

Dynamic Search Conditions

Erland Sommarskog
SQL Server MVP

Polish SQL Server Users Group 25 February 2016



Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since April 2001

<http://www.sommarskog.se>

esquel@sommarskog.se

Dynamic Search Conditions

Be able to search on many different conditions (order id, date interval etc) with correct result **and** good performance for most or all conditions.

We will work in the database Northgale that has approx. 350 000 orders.

(To install, first run [instnwnd.sql](#) and then [Northgale.sql](#))

The Unfiltered Search

```
SELECT o.OrderID, o.OrderDate, od.UnitPrice,  
       od.Quantity, c.CustomerID, c.CompanyName,  
       c.Address, c.City, c.Region, c.PostalCode,  
       c.Country, c.Phone, p.ProductID,  
       p.ProductName, p.UnitsInStock,  
       p.UnitsOnOrder, o.EmployeeID  
FROM Orders o  
JOIN [Order Details] od ON o.OrderID = od.OrderID  
JOIN Customers c ON o.CustomerID = c.CustomerID  
JOIN Products p ON p.ProductID = od.ProductID
```

SP Interface

```
CREATE PROCEDURE static_search_1
    @orderid      int          = NULL,
    @status       char(1)      = NULL,
    @fromdate     datetime     = NULL,
    @todate       datetime     = NULL,
    @custid       nchar(5)     = NULL,
    @custname     nvarchar(40) = NULL,
    @city         nvarchar(25) = NULL,
    @region       nvarchar(15) = NULL,
    @prodid       int          = NULL,
    @prodname     nvarchar(40) = NULL AS
```

Static SQL or Dynamic SQL?

One of them is not better than the other.
Where one is strong, the other is poor – and vice versa.

Static SQL – preferred for simple problems.

Dynamic SQL – when complexity grows.

We start with static SQL and then go dynamic.

General Pattern for Static SQL

```
WHERE (o.OrderID = @orderid OR @orderid IS NULL)
      AND (o.Status = @status OR @status IS NULL)
      AND (o.OrderDate >= @fromdate OR @fromdate IS NULL)
      AND (o.OrderDate <= @todate OR @todate IS NULL)
      AND (o.CustomerID = @custid OR @custid IS NULL)
      AND (c.CompanyName LIKE @custname + '%' OR
            @custname IS NULL)
      AND (c.City = @city OR @city IS NULL)
      AND (c.Region = @region OR @region IS NULL)
      AND (od.ProductID = @prodid OR @prodid IS NULL)
      AND (p.ProductName LIKE @prodname + '%' OR
            @prodname IS NULL)

ORDER BY o.OrderID
```


What About Performance?

Let's test...

[static_search_1](#)

The plan is optimised for the first search ("parameter sniffing") and does not fit other search conditions.

We need different plans for different search conditions.

Side Track

Some people say that this is faster:

```
WHERE o.OrderID = isnull(@orderid, o.OrderID)
      AND o.Status = isnull(@status, o.Status)
      AND ...
```

[static_search_2](#)

This is a trap — does not work with NULL!

Do not use!

And, no, it does not run faster.

Different Plans for Different Search Conditions

CREATE PROCEDURE ... WITH RECOMPILE AS

Compile the procedure every time.

Any improvement?

[static_search_3](#)

Better performance, but not optimal — index scan where we would like a seek.

The plan is optimised for the input parameters, but it must still be correct even if the value of any parameter changes before the SELECT.

OPTION (RECOMPILE)

Query hint which you put after the SQL statement. It forces a recompile of that statement on every execution.

[static_search_4](#)

Since only the SQL statement is recompiled, all variables can be handled as **constants**.

To get good performance with searches with static SQL, you (almost) always need this hint.

EXEC static_search_4 @status = 'N'

```
WHERE (o.OrderID = @orderid OR @orderid IS NULL)
      AND (o.Status = @status OR @status IS NULL)
      AND (o.OrderDate >= @fromdate OR @fromdate IS NULL)
      AND ...
OPTION (RECOMPILE)
```

Variables handled as constants =>

```
WHERE (o.OrderID = NULL OR NULL IS NULL)
      AND (o.Status = 'N' OR 'N' IS NULL)
      AND (o.OrderDate >= NULL OR NULL IS NULL)
      AND ...
```

Or plain and simply:

```
WHERE o.Status = 'N'
```

=> The filtered index on Status is usable.

Check your SQL Version!

OPTION (RECOMPILE) works as WITH RECOMPILE (variables handled as variables) in SQL 2005 and early SQL 2008.

Minimum requirements:

SQL 2008 SP2

SQL 2008 R2 SP1

Compiling Always is Expensive?

It depends...

A few searches per minute and 100 ms in compile time — no problem. Not the least if total execution time goes from 5 seconds to 200 ms.

Searching on a simple condition like order ID 100 times a second — that hurts.

Specific Branches for Frequent Searches

```
IF @orderid IS NOT NULL
```

```
BEGIN
```

```
...
```

```
WHERE o.OrderID = @orderid
```

```
AND (o.Status = @status OR @status IS NULL)
```

```
-- OPTION (RECOMPILE)
```

Cached plan

```
END
```

```
ELSE
```

```
BEGIN
```

```
...
```

```
WHERE (o.Status = @status OR @status IS NULL)
```

```
AND (o.OrderDate >= @fromdate OR @fromdate IS NULL)
```

```
...
```

```
OPTION (RECOMPILE)
```

Compilation every time

```
END
```

Different Branches

Code is more difficult to maintain.

Two or three, but hardly more.

Can however be a viable alternative if at most three search conditions are indexed.

Dynamic SQL is better with high call frequency.

Multi-valued Search Conditions

Comma-separated list (CSV)

```
ALTER PROCEDURE static_search_8
...
    @employeeestr nvarchar(MAX) = NULL AS
...
AND (o.EmployeeID IN
    (SELECT n
      FROM intlist_to_tbl (@employeeestr))
 OR @employeeestr IS NULL)
```

(For functions to unpack a CSV into a table see my [*Arrays and Lists in SQL Server 2005*](#).)

Table-Valued Parameters

Need to use an intermediate variable which holds whether there is data in the table or not.

```
ALTER PROCEDURE static_search_8
    ...
    @employeeetbl intlist_tbltype READONLY AS

DECLARE @hasemptbl bit =
    CASE WHEN EXISTS (SELECT * FROM @employeeetbl)
        THEN 1
        ELSE 0
    END

...
AND (o.EmployeeID IN (SELECT val FROM @employeeetbl)
    OR @hasemptbl = 0)
```

Multi-Valued Conditions, cont'd

Optimizer has less information. For a TVP only cardinality. For a CSV not even that.

Selectivity therefore does not affect the plan choice.

Bouncing data over a temp table with statistics (not table variable) gives the optimizer more information. (May need UPDATE STATISTICS.)

Possible trick: with few values, use IN(@var1, @var2...), else use the table.

```
SELECT @cnt = (SELECT COUNT(*) FROM @employeeetbl)

IF @cnt BETWEEN 1 AND 3
BEGIN
    SELECT @emp1 = MIN(val) FROM @employeeetbl
    SELECT @emp2 = MIN(val) FROM @employeeetbl WHERE val>@emp1
    SELECT @emp3 = MIN(val) FROM @employeeetbl WHERE val>@emp2
END
ELSE IF @cnt > 3
    SELECT @hasemptbl = 1
```

```
AND (o.EmployeeID IN (@emp1,@emp2,@emp3) OR @emp1 IS NULL)
```

```
AND (o.EmployeeID IN (SELECT val FROM @employeeetbl) OR
    @hasemptbl = 0)
```

Alternate Tables

When @ishistoric = 1, read historic order tables.

```
FROM (SELECT o.OrderID, o.Status, ...
      FROM Orders o
      JOIN [Order Details] od ON o.OrderID = od.OrderID
      WHERE @ishistoric = 0
      UNION ALL
      SELECT ho.OrderID, ho.Status, ...
      FROM HistoricOrders ho
      JOIN HistoricOrderDetails hod
            ON ho.OrderID = hod.OrderID
      WHERE @ishistoric = 1) AS u
```

Works — but code is more complex. With many tables, it can become unwieldy.

Control Sort Order

CASE to the rescue – be aware of data types!

```
ORDER BY
  CASE @sortcol WHEN 'OrderID'      THEN o.OrderID
               WHEN 'EmployeeID'    THEN o.EmployeeID
               WHEN 'ProductID'     THEN od.ProductID
  END,
  CASE @sortcol WHEN 'CustomerName' THEN c.CompanyName
               WHEN 'ProductName'   THEN p.ProductName
  END,
  CASE @sortcol WHEN 'OrderDate'    THEN o.OrderDate
  END
```

If you want to provide multi-column sort, ASC/DESC, it quickly goes out of hand.

Searching on Alternate Keys

Customer lookup on one of customer ID, tax number or name. Index on all columns

What do you think of:

```
WHERE (CustomerID =@custid AND @custid IS NOT NULL)
      OR (VATno = @vatno AND @vatno IS NOT NULL)
      OR (CompanyName LIKE @custname + '%' AND
          @custname IS NOT NULL)
```

Note: No OPTION (RECOMPILE)!

Start-up Filter

The plan has searches on all three indexes with a start-up filter – only one index is used at a time.

The condition @val IS NOT NULL must be there!

Rarely (if ever) works if search columns are in different tables.

Always test that the plan is what you intended.

Other Examples with Start-up Filters

```
WHERE (@suppl_city IS NULL OR
      EXISTS (SELECT *
              FROM Suppliers s
              WHERE s.SupplierID = p.SupplierID
                   AND s.City      = @suppl_city))

SELECT o.OrderID, o.Status, ...
FROM Orders o
JOIN [Order Details] od ON o.OrderID = od.OrderID
WHERE @ishistoric = 0
UNION ALL
SELECT ho.OrderID, ho.Status, ...
FROM HistoricOrders ho
JOIN HistoricOrderDetails hod
      ON ho.OrderID = hod.OrderID
WHERE @ishistoric = 1
```


Searches with Dynamic SQL

Higher level of difficulty.

Requires more programmer discipline.

More difficult to maintain if poorly written.

Requires more testing: odd combinations may crash.

But you get a lot more flexibility.

Dynamic SQL and permissions

With dynamic SQL, ownership chaining does not apply – users must have direct permissions to the tables

Can be addressed with certificate signing or EXECUTE AS. See

<http://www.sommarskog.se/grantperm.html>

(Granting Permissions through Stored Procedures)

sp_executesql

[sp_executesql](#)

The core in all searches with dynamic SQL.
Creates a nameless SP that is saved in the cache.
The SP is identified by a hash of the query text without normalising spacing, upper/lower etc.

First parameter: @sqlstring. **nvarchar!**
Second parameter: @paramlist. **nvarchar!**
Subsequent parameters: Parameters in @paramlist.

dynamic_search_1

```
DECLARE @sql          nvarchar(MAX),
        @paramlist    nvarchar(4000),
        @nl            char(2) = char(13) + char(10)

SELECT @sql =
    'SELECT o.OrderID, o.OrderDate, ...
    FROM   dbo.Orders o
    JOIN   dbo.[Order Details] od ON o.OrderID = od.OrderID
    JOIN   dbo.Customers c ON o.CustomerID = c.CustomerID
    JOIN   dbo.Products p ON p.ProductID = od.ProductID
    WHERE  1 = 1' + @nl
```

Users may have different default schemas.
Makes it easier to add conditions.
Improves readability of the generated SQL.

Add Conditions

```
IF @orderid IS NOT NULL
    SET @sql += ' AND o.OrderID = @orderid' + @nl
```

```
IF @fromdate IS NOT NULL
    SET @sql += ' AND o.OrderDate >= @fromdate' + @nl
```

```
IF @custname IS NOT NULL
    SET @sql += ' AND c.CompanyName LIKE
                @custname + '''%' + @nl
```

+= handy to make code a little shorter.
Always a space after the opening quote.
Append @nl to make string more readable.
Need to double single quotes in the string.

Multi-Valued Parameters

```
IF EXISTS (SELECT * FROM @employeeetbl)
    SELECT @sql += ' AND o.EmployeeID IN
        (SELECT val FROM @employeeetbl) ' + @nl
```

```
IF @employeeestr IS NOT NULL
    SELECT @sql +=
        ' AND o.EmployeeID IN (SELECT n FROM
        dbo.intlist_to_tbl(@employeeestr)) ' + @nl
```

What about? **Don't even think about it!**

```
IF @employeeestr IS NOT NULL
    SELECT @sql += ' AND o.EmployeeID
        IN (' + @employeeestr + ') ' + @nl
```

We will come back to this in a few slides.

The Debug Parameter

This line should always be there when you work with dynamic SQL.

```
@debug bit = 0 AS
```

```
...
```

```
IF @debug = 1
```

```
    PRINT @sql
```

ALWAYS!

dynamic_search_1 – Making the Call

```
SELECT @paramlist =  
    '@orderid      int,  
     @status       char(1),  
     ...  
     @employeeestr varchar(MAX),  
     @employeeetbl intlist_tbltype READONLY'  
  
EXEC sp_executesql @sql, @paramlist,  
    @orderid, @status, @fromdate, @todate,  
    @custid, @custname, @city, @region,  
    @prodid, @prodname,  
    @employeeestr, @employeeetbl
```


Cache and Compilation

OPTION (RECOMPILE)

Compilation every time.

More compilation than needed.

The plan always fits the current parameters.

Dynamic SQL with parameters

Cached plan per combination of parameters.

Much less compiling.

“Parameter sniffing” can be a problem.

A Bad Example

Some people concatenate the values into the SQL string:

```
IF @orderid IS NOT NULL
    SELECT @sql += ' AND o.OrderID = ' +
                  convert(varchar(10), @orderid) + @nl
...
IF @city IS NOT NULL
    SELECT @sql += ' AND c.City = ''' + @city + '''' + @nl
...
IF @employeeestr IS NOT NULL
    SELECT @sql += ' AND o.EmployeeID IN (' +
                  @employeeestr + '))' + @nl
```

[dynamic_search_bad](#)

Opens for SQL injection!

Cache and Compilation II

```
EXEC static_search_4 11000  
EXEC static_search_4 11001  
EXEC static_search_4 11002
```

3 compilations
1 (unused) cache entry

```
EXEC dynamic_search_1 11000  
EXEC dynamic_search_1 11001  
EXEC dynamic_search_1 11002
```

1 compilation
1 cache entry

```
EXEC dynamic_search_bad 11000  
EXEC dynamic_search_bad 11001  
EXEC dynamic_search_bad 11002
```

3 compilations
3 cache entries

Problems for Parameterised SQL

Cached plan is good – but not always.

[compare_1](#)

```
EXEC dynamic_search_1 @custid = 'ERNTC',  
    @fromdate = '19980218', @todate = '19980218'  
EXEC dynamic_search_1 @custid = 'BOLSR',  
    @fromdate = '19960101', @todate = '19961231'
```

Same parameters used, but which is the best index depends on the parameter values.

```
EXEC dynamic_search_1 @status = 'E'
```

Optimizer cannot use filtered indexes.

Tricks We Can Use

OPTION (RECOMPILE)

```
SELECT @sql += ' ORDER BY o.OrderID' + @nl
IF @custid IS NOT NULL AND
   (@fromdate IS NOT NULL OR @todate IS NOT NULL)
  SELECT @sql += ' OPTION (RECOMPILE)' + @nl
```

Change the query text depending on the values

```
IF @fromdate IS NOT NULL AND @todate IS NOT NULL
BEGIN
  SELECT @diff = datediff(DAY, @fromdate, @todate)
  SELECT @sql += CASE WHEN @diff = 0 THEN ''
                     WHEN @diff <= 7 THEN ' AND 2=2 '
                     WHEN @diff <= 30 THEN ' AND 3=3 '
                     ...
```

Handle common cases separately

```
IF @fromdate = @todate
    SELECT @sql += ' AND o.OrderDate = @fromdate' + @nl
ELSE
BEGIN
    IF @fromdate IS NOT NULL
        SELECT @sql += ' AND o.OrderDate >= @fromdate' + @nl

    IF @todate IS NOT NULL
        SELECT @sql += ' AND o.OrderDate <= @todate' + @nl
END
```

Add condition from index filter

```
IF @status IS NOT NULL
    SELECT @sql += ' AND o.Status = @status' +
        CASE WHEN @status <> 'C'
            THEN ' AND o.Status <> ''C'''
            ELSE ''
        END + @nl
```

Concatenate Values into the Query?

May be a good idea for columns with only a handful possible values and skewed distribution.

```
IF @status IS NOT NULL
    SELECT @sql += ' AND o.Status = ' +
               quotename(@status, ''') + @nl
```

Bad idea for customer, city etc since the cache is littered. But maybe for very common values.

Lists: unpack and build a new list. Again, only if there are few possible values.

Select Sort Order

Could it be this simple?

```
@sql += ' ORDER BY ' + @sortcol
```

How neat – @sortcol could be a list of columns!

No! Risk for SQL injection. Use quotename!

```
@sql += ' ORDER BY ' + quotename(@sortcol)
```

Now lists will not work, but you need @sortcol1, @sortcol2 etc. Or decode @sortcol.

Select Sort Order, cont'd

Thus, this is likely to be better after all:

```
SELECT @sql += ' ORDER BY ' +  
    CASE @sortcol1  
        WHEN 'OrderID'           THEN 'o.OrderID'  
        WHEN 'EmployeeID'        THEN 'o.EmployeeID'  
        WHEN 'ProductID'         THEN 'od.ProductID'  
        WHEN 'CustomerName'      THEN 'c.CompanyName'  
        WHEN 'ProductName'       THEN 'p.ProductName'  
        ELSE 'o.OrderID'  
    END +  
    CASE @isdesc1 WHEN 0 THEN ' ASC' ELSE ' DESC' END
```

But the best may be to sort client-side...

Alternate Tables

As with sorting – don't send in table names, but translate to dbo.tblname etc.

Implementing @ishistoric

```
FROM    dbo.' + CASE @ishistoric
          WHEN 0 THEN 'Orders'
          WHEN 1 THEN 'HistoricOrders'
        END + ' o
JOIN    dbo.' + CASE @ishistoric
          WHEN 0 THEN '[Order Details]'
          WHEN 1 THEN 'HistoricOrderDetails'
        END + ' od ON o.OrderID = od.OrderID
```

GROUP BY, Aggregation, Select Columns etc

There is a limit for dynamic SQL in an SP as well.

Key problem: How express all this in a parameter list?

May be simpler to construct the SQL client-side in a class with an O-O interface.

Never send SQL syntax to a stored procedure!

Conclusion

Static SQL with OPTION (RECOMPILE) – good for the simple cases, but complexity grows fast.

Dynamic SQL – too much hassle for simple cases, but scales better in complexity.

Make a decision from case to case what to use.

That's All Folks!

Erland Sommarskog

esquel@sommarskog.se

<http://www.sommarskog.se/present>

Slides and all scripts, including scripts to create the database.