**Advanced concepts**

**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.**

**Table Categories:**

CREATE TABLE Categories (

CategoryID INT PRIMARY KEY,

Name VARCHAR(100)

);

**Table Products:**

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

Name VARCHAR(100),

Price DECIMAL(10, 2),

CategoryID INT,

FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)

);

**Steps:**

1. **Use ROW\_NUMBER() to assign a unique rank within each category.**

WITH RankedProducts AS (

SELECT

p.ProductID,

p.Name AS ProductName,

p.Price,

c.Name AS Category,

ROW\_NUMBER() OVER (PARTITION BY p.CategoryID ORDER BY p.Price DESC) AS RowNum

FROM Products p

JOIN Categories c ON p.CategoryID = c.CategoryID

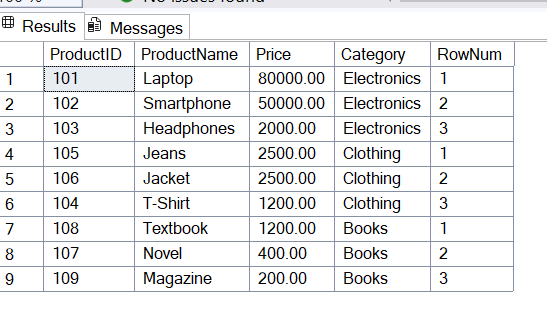
)

SELECT \*

FROM RankedProducts

WHERE RowNum <= 3;

Output:



1. **Use RANK() and DENSE\_RANK() to compare how ties are handled.**

Using RANK():

WITH RankedProducts AS (

SELECT

p.ProductID,

p.Name AS ProductName,

p.Price,

c.Name AS Category,

RANK() OVER (PARTITION BY p.CategoryID ORDER BY p.Price DESC) AS RankPos

FROM Products p

JOIN Categories c ON p.CategoryID = c.CategoryID

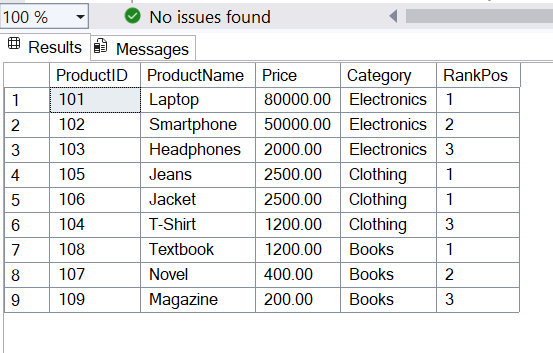
)

SELECT \*

FROM RankedProducts

WHERE RankPos <= 3;

Output:



Using DENSE\_RANK():

WITH RankedProducts AS (

SELECT

p.ProductID,

p.Name AS ProductName,

p.Price,

c.Name AS Category,

DENSE\_RANK() OVER (PARTITION BY p.CategoryID ORDER BY p.Price DESC) AS DenseRankPos

FROM Products p

JOIN Categories c ON p.CategoryID = c.CategoryID

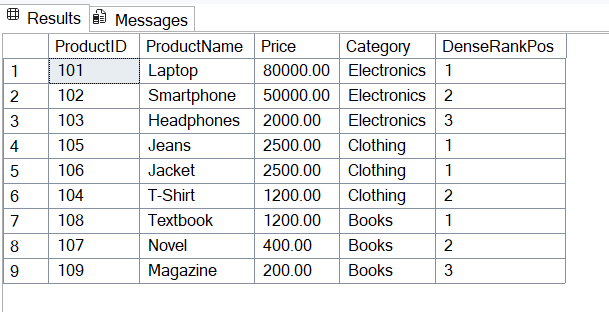
)

SELECT \*

FROM RankedProducts

WHERE DenseRankPos <= 3;

Output:



1. **Use PARTITION BY Category and ORDER BY Price DESC.**

All the above queries use Partition by Category and Order by Price DESC.For example

WITH RankedProducts AS (

SELECT

p.ProductID,

p.Name AS ProductName,

p.Price,

c.Name AS Category,

DENSE\_RANK() OVER (PARTITION BY p.CategoryID ORDER BY p.Price DESC) AS DenseRankPos

FROM Products p

JOIN Categories c ON p.CategoryID = c.CategoryID

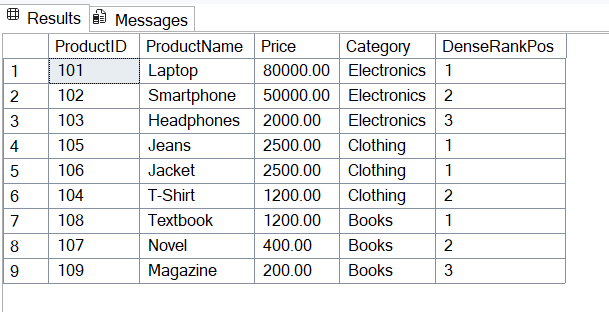
)

SELECT \*

FROM RankedProducts

WHERE DenseRankPos <= 3;

Output:



**Stored Procedures**

**Exercise 1:Create a Stored Procedure**

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

**2. Write the SQL query to select employee details based on the DepartmentID.**

Query

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

-- Step 2: SQL query

SELECT

EmployeeID,

FirstName,

LastName,

DepartmentID,

Salary,

JoinDate

FROM

Employees

WHERE

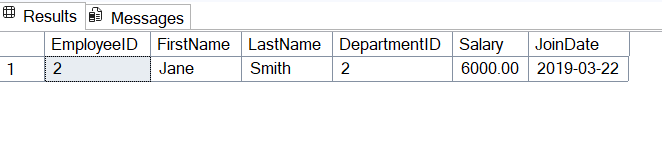
DepartmentID = @DepartmentID;

END;

GO

EXEC sp\_GetEmployeesByDepartment 2;

Output:



**3.Create a stored procedure named `sp\_InsertEmployee` with the following code**

Query

CREATE PROCEDURE sp\_InsertEmployees

@EmployeeID INT,

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

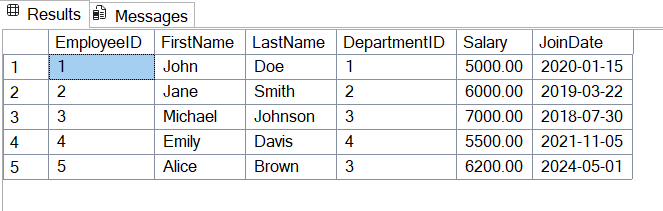
INSERT INTO Employees (EmployeeID, FirstName,LastName, DepartmentID, Salary, JoinDate)

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

END

EXEC sp\_InsertEmployees 5,'Alice', 'Brown', 3, 6200.00, '2024-05-01';

Updated table(Output):



**Exercise 5: Return Data from stored procedure**

**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**

Query:

CREATE PROCEDURE sp\_CountEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = @DepartmentID;

END

EXEC sp\_CountEmployeesByDepartment @DepartmentID = 3;

Output

