SQL Studies

Writing Dynamic SQL (A how to)

Kenneth Fisher 6 years ago

A little while back I wrote Best practice recommendations for writing Dynamic SQL where I made a number of suggestions for good habits when writing dynamic SQL. Along the same lines, this is my methodology for writing Dynamic SQL. Again this has nothing to do with when it is a good idea to USE dynamic SQL.

As a demonstration I'm going to write a stored procedure that will search every "text" column in a given table for a given character value. I should warn you I'm going to use a cursor. I know, "How could I, cursors are horrible bad things". But please remember this is a demonstration of dynamic SQL. Running dynamic SQL in batch mode is impossible to the best of my knowledge so I need a loop. And the type of loop I'm using will be such an insignificant part of the over all process I don't think any performance issues caused by using a "CURSOR" (imagine ominous music here) are really going to be significant.

To start with I'm going to write a query to check if a value exists in specific column. The value will be passed in as a variable.

```
DECLARE @StringTest varchar(max)


SELECT COUNT(1)
FROM Person.Address
WHERE AddressLine1 LIKE @StringTest
```

Next I'm going to put in place the best practices I mentioned in the previous post. I'm also going to add a literal column to let me know which column I'm querying off of. This will be important when I'm querying off a large number of columns and want to know which one actually had returned non zero values in the result set.

```
SELECT 'AddressLine1' AS ColumnName, COUNT(1)
FROM [Person].[Address]
WHERE [AddressLine1] LIKE @StringTest;
```

Notice the brackets, semicolon, formatting etc. Schema is easy in this case since it is not dbo. Next we put single quotes around the code, making sure to double any single quotes inside the code. I also like to add a + CHAR(13) to the end of each line to maintain the formatting, although of course there are a number of end characters you can use (CHAR(10) for example).

```
'SELECT ''AddressLine1'' AS ColumnName, COUNT(1) ' + CHAR(13) +

'FROM [Person].[Address] ' + CHAR(13) +

'WHERE [AddressLine1] LIKE @StringTest; '
```

From here there are two options. If I was going to be running just the single line of code then I would declare a variable (@sql nvarchar(max) for example) and set it equal to my string. In this case I'm creating a series of commands so I'm going to put the word SELECT in front of it and the rest of my query after.

With a more complex query I'd write the basic query that pulls the "dynamic" information I need and then merge in the additional data. This particular query is pretty simple so I don't need this step, but here is the basic query anyway.

Basic query

```
SELECT *

FROM INFORMATION_SCHEMA.COLUMNS

WHERE DATA_TYPE IN ('char','nchar','varchar','nvarchar','text','ntext')
```

Dynamic string merged in

```
SELECT

        'SELECT ''AddressLine1'' AS ColumnName, COUNT(1) ' + CHAR(13) +

        'FROM [Person].[Address] ' + CHAR(13) +

        'WHERE [AddressLine1] LIKE @StringTest; '

FROM INFORMATION_SCHEMA.COLUMNS

WHERE DATA_TYPE IN ('char','nchar','varchar','nvarchar','text','ntext')
```

Now we replace anything that is going to become "dynamic" with '++'.

```
SELECT

        'SELECT '''++''' AS ColumnName, COUNT(1) ' + CHAR(13) +

        'FROM ['++'].['++'] ' + CHAR(13) +

        'WHERE ['++'] LIKE @StringTest; '

FROM INFORMATION_SCHEMA.COLUMNS

WHERE DATA_TYPE IN ('char','nchar','varchar','nvarchar','text','ntext')
```

And add the correct columns from my query between the ++s.

```
SELECT
        'SELECT '''+ COLUMN_NAME +''' AS ColumnName, COUNT(1) ' + CHAR(13) +
        'FROM ['+TABLE_SCHEMA+'].['+TABLE_NAME+'] ' + CHAR(13) +
        'WHERE ['+COLUMN_NAME+'] LIKE @StringTest; '
FROM INFORMATION_SCHEMA.COLUMNS
WHERE DATA_TYPE IN ('char','nchar','varchar','nvarchar','text','ntext')
```

I'm going to do this next bit in one big step since it isn't really part of the dynamic SQL process. I'm going to set up the code for the procedure, the parameters, the cursor loop etc. Note: I'm not running my dynamic SQL yet.

```
CREATE PROCEDURE usp_StringTest (@SchemaName nvarchar(255),
                                                                @TableName nvarchar(255),
                                                                @StringTest nvarchar(max))
        AS

DECLARE SQL_Cur CURSOR
READ_ONLY
FOR SELECT
        N'SELECT '''+ COLUMN_NAME +''' AS ColumnName, COUNT(1) ' + CHAR(13) +
        'FROM ['+TABLE_SCHEMA+'].['+TABLE_NAME+'] ' + CHAR(13) +
        'WHERE ['+COLUMN_NAME+'] LIKE @StringTest; ' AS SQL_String
FROM INFORMATION_SCHEMA.COLUMNS
WHERE DATA_TYPE IN ('char','nchar','varchar','nvarchar','text','ntext')
  AND TABLE_SCHEMA = @SchemaName
  AND TABLE_NAME = @TableName

DECLARE @SQL_String nvarchar(MAX)
OPEN SQL_Cur

FETCH NEXT FROM SQL_Cur INTO @SQL_String
WHILE (@@fetch_status <> -1)
BEGIN
        IF (@@fetch_status <> -2)
        BEGIN
                PRINT @SQL_String
        END
        FETCH NEXT FROM SQL_Cur INTO @SQL_String
```

```
END


CLOSE SQL_Cur

DEALLOCATE SQL_Cur

GO
```

A few things of note at this point. First I'm printing my SQL string not executing it. This is so I can debug at this step rather than when I'm trying to execute the dynamic SQL. And in fact when I was writing my example I forgot to set the @SQL_String variable to varchar(MAX) and had it set to varchar(40) from the original cursor template. I noticed this because all of my output strings were chopped off.

Second, if you run the stored procedure as it stands you will also notice that all of the output strings are formatted queries. I'm a very strong believer in formatting your code (even or especially the dynamic code) as it makes it MUCH easier to read.

Third, all of my parameters and variables are nvarchar. And I have an N in front of my dynamic string to the string into Unicode. The explanation why is farther down.

And fourth, if you look at the output strings you will see that I still have a variable. I could have added it into my execution string as a literal but that would leave me open to SQL Injection attacks. If the input string looked like this ''''; select * from sys.tables; –' then I might be in some trouble. So you might ask how am I going to have a variable in my execution string? The answer is sp_execute_sql, which requires Unicode strings as parameters. And that explains the third point.

Here is the modified BEGIN – END block using sp_executesql.

```
        BEGIN

                --PRINT @SQL_String

                EXEC sp_executeSql @SQL_String, N'@StringTest nvarchar(max)', @StringTest

        END
```

I'm not going to go into great detail on sp_executeSql since you can easily look it up in BOL. The big thing about sp_executesql is that it lets you combine parameters with dynamic SQL. This takes a method of making reusable code (dynamic SQL) and combines it with another method of making reusable code (parameters). It can be a bit more complicated but once you have it down it lets you do some amazing stuff. Also as I stated above this is one of the easier methods (at least to me) to avoid SQL Injection.

So here is the final code using dynamic SQL with parameters.

```
CREATE PROCEDURE usp_StringTest (@SchemaName nvarchar(255),
                                                                 @TableName nvarchar(255),
                                                                 @StringTest nvarchar(max))
        AS

DECLARE SQL_Cur CURSOR
READ_ONLY
FOR SELECT
        N'SELECT '''+ COLUMN_NAME +''' AS ColumnName, COUNT(1) ' + CHAR(13) +
        'FROM ['+TABLE_SCHEMA+'].['+TABLE_NAME+'] ' + CHAR(13) +
        'WHERE ['+COLUMN_NAME+'] LIKE @StringTest; ' AS SQL_String
FROM INFORMATION_SCHEMA.COLUMNS
WHERE DATA_TYPE IN ('char','nchar','varchar','nvarchar','text','ntext')
  AND TABLE_SCHEMA = @SchemaName
  AND TABLE_NAME = @TableName

DECLARE @SQL_String nvarchar(MAX)
OPEN SQL_Cur

FETCH NEXT FROM SQL_Cur INTO @SQL_String
WHILE (@@fetch_status <> -1)
BEGIN
        IF (@@fetch_status <> -2)
        BEGIN
                --PRINT @SQL_String
                EXEC sp_executeSql @SQL_String, N'@StringTest nvarchar(max)', @StringTest
        END
        FETCH NEXT FROM SQL_Cur INTO @SQL_String
END

CLOSE SQL_Cur
DEALLOCATE SQL_Cur
GO
```

I want you to imagine trying to write the dynamic portion of this code straight out and compare it to this method. Personally I find this much easier to work with and far less error prone. I do want to point out that this is MY method of writing dynamic SQL. I seriously doubt it's the only one, and it may not even be the best. But if you are having a hard time with dynamic SQL I suggest giving it a try and seeing how it works for you.

Categories: Dynamic SQL, Microsoft SQL Server, SQLServerPedia Syndication, T-SQL

Tags: code language, dynamic sql, language sql, microsoft sql server, sql statements, T-SQL

Leave a Comment

SQL Studies

Blog at WordPress.com. Back to top