Day 4 Assignment: Amaan Shaikh

Joins and Subqueries

Types of Joins in SQL

1.INNER JOIN

- Returns only the rows where there is a match in both tables.
- **Example**:

SELECT Employees.EmployeeID, Employees.FirstName, Departments.DepartmentName

FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

2. LEFT JOIN

- Returns all rows from the left table, and matched rows from the right table. If there is no match, the result is NULL on the right side.
- **Example**:

SELECT Employees.EmployeeID, Employees.FirstName, Departments.DepartmentName

FROM Employees

LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

3. RIGHT JOIN

- Returns all rows from the right table, and matched rows from the left table. If there is no match, the result is NULL on the left side.
- **Example**:

SELECT Employees.EmployeeID, Employees.FirstName, Departments.DepartmentName

FROM Employees

RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

4. FULL OUTER JOIN

- Returns all rows when there is a match in either table. If there is no match, the result is NULL on the side without a match.
- **Example**:

SELECT Employees.EmployeeID, Employees.FirstName, Departments.DepartmentName

FROM Employees

FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

5. CROSS JOIN

- Returns the Cartesian product of the two tables, meaning every row in the first table is paired with every row in the second table.
- **Example**:

SELECT Employees.FirstName, Departments.DepartmentName

FROM Employees

CROSS JOIN Departments;

6. SELF JOIN

- Joins a table to itself, useful for hierarchical or recursive relationships within the same table.
- **Example**:

SELECT e1.EmployeeID AS EmployeeID, e1.FirstName AS EmployeeName, e2.FirstName AS ManagerName

FROM Employees e1

INNER JOIN Employees e2 ON e1.ManagerID = e2.EmployeeID;

7. EQUI JOIN

- A join that uses an equality condition to match rows between tables.
- **Example**:

SELECT Employees.EmployeeID, Employees.FirstName, Departments.DepartmentName

FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

Using GROUP BY and HAVING with Joins

- GROUP BY is used to group rows that have the same values in specified columns and allows aggregate functions like SUM, COUNT, AVG, etc., to be applied to each group.
- HAVING is used to filter groups after the GROUP BY clause has been applied. It is typically used with aggregate functions.
- **Example**:

SELECT Departments.DepartmentName, COUNT(Employees.EmployeeID) AS EmployeeCount

FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID

GROUP BY Departments.DepartmentName

HAVING COUNT(Employees.EmployeeID) > 5;

This query groups employees by department, counts the number of employees in each department, and only returns departments that have more than 5 employees.

Subqueries

A subquery is a query nested inside another query. It can be used in different parts of a SQL statement:

Subquery in the SELECT Clause

SELECT EmployeeID, FirstName, (SELECT DepartmentName FROM Departments WHERE DepartmentID = Employees.DepartmentID) AS DepartmentName

FROM Employees;

The subquery in the SELECT clause fetches the department name for each employee based on their DepartmentID.

Subquery in the FROM Clause

SELECT DepartmentName, AvgSalary

FROM (SELECT DepartmentID, AVG(Salary) AS AvgSalary FROM Employees GROUP BY DepartmentID) AS DeptSalaries

INNER JOIN Departments ON DeptSalaries.DepartmentID = Departments.DepartmentID;

The subquery in the FROM clause calculates the average salary per department, which is then joined with the Departments table to get department names.

Subquery in the WHERE Clause

SELECT FirstName, LastName

FROM Employees

WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE Location = 'New York');

The subquery in the WHERE clause filters employees who belong to departments located in 'New York'.