

# ADVANCE SQL

## Exercise1: RANKING AND WINDOW FUNCTIONS

Goal: Use ROW\_NUMBER(), RANK(), DENSE\_RANK(), OVER(), and PARTITION BY.

Scenario:

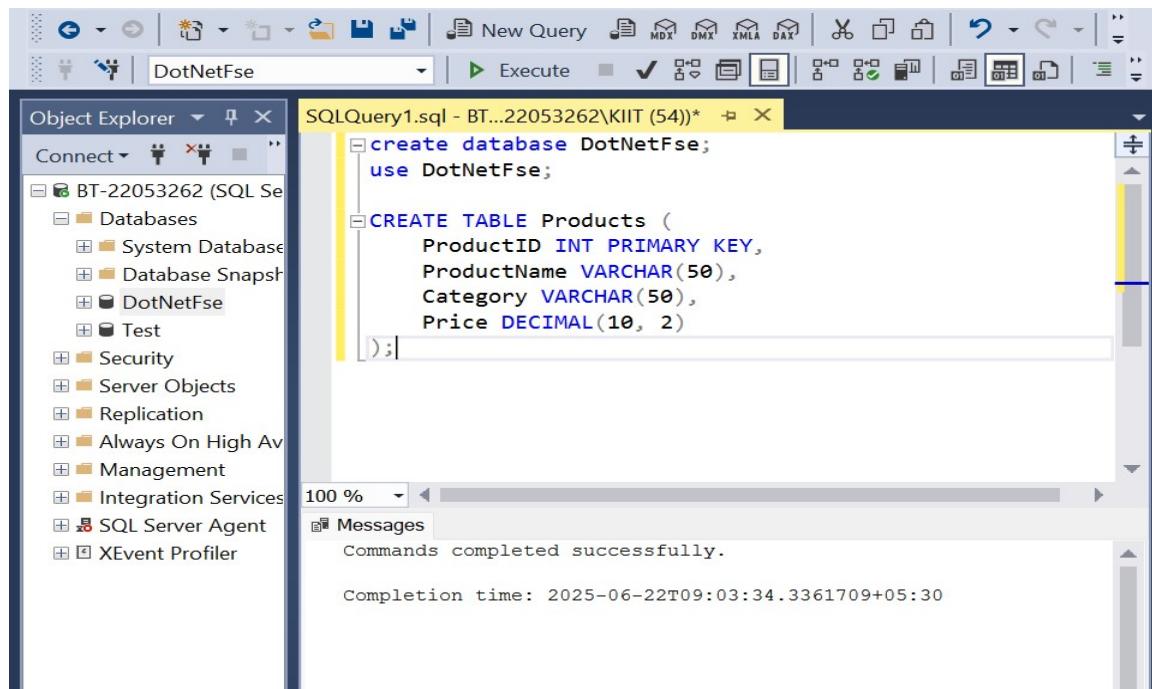
Find the top 3 most expensive products in each category using different ranking functions.

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;**

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'BT-22053262'. The main window displays a query results grid for the 'Products' table. The results show 12 rows of product information, including ProductID, ProductName, Category, and Price.

ProductID	ProductName	Category	Price
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

--Using ROW\_NUMBER()

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'BT-22053262'. The main window displays three queries demonstrating ranking functions. The first query uses ROW\_NUMBER() to assign row numbers partitioned by category. The second query uses RANK() to assign ranks partitioned by category. The third query uses DENSE\_RANK() to assign dense ranks partitioned by category. The results show the same 12 products from the 'Products' table, with the rank assigned by each function.

Category	ProductName	Price	RowNum
Electronics	MacBook Pro	2500.00	1
Electronics	HP Laptop	1200.00	2
Electronics	iPhone 15	1200.00	3
Electronics	Samsung Galaxy	1150.00	4
Electronics	Dell Monitor	400.00	5
Footwear	Heels	170.00	1
Footwear	Formal Shoes	150.00	2
Footwear	Sneakers	120.00	3
Footwear	Running Shoes	120.00	4
Footwear	Sandals	80.00	5
Home Furniture	Luxury Sofa	1200.00	1
Home Furniture	Dining Table Set	900.00	2

--Using RANK()

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

Object Explorer

SQLQuery1.sql - BT-22053262.DotNetFse (BT-22053262\KIIT (54)) - Microsoft SQL Server Management Studio

```

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;

```

	Category	ProductName	Price	RankNum
1	Electronics	MacBook Pro	2500.00	1
2	Electronics	HP Laptop	1200.00	2
3	Electronics	iPhone 15	1200.00	2
4	Electronics	Samsung Galaxy	1150.00	4
5	Electronics	Dell Monitor	400.00	5
6	Footwear	Heels	170.00	1
7	Footwear	Formal Shoes	150.00	2
8	Footwear	Sneakers	120.00	3
9	Footwear	Running Shoes	120.00	3
10	Footwear	Sandals	80.00	5
11	Home Furniture	Luxury Sofa	1200.00	1
12	Home Furniture	Dining Table Set	900.00	2

Query executed successfully.

--Using DENSE\_RANK()

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

Object Explorer

SQLQuery1.sql - BT-22053262.DotNetFse (BT-22053262\KIIT (54)) - Microsoft SQL Server Management Studio

```

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;

```

	Category	ProductName	Price	DenseRank
1	Electronics	MacBook Pro	2500.00	1
2	Electronics	HP Laptop	1200.00	2
3	Electronics	iPhone 15	1200.00	2
4	Electronics	Samsung Galaxy	1150.00	3
5	Electronics	Dell Monitor	400.00	4
6	Footwear	Heels	170.00	1
7	Footwear	Formal Shoes	150.00	2
8	Footwear	Sneakers	120.00	3
9	Footwear	Running Shoes	120.00	3
10	Footwear	Sandals	80.00	4
11	Home Furniture	Luxury Sofa	1200.00	1
12	Home Furniture	Dining Table Set	900.00	2

Query executed successfully.

--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;
```

--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;
```

ProductID	ProductName	Category	Price	RowNum
1	3	MacBook Pro	2500.00	1
2	4	HP Laptop	1200.00	2
3	1	iPhone 15	1200.00	3
4	9	Heels	170.00	1
5	7	Formal Shoes	150.00	2
6	8	Sneakers	120.00	3
7	11	Luxury Sofa	1200.00	1
8	12	Dining Table Set	900.00	2

Query executed successfully.

--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;
```

--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;
```

--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;
```

ProductID	ProductName	Category	Price	RowNum
1	3	MacBook Pro	2500.00	1
2	4	HP Laptop	1200.00	2
3	1	iPhone 15	1200.00	2
4	9	Heels	170.00	1
5	7	Formal Shoes	150.00	2
6	8	Sneakers	120.00	3
7	6	Running Shoes	120.00	3
8	11	Luxury Sofa	1200.00	1
9	12	Dining Table Set	900.00	2

Query executed successfully.

--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;
```

```

SQLQuery1.sql - BT-22053262.DotNetFse (BT-22053262\KIIT (54)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
Connect - DotNetFse | Execute | New Query | Quick Launch (Ctrl+Q) | 
Object Explorer | SQL Server Object Explorer | 
Database | System Databases | Database Snapshots | DotNetFse | Test | Security | Server Objects | Replication | Always On High Availability | Management | Integration Services Catalogs | SQL Server Agent | XEvent Profiler |
SQLQuery1.sql - BT-22053262\KIIT (54)* | 
--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;

```

	ProductID	ProductName	Category	Price	RowNum
1	3	MacBook Pro	Electronics	2500.00	1
2	4	HP Laptop	Electronics	1200.00	2
3	1	iPhone 15	Electronics	1200.00	2
4	2	Samsung Galaxy	Electronics	1150.00	3
5	9	Heels	Footwear	170.00	1
6	7	Formal Shoes	Footwear	150.00	2
7	8	Sneakers	Footwear	120.00	3
8	6	Running Shoes	Footwear	120.00	3
9	11	Luxury Sofa	Home Furniture	1200.00	1
10	12	Dining Table Set	Home Furniture	900.00	2

Query executed successfully.

BT-22053262 (16.0 RTM) | BT-22053262\KIIT (54) | DotNetFse | 00:00:00 | 10 rows

## 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)
);

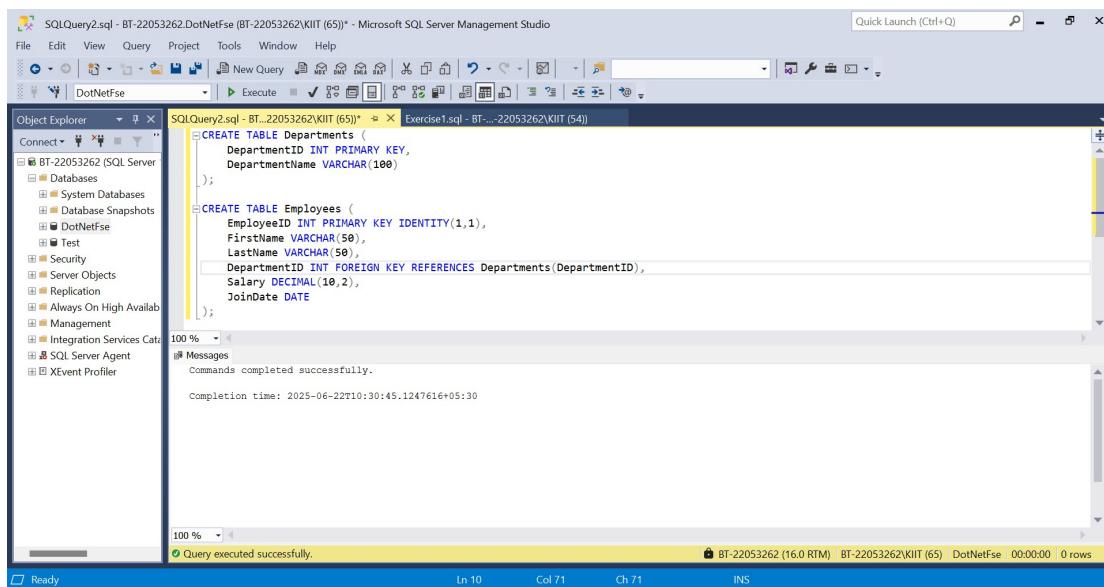
```

```

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;

```

The screenshot shows the Object Explorer on the left with 'BT-22053262 (SQL Server)' selected. In the center, a query window titled 'SQLQuery2.sql - BT...22053262\KIIT (65)\*' contains the following SQL code:

```

CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName NVARCHAR(50)
);

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

SELECT * FROM Departments;

```

The results pane shows the following data:

DepartmentID	DepartmentName
1	HR
2	Finance
3	IT
4	Marketing

At the bottom, a message bar says 'Query executed successfully.' and the status bar shows 'BT-22053262 (16.0 RTM) BT-22053262\KIIT (65) DotNetFse 00:00:00 | 4 rows'.

```

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;

```

The screenshot shows the Object Explorer on the left with 'BT-22053262 (SQL Server)' selected. In the center, a query window titled 'SQLQuery2.sql - BT...22053262\KIIT (65)\*' contains the following SQL code:

```

CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName NVARCHAR(50)
);

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;

```

The results pane shows the following data:

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	Michael	Johnson	3	7000.00	2018-07-30
4	Emily	Davis	4	5500.00	2021-11-05

At the bottom, a message bar says 'Query executed successfully.' and the status bar shows 'BT-22053262 (16.0 RTM) BT-22053262\KIIT (65) DotNetFse 00:00:00 | 4 rows'.

--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;

```

The screenshot shows the Object Explorer on the left with the database 'BT-22053262' selected. In the center query editor, a new stored procedure is being created:

```

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

```

The status bar at the bottom indicates the command was executed successfully.

--Test sp\_GetEmployeesByDepartment

```
EXEC sp_GetEmployeesByDepartment @DepartmentID = 2;
```

The screenshot shows the Object Explorer on the left with the database 'BT-22053262' selected. In the center query editor, the stored procedure is tested:

```

--Test sp_GetEmployeesByDepartment
EXEC sp_GetEmployeesByDepartment @DepartmentID = 2;

```

The results pane shows the output of the query:

	EmployeeID	FirstName	LastName	DepartmentName	Salary	JoinDate
1	2	Jane	Smith	Finance	6000.00	2019-03-22

The status bar at the bottom indicates the command was executed successfully.

--Create sp\_InsertEmployee Procedure

```
GO
```

```

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;

```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, a database named BT-22053262 is selected. In the center pane, a query window titled 'SQLQuery2.sql - BT-22053262\KITT (65)\*' contains the T-SQL code for creating the stored procedure. The bottom pane displays the execution results, showing a message that the command completed successfully and the completion time.

```

--Create sp_InsertEmployee Procedure
GO
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;

```

100 % ↴ 100 % ↴

Messages

Commands completed successfully.

Completion time: 2025-06-22T10:41:22.5424622+05:30

BT-22053262 (16.0 RTM) | BT-22053262\KITT (65) | DotNetFse | 00:00:00 | 0 rows

--Test sp\_InsertEmployee

```

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

```

--Test sp\_InsertEmployee

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

(1 row affected)

Completion time: 2025-06-22 10:42:09.7226752+05:30

Query executed successfully.

--Verify

**SELECT \* FROM Employees;**

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

--Test sp\_InsertEmployee

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

--Verify
```

**SELECT \* FROM 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	Michael	Johnson	3	7000.00	2018-07-30
4	Emily	Davis	4	5500.00	2021-11-05
5	Amit	Kumar	3	6500.00	2022-08-01

Query executed successfully.