How Dynamic SQL Can Be Static SQL

Introduction

Here we'll look at some ways we can get the result set of a static SQL query to change based on variables we assign values to. These techniques will help you avoid using dynamic SQL while enabling your result set to change as you send in different values to different variables. Some of these queries may not be as optimized as they could be if you simply created one for each variable or change you need. They do avoid the security problems that you will encounter with dynamic SQL and avoid creating multiple stored procedures. Some of what I show here, such as using the CASE function after the ON keyword of a JOIN, may not make sense to use. I include it here to demonstrate the flexibility of static SQL.

Most of the flexibility static SQL has is due to the use of the CASE function. I do not plan to cover every possible way to use the CASE function as many of these ways are covered in the following articles by Neil Boyle: **Case Statement Tricks and Complex updates using the Case statement.** These two articles also demonstrate ways to modify the results of a query without using dynamic SQL.

Throughout this article I will use the Northwind database for all sample code.

Example 1 – Where Clause

First lets take a look at how we can use the CASE function to limit the result set with the WHERE clause. Here is the code we will use:

```
DECLARE @column varchar(10), @value varchar(20)

SET @Column = 'Title'
SET @value = '%vice%'

SELECT *
FROM employees
WHERE CASE @column WHEN 'Last' THEN LastName
                   WHEN 'First' THEN FirstName
                   WHEN 'Title' THEN Title
                   ELSE @value
                   END LIKE @value
```

After executing this in Query analyzer you can change the column name to First and use a different value, say the letter a. Now change the column name to Last. Please note that the else uses the variable name such that if you use a column name that is not used in any WHEN THEN section of CASE you will basically be saying WHERE @value LIKE @value which essentially negates the WHERE clause and gives you all the rows in the employees table.

Example 2 – Select Clause

Now let us look at the SELECT clause. This too can be customized to display only the columns we want to see based on variable values. You can copy and past the following code into Query Analyzer and execute it against the Northwind database:

```
DECLARE @column varchar(10)

SET @Column = 'title'
```

```
SELECT EmployeeID,
CASE @column
        WHEN 'Name' THEN LastName
        WHEN 'Title' THEN Title
        ELSE LastName
        END AS Column1,
CASE @column
        WHEN 'Name' THEN FirstName
        WHEN 'Title' THEN LastName
        ELSE CAST(BirthDate as varchar(20))
        END AS Column2,
CASE @column
        WHEN 'Title' THEN CAST(HireDate as varchar(20))
        ELSE ''
        END AS Column3
FROM employees
```

Change the value of @Column to be name and then change it to birthdate. Here the use of birthdate or any other value not used in the WHEN THEN sections will cause the column or value in the ELSE clause to be used.

You will notice that both datetime columns were cast as varchar. For Column2 it was needed to avoid SQL Server trying to convert the other columns to a datetime data type. For Column3 it is not required, however without it you would get a weird date, usually in the year 1900, when the ELSE clause is used.

One thing we can't do here is change the column alias based on the value of @Column. When using another application to execute the query this won't be a problem as you can designate your own titles for each column wherever you display that column and since you will know what value is sent in for @Column you will be able to use better labels than Column1, Column2, etc.

Example 3 – Order By Clause

Here is an example of how to design an ORDER BY clause that changes depending on the values of two variables (@OrderBy and @Sequence):

```
DECLARE @OrderBy varchar(10), @Sequence varchar(4)

SET @OrderBy = 'LastName'
SET @Sequence = 'DESC'

SELECT *
FROM employees
ORDER BY
CASE @Sequence
WHEN 'ASC' THEN CASE @OrderBy
                WHEN 'LastName' THEN LastName
                WHEN 'Title' THEN Title END
END ASC,
CASE @Sequence
WHEN 'DESC' THEN CASE @OrderBy
                 WHEN 'LastName' THEN LastName
                 WHEN 'Title' THEN Title END
END DESC
```

For this example, only one of the two CASE statements will actually influence the order of the result set. This could just as easily have been made so that both will run and have one order by last name and the other by first name. Providing a value not included in one of the two CASE statements will cause none of them to be used.

Example 4 – Group By and Having

The same kind of flexibility also applies to group by and having. We can use the CASE function in the SELECT clause as well as in the GROUP BY clause and the HAVING clause to return very different result sets depending on what values we put into variables. Here is sample SQL that demonstrates this:

```
DECLARE @column varchar(10), @ActivateCount bit, @Count int

SET @Column = 'name'
SET @ActivateCount = 1
SET @Count = 4

SELECT
CASE @column
        WHEN 'Name' THEN 'Number of Employees'
        WHEN 'Title' THEN 'Number of Titles'
        END AS Type,
CASE @column
        WHEN 'Name' THEN COUNT(employeeID)
        WHEN 'Title' THEN COUNT(Title)
        END AS QTY,
CASE @column
        WHEN 'Name' THEN Country
        WHEN 'Title' THEN Title
        END AS GroupBy
FROM employees
GROUP BY CASE @column
        WHEN 'Name' THEN Country
        WHEN 'Title' THEN Title
        END
HAVING CASE @column + LTRIM(STR(@ActivateCount))
        WHEN 'Name1' THEN COUNT(employeeID)
        WHEN 'Title1' THEN COUNT(Title)
        ELSE @Count + 1
        END > @Count
```

You'll notice that whenever the column Country is included in the SELECT list the same column is used in the GROUP BY clause. The same is true for the Title column. The HAVING clause is used here to eliminate counts that are below a specified threshold.

To disable the HAVING clause simply change the value of @ActivateCount to 0. You'll see that the result set has another row now. Next, set @Column = 'title' and see how the result set changes. Now reactivate the HAVING clause and see how the result set is reduced to 1 record.

Example 5 – Joins

Now we are going to look at how to use a CASE function after the keyword ON for a join. As I mentioned at the beginning of the article it is possible that you will never find a reason to use the CASE function in a JOIN. However, since you never know what you may need to do, here is a hypothetical situation using the Northwind database and a temp table.

In this case we'll assume we have a need to see order data linked with the employees that are assigned to those orders. In addition we have a region supervisor who is in charge of certain employees and certain orders based on city. This is where the temp table comes in. It contains employee ID's linked to the cities they supervise. The results we want should always include the name of a supervisor. Sometimes it will be the supervisor of the order and sometimes it will be the supervisor of the employee. This is where the CASE function can be used to make a JOIN give us the desired result set. Here is the code you'll need to run in Query Analyzer:

```
CREATE TABLE #RegionSupervisors (City varchar(20), employeeID int)

INSERT INTO #RegionSupervisors (City, EmployeeID)
SELECT 'Bern', 1        UNION SELECT 'Genève', 1  UNION SELECT 'Köln', 1       UNION
```

```
SELECT 'Albuquerque', 5 UNION SELECT 'Seattle', 5 UNION SELECT 'Redmond', 5    UNION SELECT 'Kirkland',
5 UNION
SELECT 'London', 4      UNION SELECT 'Cowes', 4   UNION SELECT 'Colchester', 4

DECLARE @supervisors varchar(10)

SET @Supervisors = 'employee'

SELECT CustomerID, OrderDate, ShipCity, e.LastName AS [Assigned To],e.City AS [Employee Home Office],
CASE @Supervisors WHEN 'employee' THEN 'employee supervisor: ' + se.LastName
                  WHEN 'order' THEN 'order supervisor: ' + se.LastName END AS [Region Supervisor]
FROM Orders o
INNER JOIN Employees e ON o.EmployeeID = e.EmployeeID
INNER JOIN #RegionSupervisors s ON CASE @Supervisors
                                   WHEN 'employee' THEN e.City
                                   WHEN 'order' THEN o.ShipCity END = s.City
INNER JOIN Employees se ON se.EmployeeID = s.EmployeeID
```

As you can see in the above code which supervisor shows in the result set depends on what value is assigned to @supervisors. Change the value from ‘employee’ to ‘order’, highlight the code from the DECLARE down and execute. You’ll notice the results for Region Supervisor changes.

Conclusions

The result set of static SQL can be modified based on variables you set before executing a query. All the techniques shown here can be combined if needed to allow great flexibility in how you see data. I believe the CASE function can be used any place a column can be used. All examples are solutions that avoid dynamic SQL and its pitfalls.