**Advanced SQL:-**

**Ranking and Window Function:-**

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:-**

create a new test database:

CREATE DATABASE RetailStore;

GO

USE RetailStore;

GO

Create Categories table:

CREATE TABLE Categories (

CategoryID INT PRIMARY KEY,

CategoryName VARCHAR(100)

);

Create Products table:

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

CategoryID INT,

Price DECIMAL(10, 2),

FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)

);

Insert into Categories:

INSERT INTO Categories (CategoryID, CategoryName) VALUES

(1, 'Electronics'),

(2, 'Clothing'),

(3, 'Books');

Insert into Products:

INSERT INTO Products (ProductID, ProductName, CategoryID, Price) VALUES

(1, 'Laptop', 1, 999.99),

(2, 'Smartphone', 1, 699.99),

(3, 'Tablet', 1, 399.99),

(4, 'Smartwatch', 1, 199.99),

(5, 'T-Shirt', 2, 29.99),

(6, 'Jeans', 2, 49.99),

(7, 'Jacket', 2, 89.99),

(8, 'Hat', 2, 19.99),

(9, 'Novel', 3, 14.99),

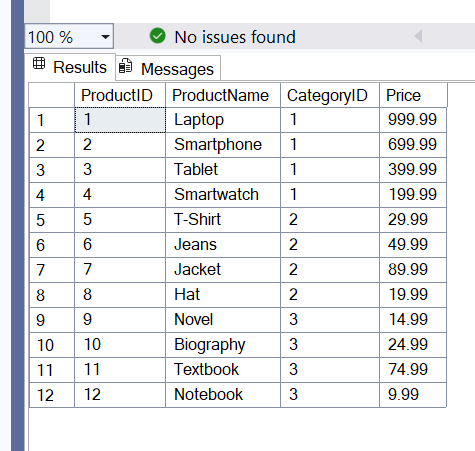
(10, 'Biography', 3, 24.99),

(11, 'Textbook', 3, 74.99),

(12, 'Notebook', 3, 9.99);

Run a quick check:

SELECT \* FROM Products;



Apply ranking functions:

WITH RankedProducts AS (

SELECT

p.ProductID,

p.ProductName,

c.CategoryName,

p.Price,

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

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

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

FROM Products p

JOIN Categories c ON p.CategoryID = c.CategoryID

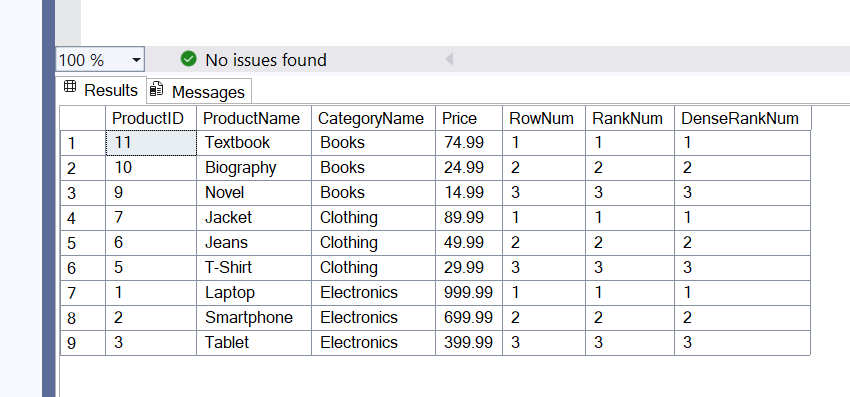
)

SELECT \*

FROM RankedProducts

WHERE RowNum <= 3

ORDER BY CategoryName, Price DESC;



**Exercise-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;

**Solution:-**

These are the parameters that we will pass to the stored procedure:

@FirstName – First name of the employee

@LastName – Last name of the employee

@DepartmentID – ID of the department the employee belongs to

@Salary – Salary of the employee

@JoinDate – Date the employee joined

These parameters allow dynamic input every time the procedure is called.

Create your database:

CREATE DATABASE EmployeeDB;

GO

USE EmployeeDB;

GO

Create the Departments table:

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

Insert sample departments:

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

Create the Employees table:

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

);

Create the stored procedure sp\_InsertEmployee:

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;

Test the stored procedure:

EXEC sp\_InsertEmployee

@FirstName = 'Amit',

@LastName = 'Sharma',

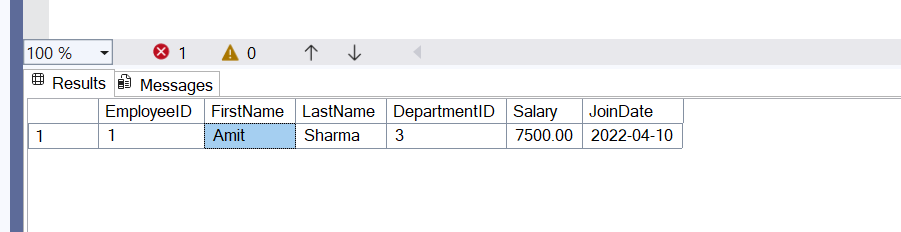
@DepartmentID = 3,

@Salary = 7500.00,

@JoinDate = '2022-04-10';

Check if employee was added:

SELECT \* FROM Employees;



SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

d.DepartmentName,

e.Salary,

e.JoinDate

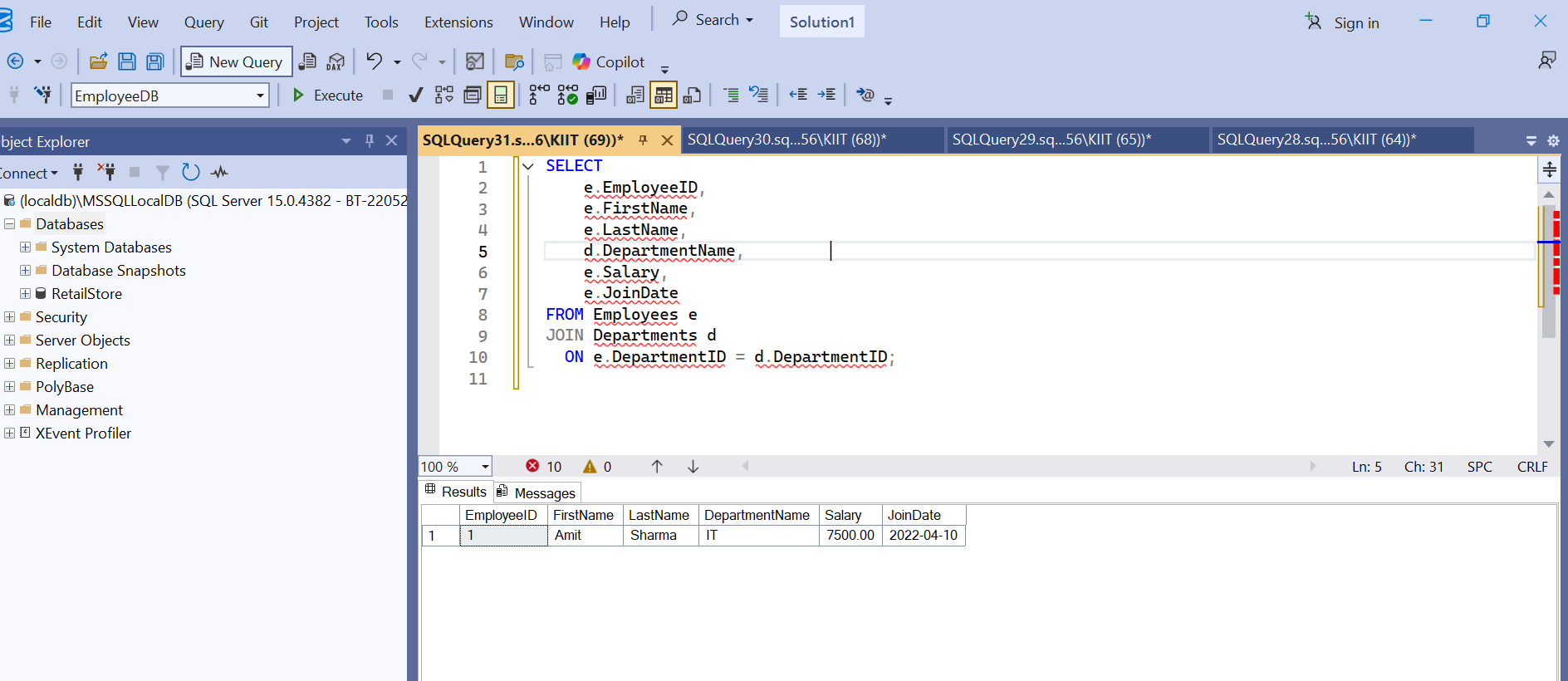
FROM Employees e

JOIN Departments d

ON e.DepartmentID = d.DepartmentID;

· e.DepartmentID is a foreign key (just a number)

· JOIN with Departments gives the readable name like 'HR' or 'IT'



**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

**Solution:**

The procedure takes a **DepartmentID** as input.

It runs a SELECT COUNT(\*) query to count how many employees belong to that department.

This allows you to **dynamically get totals** for different departments without writing the query each time.

This is useful in reporting or dashboard systems where you want to view staff count department-wise using a single procedure.

The stored procedure sp\_CountEmployeesByDepartment is designed to return the number of employees in a given department. It accepts a DepartmentID as input and uses a SELECT COUNT(\*) query with a WHERE clause to filter employees. This allows efficient reuse and dynamic querying for different departments.

Database:

USE EmployeeDB;

GO

Create the stored procedure:

CREATE PROCEDURE sp\_CountEmployeesByDepartment

@DeptID INT

AS

BEGIN

SELECT

COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DeptID;

END;

Run the procedure to test it:

EXEC sp\_CountEmployeesByDepartment @DeptID = 3;

