SQL Server Stored Procedures

- ❖ SQL Server stored procedures are used to group one or more Transact-SQL statements into logical units. The stored procedure is stored as a named object in the SQL Server Database Server.
- ❖ When you call a stored procedure for the first time, SQL Server creates an execution plan and stores it in the cache. In the subsequent executions of the stored procedure, SQL Server reuses the plan to execute the stored procedure very fast with reliable performance.

Section 1

Getting started with SQL Server Stored Procedures

Basic Guide to Stored Procedures in SQL Server

A Stored Procedure is a group of SQL statements that are stored and executed on the server. You can pass parameters to stored procedures, get return values, and manage complex operations. Below is a step-by-step guide on how to create, execute, modify, and drop stored procedures.

1. Creating a Stored Procedure

The syntax for creating a basic stored procedure is:

```
CREATE PROCEDURE ProcedureName
AS
BEGIN
   -- SQL Statements go here
END
```

Example: Create a Stored Procedure
Let's create a simple stored procedure that selects all employees from the `Employees` table.

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

This stored procedure doesn't take any parameters; it simply returns all data from the `Employees` table.
2. Executing a Stored Procedure
To execute a stored procedure, you use the `EXEC` or `EXECUTE` command.

Example: Execute the Stored Procedure

```
EXEC GetAllEmployees;
```

This will run the `GetAllEmployees` procedure and return all rows from the `Employees` table.

3. Creating a Stored Procedure with Parameters

You can pass parameters to a stored procedure, either input parameters (to supply data) or output parameters (to retrieve data).

Example: Stored Procedure with Input Parameter

Here's an example of a stored procedure that retrieves employee details based on an employee's ID.

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

- `@EmployeeID` is an input parameter of type `INT`.

Example: Execute a Stored Procedure with a Parameter
To execute a stored procedure with a parameter, pass the value of the parameter using `EXEC`:

```
EXEC GetEmployeeById @EmployeeID = 1;
```

This will retrieve the details of the employee with an `EmployeeID` of 1.
4. Modifying a Stored Procedure

To modify an existing stored procedure, you use the `ALTER PROCEDURE` command. This allows you to change the logic or add new functionality without deleting the procedure.
Example: Modify the Stored Procedure

```
ALTER PROCEDURE GetEmployeeById
   @EmployeeID INT
AS
BEGIN
   SELECT EmployeeID, Name, Department FROM Employees
   WHERE EmployeeID = @EmployeeID;
END
```

In this modification, the stored procedure now returns only the `EmployeeID`, `Name`, and `Department` columns.
5. Dropping a Stored Procedure
If you no longer need a stored procedure, you can delete it using the `DROP PROCEDURE` command.

Example: Drop the Stored Procedure

```
DROP PROCEDURE GetAllEmployees;
```

This will remove the `GetAllEmployees` stored procedure from the database.

6. Stored Procedure with Output Parameters

You can also use output parameters to return values from a stored procedure. Here's an example.

Example: Stored Procedure with an Output Parameter

```
CREATE PROCEDURE GetEmployeeName
   @EmployeeID INT,
   @EmployeeName NVARCHAR(50) OUTPUT
AS
BEGIN
   SELECT @EmployeeName = Name FROM Employees WHERE EmployeeID = @EmployeeID;
END
```

- `@EmployeeName` is an output parameter that returns the employee's name.

Example: Execute a Stored Procedure with Output Parameters

To execute a stored procedure with output parameters:

```
DECLARE @Name NVARCHAR(50);
EXEC GetEmployeeName @EmployeeID = 1, @EmployeeName = @Name OUTPUT;
SELECT @Name AS EmployeeName;
```

This example declares a local variable `@Name`, passes it to the stored procedure, and captures the output in `@Name`.
7. Stored Procedures with Control Flow and Transactions

Stored procedures can also include control flow statements like `IF`, `WHILE`, and `BEGIN...END` blocks, as well as transaction management using `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK`.

Example: Stored Procedure with a Transaction
```
CREATE PROCEDURE UpdateEmployeeSalary
   @EmployeeID INT,
   @NewSalary DECIMAL(10, 2)
AS
BEGIN
   BEGIN TRANSACTION;
   BEGIN TRY
      UPDATE Employees SET Salary = @NewSalary WHERE EmployeeID = @EmployeeID;
      COMMIT;
   END TRY
   BEGIN CATCH
      ROLLBACK;
      THROW;
   END CATCH;
END
```

- This stored procedure updates the salary for an employee and ensures that if anything goes wrong, the changes are rolled back using transactions.

Summary:

- Creating a Stored Procedure: Use the `CREATE PROCEDURE` statement.
- Executing a Stored Procedure: Use the `EXEC` command.
- Modifying a Stored Procedure: Use the `ALTER PROCEDURE` statement.
- Dropping a Stored Procedure: Use the `DROP PROCEDURE` statement.
- Stored Procedures with Parameters: You can pass input and output parameters.
- Transactions and Control Flow: You can include complex logic and transaction handling in stored procedures.

This basic guide helps you understand the most common uses of stored procedures in SQL Server. Would you like to dive deeper into any specific part or see more complex examples?