**Adding a Query Governor to Dynamic-Search Stored Procedures**

**By** [**Dwain Camps**](http://www.sqlservercentral.com/Authors/Articles/Dwain_Camps/1444841/)**, 2014/12/15**

**FROM:** [**http://www.sqlservercentral.com/articles/Dynamic+Search/118347/**](http://www.sqlservercentral.com/articles/Dynamic+Search/118347/)

Sometime back I wrote a piece that SQLServerCentral kindly published called “[How to Design, Build and Test a Dynamic Search Stored Procedure](http://www.sqlservercentral.com/articles/SQL+catch+all+query/103529/).”  While I believe that the technical advice given in that article is sound, there is a case where it could use a small improvement.

That article focused on how to build up highly-efficient, dynamic SQL to retrieve a row set from a table, based on filtering criteria passed into the stored procedure.  Presumably this filtering criteria is the result of an application’s user visiting a form and entering the filtering criteria from a set of form fields corresponding to the filtering criteria expected by the stored procedure.

What the technical aspects lacked was a practical focus on a general user’s behavior.  So let’s consider that for a moment.  When a user comes to a search form he is offered an array of filtering criteria to select his choice of rows from the table for display in the application.  Perhaps a bewildering array of choices.  The user must consciously think about what makes his need important and what filtering criteria will get him there the fastest.  So the user pauses to contemplate these deep questions.

Most people in the Information Technology industry have heard the term “dumb user” before.  Normally we try to avoid using it when we happen to be around one of the potential users of the system for fear of the political repercussions.  But the term persists and has been around as long as I can remember.  It originated from the concept that each form in an application needs to be coded around the principle of the dumbest possible user operating that form.  The idea was that the application should be the shepherd, leading its flock (the user community) to a desired outcome, this being that expected results should be delivered quickly and efficiently.

Let’s go back now to our intensely dumb user who has our search form displayed on his monitor and is now pondering his choices.  What makes the record this person is seeking unique?  How can it be found the quickest?  Is he even sure how to limit the choice?  How is a dumb user expected to react?

Why not hit the query button with no filtering criteria present?  His lizard-like brain comes to the conclusion that the machine is probably not as capable of filtering the records to just the one he would like to see, and knows that he’s the dominant life form on this planet so probably can do a better job of filtering the results by eyeball.

That’s fine when there’s just a few rows in the table, but what if there are tens of millions?  Our poor web application just retrieved all of those rows, and hopefully was at least smart enough to display a limited set (called a page) at a time.  But somewhere all of those tens of millions of cached records exist, pining away for our lizard to hit the “next page” key and show another tiny slice of the cached results.

Enter the Query Governor, better known as a way to limit rows retrieved for an unfiltered query and still be most likely to retrieve what the user needs.

**Some Sample Data and a Simplified Dynamic Search Stored Procedure**

Since I’m big into code reuse, let’s reuse some of the code from that article to construct our sample data.

CREATE TABLE dbo.Shipments

(

Consignment\_No VARCHAR(30) NOT NULL

,ReferenceNo VARCHAR(30) NULL

,CustID VARCHAR(20) NULL

,Total\_Pkgs INT NOT NULL

,Last\_Status VARCHAR(12) NULL

,Last\_Status\_DT DATETIME NULL

,PRIMARY KEY CLUSTERED (Consignment\_No)

);

INSERT INTO dbo.Shipments (Consignment\_No, ReferenceNo, CustID, Total\_Pkgs,Last\_Status\_DT)

VALUES ('8033701','PO 12345','ACME',1, '2013-10-23 20:12:00.000')

,('0152037',NULL,'ACME',1, '2013-10-28 13:11:00.000')

,('4292860',NULL,'ATLAS',1, '2013-10-25 08:49:00.000')

,('0806473','INV 43523','ATLAS',1, '2013-10-24 10:54:00.000')

,('6289811',NULL,'APPLE',1, '2013-10-27 14:45:00.000')

,('0642191','SO 1111','APPLE',1, '2013-10-29 11:37:00.000')

,('8363496',NULL,NULL,1, '2013-10-22 02:44:00.000')

,('7271931',NULL,NULL,1, '2013-10-29 00:36:00.000');

GO

--DROP PROCEDURE dbo.Shipment\_Tracking;

--DROP PROCEDURE dbo.Shipment\_Tracking\_QG1;

--DROP PROCEDURE dbo.Shipment\_Tracking\_QG2;

--DROP TABLE dbo.QueryGovernor;

--DROP TABLE dbo.Shipments;

The DROPs are provided to ultimately clean out of your sandbox all of the objects we’ll create throughout this article.

We’ll use a trimmed down version of the dynamic search stored procedure from the article, showing just some of the criteria for filtering.

CREATE PROCEDURE [dbo].[Shipment\_Tracking]

(

@StartDT DATETIME = NULL

,@EndDT DATETIME = NULL

,@Consignment\_No VARCHAR(30) = NULL

,@CustID VARCHAR(20) = NULL

,@debug TINYINT = 0 -- 1=Display debug results

)

AS BEGIN

SET NOCOUNT ON;

-- Truncate time component from start date

DECLARE @StartDate DATETIME = DATEADD(day, 0, DATEDIFF(day, 0, @StartDT))

-- Truncate time component from end date and add 1

,@EndDate DATETIME = DATEADD(day, 1, DATEDIFF(day, 0, @EndDT))

,@SQL NVARCHAR(MAX)

,@SQLParms NVARCHAR(MAX) = N'

@StartDT DATETIME

,@EndDT DATETIME

,@Consignment\_No VARCHAR(30)

,@CustID VARCHAR(20)';

-- If @EndDate is NULL, then use @StartDT + 1

SELECT @EndDate = ISNULL(@EndDate, DATEADD(day, 1, DATEDIFF(day, 0, @StartDT)))

,@Consignment\_No = NULLIF(@Consignment\_No, '')

,@CustID = NULLIF(@CustID, '')

,@debug = CASE WHEN @debug IS NULL THEN 0

WHEN @debug IN (0,1) THEN @debug

ELSE 0 END;

IF @debug = 1

BEGIN

PRINT CONVERT(VARCHAR(20), @StartDate, 120);

PRINT CONVERT(VARCHAR(20), @EndDate, 120);

END

SELECT @SQL = N'

SELECT a.Consignment\_No, a.ReferenceNo, a.CustID, a.Total\_Pkgs

,a.Last\_Status, a.Last\_Status\_DT

FROM dbo.Shipments a' + CHAR(10) +

CASE WHEN @StartDate IS NOT NULL

THEN N' AND a.Last\_Status\_DT >= @StartDT ' + CHAR(10) +

N' AND a.Last\_Status\_DT < @EndDT ' + CHAR(10)

ELSE N'' END +

CASE WHEN @Consignment\_No IS NOT NULL

THEN N' AND a.Consignment\_No = @Consignment\_No ' + CHAR(10)

ELSE N'' END +

CASE WHEN @CustID IS NOT NULL

THEN N' AND a.CustID = @CustID ' + CHAR(10)

ELSE N'' END + CHAR(10) +

N'ORDER BY Consignment\_No';

IF @debug = 1

PRINT @SQL;

EXEC sp\_executesql @SQL

,@SQLParms

,@StartDT = @StartDate

,@EndDT = @EndDate

,@Consignment\_No = @Consignment\_No

,@CustID = @CustID;

END

If we execute the following, we find that both queries return the same results.  First the query:

SELECT \*

FROM dbo.Shipments

ORDER BY Consignment\_No;

EXEC dbo.Shipment\_Tracking

@StartDT = NULL

,@EndDT = NULL

,@Consignment\_No = NULL

,@CustID = NULL

,@debug = 0;

And now the results:

Consignment\_No ReferenceNo CustID Total\_Pkgs Last\_Status Last\_Status\_DT

0152037 NULL ACME 1 MDE 2013-10-28 13:11:00.000

0642191 SO 1111 APPLE 1 MDE 2013-10-29 11:37:00.000

0806473 INV 43523 ATLAS 1 MDE 2013-10-24 10:54:00.000

4292860 NULL ATLAS 1 POD 2013-10-25 08:49:00.000

6289811 NULL APPLE 1 POD 2013-10-27 14:45:00.000

7271931 NULL NULL 1 WGT 2013-10-29 00:36:00.000

8033701 PO 12345 ACME 1 POD 2013-10-23 20:12:00.000

8363496 NULL NULL 1 SIP 2013-10-22 02:44:00.000

**The Most Common Query Governor**

When most T-SQL folks want to limit the returned rows, the normal thought process is to use the TOP clause of the SELECT statement.  So the idea then is to implement something like “if the user has specified no filtering criteria, return only the top n rows.”

You must also think about, TOP n based on what?  The clever developer will think “how about returning the most current information?”  Indeed, that is most likely to satisfy the need of the lizard-brain, that prefers tactile skills (the execute query button push) over cerebral ones (like thinking about the filtering criteria).

Let’s modify our SP to wrap a TOP clause that returns the TOP 5 rows that are the most current around our base query.

CREATE PROCEDURE dbo.Shipment\_Tracking\_QG1

(

@StartDT DATETIME = NULL

,@EndDT DATETIME = NULL

,@Consignment\_No VARCHAR(30) = NULL

,@CustID VARCHAR(20) = NULL

,@debug TINYINT = 0 -- 1=Display debug results

)

AS BEGIN

SET NOCOUNT ON;

-- Truncate time component from start date

DECLARE @StartDate DATETIME = DATEADD(day, 0, DATEDIFF(day, 0, @StartDT))

-- Truncate time component from end date and add 1

,@EndDate DATETIME = DATEADD(day, 1, DATEDIFF(day, 0, @EndDT))

,@SQL NVARCHAR(MAX) = N''

,@SQLParms NVARCHAR(MAX) = N'

@StartDT DATETIME

,@EndDT DATETIME

,@Consignment\_No VARCHAR(30)

,@CustID VARCHAR(20)';

-- If @EndDate is NULL, then use @StartDT + 1

SELECT @EndDate = ISNULL(@EndDate, DATEADD(day, 1, DATEDIFF(day, 0, @StartDT)))

,@Consignment\_No = NULLIF(@Consignment\_No, '')

,@CustID = NULLIF(@CustID, '')

,@debug = CASE WHEN @debug IS NULL THEN 0

WHEN @debug IN (0,1) THEN @debug

ELSE 0 END;

IF @debug = 1

BEGIN

PRINT CONVERT(VARCHAR(20), @StartDate, 120);

PRINT CONVERT(VARCHAR(20), @EndDate, 120);

END

IF @StartDT IS NULL AND @EndDT IS NULL AND

@Consignment\_No IS NULL AND @CustID IS NULL

SELECT @SQL = @SQL + N'SELECT Consignment\_No, ReferenceNo, CustID, Total\_Pkgs

,Last\_Status, Last\_Status\_DT' + CHAR(10) +

N'FROM (' + CHAR(10);

SELECT @SQL = @SQL + N'SELECT ' +

CASE WHEN @StartDT IS NULL AND @EndDT IS NULL AND

@Consignment\_No IS NULL AND @CustID IS NULL

THEN N'TOP 5 '

ELSE '' END +

N' a.Consignment\_No, a.ReferenceNo, a.CustID, a.Total\_Pkgs

,a.Last\_Status, a.Last\_Status\_DT

FROM dbo.Shipments a' + CHAR(10) +

CASE WHEN @StartDate IS NOT NULL

THEN N' AND a.Last\_Status\_DT >= @StartDT ' + CHAR(10) +

N' AND a.Last\_Status\_DT < @EndDT ' + CHAR(10)

ELSE N'' END +

CASE WHEN @Consignment\_No IS NOT NULL

THEN N' AND a.Consignment\_No = @Consignment\_No ' + CHAR(10)

ELSE N'' END +

CASE WHEN @CustID IS NOT NULL

THEN N' AND a.CustID = @CustID ' + CHAR(10)

ELSE N'' END + CHAR(10) +

CASE WHEN @StartDT IS NULL AND @EndDT IS NULL AND

@Consignment\_No IS NULL AND @CustID IS NULL

THEN N'ORDER BY Last\_Status\_DT DESC) Shipments' + CHAR(10)

ELSE '' END +

N'ORDER BY Consignment\_No';

IF @debug = 1

PRINT @SQL;

EXEC sp\_executesql @SQL

,@SQLParms

,@StartDT = @StartDate

,@EndDT = @EndDate

,@Consignment\_No = @Consignment\_No

,@CustID = @CustID;

END

You can see how in the above we’ve added some SQL that is specific to the case where all four of the SP’s input parameters are passed as NULL.  While just a bit messy in my opinion, it can be made to work.

Running this query, produces a new results set:

EXEC dbo.Shipment\_Tracking\_QG1

@StartDT = NULL

,@EndDT = NULL

,@Consignment\_No = NULL

,@CustID = NULL

,@debug = 1;

The dynamic SQL produced by the PRINT statement (when @debug=1) now looks like this (after a bit of touch up formatting):

SELECT Consignment\_No, ReferenceNo, CustID, Total\_Pkgs

,Last\_Status, Last\_Status\_DT

FROM (

SELECT TOP 5 a.Consignment\_No, a.ReferenceNo, a.CustID, a.Total\_Pkgs

,a.Last\_Status, a.Last\_Status\_DT

FROM dbo.Shipments a

ORDER BY Last\_Status\_DT DESC

) Shipments

ORDER BY Consignment\_No

And the results returned are the “governed” results:

Consignment\_No ReferenceNo CustID Total\_Pkgs Last\_Status Last\_Status\_DT

0152037 NULL ACME 1 MDE 2013-10-28 13:11:00.000

0642191 SO 1111 APPLE 1 MDE 2013-10-29 11:37:00.000

4292860 NULL ATLAS 1 POD 2013-10-25 08:49:00.000

6289811 NULL APPLE 1 POD 2013-10-27 14:45:00.000

7271931 NULL NULL 1 WGT 2013-10-29 00:36:00.000

Of course, this approach begs the question, how many rows is the right number to return to maximize the likelihood that the user will find what he needs?  That would probably depend on the daily transaction volume, so what if over time that increases ten or one hundred-fold?  If we start by returning 1,000 rows, it might require 10,000 rows a year from now.

**An Alternative Query Governor**

Let’s define a table that contains rows that are based on the search form.

CREATE TABLE dbo.QueryGovernor

(

FormName VARCHAR(20)

,UserRole VARCHAR(20)

,LookbackDays INT

    ,PRIMARY KEY (FormName, UserRole)

);

INSERT INTO dbo.QueryGovernor (FormName, UserRole, LookbackDays)

SELECT 'MySearchQuery', 'NORMAL', 5;

SELECT \*

FROM dbo.QueryGovernor;

Note how, in our table we have left a placeholder column for the UserRole, which is to say different roles may have a different number of look back days assigned.  The assumption is that a super-user of the system might not be quite as lizard-brained as a normal user, so for them we might allow a longer look back period.

The modifications to our original SP are a little bit more straightforward and a lot less messy.

CREATE PROCEDURE [dbo].[Shipment\_Tracking\_QG2]

(

@StartDT DATETIME = NULL

,@EndDT DATETIME = NULL

,@Consignment\_No VARCHAR(30) = NULL

,@CustID VARCHAR(20) = NULL

,@debug TINYINT = 0 -- 1=Display debug results

)

AS BEGIN

SET NOCOUNT ON;

-- Truncate time component from start date

DECLARE @StartDate DATETIME = DATEADD(day, 0, DATEDIFF(day, 0, @StartDT))

-- Truncate time component from end date and add 1

,@EndDate DATETIME = DATEADD(day, 1, DATEDIFF(day, 0, @EndDT))

,@CurrentDT DATETIME = '2013-10-30' -- Normally you should assign GETDATE()

-- Days to look back from the QueryGovernor table

,@LookbackDays INT =

(

SELECT LookbackDays

FROM dbo.QueryGovernor

WHERE FormName = 'MySearchQuery' -- AND the type of user if applicable

)

,@SQL NVARCHAR(MAX)

,@SQLParms NVARCHAR(MAX) = N'

@StartDT DATETIME

,@EndDT DATETIME

,@Consignment\_No VARCHAR(30)

,@CustID VARCHAR(20)

,@CurrentDT DATETIME

,@LookbackDays INT';

-- If @EndDate is NULL, then use @StartDT + 1

SELECT @EndDate = ISNULL(@EndDate, DATEADD(day, 1, DATEDIFF(day, 0, @StartDT)))

,@Consignment\_No = NULLIF(@Consignment\_No, '')

,@CustID = NULLIF(@CustID, '')

,@debug = CASE WHEN @debug IS NULL THEN 0

WHEN @debug IN (0,1) THEN @debug

ELSE 0 END;

IF @debug = 1

BEGIN

PRINT CONVERT(VARCHAR(20), @StartDate, 120);

PRINT CONVERT(VARCHAR(20), @EndDate, 120);

END

SELECT @SQL = N'

SELECT a.Consignment\_No, a.ReferenceNo, a.CustID, a.Total\_Pkgs

,a.Last\_Status, a.Last\_Status\_DT

FROM dbo.Shipments a' + CHAR(10) +

CASE WHEN @StartDate IS NOT NULL

THEN N' AND a.Last\_Status\_DT >= @StartDT ' + CHAR(10) +

N' AND a.Last\_Status\_DT < @EndDT ' + CHAR(10)

ELSE N'' END +

CASE WHEN @Consignment\_No IS NOT NULL

THEN N' AND a.Consignment\_No = @Consignment\_No ' + CHAR(10)

ELSE N'' END +

CASE WHEN @CustID IS NOT NULL

THEN N' AND a.CustID = @CustID ' + CHAR(10)

ELSE N'' END + CHAR(10) +

CASE WHEN @StartDT IS NULL AND @EndDT IS NULL AND

@Consignment\_No IS NULL AND @CustID IS NULL

THEN N'WHERE a.Last\_Status\_DT >= DATEADD(day,-@LookbackDays,@CurrentDT)' + CHAR(10)

ELSE N'' END + CHAR(10) +

N'ORDER BY Consignment\_No';

IF @debug = 1

PRINT @SQL;

EXEC sp\_executesql @SQL

,@SQLParms

,@StartDT = @StartDate

,@EndDT = @EndDate

,@Consignment\_No = @Consignment\_No

,@CustID = @CustID

,@CurrentDT = @CurrentDT

,@LookbackDays = @LookbackDays;

END

Now when we run our SP and trigger the query governor, our results are the same as limiting to the TOP 5 rows but will now automatically vary depending on the number of transactions created each day:

Consignment\_No ReferenceNo CustID Total\_Pkgs Last\_Status Last\_Status\_DT

0152037 NULL ACME 1 MDE 2013-10-28 13:11:00.000

0642191 SO 1111 APPLE 1 MDE 2013-10-29 11:37:00.000

4292860 NULL ATLAS 1 POD 2013-10-25 08:49:00.000

6289811 NULL APPLE 1 POD 2013-10-27 14:45:00.000

7271931 NULL NULL 1 WGT 2013-10-29 00:36:00.000

The SQL created by our SP now looks like this (also more straightforward):

SELECT a.Consignment\_No, a.ReferenceNo, a.CustID, a.Total\_Pkgs

,a.Last\_Status, a.Last\_Status\_DT

FROM dbo.Shipments a

WHERE a.Last\_Status\_DT >= DATEADD(day, [-@LookbackDays](mailto:-@LookbackDays), @CurrentDT)

ORDER BY Consignment\_No

In reality, the @CurrentDT local variable and parameter we passed when executing the dynamic SQL is not really needed because you can simply replace it with GETDATE().  We did that so you could replicate and play around with the SQL provided in this article.

Of course, if you need to vary the query governor by role, you’d probably also need to pass the user’s login into the SP and apply it where the @Dayslookback local variable’s initial value is set (in the sub-query’s WHERE clause).

**The Down-side Risk of Using one of these Approaches**

When our dumb user with the lizard-brain runs one of his “I’m too lazy to think about what I want, so give me everything queries” and then does a manual search through the returned results, he will be mighty disconcerted when he doesn’t find that which he seeks.

If the query governor is quietly limiting the results returned, users may be asking the question “I know what I want is there, so why can’t I see it?”  It is quite possible that they’ll assume there’s a bug in the application for this case.

Communication is essential to limiting misunderstandings.  Tell the user community why the query governor is employed, and better yet give them a form that allows them control over its settings on a form-by-form basis.

And be sure to remind them that if they are careful in considering what filtering criteria to apply, it overrides the query governor.

**Conclusion**

When searching against a transactions table, my belief is that users for the most part are interested in looking at recent transactions.  The more current transactions are more likely to be of interest than those from the distant past.  Most transaction tables have some sort of date on each row that you can use to identify what is current and what is not.

So my preferred approach is to implement a query governor that is configurable.  Give the user access to the governor and allow them to define the governors’ limits by form and based on role.

The main point of this article though is to limit the loading of cached records onto your application server and improving the overall performance of your application, when you’re not dealing with users that have a concern for such things.

Dwain Camps

SQL Performance Evangelist