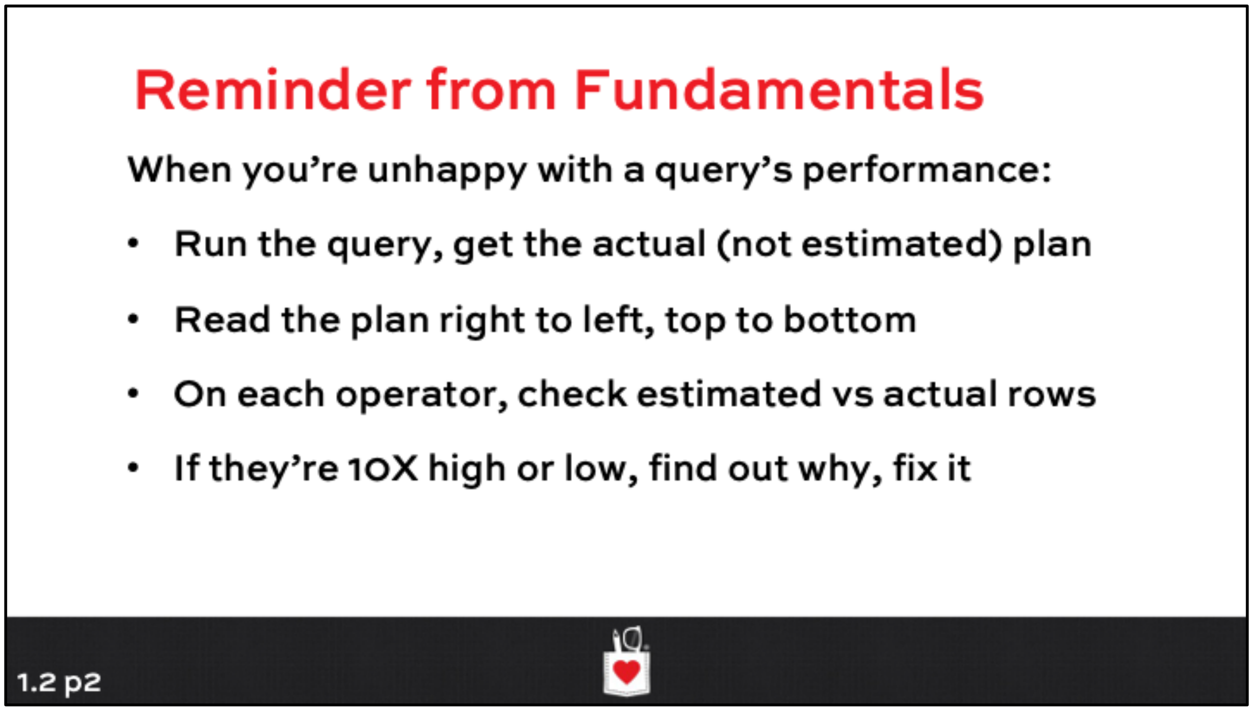
BRENT OZAR
UNLIMITED®

# When the Architect Gets an Estimate Wrong

1.2 p1

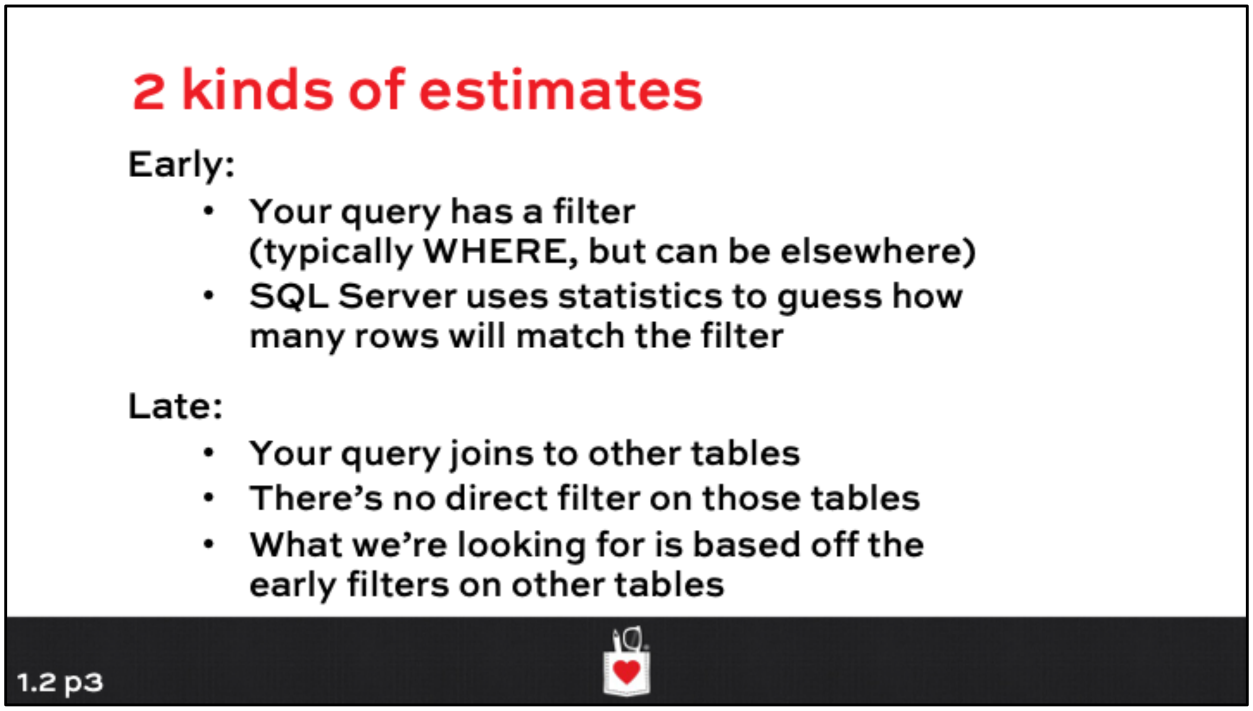


## Reminder from Fundamentals

When you're unhappy with a query's performance:

- Run the query, get the actual (not estimated) plan
- Read the plan right to left, top to bottom
- On each operator, check estimated vs actual rows
- If they're 10X high or low, find out why, fix it

1.2 p2



## 2 kinds of estimates

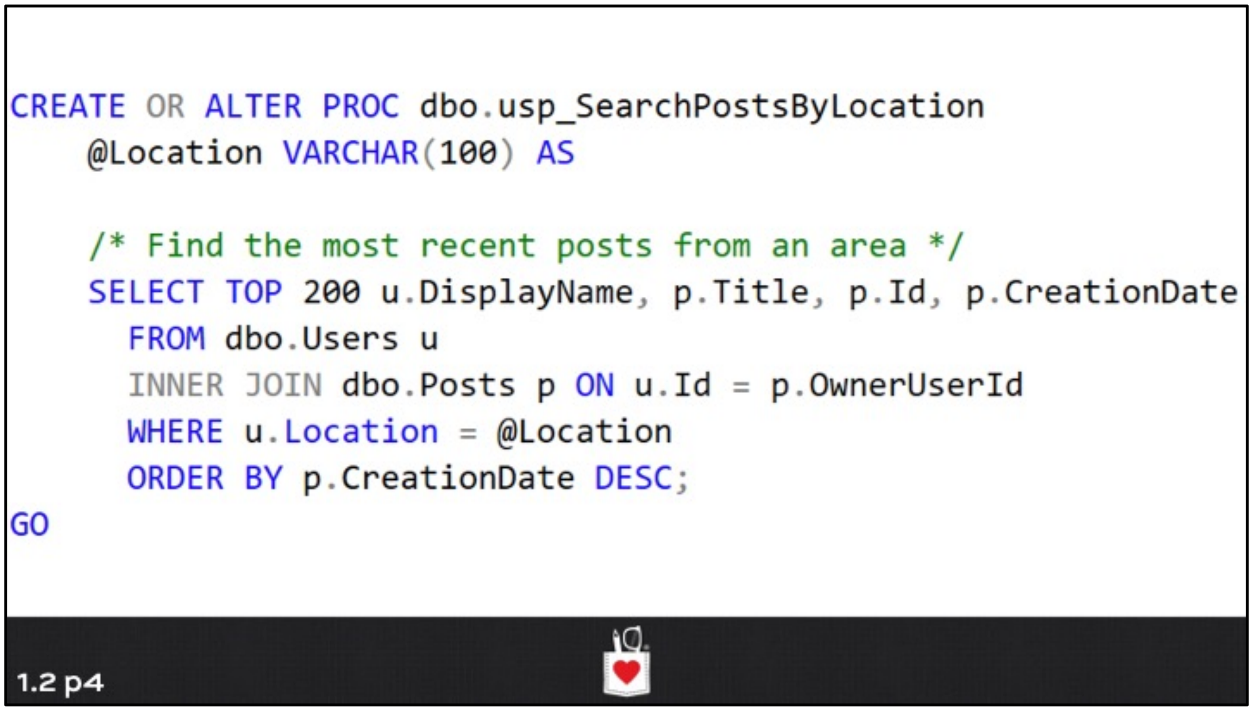
Early:

- Your query has a filter
  (typically WHERE, but can be elsewhere)
- SQL Server uses statistics to guess how many rows will match the filter

Late:

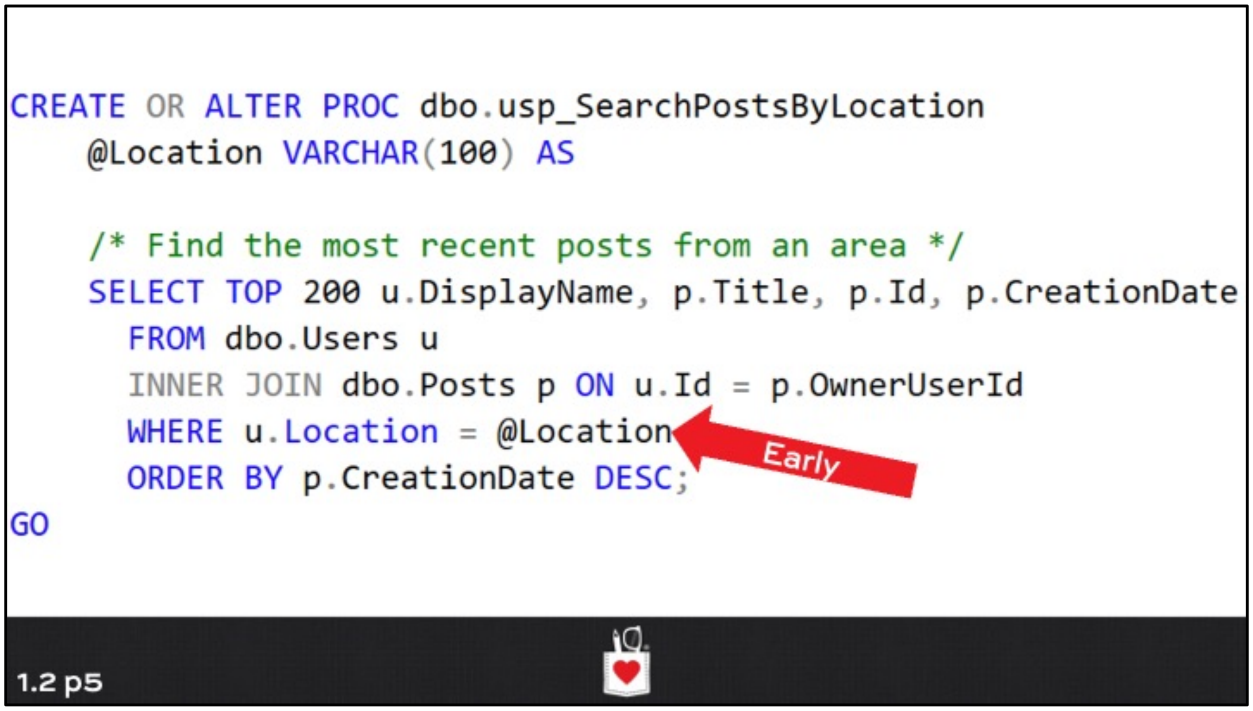
- Your query joins to other tables
- There's no direct filter on those tables
- What we're looking for is based off the early filters on other tables

1.2 p3

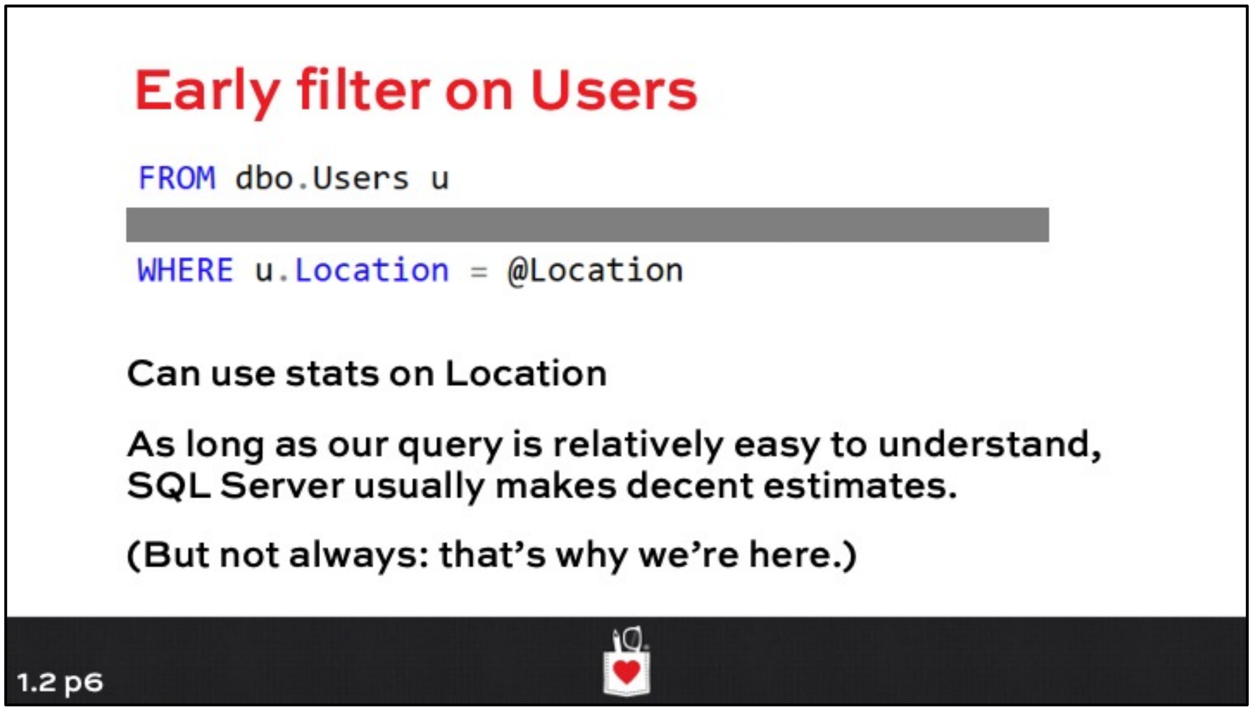


```sql
CREATE OR ALTER PROC dbo.usp_SearchPostsByLocation
    @Location VARCHAR(100) AS

    /* Find the most recent posts from an area */
    SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate
      FROM dbo.Users u
      INNER JOIN dbo.Posts p ON u.Id = p.OwnerUserId
      WHERE u.Location = @Location
      ORDER BY p.CreationDate DESC;
GO
```

1.2 p4



```
CREATE OR ALTER PROC dbo.usp_SearchPostsByLocation
    @Location VARCHAR(100) AS

    /* Find the most recent posts from an area */
    SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate
      FROM dbo.Users u
      INNER JOIN dbo.Posts p ON u.Id = p.OwnerUserId
      WHERE u.Location = @Location
      ORDER BY p.CreationDate DESC;
GO
```



## Early filter on Users

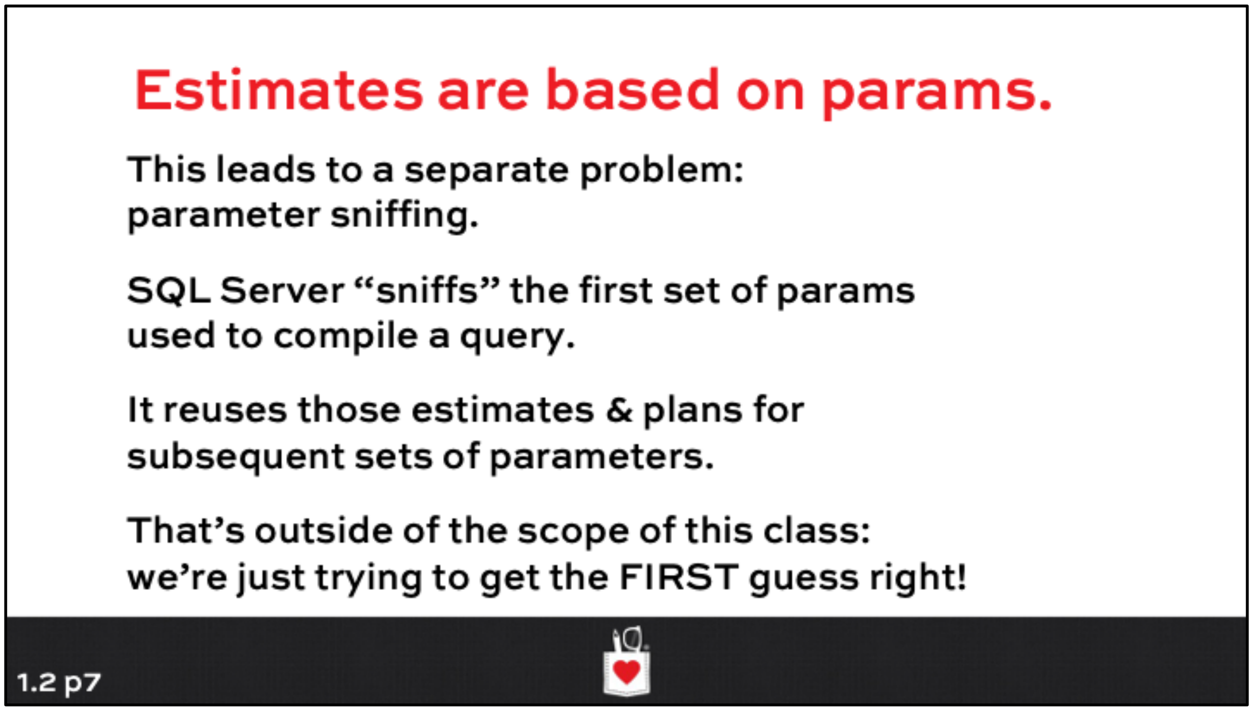
```
FROM dbo.Users u
[redacted]
WHERE u.Location = @Location
```

**Can use stats on Location**

**As long as our query is relatively easy to understand, SQL Server usually makes decent estimates.**

**(But not always: that's why we're here.)**

1.2 p6



## Estimates are based on params.

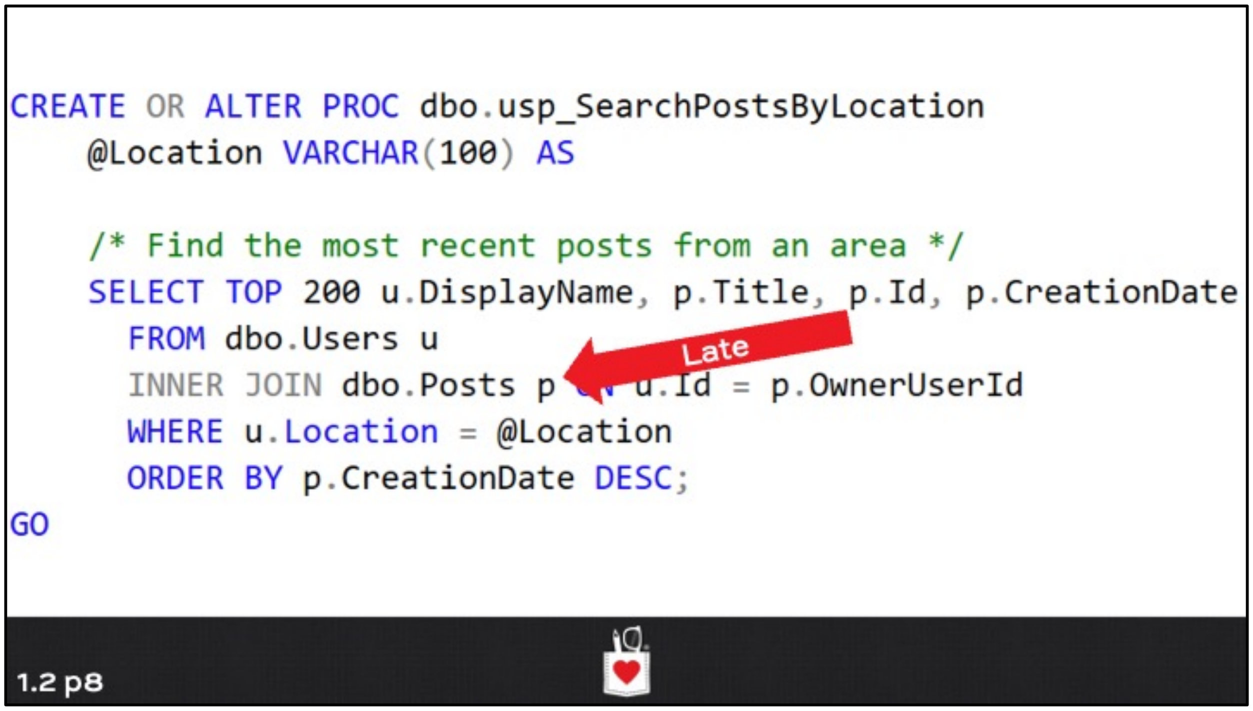
This leads to a separate problem:
parameter sniffing.

SQL Server "sniffs" the first set of params
used to compile a query.

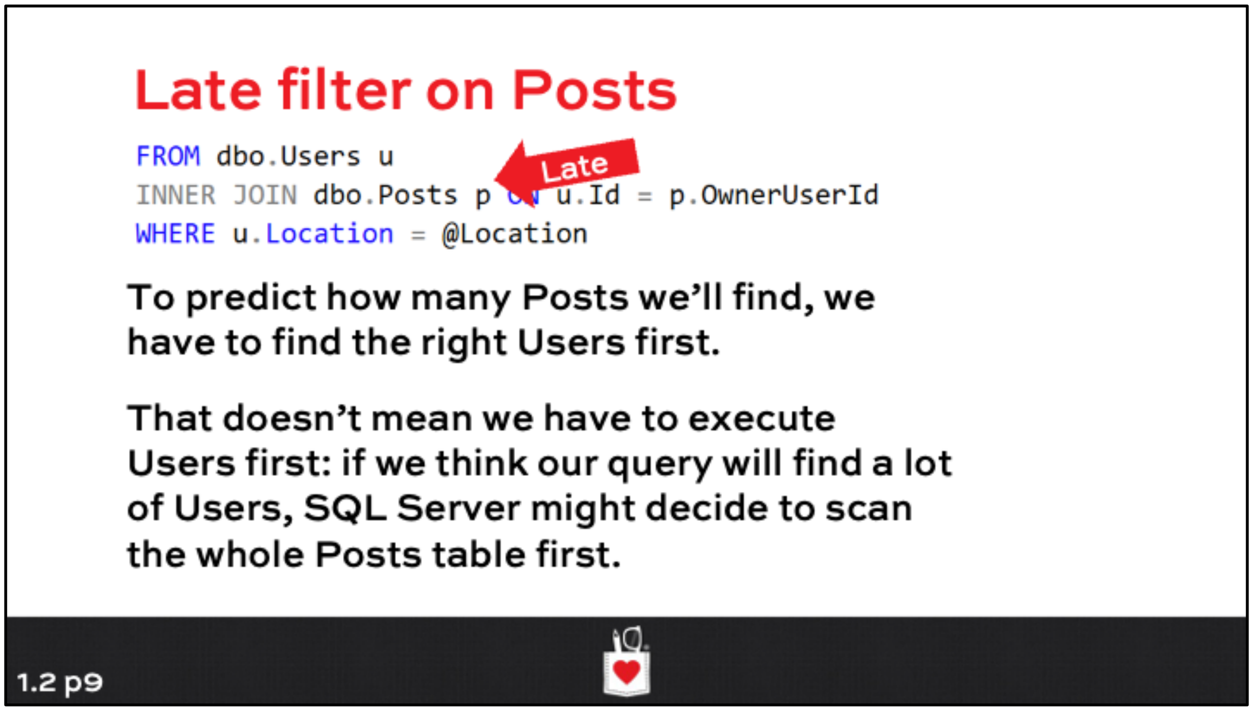
It reuses those estimates & plans for
subsequent sets of parameters.

That's outside of the scope of this class:
we're just trying to get the FIRST guess right!

1.2 p7



```
CREATE OR ALTER PROC dbo.usp_SearchPostsByLocation
    @Location VARCHAR(100) AS

    /* Find the most recent posts from an area */
    SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate
      FROM dbo.Users u
      INNER JOIN dbo.Posts p ON u.Id = p.OwnerUserId
      WHERE u.Location = @Location
      ORDER BY p.CreationDate DESC;
GO
```



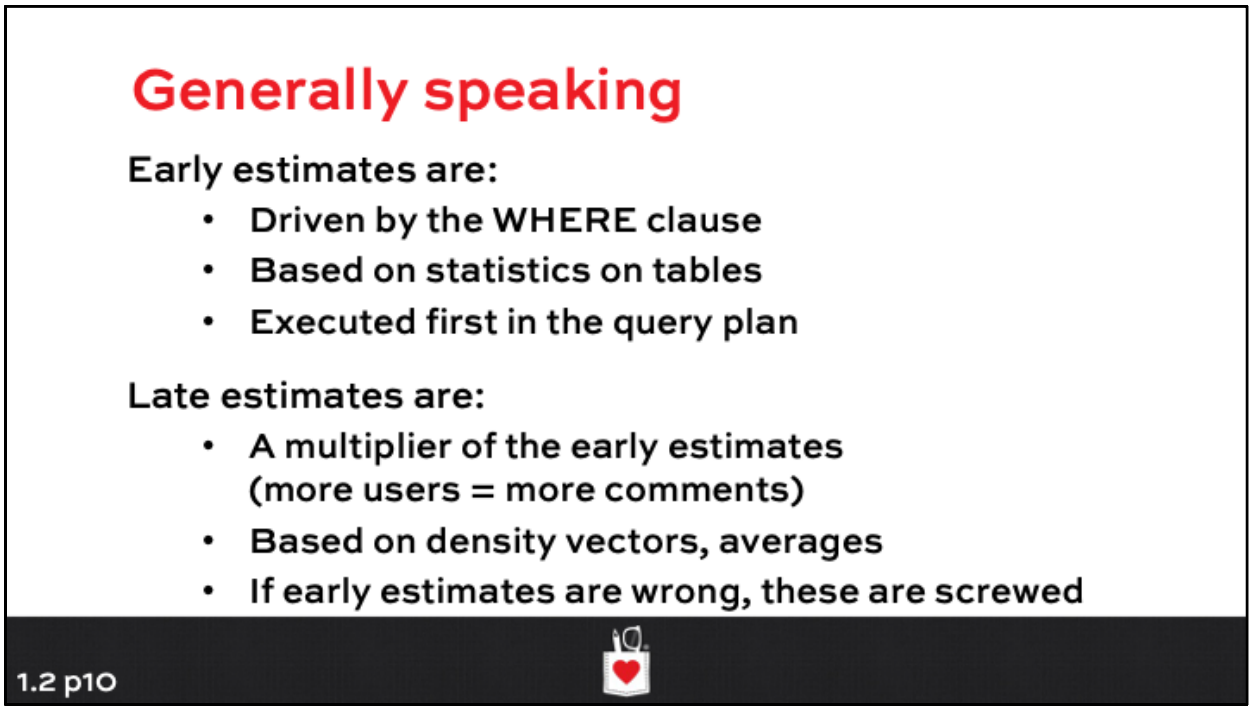
## Late filter on Posts

```
FROM dbo.Users u
INNER JOIN dbo.Posts p ON u.Id = p.OwnerUserId
WHERE u.Location = @Location
```

**To predict how many Posts we'll find, we have to find the right Users first.**

**That doesn't mean we have to execute Users first: if we think our query will find a lot of Users, SQL Server might decide to scan the whole Posts table first.**

1.2 p9



## Generally speaking

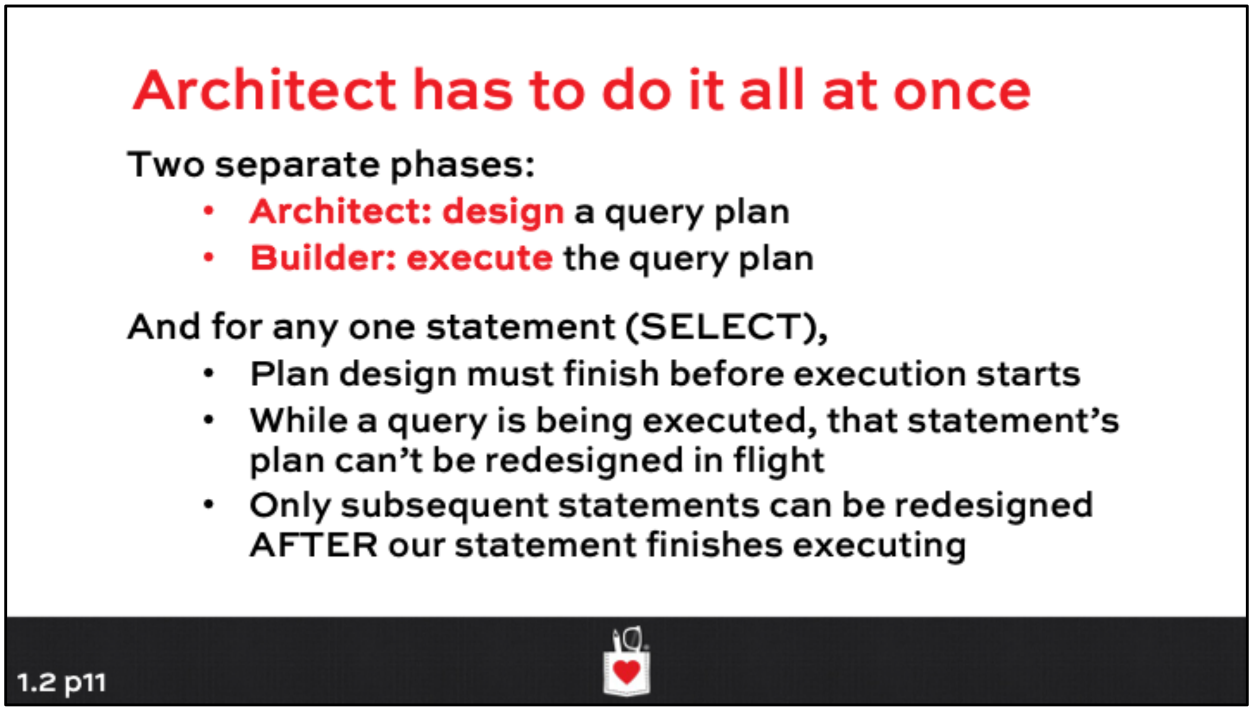
Early estimates are:

- Driven by the WHERE clause
- Based on statistics on tables
- Executed first in the query plan

Late estimates are:

- A multiplier of the early estimates
  (more users = more comments)
- Based on density vectors, averages
- If early estimates are wrong, these are screwed

1.2 p10



## Architect has to do it all at once

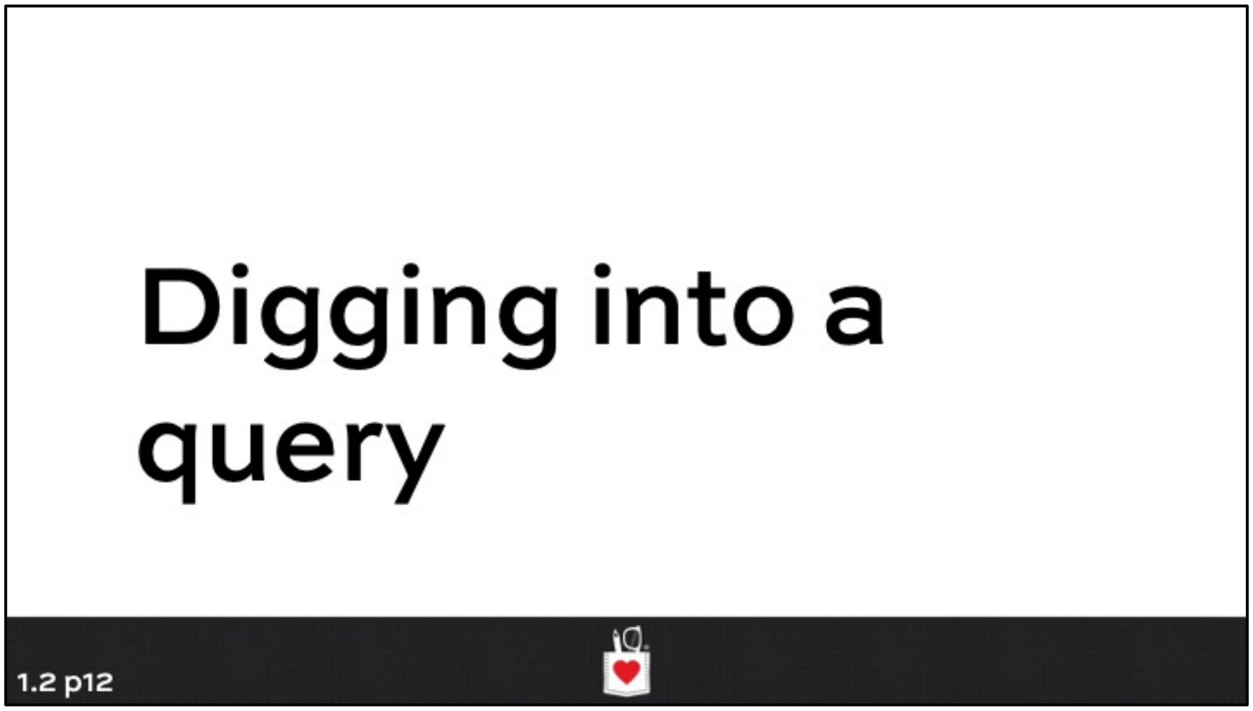
Two separate phases:

- **Architect: design** a query plan
- **Builder: execute** the query plan

And for any one statement (SELECT),

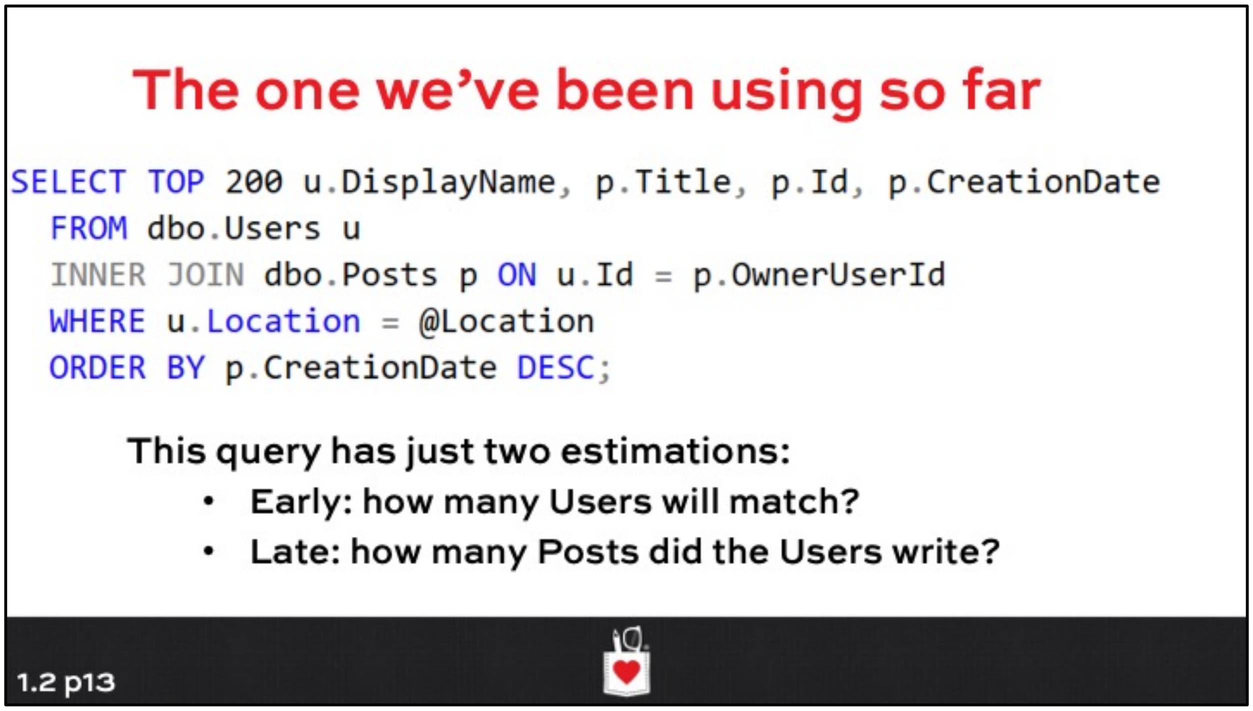
- Plan design must finish before execution starts
- While a query is being executed, that statement's plan can't be redesigned in flight
- Only subsequent statements can be redesigned AFTER our statement finishes executing

1.2 p11



## Digging into a query

1.2 p12



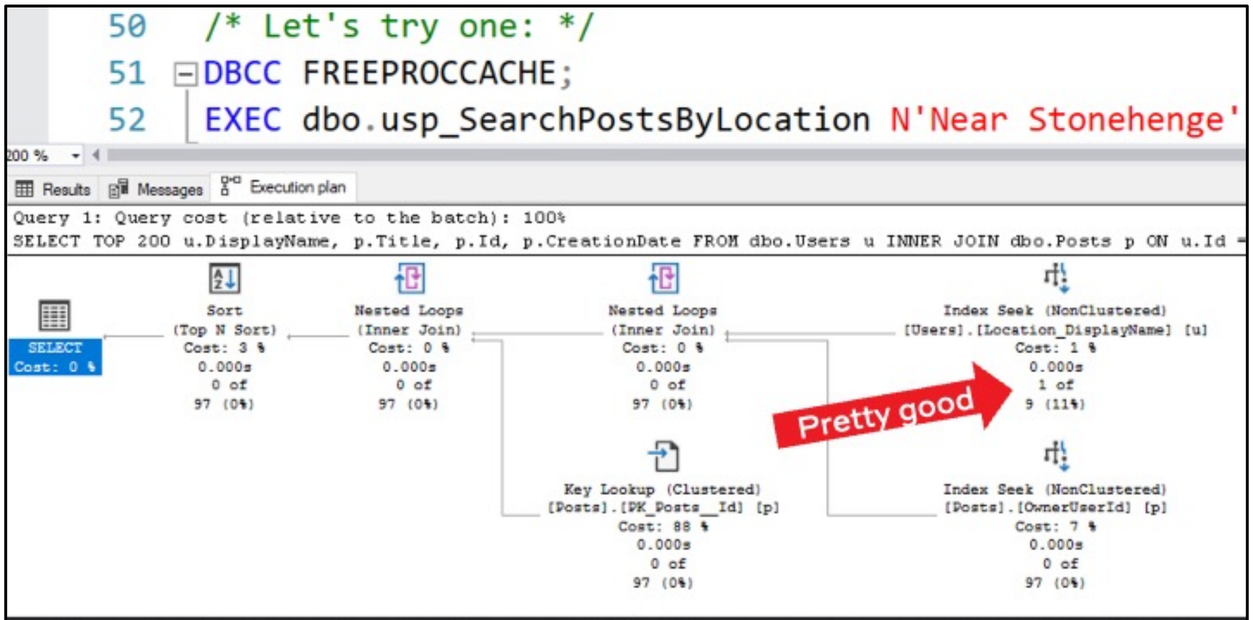
## The one we've been using so far

```
SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate
  FROM dbo.Users u
  INNER JOIN dbo.Posts p ON u.Id = p.OwnerUserId
  WHERE u.Location = @Location
  ORDER BY p.CreationDate DESC;
```

**This query has just two estimations:**

- **Early: how many Users will match?**
- **Late: how many Posts did the Users write?**

1.2 p13

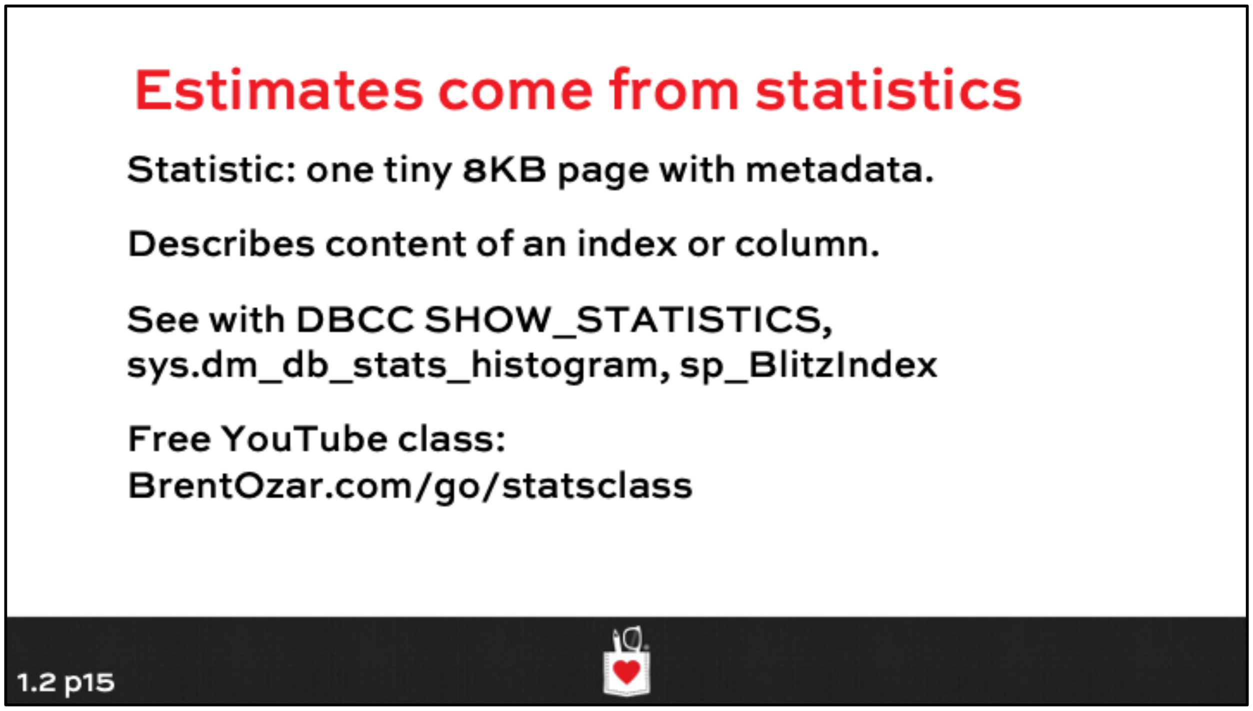


```
/* Let's try one: */
DBCC FREEPROCCACHE;
EXEC dbo.usp_SearchPostsByLocation N'Near Stonehenge'
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate FROM dbo.Users u INNER JOIN dbo.Posts p ON u.Id =
```

```
SELECT        Sort            Nested Loops     Nested Loops     Index Seek (NonClustered)
Cost: 0 %     (Top N Sort)    (Inner Join)     (Inner Join)     [Users].[Location_DisplayName] [u]
              Cost: 3 %       Cost: 0 %        Cost: 0 %        Cost: 1 %
              0.000s          0.000s           0.000s           0.000s
              0 of            0 of             0 of             1 of
              97 (0%)         97 (0%)          97 (0%)          9 (11%)

                                               Key Lookup (Clustered)        Index Seek (NonClustered)
                                               [Posts].[PK_Posts__Id] [p]    [Posts].[OwnerUserId] [p]
                                               Cost: 88 %                    Cost: 7 %
                                               0.000s                        0.000s
                                               0 of                          0 of
                                               97 (0%)                       97 (0%)
```

1.2 p14



## Estimates come from statistics

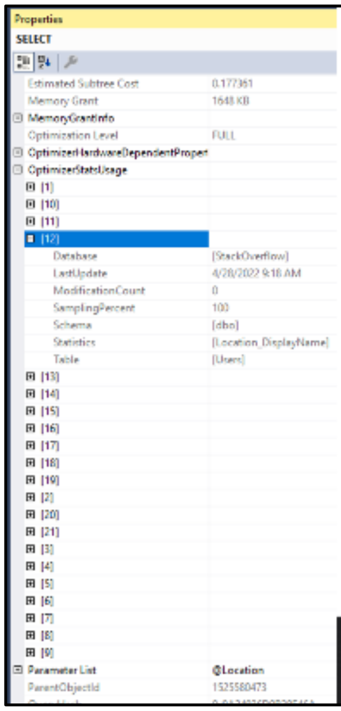
Statistic: one tiny 8KB page with metadata.

Describes content of an index or column.

See with DBCC SHOW_STATISTICS, sys.dm_db_stats_histogram, sp_BlitzIndex

Free YouTube class: BrentOzar.com/go/statsclass

1.2 p15



Properties

SELECT

| | |
|---|---|
| Estimated Subtree Cost | 0.177361 |
| Memory Grant | 1648 KB |
| ⊞ MemoryGrantInfo | |
| Optimization Level | FULL |
| ⊞ OptimizerHardwareDependentPropert | |
| ⊟ OptimizerStatsUsage | |
| ⊞ [1] | |
| ⊞ [10] | |
| ⊞ [11] | |
| ⊟ [12] | |
| Database | [StackOverflow] |
| LastUpdate | 4/28/2022 9:18 AM |
| ModificationCount | 0 |
| SamplingPercent | 100 |
| Schema | [dbo] |
| Statistics | [Location_DisplayName] |
| Table | [Users] |
| ⊞ [13] | |
| ⊞ [14] | |
| ⊞ [15] | |
| ⊞ [16] | |
| ⊞ [17] | |
| ⊞ [18] | |
| ⊞ [19] | |
| ⊞ [2] | |
| ⊞ [20] | |
| ⊞ [21] | |
| ⊞ [3] | |
| ⊞ [4] | |
| ⊞ [5] | |
| ⊞ [6] | |
| ⊞ [7] | |
| ⊞ [8] | |
| ⊞ [9] | |
| ⊟ Parameter List | @Location |
| ParentObjectId | 1525580473 |

## Which stats were used?

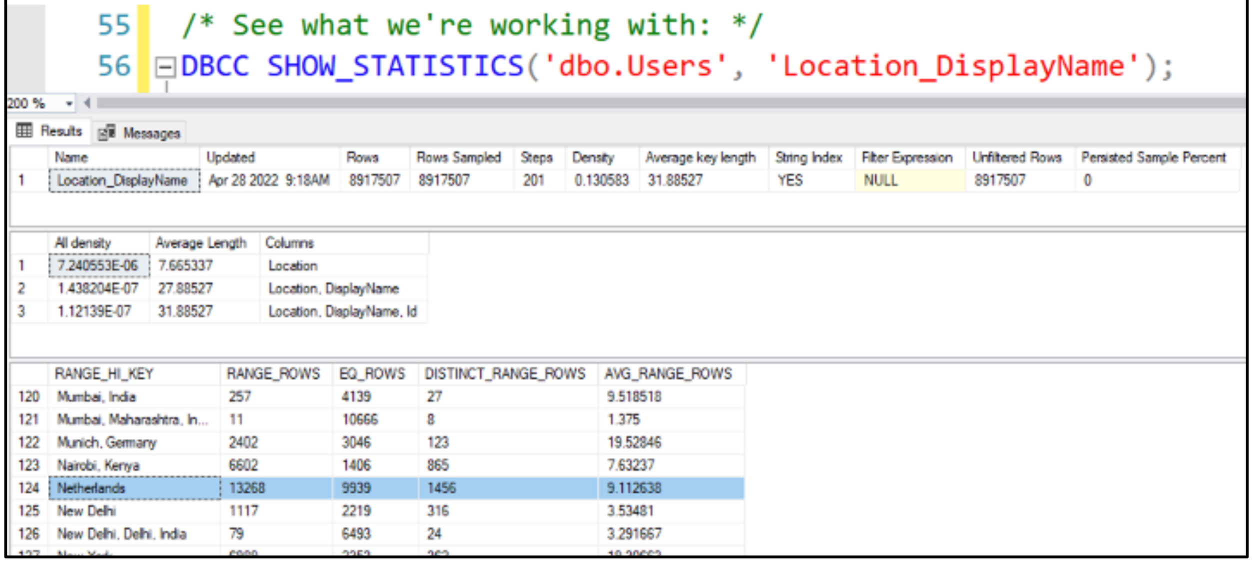
Recent versions of SQL Server show it in the query plan.

Not in any particular order.

Many of the stats may not have helped.

In this case, the big driver is the stat on the Users.Location_DisplayName index.

## That statistic’s contents



```sql
/* See what we're working with: */
DBCC SHOW_STATISTICS('dbo.Users', 'Location_DisplayName');
```

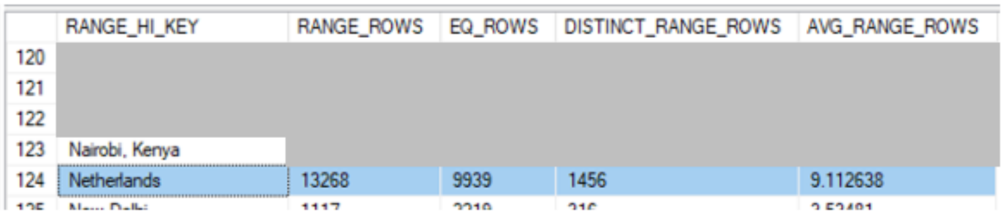
200 %

Results | Messages

|  | Name | Updated | Rows | Rows Sampled | Steps | Density | Average key length | String Index | Filter Expression | Unfiltered Rows | Persisted Sample Percent |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Location_DisplayName | Apr 28 2022 9:18AM | 8917507 | 8917507 | 201 | 0.130583 | 31.88527 | YES | NULL | 8917507 | 0 |

|  | All density | Average Length | Columns |
|---|---|---|---|
| 1 | 7.240553E-06 | 7.665337 | Location |
| 2 | 1.438204E-07 | 27.88527 | Location, DisplayName |
| 3 | 1.12139E-07 | 31.88527 | Location, DisplayName, Id |

|  | RANGE_HI_KEY | RANGE_ROWS | EQ_ROWS | DISTINCT_RANGE_ROWS | AVG_RANGE_ROWS |
|---|---|---|---|---|---|
| 120 | Mumbai, India | 257 | 4139 | 27 | 9.518518 |
| 121 | Mumbai, Maharashtra, In... | 11 | 10666 | 8 | 1.375 |
| 122 | Munich, Germany | 2402 | 3046 | 123 | 19.52846 |
| 123 | Nairobi, Kenya | 6602 | 1406 | 865 | 7.63237 |
| 124 | Netherlands | 13268 | 9939 | 1456 | 9.112638 |
| 125 | New Delhi | 1117 | 2219 | 316 | 3.53481 |
| 126 | New Delhi, Delhi, India | 79 | 6493 | 24 | 3.291667 |



| | RANGE_HI_KEY | RANGE_ROWS | EQ_ROWS | DISTINCT_RANGE_ROWS | AVG_RANGE_ROWS |
|---|---|---|---|---|---|
| 120 | | | | | |
| 121 | | | | | |
| 122 | | | | | |
| 123 | Nairobi, Kenya | | | | |
| 124 | Netherlands | 13268 | 9939 | 1456 | 9.112638 |

Bucket #124: Locations > 'Nairobi' and <= 'Netherlands'
(Near Stonehenge is in this bucket)

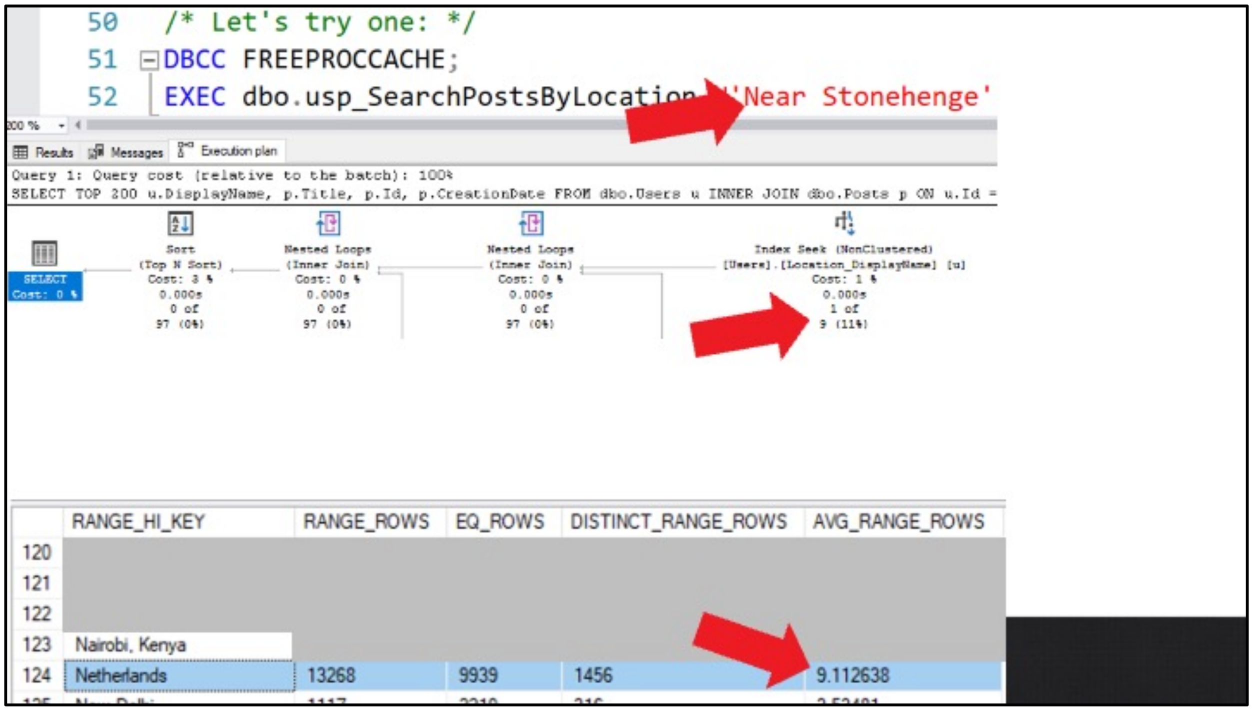
RANGE_ROWS: there are 13,268 rows in this bucket

EQ_ROWS: there are 9,939 with exactly 'Netherlands'

DISTINCT_RANGE_ROWS: there are 1,456 unique Locations in this bucket

AVG_RANGE_ROWS: for any given Location in this range, there are 9.112638 users with that location

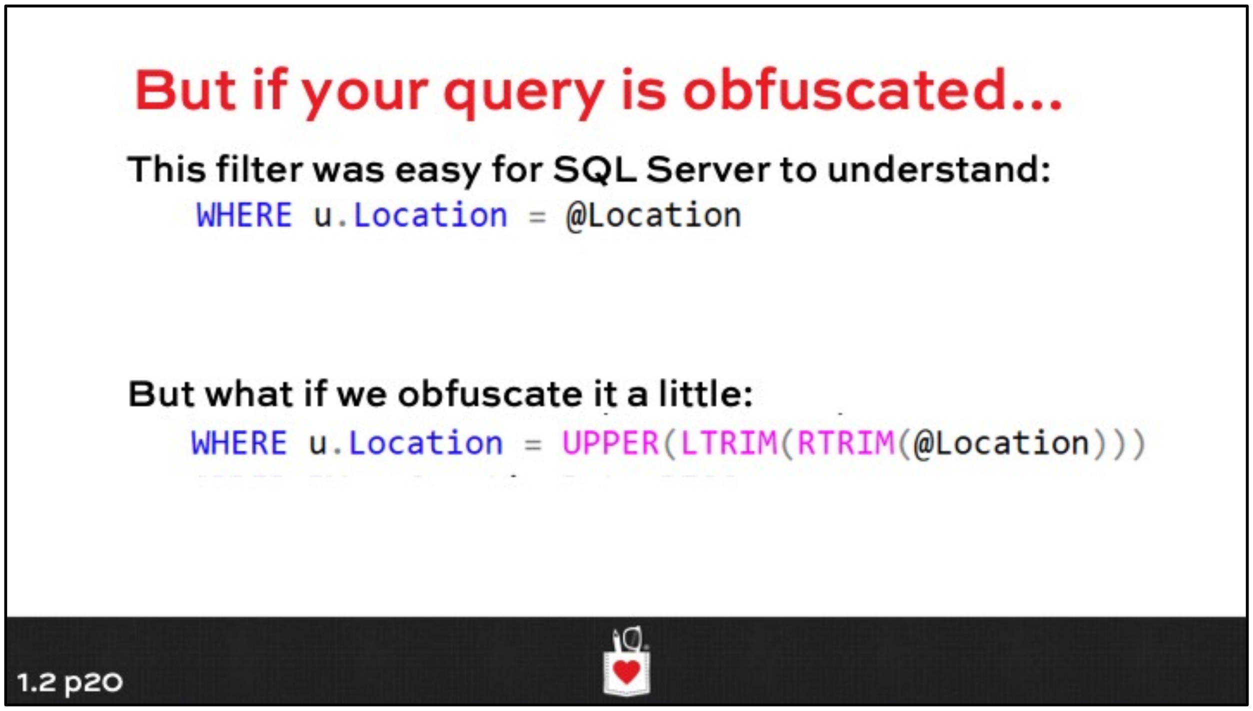
1.2 p18



```
/* Let's try one: */
DBCC FREEPROCCACHE;
EXEC dbo.usp_SearchPostsByLocation 'Near Stonehenge'
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate FROM dbo.Users u INNER JOIN dbo.Posts p ON u.Id =
```

| | RANGE_HI_KEY | RANGE_ROWS | EQ_ROWS | DISTINCT_RANGE_ROWS | AVG_RANGE_ROWS |
|---|---|---|---|---|---|
| 120 | | | | | |
| 121 | | | | | |
| 122 | | | | | |
| 123 | Nairobi, Kenya | | | | |
| 124 | Netherlands | 13268 | 9939 | 1456 | 9.112638 |



## But if your query is obfuscated...

This filter was easy for SQL Server to understand:

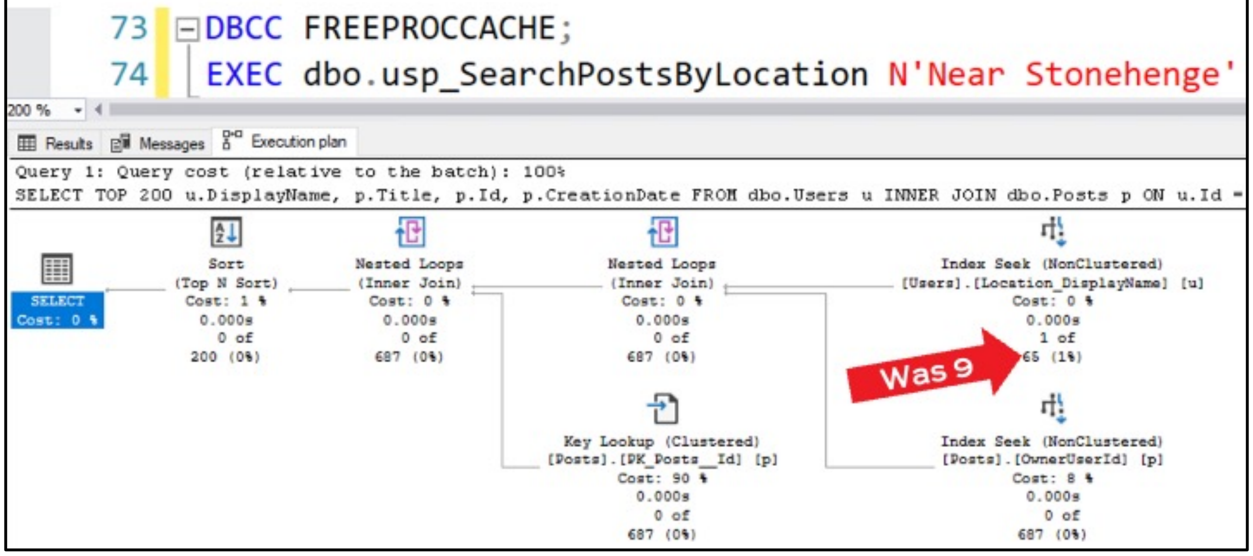
```
WHERE u.Location = @Location
```

But what if we obfuscate it a little:

```
WHERE u.Location = UPPER(LTRIM(RTRIM(@Location)))
```

1.2 p20

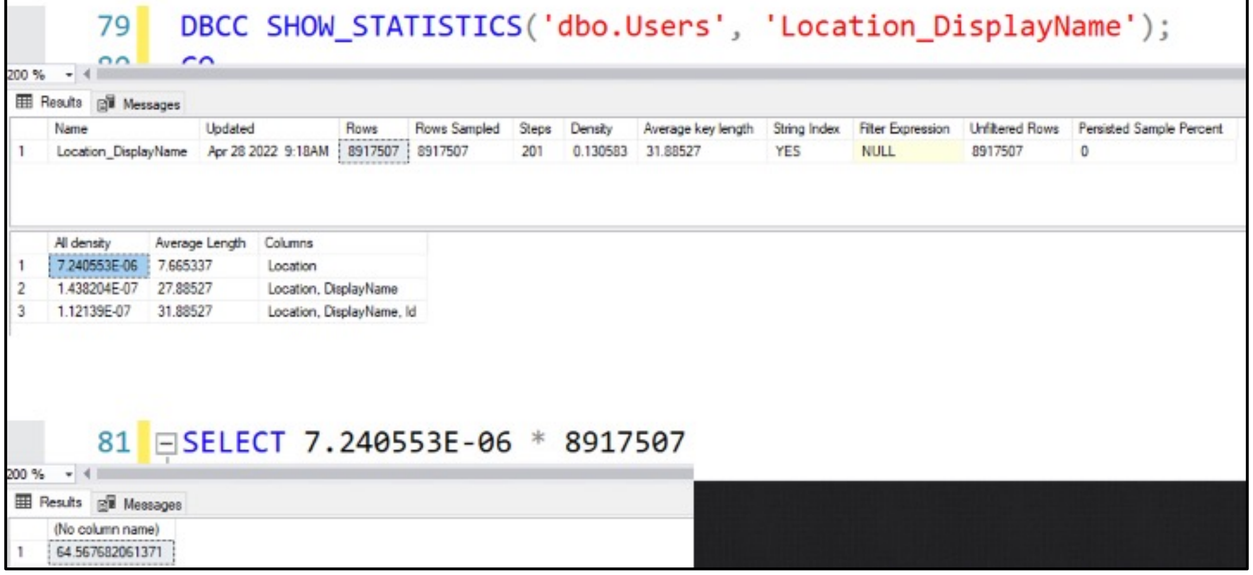
## Then we get different estimates.



```
DBCC FREEPROCCACHE;
EXEC dbo.usp_SearchPostsByLocation N'Near Stonehenge'
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate FROM dbo.Users u INNER JOIN dbo.Posts p ON u.Id =
```

## Density vector: the "average" Location



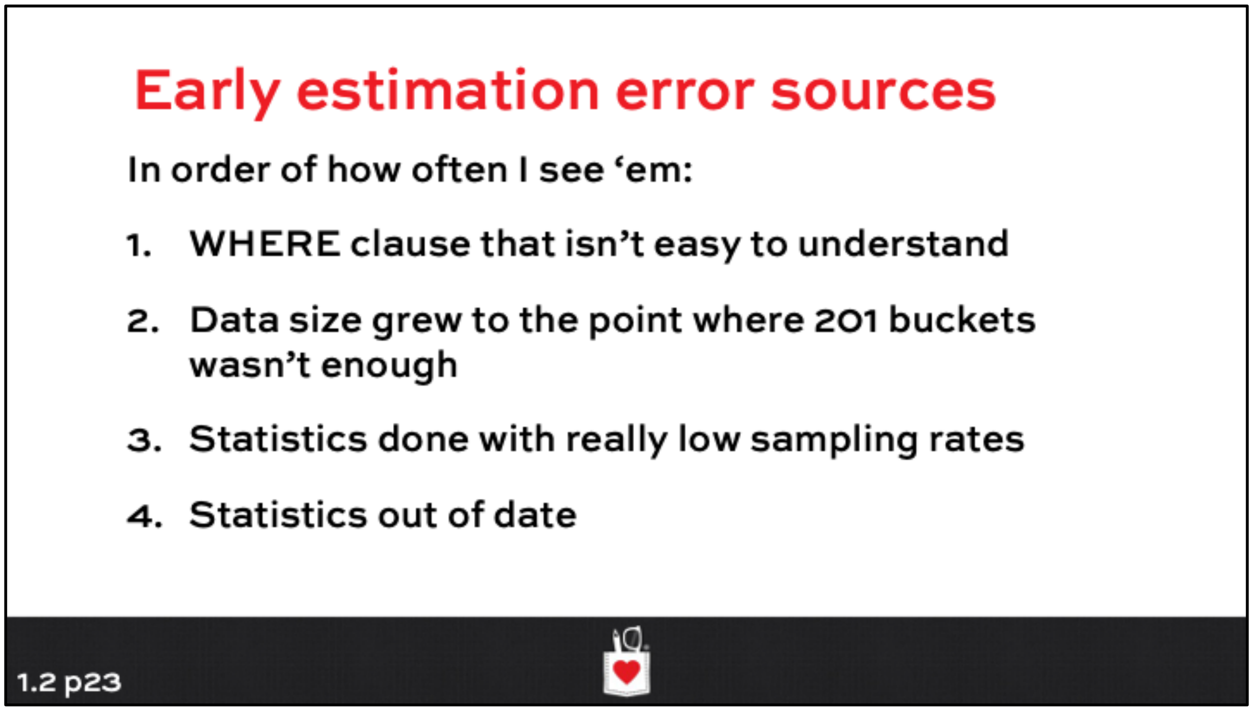
```sql
DBCC SHOW_STATISTICS('dbo.Users', 'Location_DisplayName');
GO
```

| | Name | Updated | Rows | Rows Sampled | Steps | Density | Average key length | String Index | Filter Expression | Unfiltered Rows | Persisted Sample Percent |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Location_DisplayName | Apr 28 2022 9:18AM | 8917507 | 8917507 | 201 | 0.130583 | 31.88527 | YES | NULL | 8917507 | 0 |

| | All density | Average Length | Columns |
|---|---|---|---|
| 1 | 7.240553E-06 | 7.665337 | Location |
| 2 | 1.438204E-07 | 27.88527 | Location, DisplayName |
| 3 | 1.12139E-07 | 31.88527 | Location, DisplayName, Id |

```sql
SELECT 7.240553E-06 * 8917507
```

| | (No column name) |
|---|---|
| 1 | 64.567682061371 |

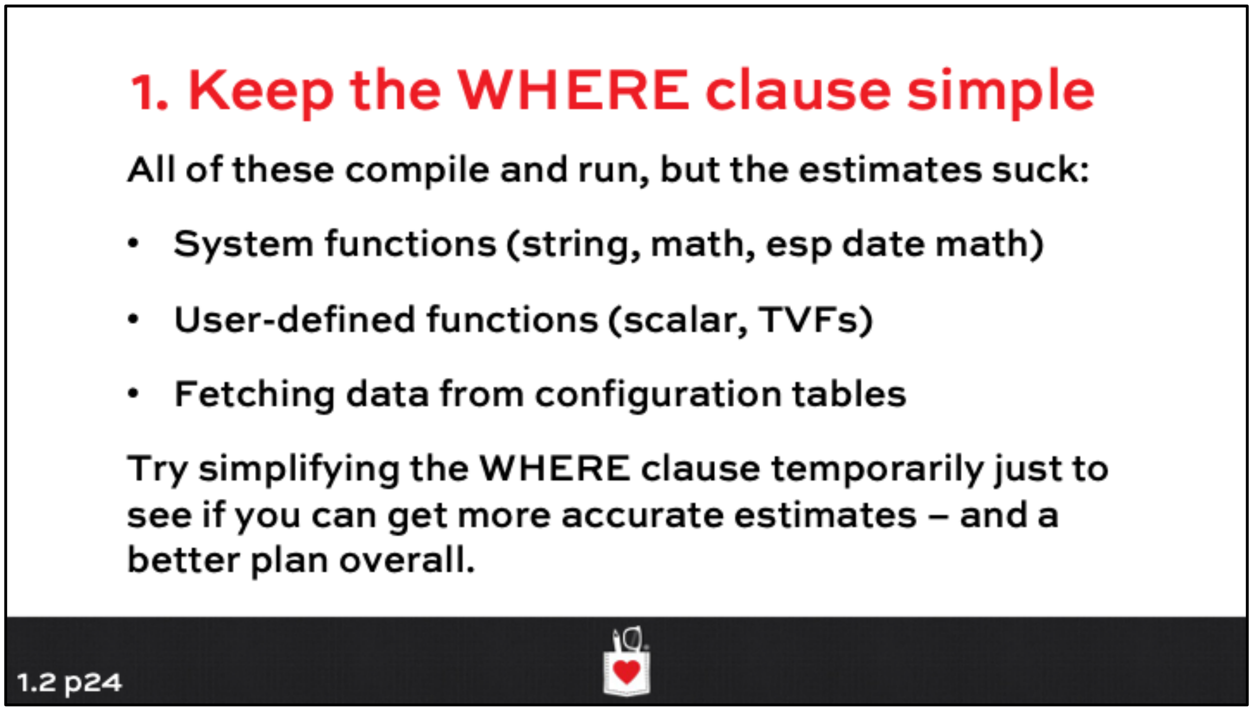


## Early estimation error sources

In order of how often I see 'em:

1. WHERE clause that isn't easy to understand
2. Data size grew to the point where 201 buckets wasn't enough
3. Statistics done with really low sampling rates
4. Statistics out of date

1.2 p23



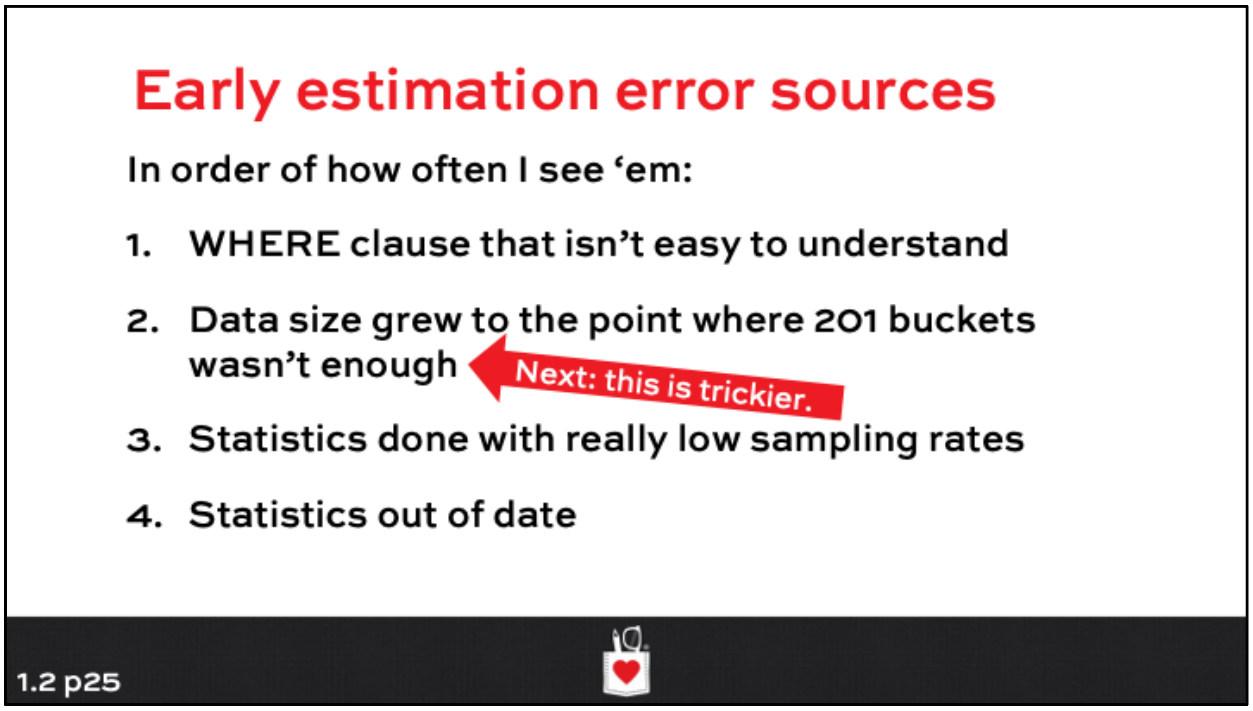
## 1. Keep the WHERE clause simple

All of these compile and run, but the estimates suck:

- System functions (string, math, esp date math)
- User-defined functions (scalar, TVFs)
- Fetching data from configuration tables

Try simplifying the WHERE clause temporarily just to see if you can get more accurate estimates – and a better plan overall.

1.2 p24



## Early estimation error sources

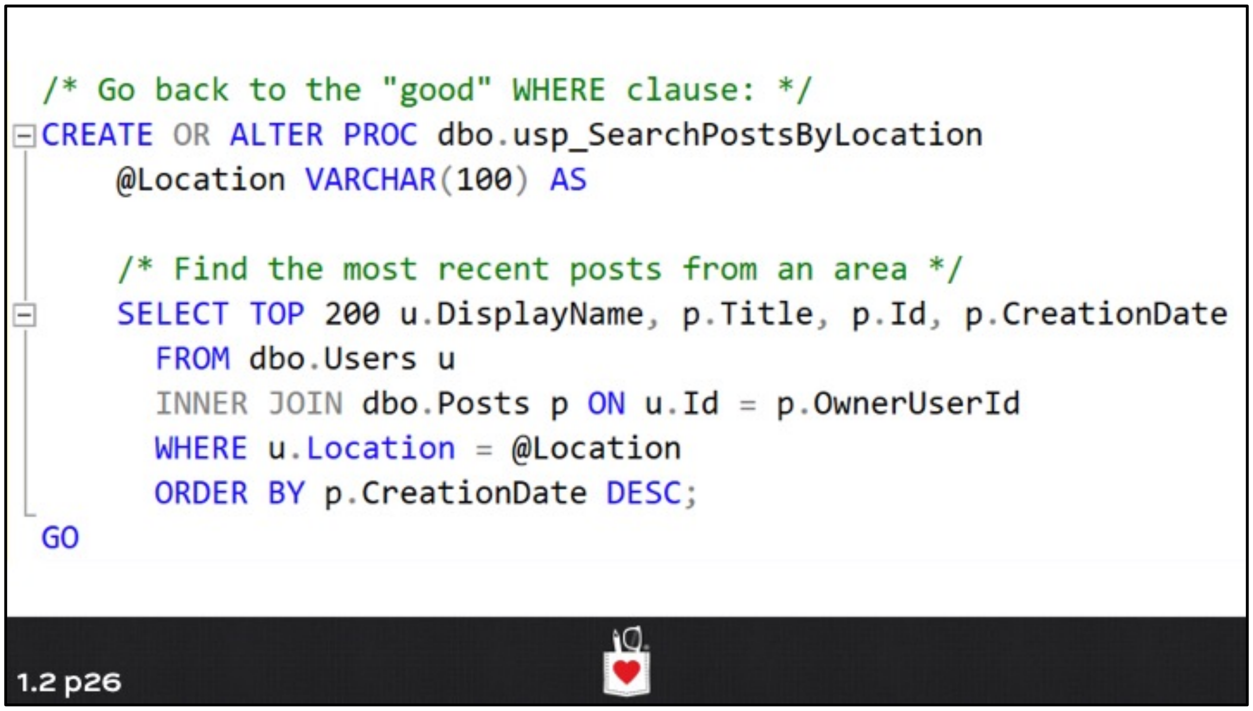
In order of how often I see 'em:

1. WHERE clause that isn't easy to understand
2. Data size grew to the point where 201 buckets wasn't enough

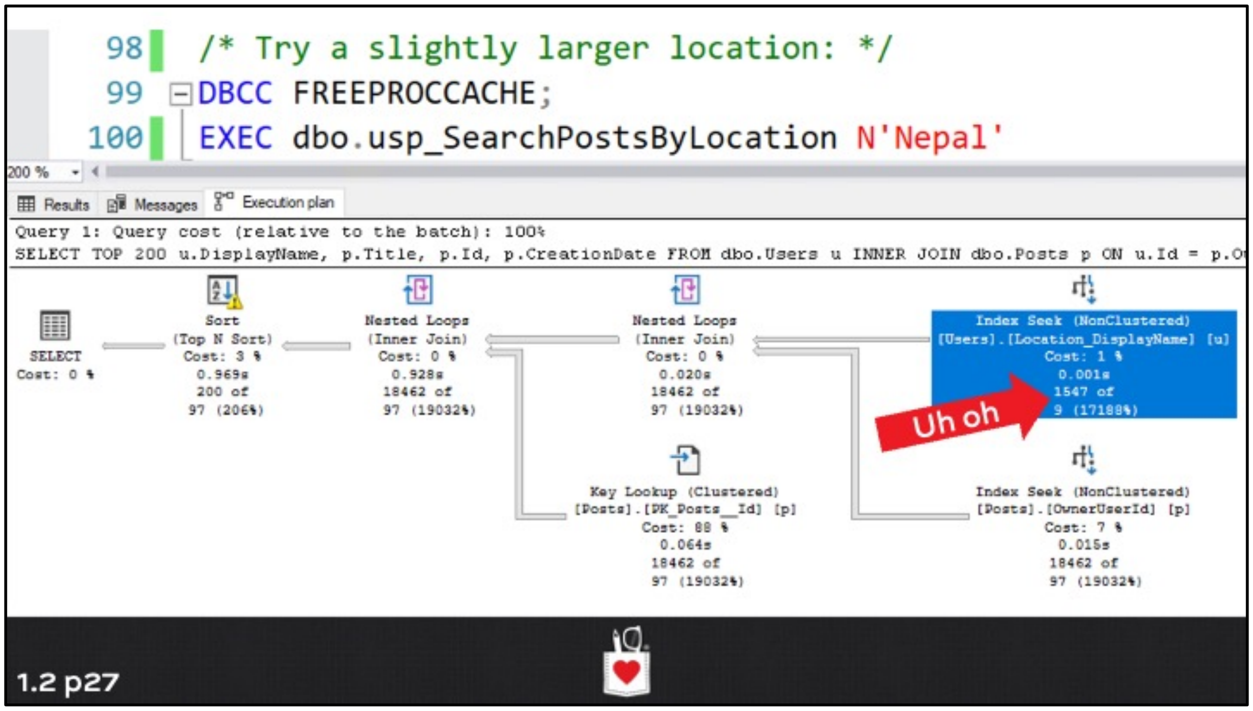
Next: this is trickier.

3. Statistics done with really low sampling rates
4. Statistics out of date

1.2 p25



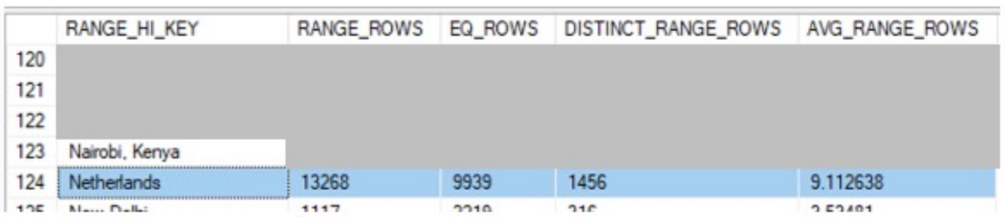
```sql
/* Go back to the "good" WHERE clause: */
CREATE OR ALTER PROC dbo.usp_SearchPostsByLocation
    @Location VARCHAR(100) AS

    /* Find the most recent posts from an area */
    SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate
      FROM dbo.Users u
      INNER JOIN dbo.Posts p ON u.Id = p.OwnerUserId
      WHERE u.Location = @Location
      ORDER BY p.CreationDate DESC;
GO
```



```
/* Try a slightly larger location: */
DBCC FREEPROCCACHE;
EXEC dbo.usp_SearchPostsByLocation N'Nepal'
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate FROM dbo.Users u INNER JOIN dbo.Posts p ON u.Id = p.O
```

```
SELECT        Sort            Nested Loops     Nested Loops     Index Seek (NonClustered)
Cost: 0 %     (Top N Sort)    (Inner Join)     (Inner Join)     [Users].[Location_DisplayName] [u]
              Cost: 3 %       Cost: 0 %        Cost: 0 %        Cost: 1 %
              0.969s          0.928s           0.020s           0.001s
              200 of          18462 of         18462 of         1547 of
              97 (206%)       97 (19032%)      97 (19032%)      9 (17188%)

                                               Key Lookup (Clustered)               Index Seek (NonClustered)
                                               [Posts].[PK_Posts__Id] [p]           [Posts].[OwnerUserId] [p]
                                               Cost: 88 %                           Cost: 7 %
                                               0.064s                               0.015s
                                               18462 of                             18462 of
                                               97 (19032%)                          97 (19032%)
```



| | RANGE_HI_KEY | RANGE_ROWS | EQ_ROWS | DISTINCT_RANGE_ROWS | AVG_RANGE_ROWS |
|---|---|---|---|---|---|
| 120 | | | | | |
| 121 | | | | | |
| 122 | | | | | |
| 123 | Nairobi, Kenya | | | | |
| 124 | Netherlands | 13268 | 9939 | 1456 | 9.112638 |

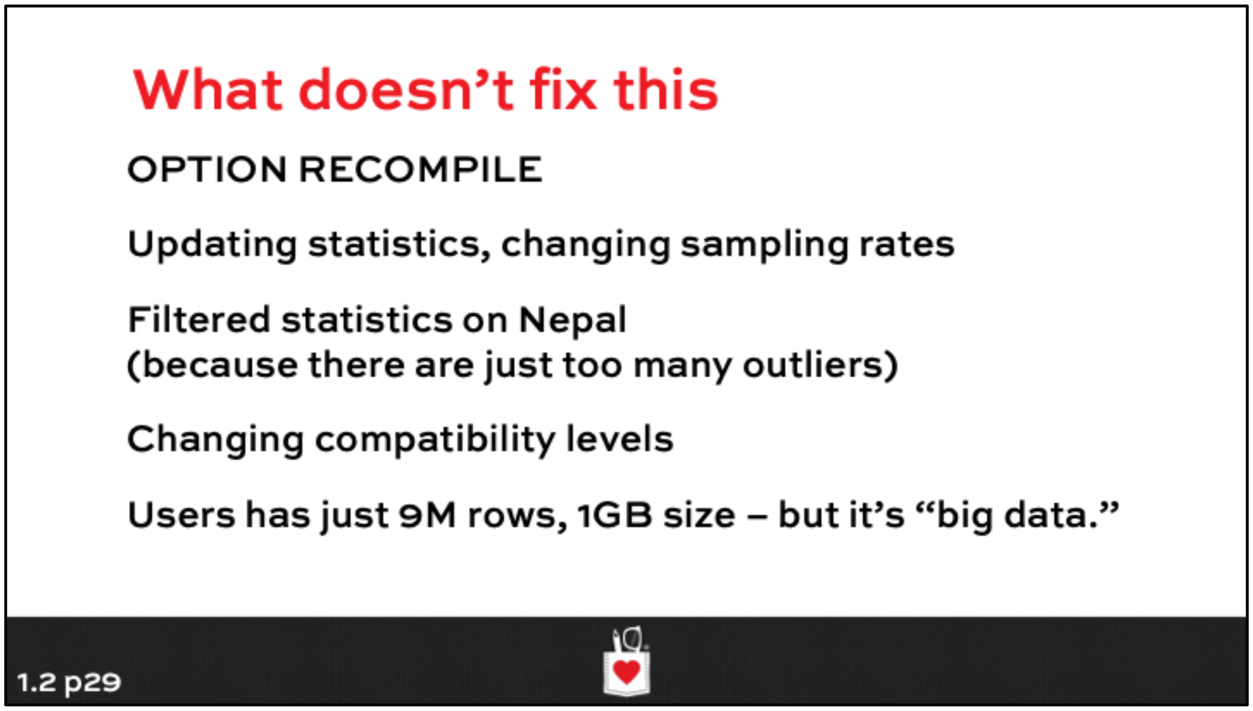
**Nepal is > Nairobi, and < Netherlands.**

**Nepal is in the same bucket as Near Stonehenge.**

**Nepal isn't big enough to get its own bucket.**

**It gets the 9.112638 AVG_RANGE_ROW estimate.**

1.2 p28



## What doesn't fix this

OPTION RECOMPILE

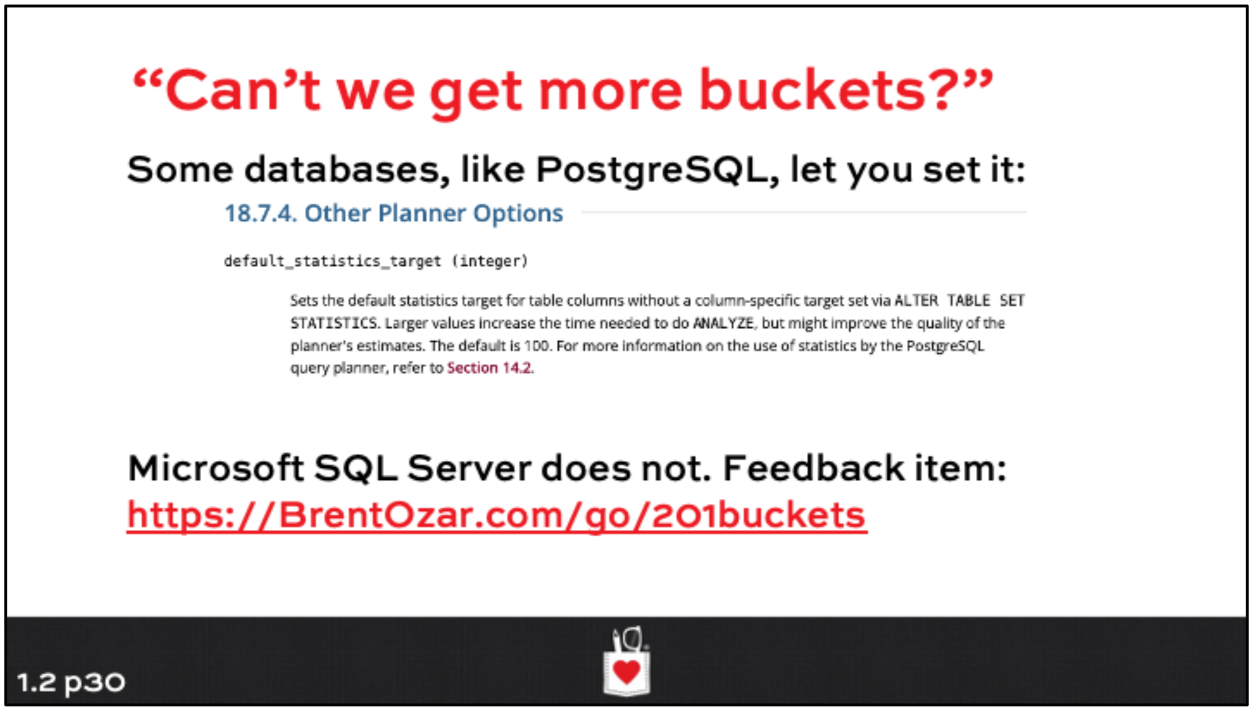
Updating statistics, changing sampling rates

Filtered statistics on Nepal
(because there are just too many outliers)

Changing compatibility levels

Users has just 9M rows, 1GB size – but it's "big data."

1.2 p29



## "Can't we get more buckets?"

**Some databases, like PostgreSQL, let you set it:**

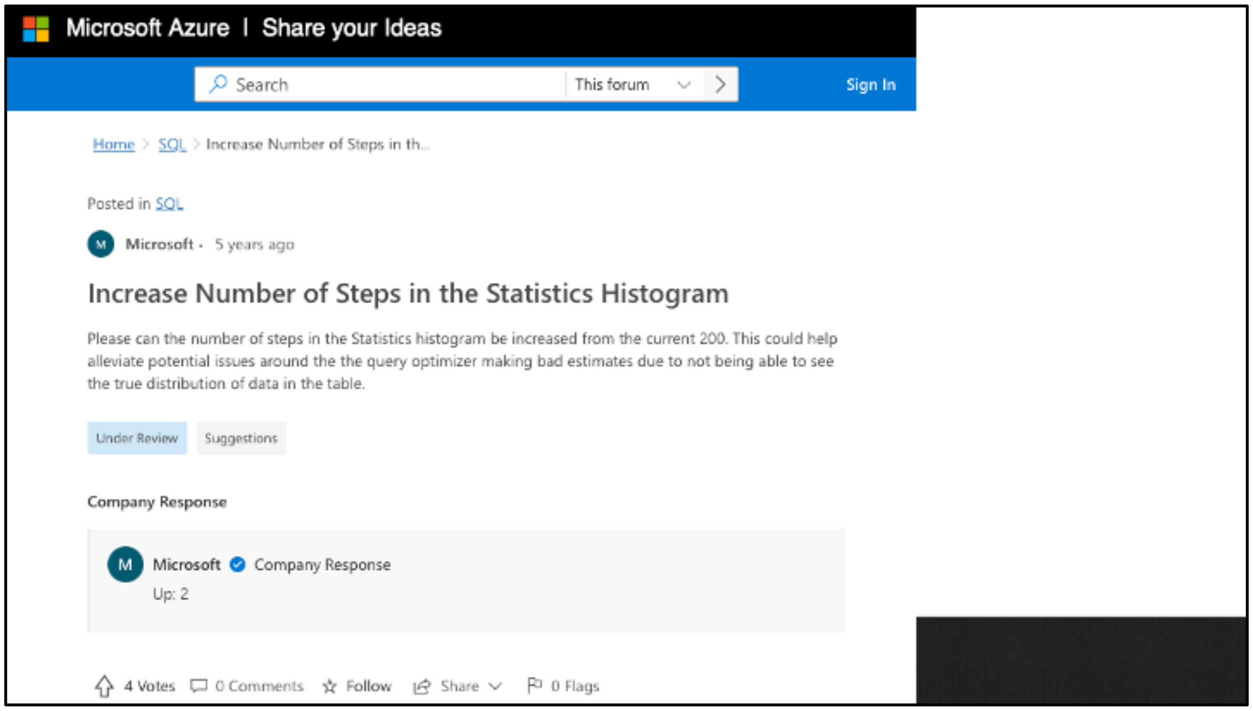
18.7.4. Other Planner Options

`default_statistics_target (integer)`

Sets the default statistics target for table columns without a column-specific target set via `ALTER TABLE SET STATISTICS`. Larger values increase the time needed to do `ANALYZE`, but might improve the quality of the planner's estimates. The default is 100. For more information on the use of statistics by the PostgreSQL query planner, refer to Section 14.2.

**Microsoft SQL Server does not. Feedback item:**
https://BrentOzar.com/go/201buckets

1.2 p30


Microsoft Azure | Share your Ideas

Search | This forum | Sign In

Home > SQL > Increase Number of Steps in th...

Posted in SQL

Microsoft · 5 years ago

Increase Number of Steps in the Statistics Histogram

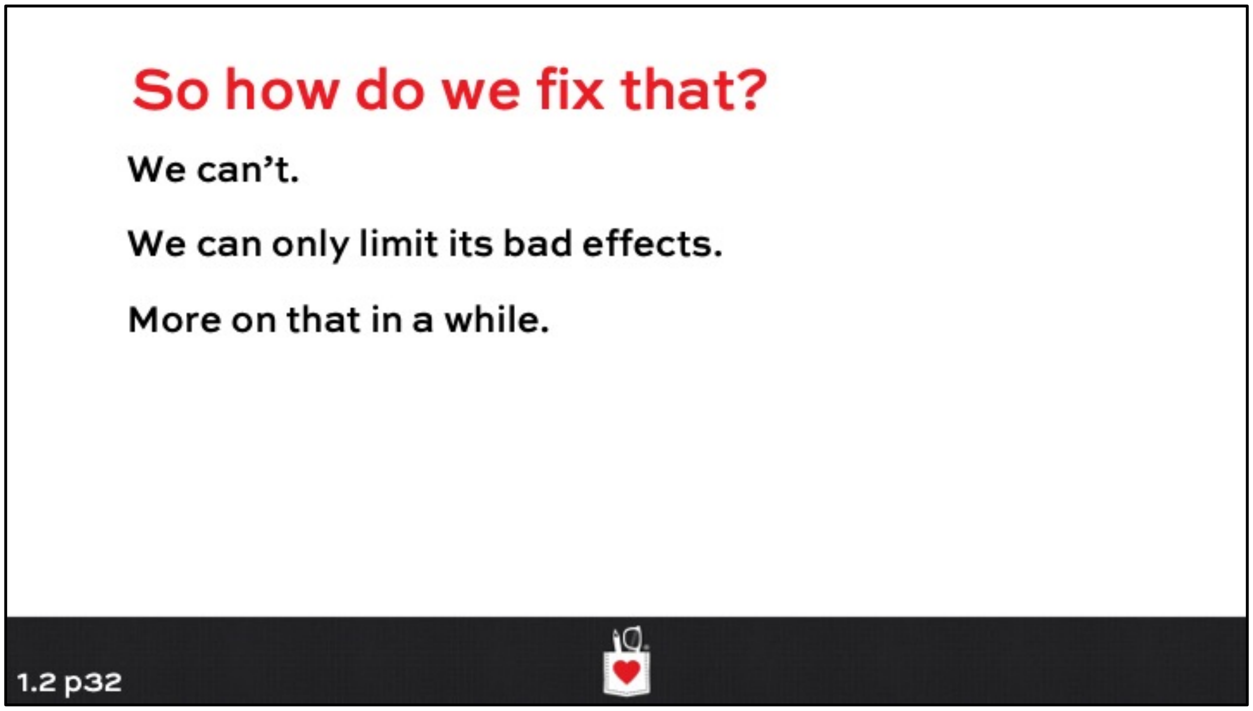
Please can the number of steps in the Statistics histogram be increased from the current 200. This could help alleviate potential issues around the the query optimizer making bad estimates due to not being able to see the true distribution of data in the table.

Under Review | Suggestions

Company Response

Microsoft Company Response
Up: 2

4 Votes | 0 Comments | Follow | Share | 0 Flags



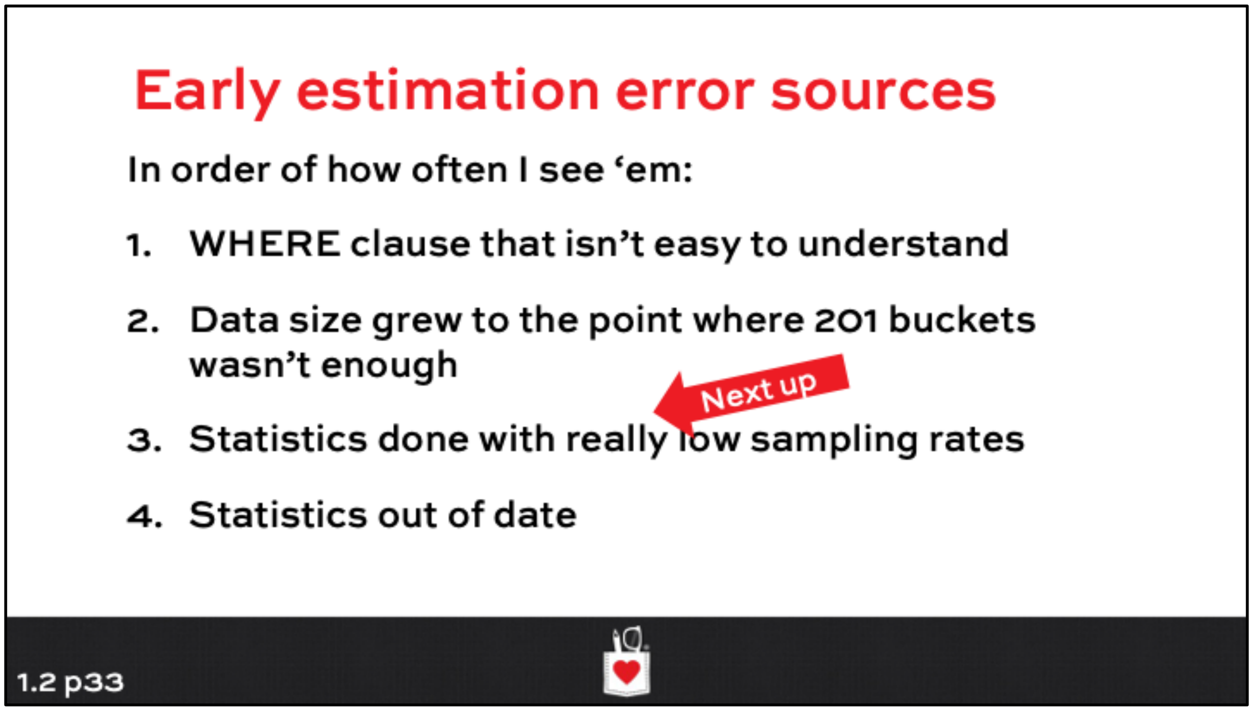
## So how do we fix that?

We can't.

We can only limit its bad effects.

More on that in a while.

1.2 p32

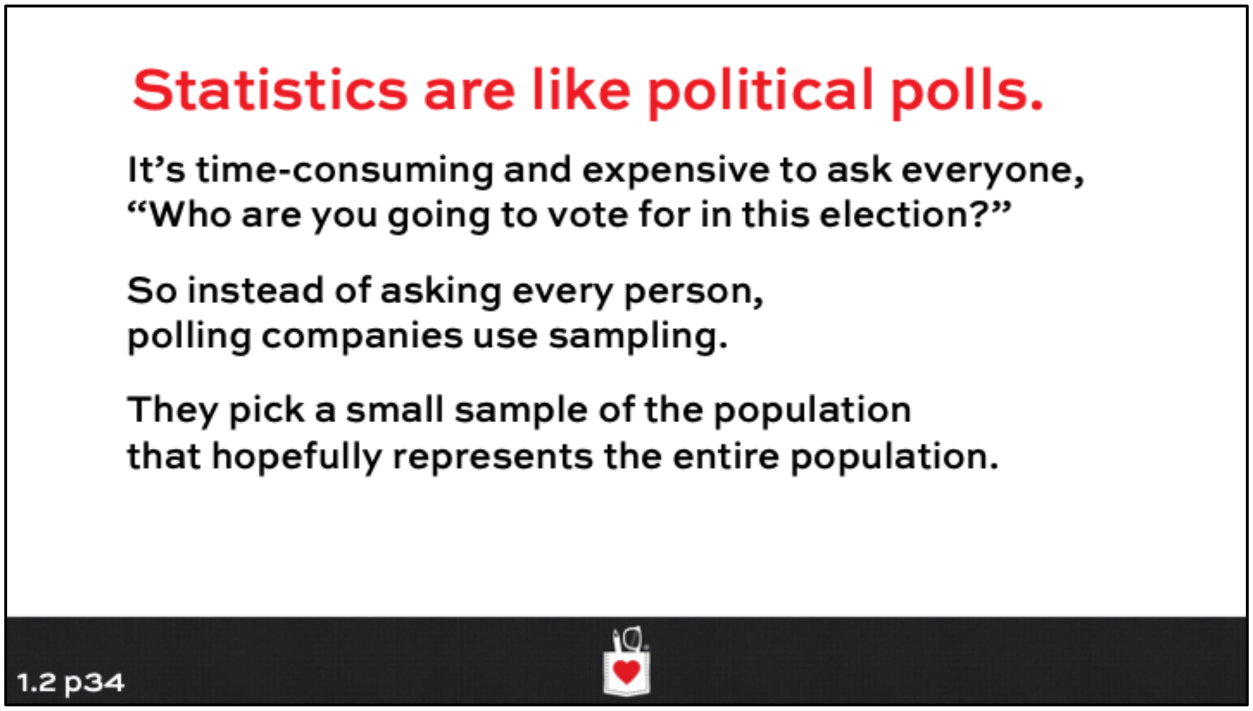


## Early estimation error sources

In order of how often I see 'em:

1. WHERE clause that isn't easy to understand
2. Data size grew to the point where 201 buckets wasn't enough
3. Statistics done with really low sampling rates
4. Statistics out of date

Next up
1.2 p33



## Statistics are like political polls.

It's time-consuming and expensive to ask everyone,
"Who are you going to vote for in this election?"

So instead of asking every person,
polling companies use sampling.

They pick a small sample of the population
that hopefully represents the entire population.

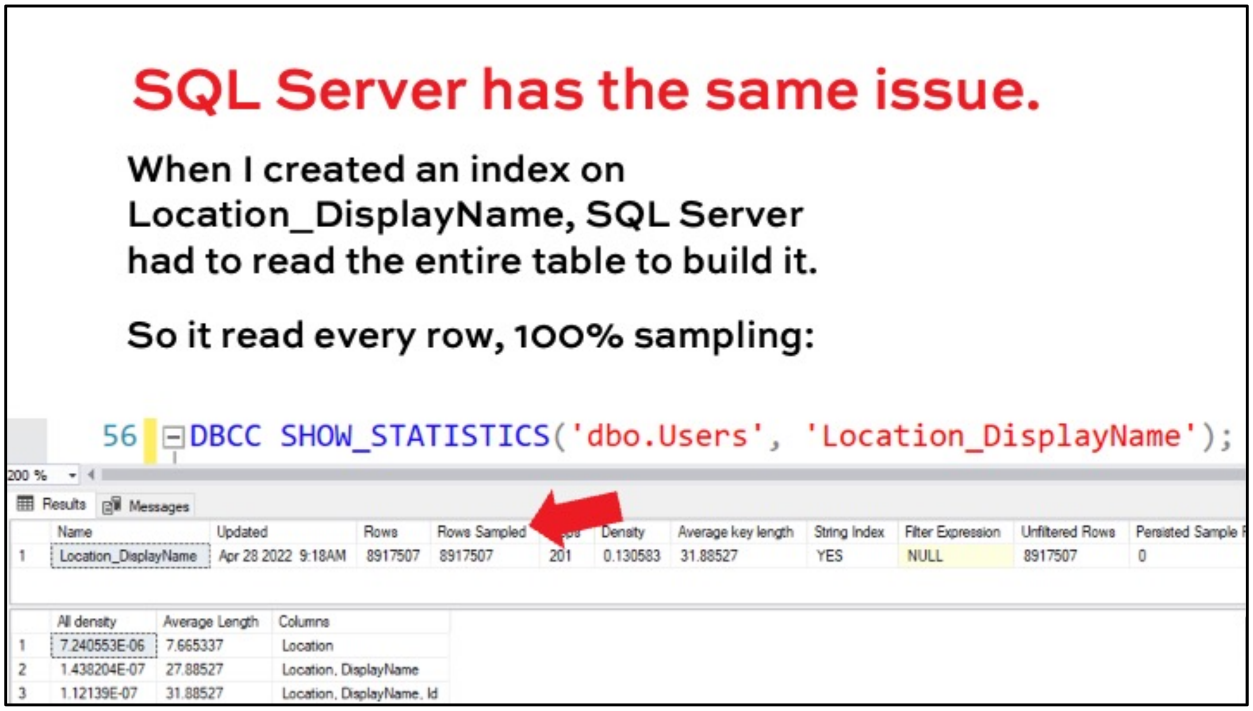
1.2 p34

## The sample can be way off.

1948: Chicago Daily Tribune relied on polls to write their headline of the presidential election.

They were wrong: Truman won.

1.2 p35



## SQL Server has the same issue.

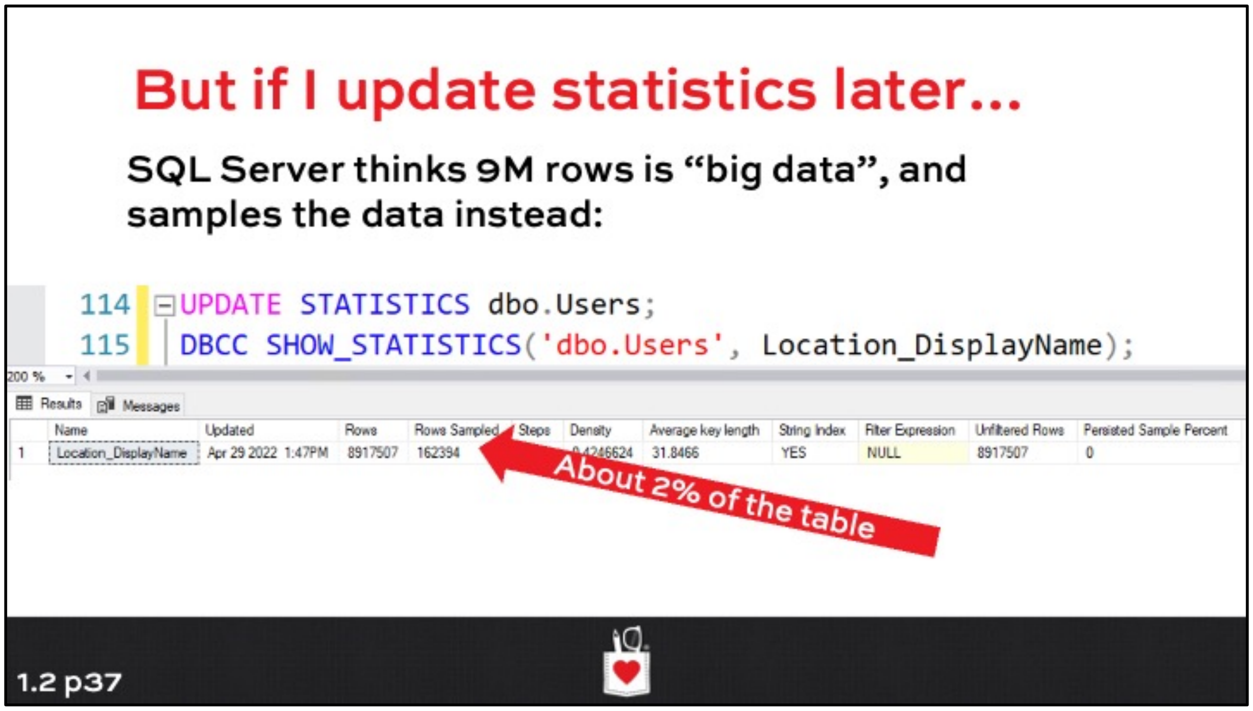
When I created an index on Location_DisplayName, SQL Server had to read the entire table to build it.

So it read every row, 100% sampling:

```sql
DBCC SHOW_STATISTICS('dbo.Users', 'Location_DisplayName');
```

|   | Name | Updated | Rows | Rows Sampled | [illegible] | Density | Average key length | String Index | Filter Expression | Unfiltered Rows | Persisted Sample |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Location_DisplayName | Apr 28 2022 9:18AM | 8917507 | 8917507 | 201 | 0.130583 | 31.88527 | YES | NULL | 8917507 | 0 |

|   | All density | Average Length | Columns |
|---|---|---|---|
| 1 | 7.240553E-06 | 7.665337 | Location |
| 2 | 1.438204E-07 | 27.88527 | Location, DisplayName |
| 3 | 1.12139E-07 | 31.88527 | Location, DisplayName, Id |



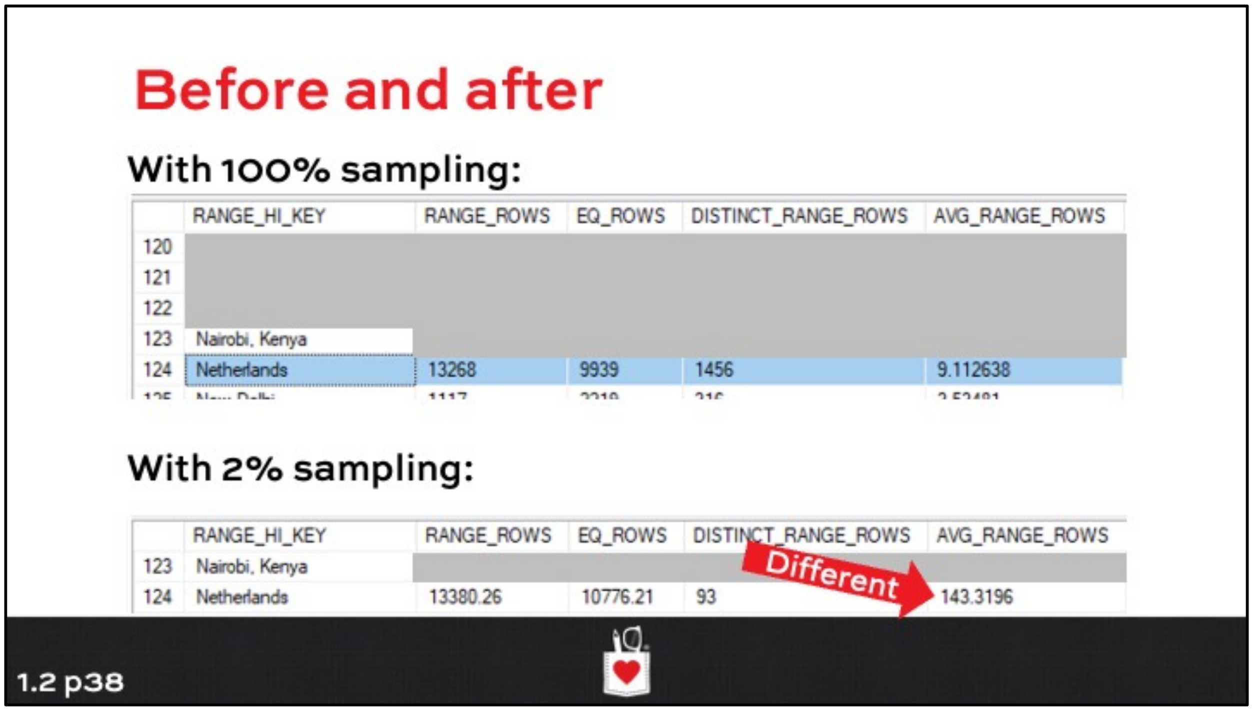
## But if I update statistics later…

SQL Server thinks 9M rows is "big data", and samples the data instead:

```
UPDATE STATISTICS dbo.Users;
DBCC SHOW_STATISTICS('dbo.Users', Location_DisplayName);
```

| | Name | Updated | Rows | Rows Sampled | Steps | Density | Average key length | String Index | Filter Expression | Unfiltered Rows | Persisted Sample Percent |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Location_DisplayName | Apr 29 2022 1:47PM | 8917507 | 162394 | | 0.4246624 | 31.8466 | YES | NULL | 8917507 | 0 |

1.2 p37



## Before and after

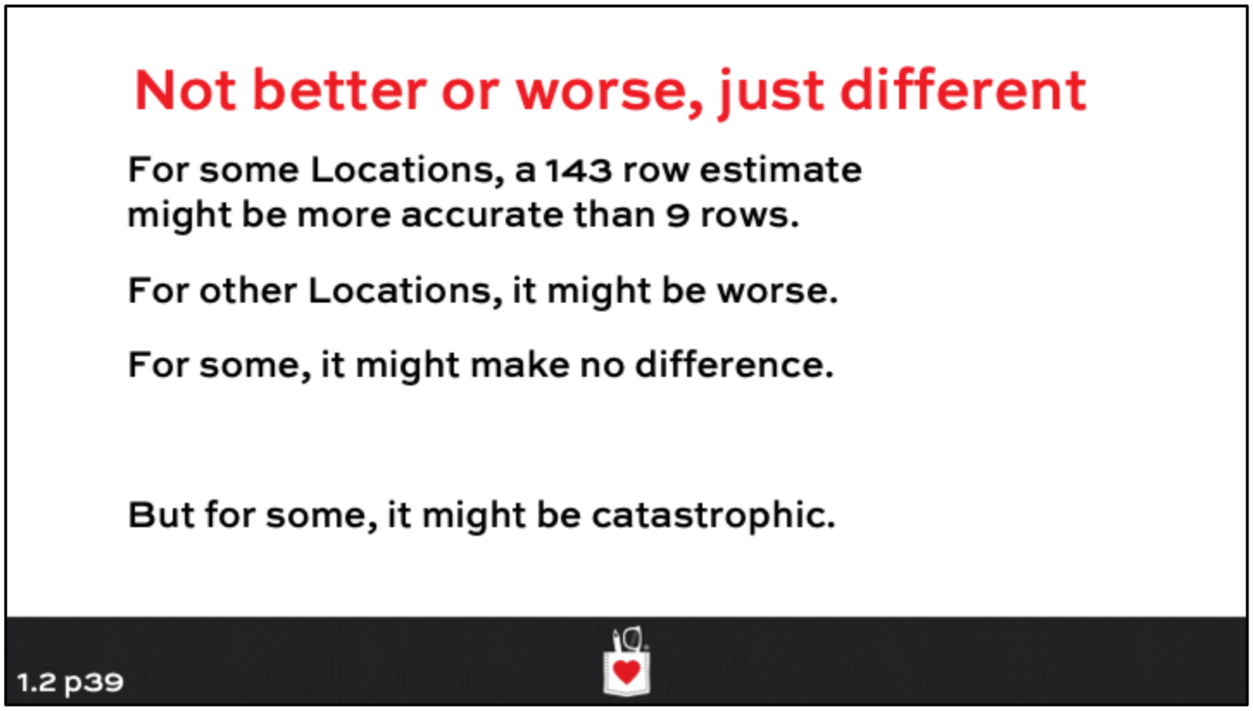
**With 100% sampling:**

| | RANGE_HI_KEY | RANGE_ROWS | EQ_ROWS | DISTINCT_RANGE_ROWS | AVG_RANGE_ROWS |
|---|---|---|---|---|---|
| 120 | | | | | |
| 121 | | | | | |
| 122 | | | | | |
| 123 | Nairobi, Kenya | | | | |
| 124 | Netherlands | 13268 | 9939 | 1456 | 9.112638 |
| 125 | New Delhi | 1117 | 2219 | 216 | 2.52481 |

**With 2% sampling:**

| | RANGE_HI_KEY | RANGE_ROWS | EQ_ROWS | DISTINCT_RANGE_ROWS | AVG_RANGE_ROWS |
|---|---|---|---|---|---|
| 123 | Nairobi, Kenya | | | | |
| 124 | Netherlands | 13380.26 | 10776.21 | 93 | 143.3196 |

1.2 p38



## Not better or worse, just different

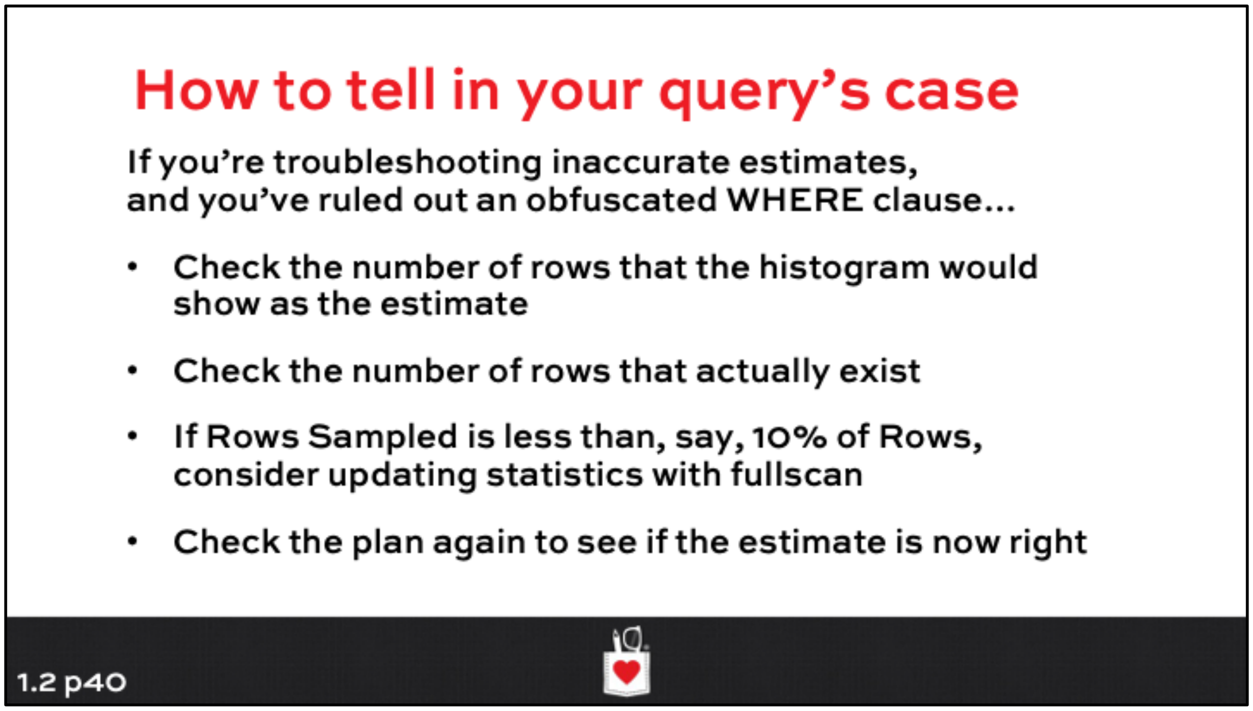
For some Locations, a 143 row estimate might be more accurate than 9 rows.

For other Locations, it might be worse.

For some, it might make no difference.

But for some, it might be catastrophic.

1.2 p39



## How to tell in your query's case

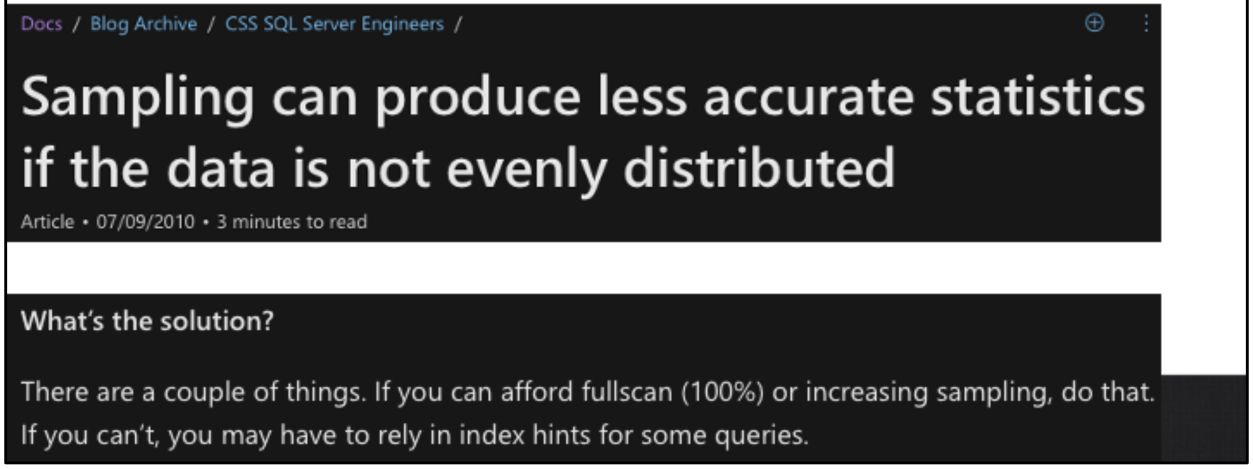
If you're troubleshooting inaccurate estimates,
and you've ruled out an obfuscated WHERE clause…

- Check the number of rows that the histogram would show as the estimate
- Check the number of rows that actually exist
- If Rows Sampled is less than, say, 10% of Rows, consider updating statistics with fullscan
- Check the plan again to see if the estimate is now right

1.2 p40

## Microsoft blogged about this

https://docs.microsoft.com/en-us/archive/blogs/psssql/sampling-can-produce-less-accurate-statistics-if-the-data-is-not-evenly-distributed

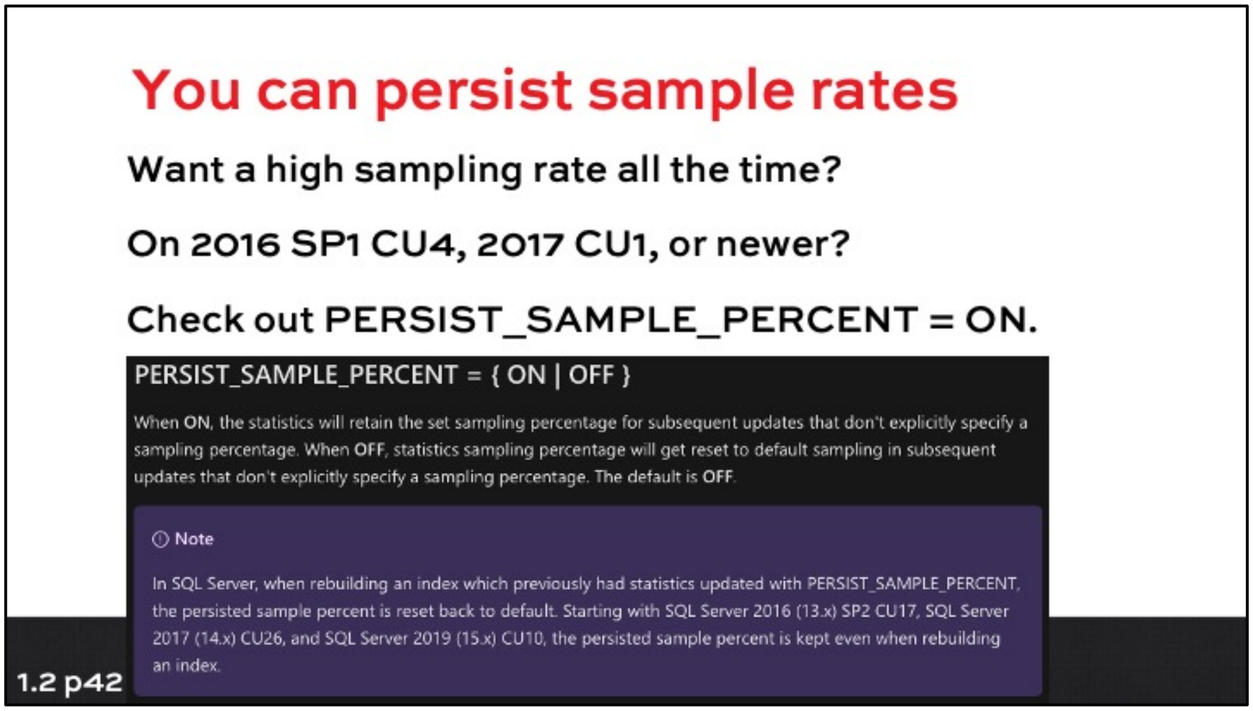

Docs / Blog Archive / CSS SQL Server Engineers /

Sampling can produce less accurate statistics if the data is not evenly distributed

Article • 07/09/2010 • 3 minutes to read

What's the solution?

There are a couple of things. If you can afford fullscan (100%) or increasing sampling, do that. If you can't, you may have to rely in index hints for some queries.



## You can persist sample rates

**Want a high sampling rate all the time?**

**On 2016 SP1 CU4, 2017 CU1, or newer?**

**Check out PERSIST_SAMPLE_PERCENT = ON.**

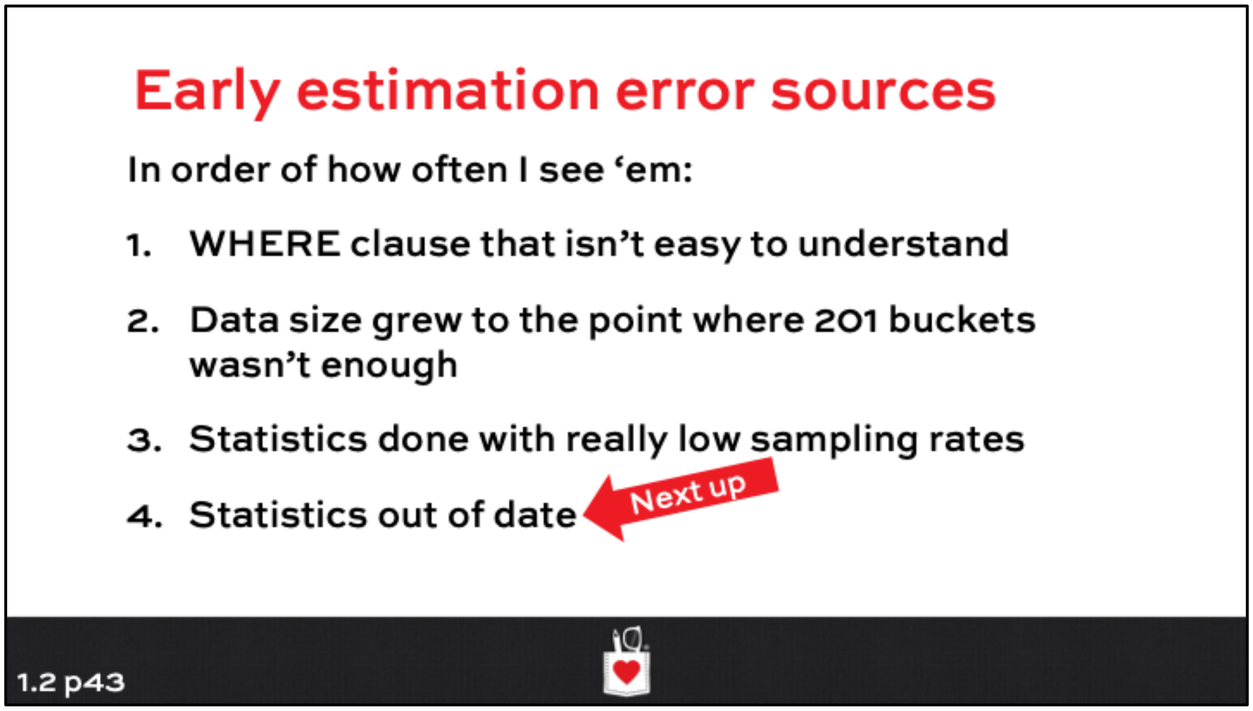
PERSIST_SAMPLE_PERCENT = { ON | OFF }

When **ON**, the statistics will retain the set sampling percentage for subsequent updates that don't explicitly specify a sampling percentage. When **OFF**, statistics sampling percentage will get reset to default sampling in subsequent updates that don't explicitly specify a sampling percentage. The default is **OFF**.

ⓘ Note

In SQL Server, when rebuilding an index which previously had statistics updated with PERSIST_SAMPLE_PERCENT, the persisted sample percent is reset back to default. Starting with SQL Server 2016 (13.x) SP2 CU17, SQL Server 2017 (14.x) CU26, and SQL Server 2019 (15.x) CU10, the persisted sample percent is kept even when rebuilding an index.

1.2 p42

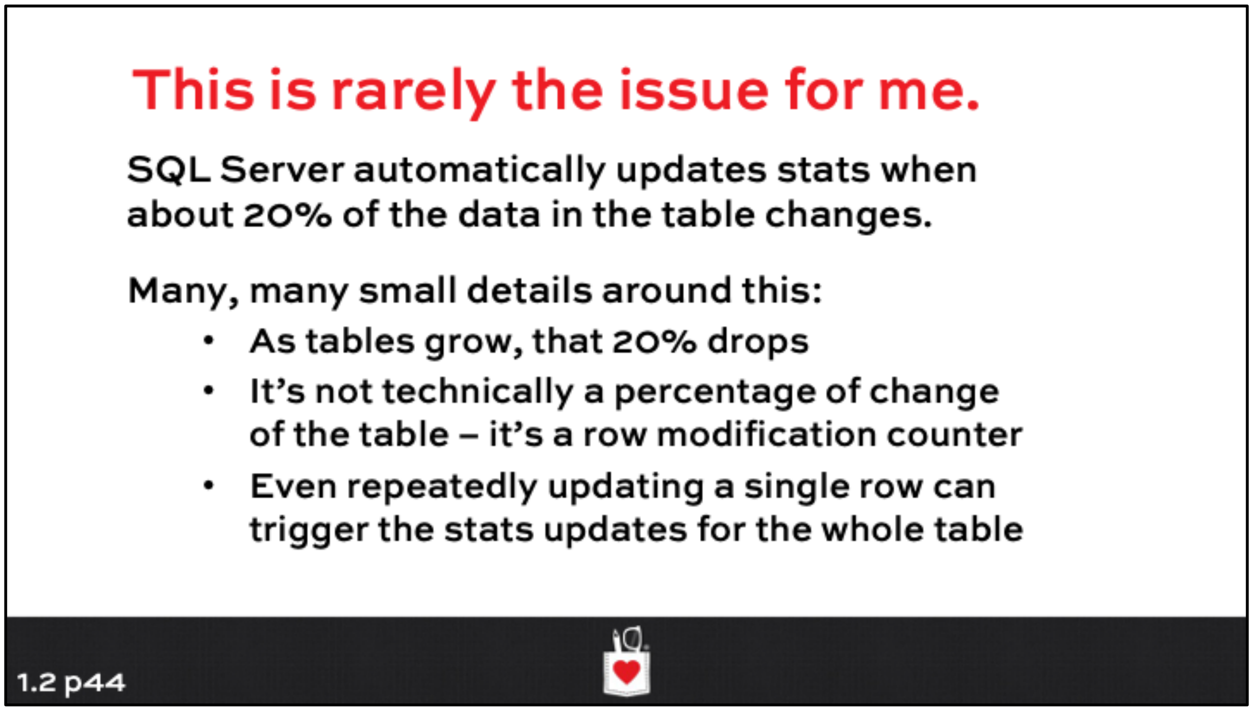


## Early estimation error sources

In order of how often I see ‘em:

1. WHERE clause that isn’t easy to understand
2. Data size grew to the point where 201 buckets wasn’t enough
3. Statistics done with really low sampling rates
4. Statistics out of date

1.2 p43



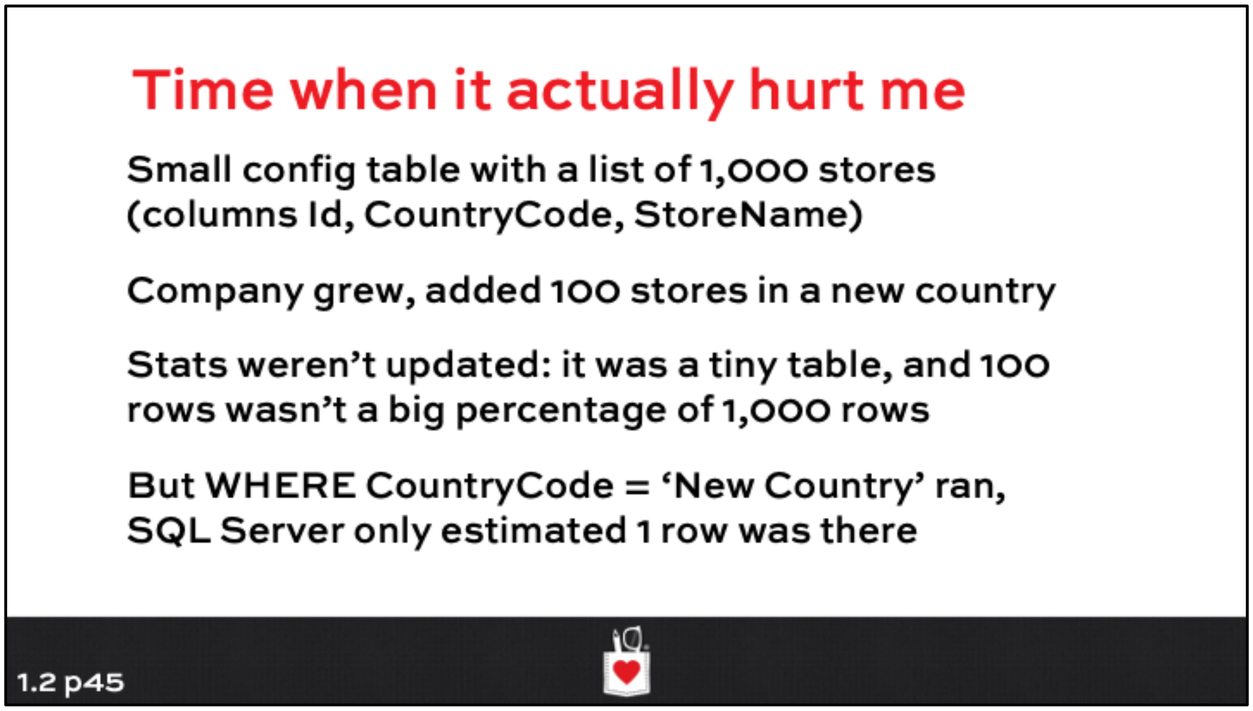
## This is rarely the issue for me.

SQL Server automatically updates stats when about 20% of the data in the table changes.

Many, many small details around this:

- As tables grow, that 20% drops
- It's not technically a percentage of change of the table – it's a row modification counter
- Even repeatedly updating a single row can trigger the stats updates for the whole table

1.2 p44



## Time when it actually hurt me

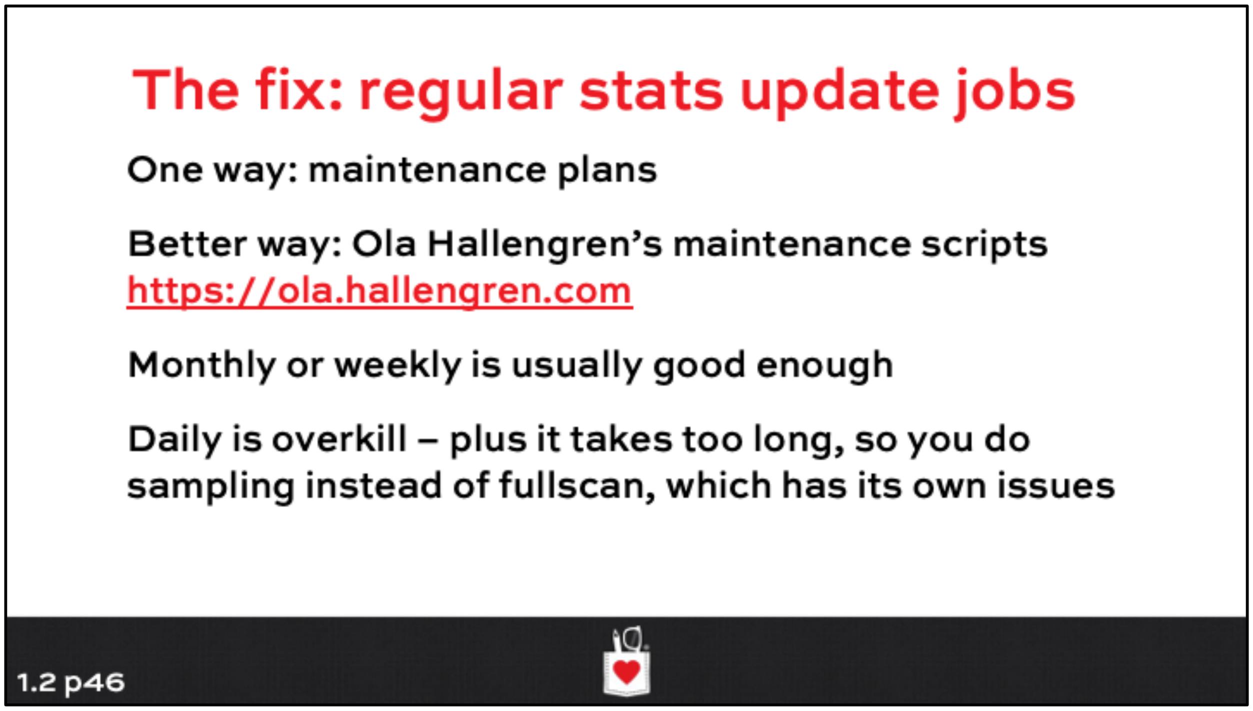
Small config table with a list of 1,000 stores (columns Id, CountryCode, StoreName)

Company grew, added 100 stores in a new country

Stats weren't updated: it was a tiny table, and 100 rows wasn't a big percentage of 1,000 rows

But WHERE CountryCode = 'New Country' ran, SQL Server only estimated 1 row was there

1.2 p45



## The fix: regular stats update jobs

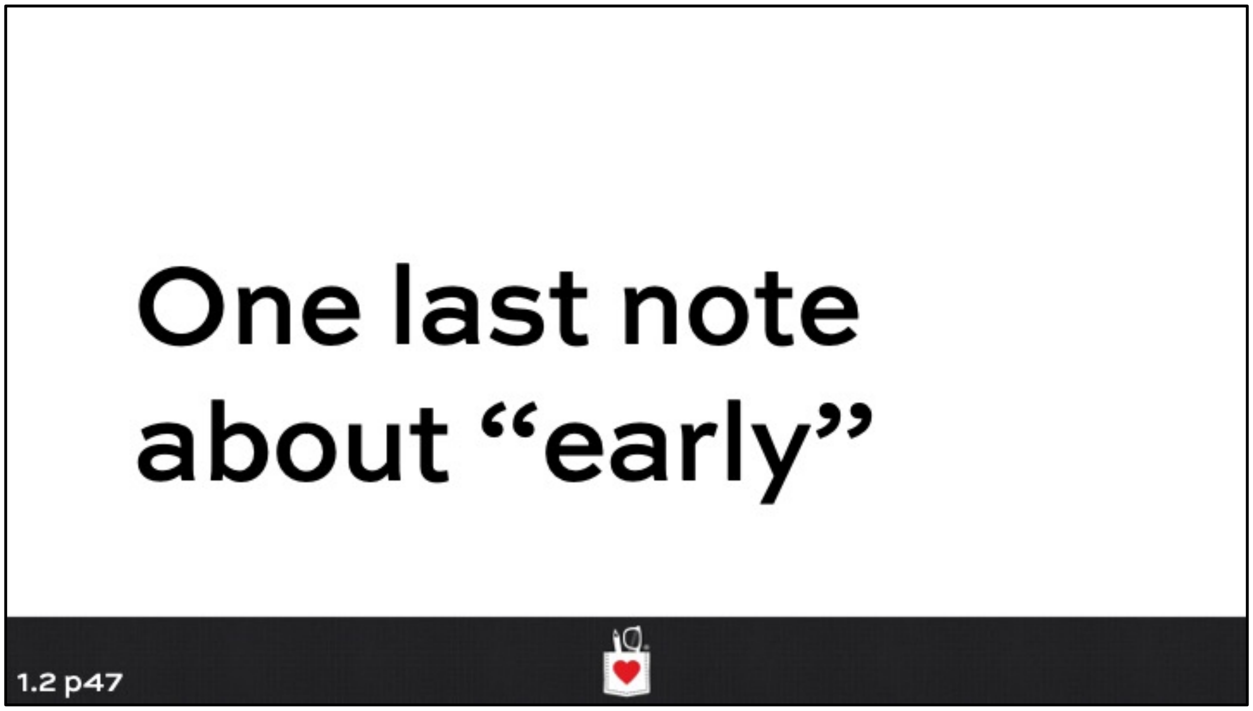
One way: maintenance plans

Better way: Ola Hallengren's maintenance scripts
https://ola.hallengren.com

Monthly or weekly is usually good enough

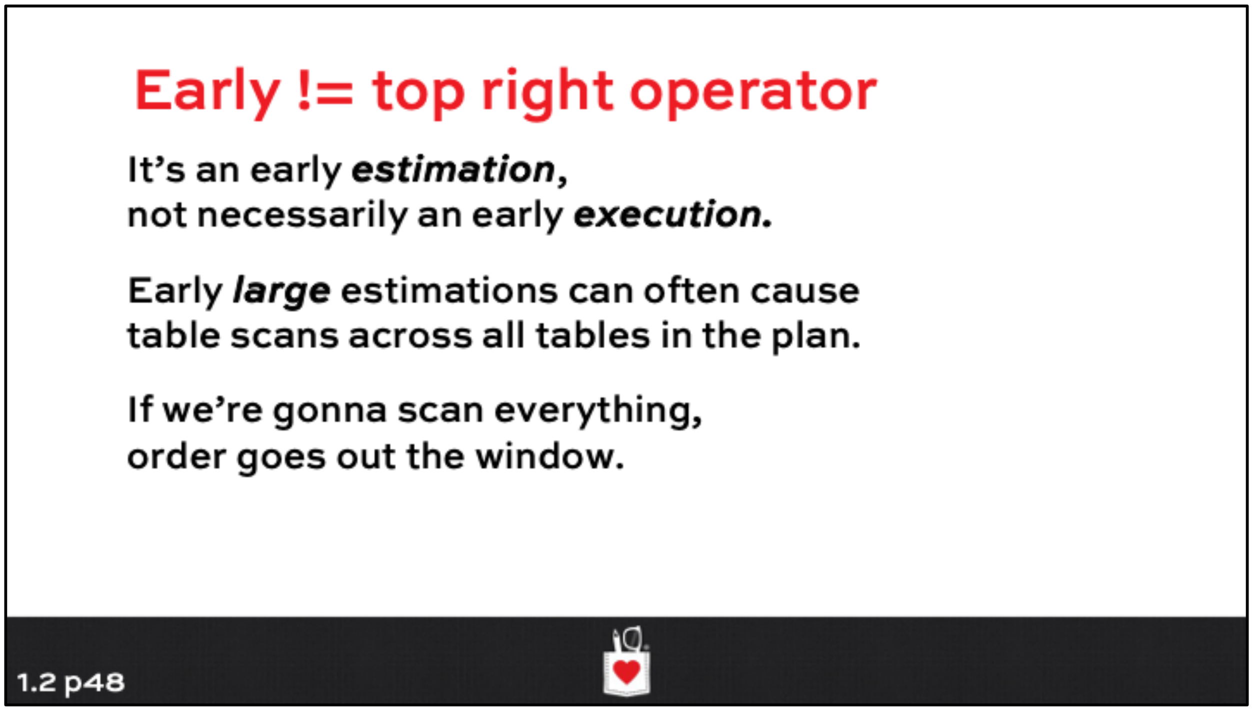
Daily is overkill – plus it takes too long, so you do sampling instead of fullscan, which has its own issues

1.2 p46



## One last note about “early”

1.2 p47



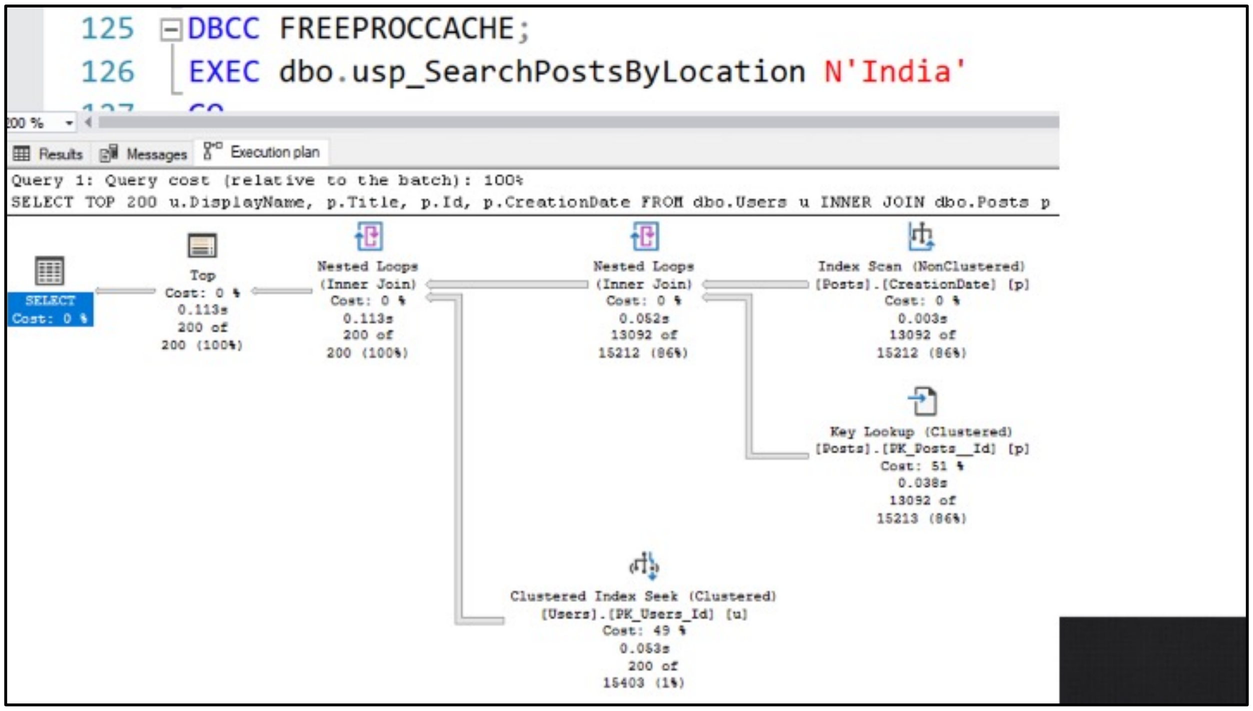
## Early != top right operator

It's an early ***estimation***,
not necessarily an early ***execution.***

Early ***large*** estimations can often cause
table scans across all tables in the plan.

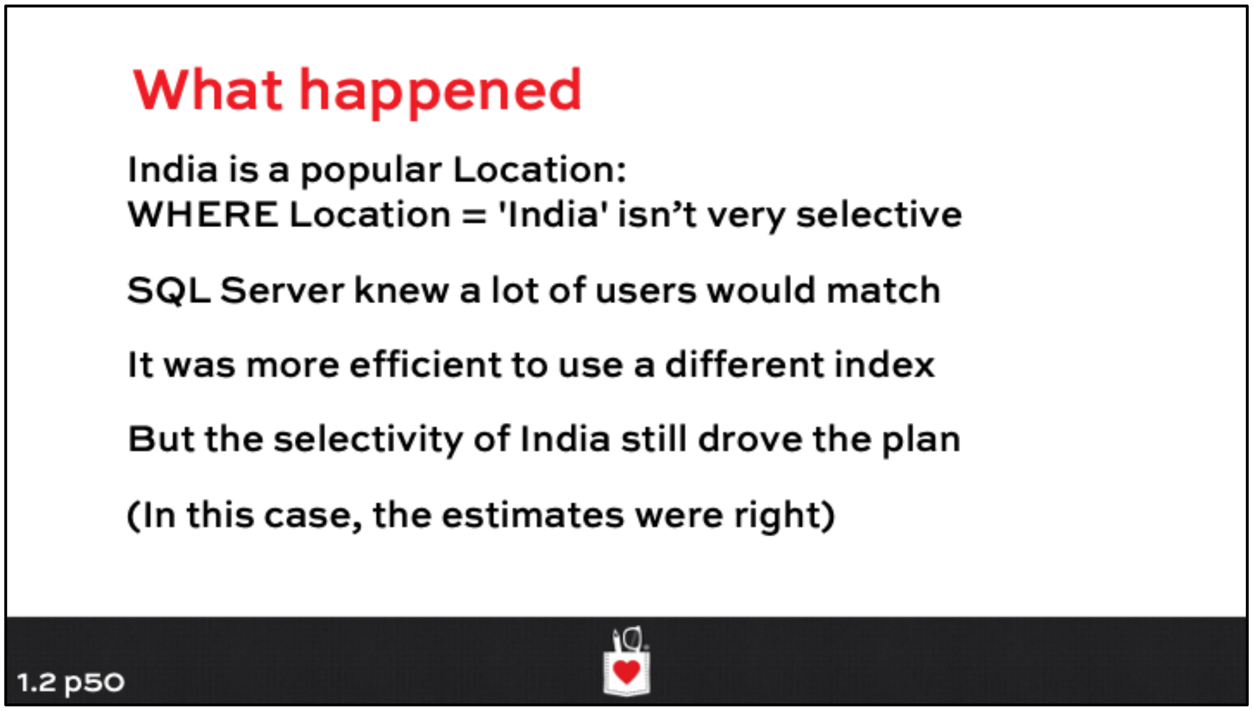
If we're gonna scan everything,
order goes out the window.

1.2 p48



```
DBCC FREEPROCCACHE;
EXEC dbo.usp_SearchPostsByLocation N'India'
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 200 u.DisplayName, p.Title, p.Id, p.CreationDate FROM dbo.Users u INNER JOIN dbo.Posts p
```



## What happened

India is a popular Location:
WHERE Location = 'India' isn't very selective

SQL Server knew a lot of users would match

It was more efficient to use a different index

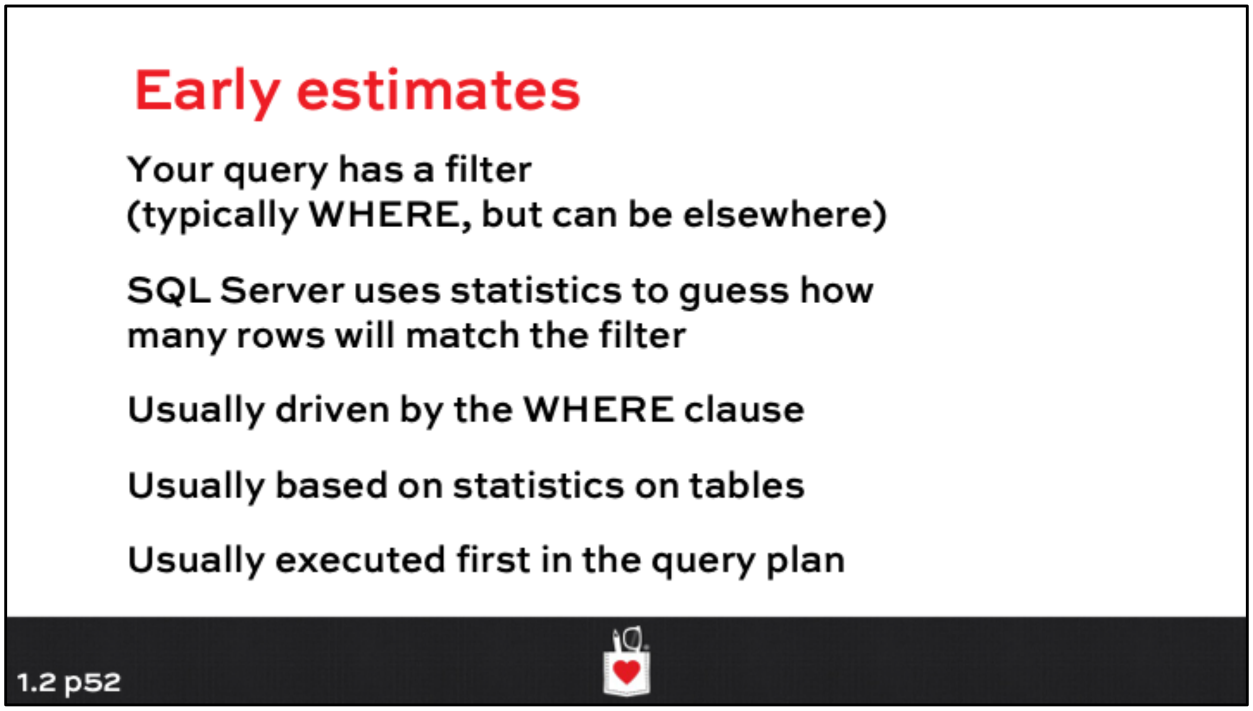
But the selectivity of India still drove the plan

(In this case, the estimates were right)

1.2 p50

## Recap

1.2 p51



## Early estimates

Your query has a filter
(typically WHERE, but can be elsewhere)

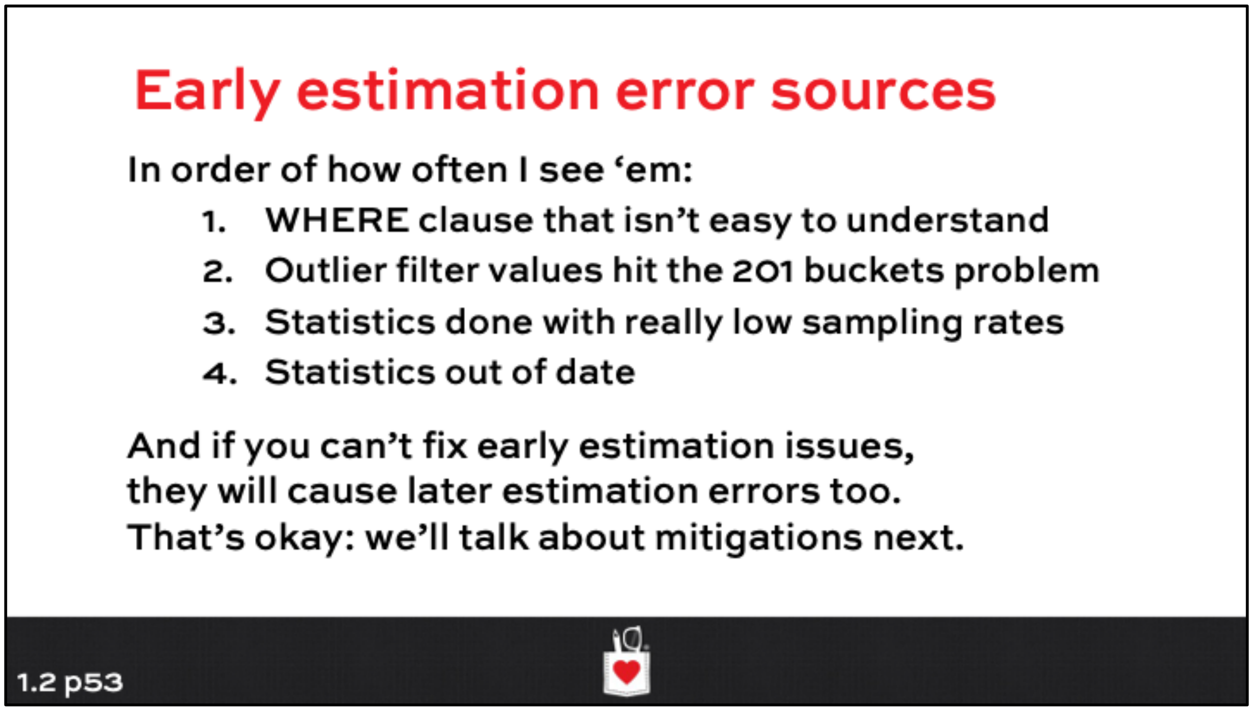
SQL Server uses statistics to guess how
many rows will match the filter

Usually driven by the WHERE clause

Usually based on statistics on tables

Usually executed first in the query plan

1.2 p52



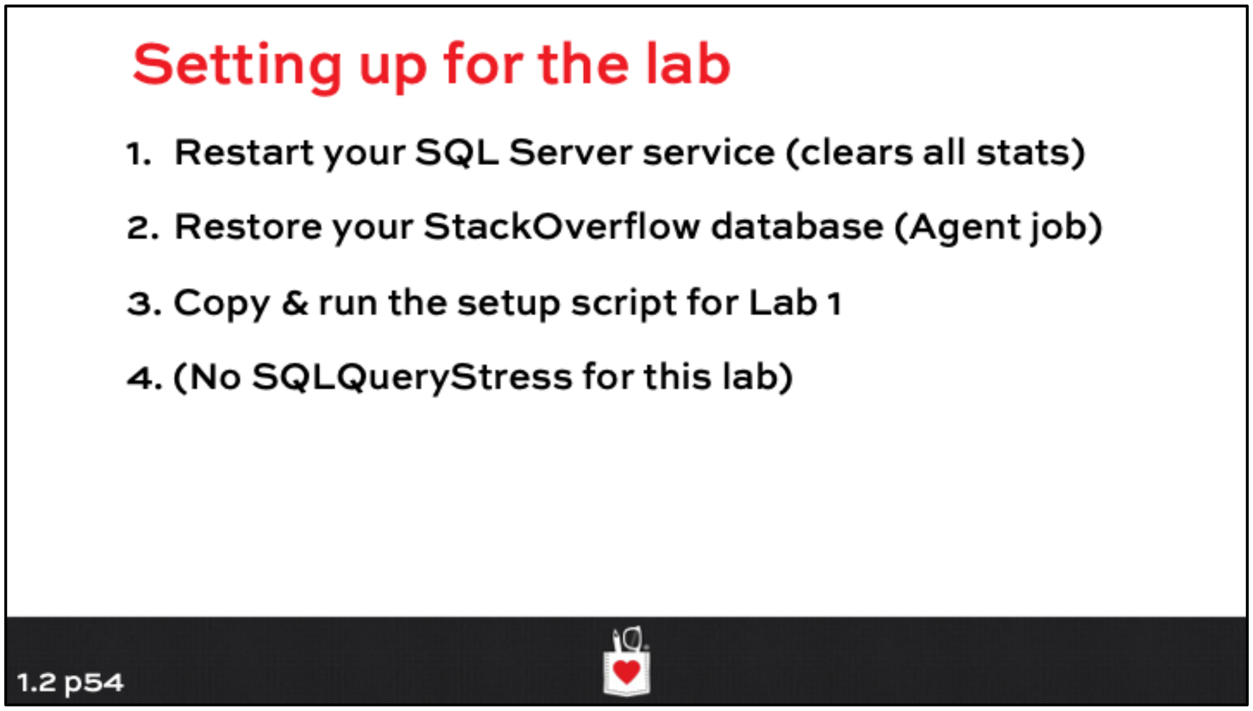
## Early estimation error sources

In order of how often I see 'em:

1. WHERE clause that isn't easy to understand
2. Outlier filter values hit the 201 buckets problem
3. Statistics done with really low sampling rates
4. Statistics out of date

And if you can't fix early estimation issues,
they will cause later estimation errors too.
That's okay: we'll talk about mitigations next.

1.2 p53



## Setting up for the lab

1. Restart your SQL Server service (clears all stats)
2. Restore your StackOverflow database (Agent job)
3. Copy & run the setup script for Lab 1
4. (No SQLQueryStress for this lab)

1.2 p54