Two Handy Techniques for Creating Dynamic SQL in Stored Procedures

DaveDbViewSharp

14 Feb 2012 CPOL

Describes two techniques to use when generating dynamic SQL in stored procedures and provides an example that demonstrates what happens when they are combined

Download source - 2.05 KB

Introduction

I've been a visitor to CodeProject for many years now and have benefited greatly from many excellent articles. I've always wanted to give something back but have never quite been able to find a good subject. Today, however, I combined two neat techniques for creating dynamic SQL within a stored procedure and the end result so surprised me in its brevity that I am inspired to share it.

Background

While it's often frowned on, there are times when only dynamic SQL will do in a stored procedure. For non-trivial cases, this can lead to some very messy code, which is tricky to follow and debug. I'd like to present an approach that almost always results in code that is shorter, clearer to understand and easier to extend.

The Normal Approach

In developing a dynamic query in a stored procedure, the most normal way would be to start by declaring a large `string` variable and gradually build the query by adding and concatenating a mixture of literal text and variables. Example:

Hide Copy Code

```
declare @query varchar(max)
set @query = 'select top '+str(@top)+' * from '+@table
if @sort_key is not null
    set @query = @query + ' order by '+@sort_key
print @query
exec (@query)
```

This simple example does not seem too bad, but even here the mix of literal text with SQL functions and variables reduces the comprehensibility of the query. Furthermore building a query this way tends to produce a script string with little regard for the format of the SQL it contains. In the example above, while the code looks simple, the final SQL contains no line breaks. As the dynamic queries become more complicated, the lack of formatting makes debugging them that much more irksome.

The Template Approach

I prefer a template approach. I've used templates in many programming environments and almost always enjoyed the experience. With this technique, the starting point is to write the final query in a template string, inserting a tag in the places where the variables should go. Generate the query to execute by replacing the tags with the variable values in the template. Example:

Hide Copy Code

```
declare @query_template varchar(max)
declare @query varchar(max)

set @query_template = '
select top %top *
  from %table
  %sort'

set @sort = case when @sort_key is null
                 then ''
                 else 'order by '+@sort_key end

set @query = replace(@query_template, '%top', @top)
set @query = replace(@query,    '%table', @table)
set @query = replace(@query,    '%sort',  @sort)

print @query
```

You can see immediately that there is a query to select the top some number of rows from some table in some order. The tag names chosen leave make it easy to figure out how the query will appear when they are replaced with parameter values. In this simple example, the extra code may not seem worthwhile, but in more complicated examples, the benefits become apparent in cleaner code and better separation of concerns. Templating is my first handy technique.

Building a String by Querying a Table

My second technique is to use a query on a table as a means of generating a single `string` containing data from each row in the table. You may have come across this used to generate a comma-separated `string`. It is brilliant trick (not invented by me: Google for "SQL Server coalesce comma" to find references) as it is almost impossible to figure it out from studying the SQL language. For anyone unfamiliar, here's a quick recap.

The canonical problem is to obtain a single `string` containing the data from a single column from all the rows of a table. So if your table is:

Hide Copy Code

```
id    name
1     John
2     James
3     Jason
```

The task is to obtain the `string` "`John, James, Jason`". There are two problems to solve; firstly concatenating the names; and secondly punctuating them correctly. This is the solution:

Hide Copy Code

```
declare @names varchar(max)
select @names = coalesce(@names+', ','') + name
  from names
 order by id
print @names
```

As each row is processed, the `string` is modified by adding the comma and the current name. On the first row, the variable `@name` is `null` so the concatenation of `', '` to `null` is also `null`, which causes the `coalesce` function to select instead the empty `string`. If you replaced the top row with `select @names = coalesce(@names+', ','The names are: ') + name` you would get "`The names are: John, James, Jason`".

The big advantage of this is that it may enable you to avoid using a loop structure in your stored procedure code. Cursor syntax in SQL is truly ugly and verbose. Any opportunity to replace it with a more eloquent alternative should be considered.

Working Together

This section presents an example that combines the two techniques. Suppose you are asked to report what percentage of values are `null` for each field in every table in a database. Here's a solution. If you select `count(pkfield)` where `pkfield` is a primary field, then this will count all of the records in the table. If you count any other field, it only counts rows with non-`null` data. The difference between the two figures is the count of `null` values for that field and with a little more maths this can be used to calculate the percentage. So a rough impression of the query is:

Hide Copy Code

```
select count(ID),
       count(ID)-count(name) as name_empty,
       (count(ID)-count(name))/count(ID) as name_empty_pc,
       count(ID)-count(addr1) as addr1_empty,
       (count(ID)-count(addr1))/count(ID) as addr1_empty_pc
       -- repeat for all other fields
  from Customers
```

This needs to be converted into a template `string` so that it will work for any table. The first try is:

Hide Copy Code

```
select count(%key),
       count(%key)-count(%field) as %field_empty,
       (count(%key)-count(%field))/count(%key) as %field_empty_pc
  from %table
```

This template won't work as it will only fit a single field. What is needed is an inner, or field, template to apply to each field with the results concatenated into a single `string`. This `string` will then replace a tag in the outer, or main, template. So break the original template into a field template:

Hide Copy Code

```
count(%key)-count(%field) as %field_empty,
(count(%key)-count(%field))/count(%key) as %field_empty_pc
```

and a main template:

Hide Copy Code

```
select count(%key), %fields
  from %table
```

Now the query will be constructed in two stages. In the first, each field will be applied in turn to the field template and these will all be concatenated to a single `string`. Then this `string` will replace the `%fields` tag in the main template and all the other tags will be replaced with the correct parameter values.

You should now have sufficient background to follow what is happening in the code below. I've made a function that returns a query for a single table. This can then be used to build a single dynamic script to query every table.

Hide Shrink ▲ Copy Code

```
create function dbo.ScriptCountNulls(@table varchar(120), @key_field varchar(120))
    returns varchar(max)
as
begin
    declare @template varchar(max),         -- main template
            @field_template varchar(max),   -- template for each field
            @sql varchar(max),              -- to hold final script with all template
                                            -- tags replaced
            @fields varchar(max)            -- to hold all field sql with tags replaced

    -- the main template is somewhat simple. All the interesting work is with the fields
    -- %fields is a placemarker for the string generated from the field query
    set @template = '
    select count(%key) as [%table_rowcount], %fields
      from %table'

    -- the field template generates 2 columns for each field:
    -- the empty count and the percentage
    set @field_template = '
            count(%key)-count(%field) as %field_empty,
            case when count(%key) > 0 then _
        cast(100.0*(count(%key)-count(%field))/count(%key) as decimal(6,2))
                 else 0 end as %field_empty_pc'

    -- build the field string replacing the tag in the template with the field name from
    -- the current row and then appending it to the field string.
    select @fields = coalesce(@fields+', ','') +
                     replace(@field_template, '%field', name)
     from sys.columns
    where object_id = object_id(@table)
 order by column_id

    -- Assemble the final template
    -- Replace the fields template first as the value itself contains tags
    set @sql = replace(@template, '%fields', @fields)
    set @sql = replace(@sql, '%table', @table)
    set @sql = replace(@sql, '%key', @key_field)

    return @sql -- this line for debugging
end
```

And to test:

Hide Copy Code

```
create table t1 (id int primary key, n25 int, n50 int, n75 int, n100 int)
go
insert into t1 (id, n25, n50, n75, n100)
select 1, NULL, NULL, NULL, NULL union
select 2,    1, NULL, NULL, NULL union
select 3,    1,    1, NULL, NULL union
select 4,    1,    1,    1, NULL
go
declare @script varchar(max)
set @script = dbo.ScriptCountNulls('t1', 'id')
exec (@script)go
drop table t1
go
```

Resulting in:

Hide Copy Code

```
t1_rowcount id_empty  id_empty_pc n25_empty  n25_empty_pc n50_empty
----------- --------- ----------- ---------- ------------ ---------
4           0         .00         1          25.00        2

n50_empty_pc n75_empty   n75_empty_pc n100_empty  n100_empty_pc
```

```
------------ ----------- ------------ ----------- -------------
50.00        3           75.00        4           100.00
```

In order to complete your assignment, you would need to apply this function to all the tables in the database. This can also be made generic.

Hide Copy Code

```
declare @script varchar(max)

select @script = coalesce(@script, '')+dbo.ScriptCountNulls(o.name, c.name)
  from sys.objects o
  join (select object_id, min(column_id) as keyid
          from sys.columns
         where is_nullable = 0
      group by object_id) sq on sq.object_id = o.object_id
  join sys.columns c on c.object_id = o.object_id and c.column_id = keyid
 where type = 'U'

exec (@script)
```

The script assumes that the first non-nullable field encountered in the field list for each table is the primary key. If not, it does not matter as any non-nullable field will provide a full row count of a table. Note the use again of the `string` concatenation technique. Running this script produces a single result row for each table in the database. The name of the table is included in the first column name of each result row.

Caveats

1. Always use `varchar(max)` (or `nvarchar(max)`) to hold scripts. In the days before SQL Server 2005, the `varchar` limit was 8000 characters. When a script overflows a `string` it is truncated without generating an error, that is until you attempt to `exec` the truncated script. It is not worth putting a time bomb like that in code unless you have to. `Varchar(max)` will avoid that.
2. If you try to replace a tag value with a parameter holding `NULL`, then this will make the entire script `string NULL`. This is quite an unexpected consequence which makes it difficult to diagnose the first time you encounter it. Below is a short scrap to illustrate this:

Hide Copy Code

```
declare @script varchar(max),
        @template varchar(max),
        @data varchar(20)

set @template = 'Data goes here [ %t ]'
print '@data is NULL'
set @script = replace(@template, '%t', @data)
print @script

set @data = 'This is the data'
print '@data is not null'
set @script = replace(@template, '%t', @data)
print @script
print '-----------------------------------'
```

Generates output:

Hide Copy Code

```
@data is NULL

@data is not null
Data goes here [ This is the data ]
-----------------------------------
```

If you need to replace a tag with the `string` "`NULL`" then use the `ISNULL()` function.

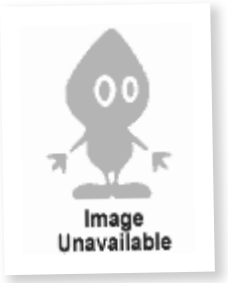
Conclusion

I hope this procedure serves to demonstrate the potential of each technique. Used separately, they can produce some elegant code, but used together I think they are dynamite.

License

Share

About the Author



DaveDbViewSharp

United Kingdom

Follow this Member

No Biography provided

Comments and Discussions