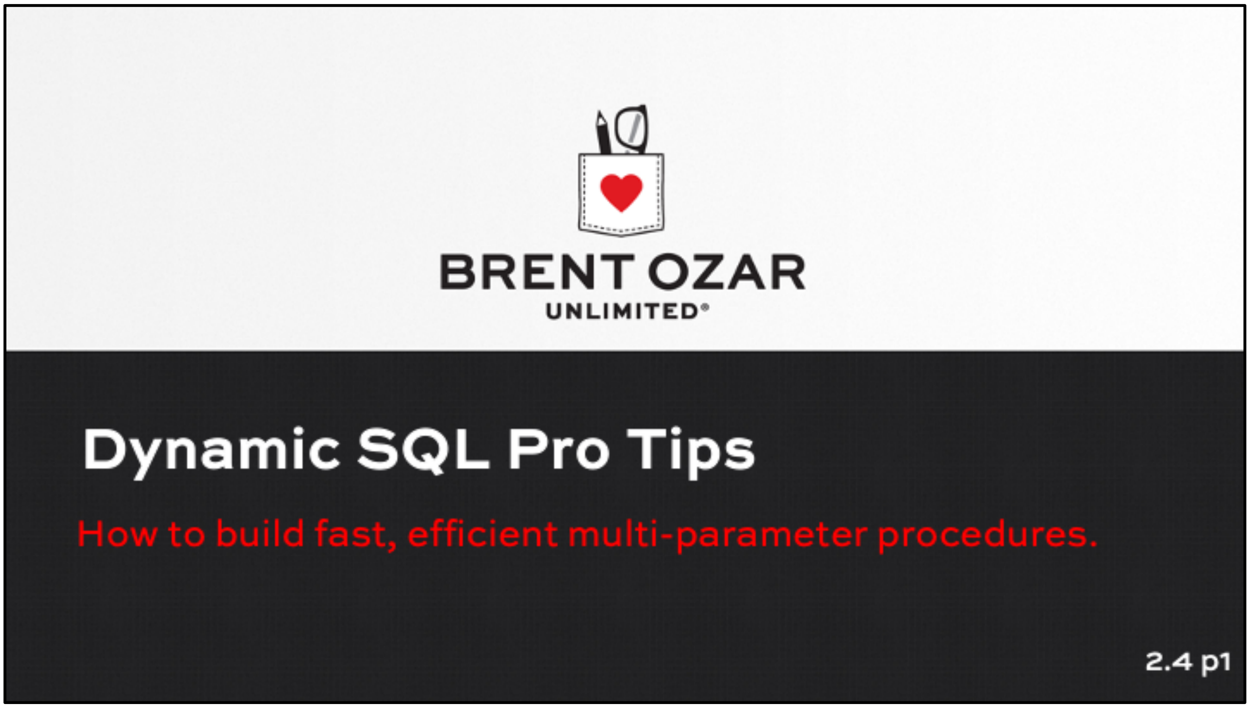
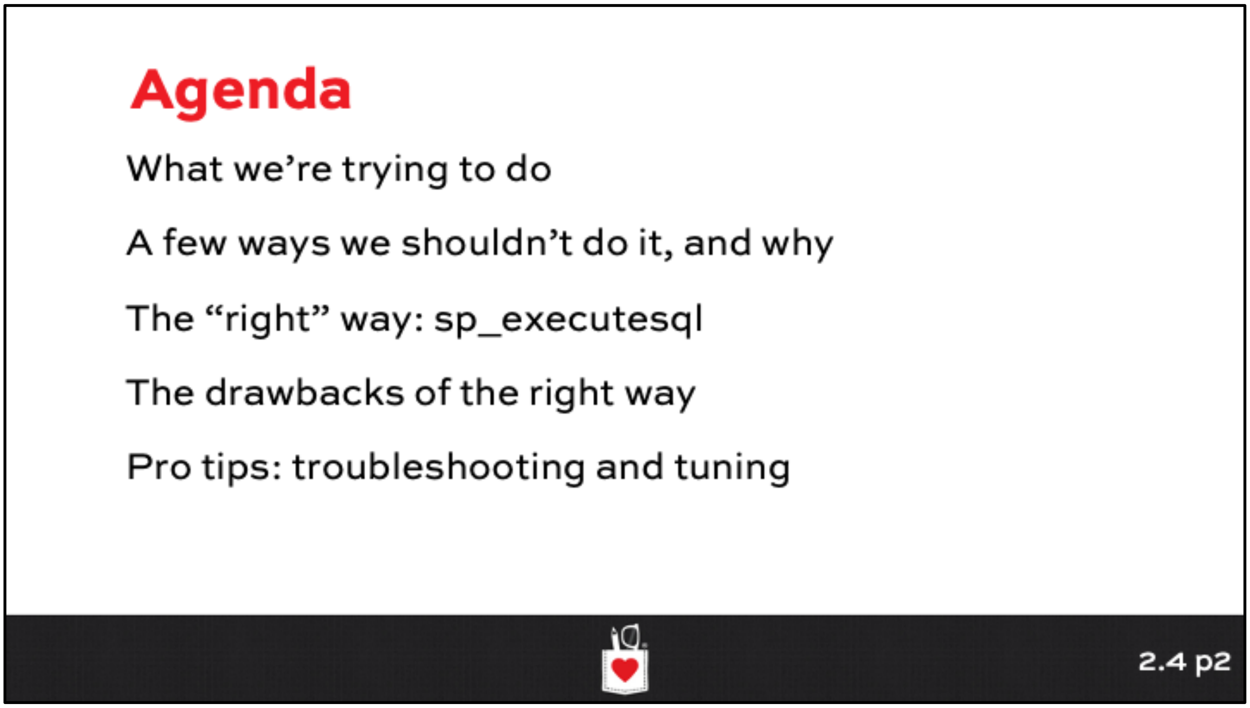


BRENT OZAR
UNLIMITED®

Dynamic SQL Pro Tips

How to build fast, efficient multi-parameter procedures.

2.4 p1



Agenda

What we're trying to do

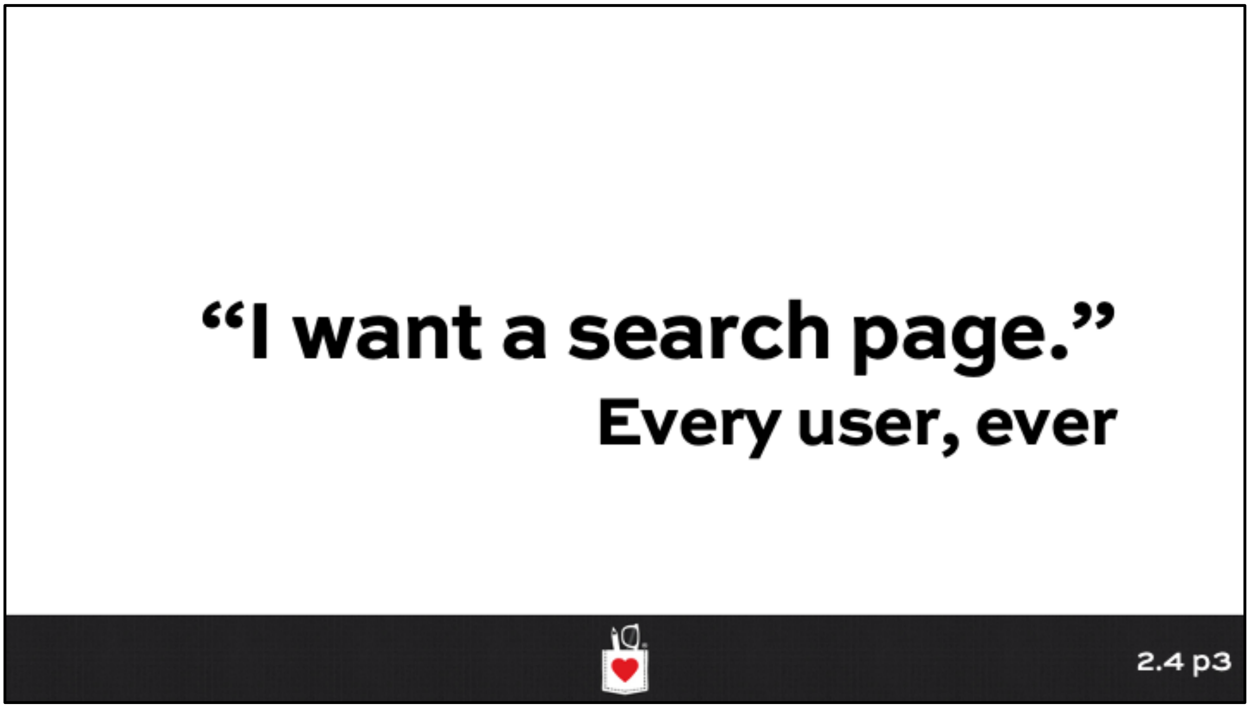
A few ways we shouldn't do it, and why

The "right" way: sp_executesql

The drawbacks of the right way

Pro tips: troubleshooting and tuning

2.4 p2



“I want a search page.”

Every user, ever

2.4 p3

stack overflow

Q&A site: you ask, other people do your job

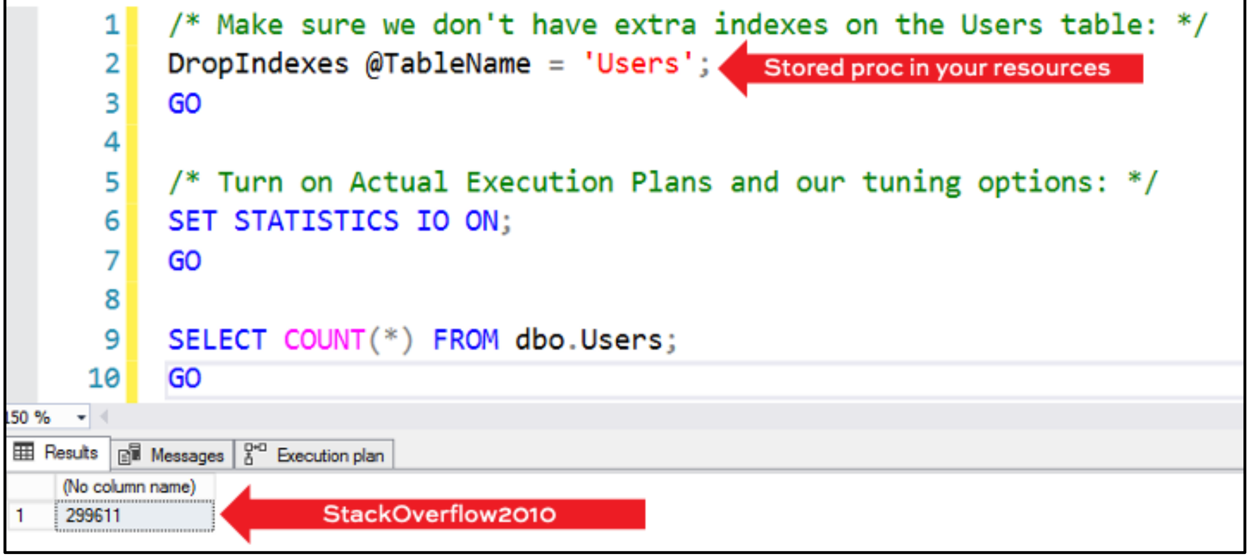
Whole database is available under Creative Commons

Download it free: BrentOzar.com/go/querystack

We’ll use the dbo.Users table

2.4 p4

How big is our Users table today?

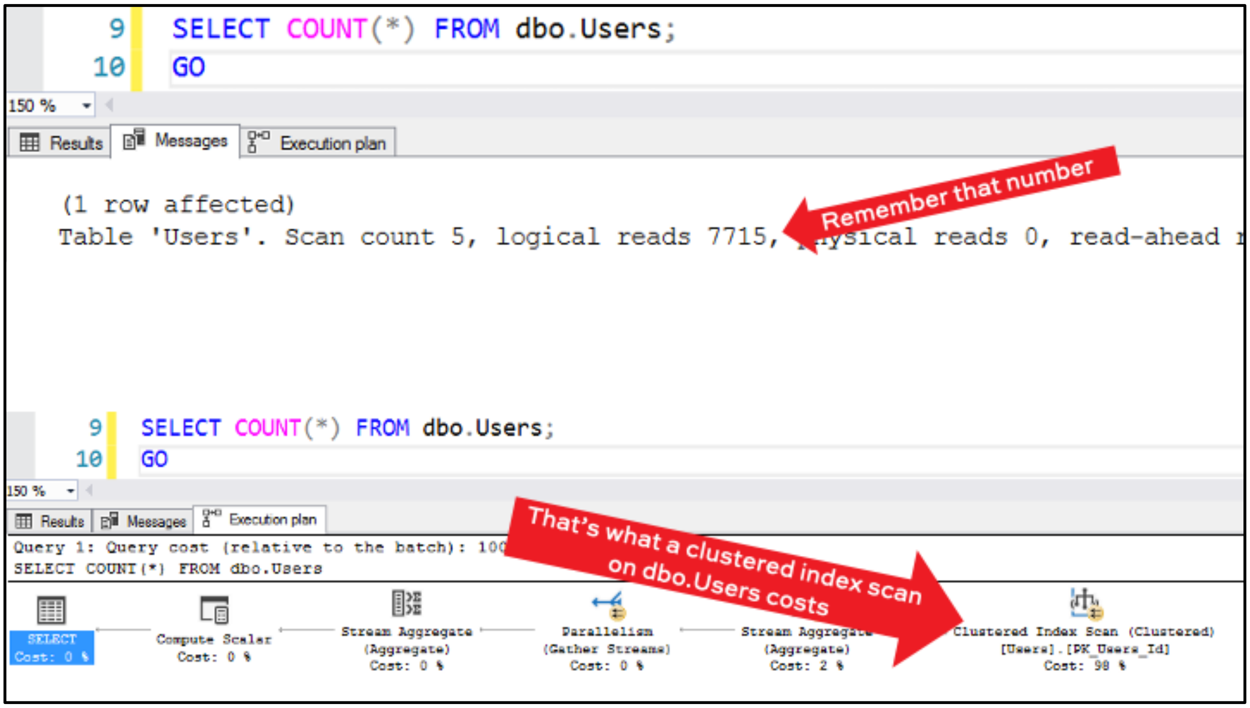


```sql
/* Make sure we don't have extra indexes on the Users table: */
DropIndexes @TableName = 'Users';
GO

/* Turn on Actual Execution Plans and our tuning options: */
SET STATISTICS IO ON;
GO

SELECT COUNT(*) FROM dbo.Users;
GO
```

Stored proc in your resources

```
   (No column name)
1  299611
```

StackOverflow2010



```sql
SELECT COUNT(*) FROM dbo.Users;
GO
```

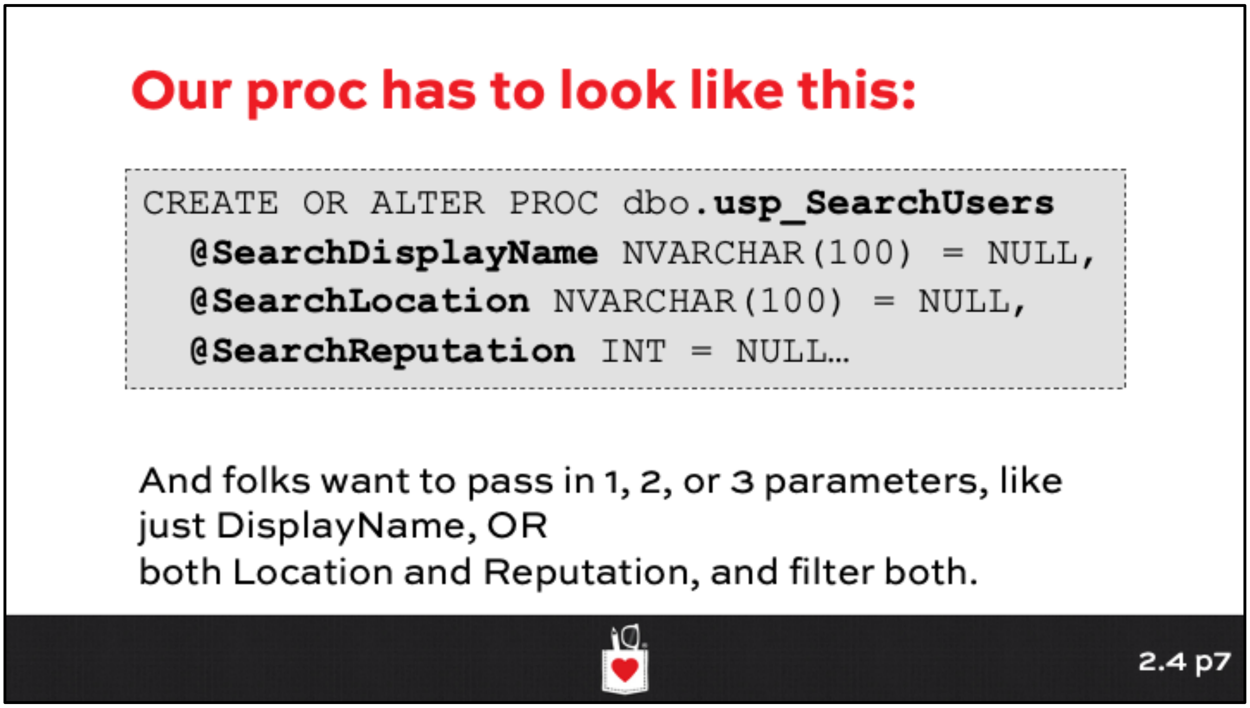
```
(1 row affected)
Table 'Users'. Scan count 5, logical reads 7715, physical reads 0, read-ahead r
```

Remember that number

```sql
SELECT COUNT(*) FROM dbo.Users;
GO
```

```
Query 1: Query cost (relative to the batch): 100
SELECT COUNT(*) FROM dbo.Users
```

That's what a clustered index scan on dbo.Users costs

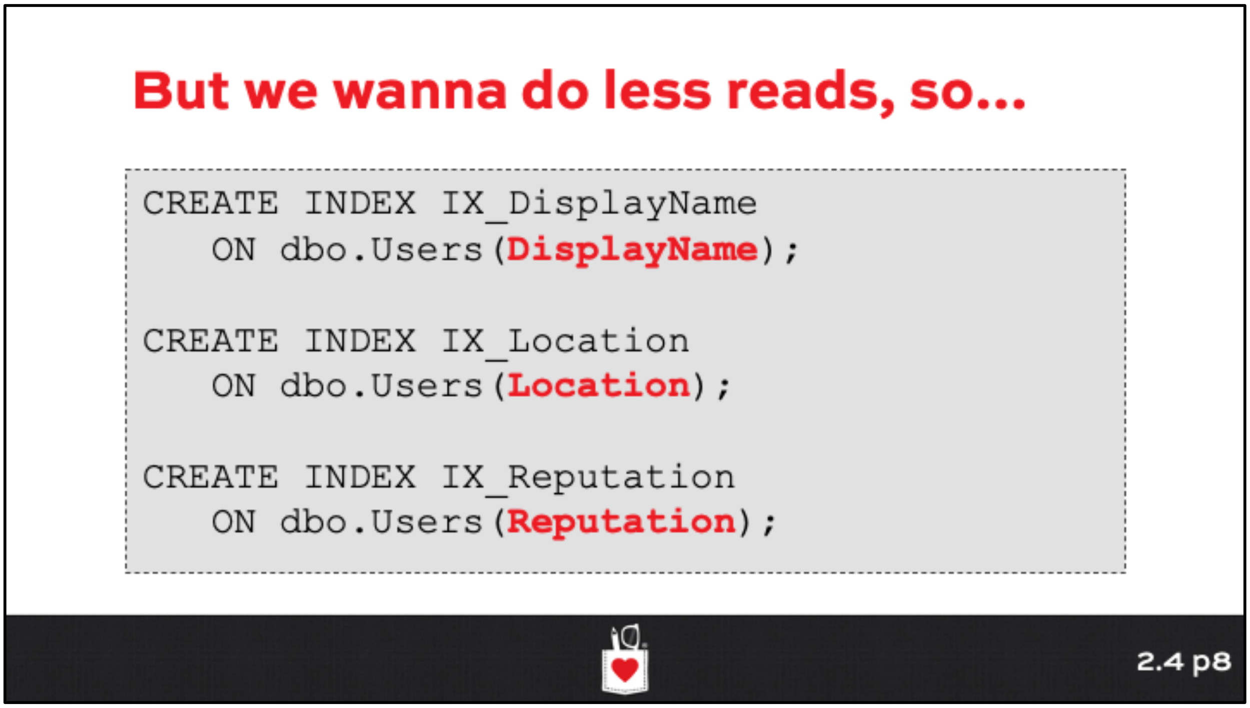


Our proc has to look like this:

```
CREATE OR ALTER PROC dbo.usp_SearchUsers
  @SearchDisplayName NVARCHAR(100) = NULL,
  @SearchLocation NVARCHAR(100) = NULL,
  @SearchReputation INT = NULL…
```

And folks want to pass in 1, 2, or 3 parameters, like
just DisplayName, OR
both Location and Reputation, and filter both.

2.4 p7



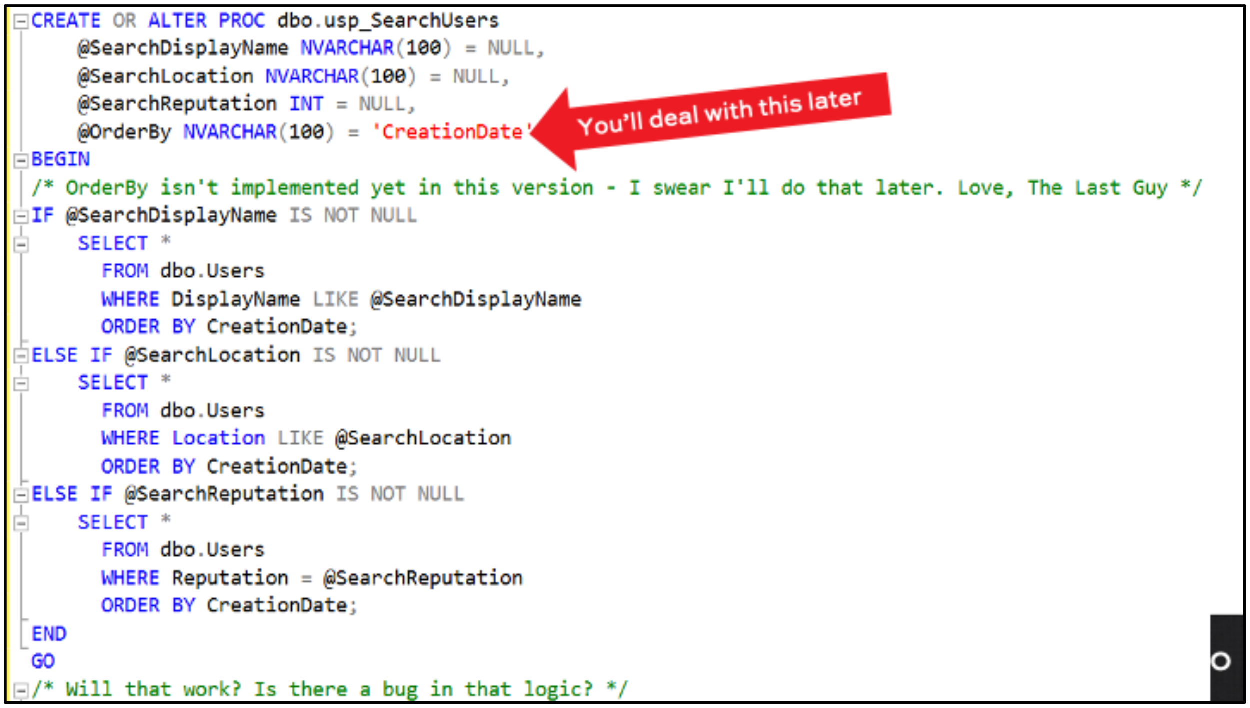
But we wanna do less reads, so…

```
CREATE INDEX IX_DisplayName
   ON dbo.Users(DisplayName);

CREATE INDEX IX_Location
   ON dbo.Users(Location);

CREATE INDEX IX_Reputation
   ON dbo.Users(Reputation);
```

2.4 p8

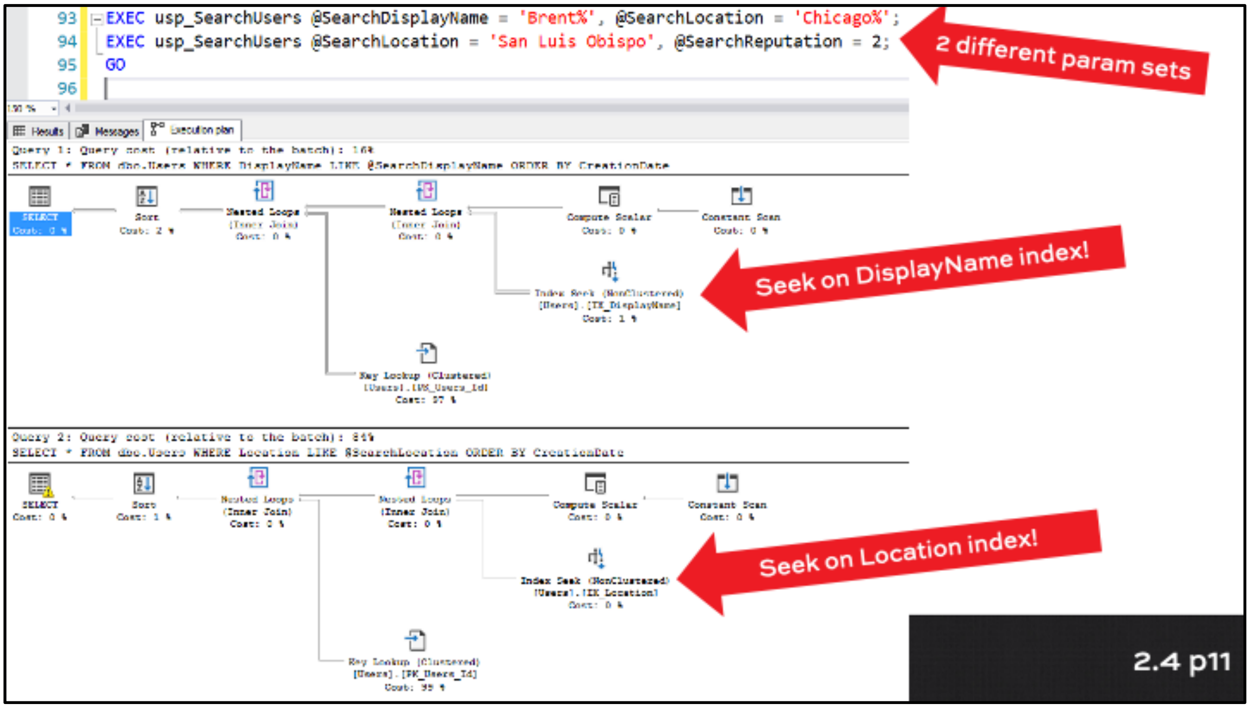
Version 1:
the really bad idea

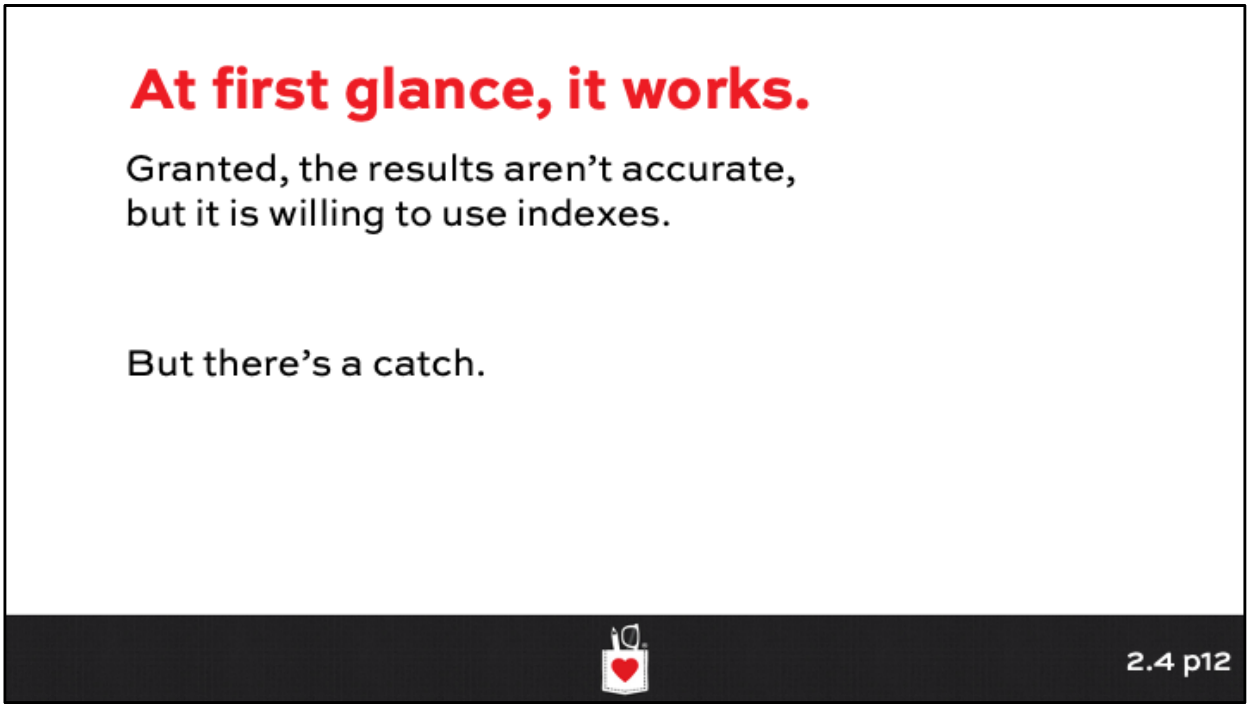
2.4 p9



```
CREATE OR ALTER PROC dbo.usp_SearchUsers
    @SearchDisplayName NVARCHAR(100) = NULL,
    @SearchLocation NVARCHAR(100) = NULL,
    @SearchReputation INT = NULL,
    @OrderBy NVARCHAR(100) = 'CreationDate'
BEGIN
/* OrderBy isn't implemented yet in this version - I swear I'll do that later. Love, The Last Guy */
IF @SearchDisplayName IS NOT NULL
    SELECT *
      FROM dbo.Users
      WHERE DisplayName LIKE @SearchDisplayName
      ORDER BY CreationDate;
ELSE IF @SearchLocation IS NOT NULL
    SELECT *
      FROM dbo.Users
      WHERE Location LIKE @SearchLocation
      ORDER BY CreationDate;
ELSE IF @SearchReputation IS NOT NULL
    SELECT *
      FROM dbo.Users
      WHERE Reputation = @SearchReputation
      ORDER BY CreationDate;
END
GO
/* Will that work? Is there a bug in that logic? */
```

You'll deal with this later



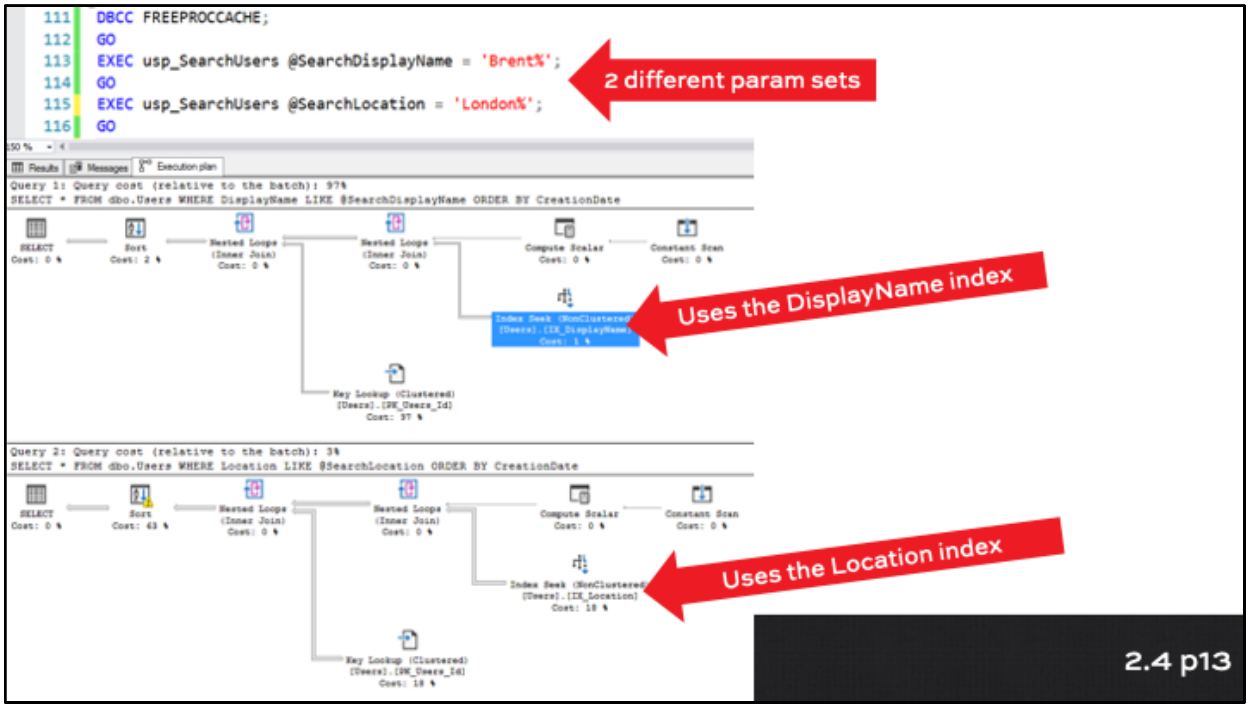


At first glance, it works.

Granted, the results aren't accurate,
but it is willing to use indexes.

But there's a catch.

2.4 p12



```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

2 different param sets

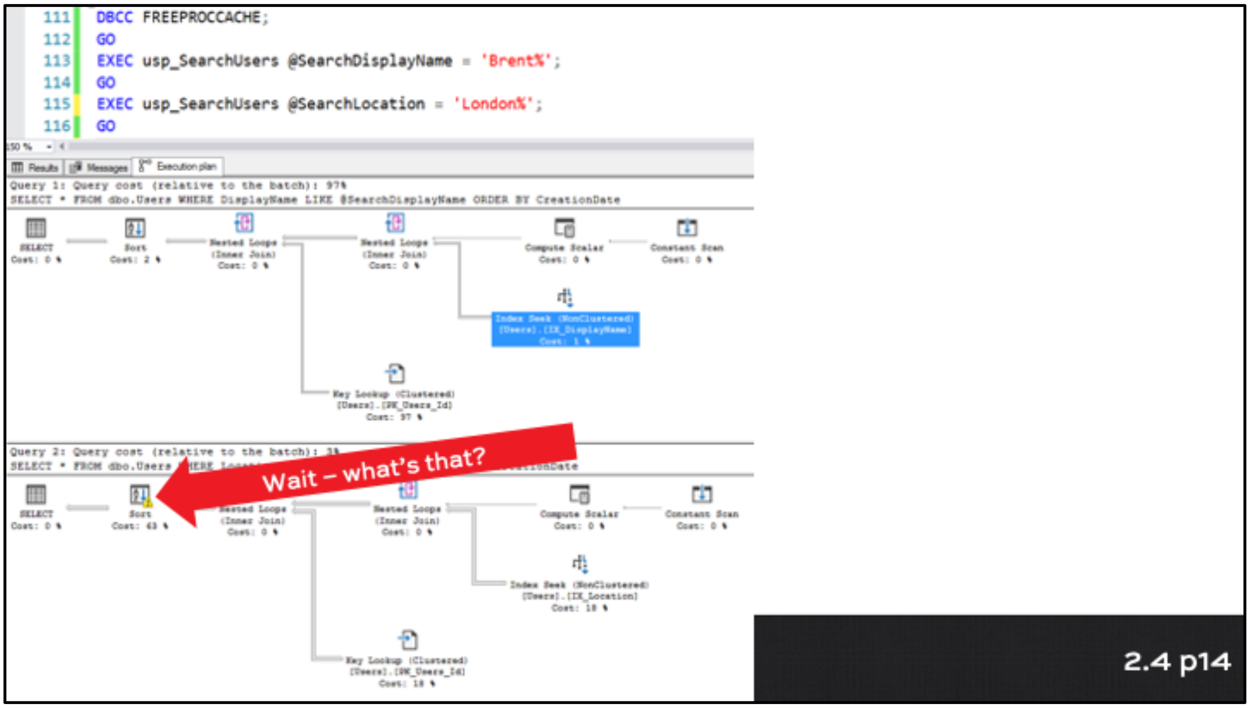
```
Query 1: Query cost (relative to the batch): 97%
SELECT * FROM dbo.Users WHERE DisplayName LIKE @SearchDisplayName ORDER BY CreationDate
```

Uses the DisplayName index

```
Query 2: Query cost (relative to the batch): 3%
SELECT * FROM dbo.Users WHERE Location LIKE @SearchLocation ORDER BY CreationDate
```

Uses the Location index

2.4 p13



```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

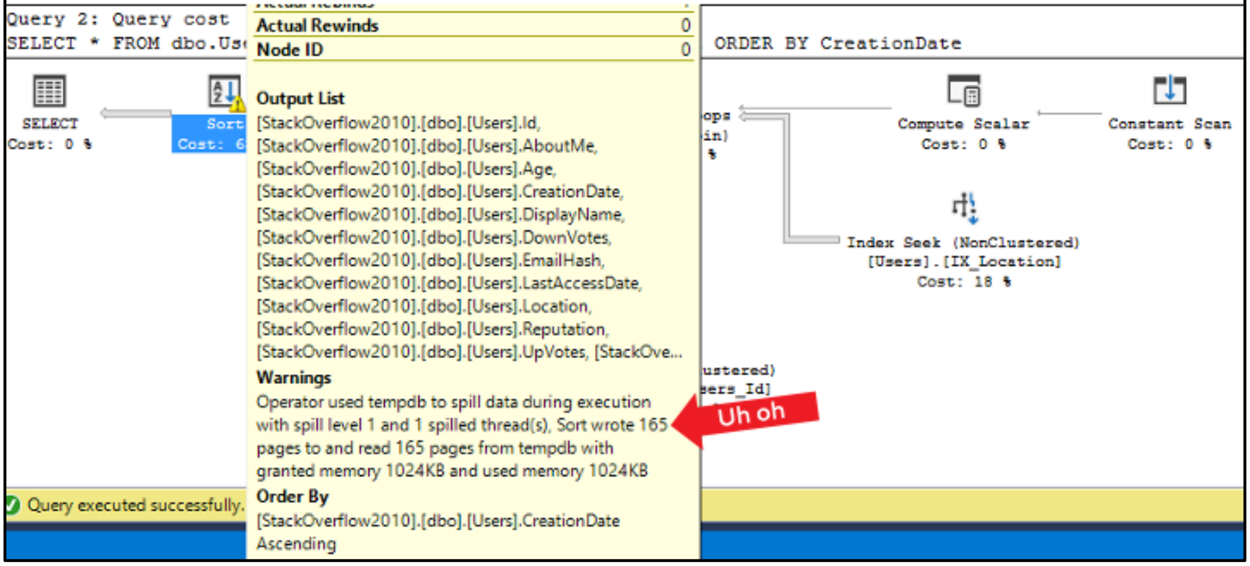
```
Query 1: Query cost (relative to the batch): 97%
SELECT * FROM dbo.Users WHERE DisplayName LIKE @SearchDisplayName ORDER BY CreationDate
```

```
Query 2: Query cost (relative to the batch): 3%
SELECT * FROM dbo.Users WHERE Locat                                 tionDate
```

Wait – what's that?

2.4 p14

The London sort is spilling to disk



```
Query 2: Query cost
SELECT * FROM dbo.Use
ORDER BY CreationDate
```

Actual Rewinds	0
Node ID	0

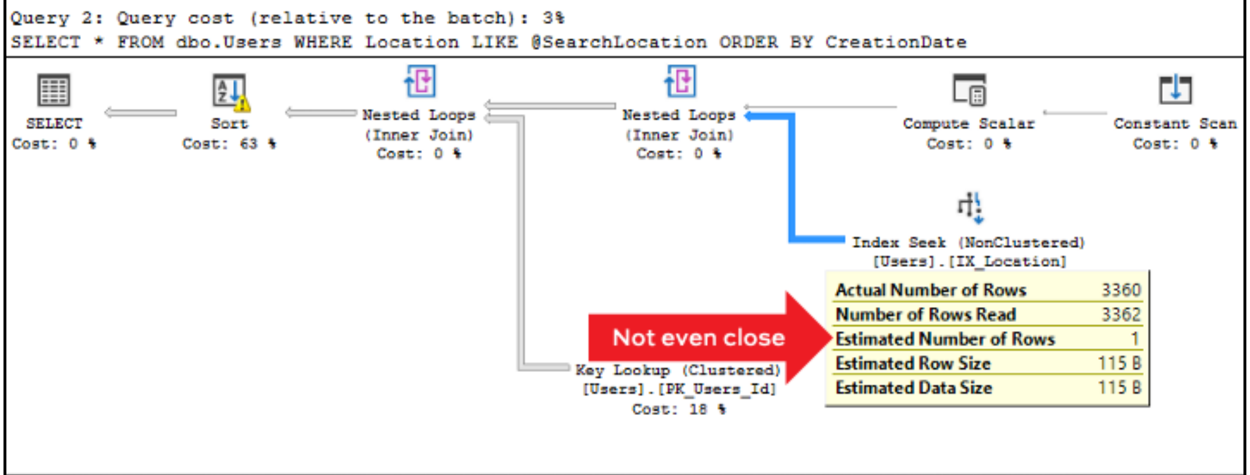
Output List
[StackOverflow2010].[dbo].[Users].Id, [StackOverflow2010].[dbo].[Users].AboutMe, [StackOverflow2010].[dbo].[Users].Age, [StackOverflow2010].[dbo].[Users].CreationDate, [StackOverflow2010].[dbo].[Users].DisplayName, [StackOverflow2010].[dbo].[Users].DownVotes, [StackOverflow2010].[dbo].[Users].EmailHash, [StackOverflow2010].[dbo].[Users].LastAccessDate, [StackOverflow2010].[dbo].[Users].Location, [StackOverflow2010].[dbo].[Users].Reputation, [StackOverflow2010].[dbo].[Users].UpVotes, [StackOve...

Warnings
Operator used tempdb to spill data during execution with spill level 1 and 1 spilled thread(s), Sort wrote 165 pages to and read 165 pages from tempdb with granted memory 1024KB and used memory 1024KB

Order By
[StackOverflow2010].[dbo].[Users].CreationDate Ascending

Query executed successfully.

Because we underestimated rows



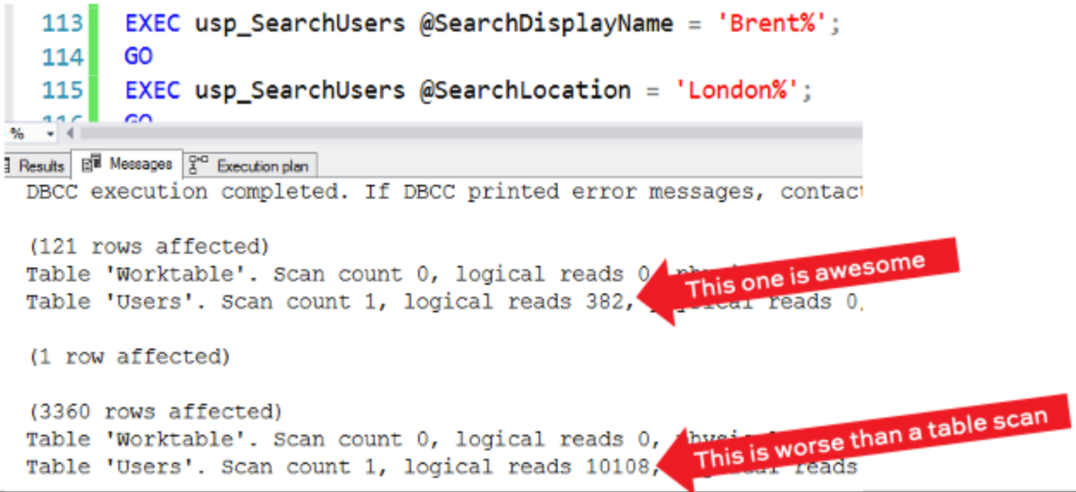
```
Query 2: Query cost (relative to the batch): 3%
SELECT * FROM dbo.Users WHERE Location LIKE @SearchLocation ORDER BY CreationDate
```

Not even close

Actual Number of Rows	3360
Number of Rows Read	3362
Estimated Number of Rows	1
Estimated Row Size	115 B
Estimated Data Size	115 B

2.4 p16

Remember, table has ~7000 pages



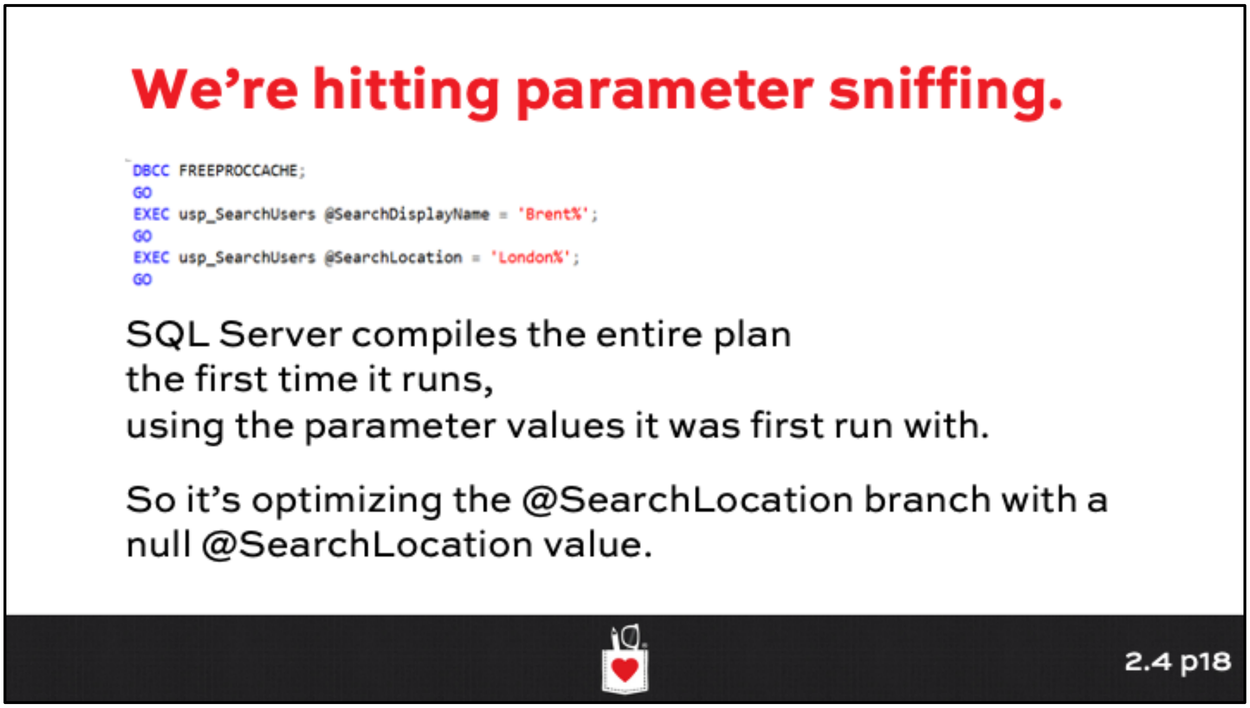
```
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

```
DBCC execution completed. If DBCC printed error messages, contact

(121 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0,
Table 'Users'. Scan count 1, logical reads 382,            reads 0,

(1 row affected)

(3360 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0,
Table 'Users'. Scan count 1, logical reads 10108,            reads
```

This one is awesome

This is worse than a table scan

2.4 p17



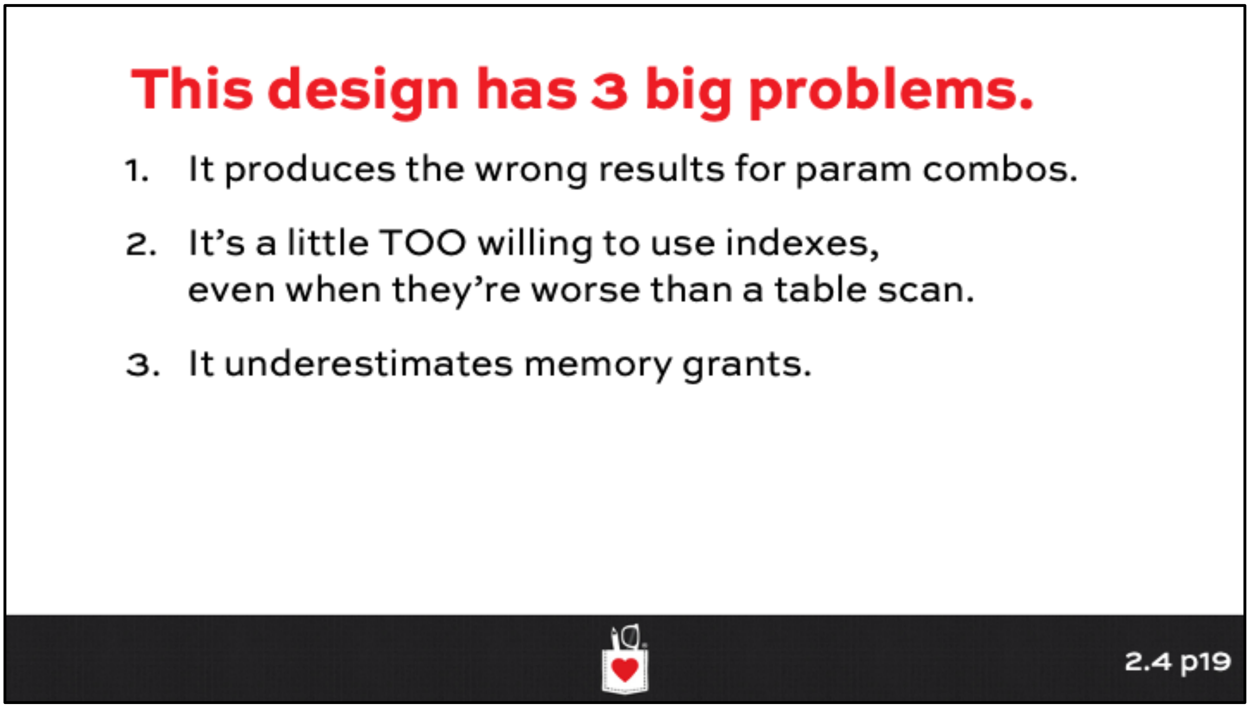
We’re hitting parameter sniffing.

```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

SQL Server compiles the entire plan
the first time it runs,
using the parameter values it was first run with.

So it’s optimizing the @SearchLocation branch with a null @SearchLocation value.

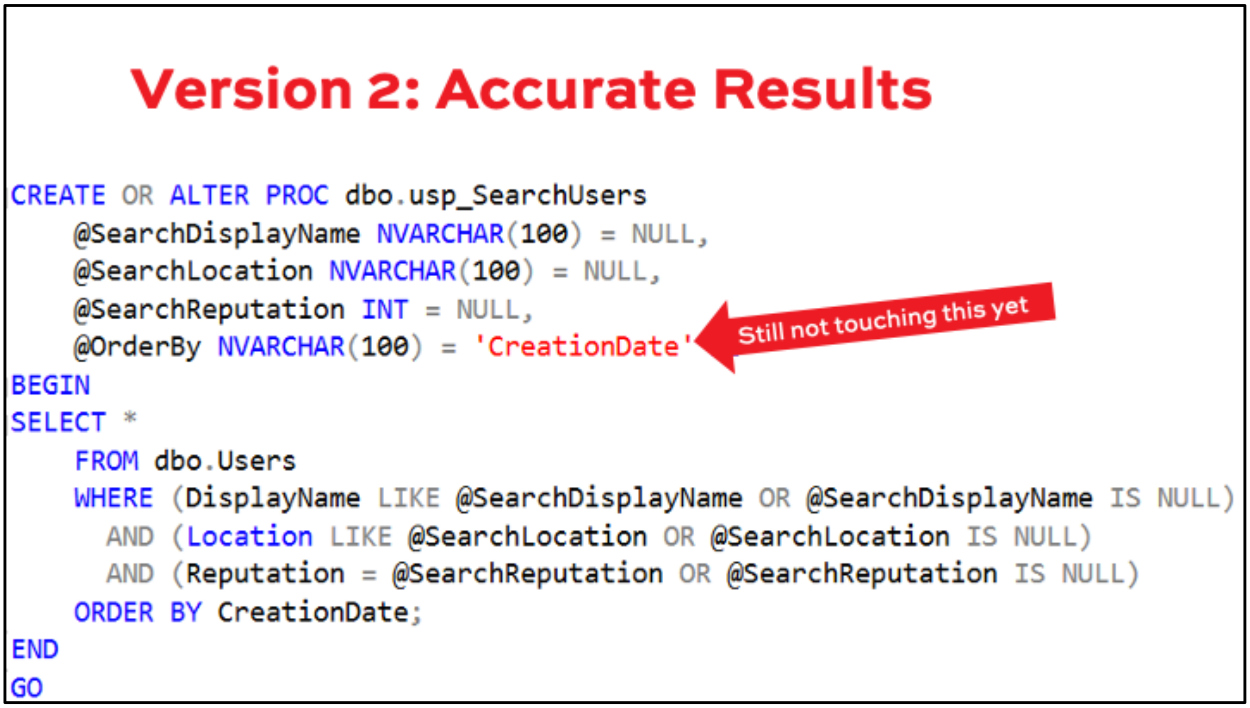
2.4 p18



This design has 3 big problems.

1. It produces the wrong results for param combos.
2. It's a little TOO willing to use indexes,
   even when they're worse than a table scan.
3. It underestimates memory grants.

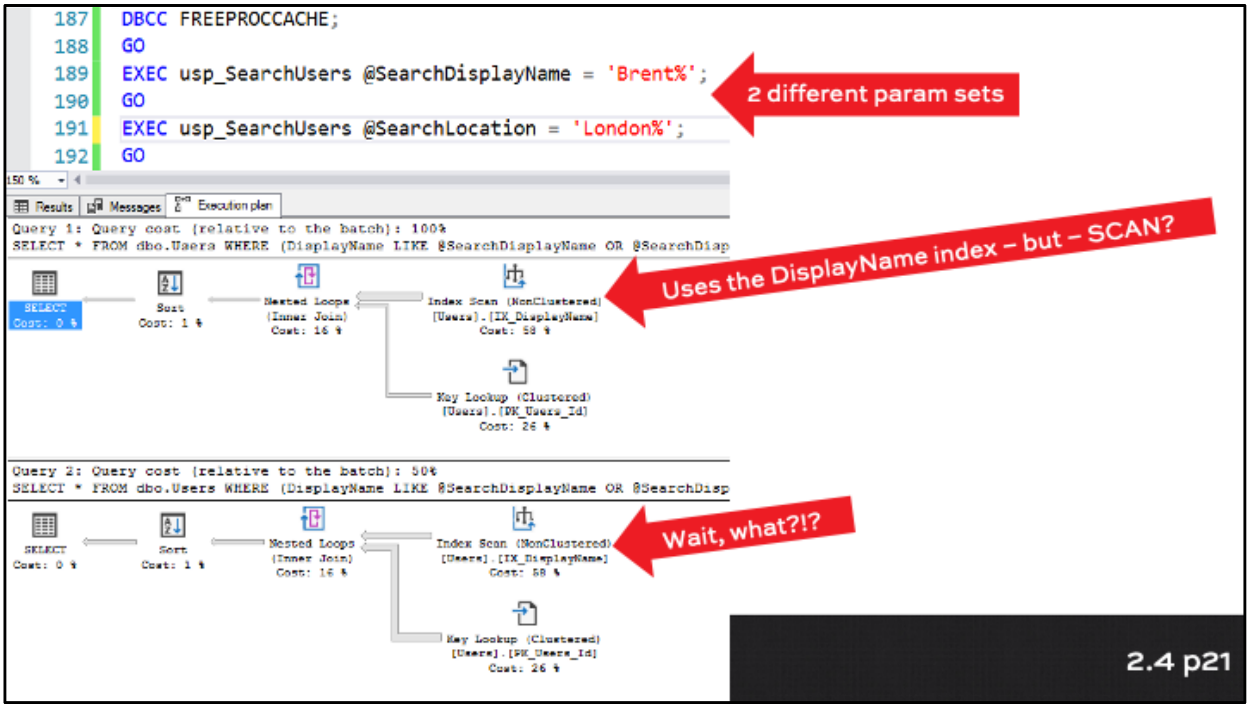
2.4 p19



Version 2: Accurate Results

```
CREATE OR ALTER PROC dbo.usp_SearchUsers
    @SearchDisplayName NVARCHAR(100) = NULL,
    @SearchLocation NVARCHAR(100) = NULL,
    @SearchReputation INT = NULL,
    @OrderBy NVARCHAR(100) = 'CreationDate'
BEGIN
SELECT *
    FROM dbo.Users
    WHERE (DisplayName LIKE @SearchDisplayName OR @SearchDisplayName IS NULL)
      AND (Location LIKE @SearchLocation OR @SearchLocation IS NULL)
      AND (Reputation = @SearchReputation OR @SearchReputation IS NULL)
    ORDER BY CreationDate;
END
GO
```

Still not touching this yet



```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE (DisplayName LIKE @SearchDisplayName OR @SearchDisp
```

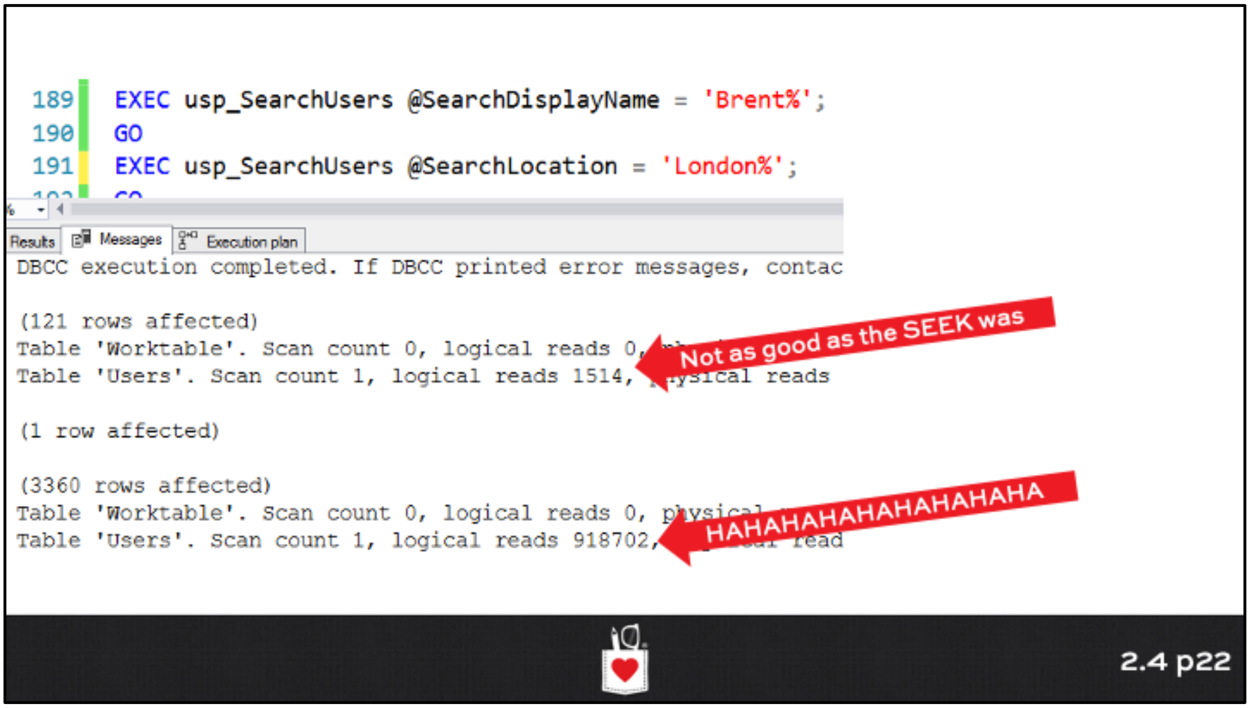
```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Users WHERE (DisplayName LIKE @SearchDisplayName OR @SearchDisp
```

2 different param sets

Uses the DisplayName index – but – SCAN?

Wait, what?!?

2.4 p21



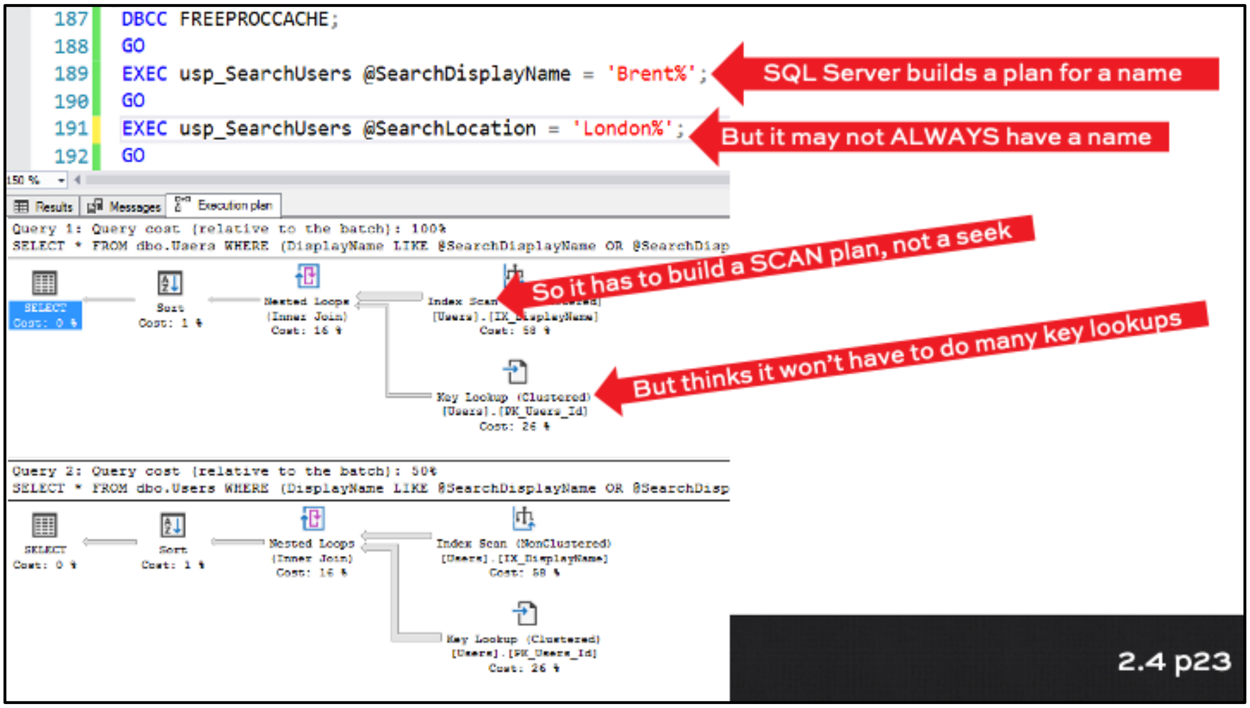
```
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

```
DBCC execution completed. If DBCC printed error messages, contac

(121 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, ph
Table 'Users'. Scan count 1, logical reads 1514, physical reads

(1 row affected)

(3360 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical
Table 'Users'. Scan count 1, logical reads 918702, physical read
```

Not as good as the SEEK was

HAHAHAHAHAHAHAHAHA

2.4 p22



```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

SQL Server builds a plan for a name

But it may not ALWAYS have a name

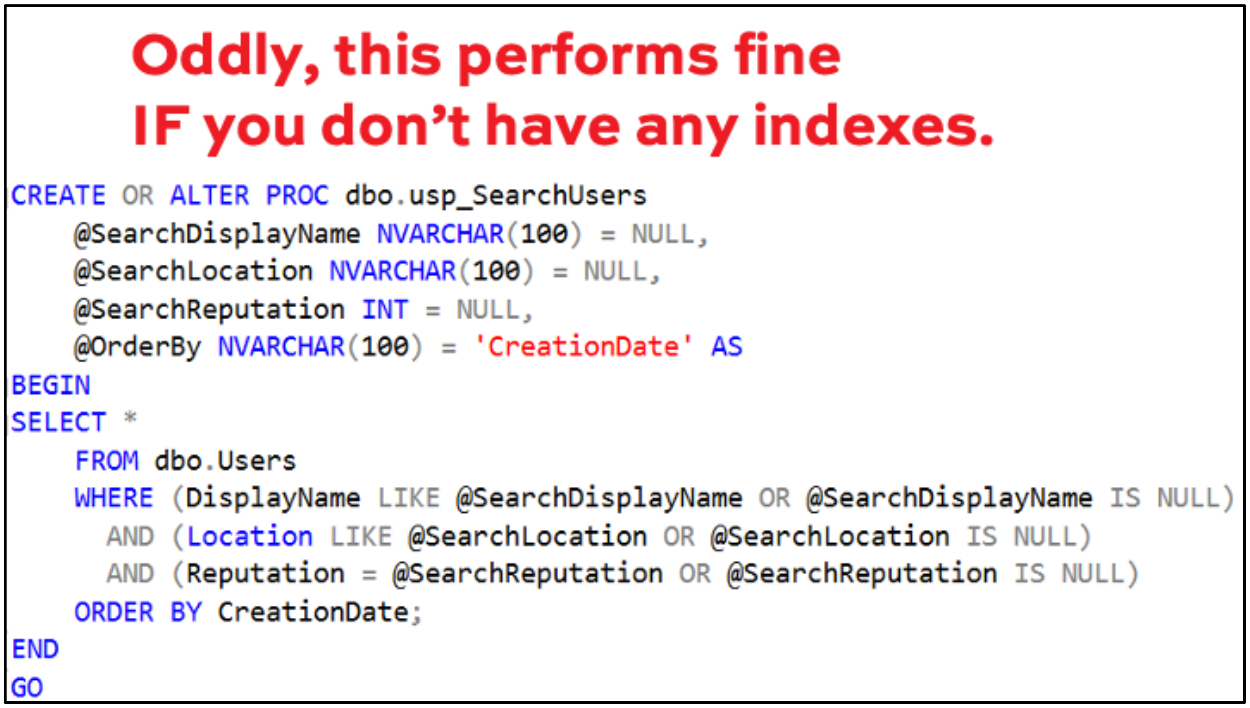
```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE (DisplayName LIKE @SearchDisplayName OR @SearchDisp
```

So it has to build a SCAN plan, not a seek

But thinks it won't have to do many key lookups

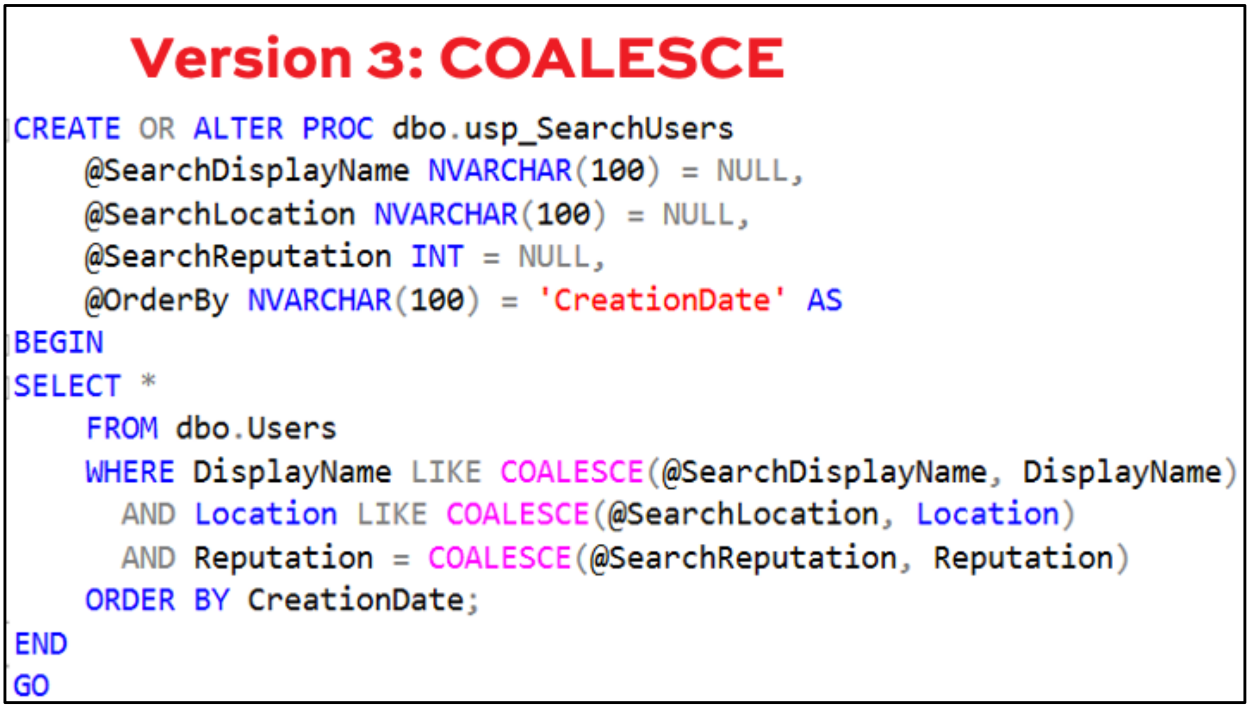
```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Users WHERE (DisplayName LIKE @SearchDisplayName OR @SearchDisp
```

2.4 p23



Oddly, this performs fine IF you don't have any indexes.

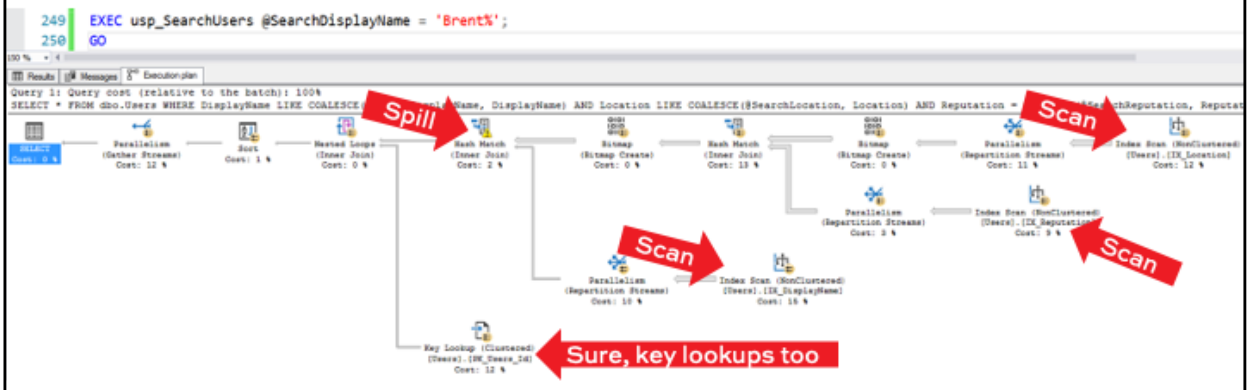
```sql
CREATE OR ALTER PROC dbo.usp_SearchUsers
    @SearchDisplayName NVARCHAR(100) = NULL,
    @SearchLocation NVARCHAR(100) = NULL,
    @SearchReputation INT = NULL,
    @OrderBy NVARCHAR(100) = 'CreationDate' AS
BEGIN
SELECT *
    FROM dbo.Users
    WHERE (DisplayName LIKE @SearchDisplayName OR @SearchDisplayName IS NULL)
      AND (Location LIKE @SearchLocation OR @SearchLocation IS NULL)
      AND (Reputation = @SearchReputation OR @SearchReputation IS NULL)
    ORDER BY CreationDate;
END
GO
```



Version 3: COALESCE

```sql
CREATE OR ALTER PROC dbo.usp_SearchUsers
    @SearchDisplayName NVARCHAR(100) = NULL,
    @SearchLocation NVARCHAR(100) = NULL,
    @SearchReputation INT = NULL,
    @OrderBy NVARCHAR(100) = 'CreationDate' AS
BEGIN
SELECT *
    FROM dbo.Users
    WHERE DisplayName LIKE COALESCE(@SearchDisplayName, DisplayName)
      AND Location LIKE COALESCE(@SearchLocation, Location)
      AND Reputation = COALESCE(@SearchReputation, Reputation)
    ORDER BY CreationDate;
END
GO
```

Welcome to Scandinavia

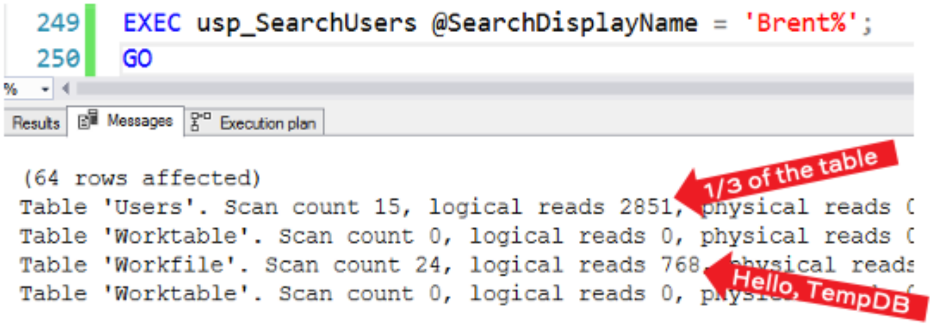


```
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE DisplayName LIKE COALESCE(@SearchDisplayName, DisplayName) AND Location LIKE COALESCE(@SearchLocation, Location) AND Reputation = COALESCE(@SearchReputation, Reputat
```

2.4 p26

Not great on reads, either



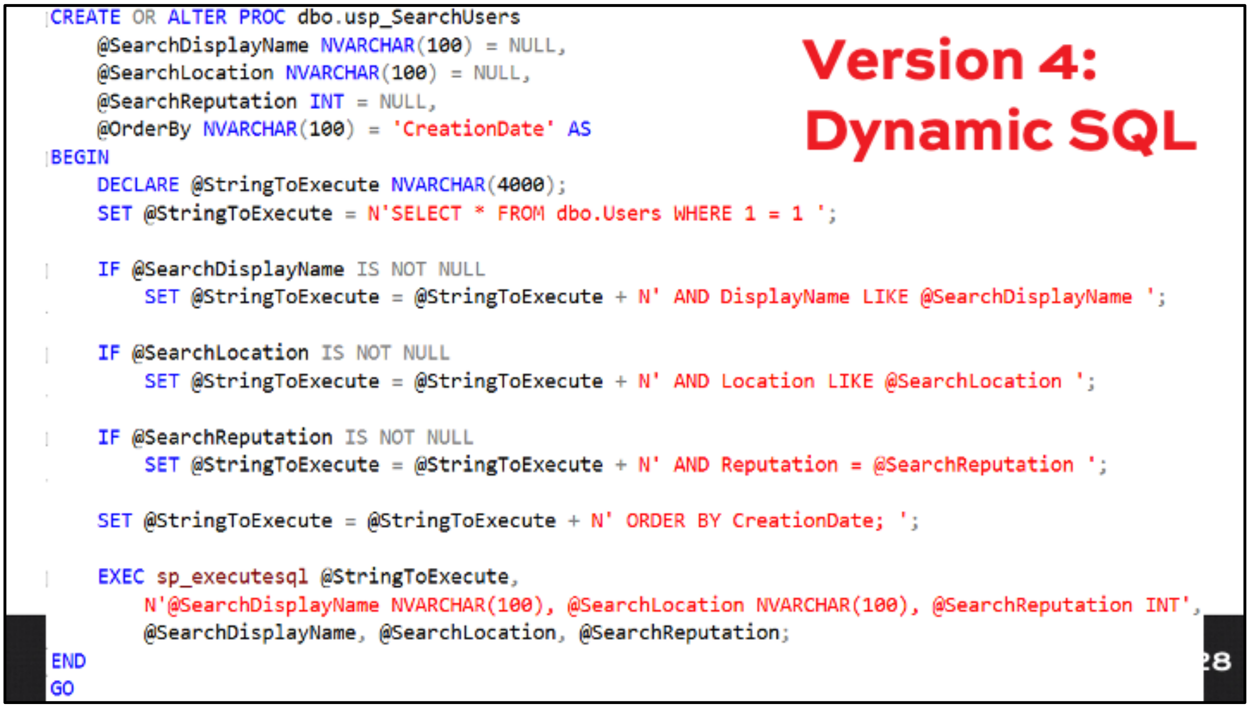
```
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
```

```
(64 rows affected)
Table 'Users'. Scan count 15, logical reads 2851, physical reads (
Table 'Worktable'. Scan count 0, logical reads 0, physical reads (
Table 'Workfile'. Scan count 24, logical reads 768, physical reads
Table 'Worktable'. Scan count 0, logical reads 0, physical
```

1/3 of the table

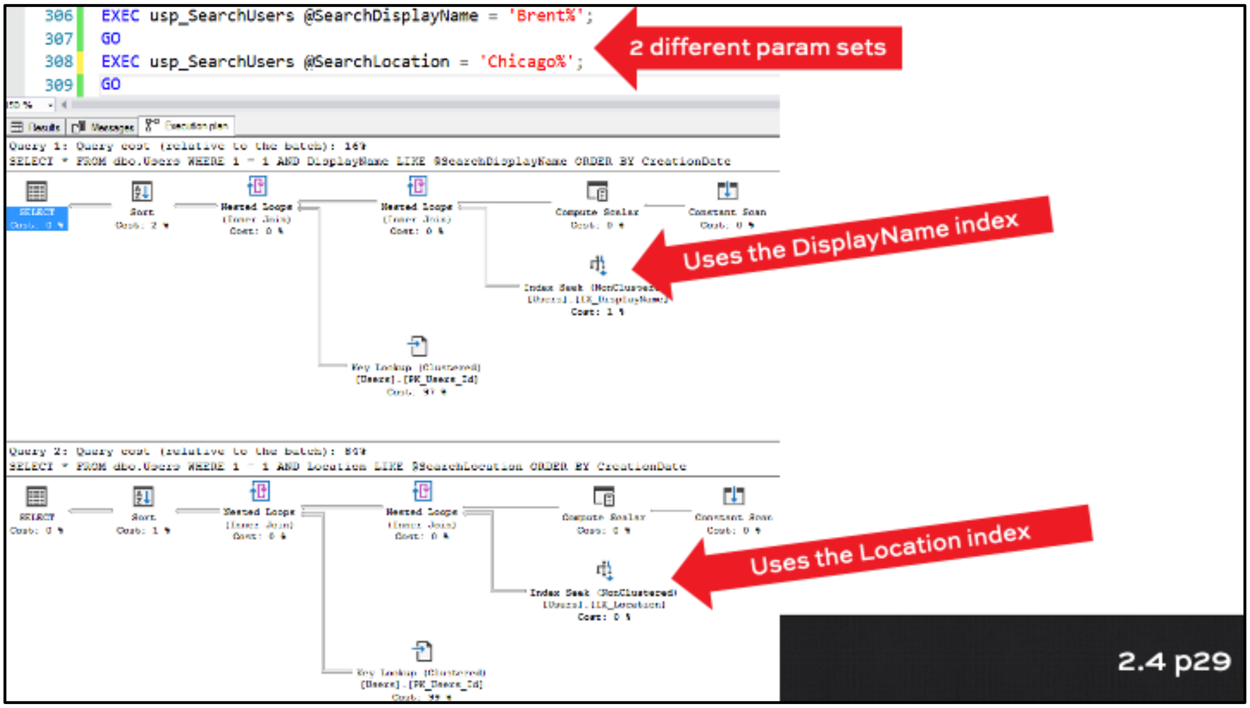
Hello, TempDB

2.4 p27



Version 4: Dynamic SQL

```
CREATE OR ALTER PROC dbo.usp_SearchUsers
    @SearchDisplayName NVARCHAR(100) = NULL,
    @SearchLocation NVARCHAR(100) = NULL,
    @SearchReputation INT = NULL,
    @OrderBy NVARCHAR(100) = 'CreationDate' AS
BEGIN
    DECLARE @StringToExecute NVARCHAR(4000);
    SET @StringToExecute = N'SELECT * FROM dbo.Users WHERE 1 = 1 ';

    IF @SearchDisplayName IS NOT NULL
        SET @StringToExecute = @StringToExecute + N' AND DisplayName LIKE @SearchDisplayName ';

    IF @SearchLocation IS NOT NULL
        SET @StringToExecute = @StringToExecute + N' AND Location LIKE @SearchLocation ';

    IF @SearchReputation IS NOT NULL
        SET @StringToExecute = @StringToExecute + N' AND Reputation = @SearchReputation ';

    SET @StringToExecute = @StringToExecute + N' ORDER BY CreationDate; ';

    EXEC sp_executesql @StringToExecute,
        N'@SearchDisplayName NVARCHAR(100), @SearchLocation NVARCHAR(100), @SearchReputation INT',
        @SearchDisplayName, @SearchLocation, @SearchReputation;
END
GO
```



```
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'Chicago%';
GO
```

```
Query 1: Query cost (relative to the batch): 16%
SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName ORDER BY CreationDate
```

```
Query 2: Query cost (relative to the batch): 84%
SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate
```

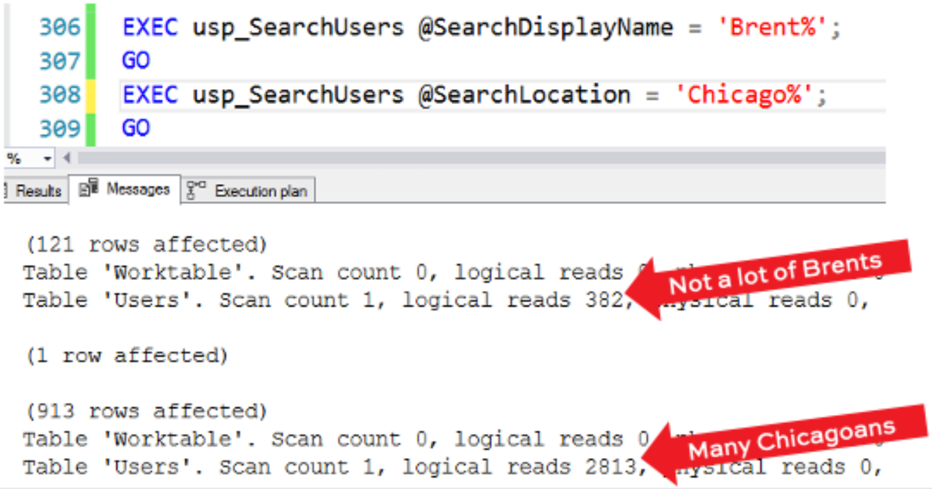
2 different param sets

Uses the DisplayName index

Uses the Location index

2.4 p29

Logical reads look good, too

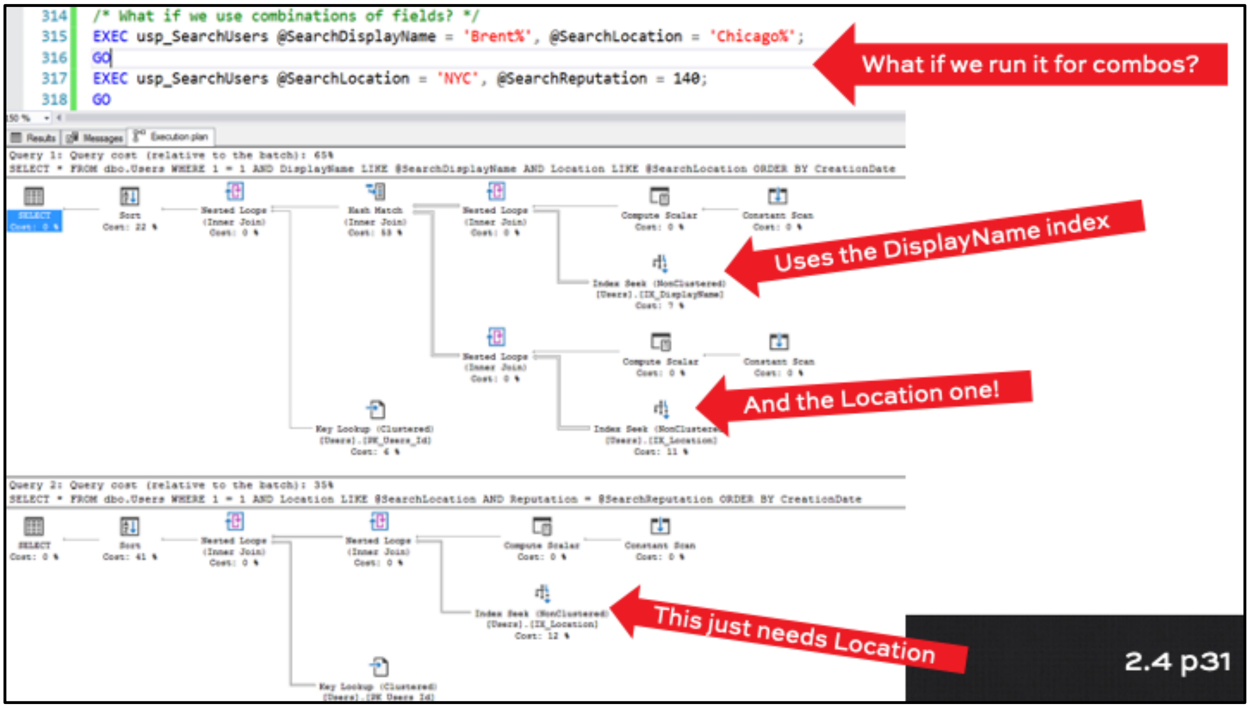


```
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'Chicago%';
GO
```

```
(121 rows affected)
Table 'Worktable'. Scan count 0, logical reads 
Table 'Users'. Scan count 1, logical reads 382,          reads 0,

(1 row affected)

(913 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0
Table 'Users'. Scan count 1, logical reads 2813,           reads 0,
```

Not a lot of Brents

Many Chicagoans



```
/* What if we use combinations of fields? */
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%', @SearchLocation = 'Chicago%';
GO
EXEC usp_SearchUsers @SearchLocation = 'NYC', @SearchReputation = 140;
GO
```

```
Query 1: Query cost (relative to the batch): 65%
SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName AND Location LIKE @SearchLocation ORDER BY CreationDate
```

```
Query 2: Query cost (relative to the batch): 35%
SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation AND Reputation = @SearchReputation ORDER BY CreationDate
```

150 %
Results
Messages
Execution plan
SELECT Cost: 0 %
Sort Cost: 22 %
Nested Loops (Inner Join) Cost: 0 %
Hash Match (Inner Join) Cost: 53 %
Nested Loops (Inner Join) Cost: 0 %
Compute Scalar Cost: 0 %
Constant Scan Cost: 0 %
Index Seek (NonClustered) [Users].[IX_DisplayName] Cost: 7 %
Nested Loops (Inner Join) Cost: 0 %
Compute Scalar Cost: 0 %
Constant Scan Cost: 0 %
Key Lookup (Clustered) [Users].[PK_Users_Id] Cost: 4 %
Index Seek (NonClustered) [Users].[IX_Location] Cost: 11 %
SELECT Cost: 0 %
Sort Cost: 41 %
Nested Loops (Inner Join) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Compute Scalar Cost: 0 %
Constant Scan Cost: 0 %
Index Seek (NonClustered) [Users].[IX_Location] Cost: 12 %
Key Lookup (Clustered) [Users].[PK_Users_Id]

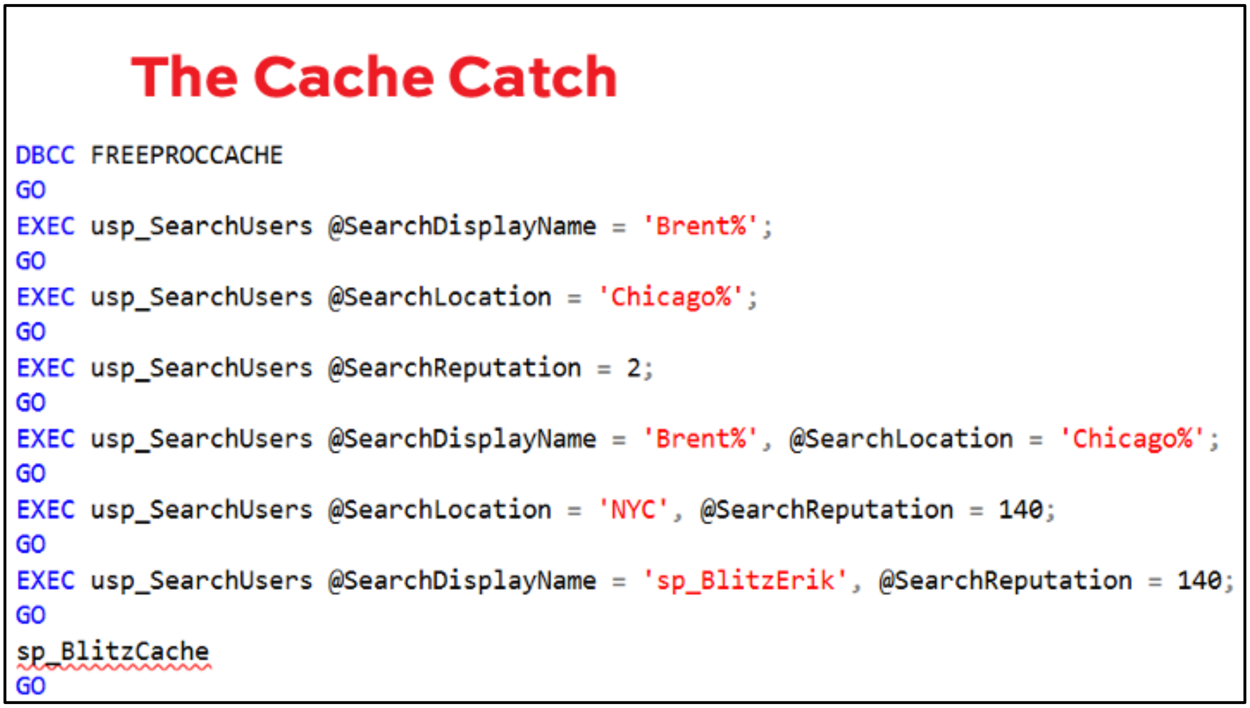
What if we run it for combos?

Uses the DisplayName index

And the Location one!

This just needs Location

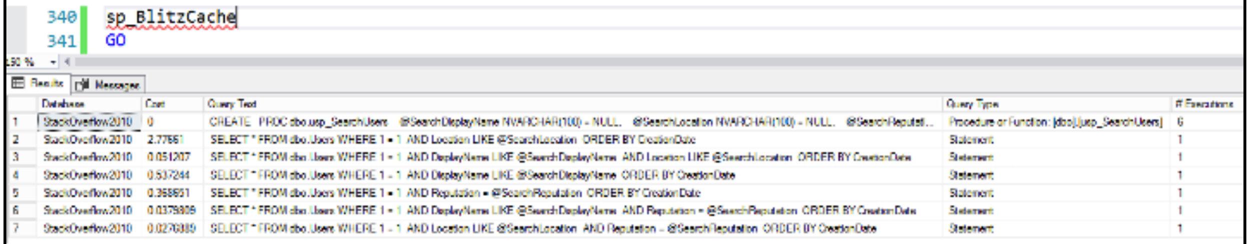
2.4 p31



The Cache Catch

```
DBCC FREEPROCCACHE
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%';
GO
EXEC usp_SearchUsers @SearchLocation = 'Chicago%';
GO
EXEC usp_SearchUsers @SearchReputation = 2;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'Brent%', @SearchLocation = 'Chicago%';
GO
EXEC usp_SearchUsers @SearchLocation = 'NYC', @SearchReputation = 140;
GO
EXEC usp_SearchUsers @SearchDisplayName = 'sp_BlitzErik', @SearchReputation = 140;
GO
sp_BlitzCache
GO
```

We bloat the plan cache a little.



```
sp_BlitzCache
GO
```

	Database	Cost	Query Text	Query Type	# Executions
1	StackOverflow2010	0	CREATE PROC dbo.usp_SearchUsers @SearchDisplayName NVARCHAR(100) = NULL, @SearchLocation NVARCHAR(100) = NULL, @SearchReputati...	Procedure or Function: [dbo].[usp_SearchUsers]	6
2	StackOverflow2010	2.77661	SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate	Statement	1
3	StackOverflow2010	0.051207	SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName AND Location LIKE @SearchLocation ORDER BY CreationDate	Statement	1
4	StackOverflow2010	0.537244	SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName ORDER BY CreationDate	Statement	1
5	StackOverflow2010	0.368651	SELECT * FROM dbo.Users WHERE 1 = 1 AND Reputation = @SearchReputation ORDER BY CreationDate	Statement	1
6	StackOverflow2010	0.0379809	SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName AND Reputation = @SearchReputation ORDER BY CreationDate	Statement	1
7	StackOverflow2010	0.0276389	SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation AND Reputation = @SearchReputation ORDER BY CreationDate	Statement	1

The proc has its own entry, executed 6 times.

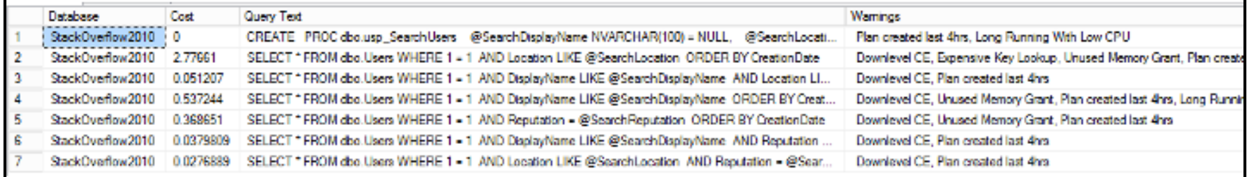
Each dynamic SQL string gets its own line.

But each dynamic plan is great* for that set of parameters!

* Not necessarily.

2.4 p33

I got 99 ~~problems~~ plans

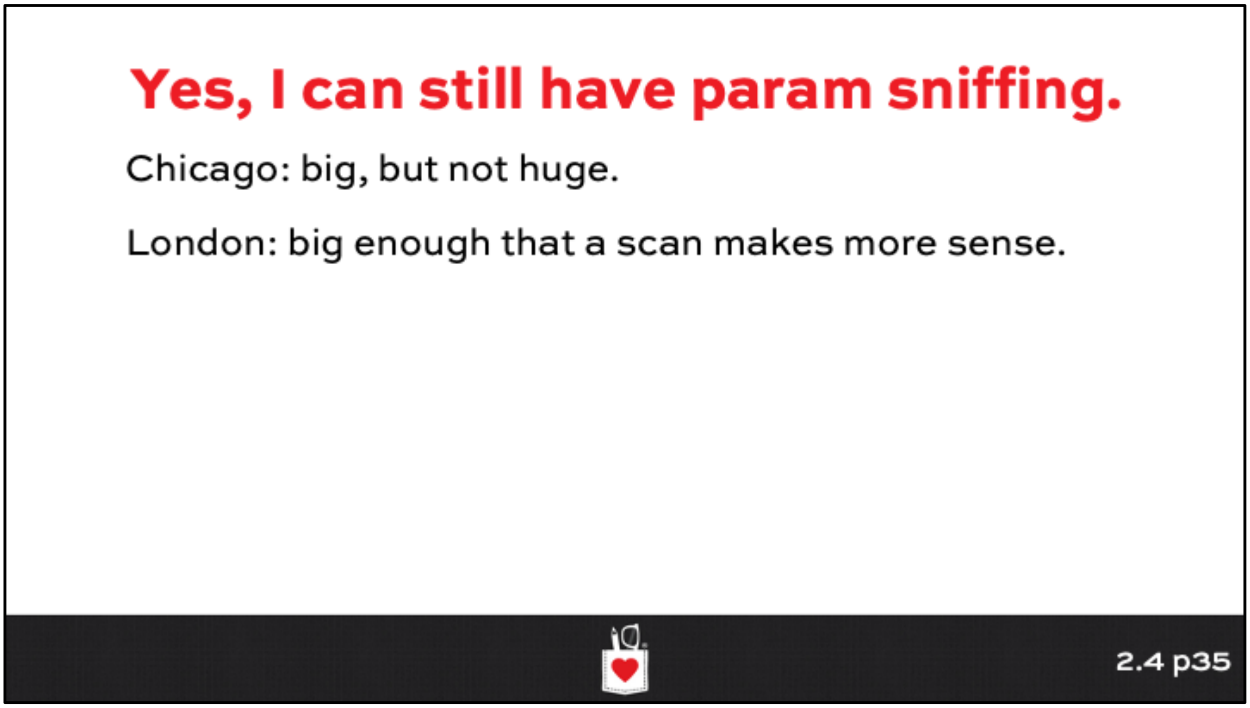


	Database	Cost	Query Text	Warnings
1	StackOverflow2010	0	CREATE PROC dbo.usp_SearchUsers @SearchDisplayName NVARCHAR(100) = NULL, @SearchLocati...	Plan created last 4hrs, Long Running With Low CPU
2	StackOverflow2010	2.77661	SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate	Downlevel CE, Expensive Key Lookup, Unused Memory Grant, Plan create
3	StackOverflow2010	0.051207	SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName AND Location LI...	Downlevel CE, Plan created last 4hrs
4	StackOverflow2010	0.537244	SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName ORDER BY Creat...	Downlevel CE, Unused Memory Grant, Plan created last 4hrs, Long Runnin
5	StackOverflow2010	0.368651	SELECT * FROM dbo.Users WHERE 1 = 1 AND Reputation = @SearchReputation ORDER BY CreationDate	Downlevel CE, Unused Memory Grant, Plan created last 4hrs
6	StackOverflow2010	0.0379809	SELECT * FROM dbo.Users WHERE 1 = 1 AND DisplayName LIKE @SearchDisplayName AND Reputation ...	Downlevel CE, Plan created last 4hrs
7	StackOverflow2010	0.0276889	SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation AND Reputation = @Sear...	Downlevel CE, Plan created last 4hrs

Each dynamic SQL plan has its own:

- Plan cache entry
- Memory grant
- Row estimations
- Parameter sniffing issues

2.4 p34

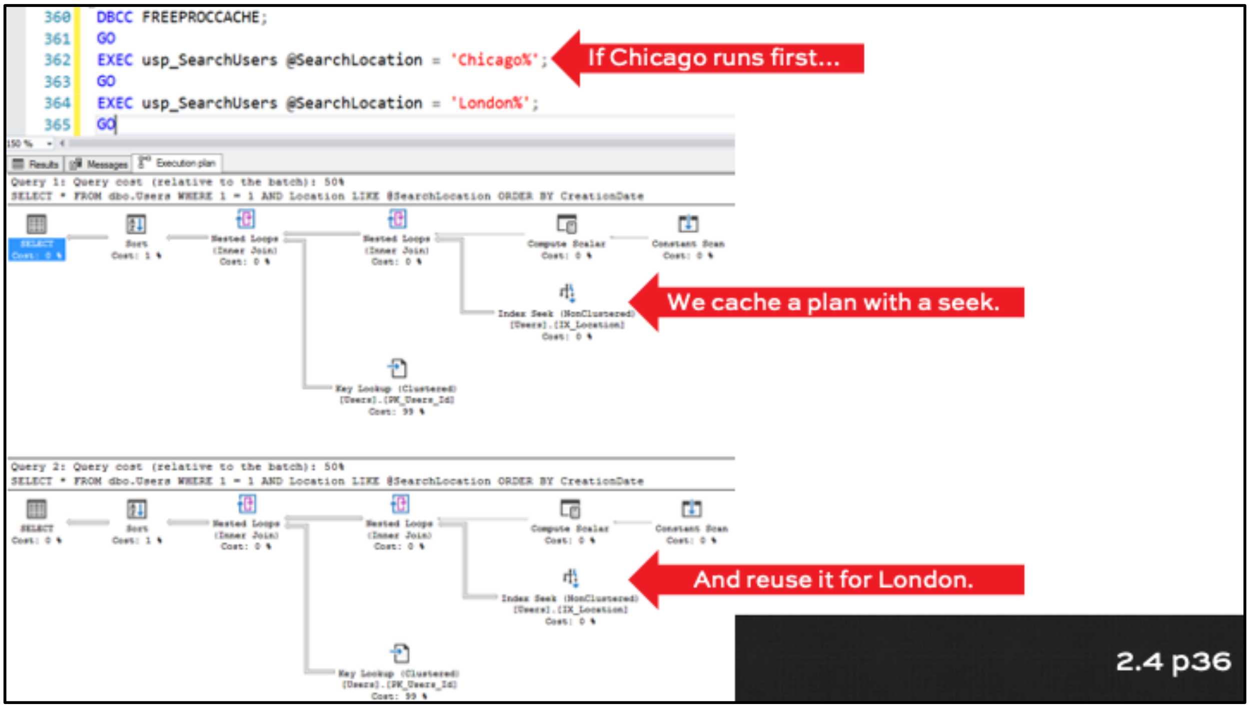


Yes, I can still have param sniffing.

Chicago: big, but not huge.

London: big enough that a scan makes more sense.

2.4 p35



```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchLocation = 'Chicago%';
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
```

If Chicago runs first…

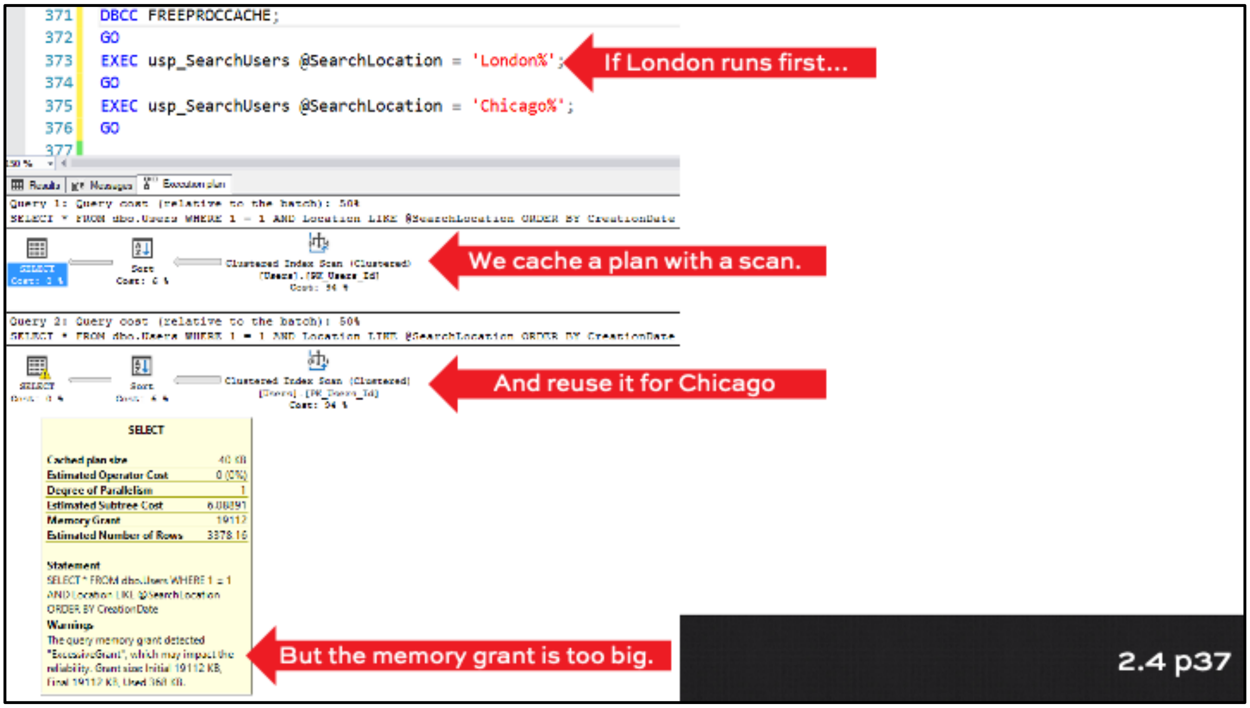
```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate
```

We cache a plan with a seek.

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate
```

And reuse it for London.

2.4 p36



```
DBCC FREEPROCCACHE;
GO
EXEC usp_SearchUsers @SearchLocation = 'London%';
GO
EXEC usp_SearchUsers @SearchLocation = 'Chicago%';
GO
```

```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate
```

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate
```

SELECT

Cached plan size	40 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	6.08891
Memory Grant	19112
Estimated Number of Rows	3378.16

Statement

SELECT * FROM dbo.Users WHERE 1 = 1 AND Location LIKE @SearchLocation ORDER BY CreationDate

Warnings

The query memory grant detected "ExcessiveGrant", which may impact the reliability. Grant size: Initial 19112 KB, Final 19112 KB, Used 368 KB.

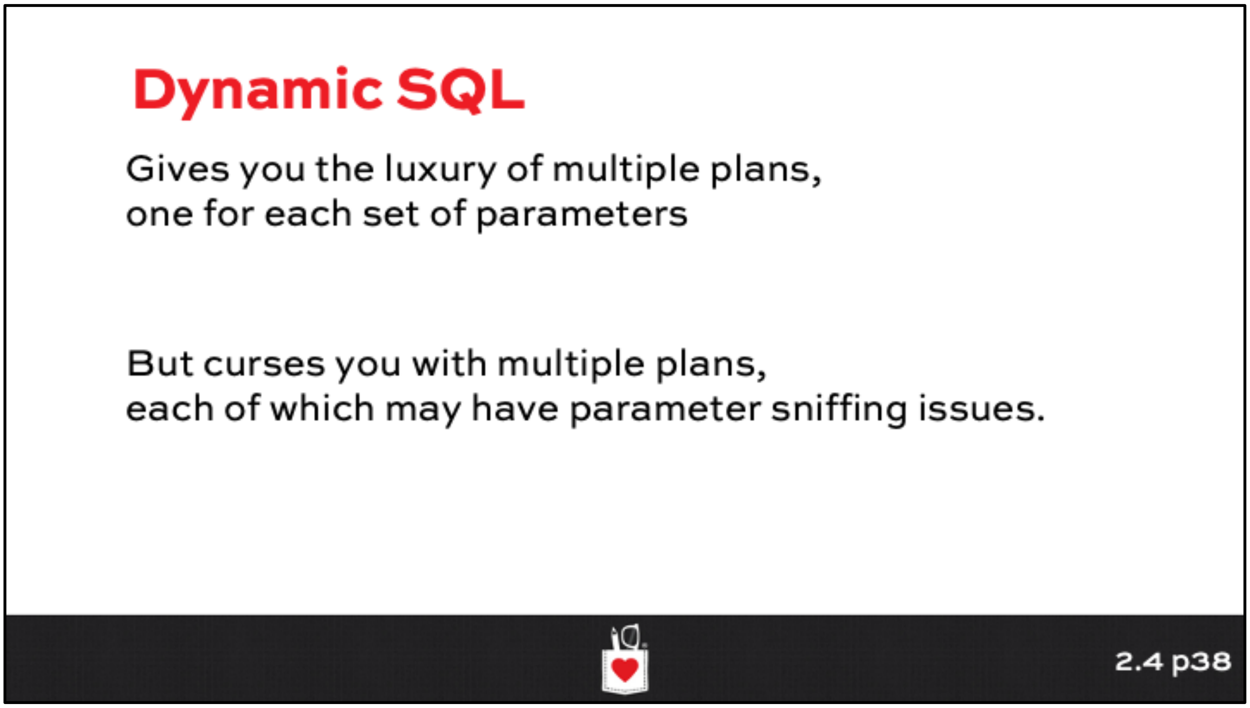
If London runs first…

We cache a plan with a scan.

And reuse it for Chicago

But the memory grant is too big.

2.4 p37

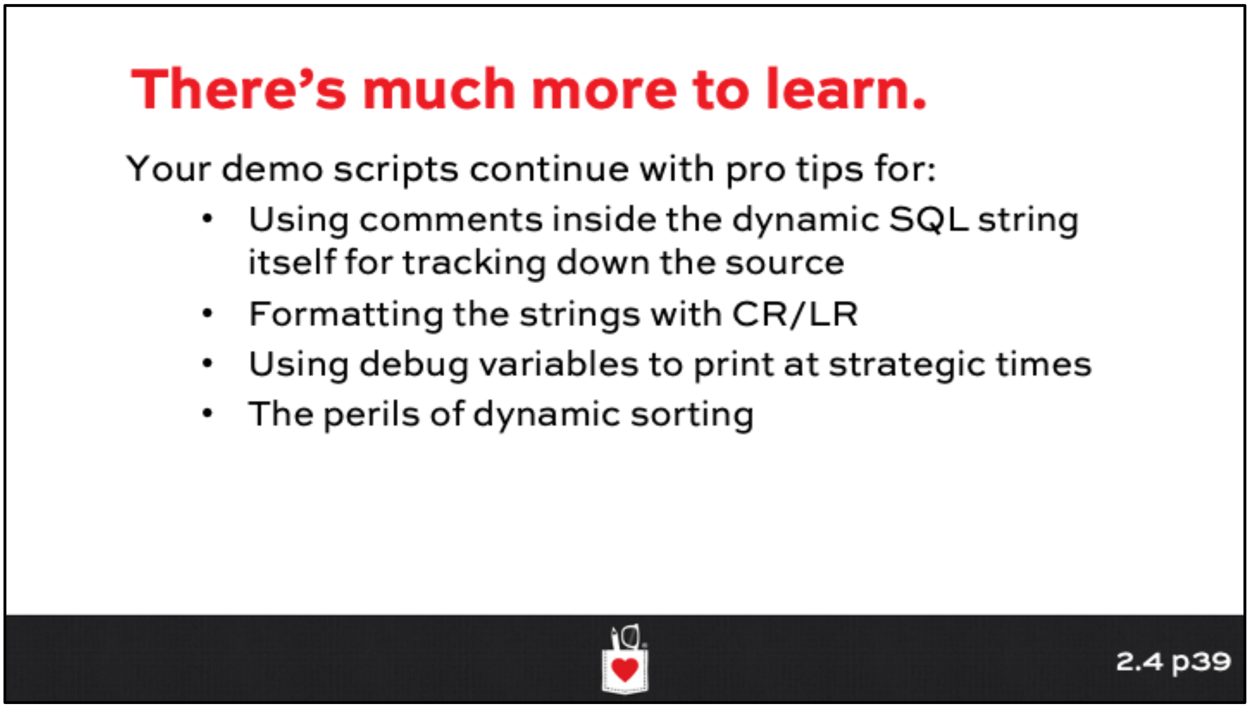


Dynamic SQL

Gives you the luxury of multiple plans,
one for each set of parameters

But curses you with multiple plans,
each of which may have parameter sniffing issues.

2.4 p38



There’s much more to learn.

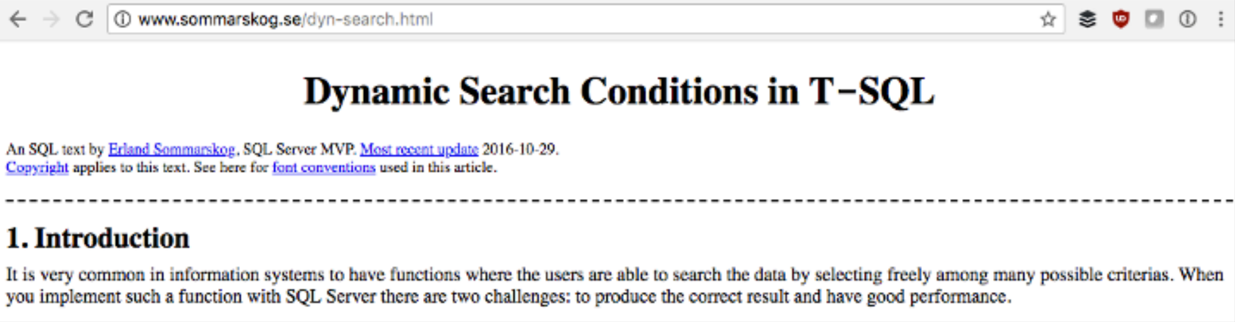
Your demo scripts continue with pro tips for:

- Using comments inside the dynamic SQL string itself for tracking down the source
- Formatting the strings with CR/LR
- Using debug variables to print at strategic times
- The perils of dynamic sorting

2.4 p39

Erland goes even deeper.

Alternate tables, forced recompilation, CLR…



www.sommarskog.se/dyn-search.html

Dynamic Search Conditions in T-SQL

An SQL text by Erland Sommarskog, SQL Server MVP. Most recent update 2016-10-29.
Copyright applies to this text. See here for font conventions used in this article.

1. Introduction

It is very common in information systems to have functions where the users are able to search the data by selecting freely among many possible criterias. When you implement such a function with SQL Server there are two challenges: to produce the correct result and have good performance.

2.4 p40