**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.

**CODE:**

CREATE DATABASE ProductsDB;

GO

USE ProductsDB;

GO

CREATE TABLE products (

product\_id INT IDENTITY(1,1) PRIMARY KEY,

product\_name NVARCHAR(100) NOT NULL,

category NVARCHAR(50) NOT NULL,

price DECIMAL(10,2) NOT NULL,

brand NVARCHAR(50),

launch\_date DATE

);

GO

INSERT INTO products (product\_name, category, price, brand, launch\_date) VALUES

-- Electronics

('Samsung Galaxy S24', 'Electronics', 79999.00, 'Samsung', '2024-01-15'),

('iPhone 15', 'Electronics', 79900.00, 'Apple', '2023-09-20'),

('OnePlus 12', 'Electronics', 64999.00, 'OnePlus', '2024-01-10'),

('Xiaomi 14', 'Electronics', 54999.00, 'Xiaomi', '2024-02-25'),

('Realme GT 6', 'Electronics', 42999.00, 'Realme', '2024-03-15'),

('Vivo V30 Pro', 'Electronics', 41999.00, 'Vivo', '2024-02-28'),

-- Clothing

('Fabindia Kurta Set', 'Clothing', 2499.00, 'Fabindia', '2024-01-01'),

('Allen Solly Shirt', 'Clothing', 1899.00, 'Allen Solly', '2024-01-15'),

('Levis Jeans', 'Clothing', 3499.00, 'Levis', '2024-02-01'),

('Raymond Suit', 'Clothing', 8999.00, 'Raymond', '2024-01-20'),

('Peter England Blazer', 'Clothing', 4999.00, 'Peter England', '2024-02-10'),

('Van Heusen Formal Shirt', 'Clothing', 1899.00, 'Van Heusen', '2024-01-25'),

-- Food & Beverages

('Amul Dark Chocolate', 'Food & Beverages', 299.00, 'Amul', '2024-01-01'),

('Tata Tea Premium', 'Food & Beverages', 450.00, 'Tata', '2024-01-05'),

('Britannia Good Day Cookies', 'Food & Beverages', 65.00, 'Britannia', '2024-01-10'),

('Haldirams Bhujia', 'Food & Beverages', 180.00, 'Haldirams', '2024-01-15'),

('MTR Ready Mix', 'Food & Beverages', 120.00, 'MTR', '2024-01-20'),

('Patanjali Honey', 'Food & Beverages', 450.00, 'Patanjali', '2024-01-25'),

-- Home & Kitchen

('Prestige Pressure Cooker', 'Home & Kitchen', 2299.00, 'Prestige', '2024-01-01'),

('Bajaj Mixer Grinder', 'Home & Kitchen', 4599.00, 'Bajaj', '2024-01-10'),

('Godrej Refrigerator', 'Home & Kitchen', 28999.00, 'Godrej', '2024-01-15'),

('LG Washing Machine', 'Home & Kitchen', 35999.00, 'LG', '2024-01-20'),

('Philips Air Fryer', 'Home & Kitchen', 8999.00, 'Philips', '2024-01-25'),

('Havells Ceiling Fan', 'Home & Kitchen', 2299.00, 'Havells', '2024-02-01');

GO

-- 1. ROW\_NUMBER() - Top 3 products in each category (unique ranking)

WITH RankedProducts AS (

SELECT

product\_name,

category,

price,

brand,

ROW\_NUMBER() OVER (PARTITION BY category ORDER BY price DESC) as row\_num

FROM products

)

SELECT

product\_name,

category,

FORMAT(price, 'N2') as price\_formatted,

brand,

row\_num

FROM RankedProducts

WHERE row\_num <= 3

ORDER BY category, row\_num;

GO

-- 2. RANK() - Top 3 ranks in each category

WITH RankedProducts AS (

SELECT

product\_name,

category,

price,

brand,

RANK() OVER (PARTITION BY category ORDER BY price DESC) as rank\_num

FROM products

)

SELECT

product\_name,

category,

FORMAT(price, 'N2') as price\_formatted,

brand,

rank\_num

FROM RankedProducts

WHERE rank\_num <= 3

ORDER BY category, rank\_num;

GO

-- 3. DENSE\_RANK() - Top 3 dense ranks in each category

WITH DenseRankedProducts AS (

SELECT

product\_name,

category,

price,

brand,

DENSE\_RANK() OVER (PARTITION BY category ORDER BY price DESC) as dense\_rank\_num

FROM products

)

SELECT

product\_name,

category,

FORMAT(price, 'N2') as price\_formatted,

brand,

dense\_rank\_num

FROM DenseRankedProducts

WHERE dense\_rank\_num <= 3

ORDER BY category, dense\_rank\_num;

GO

SELECT

product\_name,

category,

FORMAT(price, 'N2') as price\_formatted,

brand,

ROW\_NUMBER() OVER (PARTITION BY category ORDER BY price DESC) as row\_num,

RANK() OVER (PARTITION BY category ORDER BY price DESC) as rank\_num,

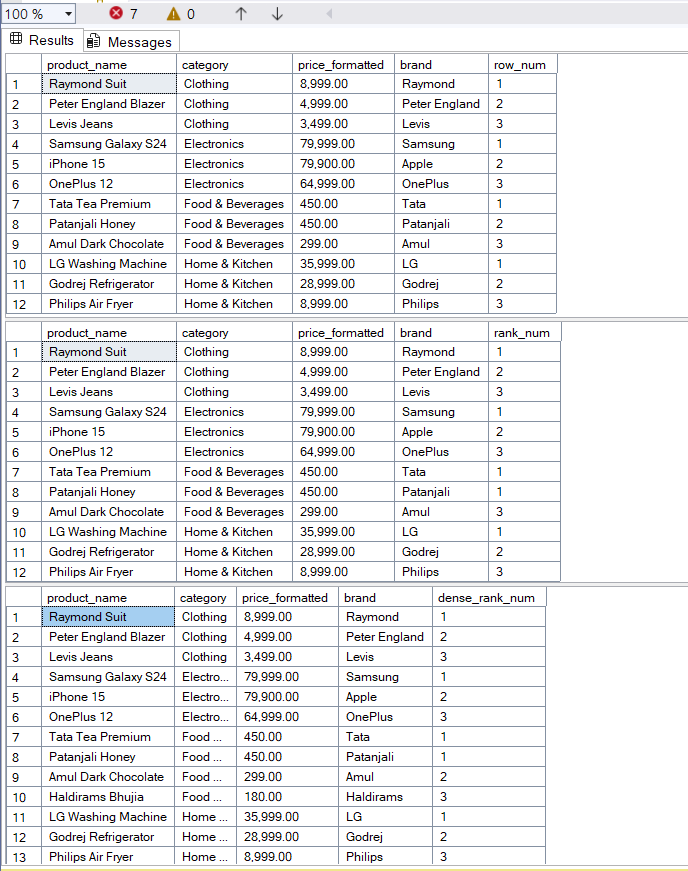
DENSE\_RANK() OVER (PARTITION BY category ORDER BY price DESC) as dense\_rank\_num

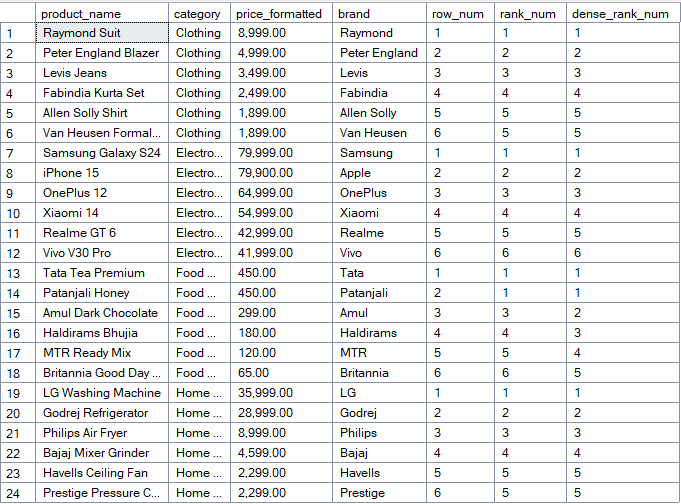
FROM products

ORDER BY category, price DESC;

GO

**OUTPUT:**





**2. SQL Exercise – Index**

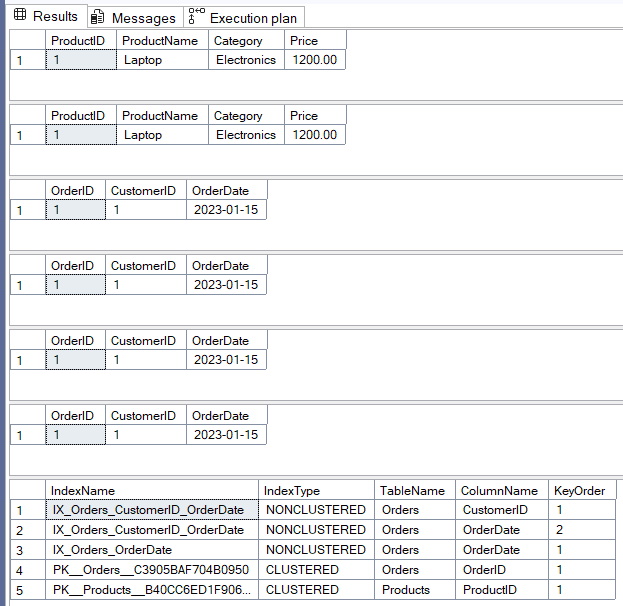
Exercise 1: Creating a Non-Clustered Index

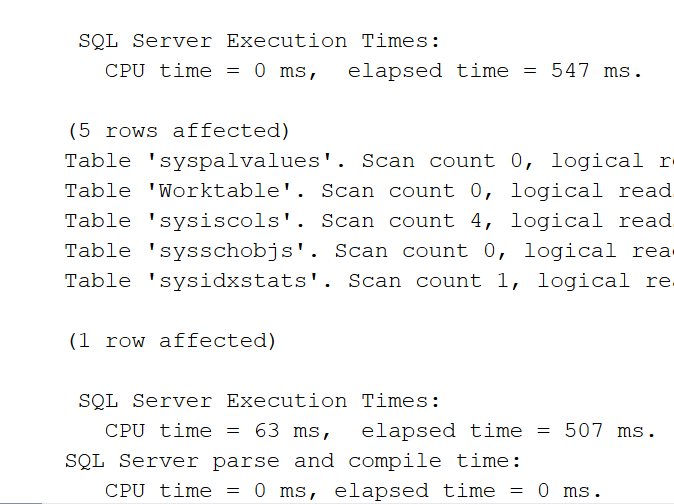
-- Goal: Create a non-clustered index on the ProductName column in the Products table and compare query execution time before and after index creation.

Exercise 2: Creating a Clustered Index -- Goal: Create a clustered index on the OrderDate column in the Orders table and compare query execution time before and after index creation.

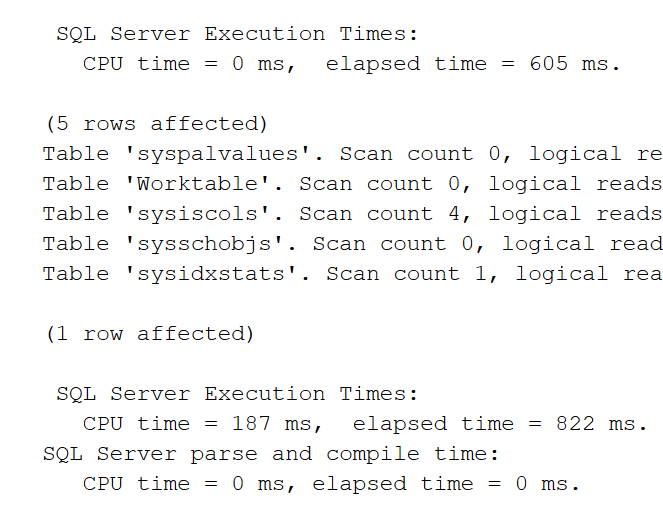
Exercise 3: Creating a Composite Index -- Goal: Create a composite index on the CustomerID and OrderDate columns in the Orders table and compare query execution time before and after index creation.

**OUTPUT:**



Before Index Creation:

After Index Creation:



**CODE:**

-- Exercise 1: Creating a Non-Clustered Index

SET STATISTICS IO ON;

SET STATISTICS TIME ON;

SELECT \* FROM Products WHERE ProductName = 'Laptop';

CREATE NONCLUSTERED INDEX IX\_Products\_ProductName

ON Products (ProductName);

SELECT \* FROM Products WHERE ProductName = 'Laptop';

DROP INDEX IX\_Products\_ProductName ON Products;

-- Exercise 2: Creating a Clustered Index

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

CREATE NONCLUSTERED INDEX IX\_Orders\_OrderDate

ON Orders (OrderDate);

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

-- Exercise 3: Creating a Composite Index

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID\_OrderDate

ON Orders (CustomerID, OrderDate);

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

SELECT

i.name AS IndexName,

i.type\_desc AS IndexType,

OBJECT\_NAME(i.object\_id) AS TableName,

COL\_NAME(ic.object\_id, ic.column\_id) AS ColumnName,

ic.key\_ordinal AS KeyOrder

FROM sys.indexes i

INNER JOIN sys.index\_columns ic ON i.object\_id = ic.object\_id AND i.index\_id = ic.index\_id

WHERE OBJECT\_NAME(i.object\_id) IN ('Products', 'Orders')

ORDER BY TableName, IndexName, KeyOrder;

**Employee Management System - SQL Exercises**

**Exercise 7: Return Data from a Scalar Function**

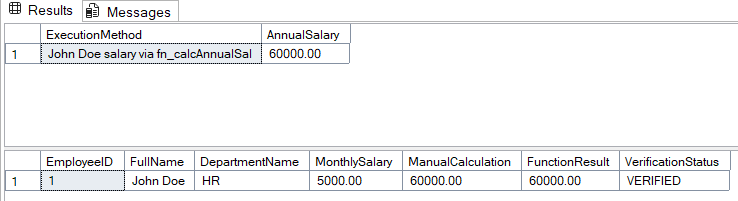
Goal: Return the annual salary for a specific employee using `fn\_CalculateAnnualSalary`.

Steps:

1. Execute the `fn\_CalculateAnnualSalary` function for an employee with `EmployeeID = 1`.

2. Verify the result.

**OUTPUT:**



**CODE:**

USE master;

GO

IF NOT EXISTS (SELECT name FROM sys.databases WHERE name = 'EmployeeManagementSystem')

BEGIN

CREATE DATABASE EmployeeManagementSystem;

END

GO

USE EmployeeManagementSystem;

GO

IF NOT EXISTS (SELECT \* FROM sysobjects WHERE name='Departments' AND xtype='U')

BEGIN

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100) NOT NULL

);

END

GO

IF NOT EXISTS (SELECT \* FROM sysobjects WHERE name='Employees' AND xtype='U')

BEGIN

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DepartmentID INT,

Salary DECIMAL(10,2) NOT NULL,

JoinDate DATE NOT NULL,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

END

GO

IF NOT EXISTS (SELECT \* FROM Departments)

BEGIN

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'IT'),

(3, 'Finance');

END

GO

IF NOT EXISTS (SELECT \* FROM Employees)

BEGIN

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

END

GO

IF OBJECT\_ID('dbo.fn\_CalculateAnnualSalary', 'FN') IS NOT NULL

DROP FUNCTION dbo.fn\_CalculateAnnualSalary;

GO

CREATE FUNCTION dbo.fn\_CalculateAnnualSalary(@EmployeeID INT)

RETURNS DECIMAL(12,2)

AS

BEGIN

DECLARE @AnnualSalary DECIMAL(12,2);

SELECT @AnnualSalary = Salary \* 12

FROM Employees

WHERE EmployeeID = @EmployeeID;

IF @AnnualSalary IS NULL

SET @AnnualSalary = 0;

RETURN @AnnualSalary;

END

GO

-- Method 1: Using SELECT statement

SELECT

'John Doe salary via fn\_calcAnnualSal' AS ExecutionMethod,

dbo.fn\_CalculateAnnualSalary(1) AS AnnualSalary;

-- Method 2: Using DECLARE and PRINT

DECLARE @Result DECIMAL(12,2);

SET @Result = dbo.fn\_CalculateAnnualSalary(1);

SELECT

e.EmployeeID,

e.FirstName + ' ' + e.LastName AS FullName,

d.DepartmentName,

e.Salary AS MonthlySalary,

e.Salary \* 12 AS ManualCalculation,

dbo.fn\_CalculateAnnualSalary(e.EmployeeID) AS FunctionResult,

CASE

WHEN (e.Salary \* 12) = dbo.fn\_CalculateAnnualSalary(e.EmployeeID)

THEN 'VERIFIED'

ELSE 'ERROR'

END AS VerificationStatus

FROM Employees e

INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID

WHERE e.EmployeeID = 1;

**SQL Exercises – Stored Procedure**

**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 PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

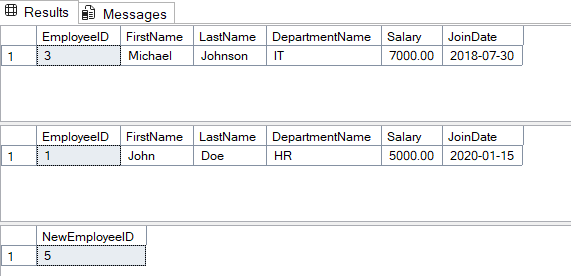
BEGIN

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

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

END;

**OUTPUT:**

****

**CODE:**

DROP TABLE IF EXISTS Employees;

DROP TABLE IF EXISTS Departments;

DROP PROCEDURE IF EXISTS sp\_GetEmployeesByDepartment;

DROP PROCEDURE IF EXISTS sp\_InsertEmployee;

DROP PROCEDURE IF EXISTS sp\_GetEmployeesByDepartment;

DROP PROCEDURE IF EXISTS sp\_InsertEmployee;

DROP TABLE IF EXISTS Employees;

DROP TABLE IF EXISTS Departments;

--

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');

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');

GO

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SET NOCOUNT ON;

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

ORDER BY e.LastName, e.FirstName;

END;

GO

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

SET NOCOUNT ON;

IF NOT EXISTS (SELECT 1 FROM Departments WHERE DepartmentID = @DepartmentID)

BEGIN

RAISERROR('Department ID %d does not exist.', 16, 1, @DepartmentID);

RETURN;

END

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

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

-- Return the new employee ID

SELECT SCOPE\_IDENTITY() AS NewEmployeeID;

END;

-- SQL query

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 3;

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 1;

EXEC sp\_InsertEmployee

@FirstName = 'Priyanshu',

@LastName = 'Ranjan',

@DepartmentID = 3,

@Salary = 42060.00,

@JoinDate = '2025-06-28';

**Exercise 4: Execute a Stored Procedure**

Goal: Execute the stored procedure to retrieve employee details for a specific department.

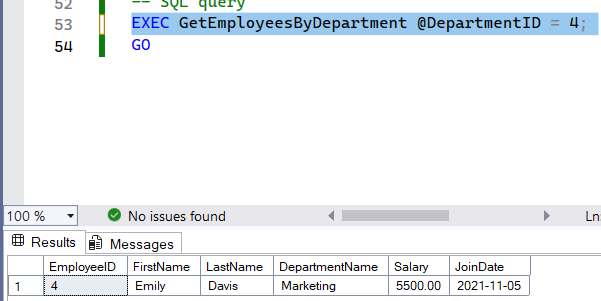
Steps:

1. Write the SQL command to execute the stored procedure with a DepartmentID

parameter.

2. Execute the command and review the results.

**OUTPUT:**

****

**CODE:**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

-- Create Employees Table

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2),

JoinDate DATE,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

-- Insert Sample Data

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');

GO

CREATE PROCEDURE GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

-- Set NOCOUNT ON to prevent extra result sets

SET NOCOUNT ON;

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

ORDER BY e.LastName, e.FirstName;

END

GO

-- SQL query

EXEC GetEmployeesByDepartment @DepartmentID = 4;

GO

**Exercise 5: Return Data from a Stored Procedure**

Goal: Create a stored procedure that returns the total number of employees in a

department.

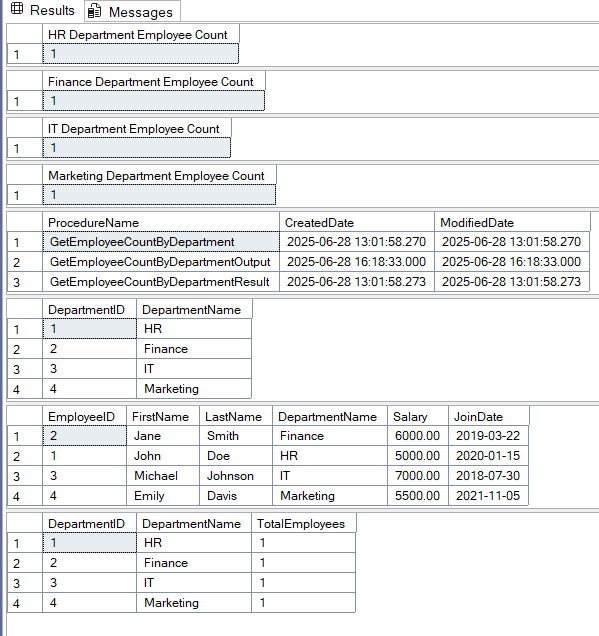
Steps:

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.

**OUTPUT:**



**CODE:**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

-- Create Employees Table

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2),

JoinDate DATE,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

-- Insert Sample Data

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');

GO

CREATE PROCEDURE GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

SET NOCOUNT ON;

SELECT

d.DepartmentID,

d.DepartmentName,

COUNT(e.EmployeeID) AS TotalEmployees

FROM Departments d

LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID

WHERE d.DepartmentID = @DepartmentID

GROUP BY d.DepartmentID, d.DepartmentName;

END

GO

EXEC GetEmployeeCountByDepartment @DepartmentID = 1;

EXEC GetEmployeeCountByDepartment @DepartmentID = 2;

EXEC GetEmployeeCountByDepartment @DepartmentID = 3;

EXEC GetEmployeeCountByDepartment @DepartmentID = 4;

EXEC GetEmployeeCountByDepartment @DepartmentID = 5;

GO

CREATE PROCEDURE GetEmployeeCountByDepartmentOutput

@DepartmentID INT,

@TotalEmployees INT OUTPUT

AS

BEGIN

SET NOCOUNT ON;

SELECT @TotalEmployees = COUNT(\*)

FROM Employees

WHERE DepartmentID = @DepartmentID;

END

GO

DECLARE @EmployeeCount INT;

EXEC GetEmployeeCountByDepartmentOutput @DepartmentID = 1, @TotalEmployees = @EmployeeCount OUTPUT;

SELECT @EmployeeCount AS 'HR Department Employee Count';

EXEC GetEmployeeCountByDepartmentOutput @DepartmentID = 2, @TotalEmployees = @EmployeeCount OUTPUT;

SELECT @EmployeeCount AS 'Finance Department Employee Count';

EXEC GetEmployeeCountByDepartmentOutput @DepartmentID = 3, @TotalEmployees = @EmployeeCount OUTPUT;

SELECT @EmployeeCount AS 'IT Department Employee Count';

EXEC GetEmployeeCountByDepartmentOutput @DepartmentID = 4, @TotalEmployees = @EmployeeCount OUTPUT;

SELECT @EmployeeCount AS 'Marketing Department Employee Count';

GO

SELECT

name AS ProcedureName,

create\_date AS CreatedDate,

modify\_date AS ModifiedDate

FROM sys.procedures

WHERE name LIKE '%GetEmployeeCount%'

ORDER BY name;

SELECT \* FROM Departments ORDER BY DepartmentID;

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

ORDER BY d.DepartmentName, e.LastName;

SELECT

d.DepartmentID,

d.DepartmentName,

COUNT(e.EmployeeID) AS TotalEmployees

FROM Departments d

LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID

GROUP BY d.DepartmentID, d.DepartmentName

ORDER BY d.DepartmentID;

GO