Building Dynamic SQL In a Stored Procedure

John-ph, 9 Oct 2007 CPOL

4.49 (64 votes)

This article explains about building and executing a Dynamic SQL in a stored procedure.

Download source - 1.56 KB

Introduction

A dynamic SQL in a stored procedure is a single `Transact-SQL` statement or a set of statements stored in a variable and executed using a SQL command. There may be several methods of implementing this in SQL Server. This article will show you a good method of doing this. Before getting into a detailed explanation, let me tell "When to Use Dynamic SQL?" We can't definitely say that a Static SQL will meet all our programming needs. A Dynamic SQL is needed when we need to retrieve a set of records based on different search parameters. Say for example - An employee search screen or a general purpose report which needs to execute a different `SELECT` statement based on a different `WHERE` clause.

NOTE: Most importantly, the Dynamic SQL Queries in a variable are not compiled, parsed, checked for errors until they are executed.

sp_executesql Vs EXECUTE Command

A dynamically build Transact-SQL statements can be executed using `EXECUTE` Command or `sp_executesql` statement. Here, in this article in my examples, I'll be using `sp_executesql` which is more efficient, faster in execution and also supports parameter substitution. If we are using `EXECUTE` command to execute the SQL String, then all the parameters should be converted to character and made as a part of the Query before execution. But the `sp_executesql` statement provides a better way of implementing this. It allows us to substitute the parameter values for any parameter specified in the SQL String. Before getting into the actual example, let me differentiate these two commands with a simple example. Say - selecting a record from the employee table using the ID in the `WHERE` clause.

The basic syntax for using `EXECUTE` command:

```
EXECUTE(@SQLStatement)
```

The basic syntax for using `sp_executesql`:

```
sp_executesql [@SQLStatement],[@ParameterDefinitionList],
[@ParameterValueList]
```

Example 1.0

```
/* Using EXECUTE Command */
/* Build and Execute a Transact-SQL String with a single parameter
 value Using EXECUTE Command */

/* Variable Declaration */
DECLARE @EmpID AS SMALLINT
```

```
DECLARE @SQLQuery AS NVARCHAR(500)
/* set the parameter value */
SET @EmpID = 1001
/* Build Transact-SQL String with parameter value */
SET @SQLQuery = 'SELECT * FROM tblEmployees WHERE EmployeeID = ' +
CAST(@EmpID AS NVARCHAR(10))
/* Execute Transact-SQL String */
EXECUTE(@SQLQuery)
```

In the above example 1.0, there are two variables declared. The first variable `@EmpID` is used as a parameter to the SQL Query and second Variable `@SQLQuery` is used to build the SQL String. You can clearly see that the variable `@EmpID` is cast to a `NVarchar` type and made as a part of the SQL String. If you print the `@SQLQuery` string (`PRINT @SQLQuery`), you will get the actual SQL query as shown below:

```
SELECT * FROM tblEmployees WHERE EmployeeID = 1001
```

Finally, the above query is executed using the `EXECUTE` command.

Example 1.1

```
/* Using sp_executesql */
/* Build and Execute a Transact-SQL String with a single parameter
value Using sp_executesql Command */

/* Variable Declaration */
DECLARE @EmpID AS SMALLINT
DECLARE @SQLQuery AS NVARCHAR(500)
DECLARE @ParameterDefinition AS NVARCHAR(100)
/* set the parameter value */
SET @EmpID = 1001
/* Build Transact-SQL String by including the parameter */
SET @SQLQuery = 'SELECT * FROM tblEmployees WHERE EmployeeID = @EmpID'
/* Specify Parameter Format */
SET @ParameterDefinition =  '@EmpID SMALLINT'
/* Execute Transact-SQL String */
EXECUTE sp_executesql @SQLQuery, @ParameterDefinition, @EmpID
```

In the example 1.1, there are two variables declared. The variable `@EmpID` is used as a parameter to the SQL Query and second variable `@SQLQuery` is used to build the SQL String, the third variable `@ParameterDefinition` is used to specify the parameter format before executing the SQL string. If you print the `@SQLQuery` string (`PRINT @SQLQuery`), you will get the query as shown below:

```
SELECT * FROM tblEmployees WHERE EmployeeID = @EmpID
```

Here, in this example, you can clearly see the parameter `@EmpID` is included in the statement. Finally, `sp_executesql` takes the necessary information to do the parameter substitution and execute the dynamically built SQL string.

1. `@SQLQuery` --> contains the SQL statement
2. `@ParameterDefinition` --> contains the Parameter Definition
3. `@EmpID` --> contains the parameter value to be substituted to the parameter in the SQL statement.

NOTE: The parameters included in the Dynamic SQL string must have a corresponding entry in the Parameter Definition List and Parameter Value List.

Dynamic SQL in a Stored Procedure

This part of the article explains with a real-world example and sample procedure "How to Build and Execute a Dynamic SQL in a stored procedure?"

Example 2.0

Let us take a simple example - `Employee` Table with common fields such as `EmployeeID`, `Name`, `Department`, `Designation`, `JoiningDate`, `Salary` and `Description`. You can use the following Transact-SQL `CREATE TABLE` statement to create a `Employee` Table within your database.

```
/* Transact-Sql to create the table tblEmployees */
CREATE TABLE tblEmployees
(
    EmployeeID       SMALLINT IDENTITY(1001,1) NOT NULL,
    EmployeeName     NVARCHAR(100) NOT NULL,
    Department       NVARCHAR(50) NOT NULL,
    Designation      NVARCHAR(50) NOT NULL,
    JoiningDate      DATETIME NOT NULL,
    Salary           DECIMAL(10,2) NOT NULL,
    [Description]    NVARCHAR(1000) NULL
)
```

The following `INSERT` statements insert some sample records into the `tblEmployee` table:

```
/* Transact SQL to insert some sample records into tblEmployee table */
INSERT INTO tblEmployees
(EmployeeName, Department, Designation,
 JoiningDate, Salary, [Description])
VALUES
('John Smith', 'IT Research', 'Research Analyst',
 '02/08/2005', 23000.00, 'Analyst since 2005')

INSERT INTO tblEmployees
(EmployeeName, Department, Designation,
 JoiningDate, Salary, [Description])
VALUES
('John Micheal', 'IT Operations', 'Manager',
 '07/15/2007', 15000.00, NULL)

INSERT INTO tblEmployees
(EmployeeName, Department, Designation,
 JoiningDate, Salary, [Description])
VALUES
('Will Smith', 'IT Support', 'Manager',
 '05/20/2006', 13000.00, 'Joined last year as IT Support Manager')
```

We programmers may get an assignment to develop an `Employee` search screen or generate an `Employee` listing report which will search the database and return a result based on the search criteria. In this case, the search Interface should be flexible enough to search the database for all possible criteria. A user may require to search for the following details:

- Search for specific `Employee` Detail with the `Name`
- List of `Employee`s in a specific `Department`
- List of `Employee`s in a specific `Designation`
- List of `Employee`s joined the organization last year
- List of `Employee`s whose `Salary` >= some specific Amount
- Any of these conditions listed above or all of these

I have listed few possible conditions here. There could be many other possibilities also which completely depend on the user requirement. Here let us take these listed few possible criteria and write a single stored procedure that builds a Dynamic SQL which will serve our purpose in searching for the Details in the `Employee` Table. The following `CREATE PROCEDURE` Statement will create a stored procedure "`sp_EmployeeSelect`" with the necessary input parameters and variables to build the Dynamic SQL.

```
/* This stored procedure builds dynamic SQL and executes
using sp_executesql */
Create Procedure sp_EmployeeSelect
    /* Input Parameters */
    @EmployeeName NVarchar(100),
    @Department NVarchar(50),
    @Designation NVarchar(50),
    @StartDate DateTime,
    @EndDate DateTime,
    @Salary    Decimal(10,2)

AS
    Set NoCount ON
```

```
    /* Variable Declaration */
    Declare @SQLQuery AS NVarchar(4000)
    Declare @ParamDefinition AS NVarchar(2000)
    /* Build the Transact-SQL String with the input parameters */
    Set @SQLQuery = 'Select * From tblEmployees where (1=1) '
    /* check for the condition and build the WHERE clause accordingly */
    If @EmployeeName Is Not Null
         Set @SQLQuery = @SQLQuery + ' And (EmployeeName = @EmployeeName)'

    If @Department Is Not Null
         Set @SQLQuery = @SQLQuery + ' And (Department = @Department)'

    If @Designation Is Not Null
         Set @SQLQuery = @SQLQuery + ' And (Designation = @Designation)'

    If @Salary Is Not Null
         Set @SQLQuery = @SQLQuery + ' And (Salary >= @Salary)'

    If (@StartDate Is Not Null) AND (@EndDate Is Not Null)
         Set @SQLQuery = @SQLQuery + ' And (JoiningDate
         BETWEEN @StartDate AND @EndDate)'
    /* Specify Parameter Format for all input parameters included
     in the stmt */
    Set @ParamDefinition =      ' @EmployeeName NVarchar(100),
                @Department NVarchar(50),
                @Designation NVarchar(50),
                @StartDate DateTime,
                @EndDate DateTime,
                @Salary    Decimal(10,2)'
    /* Execute the Transact-SQL String with all parameter value's
       Using sp_executesql Command */
    Execute sp_Executesql     @SQLQuery,
                @ParamDefinition,
                @EmployeeName,
                @Department,
                @Designation,
                @StartDate,
                @EndDate,
                @Salary

    If @@ERROR <> 0 GoTo ErrorHandler
    Set NoCount OFF
    Return(0)

ErrorHandler:
    Return(@@ERROR)
GO
```

This sample stored procedure takes few parameter's as input and uses two variables to build and execute. `@SQLQuery` which is used to build the dynamic SQL-statement. `@ParamDefinition` which is used to define the Parameter's format. Whiling building the SQL string in each step, an `IF`-statement is used to check whether that inputted parameter is `Null` or not. If it is not `NULL`, then that parameter will be included in the SQL statement which basically adds a condition in the `WHERE` clause of the SQL statement. You can clearly see in the procedure that the variable `@ParamDefinition` contains all the parameter lists and finally `sp_Executesql` takes SQL-query, parameter list and the parameter values to executes a `SELECT` statement.

Let us consider some of the criteria listed above and see how this stored procedure works.

1. Search for specific `Employee` Detail with the name.

```
/* 1. Search for specific Employee Detail with the Name say 'John Smith'. */
EXEC sp_EmployeeSelect 'John Smith', NULL, NULL, NULL, NULL, NULL
```

   Executing the above statement will list the details of the `Employee` "`John Smith`".

2. List of `Employee`s in a specific `Department`, AND
3. List of `Employee`s in a specific `Designation`.

```
/* 2. List of Employees in a specific Department. AND
3. List of Employees in a specific Designation. */
/* Say Department = 'IT Operations'  AND  Designation = 'Manager'*/
EXEC sp_EmployeeSelect NULL, 'IT Operations', 'Manager', NULL, NULL, NULL
```

Executing the above statement will list the Details of Managers in the IT Operations Department.

Using Like Operator, IN Operator and Order By In Dynamic SQL

when we are building Dynamic SQL, there may be some instances where we need to use LIKE operator, IN operator and Order BY Clause. But the parameters used with these operators and Order By Clause doesn't work the way as they normally do for "=" and "Between" operator while using sp_executesql. Generally sp_executesql doesn't do a parameter substitution for order by clause and doing so causes a column-referencing problem. Straightly Using LIKE operator and IN operator causes syntax error, which cannot be rectified when we are including the parameter into the Dynamic SQL statement. This problem can be resolved by including the actual parameter value in the Dynamic SQL statement. Below are the examples that show how to use Like Operator, IN Operator and OrderBy clause while using sp_executesql.

Example 3.0 - Using LIKE Operator

Example 3.0 uses LIKE operator to select the list of Employees with the Name 'John'. Here in this example, the parameter is not included in the SQL statement, instead the actual value of the parameter is added to the SQL statement. So here, there is no need of parameter definition for executing the SQL string. The same applies to the other two examples shown below:

```
/* Variable Declaration */
DECLARE @EmpName AS NVARCHAR(50)
DECLARE @SQLQuery AS NVARCHAR(500)

/* Build and Execute a Transact-SQL String with a single parameter
value Using sp_executesql Command */
SET @EmpName = 'John'
SET @SQLQuery = 'SELECT * FROM tblEmployees
WHERE EmployeeName LIKE '''+ '%' + @EmpName + '%' + ''''
EXECUTE sp_executesql @SQLQuery
```

Example 3.1 - Using IN Operator

Example 3.1 uses IN operator to select the Employee details *(ID = 1001, 1003)*:

```
/* Variable Declaration */
DECLARE @EmpID AS NVARCHAR(50)
DECLARE @SQLQuery AS NVARCHAR(500)

/* Build and Execute a Transact-SQL String with a single
parameter value Using sp_executesql Command */
SET @EmpID = '1001,1003'
SET @SQLQuery = 'SELECT * FROM tblEmployees
WHERE EmployeeID IN(' + @EmpID + ')'
EXECUTE sp_executesql @SQLQuery
```

Example 3.2 - Using Order By Clause

Example 3.2 sorts the Employee records by "Department" column.

```
/* Variable Declaration */
DECLARE @OrderBy AS NVARCHAR(50)
DECLARE @SQLQuery AS NVARCHAR(500)

/* Build and Execute a Transact-SQL String with a single parameter
value Using sp_executesql Command */
SET @OrderBy = 'Department'
SET @SQLQuery = 'SELECT * FROM tblEmployees Order By ' + @OrderBy

EXECUTE sp_executesql @SQLQuery
```

Conclusion

In this article, I have explained with few examples "How to Build and Execute Dynamic SQL in stored procedures". Hope this article will help to understand and write Dynamic SQL in a good way.

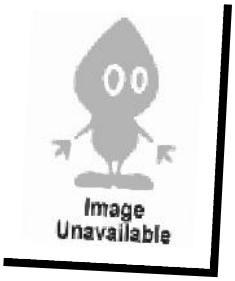
History

- 9th October, 2007: Initial post

License

Share

About the Author



John-ph
Technical Lead
India

You may also be interested in...

Comments and Discussions

31 messages have been posted for this article Visit **http://www.codeproject.com/Articles/20815/Building-Dynamic-SQL-In-a-Stored-Procedure** to post and view comments on this article, or click **here** to get a print view with messages.