



**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 cost (relative to the batch): 100%

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

```

graph TD
    COND[COND WITH QUERY  
Cost: 0 %] --> CS[Compute Scalar  
Cost: 0 %]
    CS --> NLSJ[Nested Loops  
(Left Semi Join)  
Cost: 0 %  
0.000s  
1 of  
1 (100%)]
    CS --> CS[Constant Scan  
Cost: 0 %  
0.000s  
1 of  
1 (100%)]
    NLSJ --> IS[Index Seek (NonClustered)  
(Users].[IX_LastAccessDate_Id_Displ...  
Cost: 100 %  
0.000s  
1 of  
1 (100%)]

```

2.1 p5

```

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 Executionplan

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

The execution plan consists of two main nodes:

- TOP:** **COND WITH QUERY Cost: 0 %**
- Compute Scalar Cost: 0 %** (preceding the Nested Loops node)
- Nested Loops (Left Semi Join) Cost: 0 %** (preceding the Constant Scan node)
  - Cost: 0 %
    - 0.000s
    - 1 of 1 (100%)
- Constant Scan Cost: 0 %** (preceding the Index Seek node)
  - Cost: 0 %
    - 0.000s
    - 1 of 1 (100%)
- Index Seek (NonClustered) [IX\_LastAccessDate\_Id\_DisplayName\_Age] Cost: 100 %**
  - Cost: 100 %
    - 0.000s
    - 1 of 1 (100%)

Index Seek (NonClustered) [IX\_LastAccessDate\_Id\_DisplayName\_Age] Cost: 100 %

Index Seek (NonClustered) [IX_LastAccessDate_Id_DisplayName_Age] Cost: 100 %	
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	11.0005
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	9 B
Actual Rebinds	0
Actual Rewind	0
Ordered	True
Node ID	3

Object [StackOverflow].[dbo].[Users], [IX\_LastAccessDate\_Id\_DisplayName\_Age]

Seek Predicates

Seek Key([]; Start [StackOverflow].[dbo].[Users].LastAccessDate > Scalar Operator('2018-01-01' 00:00:00))

Query executed successfully.

2.1 p6

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

150 % 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\_Dis... Cost: 100 % 0.000s 1 of 1 (100%)

**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	0.0092831
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 Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

**Object**  
 [StackOverflow].[dbo].[Users],  
 [IX\_LastAccessDate\_Id\_DisplayName\_Age]  
 Create Nonclustered

## 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.

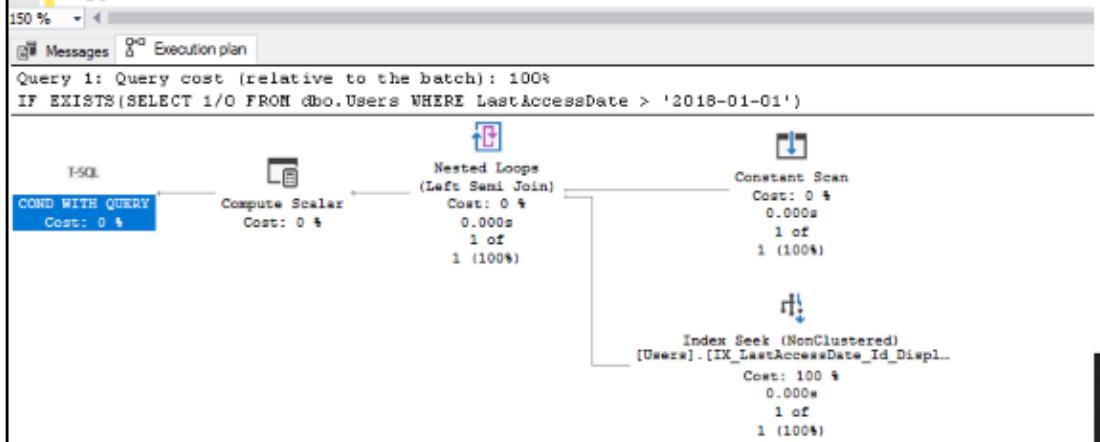


2.1 p8

## SQL Server optimizes the \* away here

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

```
IF EXISTS(SELECT 1/0 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 1/0 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%)

2.1 p9

# 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]=@i

```

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.

2.1 p13

## 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! <div><div> Actual...</div></div>	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

2.1 p14

```

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

```

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 67%

```

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

```

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

Cost: 50 %  
0.000s  
1 of  
1 (100%)

Key Lookup (Clustered)  
[Users].[PK\_Users\_Id]  
Cost: 50 %  
0.000s  
1 of  
1 (100%)

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

```

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

```

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

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

## Compare the plans

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

So what's the overhead of that?

2.1 p15

```
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,

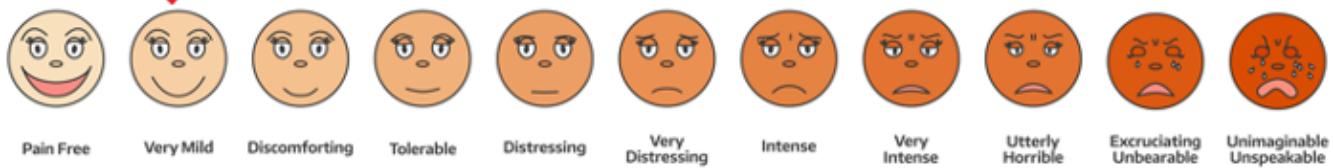
it's not worth building a covering index for that,

nor is it worth asking them to change their query.



2.1 p17

# GETTING 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

```

10 %

Results [! Messages [! Execution plan

ID	RowNum	Age	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate	Location	Reputation	UpVotes	View	WikiedURL	AnswerID	
1	10	10	2009-07-21 19:56:24.0	Sneakers O'Toole	0	NULL	2012-06-19 19:35:43.433	Lakehead, CA	101	0	3570	http://www.youtube.com/watch?v=qhMsJpkuk	0	
2	146743	NULL	2012-06-19 19:35:45.597	Rick Ranap	0	NULL	2012-06-19 19:35:49.597	NULL	1	0	2	NULL	1562237	
3	1487476	NULL	2012-06-19 19:36:57.219	Maven2929	0	NULL	2012-06-19 19:36:57.219	NULL	1	0	9	NULL	1562231	
4	521936	NULL	2010-11-26 19:37:43.570	Schabesoff	0	NULL	2012-06-19 19:37:48.779	Ontario, Canada	39	0	19	http://www.schabesoff.com	247956	
5	446070	NULL	2010-05-13 08:00:11.013	Po...weird	0	NULL	2012-06-19 19:41:45.273	NULL	1	0	1	NULL	200297	
6	1437997	NULL	2012-06-05 17:41:42.113	User1437997	0	NULL	2012-06-19 19:42:39.967	NULL	1	0	9	NULL	1544032	
7	1485985	NULL	2012-06-18 23:41:27.447	User1485985	0	NULL	2012-06-19 19:43:17.993	NULL	13	0	1	NULL	1575934	
8	1003858	NULL	NULL	2011-10-19 18:54:05.197	Andrew Speedwell	0	NULL	2012-06-19 19:45:14.680	NULL	16	0	1	NULL	385157
9	1176680	NULL	2012-01-30 17:41:36.367	Aiden Bard	0	NULL	2012-06-19 19:46:57.862	NULL	1	0	3	NULL	1269875	
10	1394474	NULL	2012-05-14 22:52:16.430	User1394474	0	NULL	2012-06-19 19:48:19.577	NULL	1	0	1	NULL	1486521	
11	1327103	NULL	2012-04-11 21:11:16.103	rggg	0	NULL	2012-06-19 19:49:31.940	NULL	21	0	9	NULL	1390332	
12	1425444	NULL	2012-06-04 15:38:54.553	Talk is Cheap Internet Cafe	0	NULL	2012-06-19 19:49:54.053	NULL	1	0	18	NULL	1540966	
13	1426220	NULL	2012-05-27 14:35:21.977	Garvel	0	NULL	2012-06-19 19:52:27.890	NULL	36	0	9	NULL	1520418	
14	1482300	NULL	2012-06-17 19:55:22.187	User1482300	0	NULL	2012-06-19 19:57:53.347	NULL	71	0	6	NULL	1575864	
15	1156492	NULL	2012-01-18 14:57:38.453	User1156492	0	NULL	2012-06-19 19:58:29.572	NULL	1	0	0	NULL	1161620	
16	1453641	NULL	2012-06-13 12:36:26.113	JohnKur	0	NULL	2012-06-19 19:58:49.725	NULL	21	0	4	NULL	1564676	
17	1487542	NULL	2012-06-19 19:31:06.853	skorffmed	0	NULL	2012-06-19 19:59:01.030	NULL	1	0	3	NULL	1562218	
18	1440252	NULL	2012-06-07 14:45:54.823	User1440252	0	NULL	2012-06-19 19:59:15.317	NULL	1	0	0	NULL	1540852	
19	385867	NULL	2010-05-27 09:32:13.310	mlkman	0	NULL	2012-06-19 20:00:01.067	NULL	21	0	37	http://Meng.com	143036	
20	1464746	NULL	2012-06-18 20:15:56:261	User1464746	0	NULL	2012-06-19 20:04:38.827	NULL	1	0	4	NULL	1579961	
21	1467471	NULL	2012-06-19 20:05:04.742	Marcel Wendler	0	NULL	2012-06-19 20:05:04.742	NULL	1	0	0	NULL	1552464	
22	1287003	NULL	2012-03-22 21:33:55.310	Ehsan	0	NULL	2012-06-19 20:08:32.967	NULL	51	0	1	NULL	1346330	
23	1289828	NULL	2012-03-22 19:35:56:889	Sejal	0	NULL	2012-06-19 20:12:21.777	NULL	61	0	4	http://www.hostinpakistan.com	1346111	
24	1040879	NULL	2011-11-10 22:14:121	essentials	0	NULL	2012-06-19 20:12:31.310	Vermont	26	0	9	NULL	1032659	

Query executed successfully.

(local) (14:0 PTM) SQL2017LAB\Administrator StackOverflow 00:09:00 40 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: 97 %
        0.000s
        43 of
        33 (130%)

```

**Yay! It uses the index.**

What's the cost?

2.1 p23

```

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

	I	AboutMe	Age	CreationDate	DisplayName
1	10	I'm not takin' my sneakers off! -brn-brn- Adul...	NULL	2008-07-31 21:57:06.240	Sneakers O'Toole
2	1467413	NULL	NULL	2012-06-19 19:35:49.597	Rick Remijn
3	1467416	NULL	NULL	2012-06-19 19:36:57.210	Mann2920
4	521536	NULL	NULL	2010-11-26 15:07:53.570	Swhalesoft
5	446070	NULL	NULL	2010-09-13 08:00:11.813	Pro_nerd
6	1437997	NULL	NULL	2012-06-05 17:41:42.113	user1437997
7	1465065	NULL	NULL	2012-06-18 23:49:27.447	user1465065
8	1003858	NULL	NULL	2011-10-19 18:54:09.197	Andrew Speedwalker
9	1178650	NULL	NULL	2012-01-30 17:41:39.367	Adam Baird
10	1394674	NULL	NULL	2012-05-14 20:52:16.630	user1394674
11	1327638	NULL	NULL	2012-04-11 21:11:16.180	rggg

	LastAccessDate	I	Id	DisplayName	Age
1	2012-06-19 19:35:43.433	10	Sneakers O'Toole	NULL	
2	2012-06-19 19:35:49.597	1467413	Rick Remijn	NULL	
3	2012-06-19 19:36:57.210	1467416	Mann2920	NULL	
4	2012-06-19 19:37:48.770	521536	Swhalesoft	NULL	
5	2012-06-19 19:41:45.273	446070	Pro_nerd	NULL	
6	2012-06-19 19:42:39.567	1437997	user1437997	NULL	
7	2012-06-19 19:43:17.660	1465065	user1465065	NULL	
8	2012-06-19 19:45:14.630	1003858	Andrew Speedwalker	NULL	
9	2012-06-19 19:45:57.060	1178650	Adam Baird	NULL	
10	2012-06-19 19:46:19.577	1394674	user1394674	NULL	
11	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



2.1 p24

```

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]>=81 AND [LastaccessDate]

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 \*...

2.1 p25

## 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
```

2.1 p26

```
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';
```

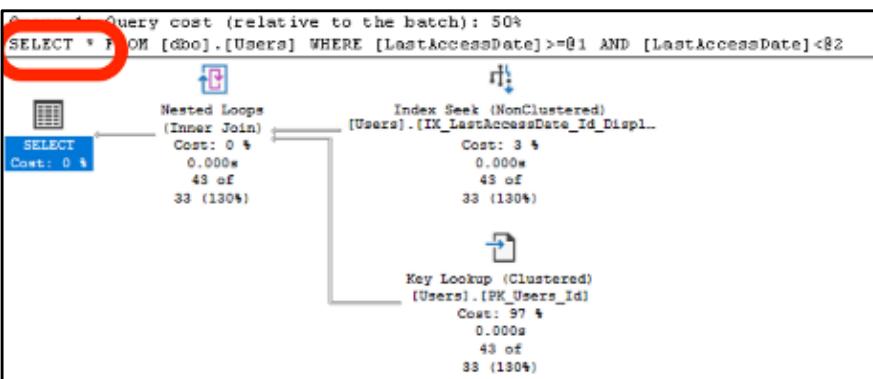
2.1 p28

```
/* 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
```

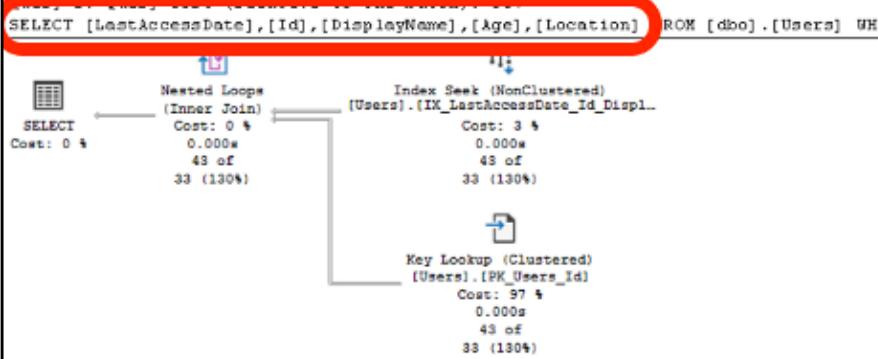


2.1 p29



No help here.

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



2.1 p30

```
/* 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)

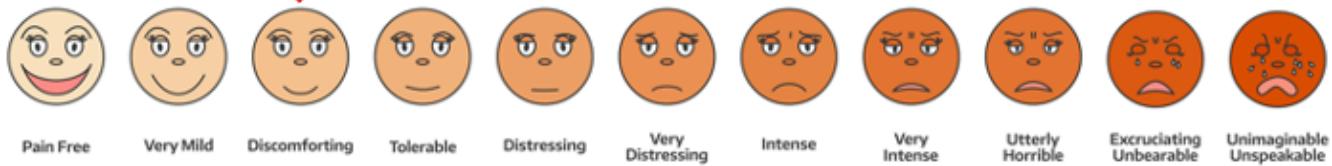
(43 rows affected)
Table 'Users'. Scan count 1, logical reads 144, physical read

(1 row affected)
```

Same reads, too.

2.1 p31

# GETTING 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

```

Results | Messages | Execution plan

	AboutMe	Age	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate	Location	Reputation	UpVotes	Views	WebsiteUrl
1	I'm not takin' my sneakers off! chs-chs. Actually, c...	NULL	2008-03-21 15:20:24.0	SneakersO'Toole	0	NULL	2012-06-19 10:35:03.433	Lakehead, CA	101	0	3670	<a href="http://www.youtube.com/watch?v=...">http://www.youtube.com/watch?v=...</a>
2	94	NULL	2008-08-01 10:20:19.037	Stephen R.	0	NULL	2012-06-22 04:18:33.037	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	2008-08-20 17:44:01.357	GemmaSantini	0	NULL	2012-06-13 17:55:03.637	Portland, OR	296	91	90	
4	2406 Programmer	NULL	2008-08-22 01:54:04.103	cavvym	0	NULL	2012-06-21 01:58:03.563	Massachusetts	276	0	36	<a href="http://cavvym.com">http://cavvym.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	59	<a href="http://www.haven.ca">http://www.haven.ca</a>
6	2930	NULL	2008-08-26 07:40:42.990	Timm	0	NULL	2012-06-12 06:29:01.037	NULL	458	2	87	
7	3351 I reading ...	NULL	2008-08-29 05:23:07.640	David	12	NULL	2012-06-20 10:42:00.073	Indiana	3216	70	119	<a href="http://www.tomlemon.org">http://www.tomlemon.org</a>
8	3533	NULL	2008-08-29 01:57:10.200	David	0	NULL	2012-06-20 17:19:50.753	Richmond, VA	843	20	53	<a href="http://www.davidaweller.com">http://www.davidaweller.com</a>
9	3548 I'm a student at Stony Brook University. Interested i...	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://...">http://...</a>
10	3550 I'm a student at Stony Brook University. Interested i...	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://...">http://...</a>

Id

1	1429571
2	590213
3	1410808
4	1429630
5	1421509
6	1321154
7	1306663
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 Messages Execution plan

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

```

SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>=@1 AND [LastAccessDate]<@2

```

Nested Loops (Inner Join)  
Cost: 0 %  
0.004ms  
21547 of  
22777 (94%)

Index Seek (NonClustered)  
[Users].IX\_LastAccessDate\_Id\_Displ...  
Cost: 0 %  
0.004ms  
21547 of  
22777 (94%)

Key Lookup (Clustered)  
[Users].PK\_Users\_Id  
Cost: 100 %  
0.004ms  
21547 of  
22777 (94%)

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

```

SELECT [Id] FROM [dbo].[Users] WHERE [LastAccessDate]>=@1 AND [LastAccessDate]<@2

```

Index Seek (NonClustered)  
[Users].IX\_LastAccessDate\_Id\_Displ...  
Cost: 100 %  
0.01ms  
21547 of  
22014 (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?

2.1 p35

```
/* 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

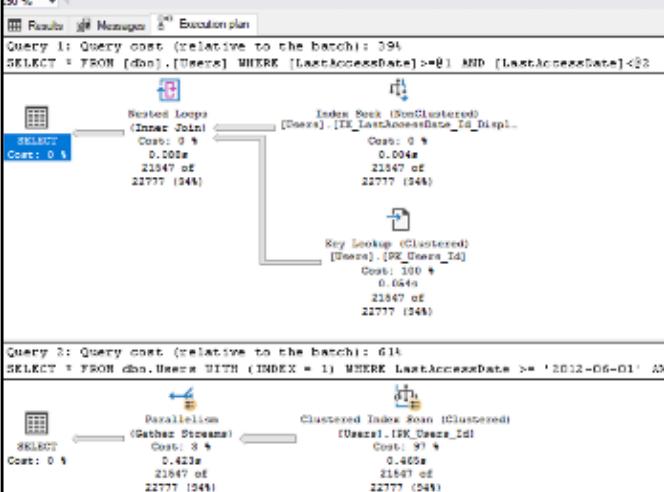
(1 row affected)

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

2.1 p36

```
/* 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.

2.1 p38

# GETTING 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

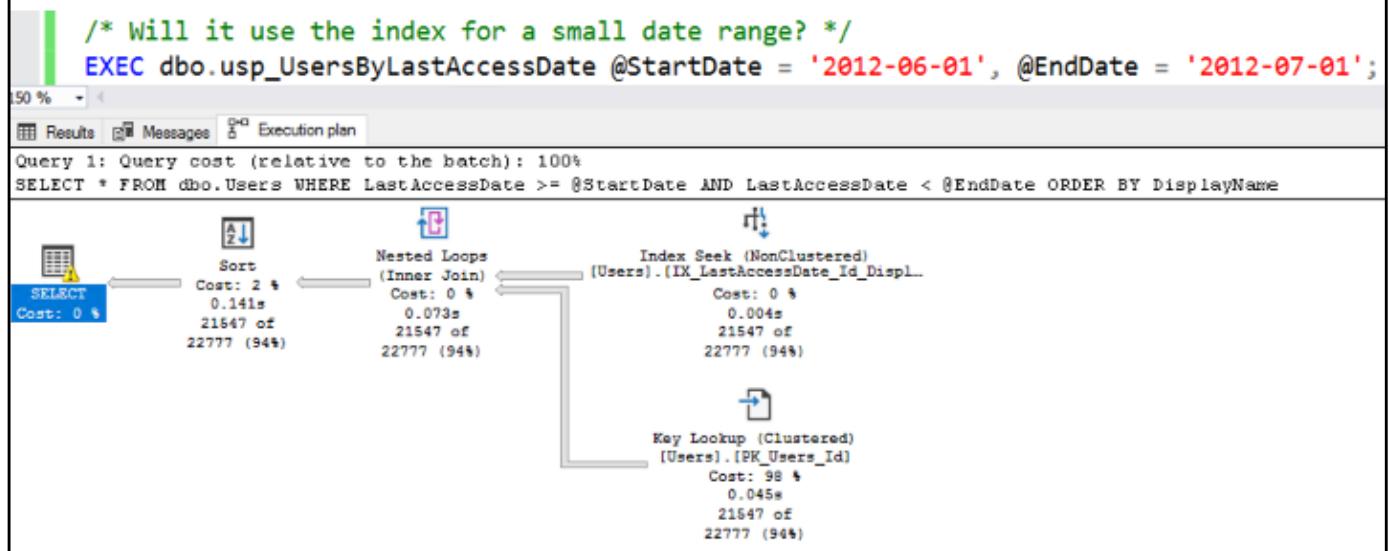
/* Here's our index: */
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
    ON dbo.Users (LastAccessDate, Id, DisplayName, Age);
GO

/* 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

```

2.1 p41

## It uses the nonclustered index...



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

(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
```



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
```

The execution plan diagram illustrates a parallel query execution. It starts with a 'Parallelism (Gather Streams)' operator, which has a cost of 5% and processes 75233 rows in 0.780 seconds. This is followed by a 'Sort' operator with a cost of 3%, processing 75233 rows in 0.545 seconds. Finally, a 'Clustered Index Scan (Clustered)' operator scans the [Users].[PK\_Users\_Id] index with a cost of 92%, processing 75233 rows in 0.482 seconds. The total cost for the entire query is 100%.

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
```

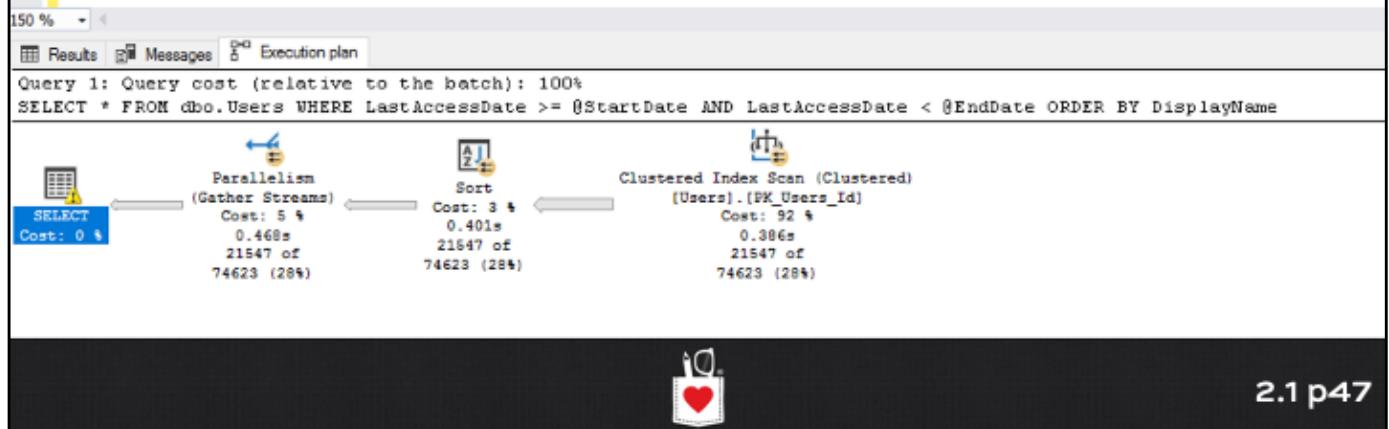


2.1 p46

**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
```



## 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 1, 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:

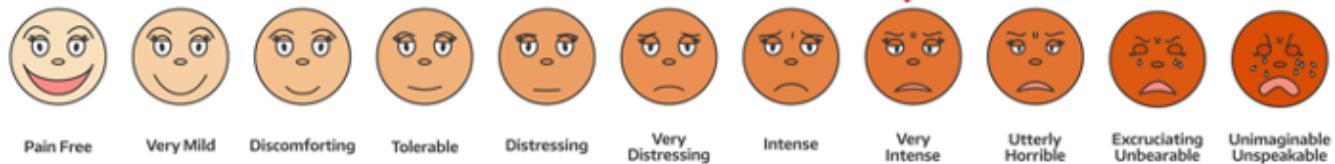
- There’s no covering index
- Sometimes we bring back a lot of data, sometimes a little
- SQL Server caches a plan based on the first set of parameters
- The plan isn’t stable because we keep rebuilding indexes

Learn more: [BrentOzar.com/go/sniff](http://BrentOzar.com/go/sniff)



2.1 p50

# VARIABLE NUMBERS OF 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	Tech geek and hopeless
561	716	7/15/09 9:47 AM	balona	3/16/10 8:45 AM	Londen	NULL	NULL
562	43	7/15/09 9:47 AM	Mike McClelland	10/29/09 2:11 PM	Bremen, Germany	24	NULL
563	101	7/15/09 9:48 AM	arturm	4/1/10 6:59 AM	UK	30	<op>I'm a Software Engineer
564	900	7/15/09 9:49 AM	Giff	3/5/10 12:13 AM	Kirkland, WA	37	<op>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	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	Software Developer</>
559	59	7/13/09 9:46 AM	lakshmanbabu	2/16/10 7:18 AM	AU	39	Net at least I know 4502!
560	83	7/13/09 9:46 AM	gosturing	3/4/10 6:47 PM	United Kingdom	30	<>Tech geek and hopeless
561	716	7/13/09 9:47 AM	balogna	4/1/10 4:00 PM	NULL	NULL	
562	43	7/13/09 9:47 AM	Mike McClelland	3/16/10 8:45 AM	London	NULL	
563	101	7/13/09 9:48 AM	arturm	10/29/09 2:11 PM	Bremen, Germany	24	NULL
564	900	7/13/09 9:49 AM	Giff	4/1/10 6:59 AM	UK	30	<>I'm a Software Enginee...
565	121	7/13/09 9:49 AM	Franci Penov	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	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Centro, CA	39	
2	126	7/12/09 10:51 PM	Geff Dalgas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the StackOver
3	101	7/12/09 10:51 PM	Jared Dixon	3/31/10 3:48 PM	Morganton, NC	31	Developer on the Stack Over
4	767	7/12/09 10:51 PM	Jed Spolsky	3/20/10 2:35 PM	New York, NY	NULL	Co-founder of Stack Overflow
535	386	7/15/09 9:33 AM	lrb	2/18/10 9:27 PM	Scotland	33	Twitter@ http://twitter.us
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	staffen	3/25/10 7:10 PM	Sweden	36	I work on the JRockit VM &
538	98	7/15/09 9:35 AM	cognos	3/19/10 10:54 PM	Lenden	NULL	NULL
539	167	7/15/09 9:37 AM	Arcturus	3/12/10 9:44 AM	NL	27	A Canadian living in London
540	101	7/15/09 9:37 AM	Dansingerman	3/21/09 4:37 PM	NULL	27	I work as a software develo
541	647	7/15/09 9:38 AM	Alexis Hinst	4/1/10 6:45 PM	England	NULL	<>Sufficiently advanced to
542	3921	7/15/09 9:38 AM	hydeking	4/1/10 6:49 PM	UK	21	NULL
543	101	7/15/09 9:38 AM	randmivinty	12/12/09 5:12 PM	London, UK	NULL	NULL
544	101	7/15/09 9:39 AM	Bon Laan	3/28/10 11:12 PM	Adelaide, Australia	23	System.StackOverflowExce
545	985	7/15/09 9:40 AM	redacted	5/1/10 2:34 PM	RnH	27	C#, Delphi, T-SQL, Architect
546	101	7/15/09 9:41 AM	hrec	8/26/09 9:59 AM	Bombay	34	System.StackOverflowExce
547	403	7/15/09 9:42 AM	Ant	4/1/10 6:24 AM	England	27	C# developer since 2005,
548	503	7/15/09 9:41 AM	Jeremy French	3/28/10 5:18 PM	UK	33	Seasoned web developer. W
549	173	7/15/09 9:42 AM	Joshua	2/26/10 5:51 PM	Orlando, FL	28	Software Engineer. Have wif
550	61	7/15/09 9:42 AM	hapasase	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	4	7/15/09 9:44 AM	Redondo	11/23/08 7:30 AM	Monte Carlo, Monaco	25	NULL
553	101	7/15/09 9:44 AM	rossf	2/7/10 10:28 AM	Sweden	NULL	NULL
554	319	7/15/09 9:44 AM	becomingGuru	4/1/10 8:43 PM	Bangalore / Hyderabad	25	curious learner!
555	3016	7/15/09 9:45 AM	MTTheFox	4/1/10 6:13 PM	Huntsville, AL	NULL	<><a href="http://forum
556	101	7/15/09 9:45 AM	Russ_Cam	11/8/09 6:10 PM	UK	NULL	Interested in C#, ASP.NET, J
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	<>2<Software Developer</
559	59	7/15/09 9:46 AM	gabba	2/18/10 7:48 AM	AU	39	Net at least I know 4502
560	83	7/15/09 9:46 AM	gabbing	3/4/10 6:47 PM	United Kingdom	30	<>3<Tech geek and hopeless
561	716	7/15/09 9:47 AM	balona	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	900	7/15/09 9:49 AM	GFaff	4/1/10 6:59 AM	UK	30	<>I'm a Software Enginee
565	121	7/15/09 9:49 AM	Franci Penov	3/5/10 12:13 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

Rep	CreationDate	DirectoryName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Alwood	4/1/10 10:35 AM El Cajon, CA	29	 <a href="http://mp377.us">http://mp377.us</a>
2	126	7/12/09 10:51 PM	Jeffrey Dalgas	3/31/10 12:35 AM Cincinnati, OH	32	Developer on the Stack Overflow
3	101	7/12/09 10:51 PM	Jared Dixon	3/31/10 12:48 PM Morganton, NC	31	Developer on the Stack Overflow
4	767	7/12/09 10:51 PM	Jeffrey Spolsky	3/30/10 12:30 PM New York, NY	NULL	Co-founder of Stack Overflow
535	386	7/15/09 1:33 AM	Jeff Atwood	2/18/10 9:27 PM Scotland	33	Tweatge: <a href="http://twitter.co">http://twitter.co</a>
536	101	7/15/09 1:34 AM	second	3/10/10 9:56 PM NULL	NULL	NULL
537	120	7/15/09 1:35 AM	staffer	1/25/10 11:07 PM Sweden	36	I work on the JRockit JVM :D
538	90	7/15/09 1:35 AM	greenies	1/19/10 10:34 PM London	NULL	A Canadian living in London.
539	167	7/15/09 1:37 AM	Artcurius	3/12/10 9:44 AM NL	27	I work as a software developer.
540	101	7/15/09 1:37 AM	DaveGingerman	2/11/10 9:37 PM NULL	NULL	<>Sufficiently advanced at
					21	NULL
542	521	7/15/09 9:38 AM	Hypersug	4/1/10 10:45 PM NULL	NULL	 <a href="http://www.hyper...">http://www.hyper...</a>
543	490	7/15/09 9:38 AM	DavidWhitney	1/27/10 9:51 PM London, UK	39	mainly C# .NET developer
544	101	7/15/09 9:39 AM	Ben Laan	3/10/10 11:12 PM Adelaide, Australia	34	C#, Delphi, T-SQL : Architect
545	885	7/15/09 9:42 AM	rebelsoft	4/1/10 3:34 PM NA	27	System.StackOverflowException
546	101	7/15/09 9:42 AM	Intrepid	8/26/09 6:59 AM Bambaray	27	C# developer since 2005.
547	404	7/15/09 9:42 AM	Ant	4/1/10 6:24 AM UK	33	Seasoned web developer. W
548	501	7/15/09 9:43 AM	Jeremy French	1/28/10 11:58 PM UK	28	Software Engineer. Have wif
549	173	7/15/09 9:42 AM	Jesusua	2/26/10 3:53 PM Orlando, FL	NULL	NULL
550	61	7/15/09 9:42 AM	hapasasa	3/16/10 4:42 PM NULL	NULL	NULL
551	341	7/15/09 9:47 AM	Tom	4/1/10 10:39 AM Würzburger, UC	NULL	NULL
552	1	7/15/09 9:44 AM	Eodus	11/29/09 7:20 AM Maracala, Venezuela	25	NULL
553	101	7/15/09 9:44 AM	moff	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	census learner!
555	1016	7/15/09 9:45 AM	MFITtheFox	4/1/10 6:11 PM Hunstville, AL	NULL	<> <a href="http://forums">http://forums</a>
556	101	7/15/09 9:45 AM	Russ Java	11/18/09 6:16 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.ragagna	4/1/10 7:41 AM IT	29	<2 Software Developer </
						</> At least, I know what it is!
559	93	7/15/09 9:46 AM	Mr. Webmonkey	2/27/10 11:01 PM NULL	30	<> Tech geek and hopeless
560	83	7/15/09 9:47 AM	Mr. Webmonkey	4/1/10 6:47 PM United Kingdom	NULL	NULL
561	716	7/15/09 9:47 AM	mrhawkins	4/1/10 6:04 PM NULL	NULL	NULL
562	43	7/15/09 9:47 AM	Marc Mclelland	1/16/10 6:45 AM London	NULL	NULL
563	101	7/15/09 9:48 AM	matthw	1/29/10 2:11 PM Berlin, Germany	24	NULL
564	950	7/15/09 9:49 AM	Mat Giff	4/1/10 6:59 AM UK	32	<>I'm a Software Engineer
565	121	7/15/09 9:49 AM	Francis Petrov	3/30/10 12:19 AM Kirkland, WA	37	<>Father 2D/0/Micsoft

**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. Users who insist on `SELECT *`,  
but are flexible on the query
3. Going nuclear and stopping `SELECT *` for good (or evil)



2.1 p59

# **1. Row tuning: using CTE pagination**



## Your goal: avoid the tipping point.

dbo.Users - Clustered Index

ID	Reg	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Centro, CA	39	<a href="http://forum...
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	NUL...	NULL	NULL
558	953	7/15/09 9:46 AM	marco.ragagna	4/1/10 7:41 AM	IT	29	<><2>Software Developer</
559	985	7/15/09 9:46 AM	mississippi	2/2/10 10:47 PM	NUL...	NULL	my first I know how to do it
560	63	7/15/09 9:46 AM	louiseng	3/4/10 6:47 PM	United Kingdom	39	IT Tech geek and hopeless
561	716	7/15/09 9:47 AM	bajrang	4/1/10 4:00 PM	NUL...	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	900	7/15/09 9:49 AM	sluff	4/1/10 6:59 AM	UK	30	<><a href="http://www...
565	121	7/15/09 9:49 AM	Franci Penov	3/5/10 12:13 AM	Kirkland, WA	37	<><a href="http://www...

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

Gooooooooogle >  
1 2 3 4 5 6 7 8 9 10      Next



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
```

2.1 p63

## 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)
  Cost: 0 %
  0.077s
  10000 of
  10000 (100%)

Top
  Cost: 0 %
  0.030s
  10000 of
  10000 (100%)

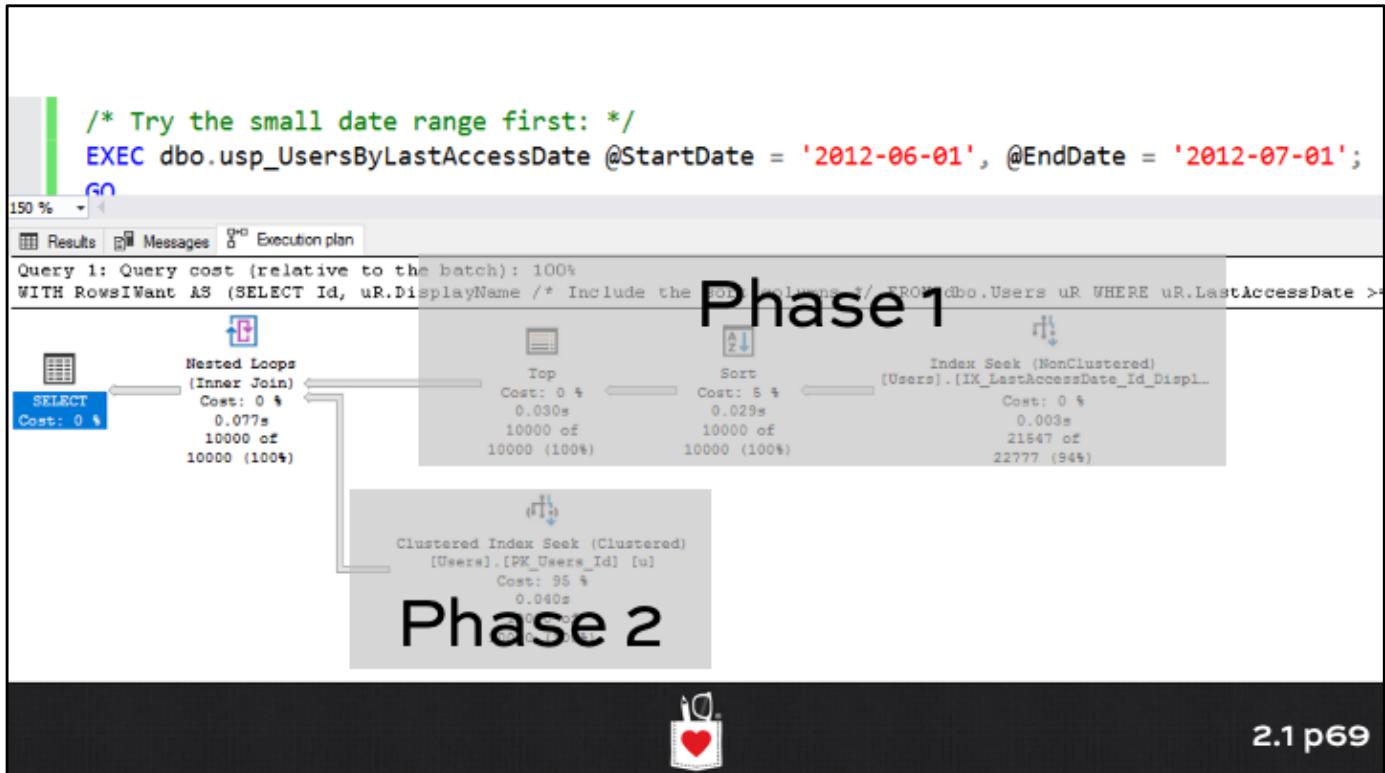
Sort
  Cost: 5 %
  0.029s
  10000 of
  10000 (100%)

Index Seek (NonClustered)
  Cost: 0 %
  0.003s
  21547 of
  22777 (94%)

Clustered Index Seek (Clustered)
  Cost: 95 %
  0.040s
  10000 of
  10000 (100%)

```

2.1 p68



## Less logical reads than a table scan

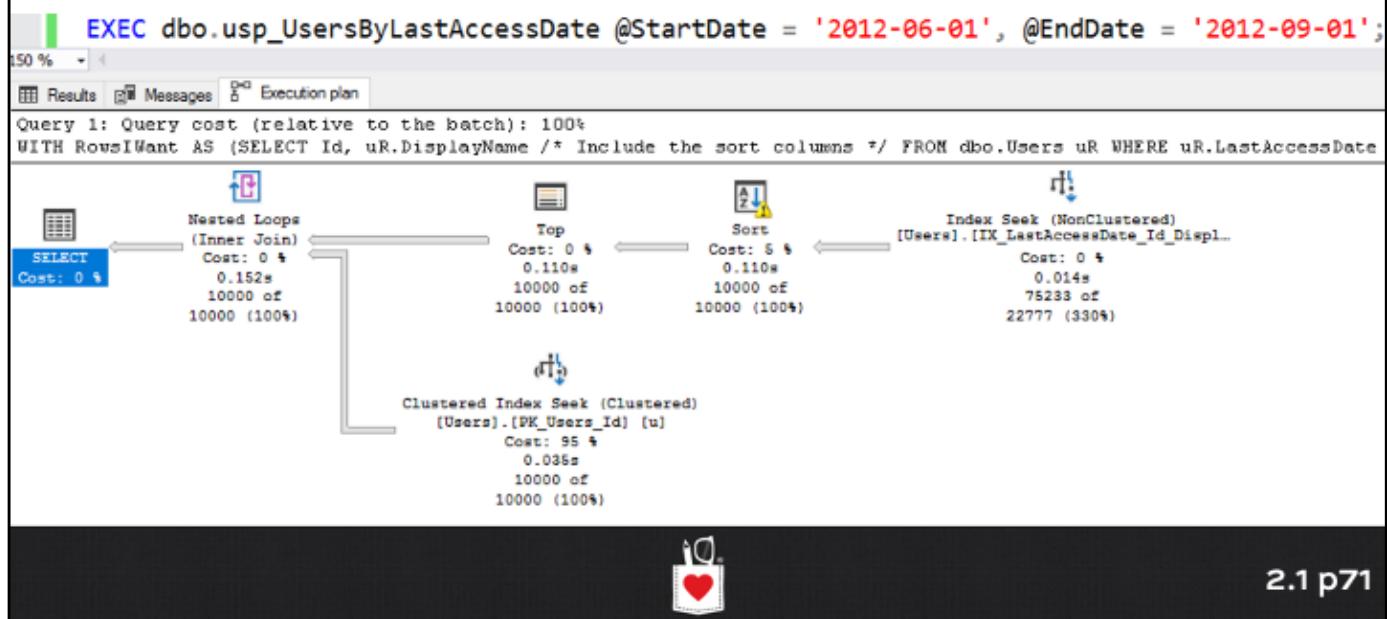
```
/* Try the small date range first: */  
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-07-01';  
GO  
150 %  
Results Messages Execution plan  
  
(10000 rows affected)  
Table 'Users'. Scan count 1, logical reads 30763, physical reads 0, read-ahead reads 0, lob logic  
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logic  
-- -- -- -- --
```

**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!

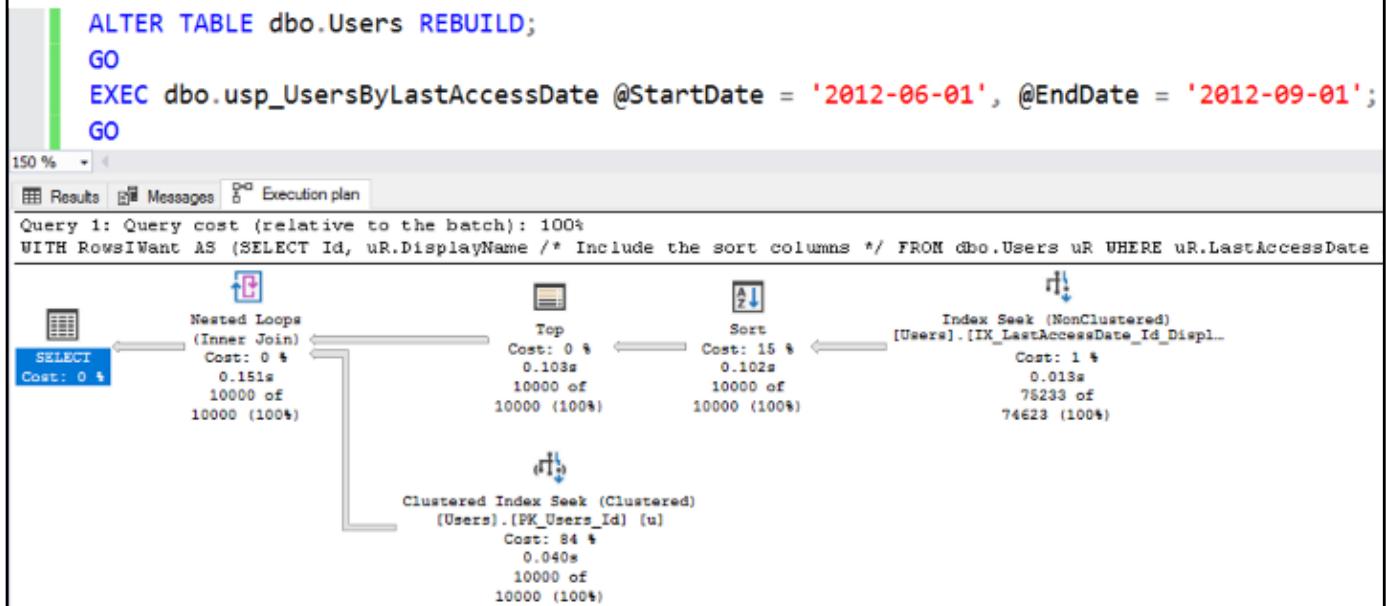
```
EXEC dbo.usp_UsersByLastAccessDate @StartDate = '2012-06-01', @EndDate = '2012-09-01';  
150 %  
Results Messages Execution plan  
(10000 rows affected)  
Table 'Users'. Scan count 1, logical reads 31073, physical reads 0, read-ahead reads 0, lob logi  
Table 'Worktable'. Scan count 0, logical reads 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...



## 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
```

What if my end users still want all of the rows AND all of the columns?



2.1 p76

## **2. Column tuning: doing SELECT \* in phases**



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;
```



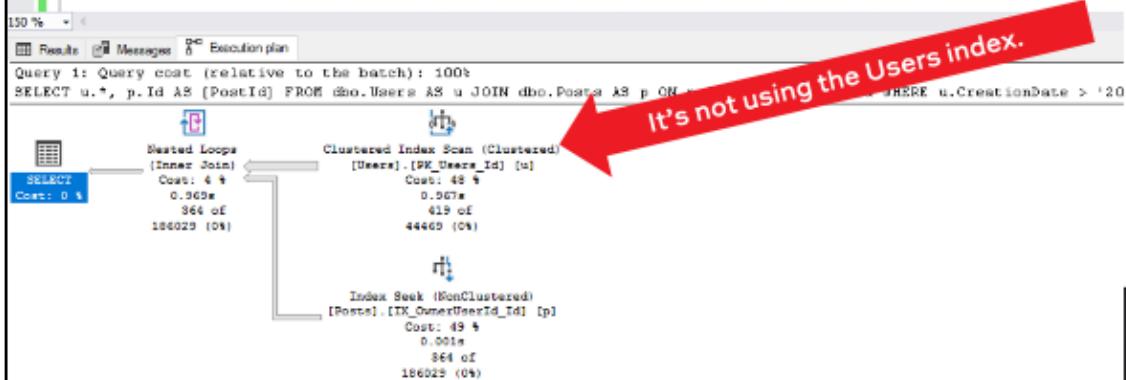
2.1 p79

```

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;

```



2.1 p80

## 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%)

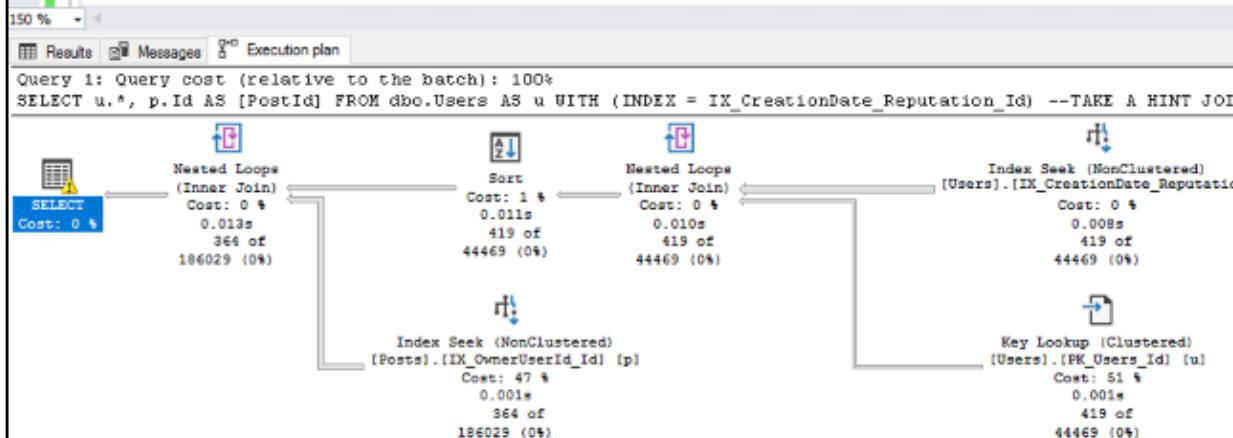


2.1 p81

```

/* 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;

```



**And it's actually faster...**

	<b>Logical reads on Users table</b>
Users clustered index scan (SQL Server's choice)	<b>141,278</b>
Nonclustered index seek (with hint)	<b>1,734</b>

So why is SQL Server doing this crazy plan?



2.1 p83

## 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 p84

# BOGUS ESTIMATES



2.1 p85

## How do we make this smarter?

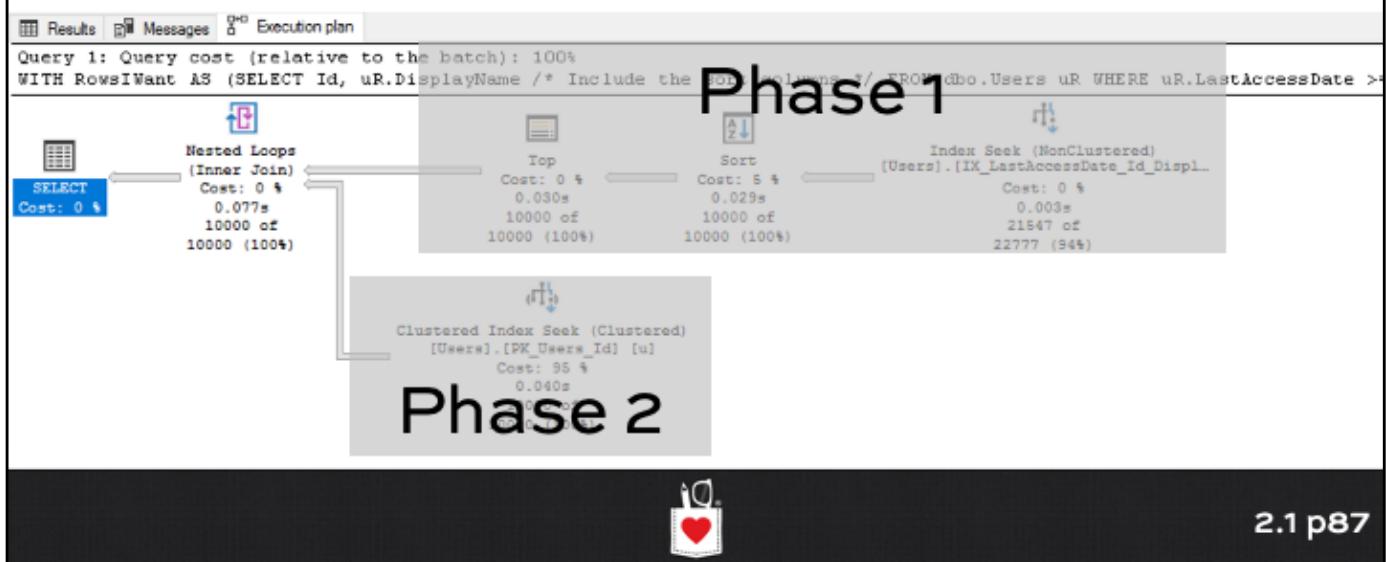
We don't want to keep that hint

- 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 p86

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 p88

```

/* 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

```

Query cost (relative to the batch): 1004

With RowWant 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 )

Missing Index (Impact 37.1105): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Posts] ([PostTypeId]) INCLUDE ([OwnerUserId])

**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 p90

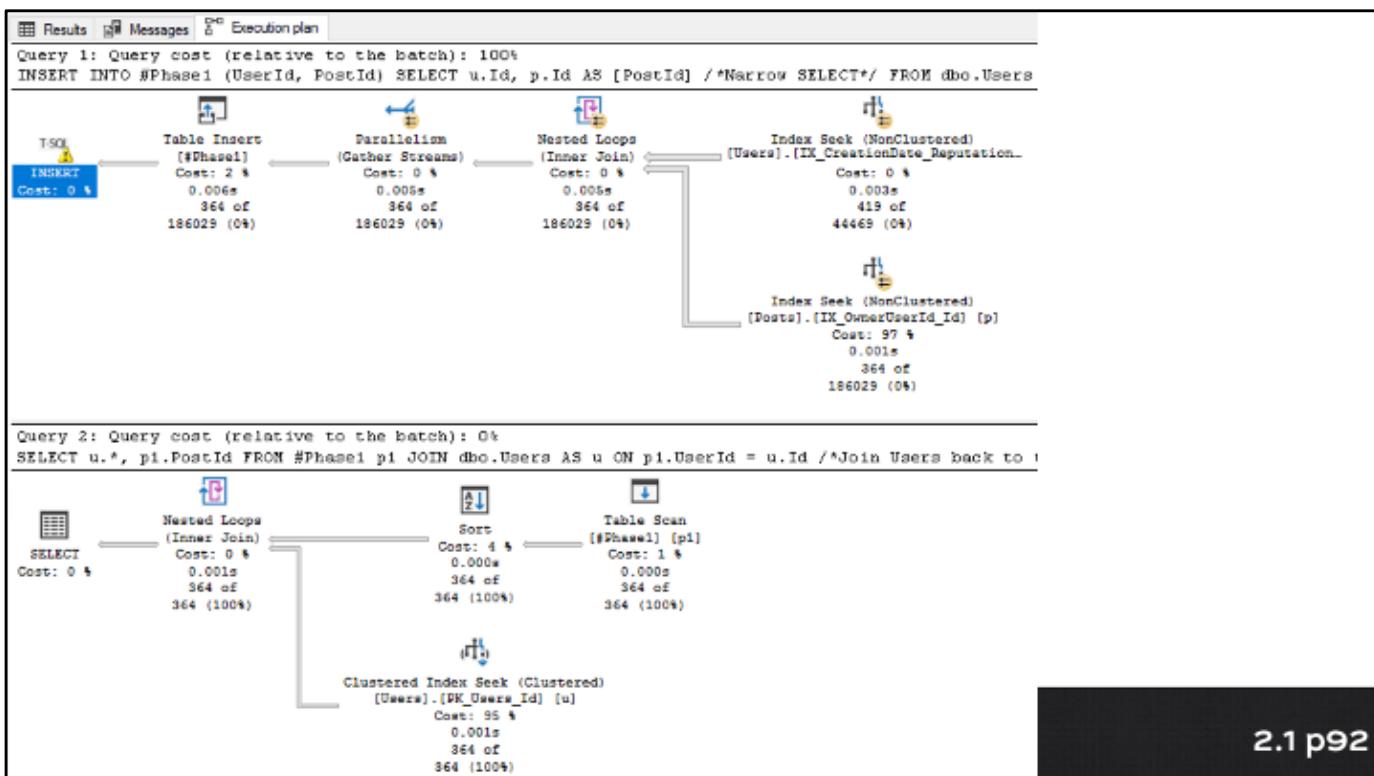
```
CREATE TABLE #Phase1 (UserId INT, PostId INT);
INSERT INTO #Phase1 (UserId, PostId)
    SELECT u.Id, p.Id AS [PostId] /*Narrow SELECT*/
        FROM dbo.Users AS u
        JOIN dbo.Posts AS p ON p.OwnerUserId = u.Id /*Same join*/
        WHERE u.CreationDate > '2018-05-01' /*Same predicates*/
        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 p91



2.1 p92

## That's what I'm talkin' about!

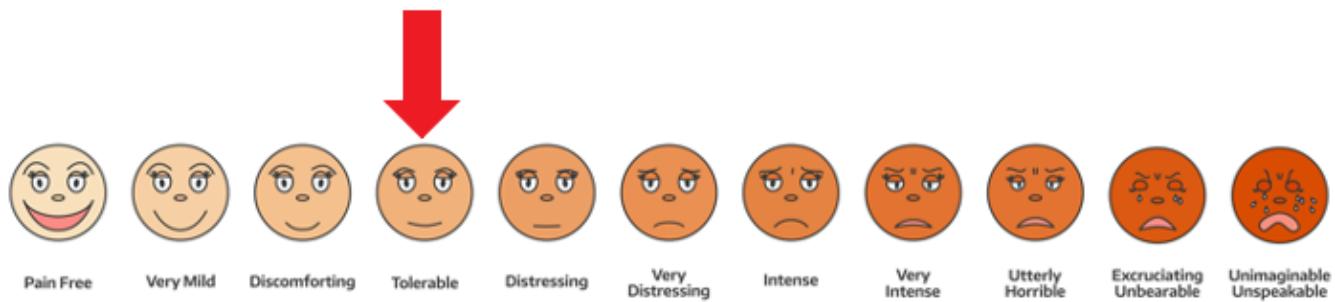
	Logical reads	Estimated subtree cost
Users clustered index scan (SQL Server's choice)	142,648	240
Nonclustered index seek (with hint)	3,104	248
CTE	93,392	
Temp table	3,036	

Ta-dah!



2.1 p93

# CTES & TEMP TABLES



2.1 p94

So what's this  
over here?



Pain Free



Very Mild



Discomforting



Tolerable



Distressing



Very  
Distressing



Intense



Very  
Intense



Utterly  
Horrible



Excruciating  
Unbearable



Unimaginable  
Unspeakable



2.1 p95

# **3. The nuclear option**



2.1 p96

**Say you're tired of being the nice DBA.**



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
```



2.1 p98

**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 p99



2.1 p100

# Recap



2.1 p1O1

## **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 p102

## 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



2.1 p103