DotNet-FSE Mandatory Hands-On

WEEK-2 **NAME:** Sri Ranjani Priya P

TOPIC :- ADVANCED SQL

EXERCISE 1:

Implement Rank and Window Functions

What I Did:

In this exercise, I created an Employees table with department and salary data. I used a window function to understand the differences between RANK(),ROW_NUMBER() and DENSE_RANK() by applying them within each department (partitioned by department and ordered by salary descending).

SQL Query:

```
DROP TABLE  IF EXISTS Employees;
CREATE TABLE Employees (
   EmpID INT,
   Name VARCHAR(50),
   Department VARCHAR(50),
   Salary INT
);

INSERT INTO Employees VALUES
(1, 'Alice', 'HR', 60000),
(2, 'Bob', 'HR', 55000),
(3, 'Carol', 'IT', 75000),
(4, 'Dave', 'IT', 75000),
(5, 'Eve', 'IT', 70000),
(6, 'Frank', 'HR', 60000);

SELECT
   Name,
   Department,
   Salary,
   ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary DESC) AS
RowNum,
   RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS RankVal,
   DENSE_RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS
DenseRankVal
FROM Employees;
```

OUTPUT:

Name	Department	Salary	RowNum	RankVal	DenseRankVal
Alice	HR	60000	1	1	1
Frank	HR	60000	2	1	1
Bob	HR	55000	3	3	2
Carol	IT	75000	1	1	1
Dave	IT	75000	2	1	1
Eve	IT	70000	3	3	2

EXERCISE 2:

Implement SQL Index

What I Did:

In the SQL Index exercise, I created a table called Employee with columns for ID, name, department, and salary. I added some sample data with employees from HR and IT departments. Then, I created an index on the Salary column to make salary-based searches faster. After that, I ran a query to find employees who earn 75000. The index helped the query run more efficiently. I also checked the execution plan in SQL Server Management Studio to see the performance improvement.

SQL Query:

```
 DROP TABLE  IF EXISTS Employees;

CREATE TABLE Employees (
    EmpID INT,
    Name VARCHAR(50),
    Department VARCHAR(50),
```

```
    Salary INT
);

INSERT INTO Employees VALUES
(1, 'Alice', 'HR', 60000),
(2, 'Bob', 'HR', 55000),
(3, 'Carol', 'IT', 75000),
(4, 'Dave', 'IT', 75000),
(5, 'Eve', 'IT', 70000),
(6, 'Frank', 'HR', 60000);

CREATE INDEX idx_salary ON Employees (Salary);

SELECT * FROM Employees WHERE Salary = 75000;
```

OUTPUT:

EmpID	Name	Department	Salary
3	Carol	IT	75000
4	Dave	IT	75000

EXERCISE 3:

Create a Stored Procedure

What I Did:

I created an Employees table with four columns: EmpID, Name, Department, and Salary. I inserted five employee records. Then, I created a stored procedure named uspHighSalaryEmployees. This procedure returns all employees whose salary is greater than 70000, ordered in descending salary order.

SQL Query:

```
 DROP TABLE IF EXISTS Employees;

CREATE TABLE Employees (
   EmpID INT,
   Name VARCHAR(50),
   Department VARCHAR(50),
   Salary INT
);

INSERT INTO Employees VALUES
(1, 'Alice', 'HR', 60000),
(2, 'Bob', 'HR', 55000),
(3, 'Carol', 'IT', 75000),
(4, 'Dave', 'IT', 72000),
(5, 'Eve', 'Finance', 80000);

CREATE PROCEDURE uspHighSalaryEmployees
AS
BEGIN
   SELECT EmpID, Name, Department, Salary
   FROM Employees
   WHERE Salary > 70000
   ORDER BY Salary DESC;
END;
```

OUTPUT:

The table employees are created successfully, sample data was inserted, and the stored procedure uspHighSalaryEmployees was created without errors.

EXERCISE 4:

Execute a Stored Procedure and Return Data from the stored Procedure

What I Did:

I executed the stored procedure uspHighSalaryEmployees to return data from the Employees table where the salary is 75000.

SQL Query:

```
EXEC uspHighSalaryEmployees;
/ADD THIS CODE WITH CREATED STORED PROCEDURE*/
```

OUTPUT:

EmplID	Name	Department	Salary
5	Eve	Finance	80000
3	Carol	IT	75000
4	Dave	IT	72000

EXERCISE 5:

Create a Scalar Function

What I Did:

You created a scalar user-defined function dbo.udfNetSale that calculates the net sale amount after applying a discount on a product.

SQL Query:

```
DROP TABLE IF EXISTS SampleItems;

CREATE TABLE SampleItems (
    ItemID INT,
    Quantity INT,
    ListPrice DECIMAL(10,2),
    Discount DECIMAL(4,2)
);

INSERT INTO SampleItems VALUES
(1, 10, 5.00, 0.10),
```

```
(2, 2, 200.00, 0.05),
(3, 1, 800.00, 0.15),
(4, 5, 30.00, 0.00);



IF OBJECT_ID('dbo.udfNetSale', 'FN') IS NOT NULL
   DROP FUNCTION dbo.udfNetSale;
GO


CREATE FUNCTION dbo.udfNetSale (
   @Quantity INT,
   @ListPrice DECIMAL(10,2),
   @Discount DECIMAL(4,2)
)
RETURNS DECIMAL(10,2)
AS
BEGIN
   RETURN @Quantity * @ListPrice * (1 - @Discount);
END;
GO
```

OUTPUT:

Function dbo.udfNetSale created successfully.

EXERCISE 6:

Return Data from the Scalar Function

What I Did:

You created a table OrderItems, inserted sample data, and used the scalar function to calculate NetAmount for each row and displayed the output.

SQL Query:

```
(ADD THIS CODE TO THE PREVIOUS CREATE SCALAR FUNCTION CODE TO GET
THE NEEDED OUTPUT)
SELECT
   OrderID,
   ProductName,
```

```
    Quantity,
    ListPrice,
    Discount,
    dbo.udfNetSale(Quantity, ListPrice, Discount) AS NetAmount
FROM
    OrderItems
ORDER BY
    OrderID;
GO
```

OUTPUT:

OrderID	ProductName	Quantity	ListPrice	Discount	NetAmount
101	Pen	10	5.00	0.10	45.00
101	Notebook	5	30.00	0.00	150.00
102	Book	2	200.00	0.05	380.00
103	Bag	1	800.00	0.15	680.00