DE235 Soumyadeep Sinha
WEEK 1 - DAY 4 - 05-11-2024

SQL QUERIES , JOINS , FUNCTIONS , SUBQUERIES , STORED PROCEDURE , RANK , DENSE_RANK

JOINS

Joins are used to combine records from two or more tables in a database based on related columns.

Types of Joins:

INNER JOIN: Returns records that have matching values in both tables.

```
SELECT a.column1, b.column2
FROM TableA a
JOIN TableB b ON a.common_column = b.common_column;
```

LEFT JOIN (or LEFT OUTER JOIN): Returns all records from the left table and matched records from the right table. If there is no match, NULLs are returned for columns from the right table.

```
SELECT a.column1, b.column2
FROM TableA a
LEFT JOIN TableB b ON a.common_column = b.common_column;
```

RIGHT JOIN (or RIGHT OUTER JOIN): Returns all records from the right table and matched records from the left table. If there is no match, NULLs are returned for columns from the left table.

```
SELECT a.column1, b.column2
FROM TableA a
RIGHT JOIN TableB b ON a.common_column = b.common_column;
```

FULL JOIN (or FULL OUTER JOIN): Returns all records when there is a match in either left or right table records.

```
SELECT a.column1, b.column2
FROM TableA a
FULL JOIN TableB b ON a.common_column = b.common_column;
```

SQL Functions

SQL functions are reusable code blocks that perform specific operations and return either a single value or a table of values. Functions can be broadly classified into **System Functions** and **User-Defined Functions (UDFs)**.

System Functions

System functions are built into SQL and perform common operations, either on aggregate data or individual values.

Types of System Functions:

Aggregate Functions: Used to perform calculations on a set of values and return a single result.

Examples: COUNT, SUM, AVG, MIN, MAX

```
SELECT COUNT(*) AS SalesEmployees
FROM Employees
WHERE department = 'Sales';
```

```
SELECT SUM(salary) AS TotalSalary
FROM Employees;
```

```
SELECT AVG(salary) AS AverageSalary
FROM Employees;
```

```
SELECT MIN(salary) AS MinimumSalary
FROM Employees;
```

```
SELECT MAX(salary) AS MaximumSalary
FROM Employees;
```

User-Defined Functions (UDFs)

UDFs are custom functions created by users to perform specific calculations or operations not covered by system functions. They can take parameters, return a single value (scalar function), or return a table (table-valued function).

Types of User-Defined Functions:

Scalar User-Defined Functions: Return a single value based on input parameters.

```
CREATE FUNCTION CalculateBonus (@salary DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
AS
BEGIN
    RETURN @salary * 0.10;
END;
```

```
SELECT name, dbo.CalculateBonus(salary) AS Bonus
FROM Employees;
```

Table-Valued User-Defined Functions: Return an entire table, allowing users to define custom sets of data.

```
CREATE FUNCTION EmployeesByDepartment (@departmentId INT)
RETURNS TABLE
AS
RETURN
(
    SELECT id, name, salary
    FROM Employees
    WHERE department_id = @departmentId
);

SELECT * FROM dbo.EmployeesByDepartment(2);
```

SUBQUERIES

A subquery is a query nested inside another query. It can return a single value or a set of values.

```
SELECT name
FROM Employees
WHERE salary > (SELECT AVG(salary) FROM Employees);
```

STORED PROCEDURES

Stored procedures are precompiled SQL statements that can be saved and reused. They can accept parameters and perform complex operations.

Syntax:

```
CREATE PROCEDURE ProcedureName (@param1 INT)
AS
BEGIN
  SELECT *
  FROM Employees
  WHERE department_id = @param1;
END;
```

Calling a Stored Procedure:

```
EXEC ProcedureName @param1 = 2;
```

RANK and DENSE_RANK

Both RANK and DENSE_RANK are window functions used to assign a rank to each row within a partition of a result set.

RANK:

RANK assigns a unique rank number to each distinct row within a partition, with gaps in the ranking if there are ties.

```
SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS SalaryRank
FROM Employees;
```

DENSE_RANK:

DENSE_RANK also assigns a unique rank number, but it does not leave gaps in the ranking when there are ties.

```
SELECT name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS
SalaryDenseRank
FROM Employees;
```

Differences between RANK and DENSE_RANK

Ranking Method:

- **RANK**: Assigns ranks with gaps. For example, if two employees are tied for rank 1, the next rank will be 3.
- **DENSE_RANK**: Assigns ranks without gaps. The next rank after a tie for rank 1 will be 2.

Use Case: Use RANK when you need to reflect the number of distinct values. Use DENSE_RANK when you want continuous ranking without gaps.