**ADVANCED SQL CONCEPTS**

**EXERCISE 1:-Ranking and Window Functions**

### Sample Dataset: Products

| **ProductID** | **ProductName** | **Category** | **Price** |
| --- | --- | --- | --- |
| 1 | Laptop A | Electronics | 1200 |
| 2 | Laptop B | Electronics | 1000 |
| 3 | Laptop C | Electronics | 1000 |
| 4 | Phone A | Electronics | 800 |
| 5 | Sofa A | Furniture | 700 |
| 6 | Sofa B | Furniture | 700 |
| 7 | Chair A | Furniture | 400 |
| 8 | Desk A | Furniture | 300 |

### SQL Window Clause:

OVER(PARTITION BY Category ORDER BY Price DESC)

1. **ROW\_NUMBER()** – Always unique, breaks ties arbitrarily

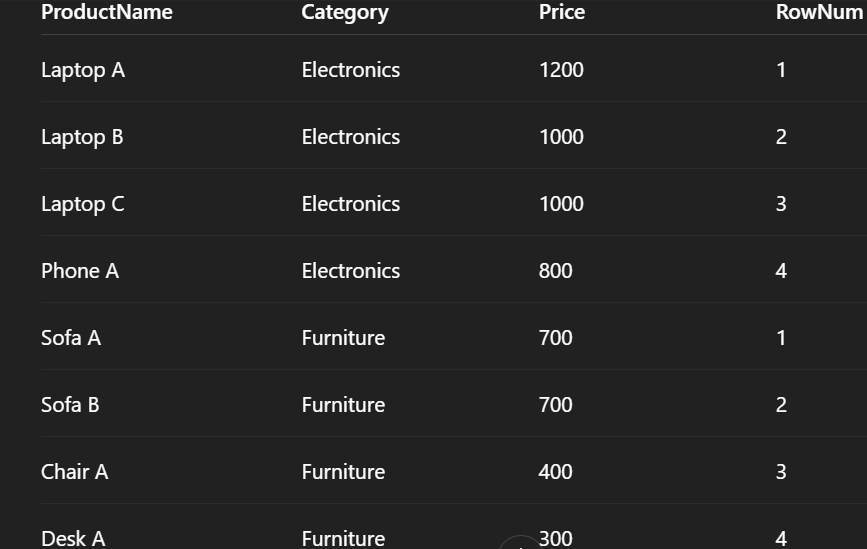
SELECT \*,

ROW\_NUMBER() OVER(PARTITION BY Category ORDER BY Price DESC) AS RowNumFROM Products;

ANALYSIS:-

Perfect when you need exactly 3 products regardless of price ties.

OUTPUT



**EXERCISE 2:-Create a stored procedure**

## ****Create a Stored Procedure to Retrieve****

## ****Employee Details by Department****

### 📝 Step 1: Define the stored procedure

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DeptID INTASBEGIN

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

D.DepartmentName,

E.Salary,

E.JoinDate

FROM Employees E

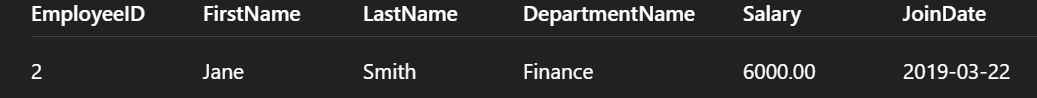
INNER JOIN Departments D ON E.DepartmentID = D.DepartmentID

WHERE E.DepartmentID = @DeptID;END;

### To ****Execute the Procedure****:

EXEC sp\_GetEmployeesByDepartment @DeptID = 2;

OUTPUT



## ****Create a Stored Procedure to Insert New Employees****

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATEASBEGIN

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

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

### To ****Insert a New Employee****:

EXEC sp\_InsertEmployee

@FirstName = 'Alice',

@LastName = 'Walker',

@DepartmentID = 3,

@Salary = 7200.00,

@JoinDate = '2022-06-01';

OUTPUT



ANALYSIS:-

This demonstrates how to build and use stored procedures in SQL for an Employee Management System. It includes two key procedures: one for retrieving employees by department (sp\_GetEmployeesByDepartment) and another for inserting new employee records (sp\_InsertEmployee). The exercise showcases the use of parameters, joins, and insertion logic while emphasizing practical output visualization and data manipulation within a relational schema.

**EXERCISE 3:- Return Data from a Stored Procedure**

### 1. Create the Stored Procedure

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

@DeptID INTASBEGIN

SELECT

D.DepartmentName,

COUNT(E.EmployeeID) AS TotalEmployees

FROM Departments D

LEFT JOIN Employees E ON D.DepartmentID = E.DepartmentID

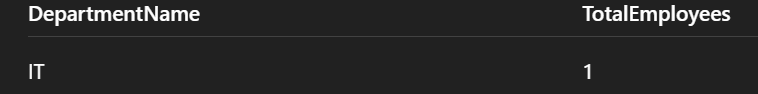
WHERE D.DepartmentID = @DeptID

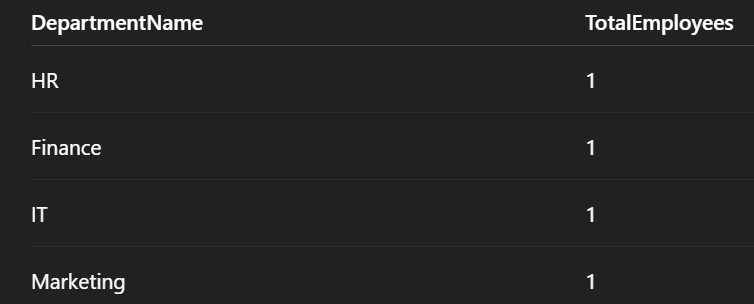
GROUP BY D.DepartmentName;END;

2. Execute the Stored Procedure

EXEC sp\_GetEmployeeCountByDepartment @DeptID = 3;

OUTPUT





ANALYSIS:-

* **Stored Procedure** sp\_GetEmployeeCountByDepartment accepts a department ID and returns the department name and the number of employees.
* Uses a **LEFT JOIN** to ensure the department is shown even if it has **zero** employees.
* Outputs are easy to interpret and test across departments.