1. **Exercise-1**

**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

Solution:   
  
Table created and data inserted:   
  
CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

INSERT INTO Products VALUES

(1, 'Laptop', 'Electronics', 1200.00),

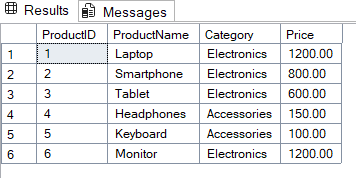
(2, 'Smartphone', 'Electronics', 800.00),

(3, 'Tablet', 'Electronics', 600.00),

(4, 'Headphones', 'Accessories', 150.00),

(5, 'Keyboard', 'Accessories', 100.00),

(6, 'Monitor', 'Electronics', 1200.00);



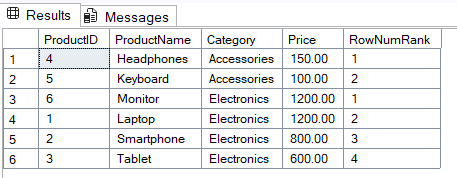
#1  
--1. ROW\_NUMBER(): Assigns unique ranks per category, ignoring ties.

SELECT

ProductID, ProductName, Category, Price,

ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNumRank

FROM Products;



Explanation:  Assigns a unique sequential rank for each product within its category, even if prices are tied. No ties in ranking

#2

--2. Use RANK() and DENSE\_RANK() to compare how ties are handled

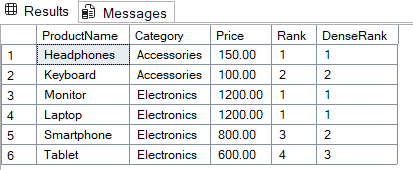
SELECT

ProductName, Category, Price,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS Rank,

DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS DenseRank

FROM Products;



Explanation: DENSE\_RANK(): Does not skip ranks; sequence remains continuous.

#3.

--3. Use PARTITION BY Category and ORDER BY Price DESC

SELECT

ProductID,

ProductName,

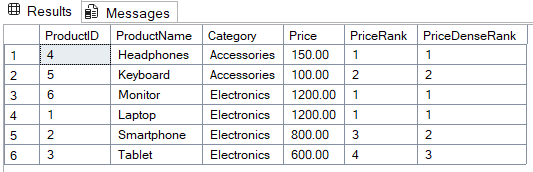
Category,

Price,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS PriceRank,

DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS PriceDenseRank

FROM Products;



Explanation: Using PARTITION BY Category and ORDER BY Price DESC ranks products by descending price within each category group.  
  
  
  
-----------------------------------------------------------------------------------------------------------------

1. **Exercise-2(SQL Exercise - Index):**

Table Creation and Data Insertion:   
  
-- Database Schema

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR(100),

    Region VARCHAR(50)

);

CREATE TABLE Products (

    ProductID INT PRIMARY KEY,

    ProductName VARCHAR(100),

    Category VARCHAR(50),

    Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

    OrderID INT PRIMARY KEY,

    CustomerID INT,

    OrderDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE OrderDetails (

    OrderDetailID INT PRIMARY KEY,

    OrderID INT,

    ProductID INT,

    Quantity INT,

    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Sample Data

INSERT INTO Customers (CustomerID, Name, Region) VALUES

(1, 'Alice', 'North'),

(2, 'Bob', 'South'),

(3, 'Charlie', 'East'),

(4, 'David', 'West');

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

(1, 'Laptop', 'Electronics', 1200.00),

(2, 'Smartphone', 'Electronics', 800.00),

(3, 'Tablet', 'Electronics', 600.00),

(4, 'Headphones', 'Accessories', 150.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(1, 1, '2023-01-15'),

(2, 2, '2023-02-20'),

(3, 3, '2023-03-25'),

(4, 4, '2023-04-30');

INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity) VALUES

(1, 1, 1, 1),

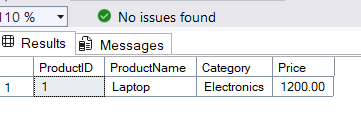
(2, 2, 2, 2),

(3, 3, 3, 1),

(4, 4, 4, 3);

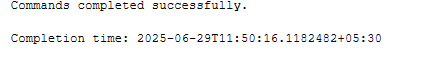
Tasks:  
  
#1  
-- Step 1: Query to fetch product details before index creation

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

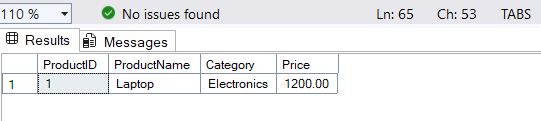


-- Step 2: Create a non-clustered index on ProductName

CREATE NONCLUSTERED INDEX IX\_Products\_ProductName

ON Products(ProductName);  


-- Step 3: Query to fetch product details after index creation

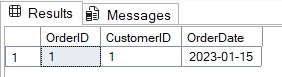
SELECT \* FROM Products WHERE ProductName = 'Laptop';  
  


#2: Creating a Clustered Index on OrderDate

Goal: Compare performance before/after creating a clustered index on OrderDate

-- Step 1: Query to fetch orders before index creation

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



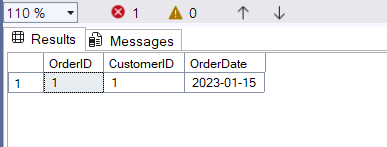
-- Step 2: Create a clustered index on OrderDate

CREATE CLUSTERED INDEX IX\_Orders\_OrderDate

ON Orders(OrderDate);

-- Step 3: Query to fetch orders after index creation

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



#3

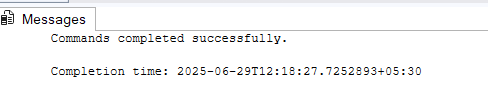
Creating a Composite Index

-- 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.

-- Step 1: Query to fetch orders before index creation

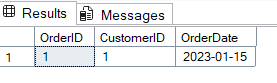
SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';  
  


-- Step 2: Create a composite index on CustomerID and OrderDate



-- Step 3: Query to fetch orders after index creation

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



**3. Exercise-3(Employee Management System SQL Exercises):**

**DATABASE SCHEMA with sample data FOR this and next two exercises 4 and 5:**CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

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

);

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');

--Creating Stored Procedure to Retrieve Employees by Department

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

SELECT

EmployeeID,

FirstName,

LastName,

DepartmentID,

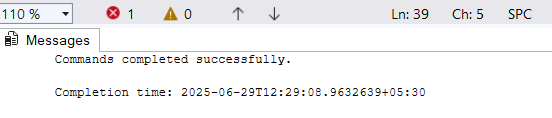
Salary,

JoinDate

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;



--Creating Stored Procedure to Insert a New Employ

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

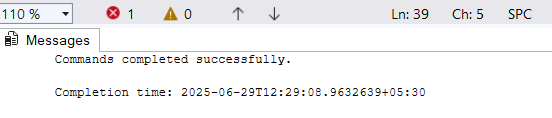
AS

BEGIN

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

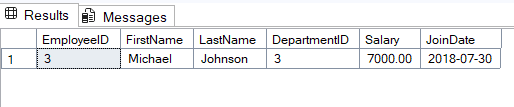
VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;



-- Testing the Procedures

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 3;



--Inserting a new Employee

CREATE OR ALTER PROCEDURE sp\_InsertEmployee

@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\_InsertEmployee

@EmployeeID = 5,

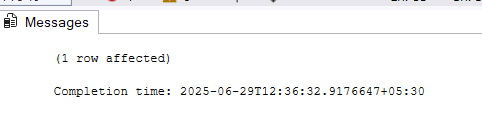
@FirstName = 'Robert',

@LastName = 'Brown',

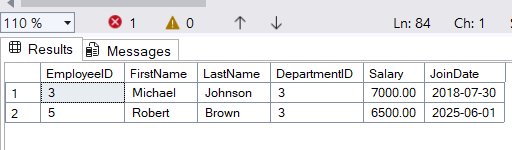
@DepartmentID = 3,

@Salary = 6500.00,

@JoinDate = '2025-06-01';



Testing:



**4. Exercise-4:**Execute the command. Database schema is as above for 3. exercise-3

**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.

--MAIN TASK-2

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 4;

**Exercise-5 (Return Data from a Stored Procedure (**Database schema is as above for 3 .exercise-3 **))**

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

--MAIN TASK-3(EXERCISE-5)

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

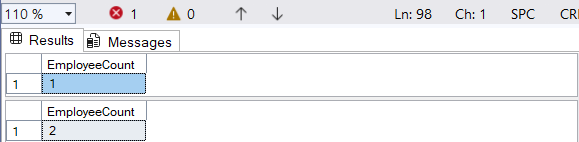
SELECT COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GetEmployeeCountByDepartment @DepartmentID = 2;



1. **Exercise-6(SQL Exercise - Functions)**

**Employee Management System - SQL Exercises**

**Database Schema**

The Employee Management System database schema consists of the following tables:

1. Departments

| Column | Data Type | Description |

|---------------|---------------|------------------------------|

| DepartmentID | INT (PK) | Unique department ID |

| DepartmentName| VARCHAR(100) | Name of the department |

2. Employees

| Column | Data Type | Description |

|---------------|---------------|------------------------------|

| EmployeeID | INT (PK) | Unique employee ID |

| FirstName | VARCHAR(50) | Employee's first name |

| LastName | VARCHAR(50) | Employee's last name |

| DepartmentID | INT (FK) | Linked to Departments |

| Salary | DECIMAL(10,2) | Monthly salary |

| JoinDate | DATE | Date of joining |

**Sample Data**

Sample data for testing:

Departments:

| DepartmentID | DepartmentName |

|--------------|----------------|

| 1 | HR |

| 2 | IT |

| 3 | Finance |

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 | Bob | Johnson | 3 | 5500.00 | 2021-07-01

TASK:  **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:  
  
  
  
create database funcn\_learn;

use funcn\_learn;

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT IDENTITY(1,1) PRIMARY KEY,

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, 'IT'),

(3, 'Finance');

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'),

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

-- Creating Scalar Function to Calculate Annual Salary

CREATE FUNCTION fn\_CalculateAnnualSalary (@EmployeeID INT)

RETURNS DECIMAL(10,2)

AS

BEGIN

DECLARE @AnnualSalary DECIMAL(10,2);

SELECT @AnnualSalary = Salary \* 12

FROM Employees

WHERE EmployeeID = @EmployeeID;

RETURN @AnnualSalary;

END;

--Testing the function:

SELECT dbo.fn\_CalculateAnnualSalary(1) AS AnnualSalary;

