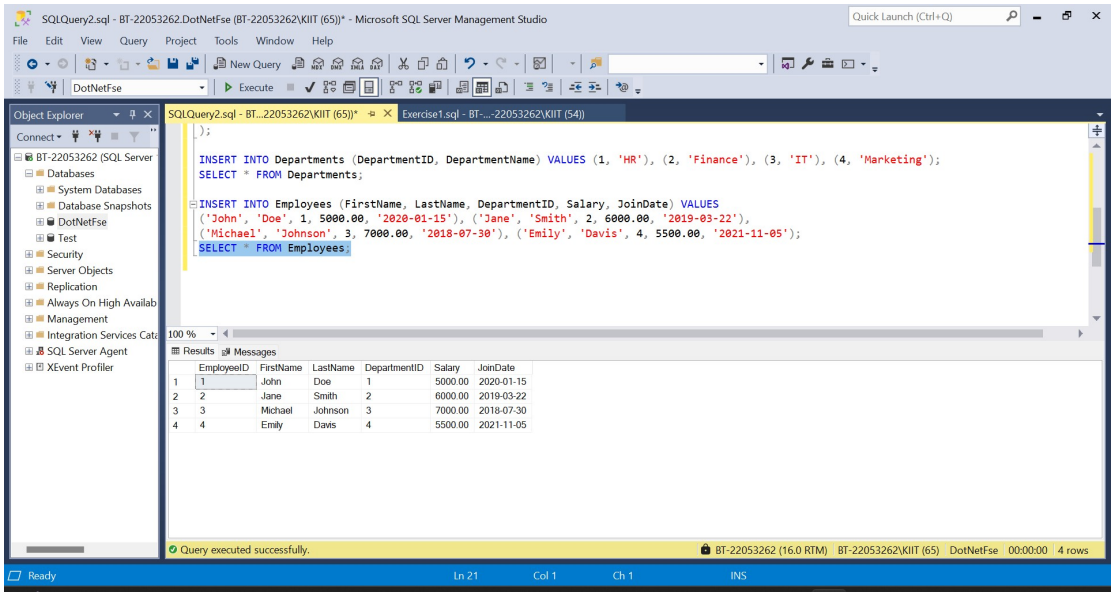
```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100)
);

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');
SELECT * FROM Departments;
```

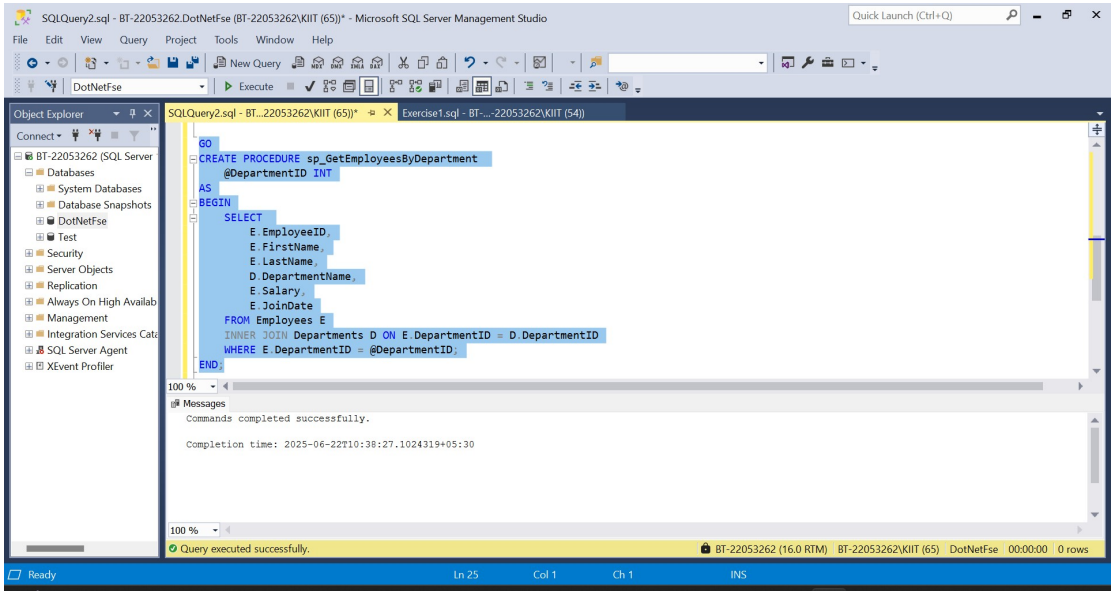



```
INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)
VALUES
('John', 'Doe', 1, 5000.00, '2020-01-15'), ('Jane', 'Smith', 2, 6000.00, '2019-
03-22'),
('Michael', 'Johnson', 3, 7000.00, '2018-07-30'), ('Emily', 'Davis', 4, 5500.00,
'2021-11-05');
SELECT * FROM Employees;
```

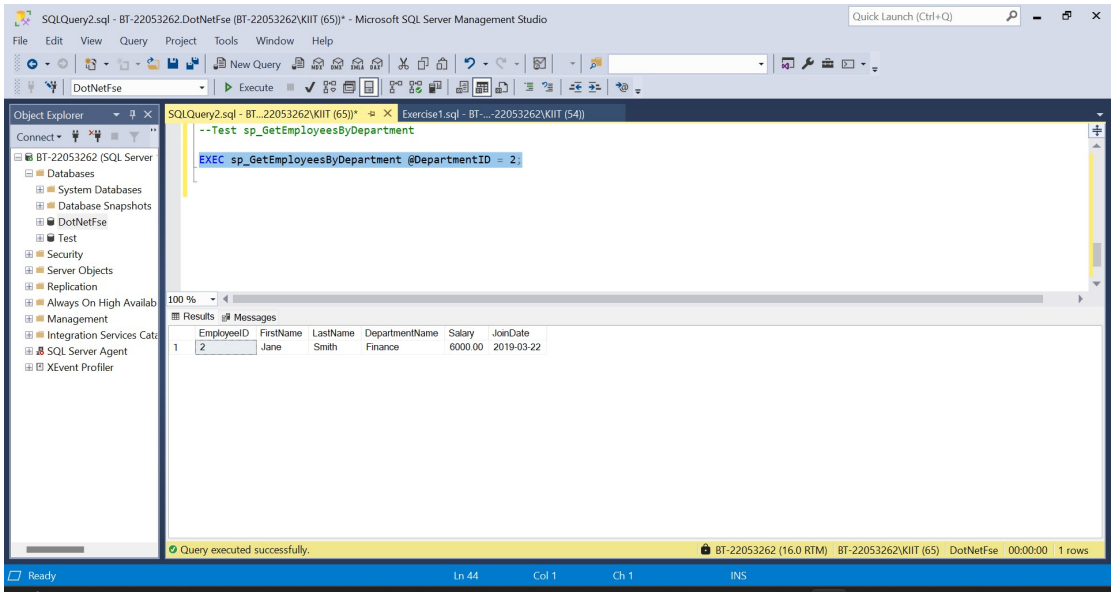



```
--Create Stored Procedure to Get Employees by Department

GO
CREATE PROCEDURE sp_GetEmployeesByDepartment
     @DepartmentID INT
AS
BEGIN
    SELECT
        E.EmployeeID,
        E.FirstName,
```

```
        E.LastName,
        D.DepartmentName,
        E.Salary,
        E.JoinDate
    FROM Employees E
    INNER JOIN Departments D ON E.DepartmentID = D.DepartmentID
    WHERE E.DepartmentID = @DepartmentID;
END;
```

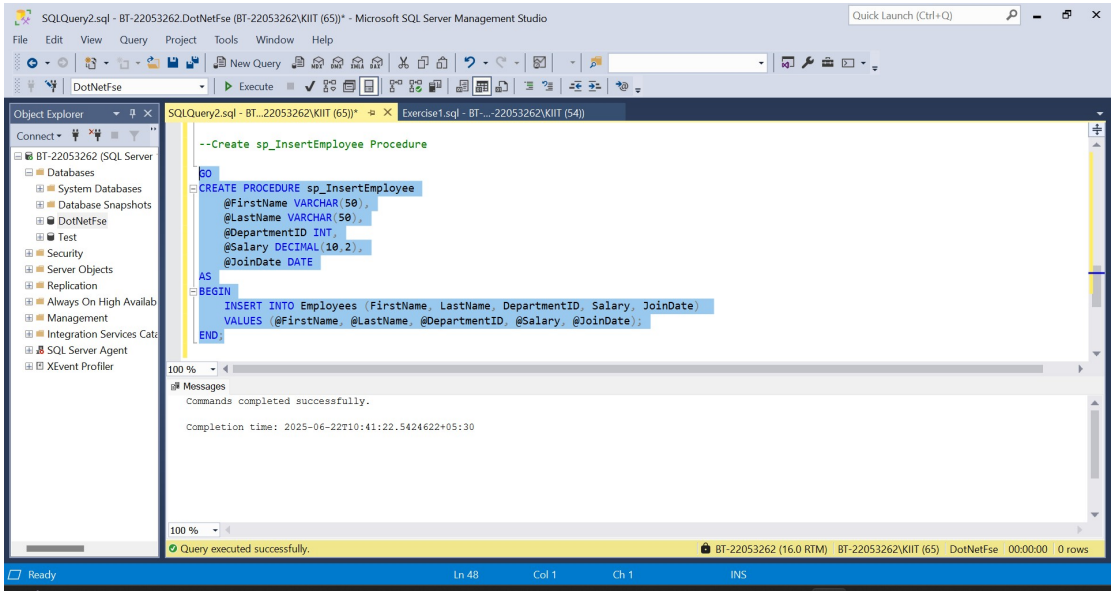



```
--Test sp_GetEmployeesByDepartment

EXEC sp_GetEmployeesByDepartment @DepartmentID = 2;
```

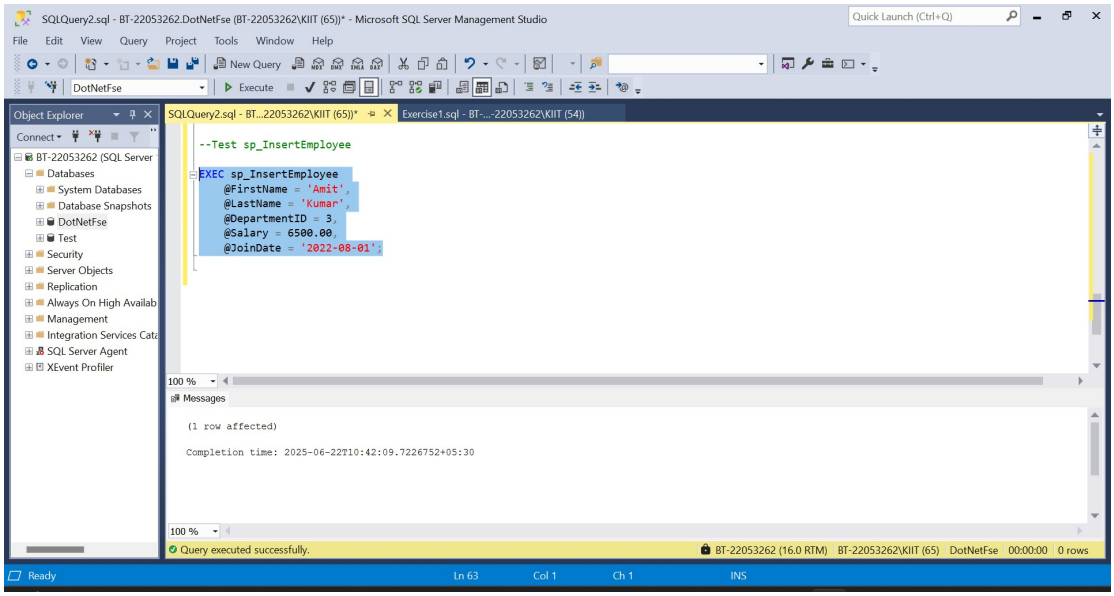



```
--Create sp_InsertEmployee Procedure

GO
```

```sql
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;
```




```sql
--Test sp_InsertEmployee

EXEC sp_InsertEmployee
    @FirstName = 'Amit',
    @LastName = 'Kumar',
    @DepartmentID = 3,
    @Salary = 6500.00,
   @JoinDate = '2022-08-01';
```




```
--Verify

SELECT * FROM Employees;
```