1. Exercise-1

Advanced SQL Exercises for Online Retail Store

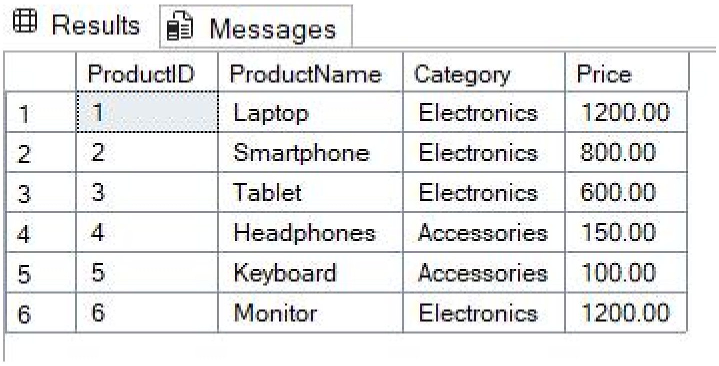
Exercise 1: Ranking and Window Functions

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

Solution:

Table created and data inserted:

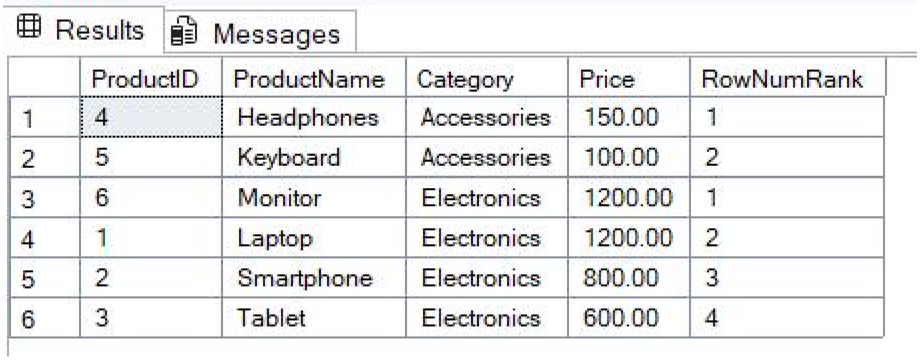
```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Category VARCHAR(50),
    Price DECIMAL(10, 2)
);

INSERT INTO Products VALUES
(1, 'Laptop', 'Electronics', 1200.00),
(2, 'Smartphone', 'Electronics', 800.00),
(3, 'Tablet', 'Electronics', 600.00),
(4, 'Headphones', 'Accessories', 150.00),
(5, 'Keyboard', 'Accessories', 100.00),
(6, 'Monitor', 'Electronics', 1200.00);
```



	ProductID	ProductName	Category	Price
1	1	Laptop	Electronics	1200.00
2	2	Smartphone	Electronics	800.00
3	3	Tablet	Electronics	600.00
4	4	Headphones	Accessories	150.00
5	5	Keyboard	Accessories	100.00
6	6	Monitor	Electronics	1200.00

#1

```
--1. ROW_NUMBER(): Assigns unique ranks per category, ignoring ties.
SELECT
    ProductID, ProductName, Category, Price,
    ROW_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS
RowNumRank
FROM Products;
```

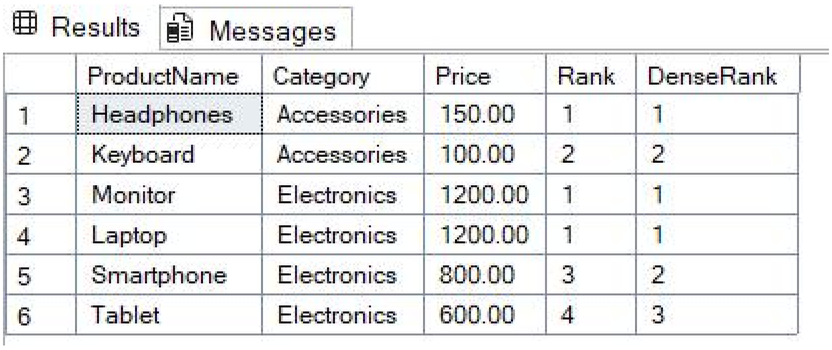


	ProductID	ProductName	Category	Price	RowNumRank
1	4	Headphones	Accessories	150.00	1
2	5	Keyboard	Accessories	100.00	2
3	6	Monitor	Electronics	1200.00	1
4	1	Laptop	Electronics	1200.00	2
5	2	Smartphone	Electronics	800.00	3
6	3	Tablet	Electronics	600.00	4

Explanation: Assigns a unique sequential rank for each product within its category, even if prices are tied. No ties in ranking

#2

```
--2.  Use RANK() and DENSE_RANK() to compare how ties are handled
SELECT
    ProductName, Category, Price,
    RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS Rank,
    DENSE_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS
DenseRank
FROM Products;
```

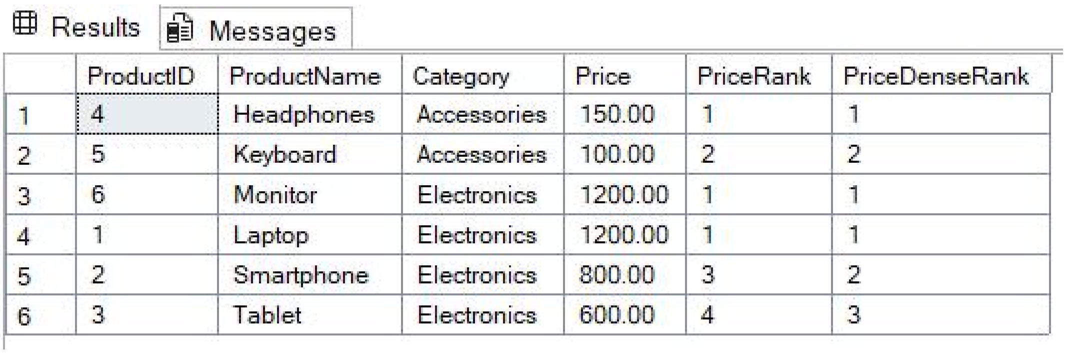


	ProductName	Category	Price	Rank	DenseRank
1	Headphones	Accessories	150.00	1	1
2	Keyboard	Accessories	100.00	2	2
3	Monitor	Electronics	1200.00	1	1
4	Laptop	Electronics	1200.00	1	1
5	Smartphone	Electronics	800.00	3	2
6	Tablet	Electronics	600.00	4	3

Explanation: DENSE_RANK(): Does not skip ranks; sequence remains continuous.

#3.

```
--3. Use PARTITION BY Category and ORDER BY Price DESC

SELECT
    ProductID,
    ProductName,
    Category,
    Price,
    RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS PriceRank,
    DENSE_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS
PriceDenseRank
FROM Products;
```



	ProductID	ProductName	Category	Price	PriceRank	PriceDenseRank
1	4	Headphones	Accessories	150.00	1	1
2	5	Keyboard	Accessories	100.00	2	2
3	6	Monitor	Electronics	1200.00	1	1
4	1	Laptop	Electronics	1200.00	1	1
5	2	Smartphone	Electronics	800.00	3	2
6	3	Tablet	Electronics	600.00	4	3

Explanation: Using PARTITION BY Category and ORDER BY Price DESC ranks products by descending price within each category group.

----

2. Exercise-2(SQL Exercise - Index):

Table Creation and Data Insertion:

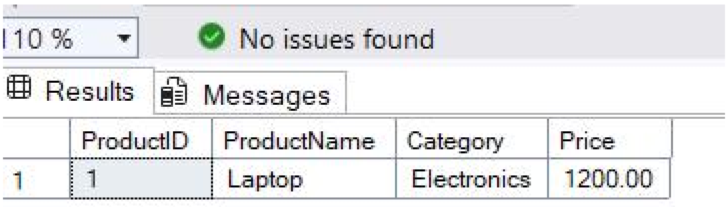
```
-- Database Schema
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Region VARCHAR(50)
);

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'),
```

```sql
(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);
```

Tasks:

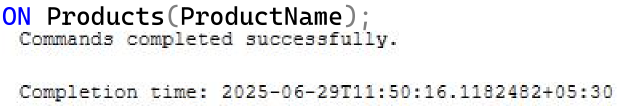
#1

```sql
-- Step 1: Query to fetch product details before index creation
SELECT * FROM Products WHERE ProductName = 'Laptop';
```



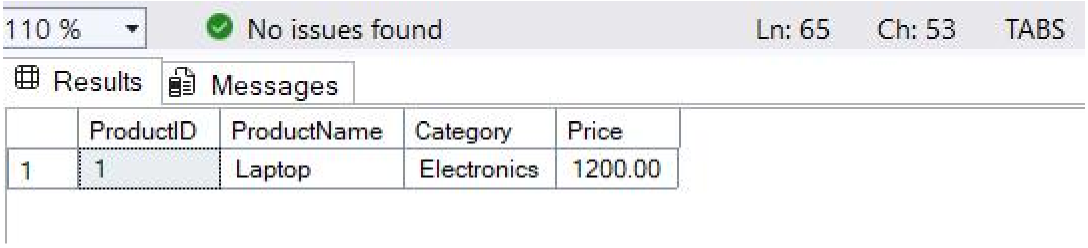
	ProductID	ProductName	Category	Price
1	1	Laptop	Electronics	1200.00

```sql
-- Step 2: Create a non-clustered index on ProductName
CREATE NONCLUSTERED INDEX IX_Products_ProductName
ON Products(ProductName);
```



```
Commands completed successfully.

Completion time: 2025-06-29T11:50:16.1182482+05:30
```

```sql
-- Step 3: Query to fetch product details after index creation
SELECT * FROM Products WHERE ProductName = 'Laptop';
```

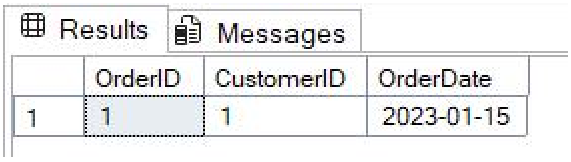


	ProductID	ProductName	Category	Price
1	1	Laptop	Electronics	1200.00

#2: Creating a Clustered Index on OrderDate

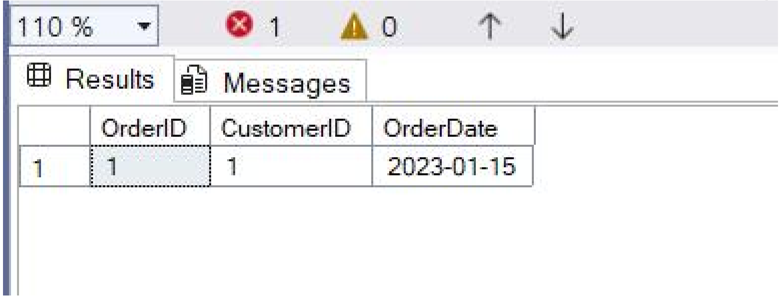
Goal: Compare performance before/after creating a clustered index on OrderDate

```
-- Step 1: Query to fetch orders before index creation
SELECT * FROM Orders WHERE OrderDate = '2023-01-15';
```



	OrderID	CustomerID	OrderDate
1	1	1	2023-01-15

```
-- Step 2: Create a clustered index on OrderDate
CREATE CLUSTERED INDEX IX_Orders_OrderDate
ON Orders(OrderDate);

-- Step 3: Query to fetch orders after index creation
SELECT * FROM Orders WHERE OrderDate = '2023-01-15';
```

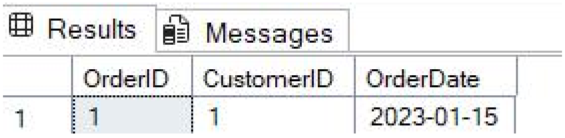


	OrderID	CustomerID	OrderDate
1	1	1	2023-01-15

#3

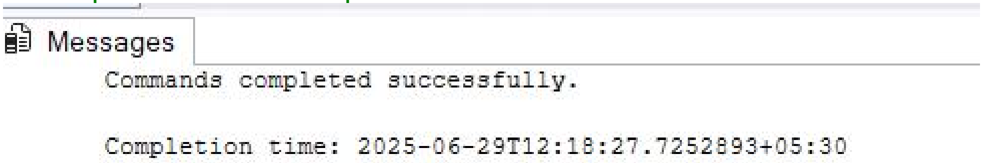
Creating a Composite Index

```
-- 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';
```



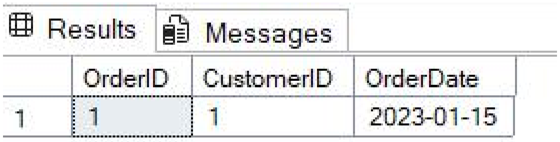
	OrderID	CustomerID	OrderDate
1	1	1	2023-01-15

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



```
Commands completed successfully.

Completion time: 2025-06-29T12:18:27.7252893+05:30
```

```
-- Step 3: Query to fetch orders after index creation
SELECT * FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';
```



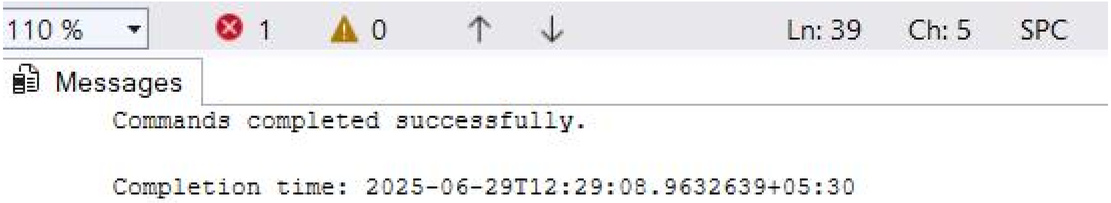
	OrderID	CustomerID	OrderDate
1	1	1	2023-01-15

---

3. Exercise-3(Employee Management System SQL Exercises):

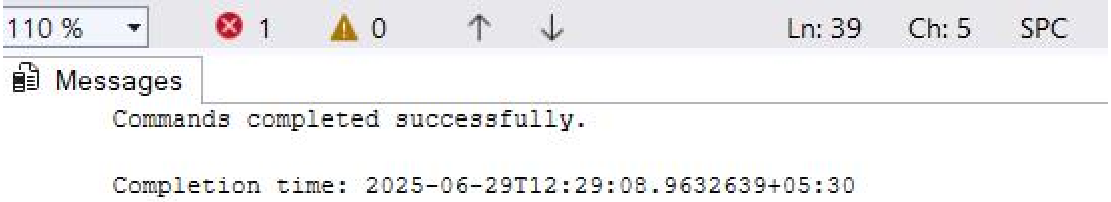
DATABASE SCHEMA with sample data FOR this and next two exercises 4 and 5:

```
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
);
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');


--Creating Stored Procedure to Retrieve Employees by Department
CREATE PROCEDURE sp_GetEmployeesByDepartment
    @DepartmentID INT
AS
BEGIN
    SELECT
        EmployeeID,
        FirstName,
        LastName,
        DepartmentID,
        Salary,
        JoinDate
    FROM Employees
    WHERE DepartmentID = @DepartmentID;
END;
```



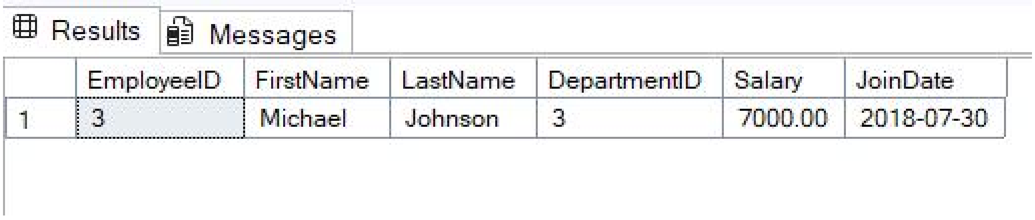
```
Commands completed successfully.

Completion time: 2025-06-29T12:29:08.9632639+05:30
```

```
--Creating Stored Procedure to Insert a New Employ

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;
```



```
Commands completed successfully.

Completion time: 2025-06-29T12:29:08.9632639+05:30
```

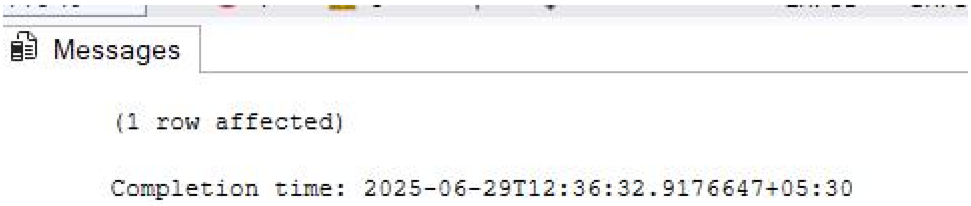
```
-- Testing the Procedures
EXEC sp_GetEmployeesByDepartment @DepartmentID = 3;
```



	EmployeeID	FirstName	LastName	DepartmentID	Salary	JoinDate
1	3	Michael	Johnson	3	7000.00	2018-07-30

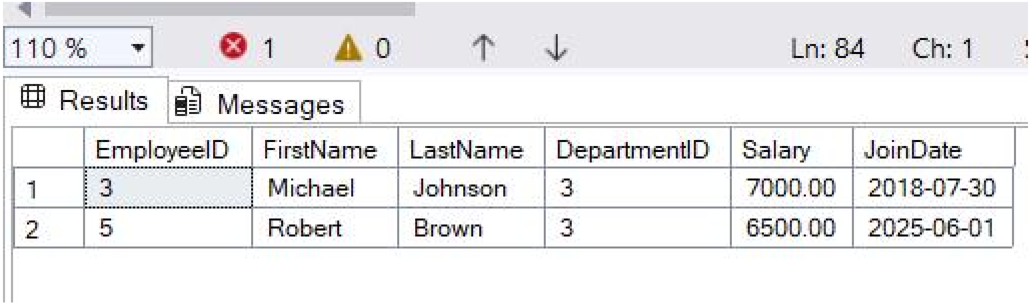
```
--Inserting a new Employee
CREATE OR ALTER PROCEDURE sp_InsertEmployee
    @EmployeeID INT,
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @DepartmentID INT,
    @Salary DECIMAL(10,2),
    @JoinDate DATE
AS
BEGIN
    INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID,
Salary, JoinDate)
    VALUES (@EmployeeID, @FirstName, @LastName, @DepartmentID, @Salary,
@JoinDate);
END;

EXEC sp_InsertEmployee
```

```
@EmployeeID = 5,
@FirstName = 'Robert',
@LastName = 'Brown',
@DepartmentID = 3,
@Salary = 6500.00,
@JoinDate = '2025-06-01';
```



```
(1 row affected)

Completion time: 2025-06-29T12:36:32.9176647+05:30
```

Testing:



	EmployeeID	FirstName	LastName	DepartmentID	Salary	JoinDate
1	3	Michael	Johnson	3	7000.00	2018-07-30
2	5	Robert	Brown	3	6500.00	2025-06-01

---

4. Exercise-4:

Execute the command. Database schema is as above for 3. exercise-3

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.

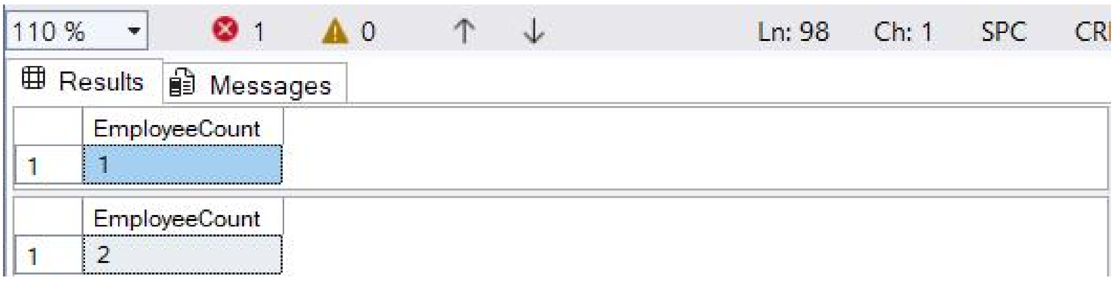
```
--MAIN TASK-2

EXEC sp_GetEmployeesByDepartment @DepartmentID = 4;
```

Exercise-5 (**Return Data from a Stored Procedure** (Database schema is as above for 3 .exercise-3))

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

```
--MAIN TASK-3(EXERCISE-5)
```

```
    CREATE PROCEDURE sp_GetEmployeeCountByDepartment
    @DepartmentID INT
AS
BEGIN
    SELECT COUNT(*) AS EmployeeCount
    FROM Employees
    WHERE DepartmentID = @DepartmentID;
END;

   EXEC sp_GetEmployeeCountByDepartment @DepartmentID = 2;
```



	EmployeeCount
1	1

	EmployeeCount
1	2

6. Exercise-6(SQL Exercise - Functions)

Employee Management System - SQL Exercises

Database Schema

The Employee Management System database schema consists of the following tables:

1. Departments

Column	Data Type	Description
DepartmentID	INT (PK)	Unique department ID
DepartmentName	VARCHAR(100)	Name of the department

2. Employees

Column	Data Type	Description
EmployeeID	INT (PK)	Unique employee ID
FirstName	VARCHAR(50)	Employee's first name
LastName	VARCHAR(50)	Employee's last name
DepartmentID	INT (FK)	Linked to Departments
Salary	DECIMAL(10,2)	Monthly salary
JoinDate	DATE	Date of joining

Sample Data

Sample data for testing:

Departments:

DepartmentID	DepartmentName
1	HR
2	IT
3	Finance

Employees:

| EmployeeID | FirstName | LastName | DepartmentID | Salary | JoinDate |

1	John	Doe	1	5000.00	2020-01-15
2	Jane	Smith	2	6000.00	2019-03-22
3	Bob	Johnson	3	5500.00	2021-07-01

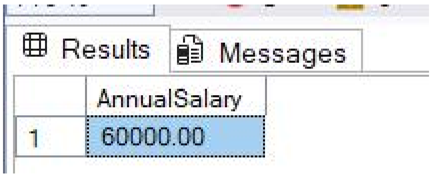
TASK: **Return Data from a Scalar Function**

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:

```sql
create database funcn_learn;
use funcn_learn;

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

CREATE TABLE Employees (
    EmployeeID INT IDENTITY(1,1) 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, 'IT'),
(3, 'Finance');

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'),
('Bob', 'Johnson', 3, 5500.00, '2021-07-01');

-- Creating Scalar Function to Calculate Annual Salary

CREATE FUNCTION fn_CalculateAnnualSalary (@EmployeeID INT)
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @AnnualSalary DECIMAL(10,2);

    SELECT @AnnualSalary = Salary * 12
    FROM Employees
    WHERE EmployeeID = @EmployeeID;

    RETURN @AnnualSalary;
END;


--Testing the function:
SELECT dbo.fn_CalculateAnnualSalary(1) AS AnnualSalary;
```



	AnnualSalary
1	60000.00