

Dynamic Search Conditions

The SQL 2025 Edition

Erland Sommarskog

Data Platform MVP

Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since 2001

esquel@sommarskog.se

<http://www.sommarskog.se>

Slides and scripts on
<http://www.sommarskog.se/present>



18th edition SQLDay Conference



11-13 May 2026, WROCŁAW + ONLINE

Platinum sponsor



Gold sponsors



Silver sponsors



Tabular Editor



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 one million orders.**
- **A production database may have 100 million orders...**

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 will start with static SQL and then go dynamic.

Today's Search Task



```
SELECT o.OrderID, o.OrderDate, od.UnitPrice, od.Quantity,  
       c.CustomerID, c.CustomerName, 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
```

We want to implement a search in this space of order lines.

Search Parameters



@orderid	int	Return this order.
@status	char(1)	N(ew), P(rocessing), E(rror), C(ompleted). 99% are C. Filtered index on <> 'C'.
@fromdate	date	>= @fromdate.
@todate	date	<= @todate.
@custid	nchar(5)	Search on specific customer.
@custname	nvarchar(40)	Starts with. 'Bla' returns both Black and Blake.
@city	nvarchar(25)	City customer comes from.
@region	nvarchar(15)	Region for customer.
@prodid	int	Specific product.
@prodname	nvarchar(40)	Starts with.

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.CustomerName 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
```


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.CustomerName 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
```

Ex: @fromdate = '2026-05-05', @city = N'Bydgoszcz'

What About Performance?

Let's test, first the old way like we are on SQL 2022 or earlier.

`static_search_1`

- The plan is optimised for the first search ("parameter sniffing") and does not fit other search conditions.
- Because plan must be correct for all possible values, there cannot be Index Seeks, only scans.
- We need different plans for different search conditions.

A Trap to Watch Out For

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

[static_search_2](#)

- Customers.Region is nullable.
- Thus you may get: NULL = NULL => UNKNOWN => Rows are filtered out when they should not be. **Do not use!**
- And, no, it does not run faster.

Optional Parameter Plan Optimisation



- This new feature in SQL 2025 sounds promising. Let's upgrade! [static_search_simple](#)
- The main plan for the query is a *dispatcher plan*.
- It defines query variants depending on optional_predicate entries that appear in the PLAN PER VALUE hint.
- At run-time, the dispatcher plan invokes the query variant depending on the actual NULL settings of the parameters.
- The plan for a variant is compiled first time it's invoked.
- We get different plans for different search conditions!

Let's Try it For Real



[static_search_1](#)

- Big disappointment.
- You can never get more than three optional_predicate entries - our procedure has ten optional parameters.
- Often you only get one, typically for the parameter you pass in the first call. (Thus, still parameter sniffing.)
- A *small* improvement - you can get Index Seeks on a few occasions. (As compared to never on older versions.)
- We need to look for something better.

OPTION (RECOMPILE)



- Query hint that you put after the SQL statement. It forces a recompile of that statement on every execution. `static_search_4`
- Since the SQL statement is recompiled every time, all variables can be handled as **constants**.
- To get good performance with searches with static SQL, you (almost) always need this hint.
- If you only have one search parameter, Optional Parameter Plan Optimisation should work well.

Expensive to Always Compile?

- It depends...
- A few searches per minute and 50 ms in compile time - no problem. Not the least if the 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
    SELECT ...
    WHERE o.OrderID = @orderid
        AND (o.Status = @status OR @status IS NULL)
        AND ...
    -- OPTION (RECOMPILE)
```

Cached plan

```
END
ELSE
BEGIN
    SELECT ...
    WHERE (o.Status = @status OR @status IS NULL)
        AND ...
    OPTION (RECOMPILE)
END
```

Compilation every time

Different Branches, cont'd

- Code is more difficult to maintain.
- Two or three branches, 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)

```
CREATE PROCEDURE static_search_5
    ...
    @employeeestr nvarchar(MAX) = NULL AS
    ...
    AND (o.EmployeeID IN (SELECT convert(int, value)
                          FROM string_split(@employeeestr, ','))
        OR @employeeestr IS NULL)
```

Table-Valued Parameters

Need a variable to hold whether the table has data.

```
CREATE PROCEDURE static_search_5
    ...
    @employeetbl dbo.intlist_tbltype READONLY AS

DECLARE @hastable bit =
    IIF(EXISTS (SELECT * FROM @employeetbl), 1, 0)

...
AND (o.EmployeeID IN (SELECT val FROM @employeetbl) OR
    @hastable = 0)
```

Multi-Valued, Performance

- Optimizer has less information.
- For a TVP it knows the number of rows, but not more.
- Good enough - if distribution is even.
- If there is a skew, you can get poor plan choices.
- `string_split`? A blind guess of 50 rows - typically too high.

Multi-Valued, Solution for Performance



To handle skew and poor estimates for CSVs:

- Bounce data over a temp table to get correct cardinality and distribution statistics.
- Run UPDATE STATISTICS on #temp table to avoid running with statistics from previous execution.
- Script with various variations. Study the cardinality estimate for the leftmost operator to see the difference.

[static_search_5](#)

Searches with Dynamic SQL

- Higher level of difficulty.
- Requires more programmer discipline.
- More difficult to maintain if poorly written.
- Requires more testing.
- Permissions - users must have direct permissions to tables.
 - Can be addressed with certificate signing or EXECUTE AS. See my article *[Packaging Permissions in Stored Procedures](#)*.
- But you get a lot more flexibility.

sp_executesql

- The core in all searches with dynamic SQL.
- Creates a nameless SP that is saved in the cache and executes it. Next time it looks up the procedure and plan in cache.
- The procedure is identified by a hash of the query text as-is - no normalising of spacing, upper/lower etc.

`sp_executesql`

First parameter: @sqlstring. **nvarchar!**
Second parameter: @paramlist. **nvarchar!**
Subsequent parameters: Parameter values for @paramlist.

dynamic_search_1

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.CustomerName LIKE @custname + '''%' + @nl
```

- += handy to make code a little shorter.
- Always a space after the opening quote.
- Append @nl to make string more readable.
- Single quotes in the string must be doubled.

Multi-Valued Parameters

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

IF @employeeestr IS NOT NULL
    SELECT @sql += ' AND o.EmployeeID IN
        (SELECT convert(int, value)
        FROM string_split(@employeeestr, ',', '''))' + @nl
```

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

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

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  dbo.intlist_tbltype READONLY'

EXEC sp_executesql @sql, @paramlist,
    @orderid, @status, @fromdate, @todate,
    @custid, @custname, @city, @region,
    @prodid, @prodname, @employeeestr, @employeeetbl
```

dynamic_search_1

All parameters to the SP are included in the parameter list.

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

dynamic_search_bad

Opens for SQL injection!

Cache and Compilation



OPTION (RECOMPILE)

- **Compilation every time.**
- **More compilation than needed.**
- **The plan always fit the current parameters.**

Parameterised dynamic SQL

- **Cached plan per combination of parameters.**
- **Much less compiling.**
- **“Parameter sniffing” can be a problem.**

[compare_1](#)

Dealing with Parameter Sniffing



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
```

Good solution as long as call frequency is not a concern.

Altering Query Text from Values

```
DECLARE @days int = datediff(DAY, @fromdate, @todate)
SELECT @sql || = ' -- ' ||
        CASE WHEN @days < 7 THEN @days || ' days.'
              WHEN @days < 35 THEN @days / 7 || ' weeks.'
              ELSE @days / 30 || ' months.'
        END || @nl
```

- Different comments -> different cache entries.
- Some cache entries will have the same plan - but not all.
- Understanding the data definitely helps when doing this trick.
- The power of dynamic SQL!

Inlining Certain Values

Columns with only a few possible values and large skew:

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

Say that 60 % of all customers are in London:

```
IF @city IS NOT NULL
    SELECT @sql += ' AND c.City = ' +
        IIF(@city = N'London', N'N''London'', '@city') + @nl
```

Don't inline on a general basis - you will litter the cache!

But Didn't Microsoft Solve This?



- In SQL 2022, Microsoft introduced PSP Optimization.
 - PSP = Parameter-Sensitive Plans.
- It only works with equi-predicates (=), so it will not help with date ranges and similar.
- For skews like 60% of customers in London, it *may* kick in.
- But it's quite rudimentary, your own fine-tuning will still be better.

[dynamic_search_2](#)

Conclusion

- **Optional Parameter Plan Optimisation does not solve the problem.**
- **Static SQL with OPTION (RECOMPILE) - good solution if:**
 - You only have a number of plain conditions.
 - You can accept compilation for every execution.
 - When you stray beyond that, complexity can grow rapidly.
- **Dynamic SQL - too much hassle for simple cases, but reduces compilation and complexity scales better.**
- **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.

(To create the database, first run [instnwnd.sql](#) and then [Northgale.sql](#))