Module 2 - Day 27

Querying Data Using Built-in Functions and T-SQL (Part 1)

1. Introduction to T-SQL (Transact-SQL)

- **T-SQL (Transact Structured Query Language)** is an extension of SQL developed by **Microsoft** for **SQL Server**.
- It **enhances standard SQL** by adding **procedural programming features**, such as variables, loops, conditions, and error handling.
- **Main Uses of T-SQL:**
    - Writing full **procedural programs** within SQL Server.
    - Defining **transactions and business logic** in stored procedures and functions.
    - Running complex **queries and aggregations** efficiently.
    - Managing **database interactions** without requiring frequent application-level processing.

2. Built-in Functions in T-SQL

SQL Server provides **many built-in functions** that help manipulate and analyze data. These functions can be grouped into several categories:

a) Scalar Functions

Return **a single value** for each row.
Examples:

- `UPPER()` → Converts text to uppercase.
- `LOWER()` → Converts text to lowercase.
- `LEN()` → Returns the length of a string.
- `GETDATE()` → Returns the current system date/time.

Example Query:

```
SELECT UPPER('hello world') AS UpperCaseText;
```

Output: `HELLO WORLD`

b) Aggregate Functions

Aggregate functions perform calculations **on a set of rows** and return **a single value**. They are used with `GROUP BY` to summarize data.

Common Aggregate Functions:

Function	Description
`AVG(column)`	Returns the average value of the column.
`MAX(column)`	Returns the maximum value of the column.
`MIN(column)`	Returns the minimum value of the column.
`SUM(column)`	Returns the sum of all values in the column.
`COUNT(column)`	Returns the number of rows (excluding NULLs).
`COUNT_BIG(column)`	Same as `COUNT`, but returns `BIGINT` (useful for large datasets).

Note: Aggregate functions **ignore NULL values** automatically.

Example Query (Using Aggregate Functions)

```
SELECT
    AVG(Salary) AS AverageSalary,
    MAX(Age) AS OldestEmployee,
    MIN(Age) AS YoungestEmployee,
    SUM(Salary) AS TotalSalaryPaid,
    COUNT(*) AS TotalEmployees
FROM PermanentEmployee;
```

c) Statistical Aggregate Functions

These functions perform **statistical calculations** on a group of values.

Function	Description
`VAR(column)`	Returns the **sample variance** of values.
`VARP(column)`	Returns the **population variance** of values.
`STDEV(column)`	Returns the **sample standard deviation** of values.
`STDEVP(column)`	Returns the **population standard deviation** of values.

Example Query:

```
SELECT
    VAR(Salary) AS SalaryVariance,
    STDEV(Salary) AS SalaryStandardDeviation
FROM PermanentEmployee;
```

d) Ranking Functions

These functions **assign ranks to rows** within a partition.

Function	Description
`ROW_NUMBER()`	Assigns a unique row number to each row.
`RANK()`	Assigns a ranking, allowing ties (same rank for duplicate values).
`DENSE_RANK()`	Similar to `RANK()`, but without gaps in ranking.
`NTILE(n)`	Divides the result set into `n` buckets.

Example Query (Using Ranking Functions):

```
SELECT Name, Salary,
       RANK() OVER (ORDER BY Salary DESC) AS SalaryRank,
       DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseSalaryRank
FROM PermanentEmployee;
```

e) Rowset Functions

These functions **return a table-like dataset** instead of a single value.
Examples:

- `OPENQUERY()` → Executes a pass-through query on a linked server.
- `OPENDATASOURCE()` → Queries data from a remote data source.
- `OPENROWSET()` → Accesses data from an external provider.

Example Query (Using `OPENROWSET`):

```
SELECT * FROM OPENROWSET('SQLNCLI',
    'Server=RemoteServer;Trusted_Connection=yes;',
    'SELECT * FROM RemoteDatabase.dbo.Employees');
```

f) Logical Functions

These functions **return Boolean values** (`TRUE`, `FALSE`, or `NULL`).
Examples:

- `ISNULL(column, default_value)` → Replaces NULL with a specified value.
- `COALESCE(value1, value2, ...)` → Returns the first non-null value.
- `CASE` → Evaluates conditions and returns different values.

Example Query (Using `CASE` and `ISNULL`):

```
SELECT Name,
       ISNULL(Department, 'Unknown') AS Department,
       CASE
           WHEN Salary > 80000 THEN 'High Salary'
           WHEN Salary BETWEEN 50000 AND 80000 THEN 'Medium Salary'
           ELSE 'Low Salary'
       END AS SalaryCategory
FROM PermanentEmployee;
```

3. User-Defined Aggregate Functions in SQL Server

SQL Server allows developers to **create their own aggregate functions** using .NET.

Steps to Create a User-Defined Aggregate Function

1. **Define the aggregate function as a class** in a .NET-supported language (C# or VB.NET).
2. **Register the assembly** in SQL Server using `CREATE ASSEMBLY`.
3. **Create the aggregate function** referencing the assembly using `CREATE AGGREGATE`.

4. Summary of Key Concepts

✅**T-SQL** extends SQL with procedural programming features.
✅**Built-in functions** include scalar, aggregate, statistical, ranking, rowset, and logical functions.
✅**Aggregate functions** summarize data (e.g., `SUM()`, `AVG()`, `COUNT()`).
✅**Statistical functions** compute variance and standard deviation.
✅**Ranking functions** help order and rank data.
✅**User-defined aggregates** allow custom calculations in SQL Server.

Example Assignment

Try the following queries on a sample Employee database:

1. Retrieve the **average salary** of employees.
2. Find the **total number of employees** in the company.
3. Retrieve the **highest and lowest salary** of employees.
4. Rank employees based on **their salaries**.
5. Use `CASE` to **categorize employees** based on their age groups.