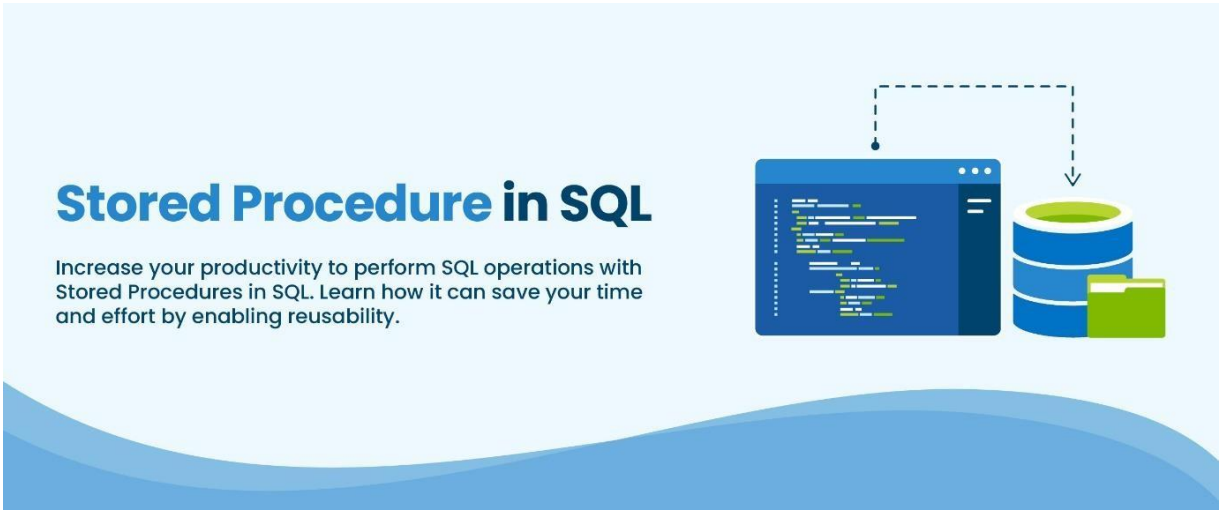
Stored Procedure in SQL

A **stored procedure** is a precompiled set of one or more SQL statements, which are saved in the database and can be executed as needed. It allows you to group multiple SQL statements into a logical unit and execute them together, often improving performance by reducing the need to send individual statements to the database server repeatedly.



Types of Stored Procedures

1. **User-defined Stored Procedures**
2. **System Stored Procedures**
3. **Temporary Stored Procedures**

1. User-Defined Stored Procedures

These are created by users to perform specific tasks, such as inserting, updating, deleting, or retrieving data. They are written in T-SQL (Transact-SQL).

```
CREATE PROCEDURE ProcedureName
AS
BEGIN
-- SQL statements
END
```

Example:

```
CREATE PROCEDURE GetEmployeeData
    AS
    BEGIN
      SELECT * FROM Employees;
    END;
```

In this example, the stored procedure GetEmployeeData retrieves all employee data from the Employees table. You can execute it by calling:

```
EXEC GetEmployeeData;
```

Here are few examples of creating the store procedure.

1. Insert New Employee Record

```
-- Create Procedure
```

```
CREATE PROCEDURE InsertEmployee
    @FirstName NVARCHAR(50),
    @LastName NVARCHAR(50),
   @Department NVARCHAR(50)
AS
BEGIN
    INSERT INTO Employees (FirstName, LastName, Department)
   VALUES (@FirstName, @LastName, @Department);
END;
GO


-- Execute Procedure
EXEC InsertEmployee @FirstName = 'John', @LastName = 'Doe',
@Department = 'IT';
```

Explanation:
This procedure inserts a new employee's details (first name, last name, and department) into the Employees table. When executed, it adds a new record to the table.

---

2. Update Employee Salary

```
-- Create Procedure
CREATE PROCEDURE UpdateEmployeeSalary
    @EmployeeID INT,
   @NewSalary DECIMAL(10,2)
AS
BEGIN
    UPDATE Employees
```

```
    SET Salary = @NewSalary
    WHERE EmployeeID = @EmployeeID;
END;
GO
```

-- Execute Procedure

EXEC UpdateEmployeeSalary @EmployeeID = 101, @NewSalary = 75000.00;

Explanation:
This procedure updates the salary of an employee based on their EmployeeID. The @NewSalary parameter is provided to set the new salary for that specific employee.

---

3. Delete an Employee Record

```
-- Create Procedure
CREATE PROCEDURE DeleteEmployee
    @EmployeeID INT
AS
BEGIN
    DELETE FROM Employees
    WHERE EmployeeID = @EmployeeID;
END;
GO
```

-- Execute Procedure

EXEC DeleteEmployee @EmployeeID = 101;

Explanation:
This procedure deletes an employee's record from the Employees table

using the provided EmployeeID. Once executed, the employee with the given ID will be removed from the database.

---

4. Get All Employees from a Specific Department

```
-- Create Procedure
CREATE PROCEDURE GetEmployeesByDepartment
  @Department NVARCHAR(50)
AS
BEGIN
  SELECT * FROM Employees
  WHERE Department = @Department;
END;
GO
```

```
-- Execute Procedure
EXEC GetEmployeesByDepartment @Department = 'IT';
```

Explanation:
This procedure fetches all employees who work in a specific department. It takes a department name as a parameter and returns the employee details for that department.

---

5. Calculate Total Sales for a Product

```
-- Create Procedure
CREATE PROCEDURE GetTotalSalesByProduct
  @ProductID INT
AS
BEGIN
```

```
    SELECT SUM(Quantity * UnitPrice) AS TotalSales
    FROM OrderDetails
    WHERE ProductID = @ProductID;
END;
GO

-- Execute Procedure
EXEC GetTotalSalesByProduct @ProductID = 101;
```

Explanation:
This procedure calculates the total sales of a specific product by multiplying the quantity sold by the unit price in the OrderDetails table. The ProductID is passed as a parameter to determine the sales for that product.

---

6. Fetch Employee Information by ID

```
-- Create Procedure
CREATE PROCEDURE GetEmployeeByID
    @EmployeeID INT
AS
BEGIN
    SELECT * FROM Employees
    WHERE EmployeeID = @EmployeeID;
END;
GO

-- Execute Procedure
EXEC GetEmployeeByID @EmployeeID = 102;
```

Explanation:
This procedure retrieves the details of an employee based on their EmployeeID. It returns all columns for the matching employee.

---

7. Update Employee Department

```
-- Create Procedure
CREATE PROCEDURE UpdateEmployeeDepartment
    @EmployeeID INT,
   @NewDepartment NVARCHAR(50)
AS
BEGIN
    UPDATE Employees
   SET Department = @NewDepartment
   WHERE EmployeeID = @EmployeeID;
END;
GO


-- Execute Procedure
EXEC UpdateEmployeeDepartment @EmployeeID = 102,
@NewDepartment = 'HR';
```

Explanation:
This procedure updates the department of an employee using their EmployeeID and sets it to the new department provided.

---

8. Get Total Number of Employees

```
-- Create Procedure
CREATE PROCEDURE GetTotalEmployees
AS
```

```
BEGIN
  SELECT COUNT(*) AS TotalEmployees
  FROM Employees;
END;
GO
```

```
-- Execute Procedure
EXEC GetTotalEmployees;
```

Explanation:
This procedure returns the total number of employees in the Employees table by counting the records.

---

9. Insert New Product Record

```
-- Create Procedure
CREATE PROCEDURE InsertProduct
  @ProductName NVARCHAR(50),
  @Price DECIMAL(10,2)
AS
BEGIN
  INSERT INTO Products (ProductName, Price)
  VALUES (@ProductName, @Price);
END;
GO
```

```
-- Execute Procedure
EXEC InsertProduct @ProductName = 'Laptop', @Price = 1000.00;
```

Explanation:
This procedure inserts a new product into the Products table with the provided ProductName and Price.

---

10. Get Products Above a Certain Price

```
-- Create Procedure
CREATE PROCEDURE GetProductsAbovePrice
  @Price DECIMAL(10,2)
AS
BEGIN
  SELECT * FROM Products
  WHERE Price > @Price;
END;
GO
```

```
-- Execute Procedure
EXEC GetProductsAbovePrice @Price = 500.00;
```

Explanation:
This procedure retrieves all products from the Products table that have a price greater than the specified amount. It filters the products based on the provided @Price parameter.

4. **Temporary Stored Procedures**

A **temporary stored procedure** is similar to a regular (user-defined) stored procedure but it is created and stored in the **tempdb** system database.

Temporary stored procedures are used for short-term or session-specific tasks, and they are automatically deleted once the session or connection that created them is closed or after the server restarts.

There are two types of temporary stored procedures:

1. **Local Temporary Stored Procedure**:
   These are available only to the session that created them. They are deleted automatically when the session ends.
2. **Global Temporary Stored Procedure**:
   These are available to all sessions. They are deleted only after all the sessions that are using the procedure have been closed.

---

1. Local Temporary Stored Procedure

A local temporary stored procedure is prefixed with a single #. It is visible only to the connection that created it.

Example: Local Temporary Stored Procedure

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #InsertTemporaryEmployee
   @FirstName NVARCHAR(50),
   @LastName NVARCHAR(50),
  @Department NVARCHAR(50)
AS
BEGIN
   INSERT INTO #TempEmployees (FirstName, LastName, Department)
  VALUES (@FirstName, @LastName, @Department);
END;
GO
```

-- Execute Procedure

EXEC #InsertTemporaryEmployee @FirstName = 'Jane', @LastName = 'Smith', @Department = 'HR';

Explanation:
This local temporary stored procedure inserts an employee's details (first name, last name, and department) into a temporary table called #TempEmployees. The procedure is only available during the current session.

---

2. Global Temporary Stored Procedure

A global temporary stored procedure is prefixed with ##. It can be accessed by any session as long as at least one session is still using it.

Example: Global Temporary Stored Procedure

```
-- Create Global Temporary Procedure
CREATE PROCEDURE ##GetTemporaryEmployees
AS
BEGIN
  SELECT * FROM #TempEmployees;
END;
GO
```

-- Execute Procedure

EXEC ##GetTemporaryEmployees;

Explanation:
This global temporary stored procedure retrieves all records from the temporary table #TempEmployees. The procedure can be used by any session, not just the one that created it, as long as the procedure still exists.

Temporary Stored Procedure Examples with Full Code and Explanations

1. Insert a Temporary Employee

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #InsertTempEmployee
    @FirstName NVARCHAR(50),
    @LastName NVARCHAR(50),
    @Department NVARCHAR(50)
AS
BEGIN
    INSERT INTO #TempEmployees (FirstName, LastName, Department)
    VALUES (@FirstName, @LastName, @Department);
END;
GO

-- Execute Procedure
EXEC #InsertTempEmployee @FirstName = 'Mark', @LastName = 'Johnson', @Department = 'Sales';
```

Explanation:
This procedure inserts a new employee's details into the #TempEmployees table. The procedure is available only to the session that created it.

---

2. Update a Temporary Employee's Department

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #UpdateTempEmployeeDepartment
    @EmployeeID INT,
    @NewDepartment NVARCHAR(50)
AS
BEGIN
    UPDATE #TempEmployees
```

```
    SET Department = @NewDepartment
    WHERE EmployeeID = @EmployeeID;
END;
GO
```

-- Execute Procedure

EXEC #UpdateTempEmployeeDepartment @EmployeeID = 1, @NewDepartment = 'Finance';

Explanation:
This procedure updates the department of an employee in the temporary table based on their EmployeeID.

---

3. Delete a Temporary Employee Record

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #DeleteTempEmployee
    @EmployeeID INT
AS
BEGIN
    DELETE FROM #TempEmployees
    WHERE EmployeeID = @EmployeeID;
END;
GO
```

-- Execute Procedure

EXEC #DeleteTempEmployee @EmployeeID = 1;

Explanation:
This procedure deletes an employee's record from the #TempEmployees table by using the provided EmployeeID.

---

4. Get Temporary Employee Details by ID

```
-- Create Global Temporary Procedure
CREATE PROCEDURE ##GetTempEmployeeByID
  @EmployeeID INT
AS
BEGIN
   SELECT * FROM #TempEmployees
  WHERE EmployeeID = @EmployeeID;
END;
GO


-- Execute Procedure
EXEC ##GetTempEmployeeByID @EmployeeID = 2;
```

Explanation:
This procedure retrieves the details of an employee from the temporary table #TempEmployees by their EmployeeID. It can be executed by any session because it's a global temporary procedure.

---

5. Calculate Temporary Employee Total Salary

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #GetTempEmployeeTotalSalary
AS
BEGIN
   SELECT SUM(Salary) AS TotalSalary
  FROM #TempEmployees;
END;
```

```
GO
```

-- Execute Procedure

EXEC #GetTempEmployeeTotalSalary;

Explanation:
This procedure calculates the total salary of all employees stored in the temporary table #TempEmployees. It is only available during the current session.

---

6. Insert New Temporary Product Record

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #InsertTempProduct
    @ProductName NVARCHAR(50),
   @Price DECIMAL(10,2)
AS
BEGIN
    INSERT INTO #TempProducts (ProductName, Price)
   VALUES (@ProductName, @Price);
END;
GO
```

-- Execute Procedure

EXEC #InsertTempProduct @ProductName = 'Tablet', @Price = 300.00;

Explanation:
This local temporary procedure inserts a new product's details into a temporary table called #TempProducts.

---

7. Get Temporary Products Above a Certain Price

```
-- Create Global Temporary Procedure
CREATE PROCEDURE ##GetTempProductsAbovePrice
  @Price DECIMAL(10,2)
AS
BEGIN
  SELECT * FROM #TempProducts
  WHERE Price > @Price;
END;
GO

-- Execute Procedure
EXEC ##GetTempProductsAbovePrice @Price = 100.00;
```

Explanation:
This procedure retrieves all temporary products from the #TempProducts table where the price is greater than the specified amount. It is accessible globally.

---

8. Delete Temporary Product Record

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #DeleteTempProduct
  @ProductID INT
AS
BEGIN
  DELETE FROM #TempProducts
  WHERE ProductID = @ProductID;
END;
GO
```

```
-- Execute Procedure
EXEC #DeleteTempProduct @ProductID = 1;
```

Explanation:
This procedure deletes a product record from the temporary table #TempProducts using the provided ProductID.

---

9. Get All Temporary Products

```
-- Create Global Temporary Procedure
CREATE PROCEDURE ##GetAllTempProducts
AS
BEGIN
   SELECT * FROM #TempProducts;
END;
GO

-- Execute Procedure
EXEC ##GetAllTempProducts;
```

Explanation:
This procedure retrieves all records from the temporary table #TempProducts and can be used by any session.

---

10. Count Total Temporary Employees

```
-- Create Local Temporary Procedure
CREATE PROCEDURE #CountTotalTempEmployees
AS
BEGIN
   SELECT COUNT(*) AS TotalEmployees
```

```
   FROM #TempEmployees;

END;

GO
```

```
-- Execute Procedure

EXEC #CountTotalTempEmployees;
```

Explanation:
This procedure counts the total number of records in the #TempEmployees table and returns the result. It is only available to the session that created it.