

# Dynamiska sökvillkor

Erland Sommarskog  
SQL Server MVP

*Stockholm 17 november*

*Göteborg 18 november*

*Malmö 19 november*

A photograph of a forest scene with a small stream flowing over rocks. The trees are tall and thin, and the ground is covered with green ferns and other vegetation. The text is overlaid on the image.

# Erland Sommarskog

SQL Server MVP sedan april 2001

<http://www.sommarskog.se>

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

Erland Sommarskog SQL-Konsult AB



# Dynamiska sökvillkor

Kunna söka på många olika villkor (orderid, kund, datumintervall etc) med korrekt resultat **och** bra prestanda för de allra flesta villkor.

Vi kommer att arbeta i databasen Northgale med c:a 350 000 ordrar.

(För att installera, kör först [instnwnd.sql](#) och sedan [Northgale.sql](#))

# SP-gränssnitt

```
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(15) = NULL,
    @region       nvarchar(15) = NULL,
    @prodid       int          = NULL,
    @prodname     nvarchar(40) = NULL AS
```

# SP-gränssnitt II

```
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
```

# Statisk SQL eller dynamisk SQL?

Den ena är inte bättre än den andra, utan den enes svaghet är den andres styrka – och omvänt.

Statisk SQL – främst för de enkla problemen.

Dynamisk SQL – när komplexiteten växer.

Statisk SQL före paus, sedan blir vi dynamiska.

# Allmän princip för statisk 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
```

# Hur blir prestanda?

Vi testar...

[static\\_search\\_1](#)

Planen optimeras för första sökningen ("parameter sniffing"), och passar inte för andra sökkriterier.

Vi behöver olika planer för olika sökkriterier.



# Stickspår

En del säger att detta går fortare:

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

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

Detta är en fälla — fungerar inte med NULL!

**Använd inte!**

Och nej, det går inte fortare.

# Olika planer för olika kriterier

CREATE PROCEDURE ... WITH RECOMPILE AS

Kompilera om proceduren varje gång.

Blir det bättre?

[static\\_search\\_3](#)

Bättre prestanda, men inte optimal – index skannas i stället för att sökas.

Planen optimeras för inparametrarna, men måste vara korrekt för alla möjliga värden – i fall de ändras innan SELECT:n nås.

# OPTION (RECOMPILE)

Query hint som sätts efter SQL-satsen och framtvingar omkompilering av just den satsen varje gång.

[static\\_search\\_4](#)

I och med att enbart SQL-satsen kompileras om, kan alla variabler hanteras som **konstanter**.

Krävs (nästan) alltid för bra prestanda för sökningar med statisk SQL.

# 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)
```

## Variabler hanteras som konstanter =>

```
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 ...
```

## Eller kort och gott

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

=> Index på Status kan användas.



# Se upp med SQL-versionen

OPTION (RECOMPILE) fungerar som WITH RECOMPILE (variabler hanteras som variabler) i SQL 2005 och tidig SQL 2008.

Minimikrav:

SQL 2008 SP2

SQL 2008 R2 SP1

# Dyrt att alltid kompilera?

Beror på...

Sökning några gånger i minuten och 100 ms för att kompilera – inget problem. Inte minst om totala tiden går från 5 sekunder till 200 ms.

Sökning på enkelt villkor som order-id 100 ggr/sek – det gör ont.

# Särskilda grenar för vanliga villkor

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

```
BEGIN
```

```
...
```

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

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

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

Cachad plan

```
END
```

```
ELSE
```

```
BEGIN
```

```
...
```

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

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

```
...
```

```
OPTION (RECOMPILE)
```

Kompilering varje gång

```
END
```

# Olika grenar för vissa villkor

Koden blir svår att underhålla.

Mer än tre grenar är knappast rimligt.

Kan dock vara alternativ om bara två-tre sökkriterier har index.

Dynamisk SQL är bättre vid hög frekvens.



# Flervärda sökkriterier

Kommaseparerad lista (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)
```

(För funktioner för att packa upp listor till tabeller, se [\*Arrays and Lists in SQL Server 2005\*](#).)

# Flervärda sökkriterier

Tabellparameter (TVP) – kolla först om tabellen har data, och använd variabeln i frågan.

```
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)
```

# Flervärda sökparametrar, forts

Optimeraren har mindre information. För TVP enbart antal rader, för CSV inte ens det.

Selektivitet påverkar därför inte planen.

Studs på temptabell ger statistik med mer info till optimeraren (kan behöva UPDATE STATISTICS).

Om det kniper: med få värden, använd IN(@var1, @var2...), annars använd tabellen.

```
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)
```



# Alternativa tabeller

När @ishistoric = 1, läs historiska ordertabeller.

```
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
```

Fungerar – men mer komplex kod. Med många tabeller kan det gå överstyr.

# Styra sorteringen

Kan lösas med CASE – se upp med datatyper!

```
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
```

Om man vill sortera på fler kolumner, välja stigande/fallande, spårar det fort ur.

# Sökning på olika nycklar

Slå upp kund på endera kundid, skatteregno eller namn. Index på alla tre kolumnerna.

Vad tror ni om:

```
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)
```

Ingen OPTION (RECOMPILE)!

# Startfilter

Planen har sökning på alla tre indexen med ett startfilter – bara ett index används åt gången.

Villkoret @val IS NOT NULL måste vara med!

Fungerar sällan om kolumner är i olika tabeller.

Testa alltid att planen blir den tänkta.



# Andra exempel på startfilter

```
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
```

# Sökningar med dynamisk SQL

Högre svårighetsgrad.

Kräver mer disciplin av programmeraren.

Svårare att underhålla – om dåligt skriven.

Svårare att testa: finns risk att udda kombinationer smäller.

Men ger mycket större flexibilitet.

# Dynamisk SQL och behörigheter

Med dynamisk SQL gäller **inte** "ownership chaining" – användarna måste ha direkt behörighet på tabellerna.

Kan lösas genom att signera SP:n med certifikat eller med EXECUTE AS. Se

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

*(Granting Permissions through Stored Procedures)*

# sp\_executesql

sp\_executesql

Kärnan i alla sökningar med dynamisk SQL.  
Skapar en namnlös SP som sparas i cachén.  
SP:n identifieras av en hash på querytexten utan normalisering av blanka, stora/små etc.

Första parameteren:	@sqlstring. <b>nvarchar!</b>
Andra parameteren:	@paramlist. <b>nvarchar!</b>
Därefter:	Parametrar i @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
```

Användare kan ha olika default-schema.  
Förenklar att hänga på villkor.  
Gör det lättare att läsa den genererade SQL:n.

# Addera villkor

```
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
```

# Flervärda parametrar

```
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
```

# Debug-parametern

Denna rad ska alltid finnas med när man sysslar med dynamisk SQL

```
IF @debug = 1  
    PRINT @sql
```

# ALLTID!



# dynamic\_search\_1 – anrop

```
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
```

# Ett dåligt exempel

Somliga bygger in alla värden i SQL-strängen:

```
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](#)

Öppnar för SQL injection!

# Cache och kompilering

## OPTION (RECOMPILE)

Kompilering varje gång.

Onödigt mycket kompilering.

Planen passar alltid parametrarna för stunden.

## Dynamisk SQL m. parametrar

Cachad plan per kombination.

Mycket mindre kompilering.

”Parameter sniffing” kan bli ett problem.

# Cache och kompilering

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

3 kompileringar  
1 (oanvänt) cache-entry

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

1 kompilering  
1 cache-entry

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

3 kompileringar  
3 cache-entryn



# Problem för parameteriserad SQL

Cachad plan är bra – men inte alltid.

[compare\\_1](#)

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

Samma parametrar, men bästa index beror på värdena.

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

Filtrerade index kan inte användas.

# Vad kan vi göra åt detta?

## 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
```

## Ändra texten beroende på parametervärden:

```
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 '
                        ...
```

# Specialbehandla vanliga fall:

```
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
```

# Filtrerade index:

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

# Konkatenera in värden?

Kan vara lämpligt för kolumner med få möjliga värden med väldigt olika fördelning.

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

Olämpligt för kund, city etc eftersom cachén skräpas ner. (Om inte speciellt för Stora Kunden.)

Listor: packa upp och gör ny lista. Återigen, bara om få möjliga värden.

# Välja sortering

Så här enkelt?

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

Så praktiskt – @sortcol kan ju vara en lista!

Nej! Risk för SQL injection. Använd quotename!

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

Men då fungerar inte listor, utan man behöver @sortcol1, @sortcol2 etc. Alt. koda av @sortcol.



# Välja sortering, forts

Så det här är nog bättre trots allt:

```
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
```

Fast bäst är kanske att sortera i klienten...

# Alternativa tabeller

Som med sortering – skicka inte in tabellnamn, utan översätt till dbo.tblname osv.

Implementera @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
```

[dynamic\\_search\\_3](#)

# GROUP BY, aggregering, välja kolumner osv

Det finns en gräns även för dynamisk SQL i en SP.

Problemet: hur uttrycka detta i en parameterlista?

Enklare att göra allt i klienten där man kan vara mer objektorienterad.

**Absolut inte skicka SQL-syntax till en SP!**

# Sammanfattning

Statisk SQL med `OPTION (RECOMPILE)` – enkelt för enkla fall, men komplexiteten växer snabbt.

Dynamisk SQL – onödigt krångligt för enkla fall, men krångligheten växer långsammare.

Avvägning från fall till fall vilket man ska välja.

# Tack för idag!

Erland Sommarskog

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

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

Presentationen + alla skript, inklusive för att skapa databasen.