Microsoft.NET Fullstack Bootcamp Training

Day22 (7th August, 2025)

Stored Procedures & UDFs in SQL Server

By

Narasimha Rao T

Microsoft.Net FSD Trainer

Professional Development Trainer

tnrao.trainer@gmail.com

1. What is Stored Procedures?

Introduction

- A **Stored Procedure (SP)** is a precompiled collection of one or more SQL statements that are stored under a name and processed as a unit.
- It acts as a subroutine or function that can be executed on demand.
- It often performing a specific task or a sequence of operations on the database.

Why Do We Use Stored Procedures?

- To **encapsulate business logic**
- To **reduce code duplication**
- To **improve performance** (SPs are compiled and cached)
- For **modular programming**
- To **secure access** to data through controlled interfaces

Advantages

- **Improved Performance**: Reuse of execution plans
- **Security**: Can grant execution rights without table access
- **Reduced Network Traffic**: Only procedure call is sent over network
- **Easy Maintenance**: Changes in SP don't affect applications
- **Reusability**: Can be reused by multiple applications or reports

Real-Time Usages

- Web applications: Fetch/update user info
- Dashboards: Generate summary reports
- Scheduled Jobs: Email alerts or daily processing
- API integrations: Controlled data access via SP

Creating a Stored Procedure

```sql
CREATE PROCEDURE usp_GetAllEmployees
AS
BEGIN
    SELECT * FROM Employees;
END
```

Stored Procedure with Parameters

IN (Input) Parameters

```sql
CREATE PROCEDURE usp_GetEmployeeByID
    @EmpID INT
AS
BEGIN
    SELECT * FROM Employees WHERE EmployeeID = @EmpID;
END
```

OUT (Output) Parameters

```sql
CREATE PROCEDURE usp_GetEmployeeCount
    @EmpCount INT OUTPUT
AS
BEGIN
    SELECT @EmpCount = COUNT(*) FROM Employees;
END;

-- Execution:
DECLARE @Total INT;
EXEC usp_GetEmployeeCount @EmpCount = @Total OUTPUT;
PRINT @Total;
```

Default Values

```
CREATE PROCEDURE usp_GetEmployeesByDept
    @DeptID INT = 1
AS
BEGIN
    SELECT * FROM Employees WHERE DepartmentID = @DeptID;
END
```

Executing a Stored Procedure

```sql
-- With parameters
EXEC usp_GetEmployeeByID @EmpID = 101;

-- Without parameters
EXEC usp_GetAllEmployees;
```

ALTERING a Procedure

```
ALTER PROCEDURE usp_GetAllEmployees
AS
BEGIN
    SELECT EmployeeID, FirstName, LastName FROM Employees;
END
```

DROPPING a Procedure

```
DROP PROCEDURE usp_GetEmployeeByID;
```

Returning Data from Stored Procedures

- Using `SELECT` statements (inline)
- Using `OUTPUT` parameters
- Can also return integer status (e.g., return 0 for success)

2. Basic Error Handling in T-SQL

TRY...CATCH

```sql
BEGIN TRY
    -- Potentially failing code
    INSERT INTO Employees (EmployeeID, FirstName) VALUES (1, 'John');
END TRY
BEGIN CATCH
    PRINT 'Error occurred:';
    PRINT ERROR_MESSAGE();
END CATCH;
```

RAISERROR

Used to raise a custom error message.

```
RAISERROR('Invalid Department ID', 16, 1);
```

Parameters:

- Message text
- Severity (1–25)
- State (user-defined number)

3. User-Defined Functions (UDFs)

What is a UDF?

A **User-Defined Function** is a function created by the user to perform calculations, return data, or encapsulate logic. UDFs **must return a value**.

Types of UDFs

1. **Scalar Functions** – return a single value.
2. **Table-Valued Functions (TVF)** – return a table.

Creating a Scalar UDF

```sql
CREATE FUNCTION dbo.fn_GetFullName
(
    @FirstName NVARCHAR(50),
    @LastName NVARCHAR(50)
)
RETURNS NVARCHAR(101)
AS
BEGIN
    RETURN (@FirstName + ' ' + @LastName);
END
```

Usage:

```sql
SELECT dbo.fn_GetFullName('John', 'Doe');
```

Creating a Table-Valued Function

```
CREATE FUNCTION dbo.fn_GetEmployeesByDept
(
    @DeptID INT
)
RETURNS TABLE
AS
RETURN (
    SELECT EmployeeID, FirstName, LastName
    FROM Employees
    WHERE DepartmentID = @DeptID
);
```

Usage:

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

Real-Time UDF Usages

- Full name formatting
- Age calculation from DOB
- Reusable business logic in views or procedures
- Dynamic filters in SELECT queries

QUIZ
TIME

Quiz Questions

1. What are the differences between Stored Procedures and UDFs?
2. Explain the benefits of using stored procedures.
3. What is the purpose of `OUTPUT` parameters?
4. When would you use RAISERROR over @@ERROR?
5. What are the different types of UDFs in SQL Server?
6. What is the difference between scalar-valued and table-valued functions?
7. How is a UDF different from a stored procedure?
8. Can a UDF call a stored procedure or vice versa? Why or why not?
9. What are the limitations of UDFs in SQL Server?
10. How do you handle exceptions in stored procedures?

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com