**Stored Procedures in T-SQL**

**What is a Stored Procedure?**

A **Stored Procedure** is a precompiled set of SQL statements stored in the database. Instead of writing SQL queries repeatedly, you can save them in the database and call the procedure whenever needed.

### ****Why Use Stored Procedures?****

1. **Reusability**: Write once, use many times.
2. **Security**: Users can execute the procedure without directly accessing the underlying tables.
3. **Performance**: Precompiled for faster execution.
4. **Modularity**: Organize complex logic into reusable pieces.
5. **Parameterization**: Pass input/output values for flexibility.

CREATE PROCEDURE ProcedureName

@Parameter1 DataType,

@Parameter2 DataType = DefaultValue -- Optional with default value

AS

BEGIN

-- SQL statements go here

END;

### ****Example 1: Stored Procedure Without Parameters****

This procedure retrieves all student records.

CREATE PROCEDURE GetAllStudents

AS

BEGIN

SELECT \* FROM Students;

END;

**To Execute the Procedure**

EXEC GetAllStudents;

### ****Example 2: Stored Procedure With Input Parameters****

This procedure retrieves students with marks greater than a given value.

CREATE PROCEDURE GetStudentsByMarks

@MinMarks INT

AS

BEGIN

SELECT \* FROM Students WHERE Marks >= @MinMarks;

END;

**To Execute the Procedure**

EXEC GetStudentsByMarks @MinMarks = 50;

### ****Example 3: Stored Procedure With Output Parameter****

This procedure calculates and returns the total number of students.

CREATE PROCEDURE GetStudentCount

@TotalStudents INT OUTPUT

AS

BEGIN

SELECT @TotalStudents = COUNT(\*) FROM Students;

END;

📌 **To Execute the Procedure**

DECLARE @Count INT;

EXEC GetStudentCount @TotalStudents = @Count OUTPUT;

PRINT 'Total Students: ' + CAST(@Count AS VARCHAR(10));

### ****Advanced Example: Using IF...ELSE in Stored Procedures****

📌 This procedure checks a student's grade and returns a message:

CREATE PROCEDURE GetStudentGrade

@Marks INT

AS

BEGIN

IF @Marks >= 90

PRINT 'Grade: A+';

ELSE IF @Marks >= 80

PRINT 'Grade: A';

ELSE

PRINT 'Grade: Fail';

END;

📌 **To Execute**

EXEC GetStudentGrade @Marks = 85;

### ****Modifying a Stored Procedure****

To change an existing procedure:

ALTER PROCEDURE ProcedureName

AS

BEGIN

-- Modified SQL statements

END;

### ****Deleting a Stored Procedure****

To remove a procedure:

DROP PROCEDURE ProcedureName;

**Exercise:**

1. Write a procedure to insert a new student into the Students table with Name and Marks as input parameters.
2. Write a procedure to update a student’s marks using their StudentID.

**User-Defined Functions (UDF) in T-SQL**

**What is a User-Defined Function (UDF)?**

A **User-Defined Function** is a piece of reusable code that performs a specific task and **returns a value**. Unlike stored procedures, functions **must return a value** and are often used in SELECT queries, WHERE conditions, or calculations.

**Types of UDFs in SQL Server**

1. **Scalar Functions**: Return a single value (e.g., INT, VARCHAR, etc.).
2. **Table-Valued Functions**: Return a table.
   * **Inline Table-Valued Functions**: Single SELECT statement.
   * **Multi-Statement Table-Valued Functions**: Multiple statements with table construction.

**Why Use UDFs?**

1. Reusability: Define once and use multiple times.
2. Simplify Queries: Replace complex expressions with function calls.
3. Modularity: Break down logic into smaller, reusable components.
4. Consistency: Ensure the same logic is used across queries.

### ****Create a Scalar UDF****

CREATE FUNCTION FunctionName (@Parameter DataType)

RETURNS ReturnType

AS

BEGIN

-- Function logic

RETURN SomeValue;

END;

### ****Examples of Scalar Functions****

#### 📌 **Example 1: Scalar Function**

Create a function to calculate the square of a number.

CREATE FUNCTION GetSquare(@Number INT)

RETURNS INT

AS

BEGIN

RETURN @Number \* @Number;

END;

**To Call the Function:**

SELECT dbo.GetSquare(5) AS Square;

first and last names into a full name.

CREATE FUNCTION GetFullName(@FirstName VARCHAR(50), @LastName VARCHAR(50))

RETURNS VARCHAR(100)

AS

BEGIN

RETURN @FirstName + ' ' + @LastName;

END;

**To Call the Function:**

SELECT dbo.GetFullName('John', 'Doe') AS FullName;

### ****Syntax: Create an Inline Table-Valued Function****

CREATE FUNCTION FunctionName (@Parameter DataType)

RETURNS TABLE

AS

RETURN

(

SELECT Columns FROM Table WHERE Condition

);

### ****Examples of Table-Valued Functions****

#### 📌 **Example 3: Inline Table-Valued Function**

Return all students with marks above a certain value.

CREATE FUNCTION GetTopStudents(@MinMarks INT)

RETURNS TABLE

AS

RETURN

(

SELECT \* FROM Students WHERE Marks >= @MinMarks

);

**To Call the Function:**

SELECT \* FROM dbo.GetTopStudents(80);

**Key Concept**

1. **Stored Procedures**:
   * Stored procedures **can return values**, but it’s not mandatory.
   * They are often used to perform actions, like updating, inserting, or deleting data.
   * Stored procedures can return multiple outputs (via OUTPUT parameters), or they may simply execute without returning anything.
2. **Functions**:
   * Functions **must always return a value** (a scalar value, a table, etc.).
   * They are primarily used in calculations or to retrieve data, often within queries like SELECT or WHERE.