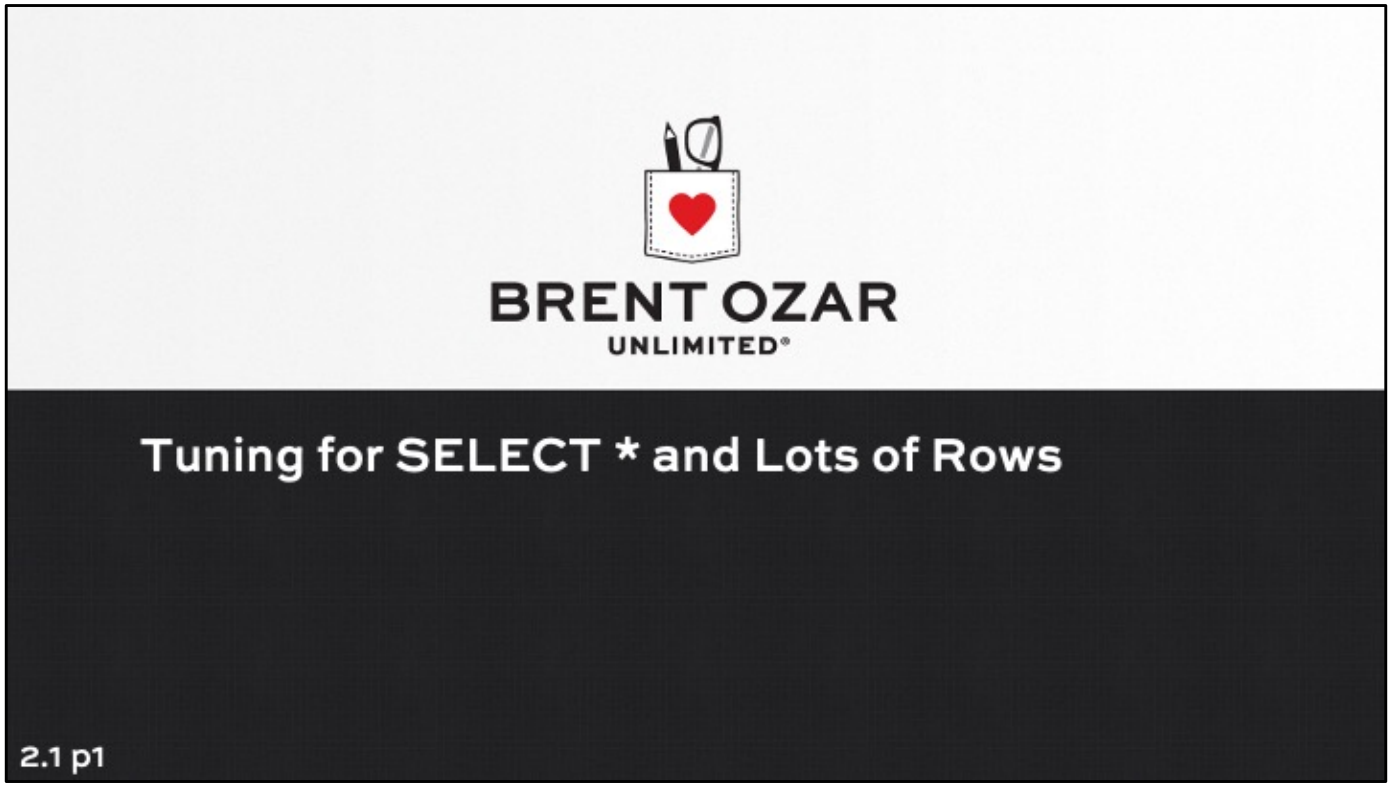
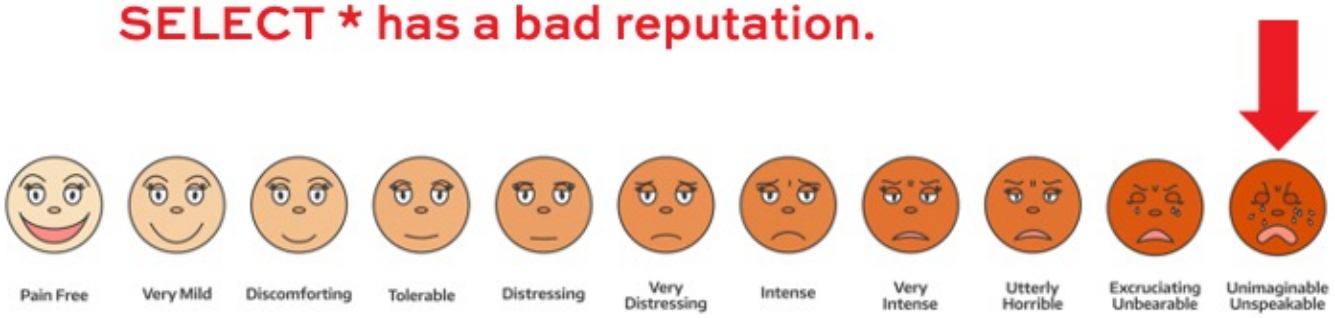


SELECT * has a bad reputation.



But let's test a few scenarios to see what we really need to fix.

2.1 p2

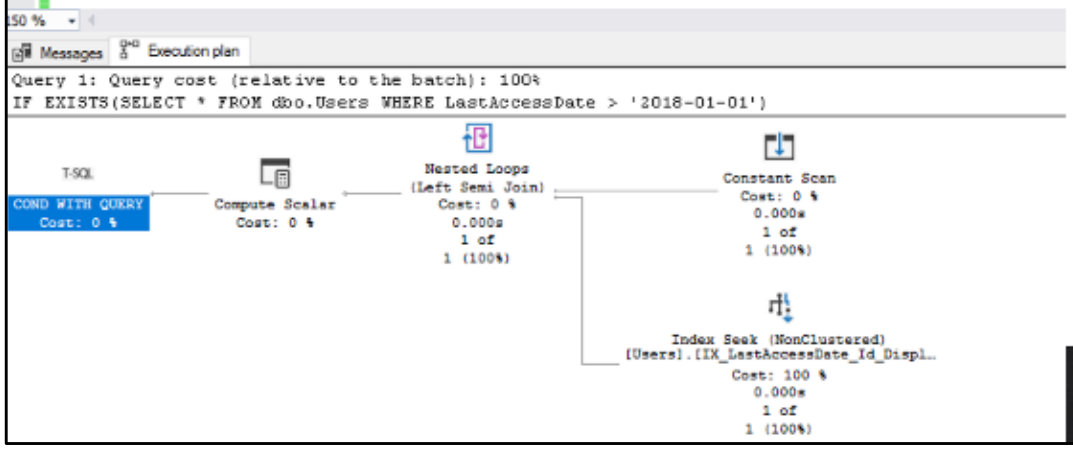
IF EXISTS (SELECT *

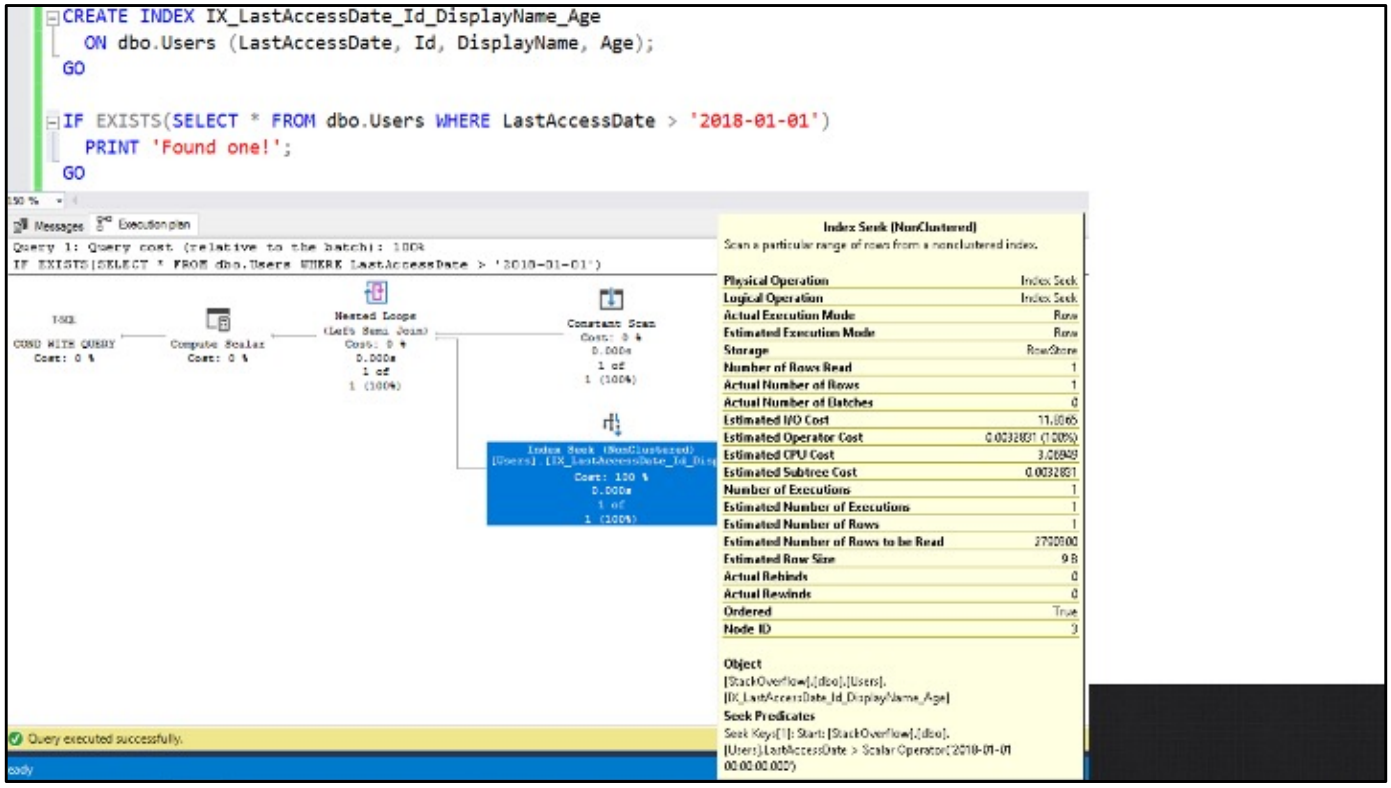
2.1 p3

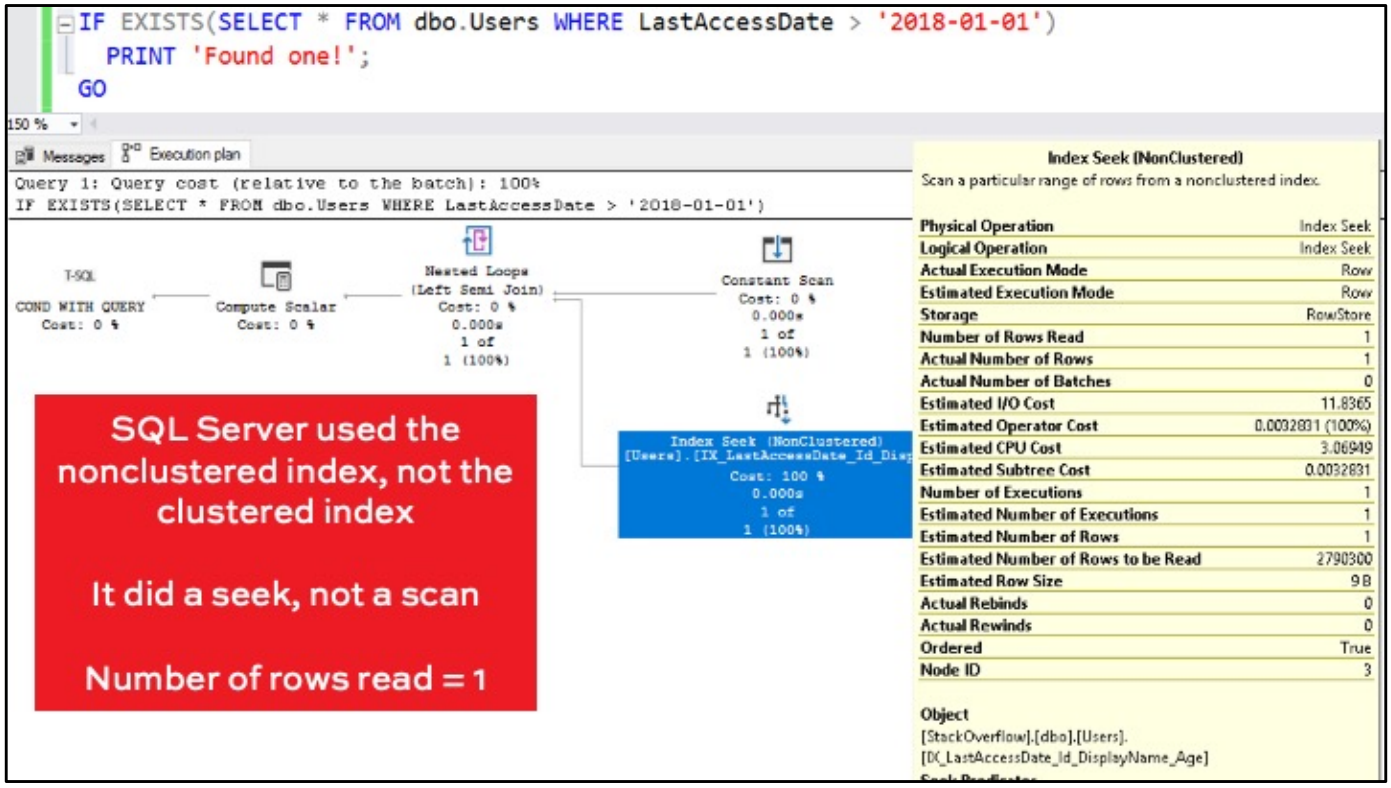
```sql
/* Is SELECT * a big deal in IF-EXISTS queries?

Say we've got this index from How to Think Like the Engine: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

IF EXISTS(SELECT * FROM dbo.Users WHERE LastAccessDate > '2018-01-01')
  PRINT 'Found one!';
GO
```

2.1 p4

```sql
/* Is SELECT * a big deal in IF-EXISTS queries?

Say we've got this index from How to Think Like the Engine: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

IF EXISTS(SELECT * FROM dbo.Users WHERE LastAccessDate > '2018-01-01')
  PRINT 'Found one!';
GO
```







More proof in SET STATISTICS IO ON

```
IF EXISTS(SELECT * FROM dbo.Users WHERE LastAccessDate > '2018-01-01')
  PRINT 'Found one!';
GO
```

50 %

Messages | Execution plan

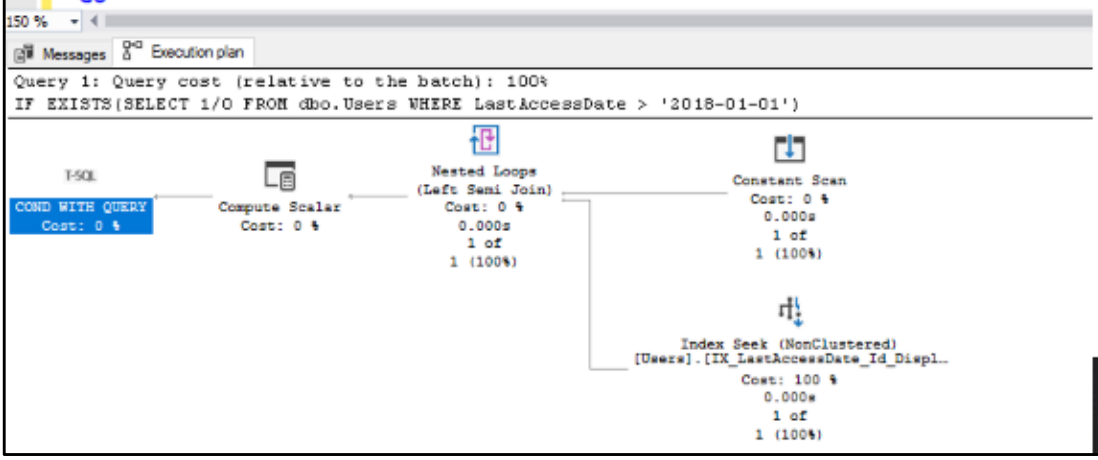
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.
Table 'Users'. Scan count 1, logical reads 4, physical reads 0, read-ahead rea
```

2.1 p8

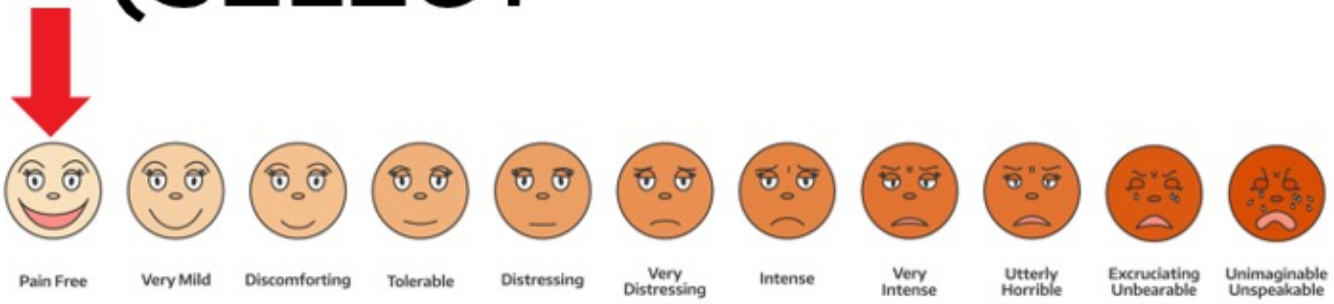
SQL Server optimizes the * away here

Heck, it'll optimize away just about anything in this scenario, like 1/0:

```
IF EXISTS(SELECT 1/0 FROM dbo.Users WHERE LastAccessDate > '2018-01-01'
  PRINT 'Found one!';
GO
```



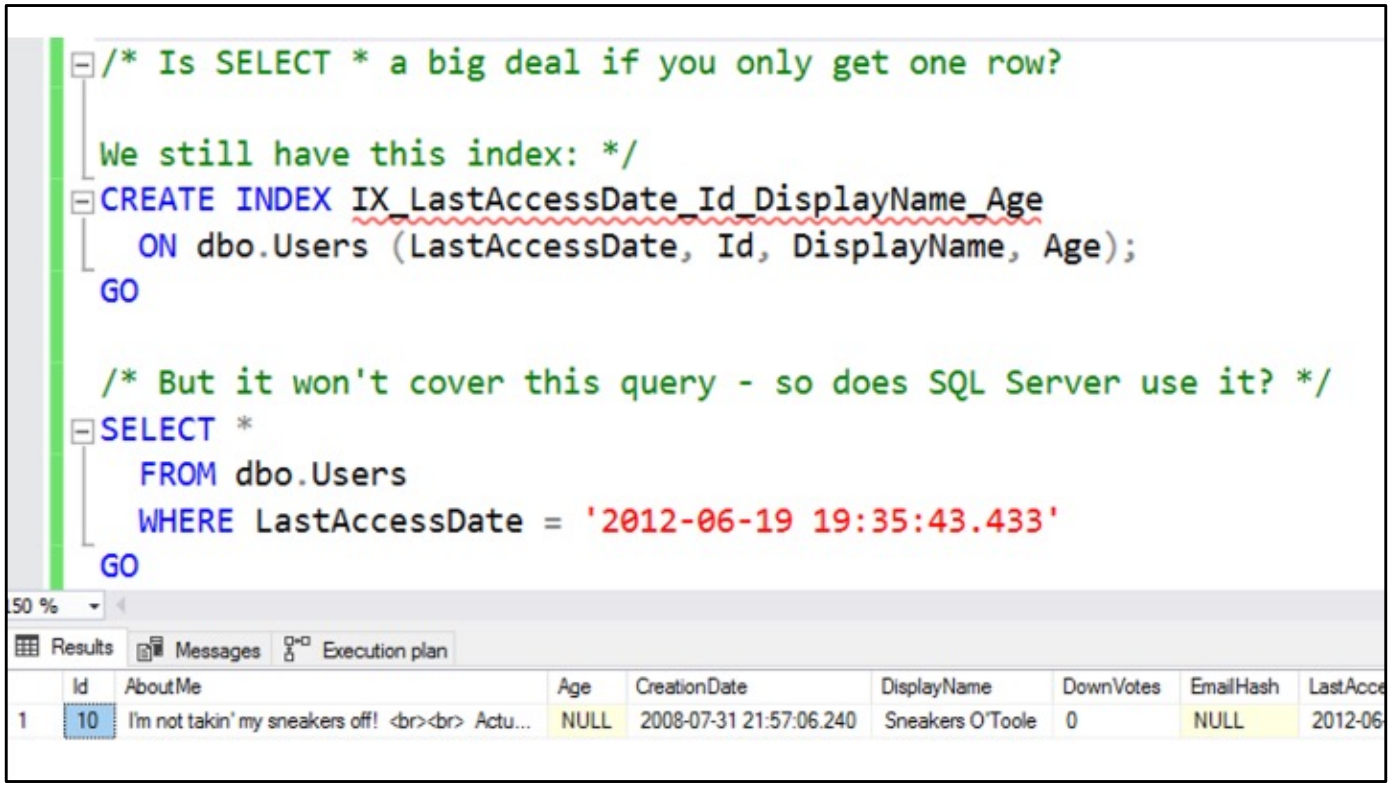
IF EXISTS (SELECT *



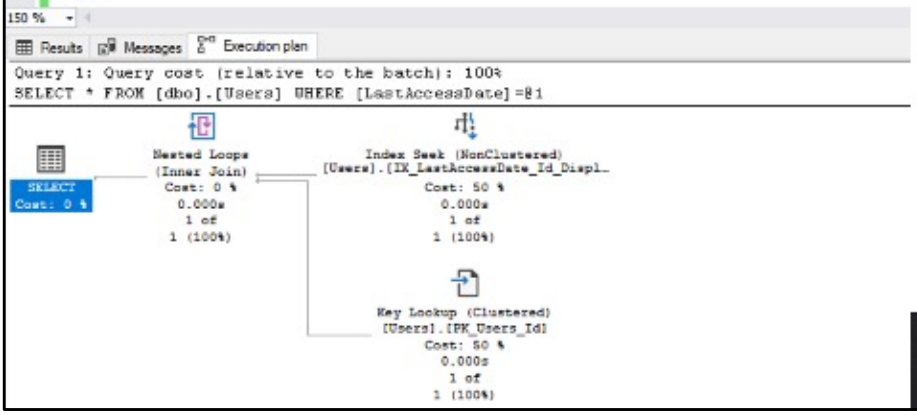
2.1 p10

Next up:

GETTING ONE ROW



```sql
/* Is SELECT * a big deal if you only get one row?

We still have this index: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

/* But it won't cover this query - so does SQL Server use it? */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate = '2012-06-19 19:35:43.433'
GO
```



Yes.

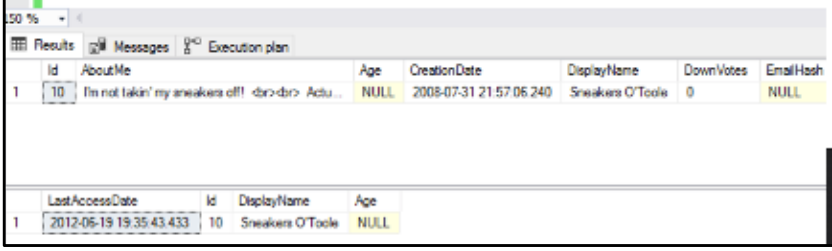
SQL Server starts with an index seek...

Then does a key lookup to output the additional columns (*) that weren't included on the index.

Is SELECT * much more work?

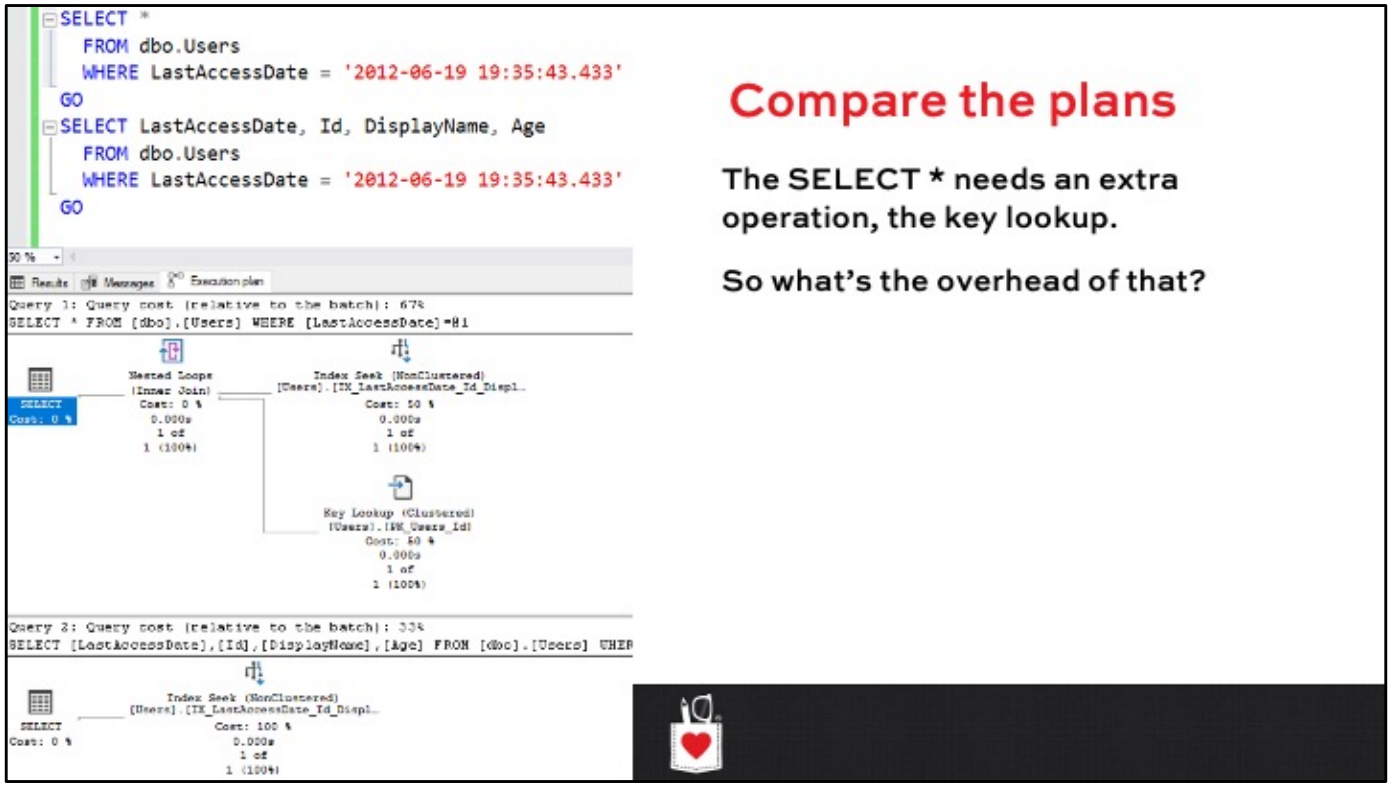
We'll run two versions – one asking for all the columns, and one only asking for columns in the nonclustered index.

```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate = '2012-06-19 19:35:43.433'
GO
SELECT LastAccessDate, Id, DisplayName, Age
  FROM dbo.Users
  WHERE LastAccessDate = '2012-06-19 19:35:43.433'
GO
```



	Id	AboutMe	Age	CreationDate	DisplayName	DownVotes	EmailHash
1	10	I'm not takin' my sneakers off! Actu...	NULL	2008-07-31 21:57:06.240	Sneakers O'Toole	0	NULL

	LastAccessDate	Id	DisplayName	Age
1	2012-06-19 19:35:43.433	10	Sneakers O'Toole	NULL



```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate = '2012-06-19 19:35:43.433'
GO
SELECT LastAccessDate, Id, DisplayName, Age
  FROM dbo.Users
  WHERE LastAccessDate = '2012-06-19 19:35:43.433'
```

150 % ▾

Results | Messages | Execution plan

```
(1 row affected)
Table 'Users'. Scan count 1, logical reads 7, physical re

(1 row affected)

(1 row affected)
Table 'Users'. Scan count 1, logical reads 4, physical re

(1 row affected)
```

7 reads vs 4 reads.

We read an extra 24 kilobytes.

That's...not really a big deal for one query.

(Although it's big if this query runs thousands of times per second.)

2.1 p16

I wouldn't try to cover that SELECT *.

If someone's only getting one row at a time,

and that query only runs a few times per second,

it's not worth building a covering index for that,

nor is it worth asking them to change their query.

ONE ROW

2.1 p18

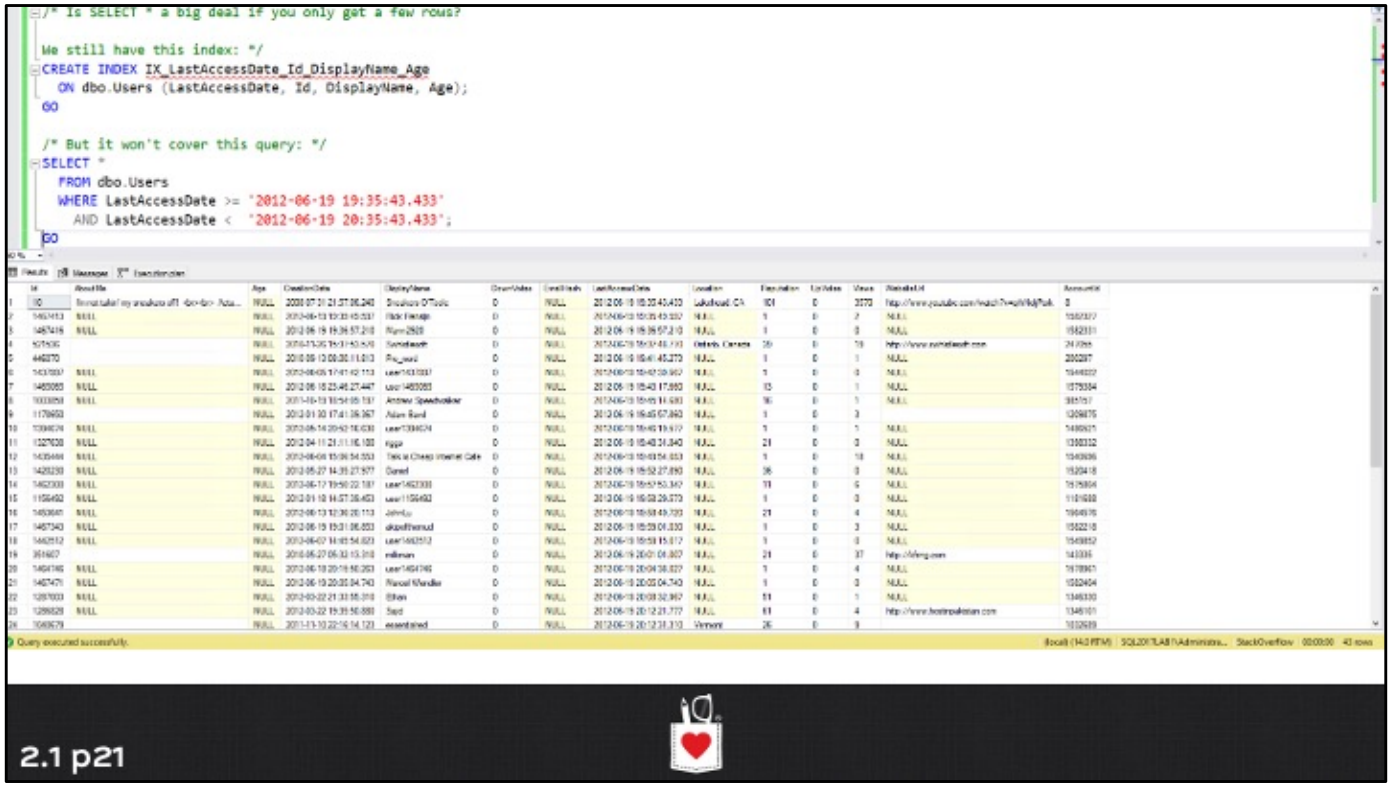
Next up:

GETTING A FEW ROWS

2.1 p19

```sql
/* Is SELECT * a big deal if you only get a few rows?

We still have this index: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

/* But it won't cover this query: */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
```

2.1 p20

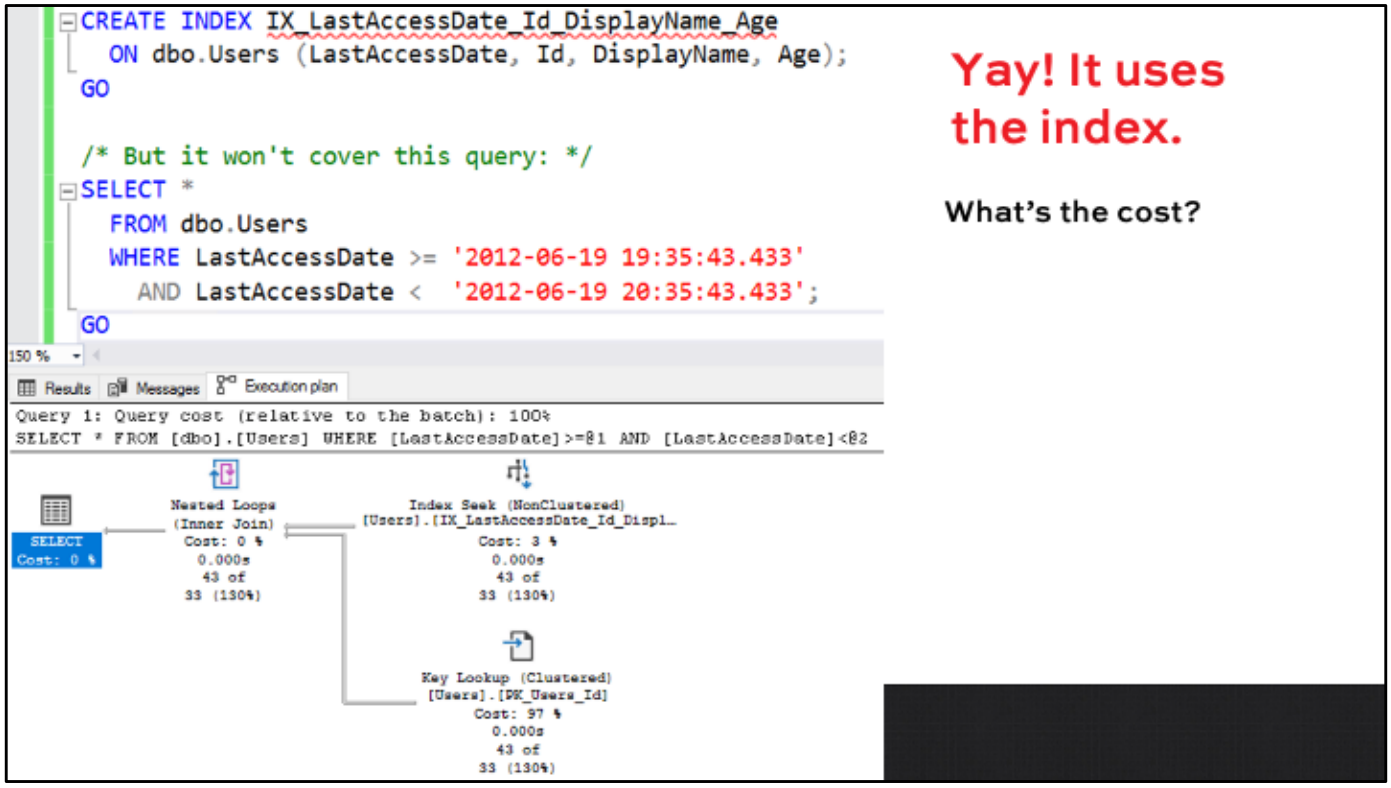


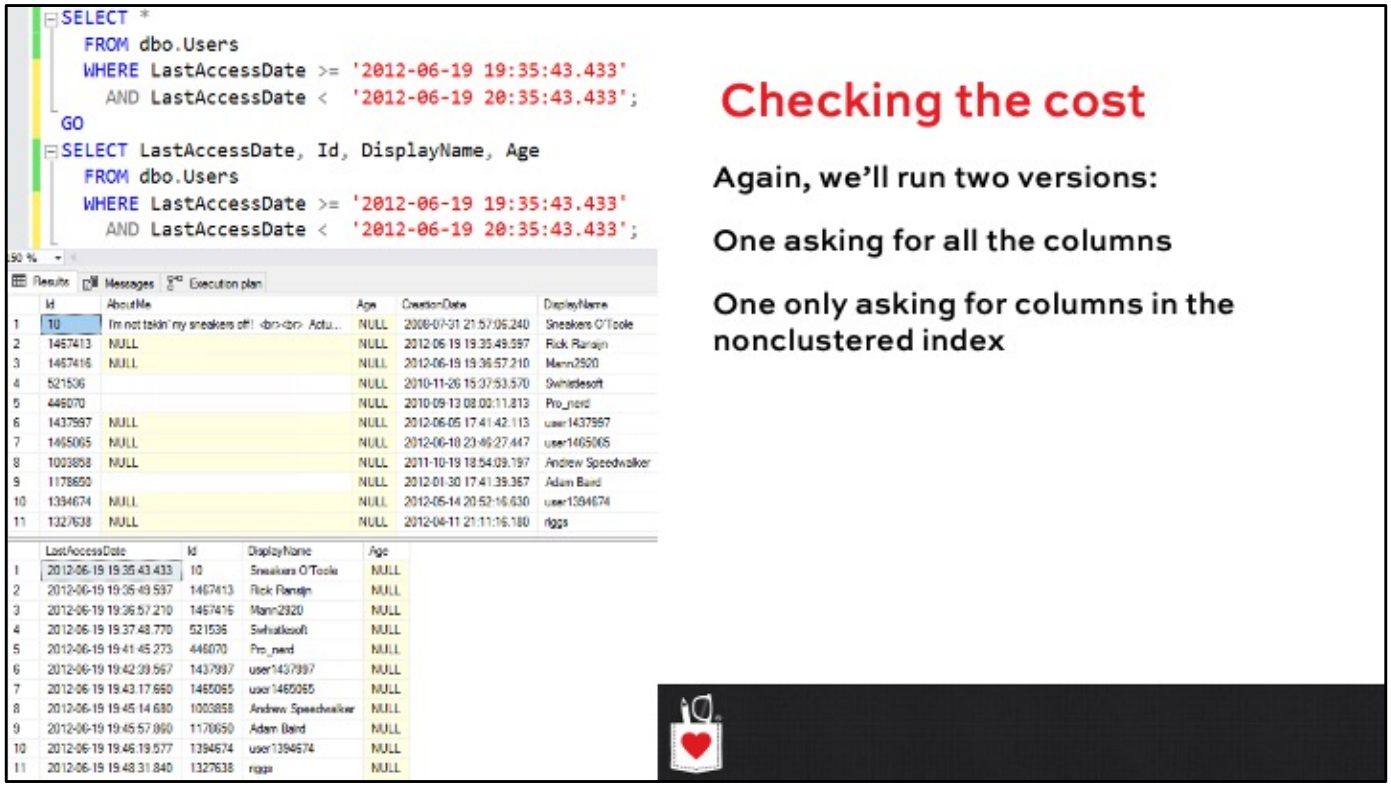
That 1-hour range produces 43 rows:

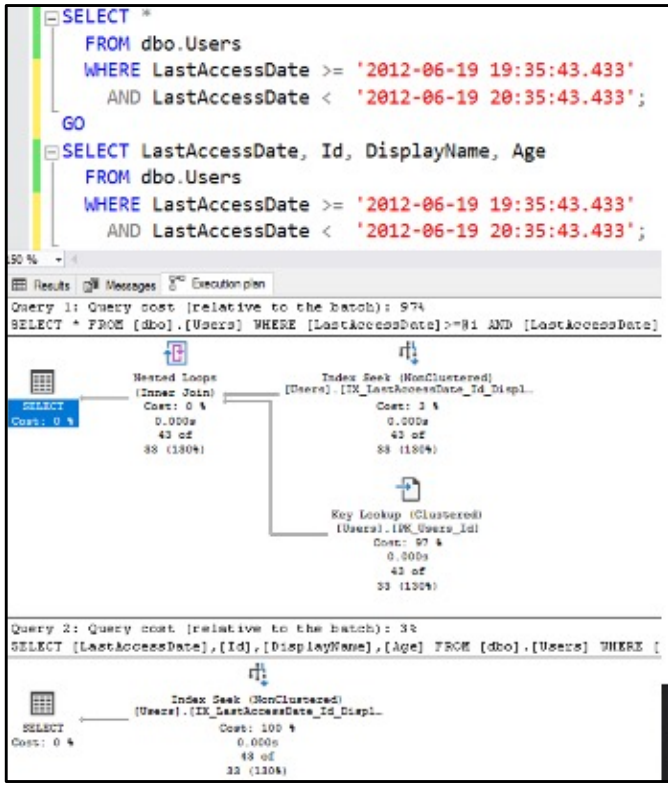
	1549852
	143335
	1578961
	1582404
	1346330
an.com	1346101
	1032689

(local) (14.0 RTM) | SQL2017LAB1\Administra... | StackOverflow | 00:00:00 | 43 rows

2.1 p22







Different plans

The top one needs the key lookup since it's asking for SELECT *...

Now the reads are starting to grow.

```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
GO
SELECT LastAccessDate, Id, DisplayName, Age
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
```

50 %

Results | Messages | Execution plan

```
(43 rows affected)
Table 'Users'. Scan count 1, logical reads 144, physical reac

(1 row affected)

(43 rows affected)
Table 'Users'. Scan count 1, logical reads 4, physical reads
```

```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
GO
SELECT LastAccessDate, Id, DisplayName, Age
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
```

50 %

Results | Messages | Execution plan

```
(43 rows affected)
Table 'Users'. Scan count 1, logical reads 144, physical reac

(1 row affected)

(43 rows affected)
Table 'Users'. Scan count 1, logical reads 4, physical reads
```

144 reads vs 4 reads.

We read an extra 1,120 kilobytes.

Is that a big deal?

Only you can answer that.

2.1 p27

Do they really need the columns?

This is where it becomes a religious war.

The developers need data to show stuff on screens.

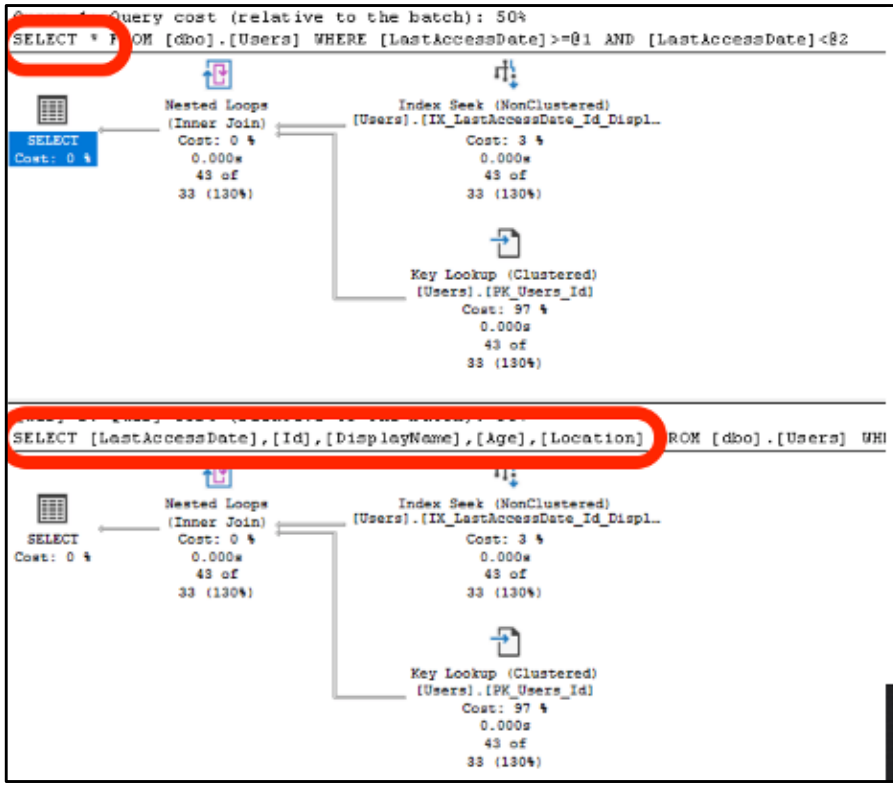
Will it actually help if we make them pick a specific list of columns instead of *?

Especially if they still need columns that aren't on the index?

```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
GO
SELECT LastAccessDate, Id, DisplayName, Age
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
```

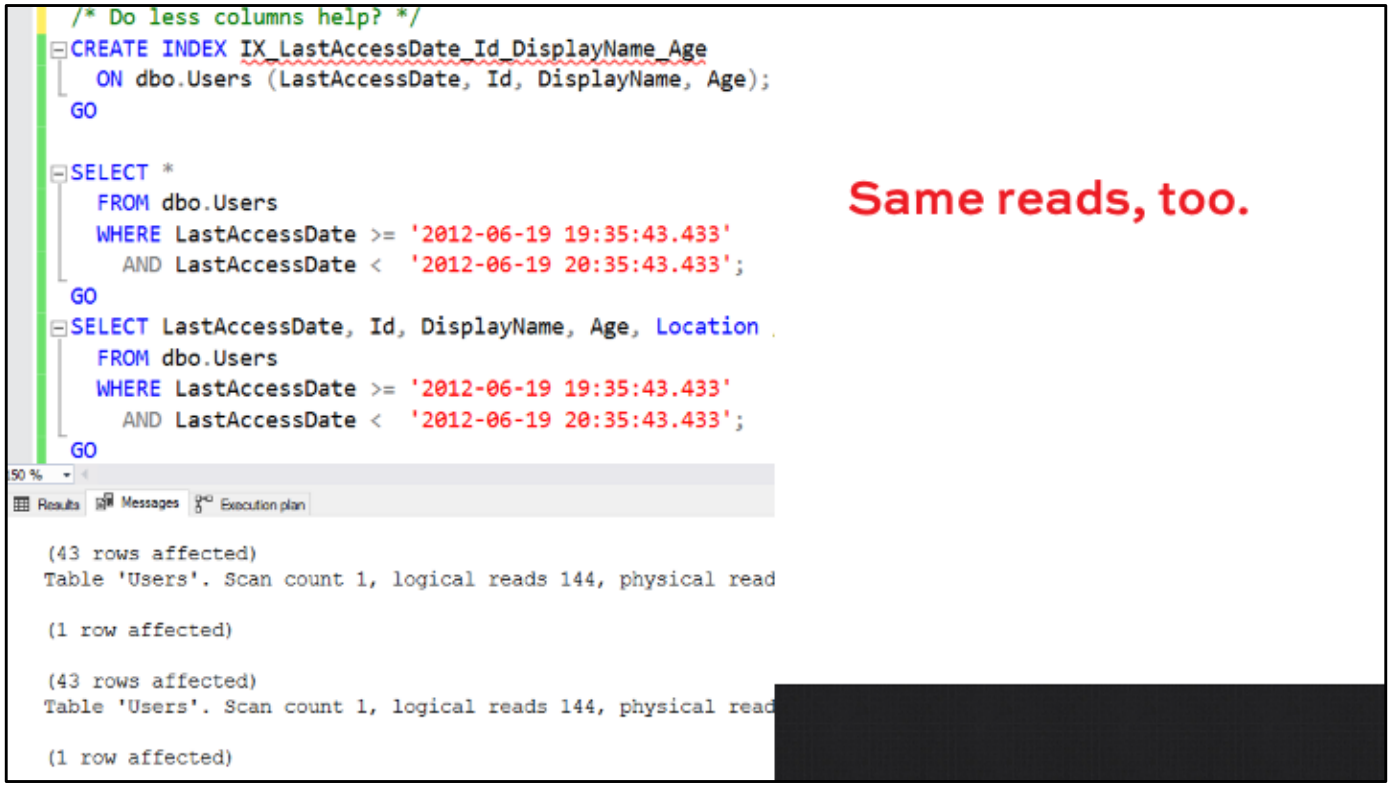
```sql
/* Do less columns help? */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
GO
SELECT LastAccessDate, Id, DisplayName, Age, Location /* Location isn't in the index */
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
    AND LastAccessDate <  '2012-06-19 20:35:43.433';
GO
```

2.1 p29

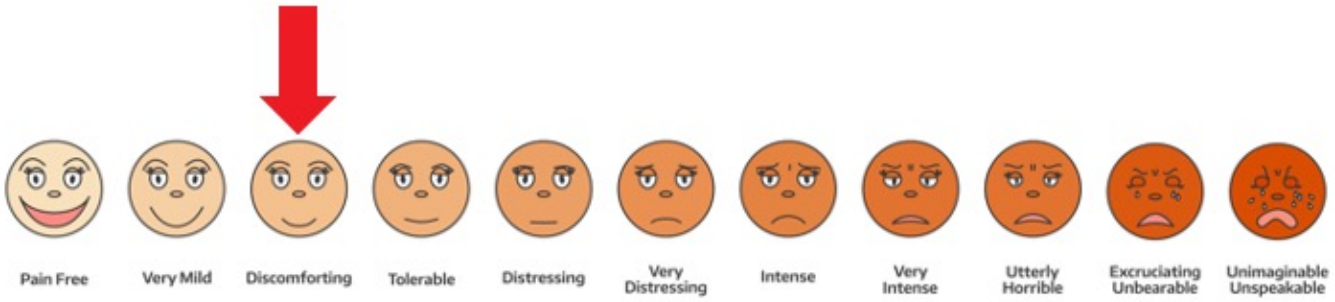


No help here.

If we even ask for 1 extra column (Location) that isn't on the index, we get the key lookup.



A FEW ROWS

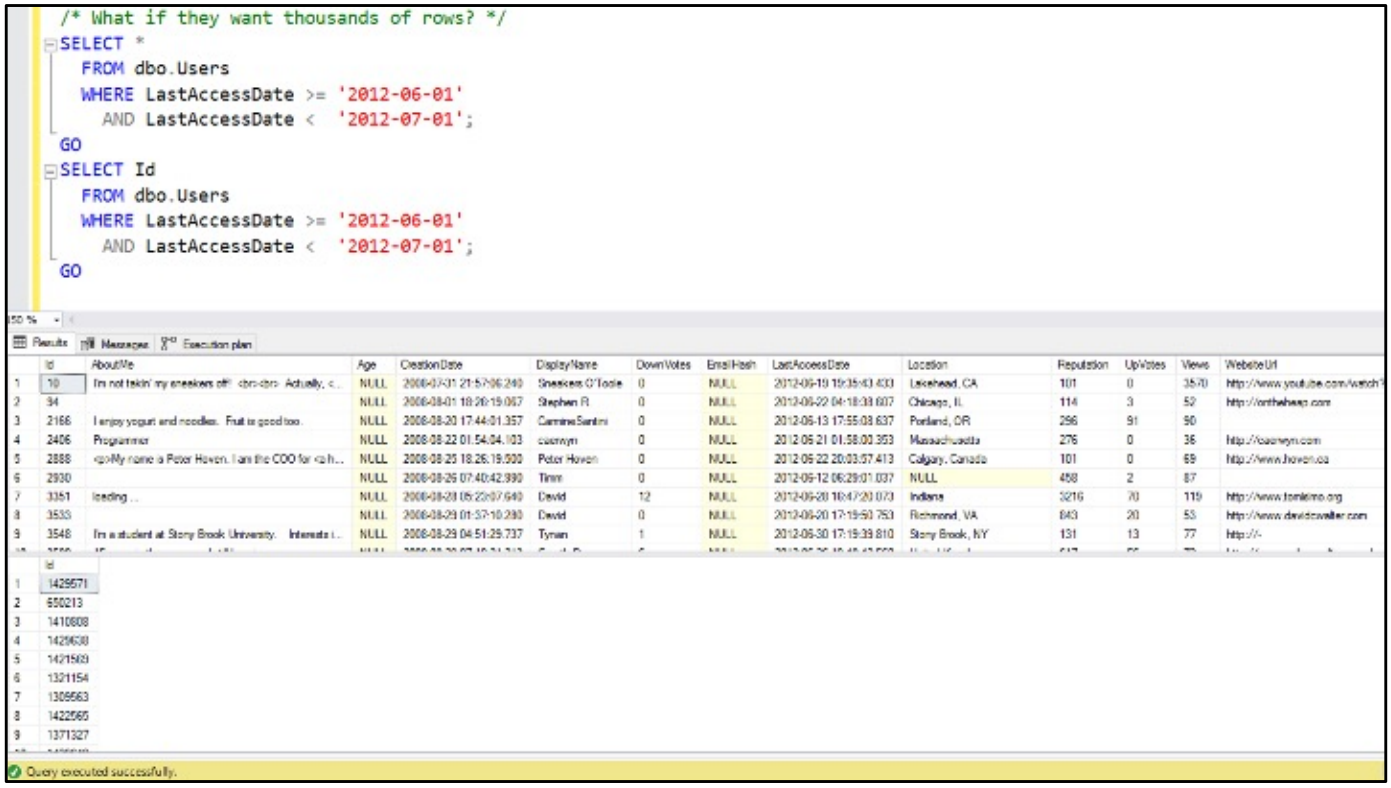


2.1 p32

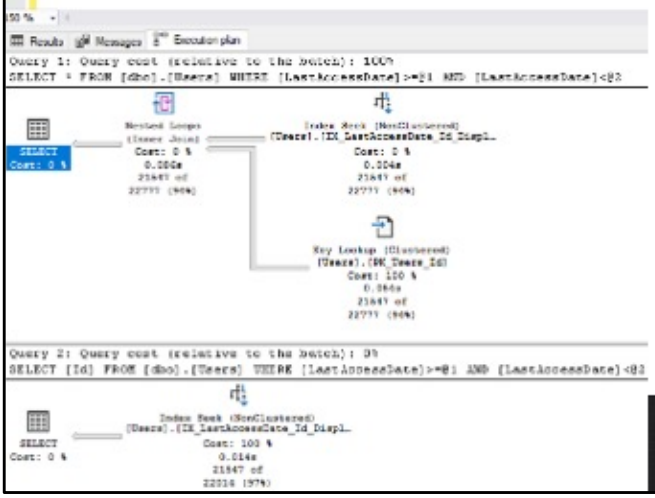
Next up:

GETTING THOUSANDS OF ROWS

2.1 p33



```sql
/* What if they want thousands of rows? */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-01'
    AND LastAccessDate <  '2012-07-01';
GO
SELECT Id
  FROM dbo.Users
  WHERE LastAccessDate >= '2012-06-01'
    AND LastAccessDate <  '2012-07-01';
GO
```

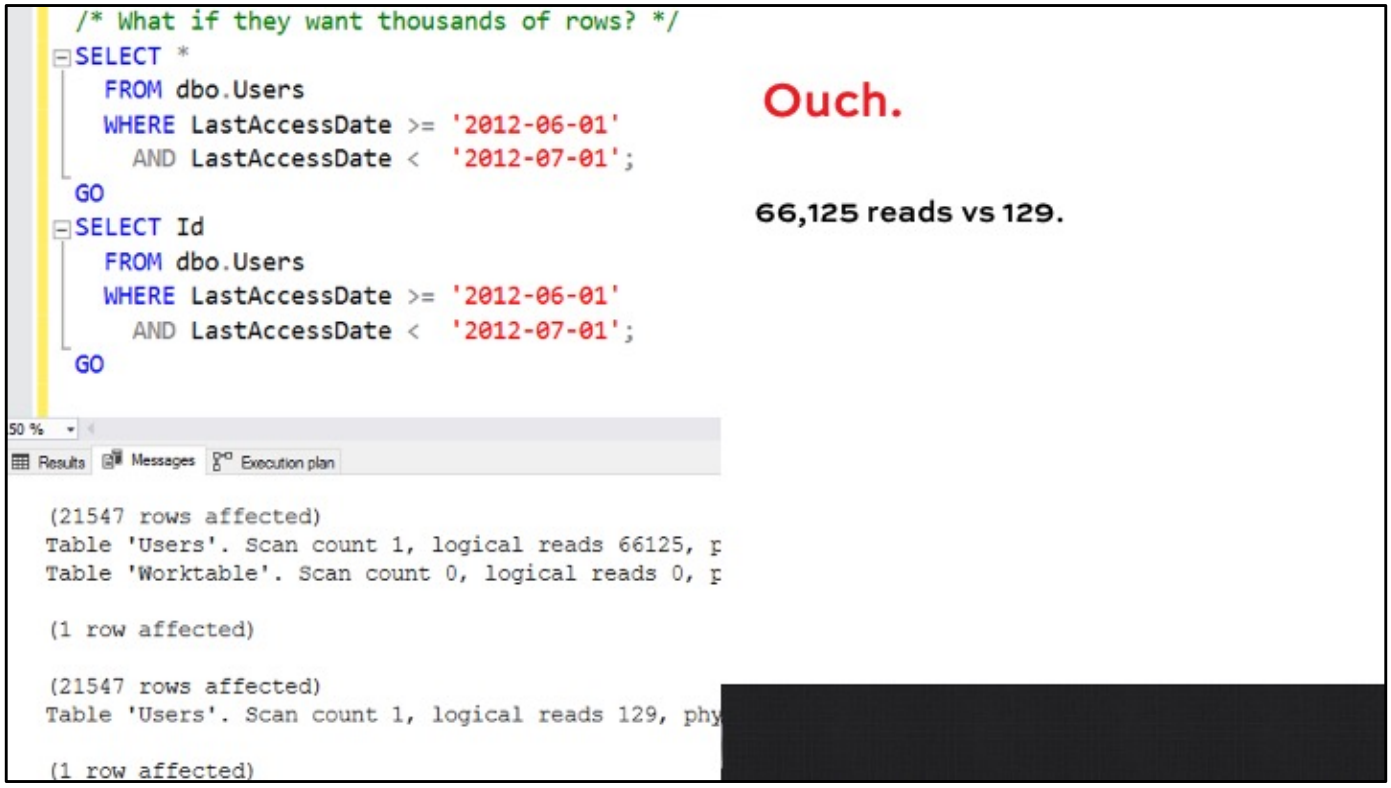


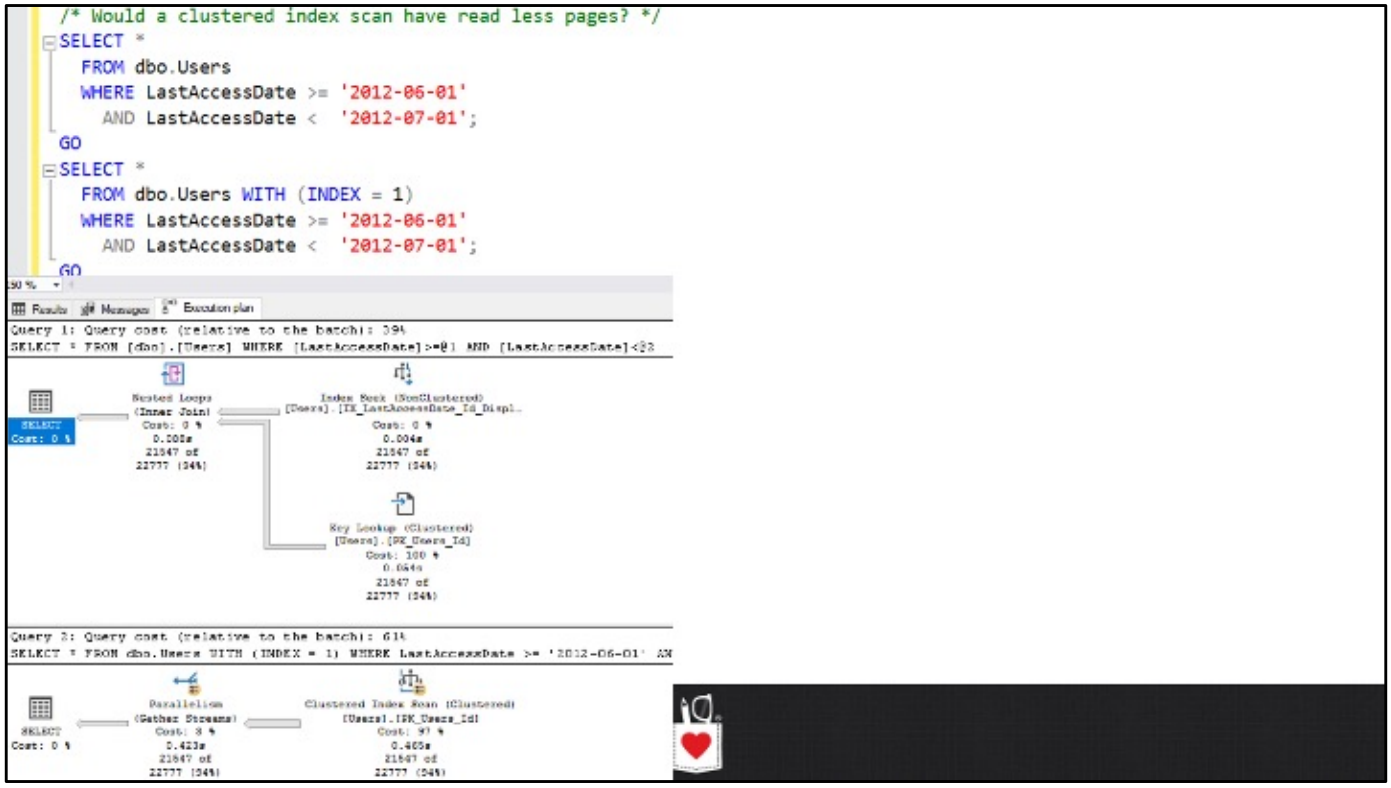
Big data

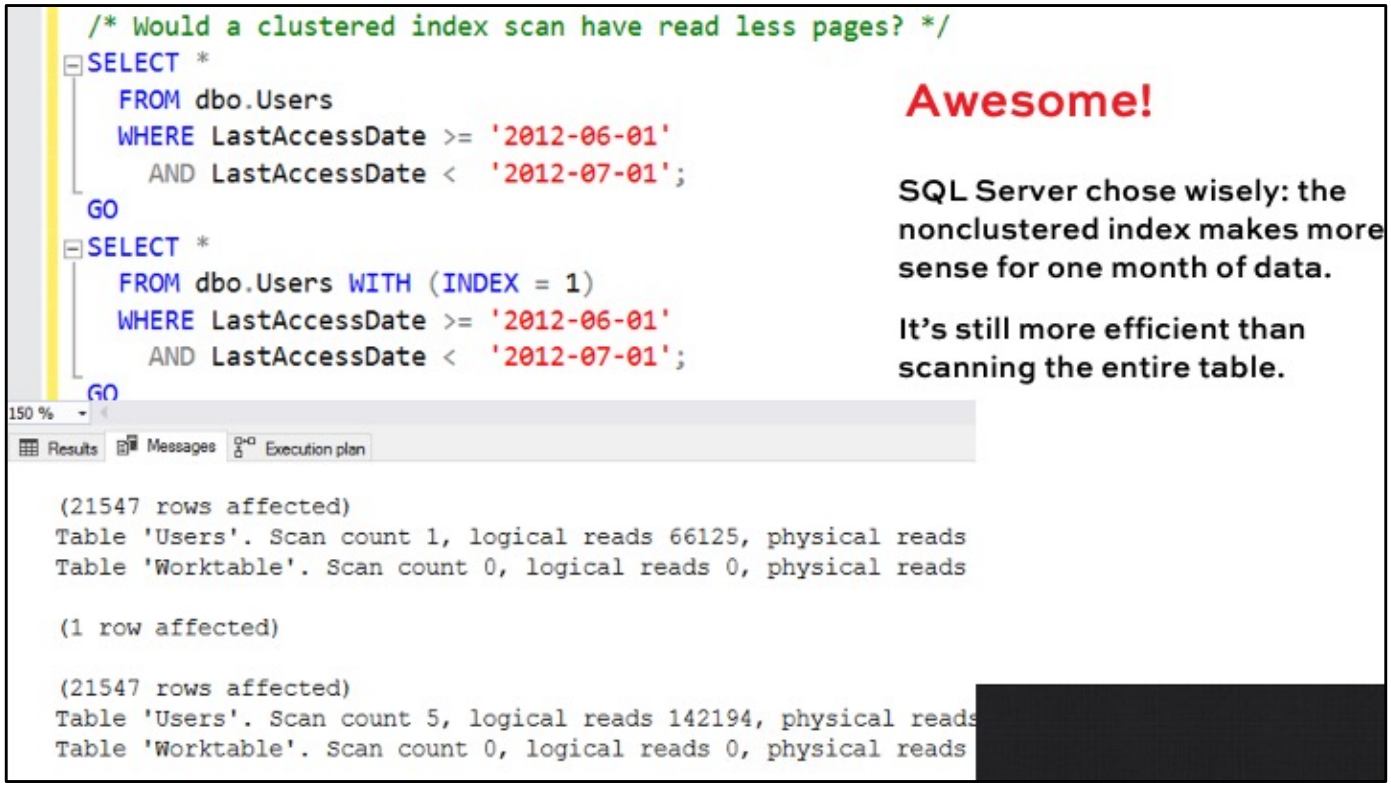
Now we're up over 20,000 rows, and SQL Server is still willing to use the index!

That's kinda awesome.

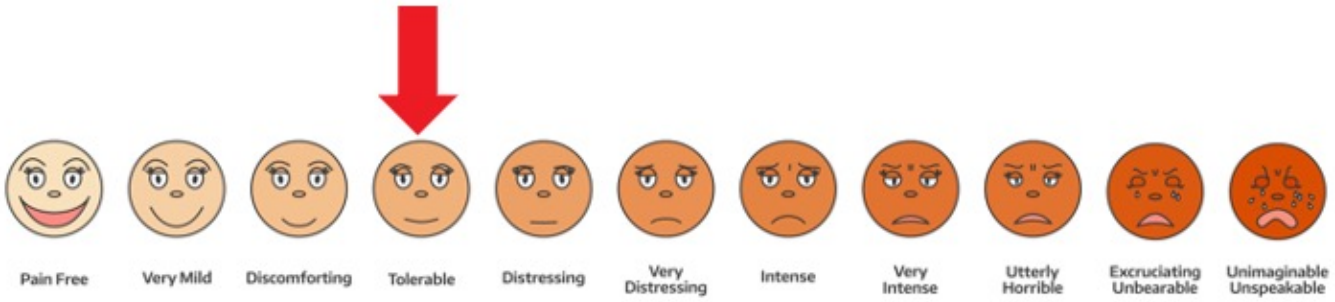
What's the overhead of the key lookup?







THOUSANDS OF ROWS



2.1 p39

Next up:

GETTING VARIABLE NUMBERS OF ROWS

2.1 p40

```sql
/* Let's add parameters and an ORDER BY: */

CREATE OR ALTER PROC dbo.usp_UsersByLastAccessDate
    @StartDate DATETIME, @EndDate DATETIME AS
BEGIN
    SELECT *
      FROM dbo.Users
      WHERE LastAccessDate >= @StartDate
        AND LastAccessDate <  @EndDate
      ORDER BY DisplayName;
END
GO

/* Here's our index: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

/* Will it use the index for a small date range? */
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';
GO
/* Or for a larger date range? */
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
GO
```

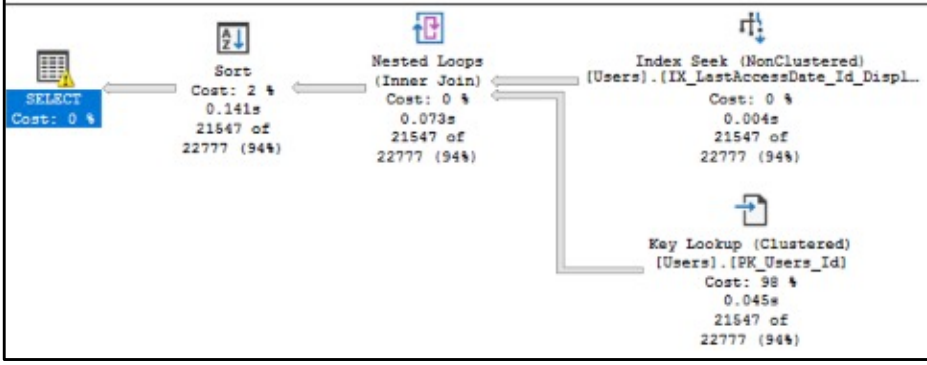
It uses the nonclustered index...

```
/* Will it use the index for a small date range? */
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';
```

150 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE LastAccessDate >= @StartDate AND LastAccessDate < @EndDate ORDER BY DisplayName
```



And it does more reads than there are pages in the table.

```
/* Or for a larger date range? */
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
GO
```

150 %

Results | Messages | Execution plan

```
(75233 rows affected)
Table 'Worktable'. Scan coun
physical reads 0, read-ahead reads 0, lob logi
Table 'Users'. Scan count 1,  logical reads 230849,  physical reads 0, read-ahead reads 0, lob log
```

2.1 p43

If someone happens to...

Restart Windows

Restart the SQL Server service

Fail over the cluster

Free the plan cache

Make server-level config changes like CTFP or MAXDOP

Rebuild indexes

Update statistics

2.1 p44

And we run the “big” date range again...

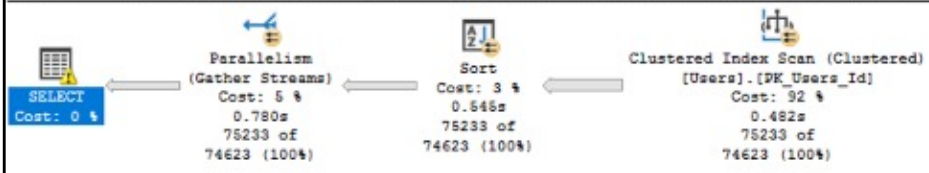
Now we get the plan designed for big data: a table scan.

```
ALTER TABLE dbo.Users REBUILD;
GO
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
GO
```

150 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE LastAccessDate >= @StartDate AND LastAccessDate < @EndDate ORDER BY DisplayName
```



2.1 p45

Which makes sense

Because it's less reads than it was doing previously:
now we're only reading the whole table once.

```
ALTER TABLE dbo.Users REBUILD;
GO
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
GO
```

150 %

Results | Messages | Execution plan

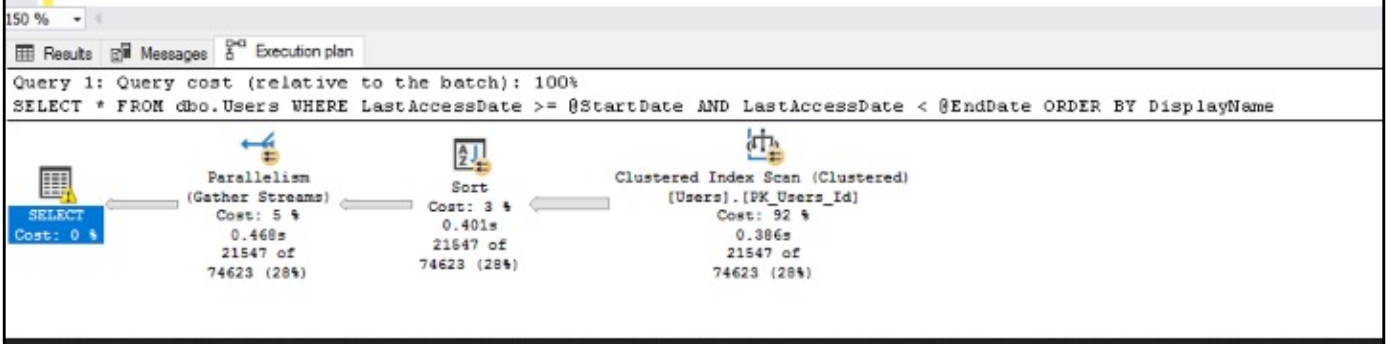
```
(75233 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logic
Table 'Users'. Scan count 5, logical reads 142192, physical reads 0, read-ahead reads 15, lob log
```

2.1 p46

But if we now run the small date range...

We reuse the clustered index scan plan...

```
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';
GO
```



2.1 p47

Which does way more reads than we used to.

Scanning the table is a *fixed* number of reads though: this is sometimes a safer plan if we can't predict how many rows we're going to bring back.

```
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';
GO
```

150 %

Results | Messages | Execution plan

```
(21547 rows affected)
Table 'Worktable'. Scan count [illegible], physical reads 0, read-ahead reads 0, lob logic
Table 'Users'. Scan count 5, logical reads 142192, physical reads 0, read-ahead reads 0, lob logi
```

2.1 p48

Reusable plans are SELECT *'s weak point.

When you have a stored procedure (or any reusable plan), and SQL Server has to build and reuse a plan for varying numbers of rows – especially for wide variances in row counts.

```
CREATE OR ALTER PROC dbo.usp_UsersByLastAccessD
    @StartDate DATETIME, @EndDate DATETIME AS
BEGIN
    SELECT *
      FROM dbo.Users
      WHERE LastAccessDate >= @StartDate
        AND LastAccessDate <  @EndDate
      ORDER BY DisplayName;
END
GO
```

2.1 p49

Parameter sniffing: a classic problem

"Why was my query fast yesterday, but it's slow today?"

"Why is the query slow in the app, but fast in SSMS?"

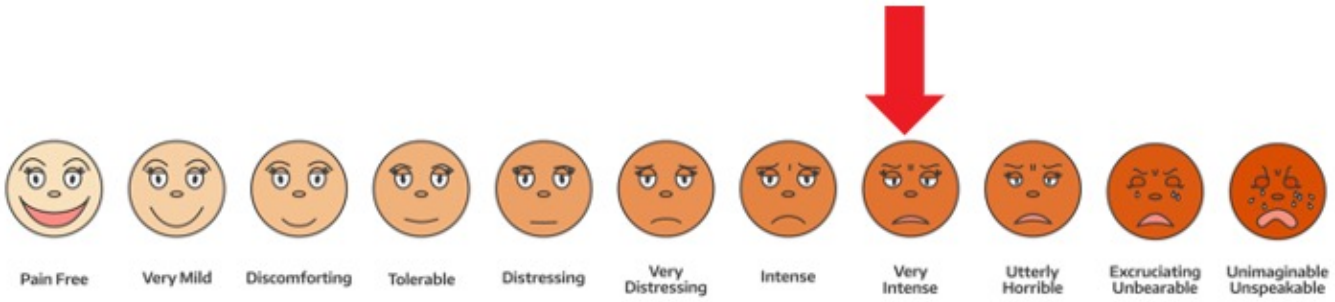
"I swear nobody changed anything!"

One of the most classic causes of this symptom:

- There's no covering index
- Sometimes we bring back a lot of data, sometimes a little
- SQL Server caches a plan based on the first set of parameters
- The plan isn't stable because we keep rebuilding indexes

Learn more: BrentOzar.com/go/sniff

VARIABLE ROWS



2.1 p51

Your tables are like long spreadsheets.

dbo.Users - Clustered Index

Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Cerrito, CA	39	<img src="http://img377.ir
2	126	7/12/09 10:51 PM	Geoff Dalgas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the StackOver
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morganton, NC	31	Developer on the Stack Ove
4	767	7/12/09 10:51 PM	Joel Spolsky	3/30/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflo
535	386	7/15/09 9:33 AM	izb	2/18/10 9:27 PM	Scotland	33	Twittage: http://twitter.co
536	101	7/15/09 9:34 AM	second	3/10/10 9:56 PM	NULL	NULL	NULL
537	120	7/15/09 9:35 AM	staffan	1/25/10 7:10 PM	Sweden	36	I work on the JRockit JVM d
538	90	7/15/09 9:35 AM	cgreeno	1/19/10 10:54 PM	London	NULL	A Canadian living in London
539	167	7/15/09 9:37 AM	Arcturus	3/12/10 9:44 AM	NL	27	I work as a software develo
540	101	7/15/09 9:37 AM	DanSingerman	12/1/09 4:37 PM	NULL	NULL	<p>Sufficiently advanced st
541	647	7/15/09 9:37 AM	Alexis Hirst	4/1/10 6:37 AM	England	21	NULL
542	5921	7/15/09 9:38 AM	hypenslug	4/1/10 10:49 PM	NULL	NULL	<img src="http://www.hype
543	490	7/15/09 9:38 AM	DavidWhitney	12/12/09 5:12 PM	London, UK	25	Primarily C# .NET developer
544	101	7/15/09 9:39 AM	Ben Laan	3/28/10 11:12 PM	Adelaide, Australia	34	C#, Delphi, T-SQL : Archite
545	985	7/15/09 9:40 AM	redacted	4/1/10 3:34 PM	NaN	23	NaN
546	101	7/15/09 9:41 AM	hitec	8/26/09 9:59 AM	Bombay	27	System.StackOverflowExce
547	404	7/15/09 9:41 AM	Ant	4/1/10 8:24 AM	England	27	C# developer since 2005.
548	501	7/15/09 9:41 AM	Jeremy French	1/28/10 5:18 PM	UK	33	Seasoned web developer. W
549	173	7/15/09 9:42 AM	Jeshua	2/26/10 5:53 PM	Orlando, FL	28	Software Engineer. Have wif
550	61	7/15/09 9:42 AM	hopsasa	3/16/10 4:42 PM	NULL	NULL	NULL
551	341	7/15/09 9:43 AM	Tom	4/1/10 10:39 AM	Worcester, UK	NULL	NULL
552	1	7/15/09 9:44 AM	Exodus	11/29/09 7:20 AM	Maracaibo, Venezuela	25	NULL
553	101	7/15/09 9:44 AM	maff	2/7/10 10:28 AM	Sweden	NULL	NULL
554	1158	7/15/09 9:44 AM	becomingGuru	4/1/10 8:43 PM	Bangalore / Hyderabad	25	curious learner!
555	1016	7/15/09 9:45 AM	MiffTheFox	4/1/10 6:11 PM	Huntsville, AL	NULL	<p><a href="http://forums
556	101	7/15/09 9:45 AM	Russ Cam	11/8/09 6:10 PM	UK	NULL	Interested in C#, ASP.NET, :
557	1	7/15/09 9:45 AM	Beto	2/13/10 11:01 AM	NULL	NULL	NULL
558	953	7/15/09 9:46 AM	marco.ragagna	4/1/10 7:41 AM	IT	29	<h2>Software Developer</
559	593	7/15/09 9:46 AM	moobaa	2/18/10 7:14 AM	AU	39	Hey, at least I know 6502!
560	83	7/15/09 9:46 AM	rjstelling	3/4/10 6:47 PM	United Kingdom	30	<h3>Tech geek and hopele
561	716	7/15/09 9:47 AM	balpha	4/1/10 4:00 PM	NULL	NULL	NULL
562	43	7/15/09 9:47 AM	Mike McClelland	3/16/10 8:45 AM	London	NULL	NULL
563	101	7/15/09 9:48 AM	arturn	10/29/09 2:11 PM	Bremen, Germany	24	NULL
564	960	7/15/09 9:49 AM	Stiff	4/1/10 6:59 AM	UK	30	<p>I'm a Software Enginee
565	121	7/15/09 9:49 AM	Franci Penov	3/5/10 12:13 AM	Kirkland, WA	37	<p>Father, SDE@Microsoft

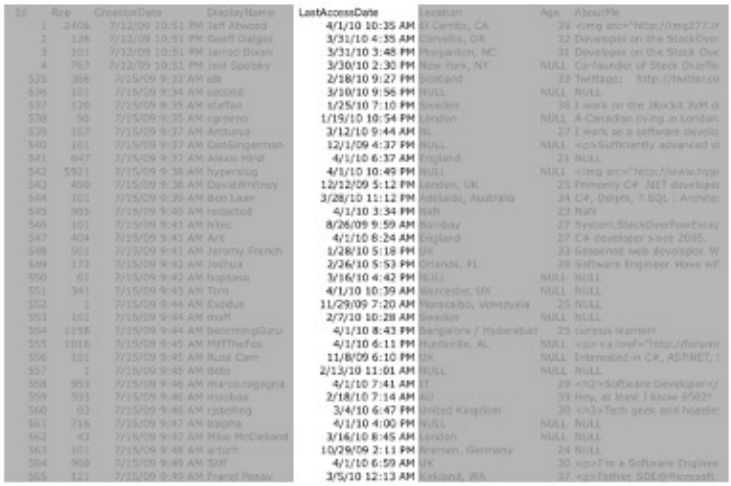
Not just this page,
but many pages in a row.

A ROW, GET IT

2.1 p52

Ask for few columns, and you get index seeks.

dbo.Users - Clustered Index



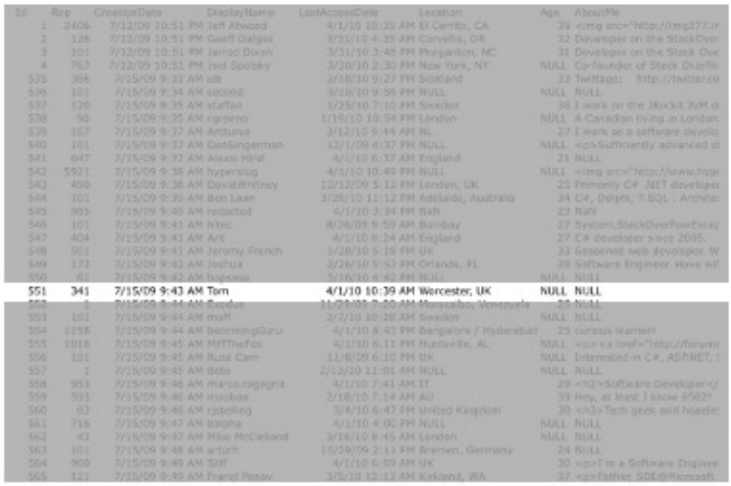
This query is likely to use an index even if it returns all of the rows:

```
SELECT LastAccessDate
FROM dbo.Users;
```

2.1 p53

If you only want a few rows, you get seeks too.

dbo.Users - Clustered Index



Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Cerrito, CA	39	<img src="http://img277.ir
2	126	7/12/09 10:51 PM	Geoff Dalgas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the StackOver
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morganton, NC	31	Developer on the Stack Ove
4	767	7/12/09 10:51 PM	Joel Spolsky	3/30/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflo
535	386	7/15/09 9:33 AM	idb	2/18/10 9:27 PM	Scotland	33	Twittago: http://twitter.co
536	101	7/15/09 9:34 AM	second	3/30/10 9:56 PM	NULL	NULL	NULL
537	120	7/15/09 9:35 AM	staffan	1/25/10 7:10 PM	Sweden	36	I work on the JRockit JVM a
538	90	7/15/09 9:35 AM	cgreeno	1/19/10 10:54 PM	London	NULL	A Canadian living in London
539	167	7/15/09 9:37 AM	Arcturus	3/12/10 9:44 AM	NL	27	I work as a software develo
540	101	7/15/09 9:37 AM	DanSingerman	12/1/09 4:37 PM	NULL	NULL	<p>Sufficiently advanced st
541	647	7/15/09 9:37 AM	Alexis Hirst	4/1/10 6:17 AM	England	21	NULL
542	5921	7/15/09 9:38 AM	hypersiug	4/1/10 10:48 PM	NULL	NULL	<img src="http://www.hyp
543	409	7/15/09 9:38 AM	DavidWhitney	12/12/09 5:12 PM	London, UK	25	Primarily C# .NET developer
544	101	7/15/09 9:39 AM	Ben Laan	3/28/10 11:12 PM	Adelaide, Australia	34	C#, Delphi, T-SQL : Architar
545	985	7/15/09 9:40 AM	redacted	4/1/10 3:34 PM	NaN	23	NaN
546	101	7/15/09 9:41 AM	hitec	8/26/09 9:59 AM	Bombay	27	System.StackOverflowExce
547	404	7/15/09 9:41 AM	Ant	4/1/10 8:24 AM	England	27	C# developer since 2005.
548	501	7/15/09 9:41 AM	Jeremy French	1/28/10 5:18 PM	UK	33	Seasoned web developer. W
549	173	7/15/09 9:42 AM	Jezhos	3/28/10 5:53 PM	Orlando, FL	28	Software Engineer. Have wif
550	61	7/15/09 9:42 AM	hopsasa	3/16/10 6:42 PM	NULL	NULL	NULL
551	341	7/15/09 9:43 AM	Tom	4/1/10 10:39 AM	Worcester, UK	NULL	NULL
552	1	7/15/09 9:44 AM	Paulo	11/20/09 7:39 AM	Maracaibo, Venezuela	35	NULL
553	101	7/15/09 9:44 AM	msff	2/2/10 10:28 AM	Sweden	NULL	NULL
554	1158	7/15/09 9:44 AM	becomingGuru	4/1/10 8:43 PM	Bangalore / Hyderabad	25	curious learner!
555	1016	7/15/09 9:45 AM	MPTheFox	4/1/10 6:11 PM	Huntsville, AL	NULL	<p><a href="http://forums
556	101	7/15/09 9:45 AM	Russ Cam	11/8/09 6:10 PM	UK	NULL	Interested in C#, ASP.NET, T
557	1	7/15/09 9:45 AM	Beto	2/13/10 11:01 AM	NULL	NULL	NULL
558	953	7/15/09 9:46 AM	marco.ragogna	4/1/10 7:41 AM	IT	29	<h2>Software Developer</
559	593	7/15/09 9:46 AM	inoobaa	2/18/10 7:14 AM	AU	39	Hey, at least I know 6502!
560	63	7/15/09 9:46 AM	rjstelling	3/4/10 6:47 PM	United Kingdom	30	<h3>Tech geek and hopeles
561	716	7/15/09 9:47 AM	balpha	4/1/10 4:00 PM	NULL	NULL	NULL
562	43	7/15/09 9:47 AM	Mike McClelland	3/16/10 8:45 AM	London	NULL	NULL
563	101	7/15/09 9:48 AM	artum	10/29/09 2:13 PM	Bremen, Germany	24	NULL
564	900	7/15/09 9:49 AM	Stiff	4/1/10 6:59 AM	UK	30	<p>I'm a Software Enginee
565	121	7/15/09 9:49 AM	Pranel Penov	3/5/10 12:13 AM	Kirkland, WA	27	<p>Father, SDE@Microsoft

If your filter is really focused, you're also likely to get seeks + key lookups even if you SELECT *.

```
SELECT *
FROM dbo.Users
WHERE LastAccessDate =
'4/1/10 10:49PM';
```

2.1 p54

But the more rows & columns you want...

dbo.Users - Clustered Index

Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Cerrito, CA	39	<img src="http://img277.ir
2	126	7/12/09 10:51 PM	Geoff Dalgas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the StackOver
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morganton, NC	31	Developer on the Stack Ove
4	767	7/12/09 10:51 PM	Joel Spolsky	3/30/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflo
535	386	7/15/09 9:33 AM	idb	2/18/10 9:27 PM	Scotland	33	Twittago: http://twitter.co
536	101	7/15/09 9:34 AM	second	3/30/10 9:56 PM	NULL	NULL	NULL
537	120	7/15/09 9:35 AM	staffan	1/25/10 7:10 PM	Sweden	36	I work on the JRockit JVM a
538	90	7/15/09 9:35 AM	cgreeno	1/19/10 10:54 PM	London	NULL	A Canadian living in London
539	167	7/15/09 9:37 AM	Arcturus	3/12/10 9:44 AM	NL	27	I work as a software develo
540	101	7/15/09 9:37 AM	DanSingerman	12/1/09 4:37 PM	NULL	NULL	<p>Sufficiently advanced st
541	647	7/15/09 9:37 AM	Alexis Hirst	4/1/10 6:17 AM	England	21	NULL
542	5921	7/15/09 9:38 AM	hypervisug	4/1/10 10:49 PM	NULL	NULL	<img src="http://www.hyp
543	490	7/15/09 9:38 AM	DavidWhitney	12/12/09 5:12 PM	London, UK	25	Primarily C# .NET developer
544	101	7/15/09 9:39 AM	Ben Laan	3/28/10 11:12 PM	Adelaide, Australia	34	C#, Delphi, T-SQL : Archite
545	985	7/15/09 9:40 AM	redacted	4/1/10 3:34 PM	NaN	23	NaN
546	101	7/15/09 9:41 AM	hitec	8/26/09 9:59 AM	Bombay	27	System.StackOverflowExce
547	404	7/15/09 9:41 AM	Ant	4/1/10 8:24 AM	England	27	C# developer since 2005.
548	501	7/15/09 9:41 AM	Jeremy French	1/28/10 5:18 PM	UK	33	Seasoned web developer W
549	173	7/15/09 9:42 AM	Jashua	2/26/10 5:53 PM	Orlando, FL	28	Software Engineer. Have wif
550	61	7/15/09 9:42 AM	hopsasa	3/16/10 4:42 PM	NULL	NULL	NULL
551	341	7/15/09 9:43 AM	Tom	4/1/10 10:39 AM	Worcester, UK	NULL	NULL
552	1	7/15/09 9:44 AM	Exodus	11/29/09 7:20 AM	Maracaibo, Venezuela	25	NULL
553	101	7/15/09 9:44 AM	maff	2/7/10 10:28 AM	Sweden	NULL	NULL
554	1158	7/15/09 9:44 AM	becomingGuru	4/1/10 8:43 PM	Bangalore / Hyderabad	25	curious learner!
555	1016	7/15/09 9:45 AM	MiffTheFox	4/1/10 6:11 PM	Huntsville, AL	NULL	<p><a href="http://forums
556	101	7/15/09 9:45 AM	Russ Cam	11/8/09 6:10 PM	UK	NULL	Interested in C#, ASP.NET, :
557	1	7/15/09 9:45 AM	Beto	2/13/10 11:01 AM	NULL	NULL	NULL
558	953	7/15/09 9:46 AM	marco.ragagna	4/1/10 7:41 AM	IT	29	<h2>Software Developer</
559	993	7/15/09 9:46 AM	mxcbas	2/18/10 7:14 AM	AU	39	Hey, at least I know 65021
560	63	7/15/09 9:46 AM	rjstelling	3/4/10 6:42 PM	United Kingdom	30	<h1>Tech geek and hopeles
561	716	7/15/09 9:47 AM	balpha	4/1/10 4:00 PM	NULL	NULL	NULL
562	43	7/15/09 9:47 AM	Mike McClelland	3/16/10 8:45 AM	London	NULL	NULL
563	101	7/15/09 9:48 AM	artum	10/29/09 2:13 PM	Bremen, Germany	24	NULL
564	900	7/15/09 9:49 AM	Stiff	4/1/10 6:59 AM	UK	30	<p>I'm a Software Enginee
565	121	7/15/09 9:49 AM	Franci Penov	3/5/10 12:13 AM	Kirkland, WA	37	<p>Father, SDE@Microsoft

The more likely you are to get a clustered index scan.

```
SELECT *
FROM dbo.Users
WHERE LastAccessDate
  IN ReallyBigRange;
```

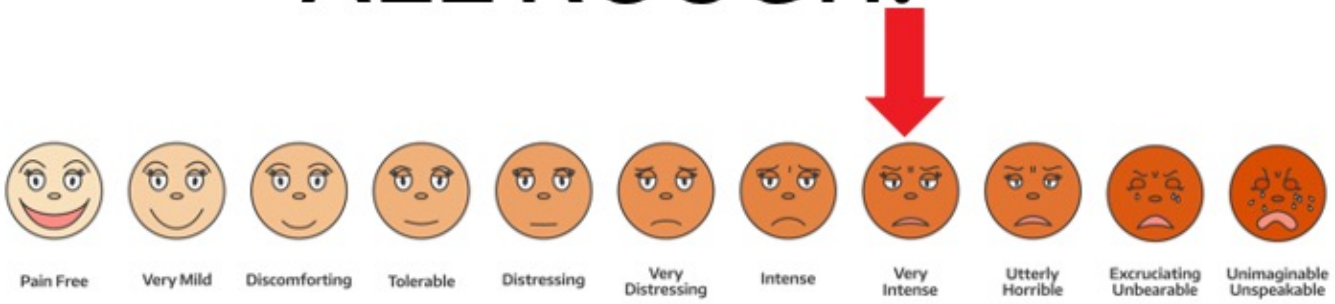
2.1 p55

We don't have great options.

1. Make a really wide nonclustered index
   (include all kinds of columns to satisfy the SELECT *)
2. Change your clustered index to match your searches
   (data warehouse technique)
3. Create a narrow nonclustered index and pray
   (but it usually gets ignored when you pass the tipping point)
4. Use recompile hints, branching procs, or other diabolical tricks
   designed to get you multiple plans for different data sizes

2.1 p56

THESE ARE ALL ROUGH.



2.1 p57

I'll give you 3 ways to get pain relief.

Neither of these should be your first go-to.

Start with these first:

- Asking if the query really needs all those columns
- Designing indexes to cover as many scenarios as practical
- Using common parameter sniffing techniques

Because the tricks I'm about to show you aren't pain-free either.

2.1 p58

Advanced query tricks can help in 3 cases:

1. Users who insist on a large number of rows, like beyond the tipping point, but are flexible on how many can be displayed on each page

2. When users want SELECT *, but SQL Server overestimates how many rows will come back, thinking it's over the tipping point

3. Going nuclear to stop SELECT * for good (or evil)

2.1 p59

1. Row tuning: using CTE pagination

2.1 p60

Your goal: avoid the tipping point.

dbo.Users - Clustered Index

Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Cerrito, CA	39	<img src="http://img277.ir
2	126	7/12/09 10:51 PM	Geoff Dalgas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the StackOver
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morganton, NC	31	Developer on the Stack Ove
4	767	7/12/09 10:51 PM	Joel Spolsky	3/30/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflo
535	386	7/15/09 9:33 AM	idb	2/18/10 9:27 PM	Scotland	33	Twittago: http://twitter.co
536	101	7/15/09 9:34 AM	second	3/30/10 9:56 PM	NULL	NULL	NULL
537	120	7/15/09 9:35 AM	staffan	1/25/10 7:10 PM	Sweden	36	I work on the JRockit JVM a
538	90	7/15/09 9:35 AM	cgreeno	1/19/10 10:54 PM	London	NULL	A Canadian living in London
539	167	7/15/09 9:37 AM	Arcturus	3/12/10 9:44 AM	NL	27	I work as a software develo
540	101	7/15/09 9:37 AM	DanSingerman	12/1/09 4:37 PM	NULL	NULL	<p>Sufficiently advanced st
541	647	7/15/09 9:37 AM	Alexis Hirst	4/1/10 6:17 AM	England	21	NULL
542	5921	7/15/09 9:38 AM	hypervisug	4/1/10 10:49 PM	NULL	NULL	<img src="http://www.hype
543	490	7/15/09 9:38 AM	DavidWhitney	12/12/09 5:12 PM	London, UK	25	Primarily C# .NET developer
544	101	7/15/09 9:39 AM	Ben Laan	3/28/10 11:12 PM	Adelaide, Australia	34	C#, Delphi, T-SQL : Archite
545	985	7/15/09 9:40 AM	redacted	4/1/10 3:34 PM	NaN	23	NaN
546	101	7/15/09 9:41 AM	hitec	8/26/09 9:59 AM	Bombay	27	System.StackOverflowExce
547	404	7/15/09 9:41 AM	Ant	4/1/10 8:24 AM	England	27	C# developer since 2005.
548	501	7/15/09 9:41 AM	Jeremy French	1/28/10 5:18 PM	UK	33	Seasoned web developer W
549	173	7/15/09 9:42 AM	Jashua	2/26/10 5:53 PM	Orlando, FL	28	Software Engineer. Have wif
550	61	7/15/09 9:42 AM	hopsasa	3/16/10 4:42 PM	NULL	NULL	NULL
551	341	7/15/09 9:43 AM	Tom	4/1/10 10:39 AM	Worcester, UK	NULL	NULL
552	1	7/15/09 9:44 AM	Exodus	11/29/09 7:20 AM	Maracaibo, Venezuela	25	NULL
553	101	7/15/09 9:44 AM	maff	2/7/10 10:28 AM	Sweden	NULL	NULL
554	1158	7/15/09 9:44 AM	becomingGuru	4/1/10 8:43 PM	Bangalore / Hyderabad	25	curious learner!
555	1016	7/15/09 9:45 AM	MiffTheFox	4/1/10 6:11 PM	Huntsville, AL	NULL	<p><a href="http://forums
556	101	7/15/09 9:45 AM	Russ Cam	11/8/09 6:10 PM	UK	NULL	Interested in C#, ASP.NET, :
557	1	7/15/09 9:45 AM	Beto	2/13/10 11:01 AM	NULL	NULL	NULL
558	953	7/15/09 9:46 AM	marco.ragagna	4/1/10 7:41 AM	IT	29	<h2>Software Developer</
559	993	7/15/09 9:46 AM	mxcbas	2/18/10 7:14 AM	AU	39	Hey, at least I know 65021
560	63	7/15/09 9:46 AM	rjstelling	3/4/10 6:42 PM	United Kingdom	30	<h1>Tech geek and hopeles
561	716	7/15/09 9:47 AM	balpha	4/1/10 4:00 PM	NULL	NULL	NULL
562	43	7/15/09 9:47 AM	Mike McClelland	3/16/10 8:45 AM	London	NULL	NULL
563	101	7/15/09 9:48 AM	artum	10/29/09 2:13 PM	Bremen, Germany	24	NULL
564	900	7/15/09 9:49 AM	Stiff	4/1/10 6:59 AM	UK	30	<p>I'm a Software Enginee
565	121	7/15/09 9:49 AM	Pranol Pesov	3/5/10 12:13 AM	Kirkland, WA	37	<p>Father, SDE@Microsoft

The more rows in your result set, the higher chance that you're going to end up with scans.

2.1 p61

Ask the developers, "Can we break this up into pages?"

Use examples from:

- Your competitors
- Your users' favorite sites
- Online stores
- The bottom of search result pages

2.1 p62

```sql
/*
Fix #1: Pagination, for when they'll accept less rows

Say we've got this query:
*/
CREATE OR ALTER PROC dbo.usp_UsersByLastAccessDate
    @StartDate DATETIME,
    @EndDate DATETIME AS
BEGIN
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate BETWEEN @StartDate AND @EndDate
  ORDER BY DisplayName;
END
GO

/* And this index from How to Think Like the Engine: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO
```

Here's our starting query and the index

```
CREATE OR ALTER PROC dbo.usp_UsersByLastAccessDate
    @StartDate DATETIME, @EndDate DATETIME AS
BEGIN
    SELECT *
      FROM dbo.Users
      WHERE LastAccessDate >= @StartDate
        AND LastAccessDate <  @EndDate
      ORDER BY DisplayName;
END
GO

/* Here's our index: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
  ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO
```

2.1 p64

```sql
CREATE OR ALTER PROC dbo.usp_UsersByLastAccessDate
    @StartDate DATETIME,
    @EndDate    DATETIME,
    @PageNumber INT = 1,
    @PageSize   INT = 10000 AS
BEGIN
WITH RowsIWant AS (SELECT Id, uR.DisplayName /* Include the sort columns */
                   FROM dbo.Users uR
                   WHERE uR.LastAccessDate >= @StartDate
                     AND uR.LastAccessDate < @EndDate
                   ORDER BY uR.DisplayName
                   OFFSET((@PageNumber - 1) * @PageSize) ROW
                   FETCH NEXT (@PageSize) ROWS ONLY )
SELECT u.*
  FROM RowsIWant r
  INNER JOIN dbo.Users u ON r.Id = u.Id
  ORDER BY r.DisplayName                    /* Use the CTE's sort columns */;
END
GO
```

Phase 1:

do our filtering with the index,
but only return the bare minimum of columns

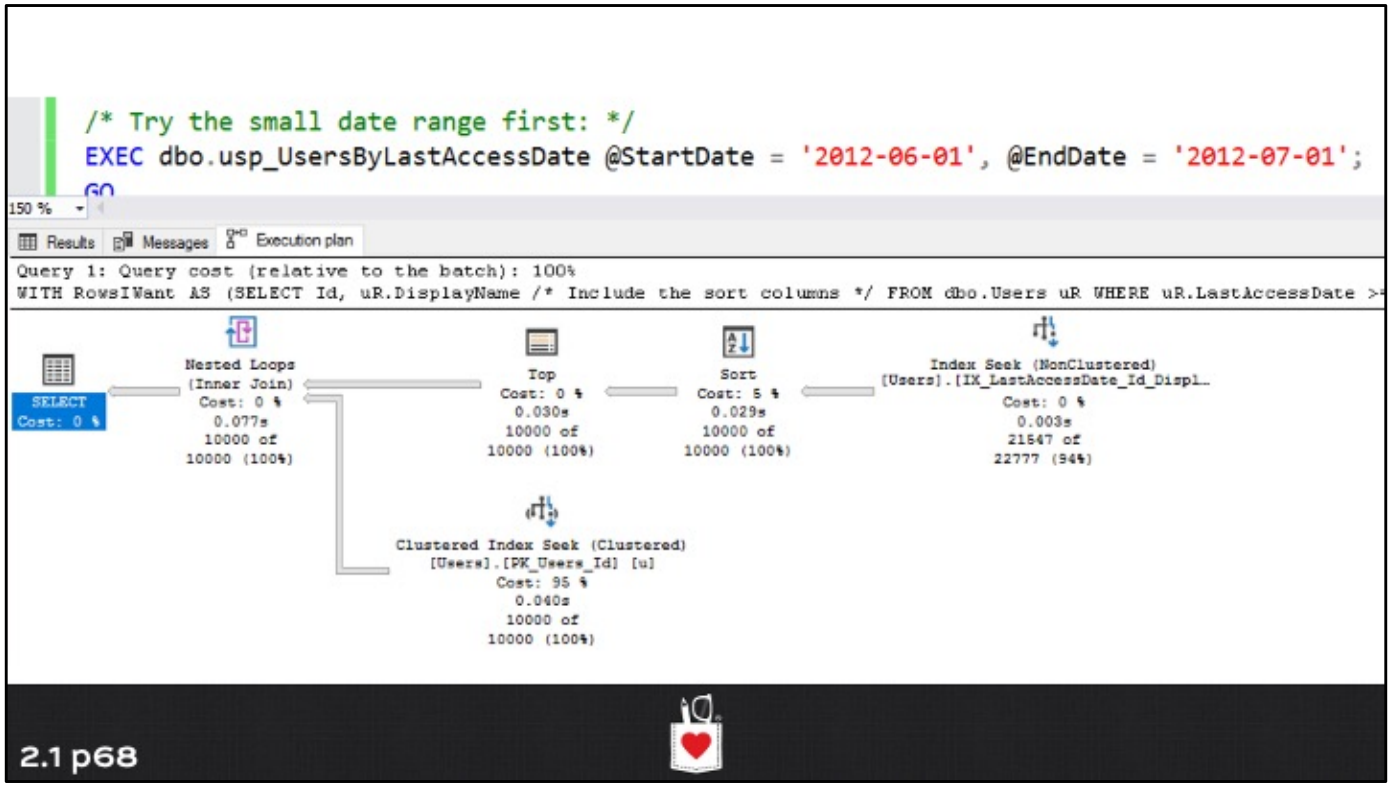
```
WITH RowsIWant AS (SELECT Id, uR.DisplayName /* Include the sort columns */
                   FROM dbo.Users uR
                   WHERE uR.LastAccessDate >= @StartDate
                     AND uR.LastAccessDate < @EndDate
                   ORDER BY uR.DisplayName
                   OFFSET((@PageNumber - 1) * @PageSize) ROW
                   FETCH NEXT (@PageSize) ROWS ONLY )
```

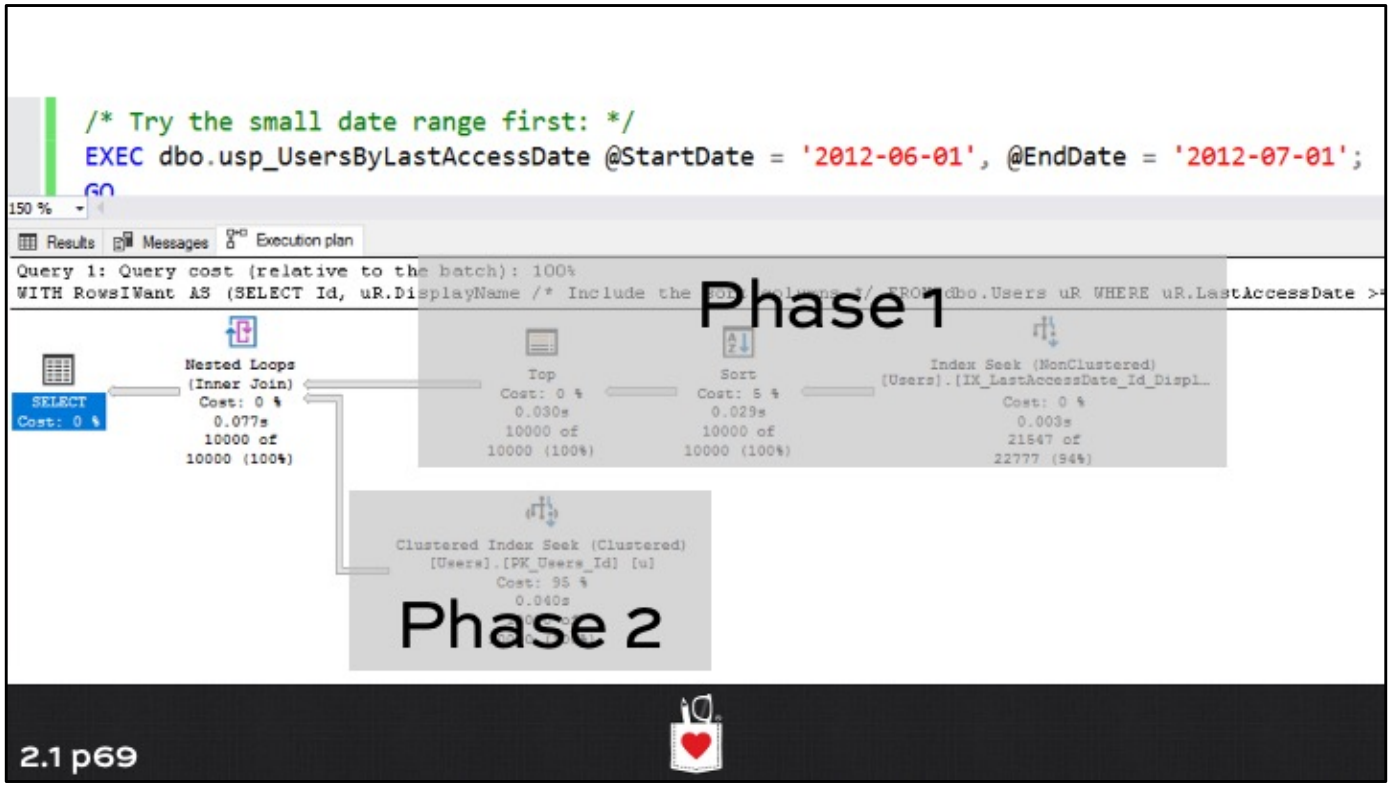
Phase 2:

go back to the clustered index to get all of the columns, but only for a limited number of rows.

This helps avoid the tipping point.

```
SELECT u.*
  FROM RowsIWant r
  INNER JOIN dbo.Users u ON r.Id = u.Id
  ORDER BY r.DisplayName                    /* Use the CTE's sort columns */;
END
GO
```





Less logical reads than a table scan

```
/* Try the small date range first: */
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';
GO
```

150 %

Results | Messages | Execution plan

```
(10000 rows affected)
Table 'Users'. Scan count 1, logical reads 30763, physical reads 0, read-ahead reads 0, lob logica
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logica
```

But it's too early to declare victory here:
the small date range always did this when it ran first.

2.1 p70

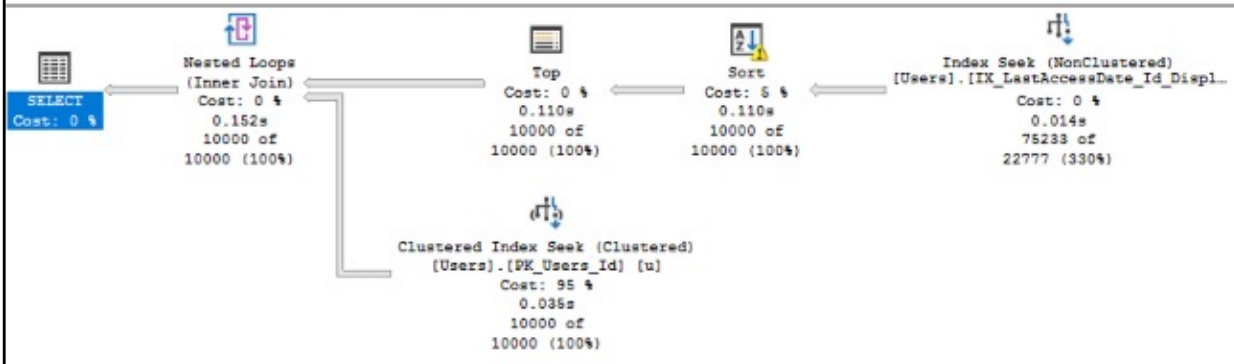
Now run the larger date range, but reusing the small date range's plan:

```
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
```

150 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
WITH RowsIWant AS (SELECT Id, uR.DisplayName /* Include the sort columns */ FROM dbo.Users uR WHERE uR.LastAccessDate
```



2.1 p71

And its logical reads are better than ever!

```
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
```

150 %

Results | Messages | Execution plan

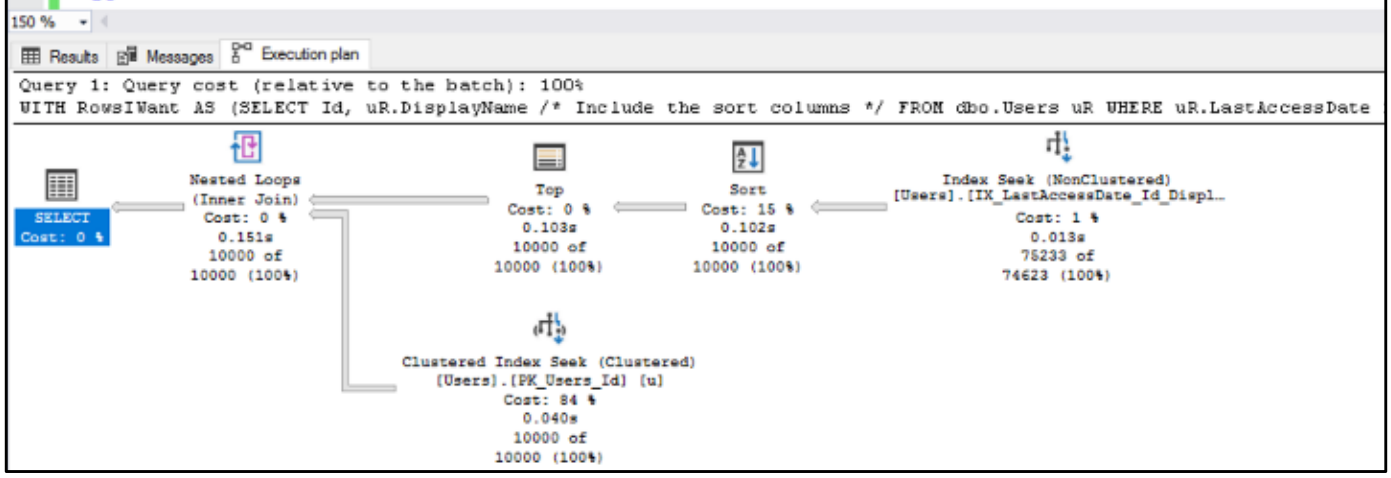
```
(10000 rows affected)
Table 'Users'. Scan count 1, logical reads 31073, physical reads 0, read-ahead reads 0, lob logi
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 301, lob lo
```

It's less than a table scan – but of course, it's because we're only returning 10,000 rows. (If someone wants terrible performance by returning all of the rows in one result set, they're still welcome to do that – but at least they're opting in at that point.)

2.1 p72

Now put the big plan in memory first...

```
ALTER TABLE dbo.Users REBUILD;
GO
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
GO
```



And it does the same number of reads!

Because the big plan is now much closer to the small plan.

```
ALTER TABLE dbo.Users REBUILD;
GO
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';
GO
```

150 %

Results | Messages | Execution plan

```
(10000 rows affected)
Table 'Users'. Scan count 1, logical reads 31073, physical reads 0, read-ahead reads 0, lob logic
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logic
```

2.1 p74

The big and small plans aren't identical.

A lot of things can still vary about the plan:

- Degrees of parallelism
- Memory grants (especially for sorts)
- Picking between multiple nonclustered indexes

But as long as you've got indexes to support your queries,
this technique means you're WAY more likely to use those indexes.

Frequently Asked Questions

What's the right PageSize?

Can someone get all the rows if they really want to?

Is there a solution for SQL 2008?

```
CREATE OR ALTER PROC dbo.usp_UsersByLastAccessDate
    @StartDate DATETIME,
    @EndDate    DATETIME,
    @PageNumber INT = 1,
    @PageSize   INT = 10000 AS
```

2.1 p76

2. Tuning key lookups around bad estimates

2.1 p77

I need a harder query for this.

The example I've been using so far only has 1 table in it.

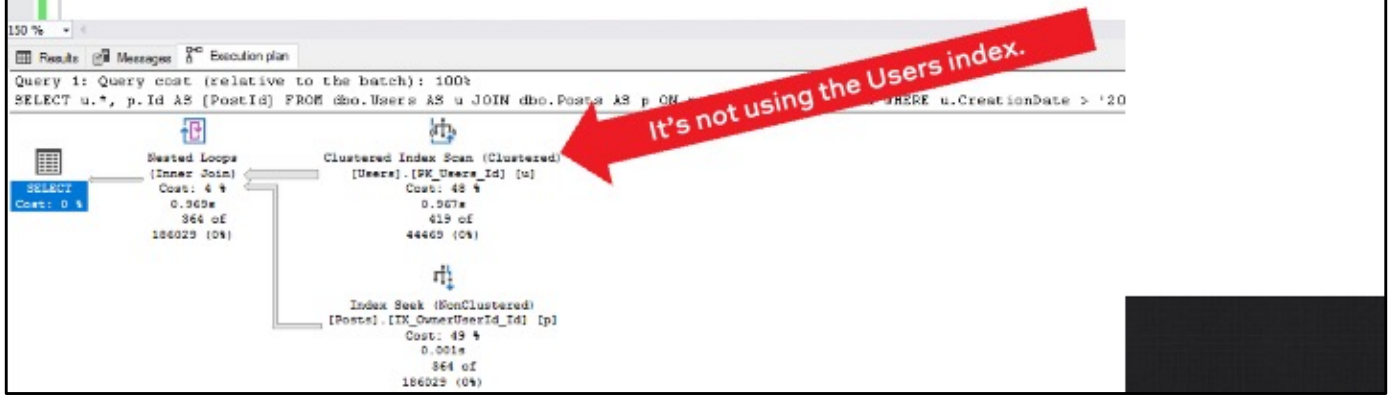
Your real-world queries don't look like that anyway.

So here comes the harder one...

```
We need a harder query on this one, especially a join between tables.
Here are two indexes we created, and the query we're trying to tune:
*/
CREATE INDEX IX_CreationDate_Reputation_Id ON dbo.Users (CreationDate, Reputation, Id);
GO
CREATE INDEX IX_OwnerUserId_Id ON dbo.Posts (OwnerUserId, Id) INCLUDE (PostTypeId);
GO

SELECT u.*, p.Id AS [PostId]
FROM   dbo.Users AS u
JOIN   dbo.Posts AS p
ON p.OwnerUserId = u.Id
WHERE  u.CreationDate > '2018-05-01'
        AND u.Reputation > 100
        AND p.PostTypeId = 1
ORDER BY u.Id;
```

2.1 p79

```
CREATE INDEX IX_CreationDate_Reputation_Id ON dbo.Users (CreationDate, Reputation, Id);
GO
CREATE INDEX IX_OwnerUserId_Id ON dbo.Posts (OwnerUserId, Id) INCLUDE (PostTypeId);
GO

SELECT u.*, p.Id AS [PostId]
FROM   dbo.Users AS u
JOIN   dbo.Posts AS p
ON p.OwnerUserId = u.Id
WHERE  u.CreationDate > '2018-05-01'
        AND u.Reputation > 100
        AND p.PostTypeId = 1
ORDER BY u.Id;
```



Why it won't use the index

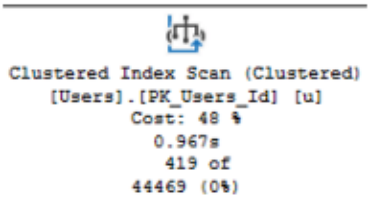
Our query has SELECT *.

The index doesn't cover all of those fields.

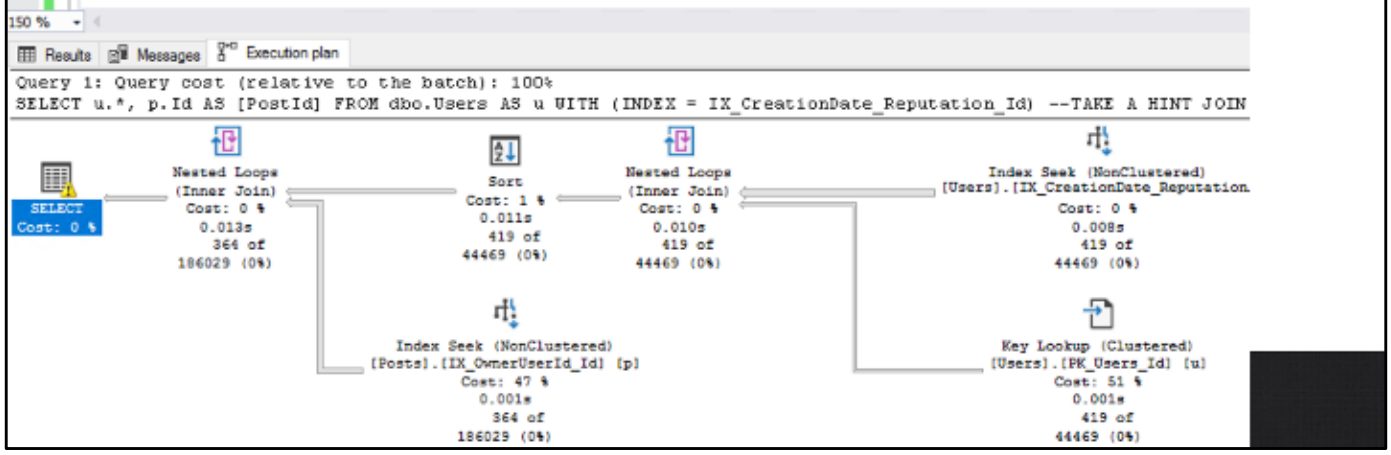
The index DOES cover the WHERE clause.

But SQL Server thinks it's going to have to do a whole lot of key lookups because it thinks a whole lot of rows are going to match.

It's wrong, though:
419 actual of
44,469 estimated.



```sql
/* We can force the index: */
SELECT u.*, p.Id AS [PostId]
FROM   dbo.Users AS u WITH (INDEX = IX_CreationDate_Reputation_Id) --TAKE A HINT
JOIN   dbo.Posts AS p
ON p.OwnerUserId = u.Id
WHERE  u.CreationDate > '2018-05-01'
        AND u.Reputation > 100
        AND p.PostTypeId = 1
ORDER BY u.Id;
```



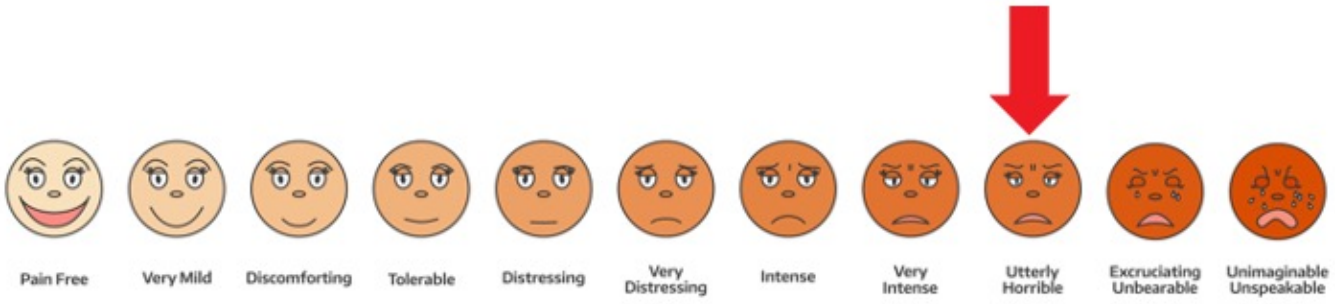
Because the estimated costs are wrong.

	Logical reads	Estimated subtree cost
Users clustered index scan (SQL Server's choice)	142,648	240
Nonclustered index seek (with hint)	3,104	248

The estimated costs of the clustered index scan are lower because SQL Server (incorrectly) expects so many rows to come back.

2.1 p83

BOGUS ESTIMATES



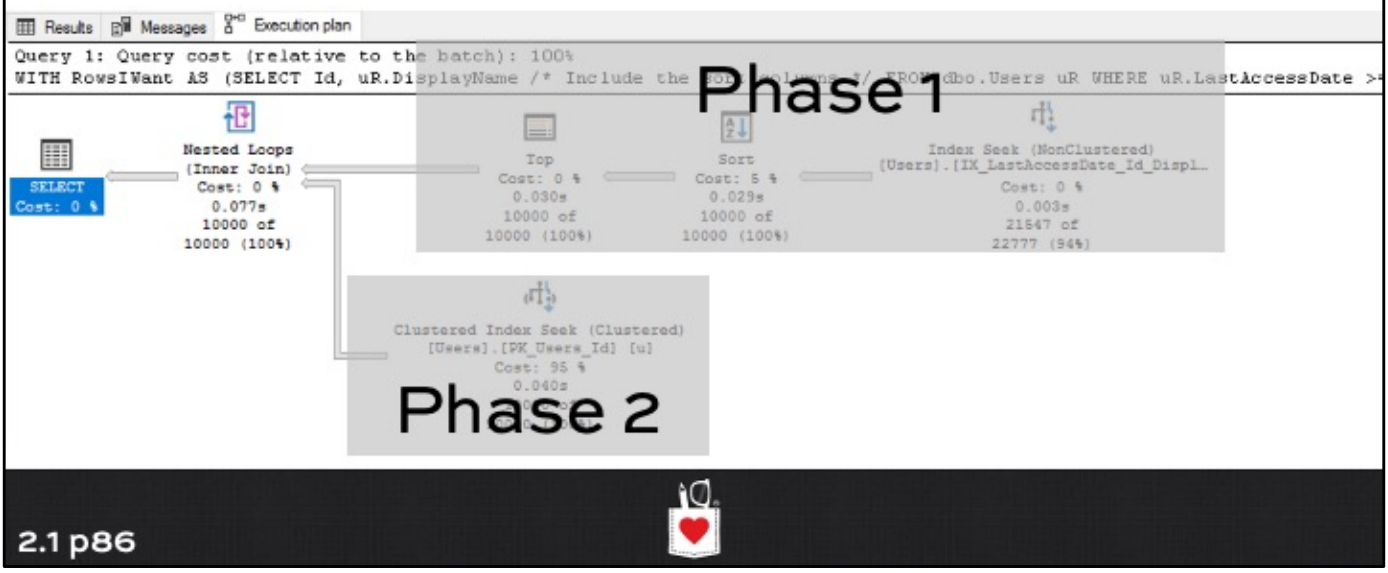
2.1 p84

How do we make this smarter?

We don't want to keep that hint

- Forcing an index hint takes choices away from the optimizer
- The Key Lookup plan might not be awesome for all predicates
- Data may change, index might not *stay* awesome for this query

Earlier, we broke queries into phases with a CTE. Can I do that here?



2.1 p86

```sql
/* CTE */
WITH RowsIWant AS (
    SELECT u.Id, p.Id AS [PostId]
    FROM dbo.Users AS u
    JOIN   dbo.Posts AS p
    ON p.OwnerUserId = u.Id
    WHERE u.CreationDate > '2018-05-01'
    AND u.Reputation > 100
    AND p.PostTypeId = 1
                )
SELECT u.*, r.PostId
FROM RowsIWant r
JOIN dbo.Users AS u
ON r.Id = u.Id
ORDER BY u.Id;
GO
```

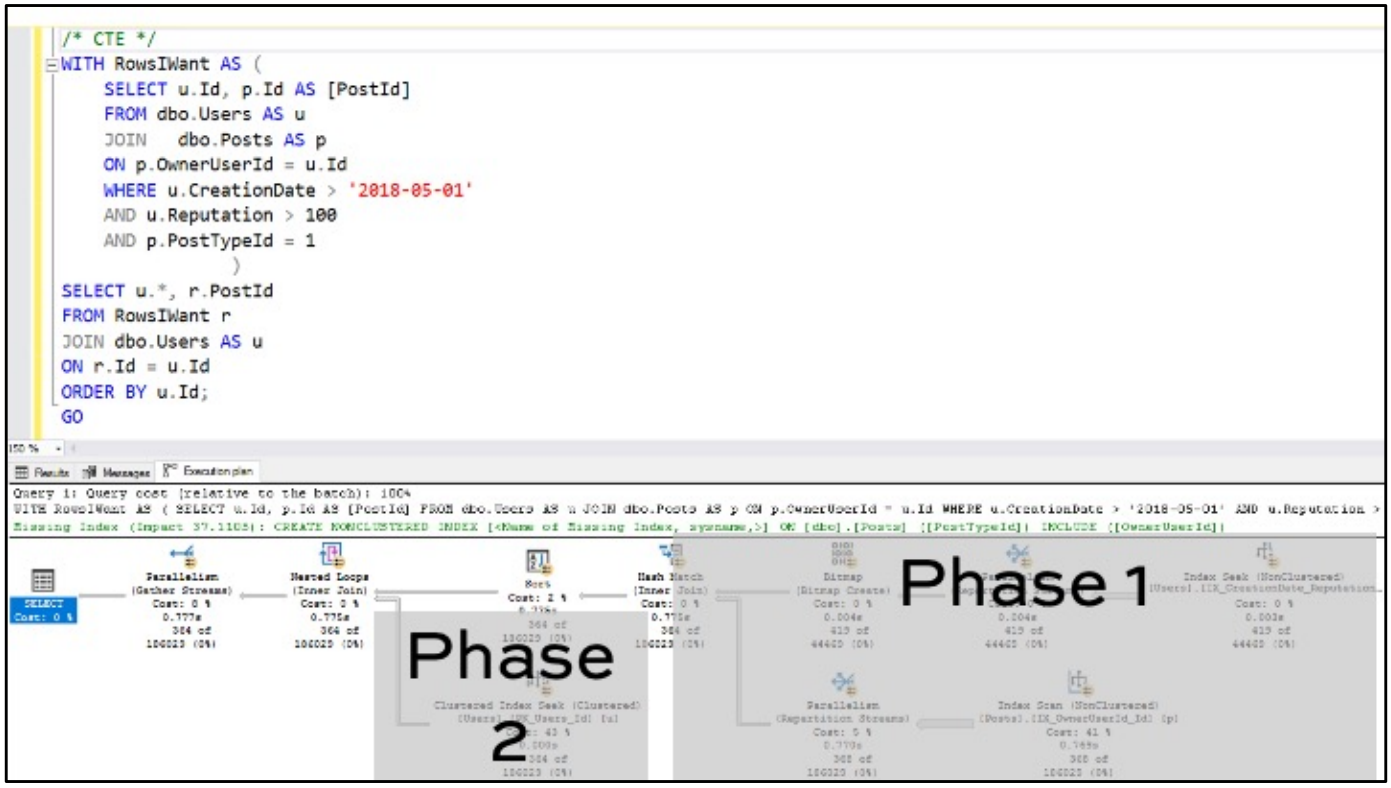
Phase 1:

filtering, should only use indexes, only return ID columns to do joins later

Phase 2:

“key lookups”
(but they’re really going to be clustered index seeks here)

2.1 p87



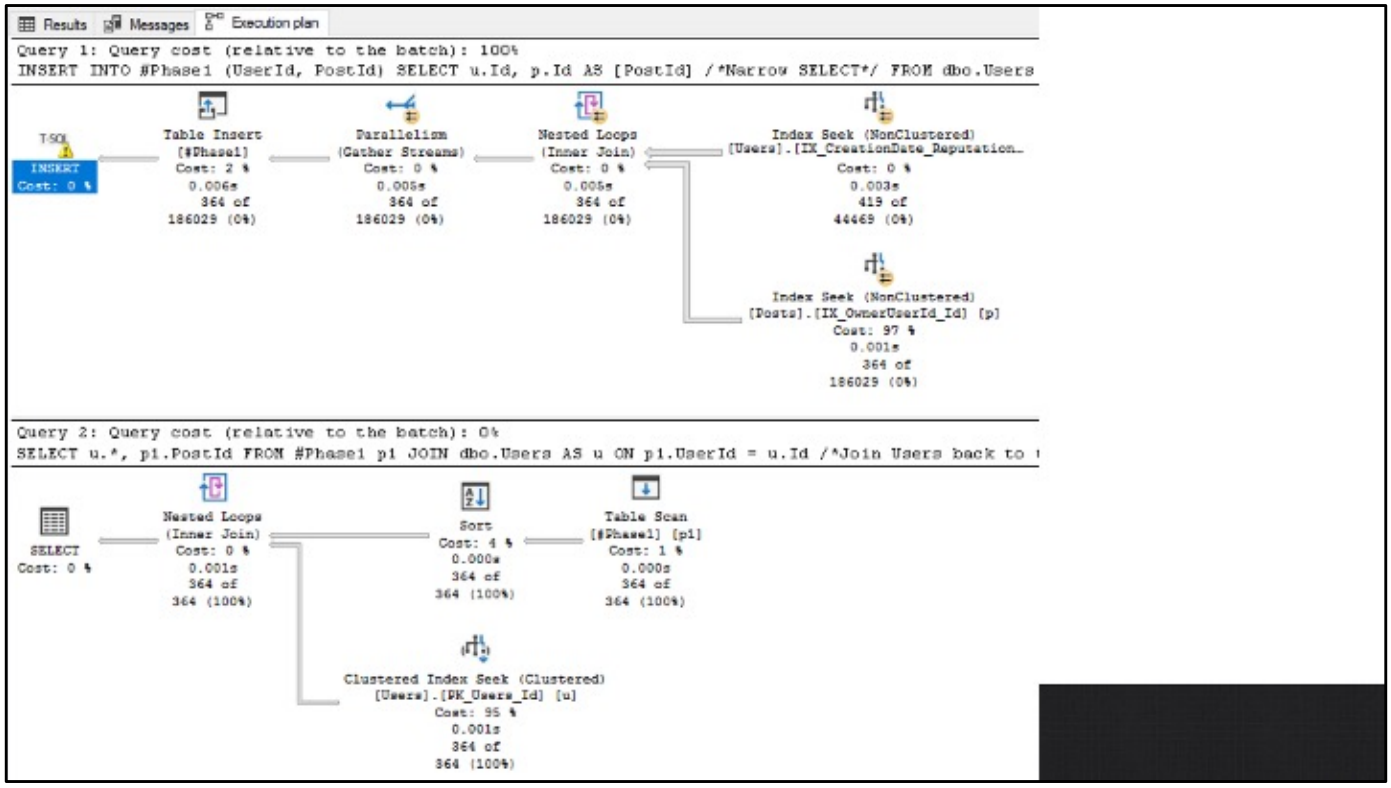
Well, better, but...not great.

	Logical reads	Estimated subtree cost
Users clustered index scan (SQL Server's choice)	142,648	240
Nonclustered index seek (with hint)	3,104	248
CTE	93,392	

I bet we could do better.

```sql
CREATE TABLE #Phase1 (UserId INT, PostId INT);
INSERT INTO #Phase1 (UserId, PostId)
    SELECT u.Id, p.Id AS [PostId] /*Narrow SELECT*/
    FROM dbo.Users AS u
    JOIN   dbo.Posts AS p ON p.OwnerUserId = u.Id /*Same join*/
    WHERE  u.CreationDate > '2018-05-01' /*Same predicates*/
    AND u.Reputation > 100 /*...*/
    AND p.PostTypeId = 1; /*...*/

/* Then join to the temp table: */
SELECT u.*, p1.PostId
FROM #Phase1 p1
JOIN dbo.Users AS u ON p1.UserId = u.Id /*Join Users back to the CTE on just the Ids we need*/
ORDER BY u.Id;
GO

DROP TABLE #Phase1;
GO
```

2.1 p90



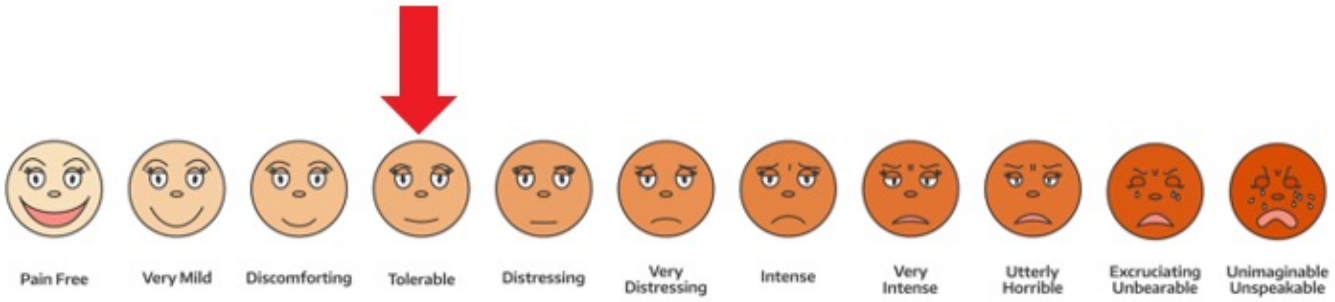
That's what I'm talkin' about!

	Logical reads	Estimated subtree cost
Users clustered index scan (SQL Server's choice)	142,648	240
Nonclustered index seek (with hint)	3,104	248
CTE	93,392	
Temp table	3,036	

Ta-dah!

2.1 p92

CTES & TEMP TABLES



2.1 p93

3. The nuclear option

2.1 p95

Say you're tired of being the nice DBA.

2.1 p96

Say you're tired of being the nice DBA.

```
/* Fix #3: going nuclear */
ALTER TABLE dbo.Users ADD ItoldYouToStopSelectingStar AS 1 / 0;
GO
```

2.1 p97

Say you're tired of being the nice DBA.

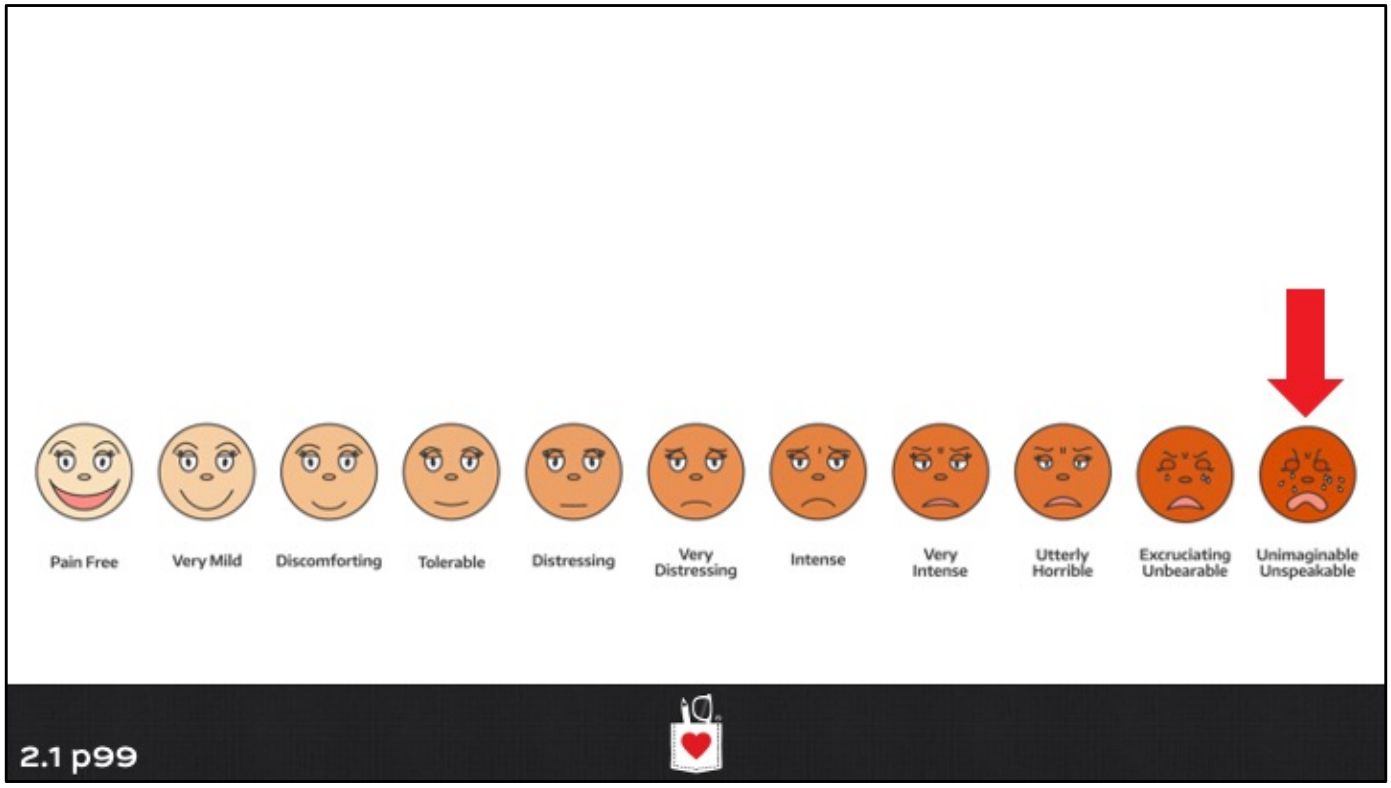
```
/* Fix #3: going nuclear */
ALTER TABLE dbo.Users ADD ItoldYouToStopSelectingStar AS 1 / 0;
GO
SELECT * FROM dbo.Users;
```

150 %

Results | Messages

```
Msg 8134, Level 16, State 1, Line 535
Divide by zero error encountered.
```

2.1 p98



Recap

2.1 p100

SELECT * gets a bad rap.

For a few to a few thousand rows, it's not that bad.

It beats trying to build giant covering indexes that kill your D/U/Is.

Just start by building indexes to support:

- WHERE
- JOINs
- GROUP BY
- ORDER BY
- Starting with just the first ones in this list,
  and gradually adding more keys until the pain is tolerable.

2.1 p101

Signs that you need these techniques

You've put at least an hour into testing different indexes for a query

Estimated number of rows is way off for one of the table operations

If you hint an index manually, you cut 80-90% of the reads or time

You're able to (significantly) change the query

If so:

- Try pagination first: maybe the app doesn't need all those rows
- Try CTE/temp table second: let SQL Server still build the plan
- Try index hints last: they're brittle technical debt