BRENT OZAR
UNLIMITED®

Tuning Indexes to Avoid Key Lookups

And understanding when to embrace them.

2.2 p1

Agenda

What a key lookup is

How to mitigate key lookups:

- Select less fields
- Add more fields to the nonclustered index

What a residual predicate is,
and why eliminating them is so important

2.2 p2

From How to Think Like the Engine

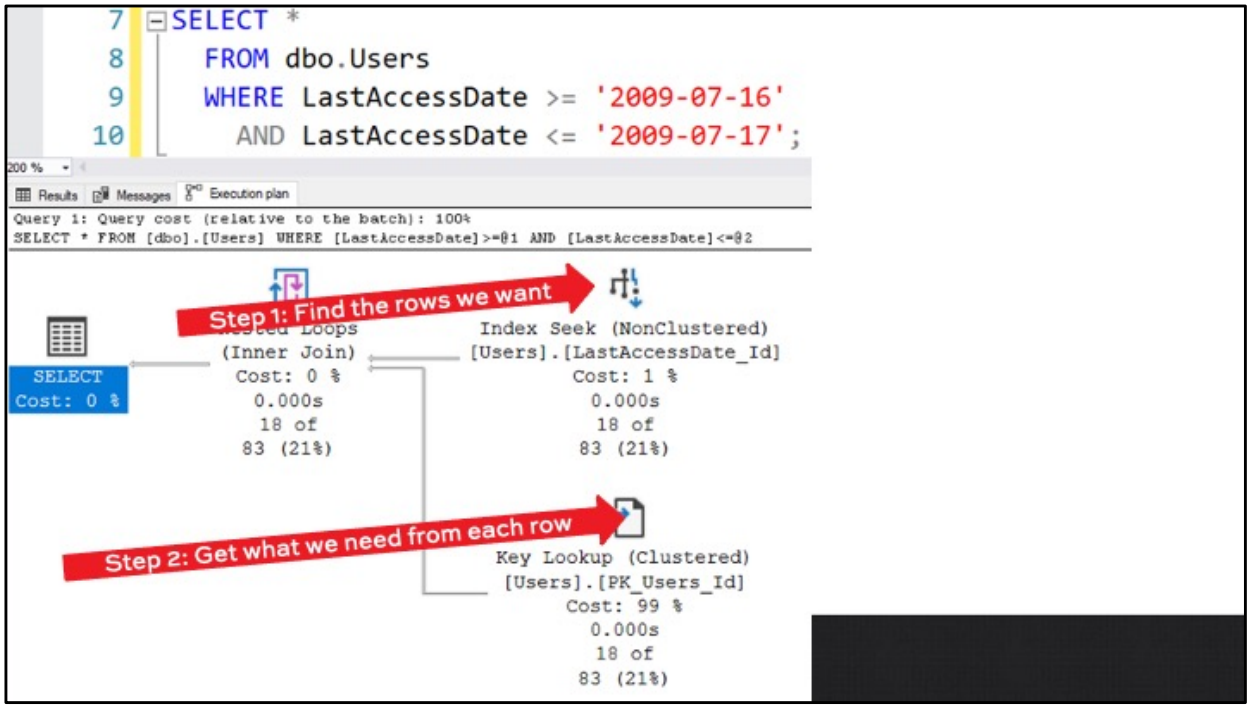
```
CREATE INDEX LastAccessDate_Id
ON dbo.Users(LastAccessDate, Id);
```

LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id
7/31/08 12:00 AM	-1	7/15/09 8:53 AM	445	7/15/09 9:10 PM	200	8/11/09 7:17 PM	39
7/15/09 7:08 AM	22	7/15/09 8:58 AM	457	7/16/09 6:22 AM	678	8/12/09 2:54 PM	943
7/15/09 7:10 AM	33	7/15/09 9:17 AM	501	7/17/09 2:30 AM	131	8/13/09 4:26 PM	364
7/15/09 7:11 AM	40	7/15/09 9:28 AM	524	7/17/09 9:30 AM	297	8/15/09 5:03 PM	910
7/15/09 7:11 AM	41	7/15/09 9:30 AM	527	7/17/09 8:43 PM	998	8/17/09 8:42 AM	202
7/15/09 7:11 AM	44	7/15/09 9:58 AM	587	7/18/09 12:38 PM	394	8/17/09 10:11 AM	628
7/15/09 7:12 AM	52	7/15/09 10:00 AM	594	7/18/09 2:15 PM	924	8/17/09 10:33 AM	157
7/15/09 7:13 AM	64	7/15/09 10:02 AM	597	7/19/09 10:26 PM	336	8/17/09 4:24 PM	1006
7/15/09 7:13 AM	65	7/15/09 10:21 AM	618	7/20/09 1:06 PM	849	8/18/09 8:06 AM	511
7/15/09 7:14 AM	68	7/15/09 10:25 AM	347	7/21/09 7:22 AM	881	8/18/09 9:00 AM	262

Build the query plan for this

```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2009-07-16'
    AND LastAccessDate <= '2009-07-17'
```

LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id
7/31/08 12:00 AM	-1	7/15/09 8:53 AM	445	7/15/09 9:10 PM	200	8/11/09 7:17 PM	39
7/15/09 7:08 AM	22	7/15/09 8:58 AM	457	7/16/09 6:22 AM	678	8/12/09 2:54 PM	943
7/15/09 7:10 AM	33	7/15/09 9:17 AM	501	7/17/09 2:30 AM	131	8/13/09 4:26 PM	364
7/15/09 7:11 AM	40	7/15/09 9:28 AM	524	7/17/09 9:30 AM	297	8/15/09 5:03 PM	910
7/15/09 7:11 AM	41	7/15/09 9:30 AM	527	7/17/09 8:43 PM	998	8/17/09 8:42 AM	202
7/15/09 7:11 AM	44	7/15/09 9:58 AM	587	7/18/09 12:38 PM	394	8/17/09 10:11 AM	628
7/15/09 7:12 AM	52	7/15/09 10:00 AM	594	7/18/09 2:15 PM	924	8/17/09 10:33 AM	157
7/15/09 7:13 AM	64	7/15/09 10:02 AM	597	7/19/09 10:26 PM	336	8/17/09 4:24 PM	1006
7/15/09 7:13 AM	65	7/15/09 10:21 AM	618	7/20/09 1:06 PM	849	8/18/09 8:06 AM	511
7/15/09 7:14 AM	68	7/15/09 10:25 AM	347	7/21/09 7:22 AM	881	8/18/09 9:00 AM	262

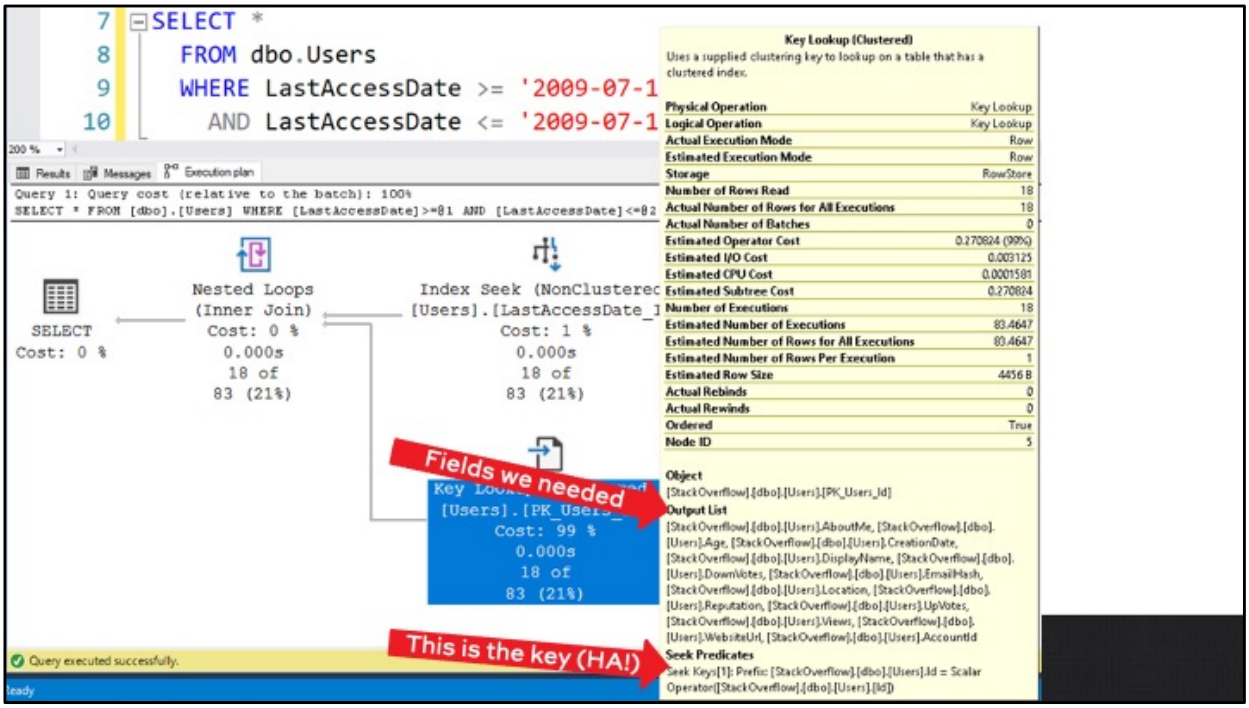


Step 1's reads are sequential.

Step 1: seek on the nonclustered index to the LastAccessDate we want.

In this step, our reads are sequential: we jump to one point on the index and read the pages in order.

LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id
7/31/08 12:00 AM	-1	7/15/09 8:53 AM	445	7/15/09 9:10 PM	200	8/11/09 7:17 PM	39
7/15/09 7:08 AM	22	7/15/09 8:58 AM	457	7/16/09 6:22 AM	678	8/12/09 2:54 PM	943
7/15/09 7:10 AM	33	7/15/09 9:17 AM	501	7/17/09 2:30 AM	131	8/13/09 4:26 PM	364
7/15/09 7:11 AM	40	7/15/09 9:28 AM	524	7/17/09 9:30 AM	297	8/15/09 5:03 PM	910
7/15/09 7:11 AM	41	7/15/09 9:30 AM	527	7/17/09 8:43 PM	998	8/17/09 8:42 AM	202
7/15/09 7:11 AM	44	7/15/09 9:58 AM	587	7/18/09 12:38 PM	394	8/17/09 10:11 AM	628
7/15/09 7:12 AM	52	7/15/09 10:00 AM	594	7/18/09 2:15 PM	924	8/17/09 10:33 AM	157
7/15/09 7:13 AM	64	7/15/09 10:02 AM	597	7/19/09 10:26 PM	336	8/17/09 4:24 PM	1006
7/15/09 7:13 AM	65	7/15/09 10:21 AM	618	7/20/09 1:06 PM	849	8/18/09 8:06 AM	511
7/15/09 7:14 AM	68	7/15/09 10:25 AM	347	7/21/09 7:22 AM	881	8/18/09 9:00 AM	262



Key lookups = more logical reads

On a heap = 1-2 more logical reads per row we need

On a clustered index = 3 or more logical reads

```
7   SELECT *
8     FROM dbo.Users
9     WHERE LastAccessDate >= '2009-07-16'
10      AND LastAccessDate <= '2009-07-17';
```

200 %

Results | Messages | Execution plan

```
(18 rows affected)
Table 'Users'. Scan count 1, logical reads 64, physical reads 0,
```

2.2 p8

Step 2's reads are random.

Step 1 gave us a list of User IDs that accessed the system on 2009-07-16.

But those IDs are all over the place: they're effectively random because people don't access the system at the same time, in order.

The rows we want start here

LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id
7/31/08 12:00 AM		[illegible]	445	7/15/09 9:10 PM	200	8/11/09 7:17 PM	39
7/15/09 7:08 AM	22	[illegible]		7/16/09 6:22 AM	678	8/12/09 2:54 PM	943
7/15/09 7:10 AM	33	7/15/09 9:17 AM	501	7/17/09 2:30 AM	131	8/13/09 4:26 PM	364
7/15/09 7:11 AM	40	7/15/09 9:28 AM	524	7/17/09 9:30 AM	297	8/15/09 5:03 PM	910
7/15/09 7:11 AM	41	7/15/09 9:30 AM	527	7/17/09 8:43 PM	998	8/17/09 8:42 AM	202
7/15/09 7:11 AM	44	7/15/09 9:58 AM	587	7/18/09 12:38 PM	394	8/17/09 10:11 AM	628
7/15/09 7:12 AM	52	7/15/09 10:00 AM	594	7/18/09 2:15 PM	924	8/17/09 10:33 AM	157
7/15/09 7:13 AM	64	7/15/09 10:02 AM	597	7/19/09 10:26 PM	336	8/17/09 4:24 PM	1006
7/15/09 7:13 AM	65	7/15/09 10:21 AM	618	7/20/09 1:06 PM	849	8/18/09 8:06 AM	511
7/15/09 7:14 AM	68	7/15/09 10:25 AM	347	7/21/09 7:22 AM	881	8/18/09 9:00 AM	262

Step 2 hits the clustered index.

We have to look up the clustered index rows for Ids 678, 131, 297, 998, 394, 924, etc...

If we're reading pages from memory,
it doesn't matter what order they're in.
(RAM = Random Access Memory)

If we're reading pages from disk,
it does matter: random disk access is slower.

2.2 p10

You've read old advice.

SQL Server wasn't the only thing built in the 1990s.

So were a lot of blog posts and books.

Back then, key lookups really were expensive because our storage was terrible.

2.2 p11

SQL Server was built on rust.

Query plan costs assume index seeks are cheap.

They assume key lookups are expensive because:

- Memory was really expensive in the 1990s
- We had magnetic hard drives, which suck at random access

They assume table scans are not that expensive because they're sequential scans. (Because they also assume we obsess over defragmentation.)

2.2 p14

Between 2009-07-16 and 7-17

```
7   SELECT *
8     FROM dbo.Users
9     WHERE LastAccessDate >= '2009-07-16'
10      AND LastAccessDate <= '2009-07-17';
```

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>=@1 AND [LastAccessDate]<=@2

SELECT
Cost: 0 %

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

Index Seek (NonClustered)
[Users].[LastAccessDate_Id]
Cost: 1 %
0.000s
18 of
83 (21%)

SELECT

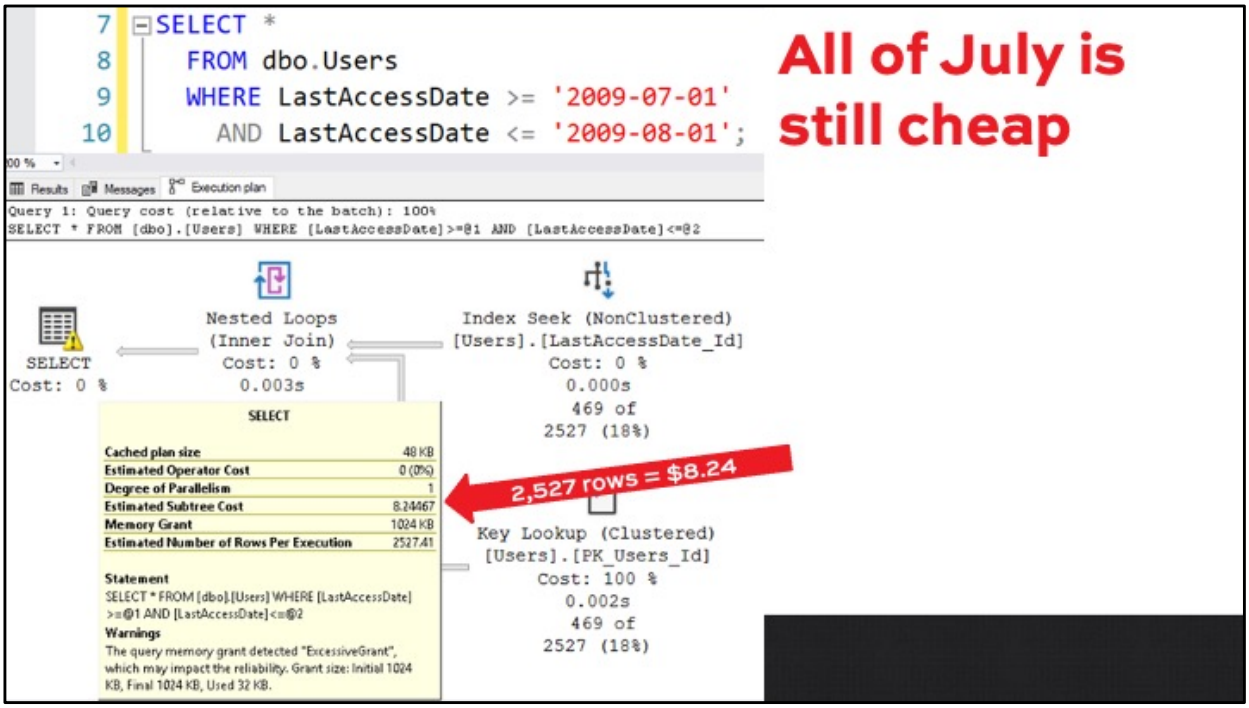
Cached plan size	48 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	0.274546
Memory Grant	1024 KB
Estimated Number of Rows Per Execution	@3.4647

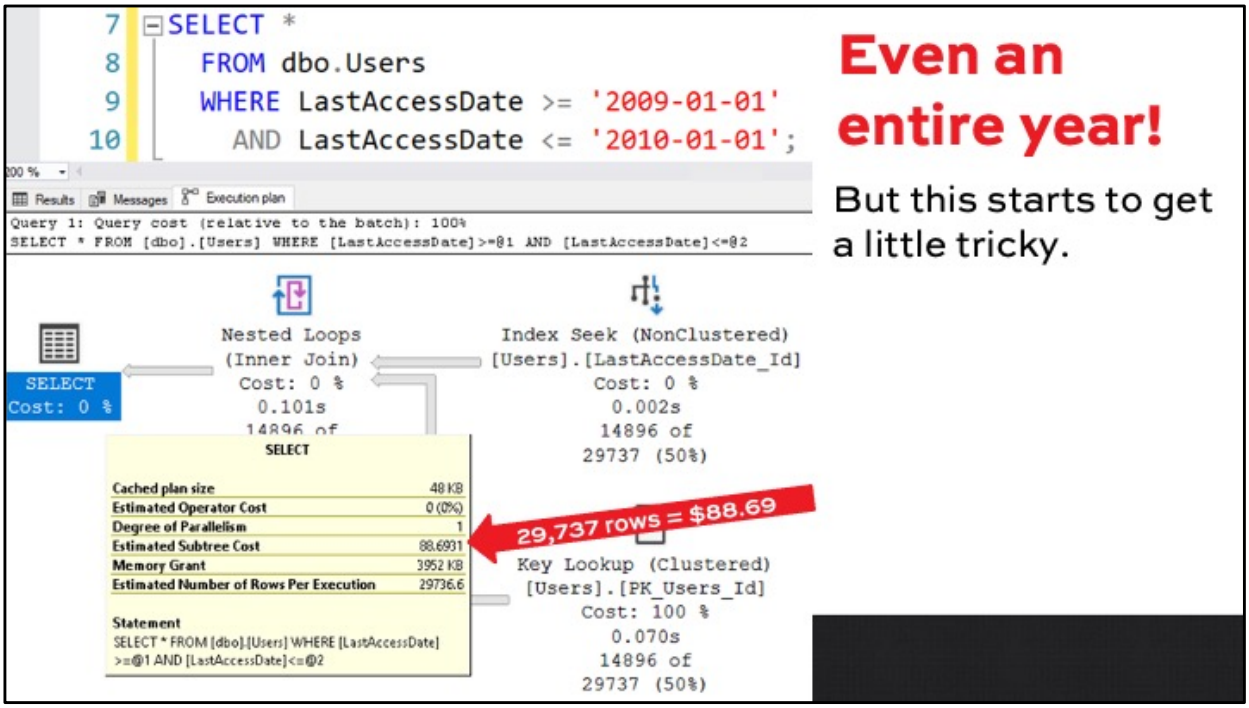
Statement
SELECT * FROM [dbo].[Users] WHERE [LastAccessDate] >=@1 AND [LastAccessDate]<=@2

83 rows are cheap

Key Lookup (Clustered)
[Users].[PK_Users_Id]
Cost: 99 %
0.000s
18 of
83 (21%)

SQL Server estimates only 83 rows will match, and the estimated subtree cost is 0.27.

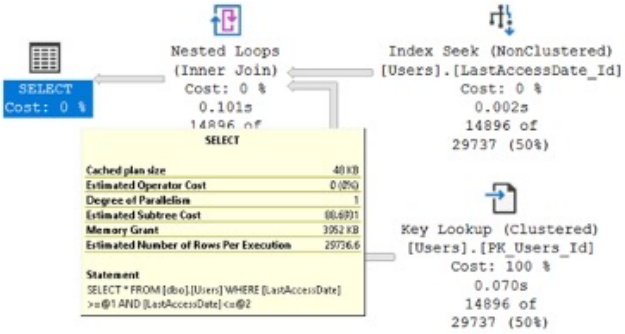




The plan is based on estimates.

SQL guessed 29,737 rows will match our date filter.

It was kinda far off (but not 10x off.)



2.2 p18

That seems like a lot of pages...

14,896 key lookups at 3 reads apiece add up fast:

```
7  SELECT *
8    FROM dbo.Users
9    WHERE LastAccessDate >= '2009-01-01'
10     AND LastAccessDate <= '2010-01-01';
```

00 %

Results | Messages | Execution plan

```
(14896 rows affected)
Table 'Users'. Scan count 1, logical reads 45665,
```

2.2 p19

But a table scan is still worse.

```
7  SELECT *
8    FROM dbo.Users WITH (INDEX = 1)
9    WHERE LastAccessDate >= '2009-01-01'
10     AND LastAccessDate <= '2010-01-01';
```

Hint

200 %

Results | Messages | Execution plan

```
(14896 rows affected)
Table 'Users'. Scan count 5, logical reads 142151,
```

Worse

2.2 p20

Key lookups are a choice.

Most of the time, SQL Server chooses wisely between:

- Index seek + key lookup:
  unpredictable number of page reads, random
- Clustered index scan:
  predictable number of page reads, sequential

And key lookups aren't that big of a deal.

2.2 p21

To “fix” key lookups

1. Select less fields, or
2. Add fields to the nonclustered index (especially if they’re residual predicates)

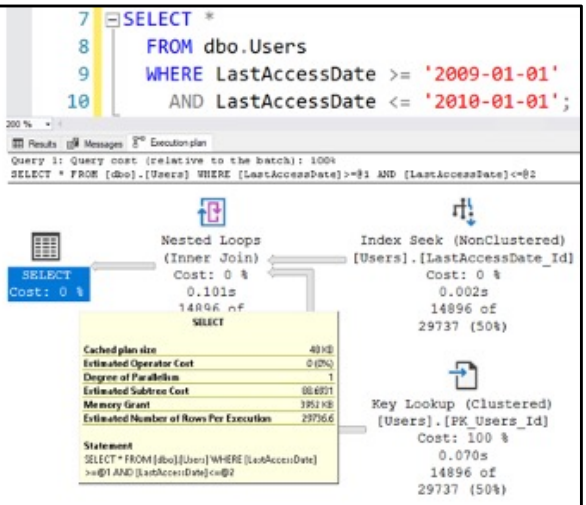
2.2 p22

In our case:

Maybe we could sweet-talk the developers out of doing SELECT *

Or maybe we really do need all those fields, and we have to live with it

But we sure can't add all of the fields to our index



2.2 p23

Residual Predicates

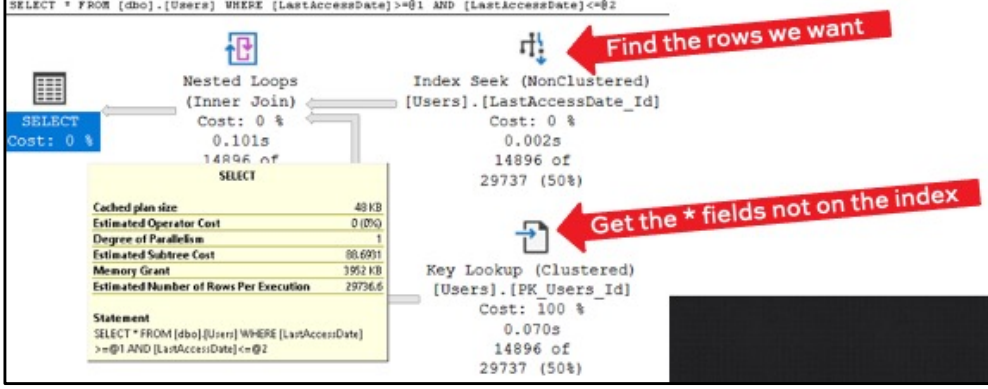
2.2 p24

Back to our one-year query:

```
7  SELECT *
8    FROM dbo.Users
9    WHERE LastAccessDate >= '2009-01-01'
10     AND LastAccessDate <= '2010-01-01';
```

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>=@1 AND [LastAccessDate]<=@2
```



Add a field to the WHERE clause

```
7   SELECT *
8     FROM dbo.Users
9     WHERE LastAccessDate >= '2009-01-01'
10      AND LastAccessDate <= '2010-01-01'
11      AND Location = N'Iceland';
12  GO
```

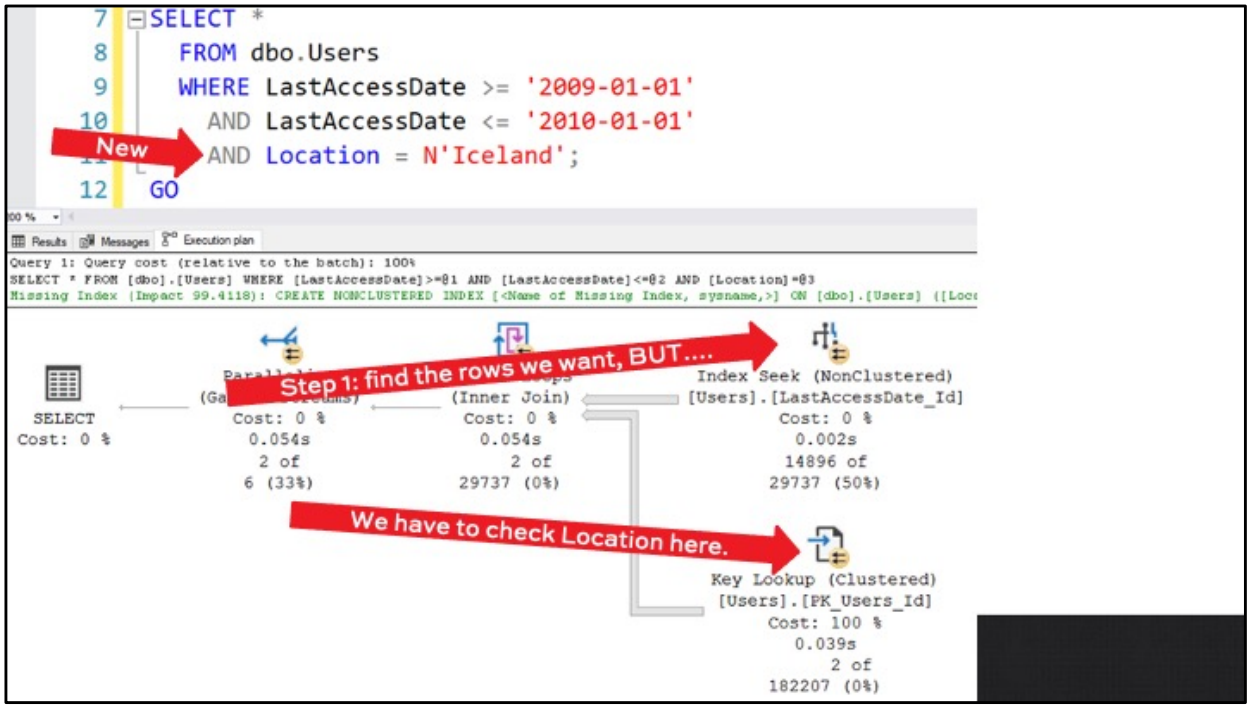
New

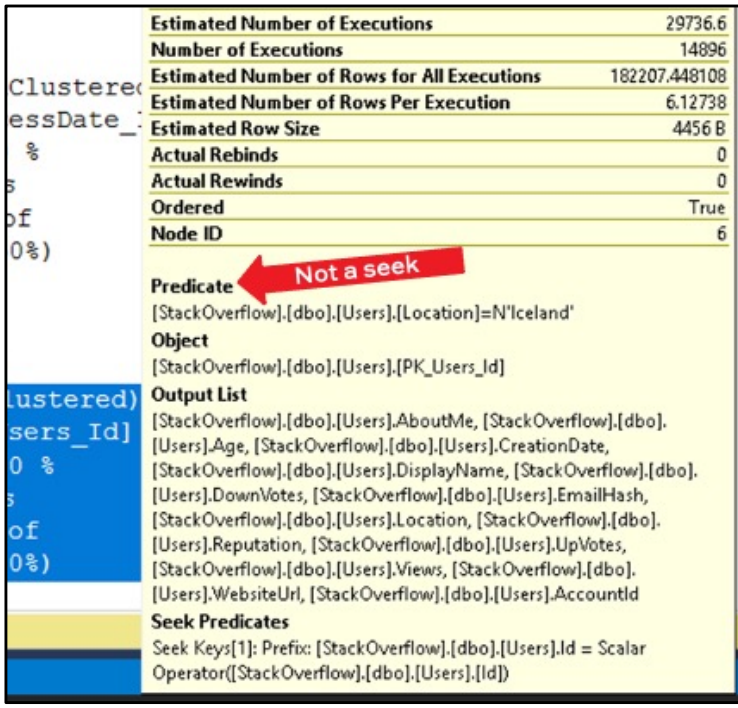
200 %

Results | Messages

Only 2 rows match

	Id	AboutMe	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate	Location
1	10442	NULL	2008-09-15 23:09:36.133	Amar	0	NULL	2009-01-14 02:35:49.663	Iceland
2	212173	NULL	2009-11-16 16:02:01.153	Jóhann Þórir Jóhannsson	0	NULL	2009-11-17 10:54:15.213	Iceland





Residual predicate

Something we're searching for, but we have to do a key lookup to check it

Fixing residual predicates

Add them to the nonclustered index

Can be anywhere: don't have to be keyed

We can check during the fast nonclustered index operation

We can cut down on the number of key lookups performed (or eliminate 'em!)

2.2 p29

Why this helps

In step 1, we'll find ONLY the rows we need.

We'll still have the Key Lookup.

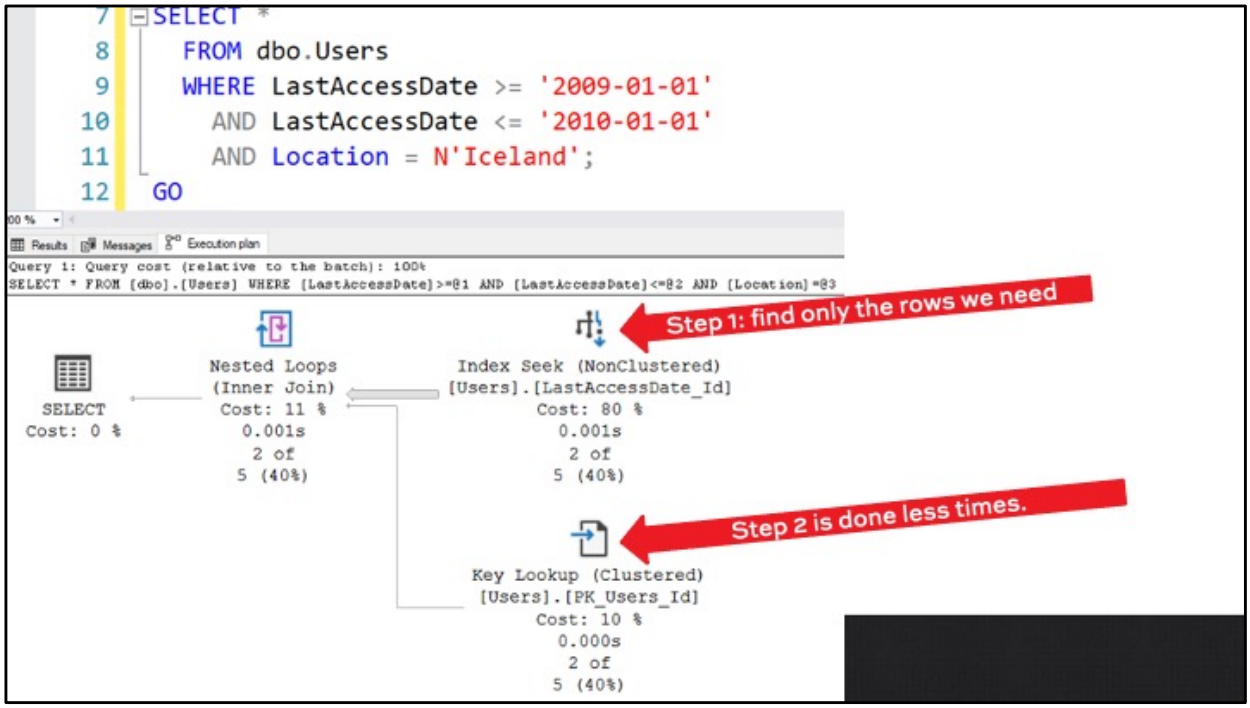
We just won't do so many of them:
we'll only look up the rows we actually need.

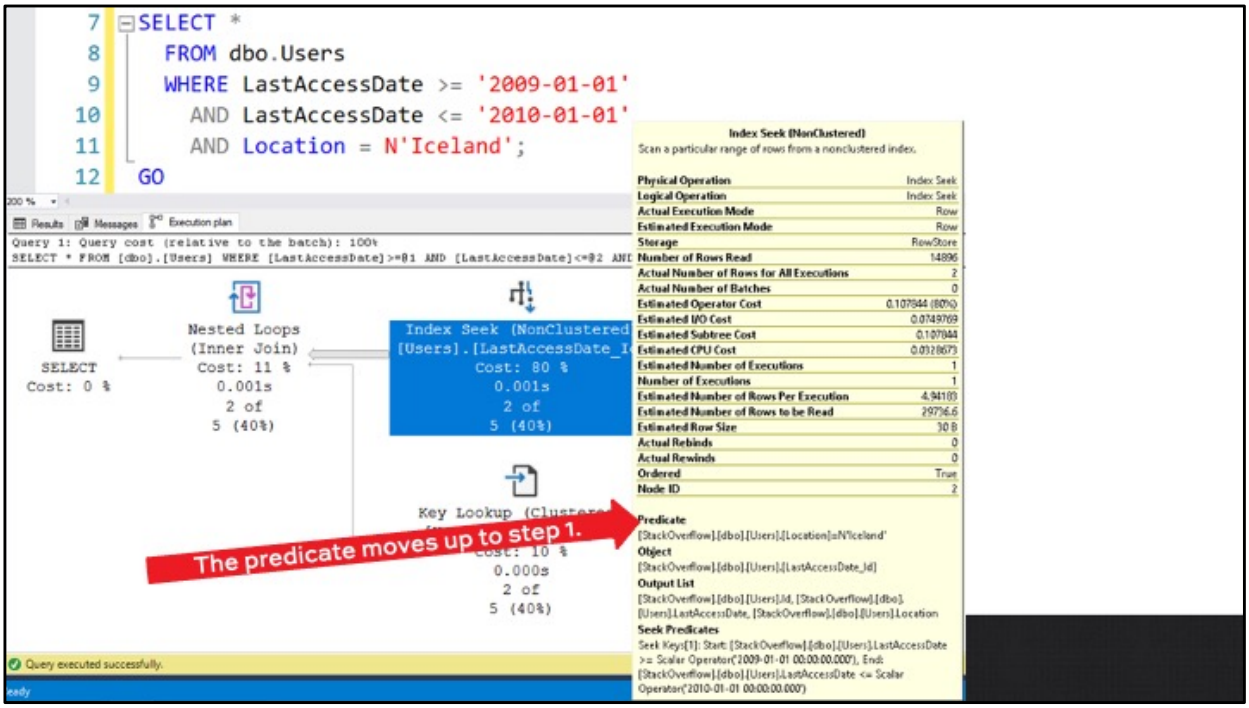
2.2 p30

```sql
/* Old index: */
CREATE INDEX LastAccessDate_Id ON dbo.Users(LastAccessDate, Id);
GO

/* New index: */
CREATE INDEX LastAccessDate_Id ON dbo.Users(LastAccessDate, Id)
    INCLUDE (Location) WITH (DROP_EXISTING = ON);
```

2.2 p31





How to tell that it helped

Number of Rows Read	14896
Actual Number of Rows for All Executions	2

Number of Rows Read: this would normally be the number of key lookups we'd have had to do in order to check Location.

Actual Number of Rows: way lower because so few rows matched Iceland.

This was a success!

2.2 p34

Way less logical reads.

Before, we did 45,665 page reads. Now:

```sql
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2009-01-01'
    AND LastAccessDate <= '2010-01-01'
    AND Location = N'Iceland';
```

200 %

Results | Messages | Execution plan

```
(2 rows affected)
Table 'Users'. Scan count 1, logical reads 49, phy
```

2.2 p35

But this trick only helps IF...

The predicate is actually selective.

The less selective it is,
the less this technique helps.

For example:

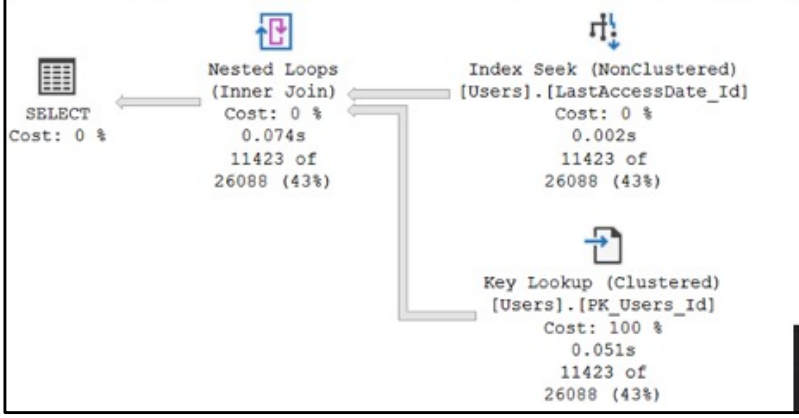
```
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2009-01-01'
    AND LastAccessDate <= '2010-01-01'
    AND Location IS NULL;
```

2.2 p36

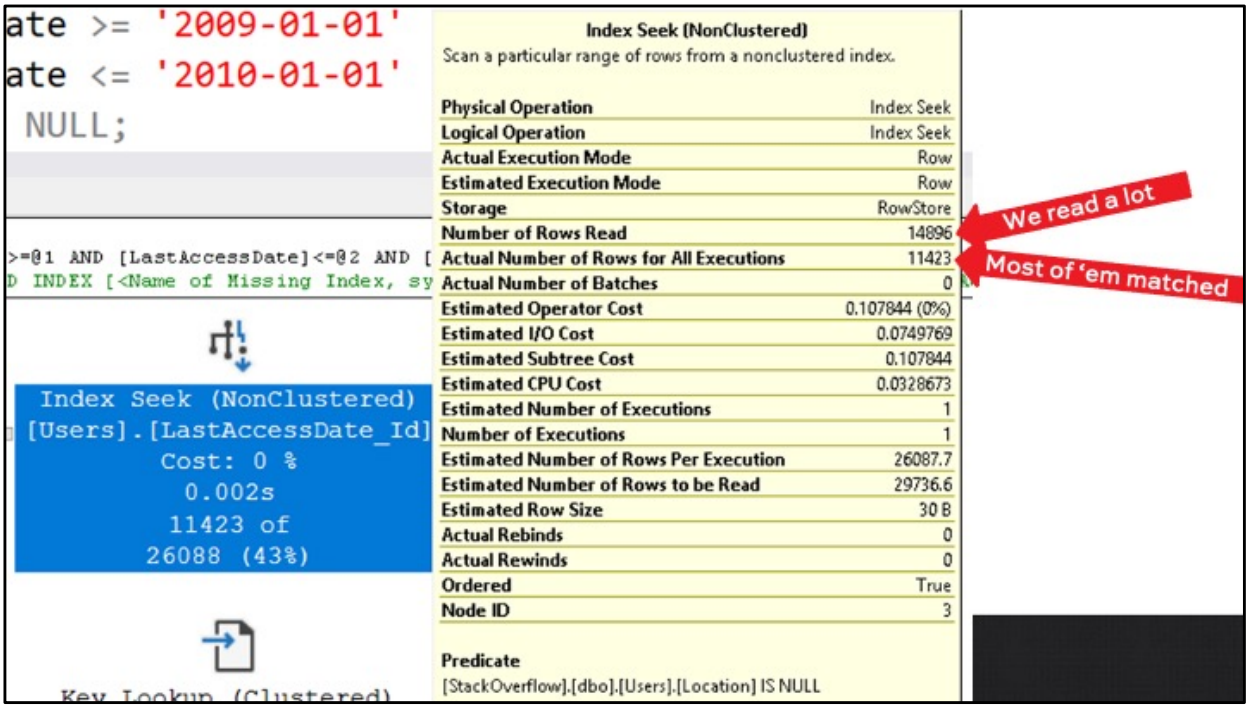
```
7  SELECT *
8    FROM dbo.Users
9    WHERE LastAccessDate >= '2009-01-01'
10     AND LastAccessDate <= '2010-01-01'
11     AND Location IS NULL;
```

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>=@1 AND [LastAccessDate]<=@2 AND [Locat
Missing Index (Impact 99.3755): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname
```



It looks the same, but...



Recap

2.2 p39

Key lookups aren't evil anymore.

They were evil for magnetic frisbees.

These days, generally, SQL Server does a good job of managing when to use 'em.

To fix 'em, you can either:

- Select less columns (you won't)
- Build wide covering indexes (you won't)

But selective residual predicates are different: cover those.

2.2 p40

Now, it's your turn.

Using the Users table:

- Create an index with at least 3 columns
- Write a query that produces a residual predicate in the key lookup

Don't test it on your VM
(since you're running the lab):
just turn it in in Slack, and I'll run it.

2.2 p41