Mastering User-Defined Functions (UDFs) in MS SQL Server

Table of Contents

---

1. Introduction to UDFs

Definition: A **User-Defined Function (UDF)** is a programmable routine you can create in MS SQL Server to encapsulate reusable logic that can return a single value or a table.

Why use UDFs?

- Encapsulate repetitive logic.
- Improve code readability.
- Reuse business logic across multiple queries, views, or stored procedures.
- Parameterized calculations or transformations.

Important Notes:

- UDFs **cannot** modify database state (no `INSERT`, `UPDATE`, `DELETE` inside the function, except using table variables in table-valued functions).
- They can be **scalar** (single value) or **table-valued** (return a table).

---

2. Types of UDFs in MS SQL

Type	Description	Example Use Case
Scalar Function	Returns a single value	Calculating tax, formatting strings
Inline Table-Valued Function (iTVF)	Returns a table via a single `SELECT` statement	Filtering customers, joining tables dynamically
Multi-Statement Table-Valued Function (mTVF)	Returns a table with multiple statements	Complex aggregations, intermediate calculations

3. Creating Scalar Functions

Syntax:

```
CREATE FUNCTION [schema_name].[function_name]
(
    @param1 data_type,
    @param2 data_type
)
RETURNS return_data_type
AS
BEGIN
    DECLARE @Result return_data_type;

    -- Function logic here
    SET @Result = ...;

    RETURN @Result;
END;
```

Example: Calculate a 10% tax on an amount

```
CREATE FUNCTION dbo.CalculateTax(@Amount DECIMAL(10,2))
RETURNS DECIMAL(10,2)
AS
BEGIN
    RETURN @Amount * 0.10;
END;
```

Usage:

```
SELECT dbo.CalculateTax(500.00) AS TaxAmount;
```

Output:

```
TaxAmount
50.00
```

---

4. Creating Inline Table-Valued Functions (iTVF)

Syntax:

```
CREATE FUNCTION [schema_name].[function_name] (@param datatype)
RETURNS TABLE
AS
RETURN
(
    SELECT ...
    FROM ...
    WHERE ...
);
```

Example: Get all orders for a specific customer

```
CREATE FUNCTION dbo.GetCustomerOrders(@CustomerID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT OrderID, OrderDate, TotalAmount
    FROM Sales.Orders
    WHERE CustomerID = @CustomerID
);
```

Usage:

```
SELECT *
FROM dbo.GetCustomerOrders(1);
```

5. Creating Multi-Statement Table-Valued Functions (mTVF)

Syntax:

```
CREATE FUNCTION [schema_name].[function_name] (@param datatype)
RETURNS @ResultTable TABLE (
    Column1 datatype,
    Column2 datatype,
    ...
)
AS
BEGIN
    -- Logic to populate the table
    INSERT INTO @ResultTable
    SELECT ...
    FROM ...
    WHERE ...;

    RETURN;
END;
```

Example: Get top N customers by total sales

```
CREATE FUNCTION dbo.TopCustomers(@TopN INT)
RETURNS @TopCustomerTable TABLE (
    CustomerID INT,
    TotalSales DECIMAL(18,2)
)
AS
BEGIN
    INSERT INTO @TopCustomerTable
    SELECT TOP(@TopN) CustomerID, SUM(TotalAmount)
    FROM Sales.Orders
    GROUP BY CustomerID
    ORDER BY SUM(TotalAmount) DESC;

    RETURN;
END;
```

Usage:

```
SELECT *
FROM dbo.TopCustomers(5);
```

6. Calling and Using UDFs

- **Scalar UDF:** Can be used anywhere an expression is valid.

```sql
SELECT CustomerName, dbo.CalculateTax(TotalAmount) AS Tax
FROM Sales.Orders;
```

- **Table-Valued UDF:** Can be used like a table.

```sql
SELECT *
FROM dbo.GetCustomerOrders(1);
```

- **Join with other tables:**

```sql
SELECT c.CustomerName, o.OrderID, o.TotalAmount
FROM Sales.Customers c
JOIN dbo.GetCustomerOrders(c.CustomerID) o
    ON c.CustomerID = o.CustomerID;
```

7. UDFs with Parameters

- UDFs can accept multiple parameters.
- Parameters can be **optional** using `DEFAULT` values.

Example: Get orders within a date range

```sql
CREATE FUNCTION dbo.GetOrdersByDate
(
    @StartDate DATE,
    @EndDate DATE = GETDATE()
)
RETURNS TABLE
AS
RETURN
(
    SELECT OrderID, CustomerID, OrderDate, TotalAmount
    FROM Sales.Orders
    WHERE OrderDate BETWEEN @StartDate AND @EndDate
);
```

Usage:

```sql
SELECT *
FROM dbo.GetOrdersByDate('2025-01-01');
```

---

8. Best Practices for UDFs

1. **Prefer inline TVFs over multi-statement TVFs** for performance.
2. **Avoid scalar UDFs in large queries**; they can be performance bottlenecks.
3. **Use schema-qualified names** (e.g., `dbo.FunctionName`) to avoid ambiguity.
4. **Document the function** with comments.
5. **Keep UDF logic deterministic** whenever possible.

---

9. Performance Considerations

- Scalar UDFs can slow down queries due to row-by-row execution. Consider **inline TVFs** or **computed columns**.
- Multi-statement TVFs store intermediate results in memory; can impact large datasets.
- Indexing on tables returned by TVFs is **not supported directly**. Consider converting to a view if indexing is required.

Tips:

- Test UDFs with `SET STATISTICS IO ON` and `SET STATISTICS TIME ON` to measure impact.

- SQL Server 2019+ has **Scalar UDF Inlining** for performance improvement.

---

10. Advanced Scenarios

1. **Recursive Functions** – for hierarchical or tree data.

```sql
CREATE FUNCTION dbo.Factorial(@n INT)
RETURNS INT
AS
BEGIN
    IF @n = 0 RETURN 1;
    RETURN @n * dbo.Factorial(@n - 1);
END;
```

> **Note:** Use recursion carefully — deep recursion can lead to stack overflow and poor performance. For hierarchical data, consider `CTE` (Common Table Expressions) or iterative approaches when appropriate.

2. **Dynamic filtering in TVFs** – combine parameters for flexible queries.
3. **Combining UDFs** – use scalar functions inside TVFs.

---

11. Practice Exercises

1. Create a **scalar UDF** to calculate a discounted price based on a percentage.
2. Create an **inline TVF** to fetch all employees from a specific department.
3. Create a **multi-statement TVF** to calculate monthly sales totals for each product.
4. Write a query that **joins a TVF with a regular table** and filters results using a scalar UDF.
5. Optimize a query that uses a scalar UDF in a `SELECT` statement for thousands of rows.

Next steps

- Save or download this document for offline reference.
- If you want, I can convert these examples to use the **AdventureWorks2022** schema (I know you practice with that DB) and provide exercises that run against its tables.

End of tutorial.