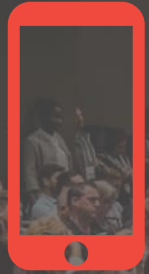




# Dynamic Search Conditions

Erland Sommarskog,  
Erland Sommarskog SQL-Konsult AB



Please silence  
cell phones



# Explore everything PASS has to offer

**Free Online Resources**  
**Newsletters**  
**PASS.org**



**24HOURS**  
of  **PASS**

Free online webinar events



**PASS**  
**LOCAL**  
**GROUPS**

Local user groups around the world



 **PASS**  
**SQLSATURDAY**

Free 1-day local training events



**PASS**  
**VIRTUAL**  
**GROUPS**

Online special interest user groups



 **PASS**  
**MARATHON**

Business analytics training



**PASS**  
**VOLUNTEERS**

Get involved

# Session evaluations

Your feedback is important and valuable.

**Submit by 5pm Friday, November 16th to win prizes.**

3 Ways to Access:



**Go to [passSummit.com](https://passSummit.com)**



**Download the GuideBook App**  
and search: PASS Summit 2018



**Follow the QR code link** displayed on session signage throughout the conference venue and in the program guide



# Erland Sommarskog

Erland Sommarskog  
SQL-Konsult AB

<http://www.sommarskog.se>

[esquel@sommarskog.se](mailto:esquel@sommarskog.se)

Independent Consultant based in  
Stockholm

Worked with SQL Server since  
1991

SQL Server MVP since 2001

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

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	Search on a specific 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
```

Ex: @fromdate = '2018-11-08', @city = 'Seattle'

# 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 = '2018-11-08', @city = 'Seattle'

# 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.

# 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 ...
```

Customers.Region is nullable, thus you may get  
NULL = NULL => UNKNOWN =>  
Rows are filtered out when they shouldn't be.

**Do not use!**

# 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 – scans instead of seeks.

Input parameters are sniffed, but the optimizer must consider that they may change before query runs – no flow analysis.

# 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 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.

Note: Requires SQL 2008 (latest service packs) or later. Works differently on SQL 2005.

# 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
```

```
...
```

```
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 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 n FROM intlist_to_tbl (@employeeestr))
        OR @employeeestr IS NULL)
```

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

# Table-Valued Parameters

Need a variable to hold whether the table has data.

```
CREATE PROCEDURE static_search_5
    ...
    @employeeetbl intlist_tbltype READONLY AS

DECLARE @hastable bit =
    IIF(EXISTS (SELECT * FROM @employeeetbl), 1, 0)
...
    AND (o.EmployeeID IN (SELECT val FROM @employeeetbl) OR
        @hastable = 0)
```

# Multi-Valued, Performance

Optimizer knows less.

For a TVP it knows the number of rows, but not more.

Good enough – if distribution is even. **Not if there is a skew.**

For a CSV the optimizer has no clue and makes a blind guess.

In SQL 2016 and earlier – SQL 2017 runs UDF before compiling.  
(But not for string\_split!)

Bounce data over a temp table to get distribution statistics.

Use UPDATE STATISTICS to avoid stale stats.

[static\\_search\\_5](#)

# 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
```

Increased complexity. 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.CustomerName
              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

We want to look up customer on one of customer ID, tax number or name. Index on all three columns.

What do you think of this:

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

Note: No OPTION (RECOMPILE)!

[cust\\_lookup](#)



# 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))
```

Note: not operator shortcutting!

# Start-up Filters, cont'd

We have seen this one before.

```
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
```

Again: Always test to verify that start-up filters are used!

# 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

`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 SP in cache.

The SP is identified by a hash of the query text as-is – no normalising of spacing, upper/lower etc.

First parameter: @sqlstring. **nvarchar!**

Second parameter: @paramlist. **nvarchar!**

Subsequent parameters: Parameter values for @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.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.

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 + ')'
```

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
```

[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) + @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 + ')'
```

dynamic search bad

**Opens for SQL injection!**

# Cache and Compilation

```
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

# Cache and Compilation II

## 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.

[compare\\_1](#)

# Dealing with Parameter Sniffing

## OPTION (RECOMPILE)

```
SELECT @sql += ' ORDER BY o.OrderID' + @n1
IF @custid IS NOT NULL AND (@fromdate IS NOT NULL OR
                             @todate IS NOT NULL)
    SELECT @sql += ' OPTION(RECOMPILE)' + @n1
```

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

# Altering Query Text Depending on Values

```
IF @fromdate IS NOT NULL AND @todate IS NOT NULL
BEGIN
    SELECT @datediff = datediff(DAY, @fromdate, @todate)
    SELECT @sql += CASE WHEN @datediff = 0 THEN '-- Single day'
                        WHEN @datediff <= 7 THEN '-- A week'
                        WHEN @datediff <= 30 THEN '-- A month'
                        ELSE '-- More than a month'
                    END + @nl
END
```

Requires understanding of the data.

Less granular than OPTION(RECOMPILE) but less compiling.

# Inlining Certain Values

Say that 60% of all customers are in London.

```
IF @city IS NOT NULL
    SELECT @sql += ' AND c.City = ' +
        CASE @city WHEN N'London' THEN N'N''London''
                    ELSE '@city' END + @nl
```

Columns with a few possible values that are skewed:

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

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



# Enabling Filtered Indexes

Add the filter condition to the query:

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

[dynamic\\_search\\_2](#)

# Select Sort Order

Could it be this simple?

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

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

No! Two problems:

- SQL injection.
- Client depends on column names.

# Select Sort Order, cont'd

Thus, I think this is 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.CustomerName'  
        WHEN 'ProductName'   THEN 'p.ProductName'  
        WHEN 'OrderDate'     THEN 'o.OrderDate'  
        ELSE                  'o.OrderID'  
    END + ' ' + IIF(@isdesc1 = 0, 'ASC', 'DESC')
```

Must have ELSE to avoid NULL in @sql!

# Alternate Tables

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.

# Session evaluations

Your feedback is important and valuable.

**Submit by 5pm Friday, November 16th to win prizes.**

3 Ways to Access:



**Go to [passSummit.com](https://passSummit.com)**



**Download the GuideBook App**  
and search: PASS Summit 2018



**Follow the QR code link** displayed on session signage throughout the conference venue and in the program guide



# Thank You

Erland Sommarskog



esquel@sommarskog.se

