



**BRENT OZAR**  
UNLIMITED®

## Tuning for **SELECT \*** and Lots of Rows

2.1 p1

**SELECT \* has a bad reputation.**



Pain Free



Very Mild



Discomforting



Tolerable



Distressing



Very Distressing



Intense



Very Intense



Utterly Horrible



Excruciating  
Unbearable



Unimaginable  
Unspeakable

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

2.1 p2



**IF EXISTS  
(SELECT \***

2.1 p3



```
/* 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
```

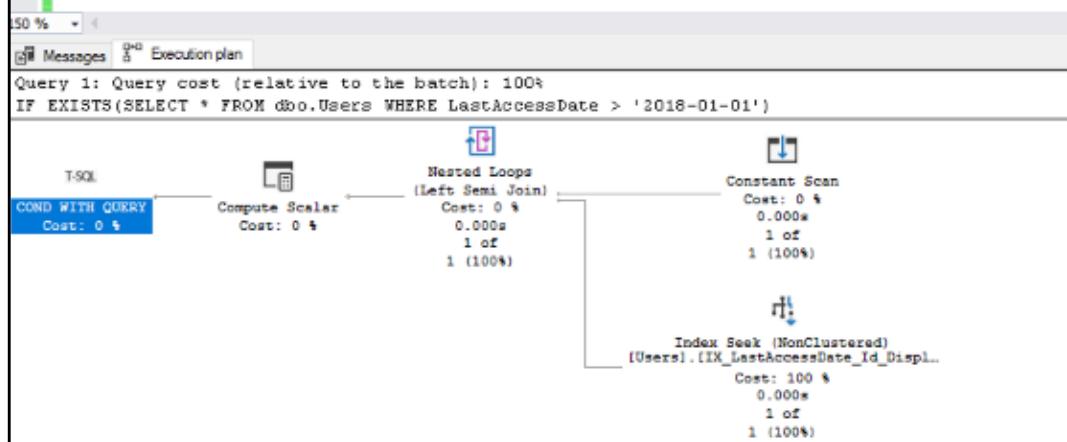
2.1 p4



```
/* 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
```

150 % 

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

IF EXISTS(SELECT \* FROM dbo.Users WHERE LastAccessDate > '2018-01-01')

T-SQL

COND WITH QUERY Cost: 0 %

Compute Scalar Cost: 0 %

Nested Loops (Left Semi Join) Cost: 0 % 0.000s 1 of 1 (100%)

Constant Scan Cost: 0 % 0.000s 1 of 1 (100%)

Index Seek (NonClustered) [Users].[IX\_LastAccessDate\_Id\_Displ...] Cost: 100 % 0.000s 1 of 1 (100%)

```

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

```

Messages Execution plan

Query 1: Query cost (relative to the batch): 1000  
 IF EXISTS(SELECT \* FROM dbo.Users WHERE LastAccessDate > '2018-01-01')

The execution plan shows a Nested Loops Left Semi Join node with a Compute Scalar child. The Compute Scalar node has a 'Cost: 0 %' and 'Cost: 0 %'. The Nested Loops node has a 'Cost: 0 + 0.000s' and '1 of 1 (100%)'. It branches to a Constant Scan node and an Index Seek node. The Constant Scan node has a 'Cost: 0 %' and '0.000s' with '1 of 1 (100%)'. The Index Seek node is highlighted in blue and has a tooltip 'Index Seek (NonClustered) IX\_LastAccessDate\_Id\_DisplayName\_Age'. The tooltip also lists various performance metrics.

Index Seek (NonClustered)  
 IX\_LastAccessDate\_Id\_DisplayName\_Age

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	1
Actual Number of Rows	1
Actual Number of Batches	0
Estimated I/O Cost	11.0065
Estimated Operator Cost	0.0032891 (1.00%)
Estimated CPU Cost	3.00000
Estimated Subtree Cost	0.0032891
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Number of Rows to be Read	2700000
Estimated Row Size	9B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

Object  
 [StackOverflow].[dbo].[Users]  
 [IX\_LastAccessDate\_Id\_DisplayName\_Age]  
 Seek Predicates  
 Seek Key([I]): Start: [StackOverflow].[dbo].[User].LastAccessDate > Scalar Operator('2018-01-01 00:00:00.000')

Query executed successfully.

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

150 %

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

IF EXISTS(SELECT \* FROM dbo.Users WHERE LastAccessDate > '2018-01-01')

**SQL Server used the nonclustered index, not the clustered index**

**It did a seek, not a scan**

**Number of rows read = 1**

**Index Seek (NonClustered)**  
 Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	1
Actual Number of Rows	1
Actual Number of Batches	0
Estimated I/O Cost	11.8365
Estimated Operator Cost	0.0032831 (100%)
Estimated CPU Cost	3.06949
Estimated Subtree Cost	0.0032831
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Number of Rows to be Read	2790300
Estimated Row Size	9B
Actual Rewinds	0
Ordered	True
Node ID	3
<b>Object</b>	
[StackOverflow].[dbo].[Users]	
[IX_LastAccessDate_Id_DisplayName_Age]	
Create Nonclustered Index	

## 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 reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

2.1 p8



## SQL Server optimizes the \* away here

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

The screenshot shows the SQL Server Management Studio interface. In the 'Messages' tab, the following T-SQL code is displayed:

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

In the 'Execution plan' tab, the plan for 'Query 1' is shown. The plan starts with a 'COND WITH QUERY' node (Cost: 0 \$) which feeds into a 'Compute Scalar' node (Cost: 0 \$). This is followed by a 'Nested Loops (Left Semi Join)' node (Cost: 0 \$, 0.000s, 1 of 1 (100%)). The right side of the semi join is a 'Constant Scan' node (Cost: 0 \$, 0.000s, 1 of 1 (100%)). The output of the semi join then feeds into an 'Index Seek (NonClustered)' node ([Users].[IX\_LastAccessDate\_Id\_Displ...], Cost: 100 \$, 0.000s, 1 of 1 (100%)).

# IF EXISTS (SELECT \*)



Pain Free



Very Mild



Discomforting



Tolerable



Distressing



Very Distressing



Intense



Very Intense



Utterly  
Horrible



Excruciating  
Unbearable



Unimaginable  
Unspeakable



2.1 p10

Next up:

# GETTING ONE ROW

2.1 p11



```

/* 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

```

50 % ▾

Results Messages Execution plan

	Id	AboutMe	Age	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate
1	10	I'm not takin' my sneakers off!    Actu...	NULL	2008-07-31 21:57:06.240	Sneakers O'Toole	0	NULL	2012-06-

```

/* 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

```

150 %

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT \* FROM [dbo].[Users] WHERE [LastAccessDate]=@1

```

Nested Loops (Inner Join)
    Cost: 0 %
    0.000s
    1 of
    1 (100%)

[Users].(IX_LastAccessDate_Id_Displ...)
    Cost: 50 %
    0.000s
    1 of
    1 (100%)

[Users].[PK_Users_Id]
    Cost: 50 %
    0.000s
    1 of
    1 (100%)

```

**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
```

100 % < Results Messages Execution plan

	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

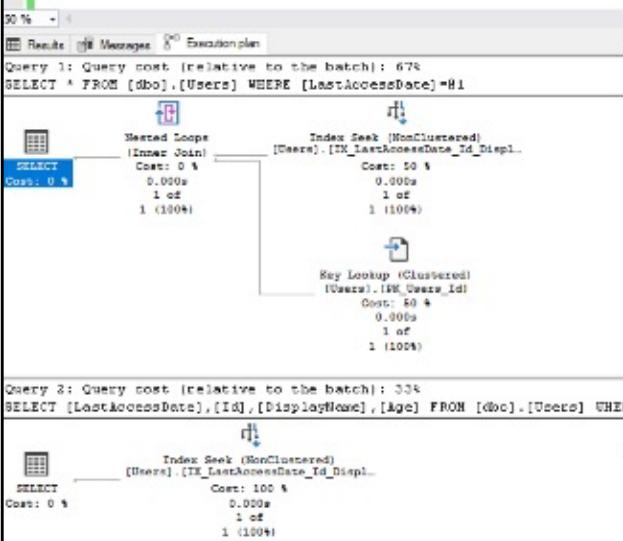
  

	LastAccessDate	Id	DisplayName	Age
1	2012-06-19 19:35:43.433	10	Sneakers O'Toole	NULL

```

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'
GO

```



## Compare the plans

The **SELECT \*** needs an extra operation, the key lookup.

So what's the overhead of that?



```
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,

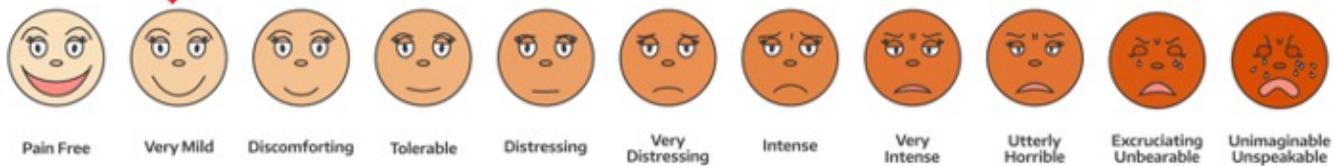
it's not worth building a covering index for that,

nor is it worth asking them to change their query.

2.1 p17



# ONE ROW



2.1 p18



Next up:

# GETTING A FEW ROWS

2.1 p19



```
/* 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



```

/* 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";
GO

```

Results (10 rows)

ID	UserId	Age	CreatedDate	DisplayName	DisplayValue	LastActivity	LastAccessDate	Location	Registration	Subscription	Voice	Verified	AccountType
1	10	30	2008-07-21 21:36:240	Brennan O'Toole	0	NULL	2012-06-19 19:35:43.433	Unknown, CA	101	0	3772	http://www.stackoverflow.com/login?go=1&id=704	0
2	546113	NULL	2012-06-13 12:31:05.031	Rick Flagg	0	NULL	2012-06-13 12:31:05.031	NULL	1	0	2	NULL	1522327
3	1487418	NULL	2012-06-19 19:35:57.219	Mike Flagg	0	NULL	2012-06-19 19:35:57.219	NULL	1	0	0	NULL	1522311
4	507506	NULL	2012-06-19 19:35:57.219	Kenneth	0	NULL	2012-06-19 19:35:57.219	Office, Canada	10	0	10	http://www.stackoverflow.com	247066
5	446270	NULL	2012-06-12 00:02:11.013	Piggy	0	NULL	2012-06-19 19:35:43.433	NULL	1	0	1	NULL	202207
6	5431933	NULL	2012-06-19 17:47:12.113	User103333	0	NULL	2012-06-19 17:47:12.113	NULL	1	0	0	NULL	1546522
7	1485085	NULL	2012-06-18 23:41:27.447	User1485085	0	NULL	2012-06-19 19:35:17.080	NULL	10	0	1	NULL	1573354
8	7003891	NULL	2011-04-13 19:51:05.131	Andrew Speedometer	0	NULL	2012-06-19 19:35:11.481	NULL	90	0	1	NULL	357517
9	1179663	NULL	2012-01-30 17:41:38.367	Alex Rand	0	NULL	2012-06-19 19:35:07.082	NULL	1	0	3	NULL	1298675
10	7300676	NULL	2012-06-14 20:51:16.031	User133424	0	NULL	2012-06-19 19:35:11.877	NULL	1	0	5	NULL	1498551
11	1227038	NULL	2012-04-11 21:31:16.101	pppe	0	NULL	2012-06-19 19:35:21.040	NULL	21	0	3	NULL	130312
12	1425480	NULL	2012-06-08 19:36:04.031	Tech A Cheap Internet Cafe	0	NULL	2012-06-19 19:35:04.031	NULL	1	0	10	NULL	1540398
13	1425239	NULL	2012-05-27 14:35:27.577	Daniel	0	NULL	2012-06-19 19:35:27.577	NULL	36	0	0	NULL	1520418
14	7462300	NULL	2012-06-17 19:50:27.181	User1462300	0	NULL	2012-06-19 19:35:53.367	NULL	71	0	6	NULL	1579561
15	1156492	NULL	2012-01-19 14:37:38.451	User1156492	0	NULL	2012-06-19 19:35:26.372	NULL	1	0	3	NULL	1161622
16	1485087	NULL	2012-06-13 12:36:26.113	Jeffrey	0	NULL	2012-06-19 19:35:49.793	NULL	21	0	4	NULL	1546516
17	1487540	NULL	2012-06-15 19:31:06.031	sk8phnrmnd	0	NULL	2012-06-19 19:35:01.030	NULL	1	0	3	NULL	1522218
18	1486212	NULL	2012-06-03 14:16:54.021	User1486212	0	NULL	2012-06-19 19:35:15.012	NULL	1	0	0	NULL	1548852
19	361667	NULL	2012-06-27 06:32:15.318	mlmilen	0	NULL	2012-06-19 19:35:01.007	NULL	21	0	37	http://tinyurl.com	143336
20	7404786	NULL	2012-06-18 20:51:56.261	User1404786	0	NULL	2012-06-19 19:35:38.027	NULL	1	0	4	NULL	1579561
21	1407471	NULL	2012-06-19 20:35:04.740	Natalie Wandler	0	NULL	2012-06-19 19:35:04.740	NULL	1	0	3	NULL	1522454
22	1289703	NULL	2012-05-22 21:31:55.318	Ethan	0	NULL	2012-06-19 19:35:32.967	NULL	11	0	1	NULL	1546330
23	1289828	NULL	2012-05-22 19:35:56.888	Sed	0	NULL	2012-06-19 19:35:21.777	NULL	11	0	4	http://www.brentozar.com	1546101
24	1688879	NULL	2011-13-10 21:16:14.121	eswartzend	0	NULL	2012-06-19 19:35:11.139	Verizon	26	0	9	NULL	1522689

Query executed successfully.

(Total 142 PFM) | SQL2017LAB\Administrator | StackOverflow | 05:09:00 | 41 rows

## 2.1 p21

**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



```

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';
GO

```

150 %

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT \* FROM [dbo].[Users] WHERE [LastAccessDate]>=81 AND [LastAccessDate]<82

```

Nested Loops (Inner Join)
    Cost: 0 %
        0.000s
        43 of
        33 (130%)

Index Seek (NonClustered)
    [Users].(IX_LastAccessDate_Id_DisplayName_Age)
    Cost: 3 %
        0.000s
        43 of
        33 (130%)

Key Lookup (Clustered)
    [Users].[PK_Users_Id]
    Cost: 37 %
        0.000s
        43 of
        33 (130%)

```

**Yay! It uses the index.**

What's the cost?

```

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';

```

Results

	ID	AboutMe	Age	CreationDate	DisplayName
1	10	I'm not skin' my sneakers off! :Dn:Dn: Atu...	NULL	2008-07-01 21:57:05.240	Sneakers O'Toole
2	1467413	NULL	NULL	2012-06-19 19:35:49.597	Rick Ranjan
3	1467416	NULL	NULL	2012-06-19 19:36:57.210	Mann2920
4	521506	NULL	NULL	2010-11-26 15:07:53.570	Swinidessoft
5	446070	NULL	NULL	2010-09-13 08:00:11.813	Pri_ned
6	1437987	NULL	NULL	2012-06-05 17:41:42.113	user1437987
7	1465065	NULL	NULL	2012-06-10 23:49:27.447	user1465065
8	1003858	NULL	NULL	2011-10-19 18:54:09.197	Andrew Speedwalker
9	1178659	NULL	NULL	2012-01-30 17:41:39.387	Adam Baird
10	1334674	NULL	NULL	2012-05-14 20:52:16.630	user1334674
11	1327638	NULL	NULL	2012-04-11 21:11:16.180	rggg

LastAccessDate	ID	DisplayName	Age
2012-06-19 19:35:43.433	10	Sneakers O'Toole	NULL
2012-06-19 19:35:49.597	1467413	Rick Ranjan	NULL
2012-06-19 19:36:57.210	1467416	Mann2920	NULL
2012-06-19 19:37:48.770	521506	Swinidessoft	NULL
2012-06-19 19:41:45.273	446070	Pri_ned	NULL
2012-06-19 19:42:39.567	1437987	user1437987	NULL
2012-06-19 19:43:17.660	1465065	user1465065	NULL
2012-06-19 19:45:14.680	1003858	Andrew Speedwalker	NULL
2012-06-19 19:45:57.060	1170659	Adam Baird	NULL
2012-06-19 19:46:19.577	1334674	user1334674	NULL
2012-06-19 19:48:31.840	1327638	rggg	NULL

## Checking the cost

Again, we'll run two versions:

One asking for all the columns

One only asking for columns in the nonclustered 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';

```

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 874

SELECT \* FROM [dbo].[Users] WHERE [LastaccessDate]>='2012-06-19 19:35:43.433' AND [LastaccessDate]<'2012-06-19 20:35:43.433'

Query 2: Query cost (relative to the batch): 33

SELECT [LastaccessDate],[Id],[DisplayName],[Age] FROM [dbo].[Users] WHERE [

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

```
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 reads
(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';
```

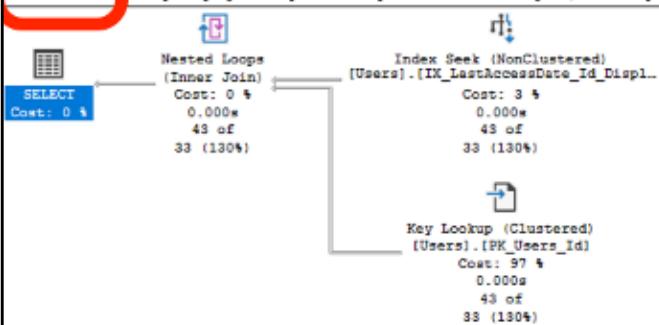
```
/* Do less columns help? */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
    ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

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

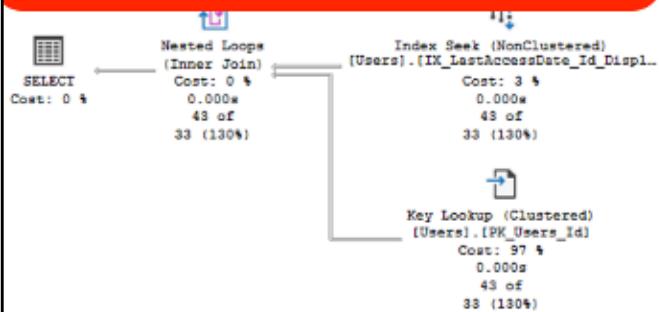
```
Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>=01 AND [LastAccessDate]<82
```



No help here.

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

```
SELECT [LastAccessDate],[Id],[DisplayName],[Age],[Location] FROM [dbo].[Users] WHERE
```



```
/* 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
    FROM dbo.Users
    WHERE LastAccessDate >= '2012-06-19 19:35:43.433'
        AND LastAccessDate <  '2012-06-19 20:35:43.433';
GO
```

50 %

Results | Messages | Execution plan

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

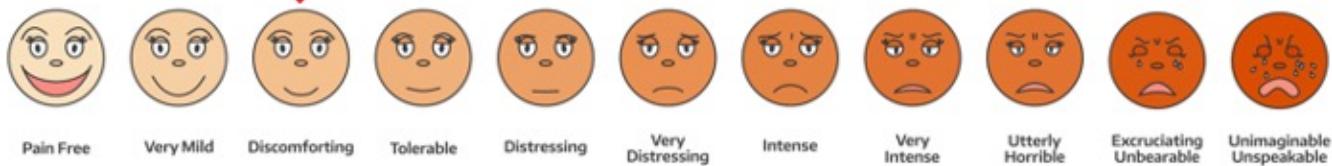
(1 row affected)

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

(1 row affected)
```

Same reads, too.

# A FEW ROWS



2.1 p32



Next up:

# GETTING THOUSANDS OF ROWS

2.1 p33



```

/* 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

```

50 %

	Results												
	Messages												
	Execution plan												
1	<a href="#">AboutMe</a>	<a href="#">Id</a>	<a href="#">Age</a>	<a href="#">CreationDate</a>	<a href="#">DisplayName</a>	<a href="#">DownVotes</a>	<a href="#">EmailHash</a>	<a href="#">LastAccessDate</a>	<a href="#">Location</a>	<a href="#">Reputation</a>	<a href="#">UpVotes</a>	<a href="#">Views</a>	<a href="#">WebsiteUrl</a>
1	I'm not takin' my sneakers off! cbn-cbs. Actually, s...	10	NULL	2006-03-21 15:26:24.0	SnakesO'Toole	0	NULL	2012-06-19 16:35:03.000	Lakehead, CA	101	0	3570	<a href="http://www.youtube.com/watch?v=14295671">http://www.youtube.com/watch?v=14295671</a>
2	34	94	NULL	2006-08-01 18:26:19.007	Stephen R.	0	NULL	2012-06-22 04:18:33.007	Chicago, IL	114	3	52	<a href="http://ontheweb.com">http://ontheweb.com</a>
3	2166	I enjoy yogurt and needles. Fruit is good too.	NULL	2006-08-20 17:44:01.357	Carmine Santini	0	NULL	2012-06-13 17:55:03.637	Portland, OR	296	91	50	
4	2406	Programmer	NULL	2008-08-22 01:54:04.103	cavemym	0	NULL	2012-06-21 01:58:03.353	Massachusetts	276	0	36	<a href="http://cavemym.com">http://cavemym.com</a>
5	2888	@My name is Peter Haven. I am the COO for valh...	NULL	2008-08-25 18:26:19.000	Peter Haven	0	NULL	2012-06-22 20:03:07.413	Calgary, Canada	101	0	69	<a href="http://www.haven.ca">http://www.haven.ca</a>
6	2930		NULL	2006-08-26 07:40:42.990	Timm	0	NULL	2012-06-12 06:29:01.037	NULL	488	2	87	
7	3351	Leading ..	NULL	2006-08-28 05:23:07.940	David	12	NULL	2012-06-20 10:41:20.073	Indiana	3216	70	119	<a href="http://www.torleme.org">http://www.torleme.org</a>
8	3533		NULL	2006-08-29 01:57:10.200	David	0	NULL	2012-06-20 17:19:50.750	Richmond, VA	643	20	53	<a href="http://www.davidcawiller.com">http://www.davidcawiller.com</a>
9	3548	I'm a student at Stony Brook University. Interested in...	NULL	2008-08-29 04:51:29.737	Tyman	1	NULL	2012-06-30 17:19:39.810	Stony Brook, NY	131	13	77	<a href="http://tyman.ca">http://tyman.ca</a>
10	3766	Hi	NULL	2006-08-29 05:23:15.513	...	...	...	2012-06-20 16:48:45.000	...	217	0	0	
		<a href="#">Id</a>											
1		14295671											
2		500213											
3		1410808											
4		1429638											
5		1421503											
6		1321154											
7		1306653											
8		1422660											
9		1371327											

Query executed successfully.

```

/* 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

```

Results grid: Execution plan

Query 1: Query cost (relative to the batch): 1000

```

SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>='2012-06-01' AND [LastAccessDate]<'2012-07-01'

```

Cost: 0 %  
0.004s  
21847 of  
22777 (94%)

Cost: 0 %  
0.004s  
21847 of  
22777 (94%)

Key Lookup (Discovered)  
[Users].[IX\_Users\_Id]  
Cost: 100 %  
0.04s  
21847 of  
22777 (94%)

Query 2: Query cost (relative to the batch): 0%

```

SELECT [Id] FROM [dbo].[Users] WHERE [LastAccessDate]>='2012-06-01' AND [LastAccessDate]<'2012-07-01'

```

Cost: 0 %  
0.014s  
21847 of  
22324 (97%)

## 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?

```
/* 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
```

Ouch.

66,125 reads vs 129.

50 % ▾

Results | Messages | Execution plan

```
(21547 rows affected)
Table 'Users'. Scan count 1, logical reads 66125, p
Table 'Worktable'. Scan count 0, logical reads 0, p

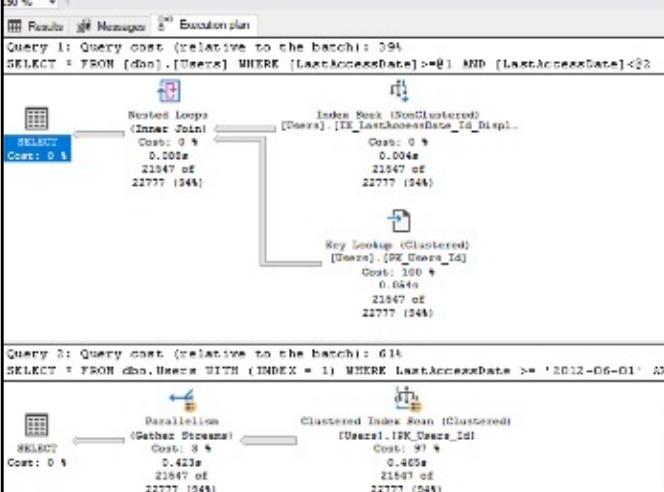
(1 row affected)

(21547 rows affected)
Table 'Users'. Scan count 1, logical reads 129, phy
(1 row affected)
```

```

/* Would a clustered index scan have read less pages? */
SELECT *
FROM dbo.Users
WHERE LastAccessDate >= '2012-06-01'
    AND LastAccessDate < '2012-07-01';
GO
SELECT *
FROM dbo.Users WITH (INDEX = 1)
WHERE LastAccessDate >= '2012-06-01'
    AND LastAccessDate < '2012-07-01';
GO

```



```
/* Would a clustered index scan have read less pages? */
SELECT *
FROM dbo.Users
WHERE LastAccessDate >= '2012-06-01'
    AND LastAccessDate < '2012-07-01';
GO
SELECT *
FROM dbo.Users WITH (INDEX = 1)
WHERE LastAccessDate >= '2012-06-01'
    AND LastAccessDate < '2012-07-01';
GO
```

150 % Results Messages Execution plan

```
(21547 rows affected)
Table 'Users'. Scan count 1, logical reads 66125, physical reads
Table 'Worktable'. Scan count 0, logical reads 0, physical reads
```

```
(1 row affected)
```

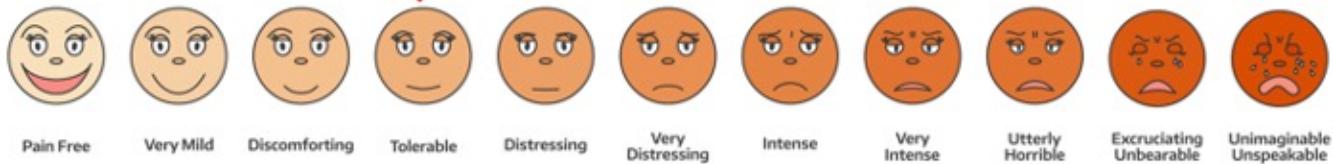
```
(21547 rows affected)
Table 'Users'. Scan count 5, logical reads 142194, physical reads
Table 'Worktable'. Scan count 0, logical reads 0, physical reads
```

**Awesome!**

**SQL Server chose wisely: the nonclustered index makes more sense for one month of data.**

**It's still more efficient than scanning the entire table.**

# THOUSANDS OF ROWS



2.1 p39



Next up:

# GETTING VARIABLE NUMBERS OF ROWS

2.1 p40



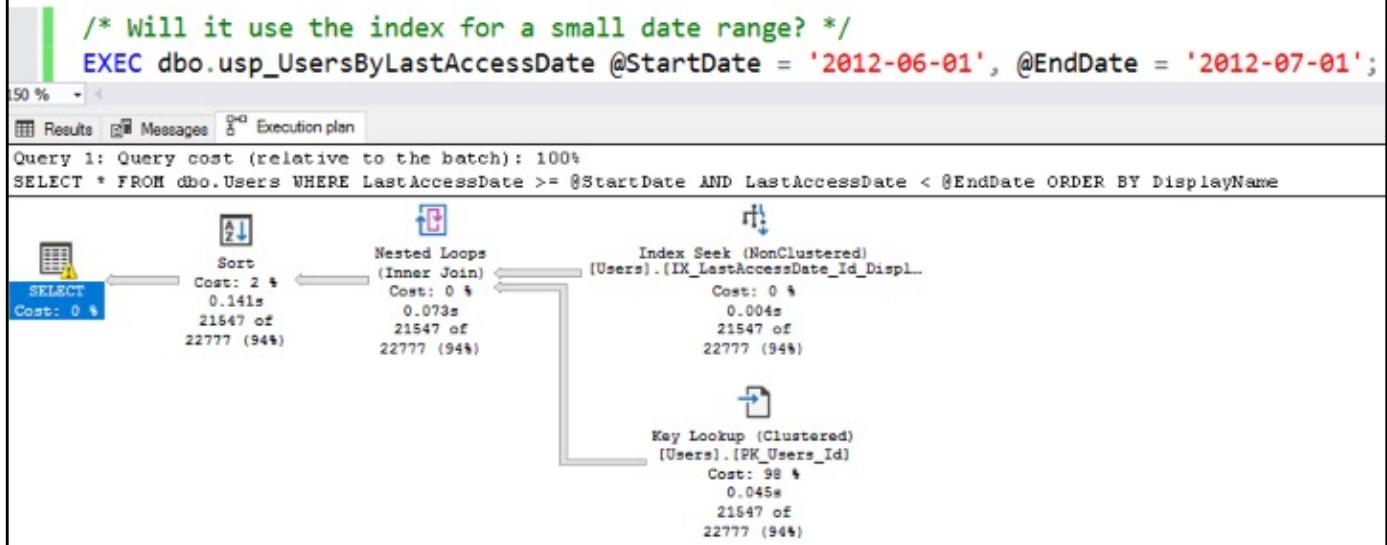
```
/* Let's add parameters and an ORDER BY: */

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...



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



The screenshot shows a SQL Server Management Studio window with the following content:

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

Results tab selected. Output:

(75233 rows affected)

Table 'Worktable'. Scan count 0, logical reads 0, 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

A red oval highlights the line "logical reads 230849" for the 'Users' table.

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**

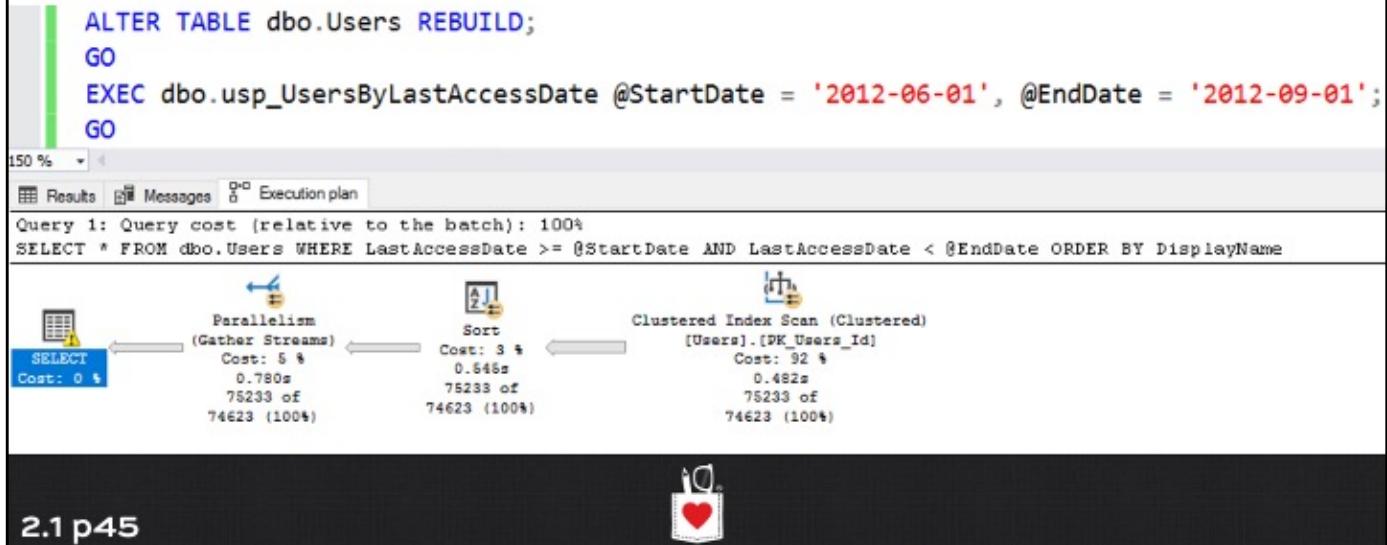
**Update statistics**

2.1 p44



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

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



## 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
```

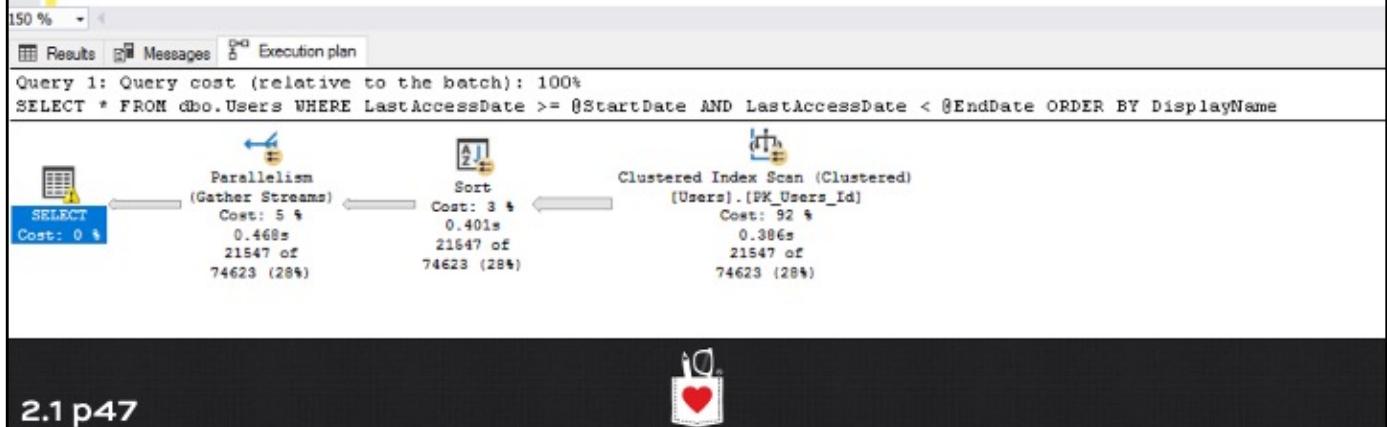
2.1 p46



## 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
```



## 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 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 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](http://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	<a href="http://forums.m
556	101	7/15/09 9:45 AM	Russ Cam	11/18/09 6:45 PM	UK	NULL	Interested in C#, ASP.NET, !
557	1	7/15/09 9:45 AM	Becky	2/21/10 7:01 AM	NULL	NULL	NULL
558	99	7/15/09 9:46 AM	marco.rogagn	4/1/10 7:41 AM	IT	29	<n3>Software Developer</
559	593	7/15/09 9:46 AM	marco.rogagn	2/18/10 7:14 AM	AU	39	Hey, at least I know 6502!
560	83	7/15/09 9:46 AM	rystelling	3/4/10 6:47 PM	United Kingdom	30	<n3>Tech geek and hothead!
561	716	7/15/09 9:47 AM	baloha	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	Lenden	NULL	NULL
563	101	7/15/09 9:48 AM	arturm	10/29/09 2:11 PM	Bremen, Germany	24	NULL
564	960	7/15/09 9:49 AM	Giff	4/1/10 6:59 AM	UK	30	<p>I'm a Software Engineer
565	121	7/15/09 9:49 AM	Franci Pemov	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

ID	Rep	CreateTime	Display Name	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Ahweed	4/1/10 10:35 AM	El Centro, CA	39	
2	126	7/12/09 10:51 PM	Goeff Dalakas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the Stack Overflow team
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morgan Hill, NC	31	Developer on the Stack Overflow team
4	767	7/12/09 10:51 PM	Jeff Spolsky	3/30/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflow
535	398	7/15/09 9:33 AM	jeff	2/18/10 9:27 PM	Scotland	23	Twitter@jeff
536	101	7/15/09 9:34 AM	second	3/10/10 9:56 PM	NULL	NULL	NULL
537	128	7/15/09 9:35 AM	staffan	5/25/10 7:18 PM	Sweden	38	I work on the JBossit API
538	98	7/15/09 9:35 AM	cgreene	1/19/10 10:54 AM	London	34	A Canadian living in London
539	107	7/15/09 9:37 AM	Antonius	3/12/10 9:58 AM	NY	27	I work as a software developer
540	101	7/15/09 9:37 AM	Cassingerman	12/7/09 6:37 PM	NULL	NULL	<>I'm fairly advanced at
541	641	7/15/09 9:38 AM	James Hart	4/1/10 6:37 AM	England	NULL	NULL
542	7071	7/15/09 9:38 AM	hynek	4/1/10 10:49 PM	NULL	NULL	NULL
543	409	7/15/09 9:38 AM	DaveChirico	12/12/09 5:12 PM	London, UK	23	Primary C# .NET developer
544	102	7/15/09 9:39 AM	Ben Laam	3/28/10 11:12 PM	Adelaide, Australia	34	C#, Delphi, T-SQL, Architect
545	985	7/15/09 9:40 AM	redacted	4/1/10 1:34 PM	NULL	23	QA
546	103	7/15/09 9:43 AM	Nitec	8/26/09 9:59 AM	Burnaby	27	System.StackOverflowExe
547	403	7/15/09 9:43 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	AK Joshua	2/26/10 5:53 PM	Orlando, FL	38	Software Engineer. Howe will
550	61	7/15/09 9:42 AM	hipposa	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	Eduard	11/29/09 7:20 AM	Moscalito, Venezuela	25	NULL
553	103	7/15/09 9:44 AM	matf	2/3/10 10:28 AM	Sweden	NULL	NULL
554	1298	7/15/09 9:44 AM	becomingGuru	4/1/10 8:43 PM	Bangalore / Hyderabad	23	curious learner!
555	1016	7/15/09 9:45 AM	MrTheFox	4/1/10 6:13 PM	Huntington, NC	NULL	<><a href="http://rossum.org">
556	101	7/15/09 9:45 AM	Russ Cam	11/18/09 6:45 PM	UK	NULL	Interested in C#, ASP.NET, :
557	1	7/15/09 9:45 AM	Beto	2/21/10 7:50 AM	NULL	NULL	NULL
558	953	7/15/09 9:46 AM	marcosnegra	4/1/10 7:41 AM	IT	29	Software Developer</
559	59	7/15/09 9:46 AM	IT	2/18/10 7:14 AM	IT	39	How at least I know 6500!
560	23	7/15/09 9:46 AM	rstelling	3/4/10 6:47 PM	United Kingdom	30	<>Tech geek, and hoodie
561	716	7/15/09 9:47 AM	balloch	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	24	NULL
563	101	7/15/09 9:48 AM	artem	10/29/09 2:11 PM	Bremen, Germany	4/1/10 6:59 AM	30 <>I'm a Software Enginee
564	900	7/15/09 9:49 AM	zbar	3/5/10 12:13 AM	Kirkland, WA	37	<>Father, SDE@Microsoft

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	CreateTime	Display Name	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Ahweed	4/1/10 10:35 AM	El Centro, CA	39	 at least I know it's20!
560	23	7/15/09 9:46 AM	rstelling	3/4/10 6:47 PM	United Kingdom	30	<> Tech geek, and hoodie!
561	716	7/15/09 9:47 AM	balloha	4/1/10 4:00 PM	NUL	NULL	NULL
562	43	7/15/09 9:47 AM	Mike McClelland	3/18/10 8:45 AM	London	NULL	NULL
563	101	7/15/09 9:48 AM	artem	10/26/09 2:13 PM	Bremen, Germany	24	NULL
564	900	7/15/09 9:49 AM	Szar	4/1/10 6:59 PM	NAN	30	<> I'm a Software Enginee
565	121	7/15/09 9:49 AM	Francis Penney	3/5/10 12:17 AM	Kirkland, WA	37	<> 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	Display Name	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Ahwedd	4/1/10 10:35 AM	El Centro, CA	39	<a href="http://img277.jp/2406" title="Jeff Ahwedd's profile picture"></a>
2	126	7/12/09 10:51 PM	Goeff Dalakas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the Stack Overflow team.
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morgan Hill, NC	31	Developer on the Stack Overflow team.
4	767	7/12/09 10:51 PM	Jeff Spolsky	3/20/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflow.
535	398	7/15/09 9:33 AM	jeff	2/18/10 9:27 PM	Scotland	23	Twitter user: http://twitter.com/jeff
536	101	7/15/09 9:34 AM	second	3/10/10 9:56 PM	NULL	NULL	NULL
537	128	7/15/09 9:35 AM	stafford	3/25/10 7:10 PM	Sweden	38	I work on the JBoss Seam 3.0.0 RC1.
538	99	7/15/09 9:35 AM	cognos	1/19/10 10:54 PM	London	NULL	A Canadian living in London.
539	107	7/15/09 9:37 AM	Antonius	3/12/10 9:44 PM	NL	27	Wayne as a software developer.
540	101	7/15/09 9:37 AM	DouglasGorman	2/1/10 8:27 PM	NULL	NULL	Extremely advanced at SQL.
541	101	7/15/09 9:38 AM	eric	3/25/10 9:56 PM	England	NULL	NULL
542	5921	7/15/09 9:38 AM	hyperiong	4/1/10 10:49 PM	NULL	NULL	<a href="http://www.hyperiong.com">Hyperiong's profile picture</a>
543	490	7/15/09 9:38 AM	DavidWhitney	1/2/10 9:51:12 PM	London, UK	25	Primarily C# .NET developer.
544	103	7/15/09 9:39 AM	Bon Laan	3/28/10 11:12 PM	Adelaide, Australia	34	C#, Delphi, T-SQL : Architect
545	985	7/15/09 9:40 AM	redacted	4/1/10 10:34 PM	NAN	23	NaN
546	103	7/15/09 9:41 AM	htec	8/26/09 9:59 AM	Bombay	27	System.StackOverflowException
547	403	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	Sesseted web developer. W
549	173	7/15/09 9:42 AM	Joshua	2/26/10 5:53 PM	Orlando, FL	28	Software Engineer. Have wif
550	61	7/15/09 9:42 AM	hippsasa	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	Morcalbo, Venezuela	25	NULL
553	101	7/15/09 9:44 AM	moff	2/7/10 10:28 PM	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	MifTheFox	4/1/10 6:11 PM	Huntsville, AL	NULL	<a href="http://forums.mifthefox.com">MifTheFox's profile picture</a>
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	2/23/10 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</h2>
559	75	7/15/09 9:46 AM	robinm0000	2/1/10 8:47 PM	United Kingdom	30	MySQL, Java, T-SQL, VB.NET
560	63	7/15/09 9:46 AM	sp00n3rd	4/1/10 4:00 PM	NULL	NULL	30+ year Tech geek, and hobbyist.
561	716	7/15/09 9:47 AM	5000000	4/1/10 4:00 PM	NULL	NULL	NULL
562	43	7/15/09 9:47 AM	Rick McClelland	3/18/10 8:45 AM	London	24	NULL
563	101	7/15/09 9:48 AM	artjom	10/29/09 2:13 PM	Bremen, Germany	4/1/10 6:59 PM	30 <a href="http://www.artjoms.com">artjom's profile picture</a>
564	900	7/15/09 9:49 AM	Shai	4/1/10 6:59 PM	UK	37	<a href="http://www.shai.org">Shai's profile picture</a>
565	121	7/15/09 9:49 AM	Frances Penney	3/5/10 12:17 AM	Kirkland, WA	37	<a href="http://www.francespenney.com">Frances Penney's profile picture</a>

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

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

2.1 p55



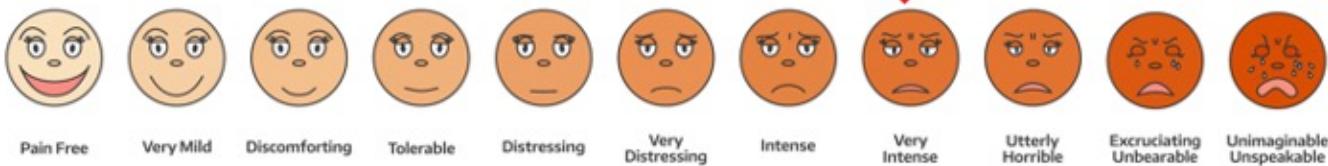
## We don't have great options.

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

Rep	CreationDate	Display Name	LastActivityDate	Location	Age	Profile
1	2/6/08	1/10/09 10:51 PM Just Added	4/1/10 10:35 AM San Jose, CA	29		
2	128	7/12/09 9:38 AM David Gallegos	3/1/10 10:35 AM Corvallis, OR	32		
3	101	7/12/09 10:51 PM Jared Dixon	3/21/10 10:48 PM Morehead, NC	31		
4	967	7/12/09 10:51 PM Joel Spolsky	3/10/10 12:30 PM New York, NY	NULL		
535	366	7/15/09 9:32 AM jbm	2/18/10 9:27 PM Scotland	33		
536	101	7/15/09 9:34 AM second	3/10/10 9:56 PM NULL	NULL		
537	126	7/15/09 9:35 AM stafford	3/25/10 9:10 PM Sweden	38		
538	96	7/15/09 9:35 AM spencero	1/19/10 10:34 PM London	NULL		
539	167	7/15/09 9:37 AM Arturust	3/12/10 9:44 AM NL	27		
404	101	7/15/09 9:38 AM DanGingerman	1/12/10 9:47 PM NULL	NULL		
540	661	7/15/09 9:38 AM David Hines	3/10/10 9:56 PM NULL	21		
542	581	7/15/09 9:38 AM hypersusy	4/1/10 10:49 PM NULL	NULL		
543	493	7/15/09 9:38 AM DavidWhitney	1/22/10 9:11 PM London, UK	25		
544	101	7/15/09 9:39 AM Ben Laan	3/10/10 11:12 PM Adelaide, Australia	29		
545	885	7/15/09 9:40 AM rehamed	4/1/10 3:34 PM NA	39		
546	101	7/15/09 9:41 AM Inter	8/26/09 9:59 AM Bambaray	27		
547	404	7/15/09 9:42 AM Ant	4/1/10 9:24 AM UK	27		
548	501	7/15/09 9:43 AM Jeremy French	1/28/10 9:18 PM UK	33		
549	173	7/15/09 9:42 AM Joshua	2/26/10 9:53 PM Orlando, FL	34		
550	61	7/15/09 9:42 AM hapasasa	3/16/10 4:42 PM NULL	NULL		
551	341	7/15/09 9:43 AM Tom	4/1/10 10:39 AM Worcester, UK	NULL		
552	1	7/15/09 9:44 AM Eudoxus	1/29/09 7:20 PM Maracay, Venezuela	25		
553	101	7/15/09 9:44 AM moff	2/7/10 10:20 PM Sweden	NULL		
554	1158	7/15/09 9:44 AM becomingGuru	4/1/10 8:43 PM Bangalore / Hyderabad	25		
555	1016	7/15/09 9:45 AM MiTFtheFox	4/1/10 10:13 PM Huntington, AL	NULL		
556	101	7/15/09 9:45 AM Russa	1/18/09 6:10 PM UK	NULL		
557	1	7/15/09 9:45 AM Beto	2/13/10 11:01 AM NULL	NULL		
558	953	7/15/09 9:46 AM marco.ragagna	4/1/10 7:41 PM AT	29		
559	63	7/15/09 9:47 AM Aminullah	2/27/10 10:45 PM NULL	30		
560	101	7/15/09 9:47 AM danielg	3/10/10 9:47 PM United Kingdom	30		
561	716	7/15/09 9:47 AM Douglas	4/1/10 9:03 PM NULL	NULL		
562	43	7/15/09 9:47 AM Max Mclelland	3/17/10 9:45 AM London	NULL		
563	101	7/15/09 9:48 AM artiem	1/19/10 9:11 PM Berlin, Germany	24		
564	900	7/15/09 9:49 AM S.H	4/1/10 9:59 PM UK	32		
565	121	7/15/09 9:49 AM Francis Penoy	3/30/10 12:12 AM Keokrad, WA	37		

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



```
/*
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



```
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
```

```

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

```

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 >=

```

Nested Loops (Inner Join)
  SELECT Cost: 0 %
    0.077s
    10000 of
    10000 (100%)
  Top Cost: 0 %
    0.030s
    10000 of
    10000 (100%)
  Sort Cost: 5 %
    0.029s
    10000 of
    10000 (100%)
  Index Seek (NonClustered)
    [Users].[IX_LastAccessDate_Id_Displ...]
    Cost: 0 %
    0.003s
    21547 of
    22777 (94%)

```

Clustered Index Seek (Clustered)

[Users].[PK\_Users\_Id] [u]

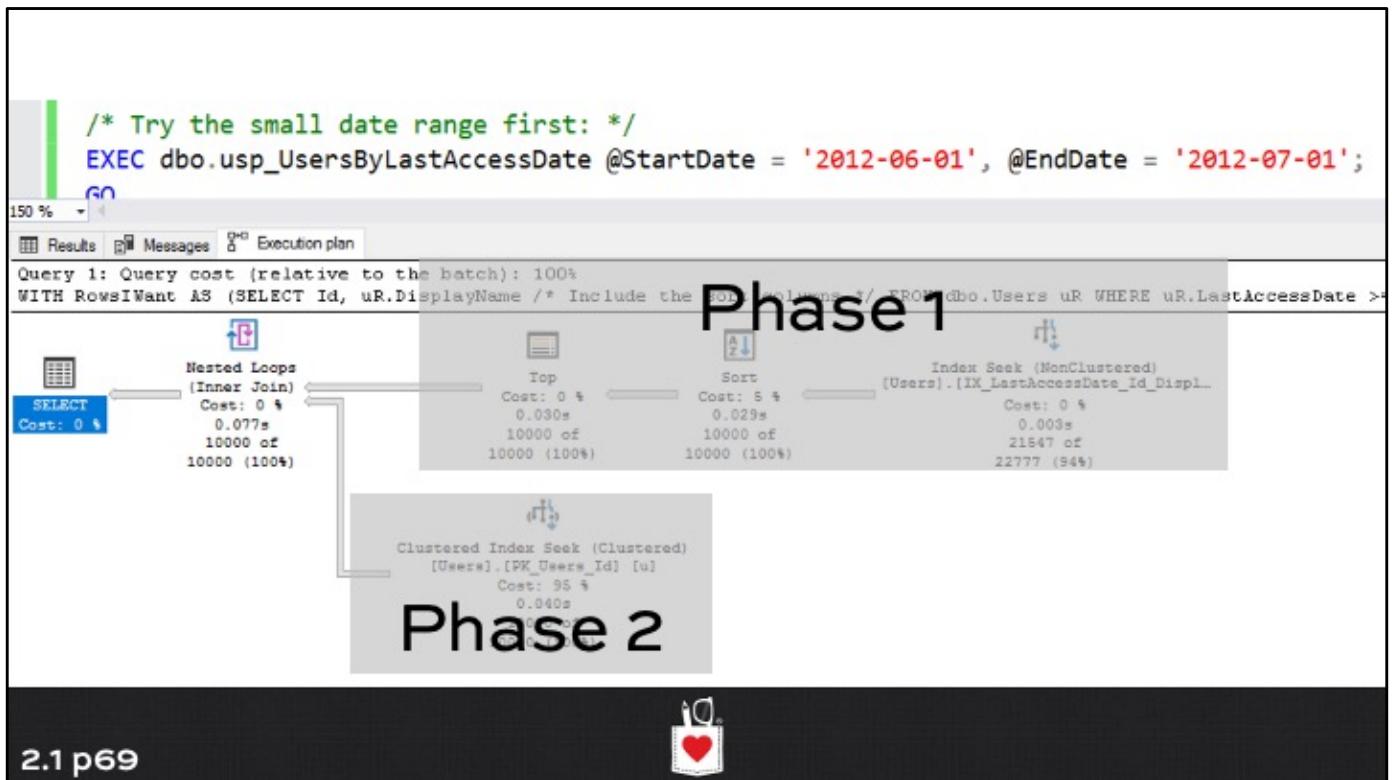
Cost: 95 %

0.040s

10000 of

10000 (100%)

2.1 p68



## Less logical reads than a table scan

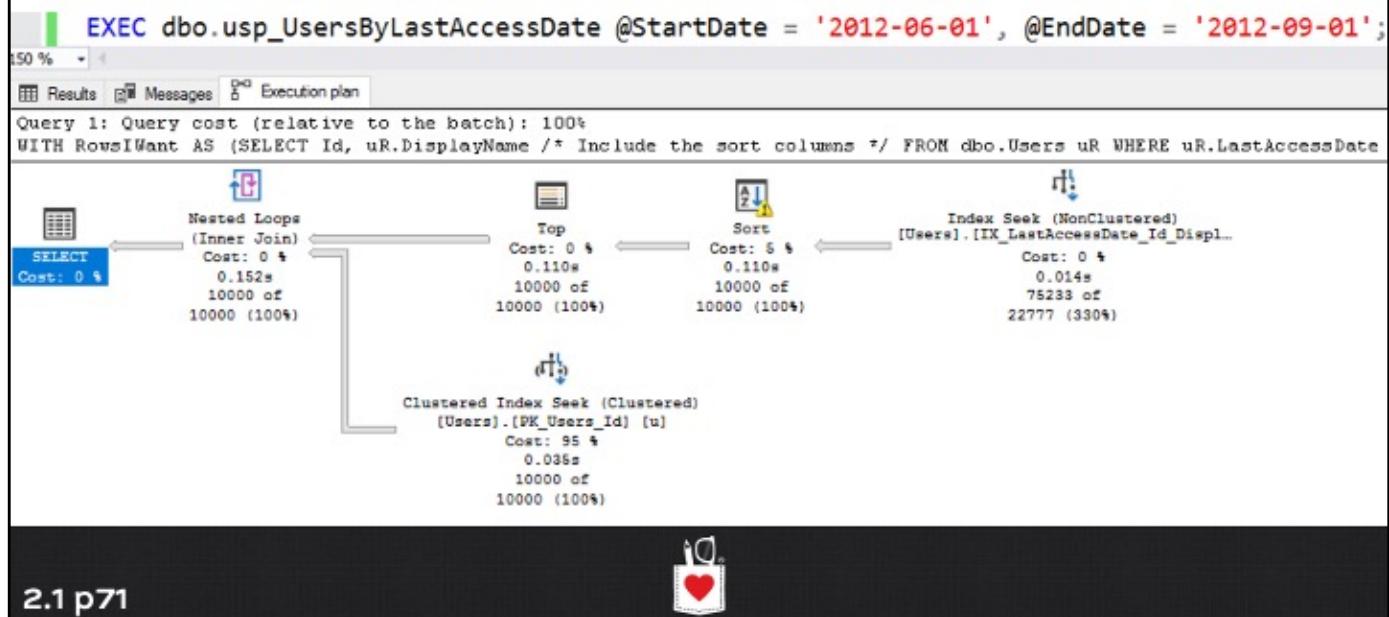
```
/* Try the small date range first: */  
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';  
GO  
Results Messages Execution plan  
  
(10000 rows affected)  
Table 'Users'. Scan count 1, logical reads 30763, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.  
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

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:**



## 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
```

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

```
graph TD
    SELECT[SELECT  
Cost: 0 %] --> NLS(Nested Loops  
(Inner Join))
    NLS --> Top[Top  
Cost: 0 %  
0.151s  
10000 of  
10000 (100%)]
    NLS --> Sort[Sort  
Cost: 15 %  
0.103s  
10000 of  
10000 (100%)]
    Sort --> IXS[Index Seek (NonClustered)  
[Users].[IX_LastAccessDate_Id_Displ...]  
Cost: 1 %  
0.013s  
75233 of  
74623 (100%)]
    IXS --> CIXS[Clustered Index Seek (Clustered)  
[Users].[PK_Users_Id] (u)  
Cost: 84 %  
0.040s  
10000 of  
10000 (100%)]
```

## 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.

2.1 p75



## 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...

2.1 p78



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;

```

150 %

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

```

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

```

```

Nested Loops (Inner Join)
    Cost: 49.4
    0.360ms
    364 of
    186029 (0%)

    SELECT Cost: 0 %

    Clustered Index Scan (Clustered)
        [Users].[PK_Users_Id] (u)
        Cost: 48.4
        0.367ms
        419 of
        44465 (0%)

        Index Seek (NonClustered)
        [Posts].[IX_OwnerUserId_Id] (p)
        Cost: 49.4
        0.001ms
        364 of
        186029 (0%)

```

It's not using the Users index.

## 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.**

```
Clustered Index Scan (Clustered)
[Users].[PK_Users_Id] [u]
Cost: 48 %
0.967s
419 of
44469 (0%)
```

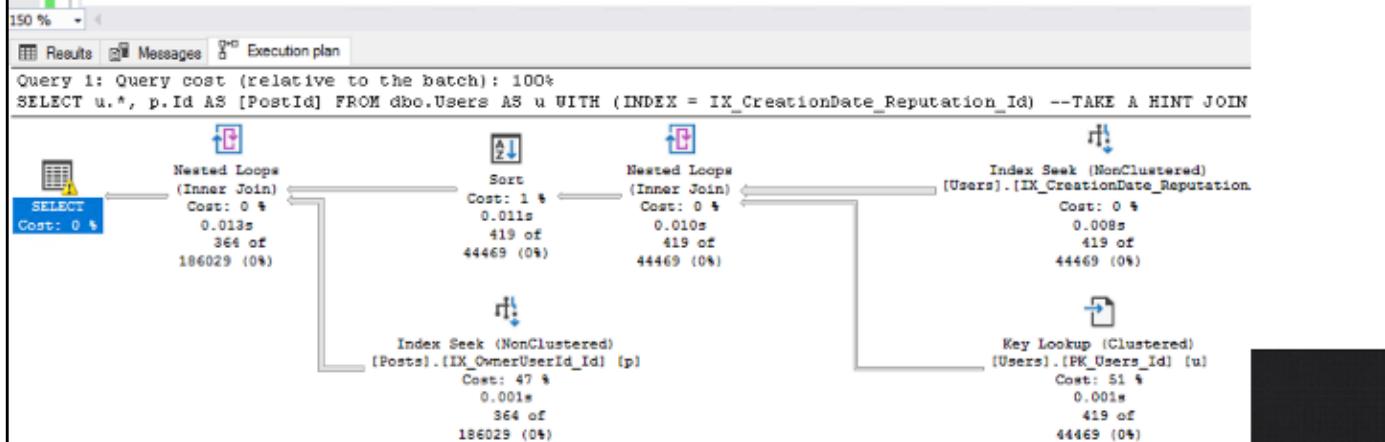
2.1 p81



```

/* 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.

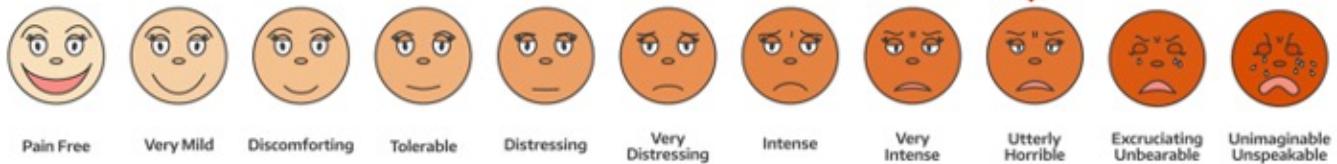
	Logical reads	Estimated subtree cost
Users clustered index scan (SQL Server's choice)	142,648	240
Nonclustered index seek (with hint)	3,104	248

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?

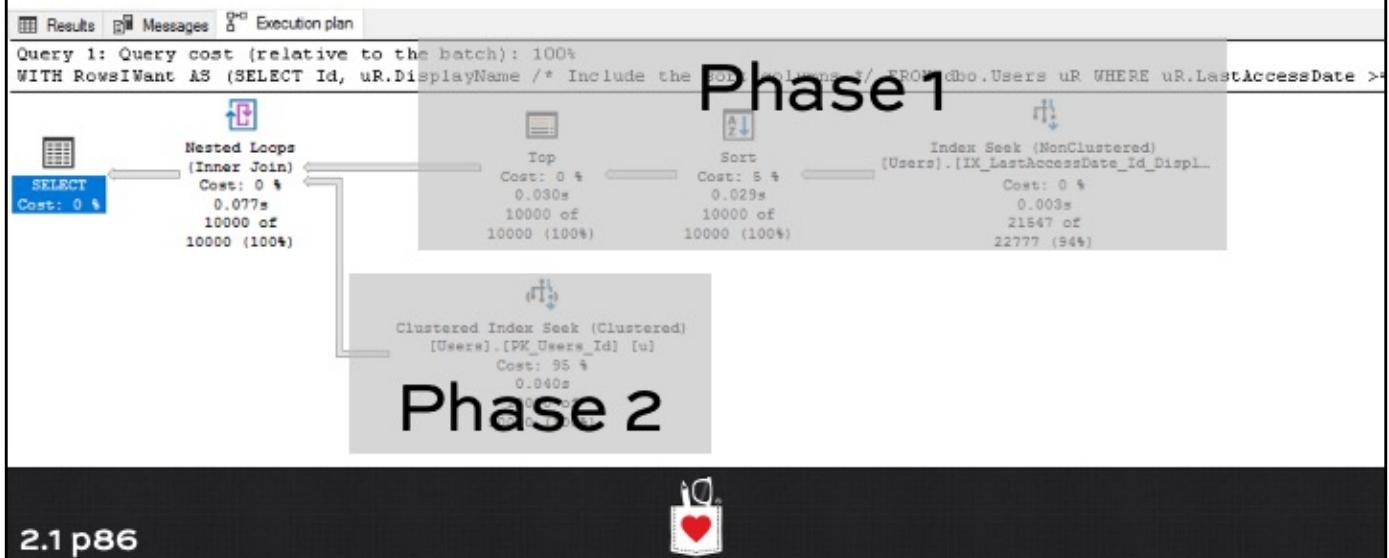
We don't want to keep that hint

- 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

2.1 p85



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



```
/* 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:

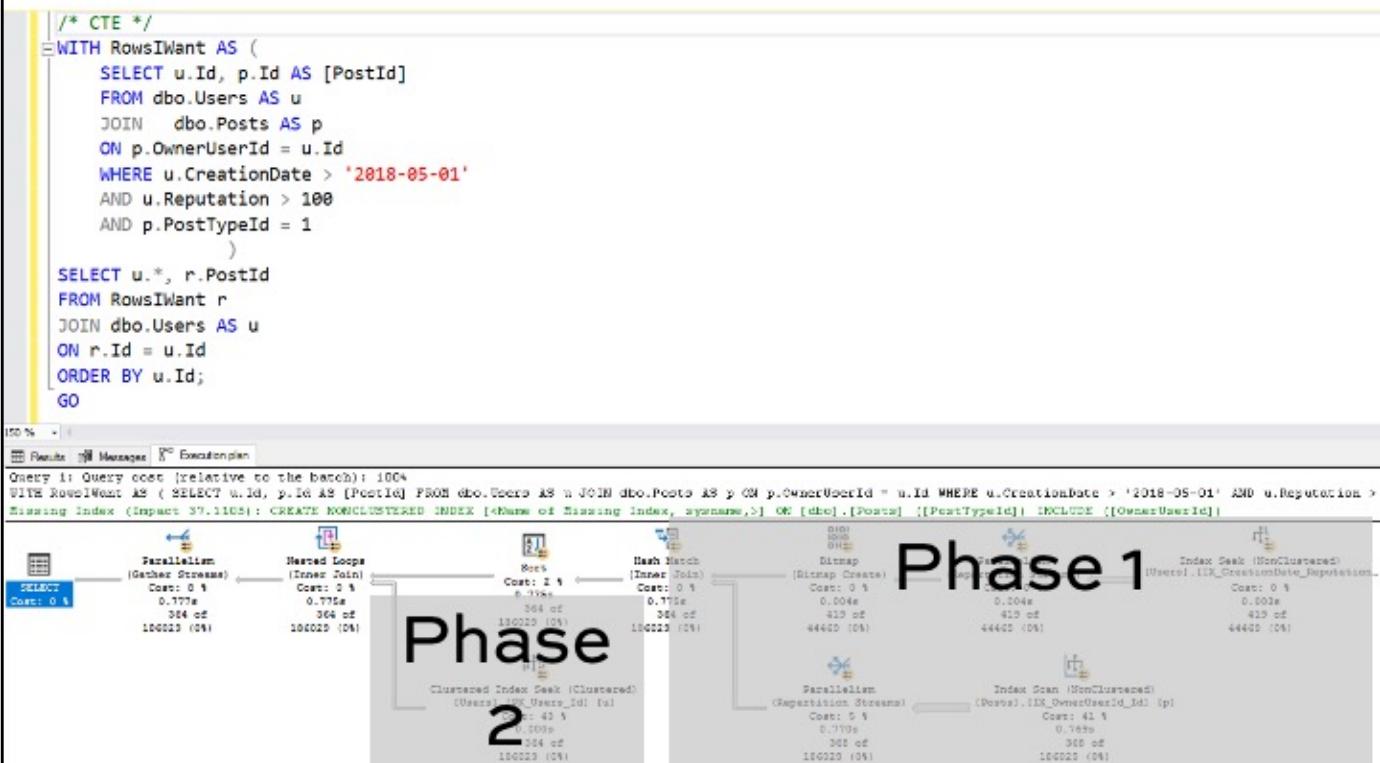
filtering, should only use indexes, only return ID columns to do joins later

## 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.

2.1 p89



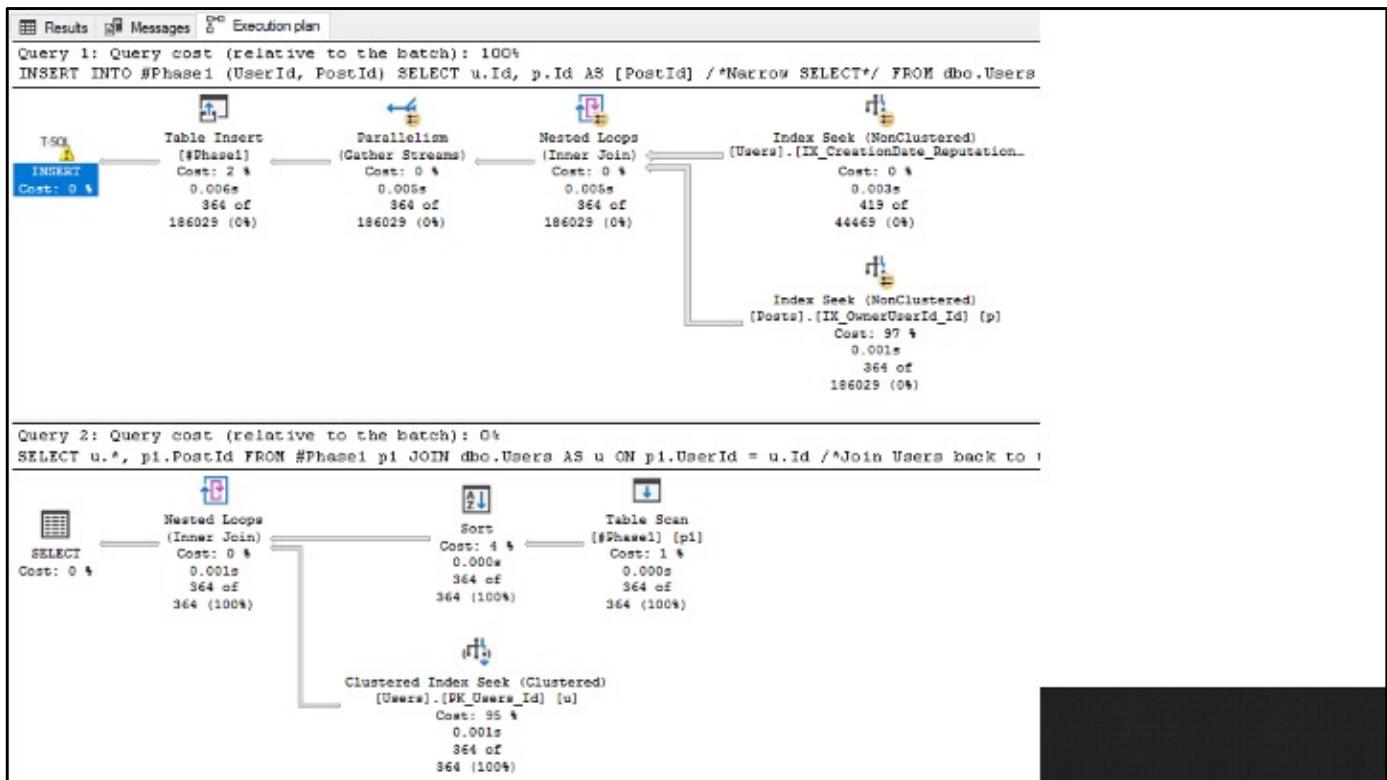
```
CREATE TABLE #Phase1 (UserId INT, PostId INT);
INSERT INTO #Phase1 (UserId, PostId)
    SELECT u.Id, p.Id AS [PostId] /*Narrow SELECT*/
        FROM dbo.Users AS u
        JOIN dbo.Posts AS p ON p.OwnerUserId = u.Id /*Same join*/
        WHERE u.CreationDate > '2018-05-01' /*Same predicates*/
  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

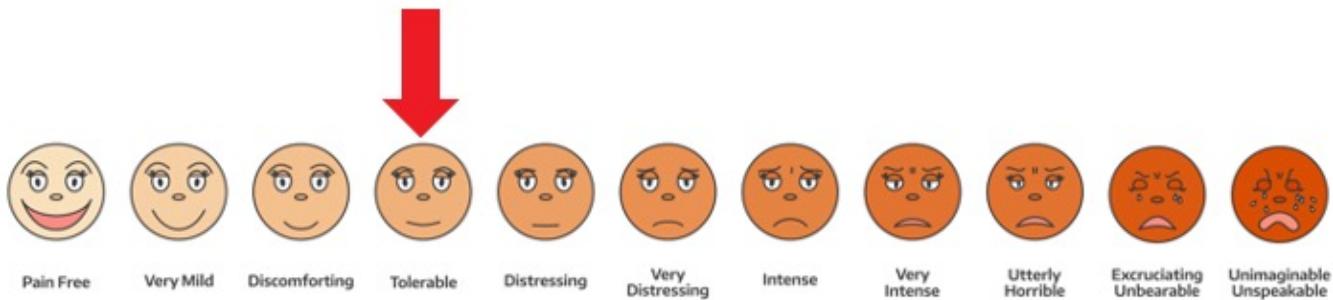
	Logical reads	Estimated subtree cost
Users clustered index scan (SQL Server's choice)	142,648	240
Nonclustered index seek (with hint)	3,104	248
CTE	93,392	
Temp table	3,036	

Ta-dah!

2.1 p92



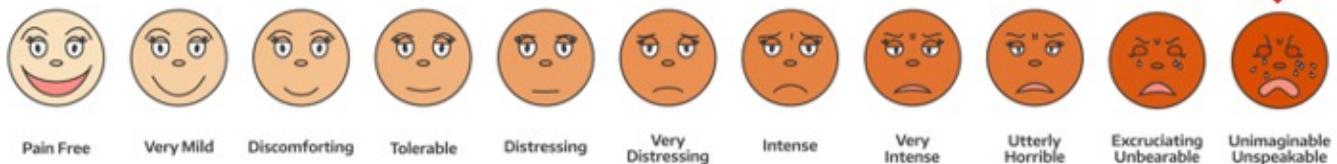
# CTES & TEMP TABLES



2.1 p93



**So what's this  
over here?**



2.1 p94



# **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;
```

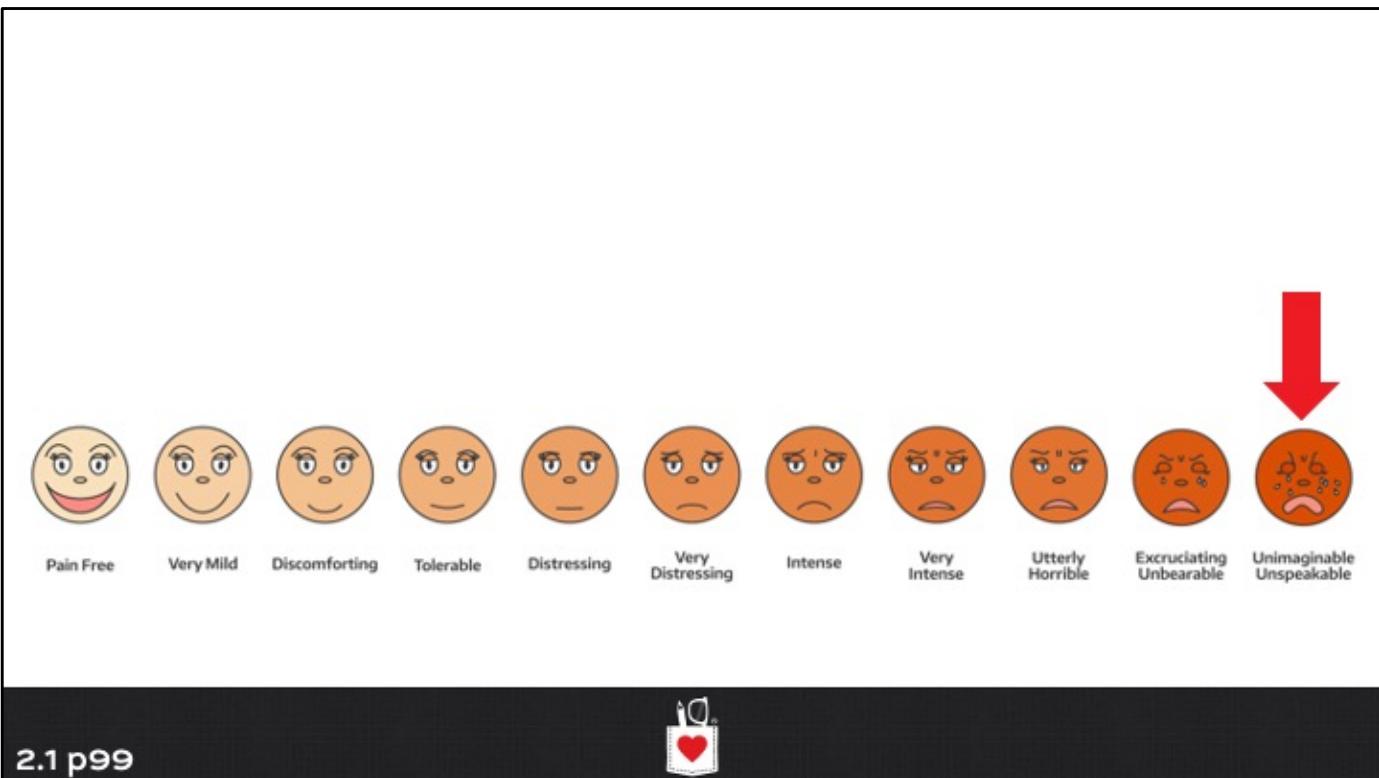
150 %

Results Messages

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

2.1 p98





2.1 p99



# 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.



## 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

