**Advanced SQL**

Suman Sourabh

6363866

Question 1:

**Exercise 1: 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.

**CODE:**

SELECT

ProductID,

ProductName,

Category,

Price,

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

RANK() OVER(PARTITION BY Category ORDER BY Price DESC) AS RankNum,

DENSE\_RANK() OVER(PARTITION BY Category ORDER BY Price DESC) AS DenseRankNum

FROM Products;

WITH RankedProducts AS (

SELECT

ProductID,

ProductName,

Category,

Price,

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

FROM Products

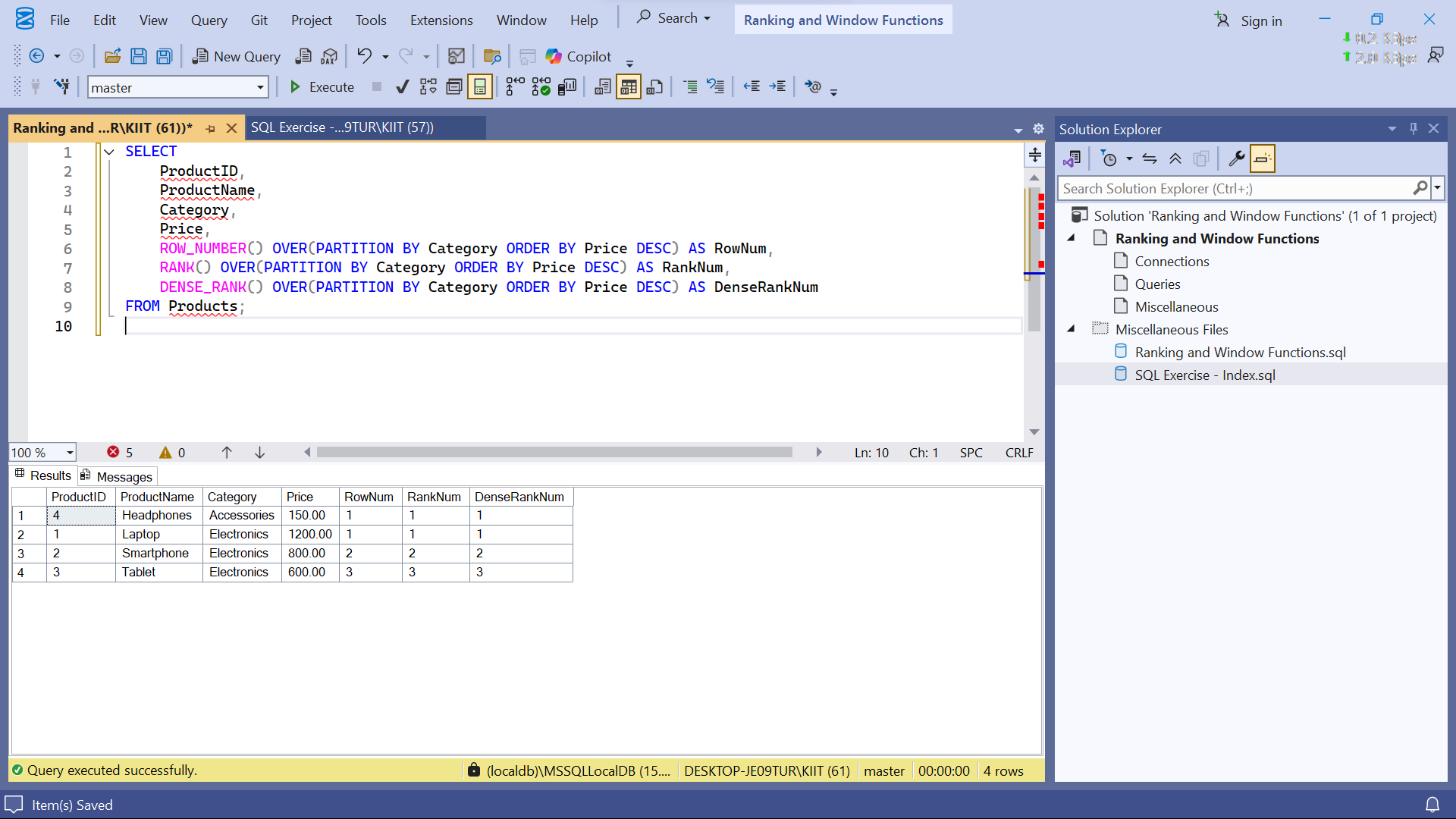
)

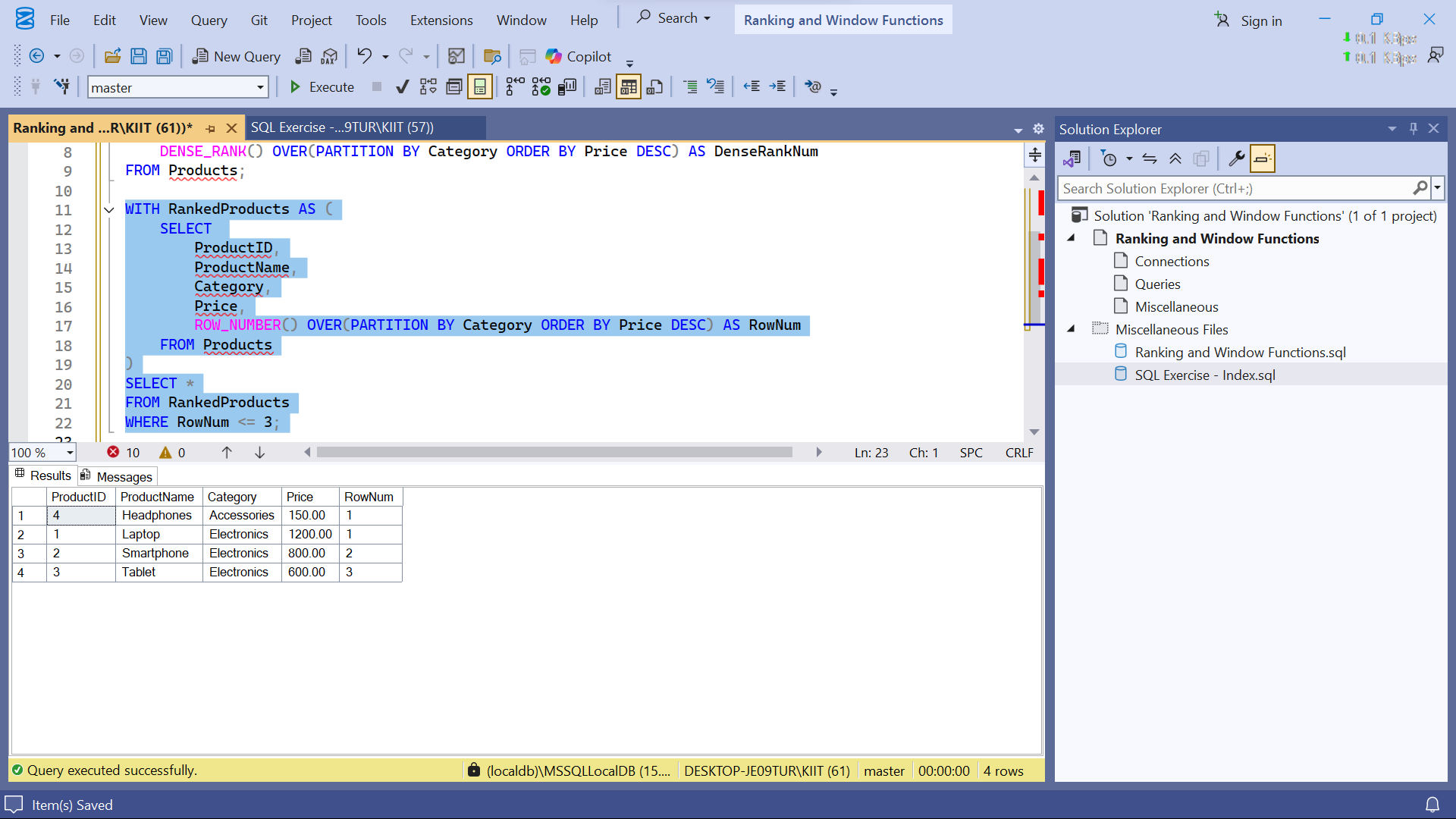
SELECT \*

FROM RankedProducts

WHERE RowNum <= 3;

**OUTPUT:**





**Question 2:**

**SQL Exercise – Index**

**CODE:**

-- Drop foreign key constraints first to avoid dependency issues

IF OBJECT\_ID('OrderDetails', 'U') IS NOT NULL

DROP TABLE OrderDetails;

IF OBJECT\_ID('Orders', 'U') IS NOT NULL

DROP TABLE Orders;

IF OBJECT\_ID('Products', 'U') IS NOT NULL

DROP TABLE Products;

IF OBJECT\_ID('Customers', 'U') IS NOT NULL

DROP TABLE Customers;

-- 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: 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

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: 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

CREATE NONCLUSTERED INDEX IX\_Orders\_OrderDate

ON Orders(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

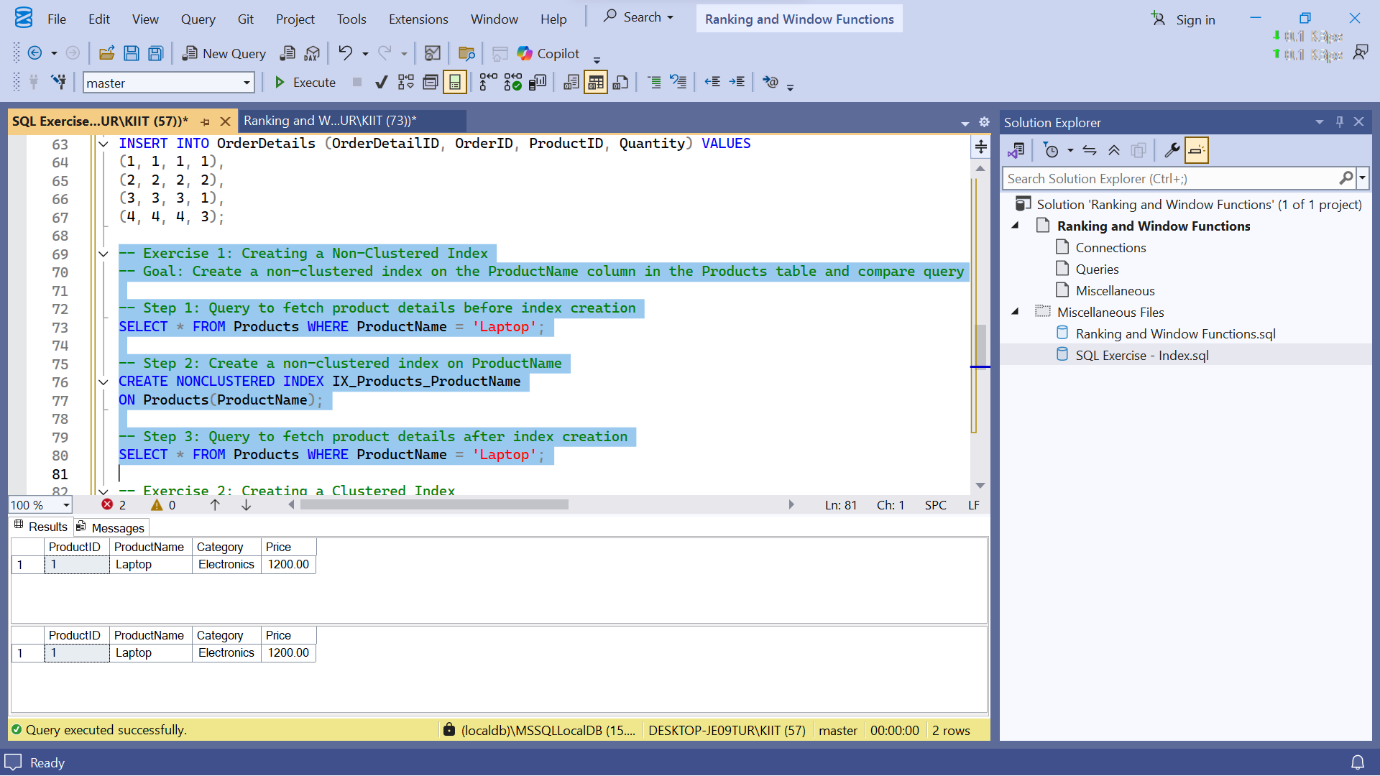
CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID\_OrderDate

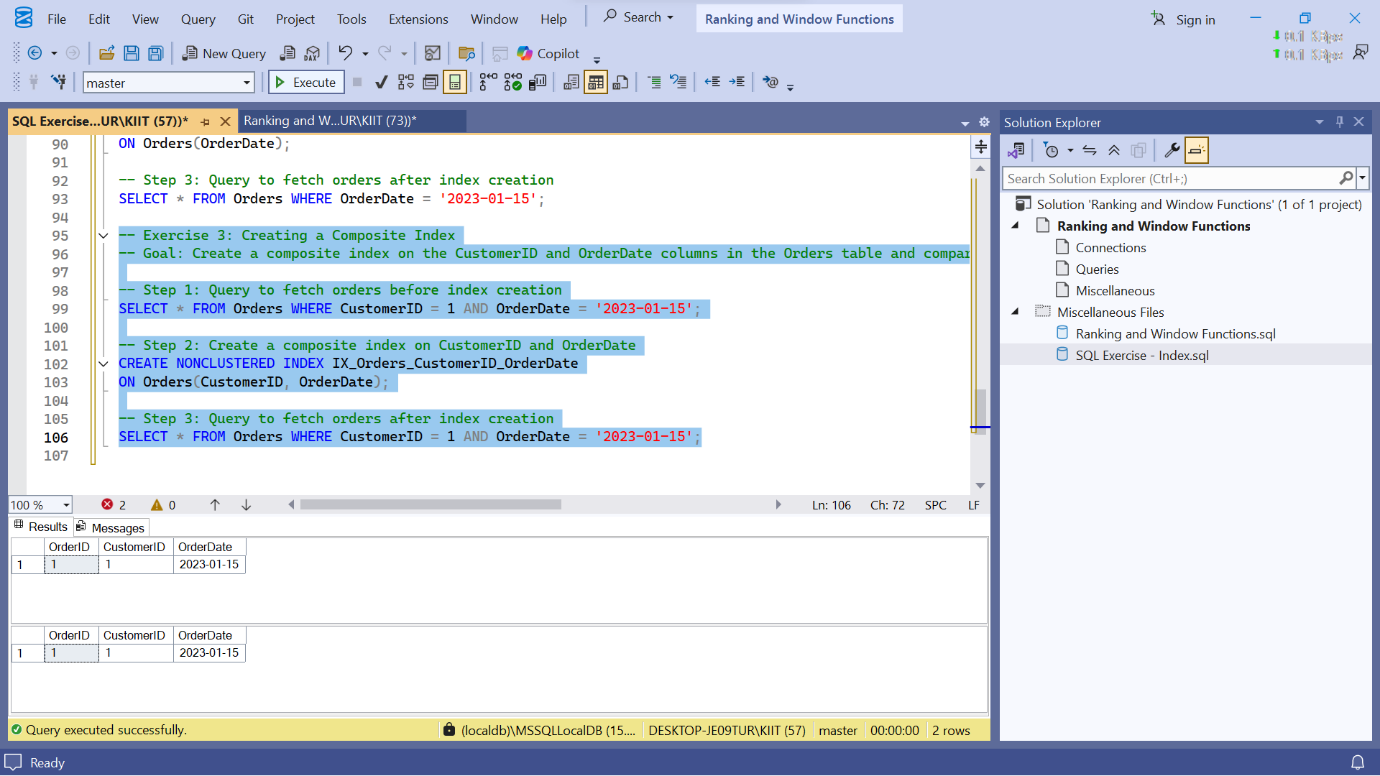
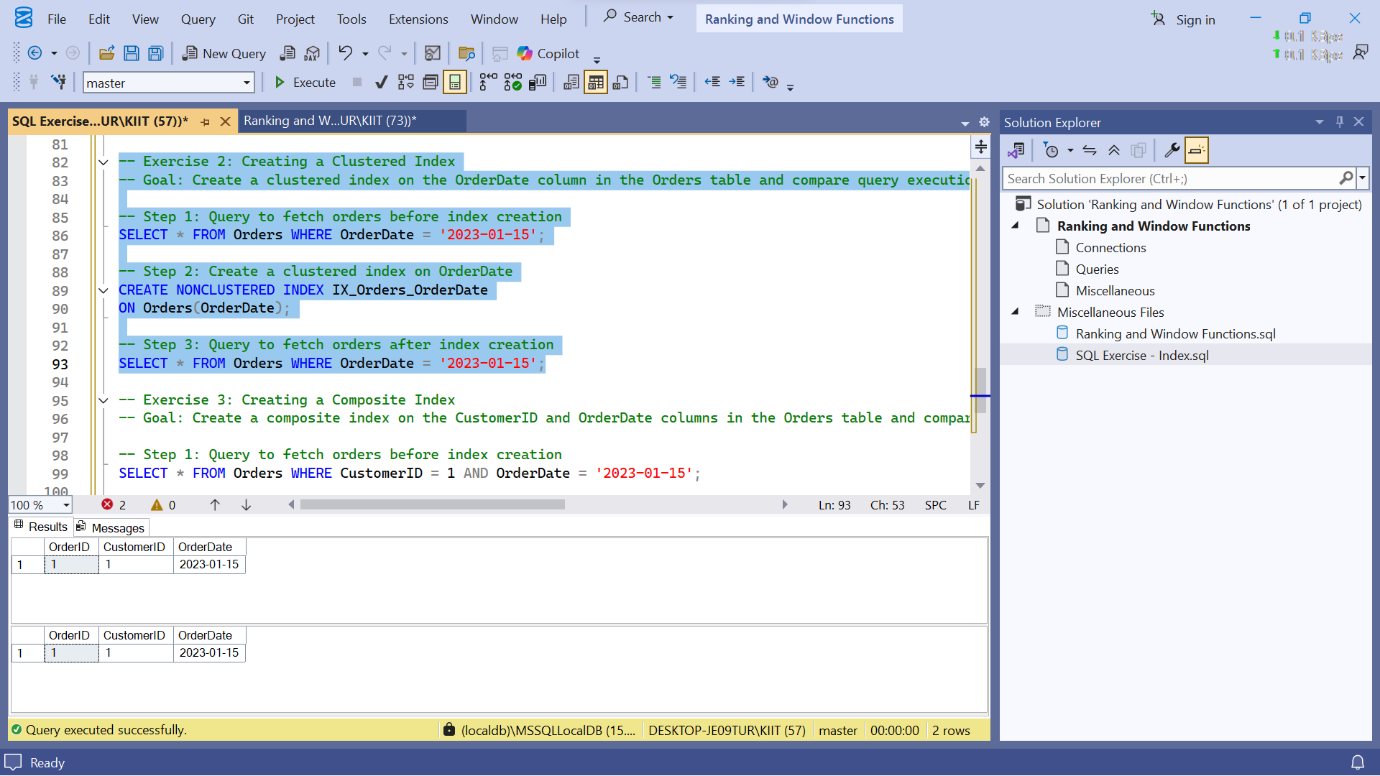
ON Orders(CustomerID, OrderDate);

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

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

OUTPUT:





Question 3:

**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 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;

CODE:

-- Departments Table

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

-- Employees Table

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY IDENTITY(1,1), -- Optional auto-increment

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10,2),

JoinDate DATE

);

-- Departments

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

-- Employees

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');

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DeptID INT

AS

BEGIN

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

E.Salary,

E.JoinDate,

D.DepartmentName

FROM Employees E

INNER JOIN Departments D ON E.DepartmentID = D.DepartmentID

WHERE E.DepartmentID = @DeptID;

END;

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

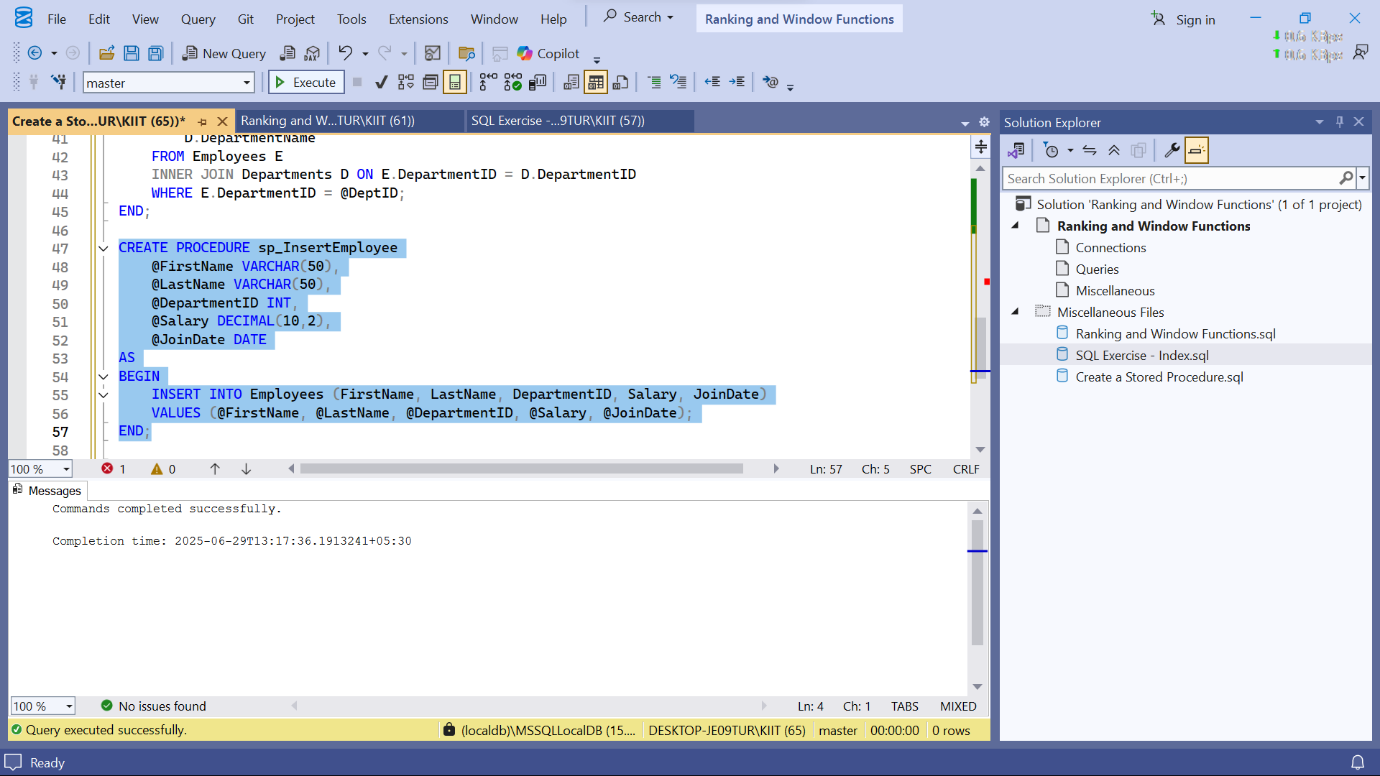
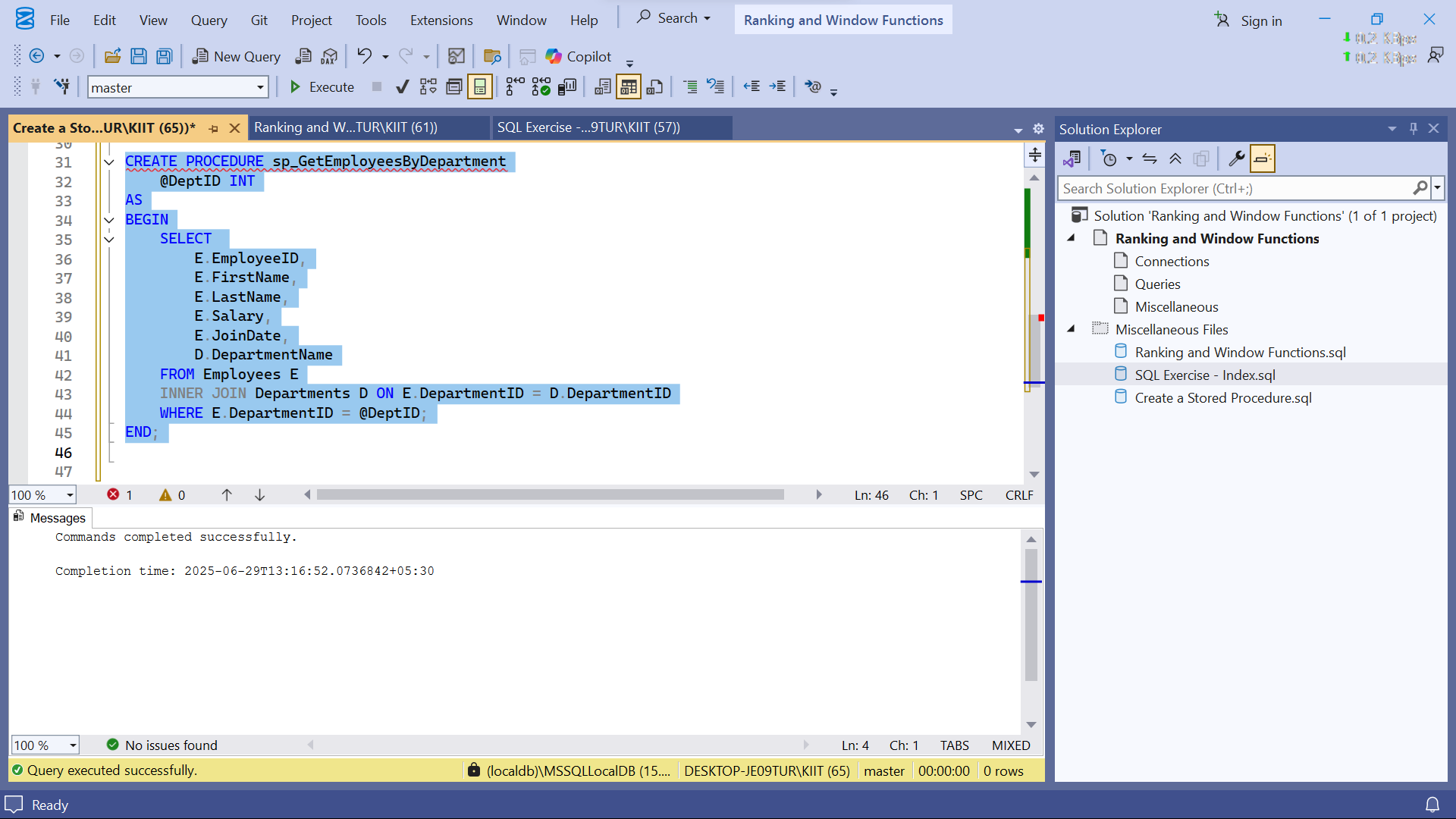
BEGIN

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

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

END;

OUTPUT:



Question -4

**Exercise 4: Execute a Stored Procedure**

Goal: Execute the stored procedure to retrieve employee details for a specific department.

Steps:

1. Write the SQL command to execute the stored procedure with a DepartmentID

parameter.

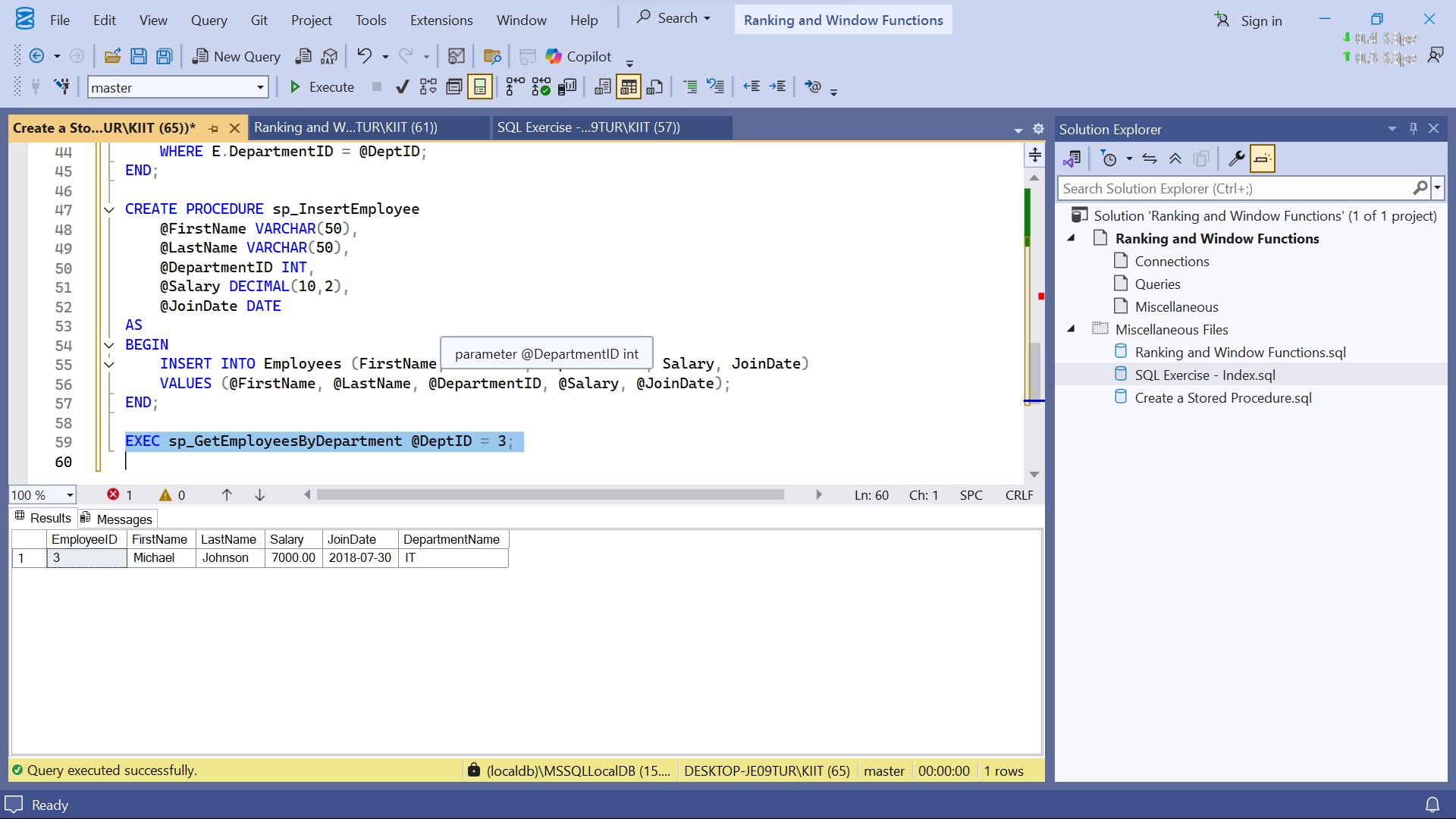
2. Execute the command and review the results.

CODE:

-- Execute a Stored Procedure

EXEC sp\_GetEmployeesByDepartment @DeptID = 3;

OUTPUT:



Question-5:

**Exercise 5: Return Data from a Stored Procedure**

Goal: Create a stored procedure that returns the total number of employees in a

department.

Steps:

1. Define the stored procedure with a parameter for DepartmentID.

2. Write the SQL query to count the number of employees in the specified department.

3. Save the stored procedure by executing the Stored procedure content.

CODE:

-- Return Data from a Stored Procedure

CREATE PROCEDURE sp\_CountEmployeesByDepartment

@DeptID INT

AS

BEGIN

SELECT

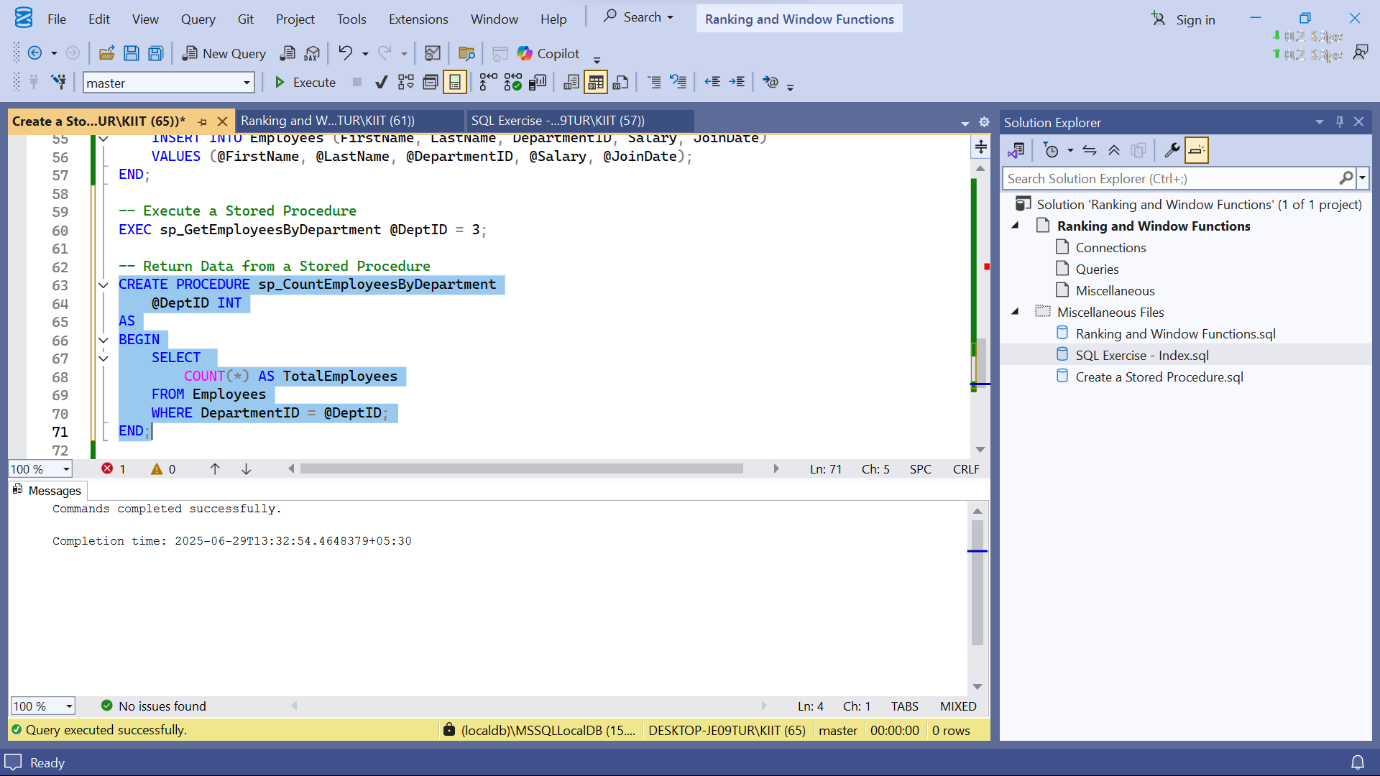
COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DeptID;

END;

OUTPUT:



Question-6:

**Exercise 1: Create a Scalar Function**

Goal: Create a scalar function to calculate the annual salary of an employee.

Steps:

1. Define a scalar function named `fn\_CalculateAnnualSalary`.

2. The function should take `Salary` as input and return `Salary \* 12`.

3. Test the function by selecting the annual salary for each employee.

CODE:

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

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

);

-- Exercise 1: Create a Scalar Function

CREATE FUNCTION fn\_CalculateAnnualSalary (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 12;

END;

-- Test

SELECT

FirstName,

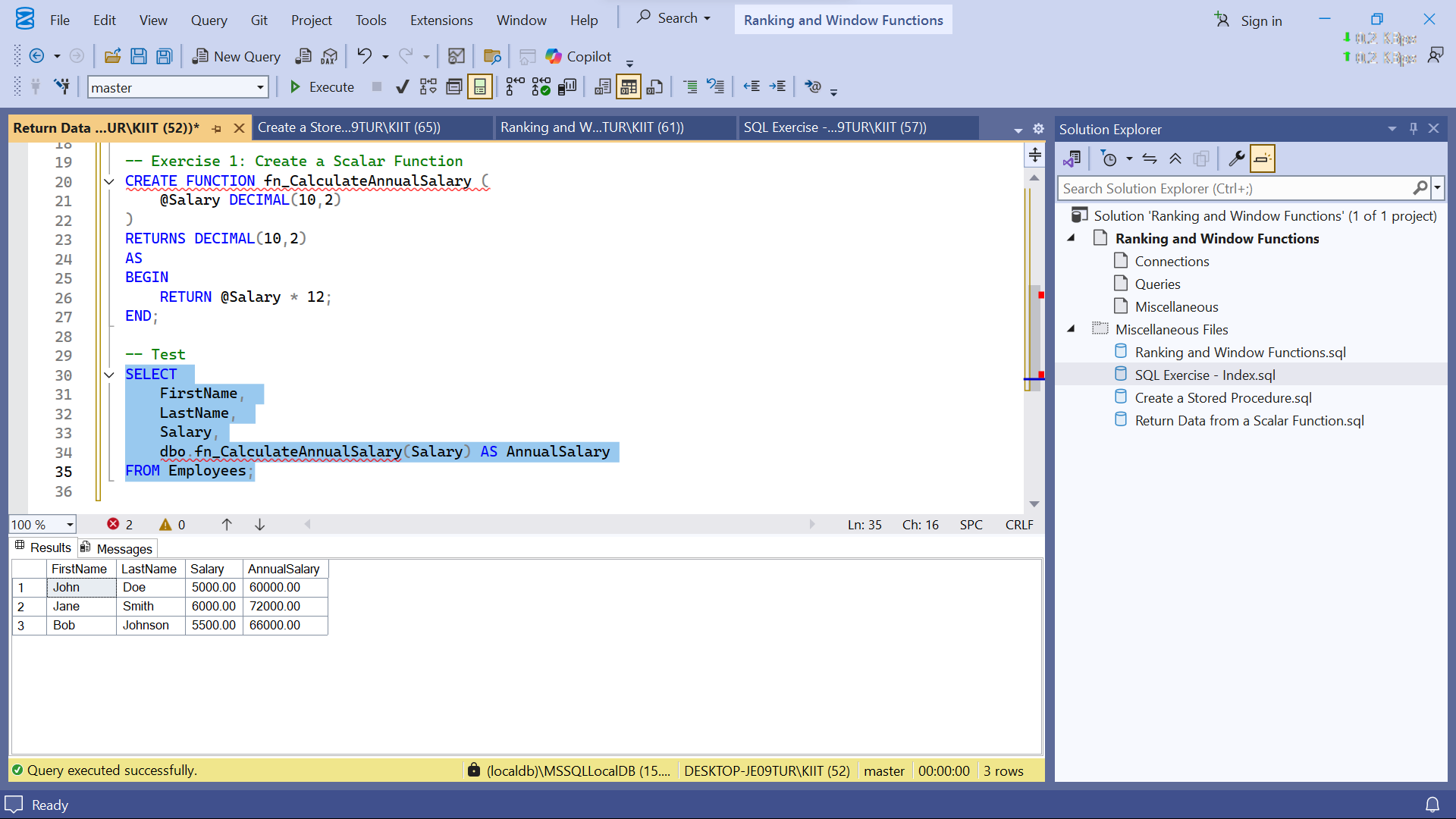
LastName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

OUTPUT:



**Exercise 2: Create a Table-Valued Function**

Goal: Create a table-valued function to return employees in a specific department.

Steps:

1. Define a table-valued function named `fn\_GetEmployeesByDepartment`.

2. The function should take `DepartmentID` as input and return a table with employee

details.

3. Test the function by selecting employees from the IT department.

CODE:

--Exercise 2: Create a Table-Valued Function

CREATE FUNCTION fn\_GetEmployeesByDepartment (

@DeptID INT

)

RETURNS TABLE

AS

RETURN (

SELECT

EmployeeID, FirstName, LastName, Salary, JoinDate

FROM Employees

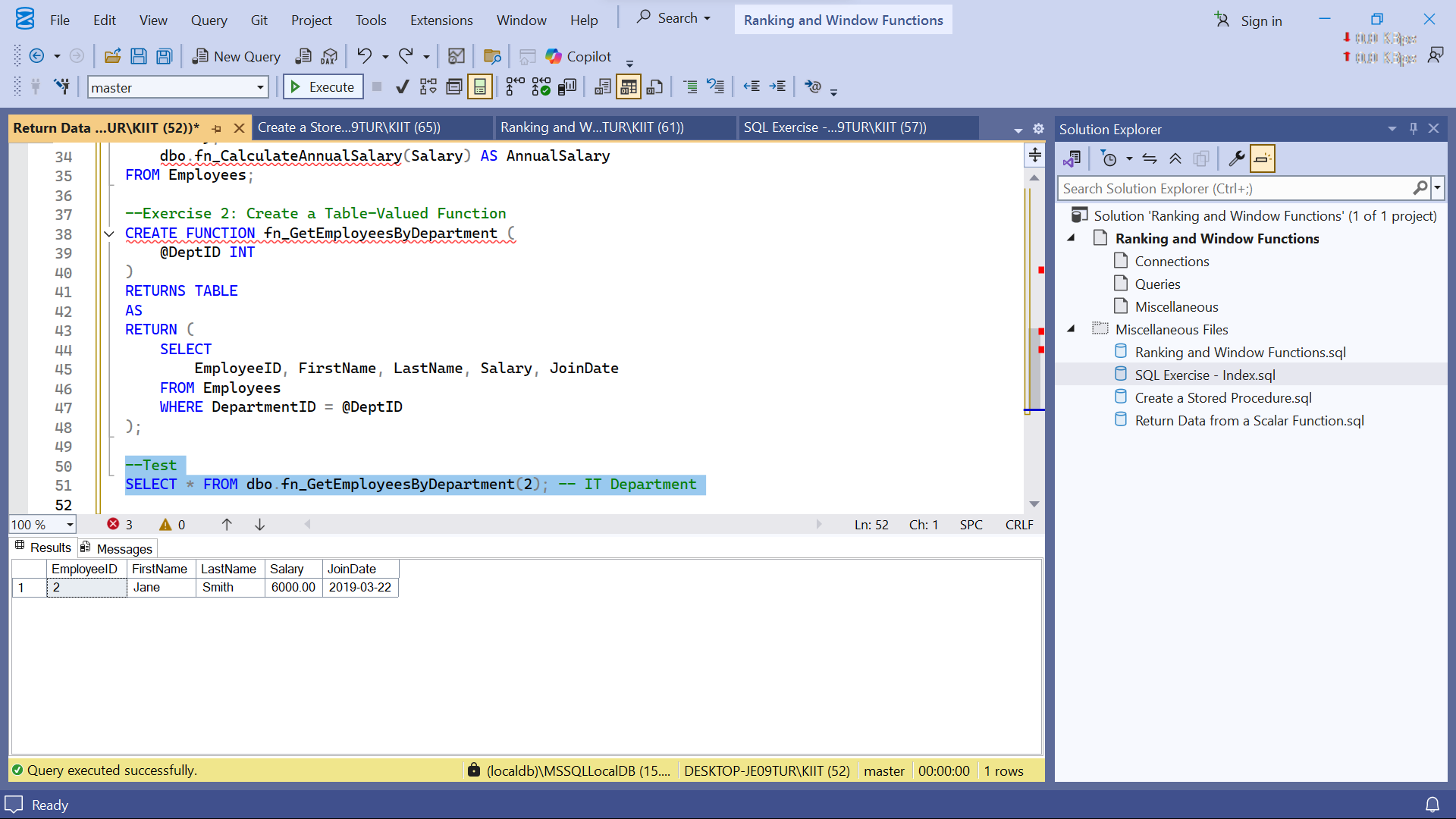
WHERE DepartmentID = @DeptID

);

--Test

SELECT \* FROM dbo.fn\_GetEmployeesByDepartment(2); -- IT Department

OUTPUT:



**Exercise 3: Create a User-Defined Function**

Goal: Create a user-defined function to calculate the bonus for an employee.

Steps:

1. Define a user-defined function named `fn\_CalculateBonus`.

2. The function should take `Salary` as input and return `Salary \* 0.10`.

3. Test the function by selecting the bonus for each employee.

CODE:

-- Exercise 3: Create a User-Defined Function

CREATE FUNCTION fn\_CalculateBonus (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

-- Test

SELECT

FirstName,

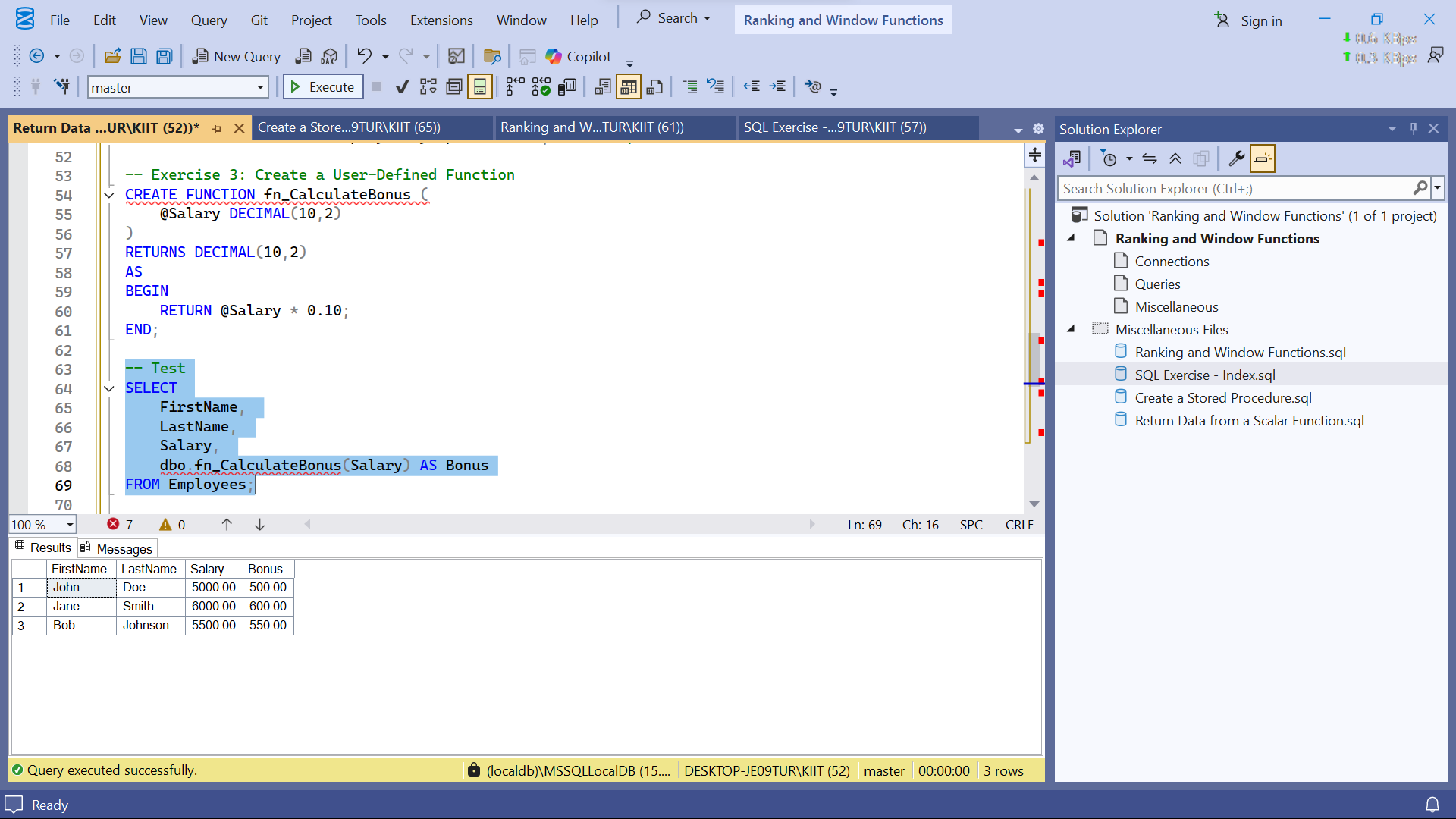
LastName,

Salary,

dbo.fn\_CalculateBonus(Salary) AS Bonus

FROM Employees;

OUTPUT:



**Exercise 4: Modify a User-Defined Function**

Goal: Modify the `fn\_CalculateBonus` function to return `Salary \* 0.15`.

Steps:

1. Alter the `fn\_CalculateBonus` function to return `Salary \* 0.15`.

2. Test the modified function by selecting the bonus for each employee.

CODE:

--Exercise 4: Modify a User-Defined Function

ALTER FUNCTION fn\_CalculateBonus (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

--Test

SELECT

FirstName,

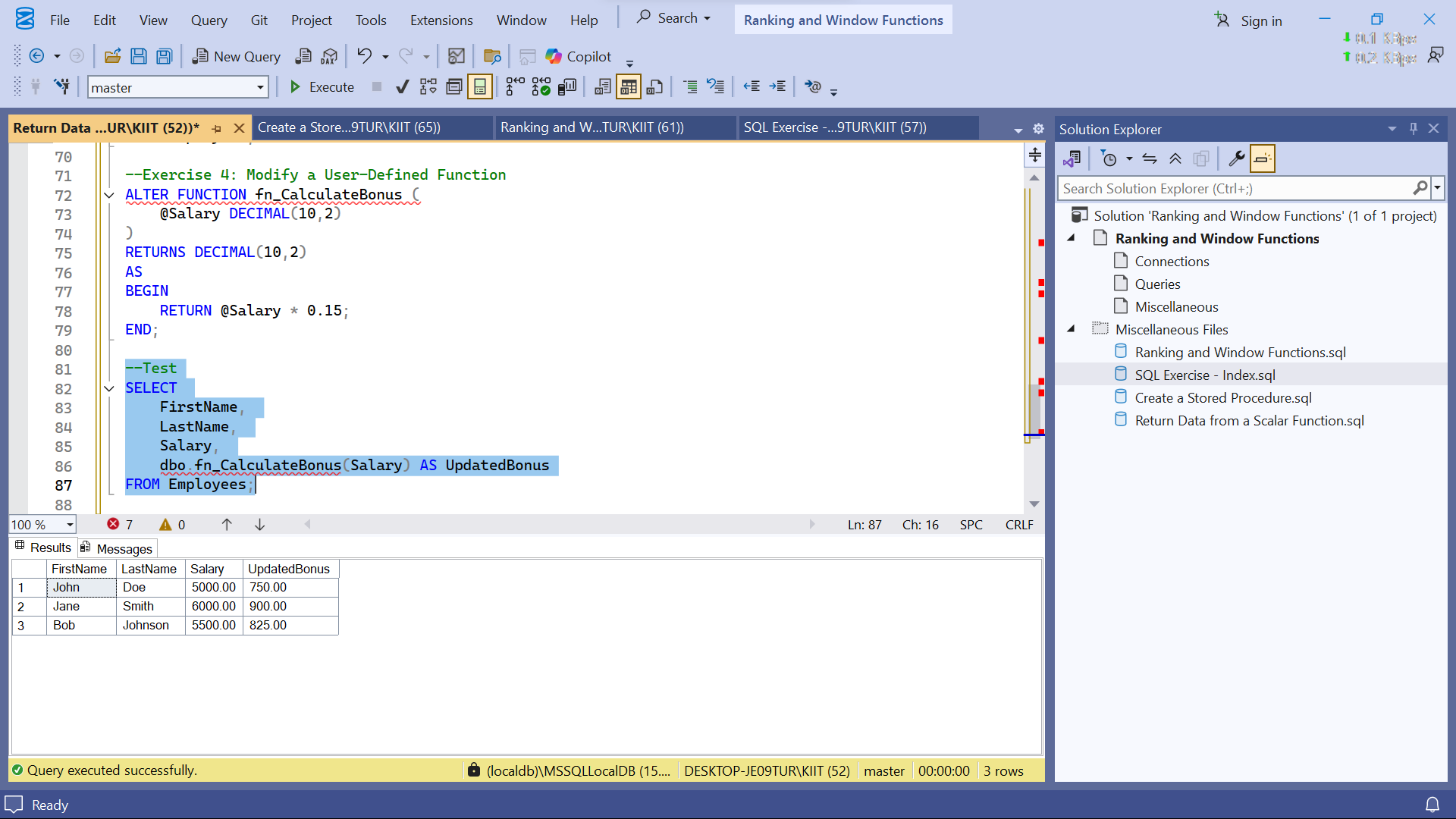
LastName,

Salary,

dbo.fn\_CalculateBonus(Salary) AS UpdatedBonus

FROM Employees;

OUTPUT:



**Exercise 5: Delete a User-Defined Function**

Goal: Delete the `fn\_CalculateBonus` function.

Steps: 1. Drop the `fn\_CalculateBonus` function.

2. Verify that the function has been deleted.

CODE:

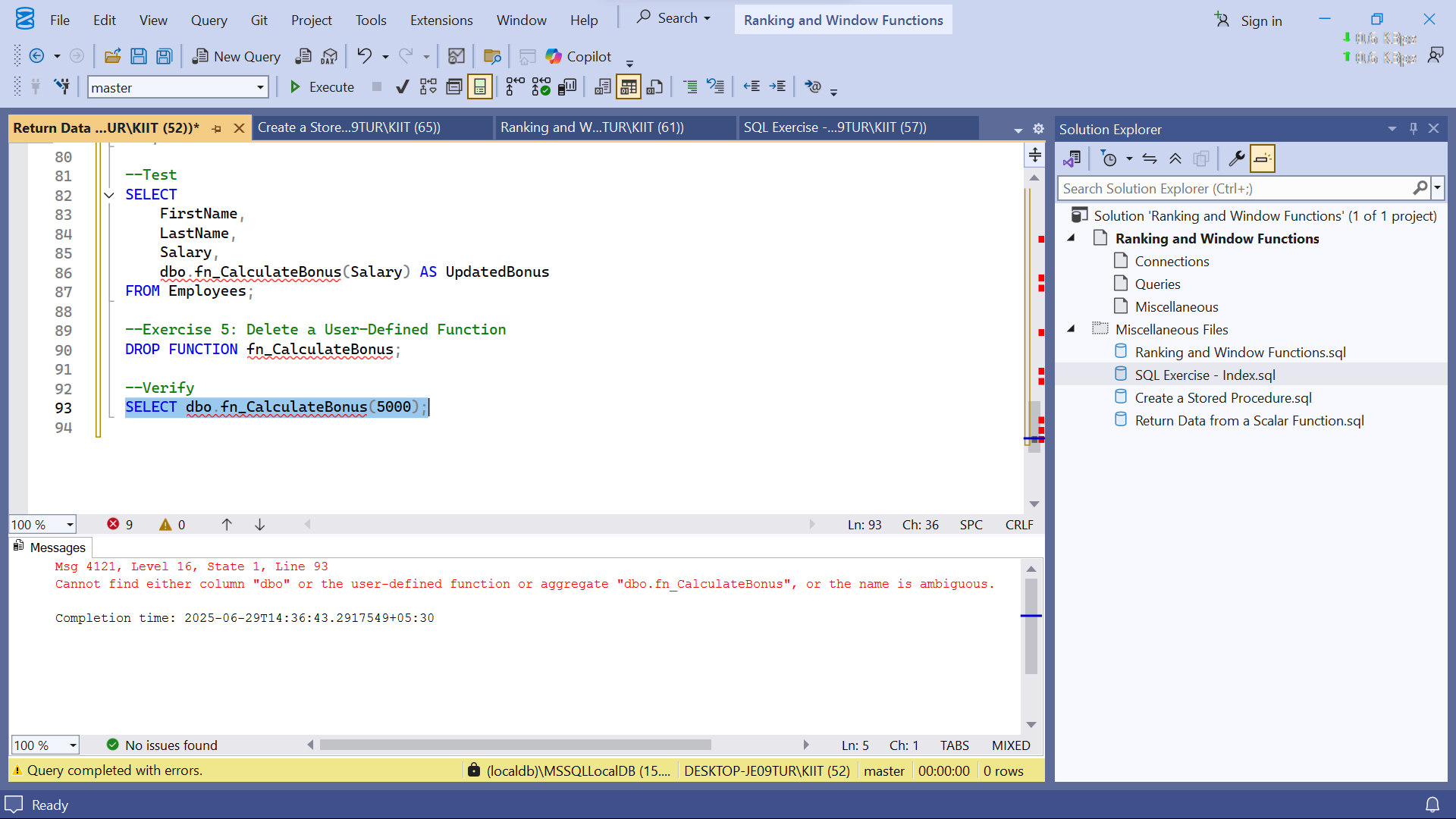
--Exercise 5: Delete a User-Defined Function

DROP FUNCTION fn\_CalculateBonus;

--Verify

SELECT dbo.fn\_CalculateBonus(5000);

OUTPUT:



**Exercise 6: Execute a User-Defined Function**

Goal: Execute the `fn\_CalculateAnnualSalary` function.

Steps:

1. Use the `fn\_CalculateAnnualSalary` function to calculate the annual salary for each

employee.

2. Verify the results.

CODE:

--Exercise 6: Execute a User-Defined Function

SELECT

EmployeeID,

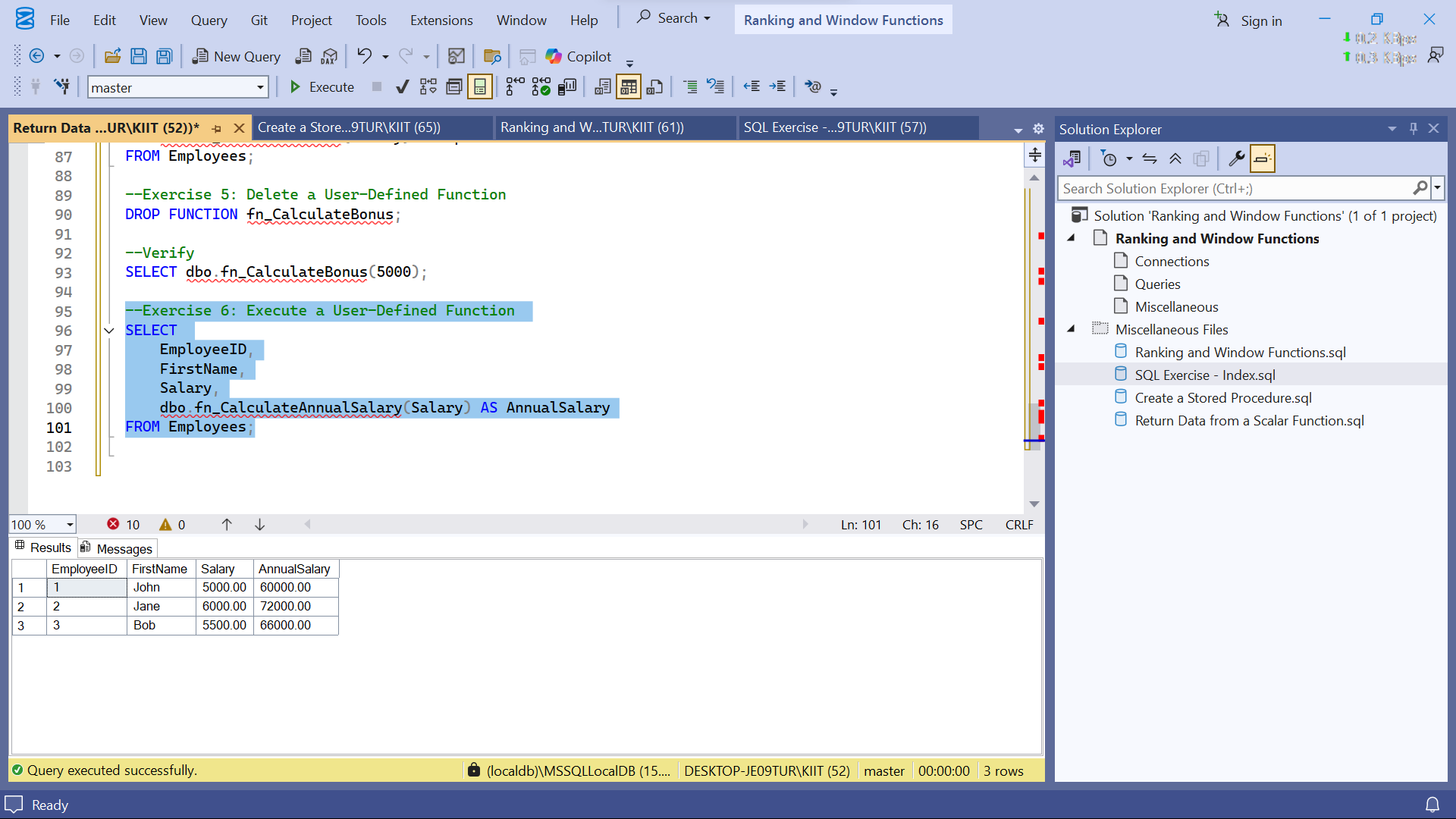
FirstName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

OUTPUT:



**Exercise 7: Return Data from a Scalar Function**

Goal: Return the annual salary for a specific employee using `fn\_CalculateAnnualSalary`.

Steps:

1. Execute the `fn\_CalculateAnnualSalary` function for an employee with `EmployeeID = 1`.

2. Verify the result.

CODE:

--Exercise 7: Return Data from a Scalar Function

SELECT

FirstName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

OUTPUT:

