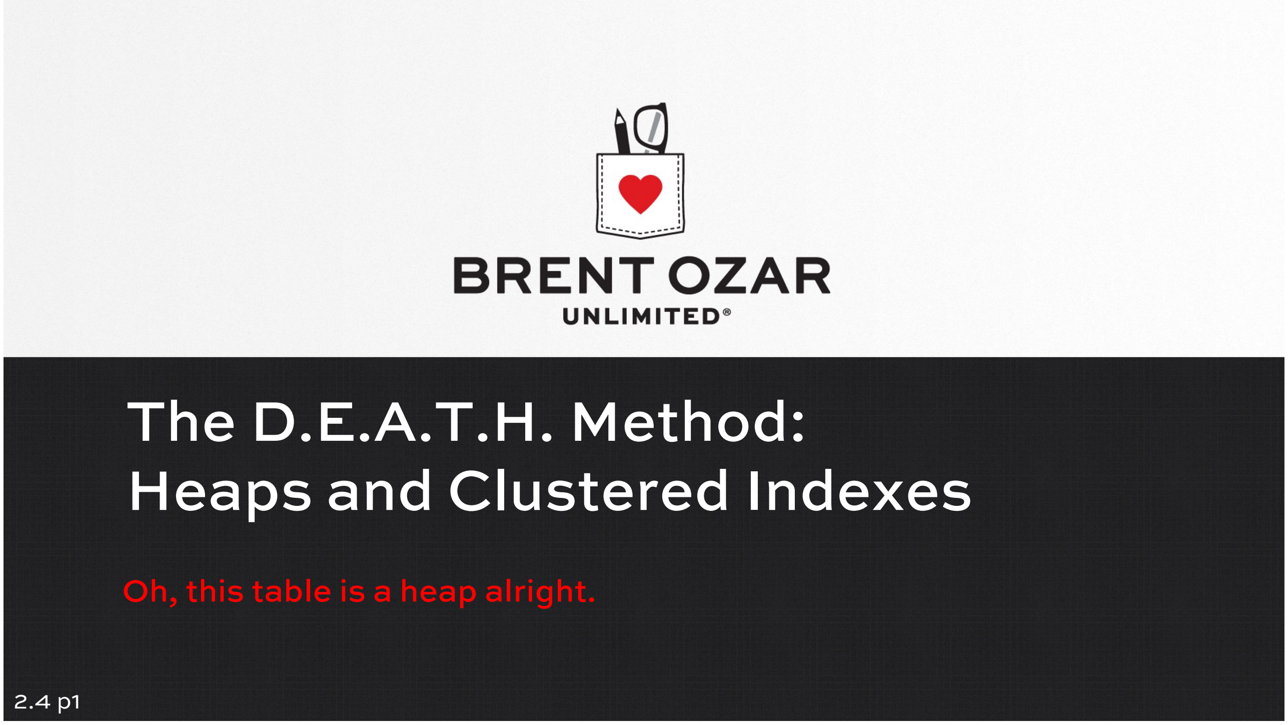
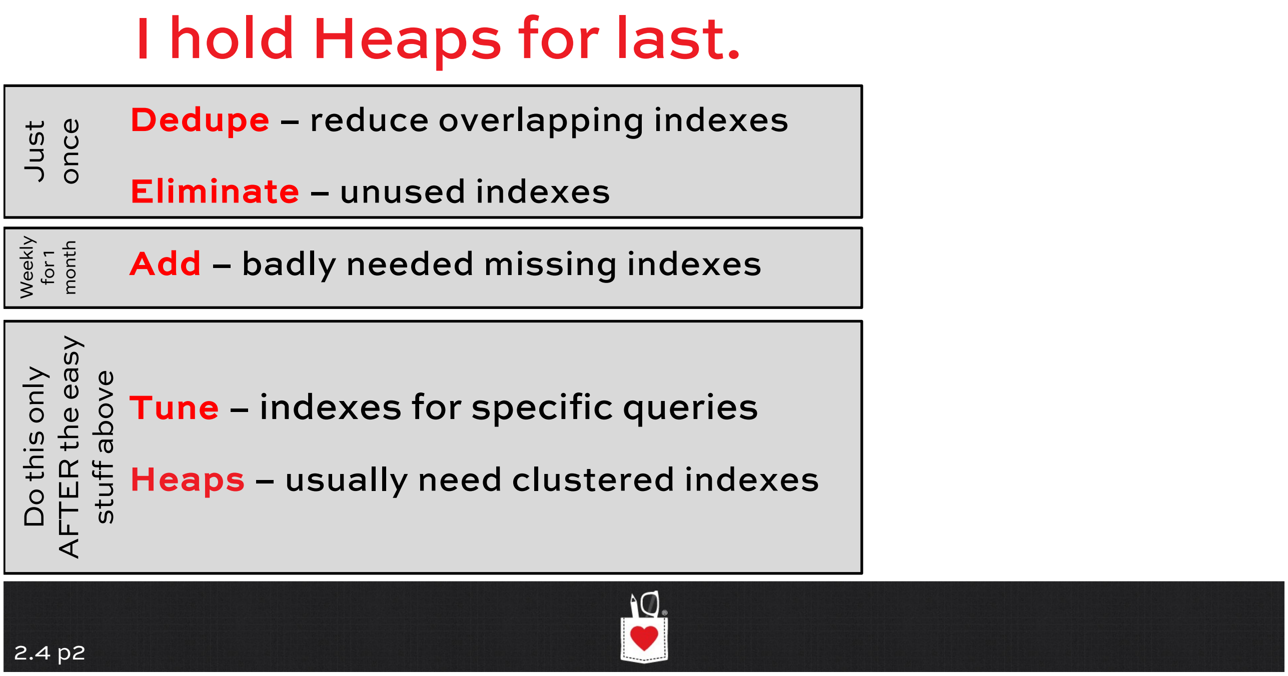


BRENT OZAR
UNLIMITED®

The D.E.A.T.H. Method: Heaps and Clustered Indexes

Oh, this table is a heap alright.

2.4 p1



I hold Heaps for last.

Just once	**Dedupe** – reduce overlapping indexes **Eliminate** – unused indexes
Weekly for 1 month	**Add** – badly needed missing indexes
Do this only AFTER the easy stuff above	**Tune** – indexes for specific queries **Heaps** – usually need clustered indexes

2.4 p2

© Brent Ozar Unlimited®. All rights reserved.

Clustered indexes are a little controversial.

Developers say things like:

- “I ran a load test and clustered indexes slowed us down.”
- “Heaps are faster for inserts.”
- “There’s nothing unique about a row here.”
- “We can’t afford to have downtime to add a clustered index.”
- “We’re just not sure what’s the right set of keys.”

So I hold clustered key implementations for last.

We’re going to cover

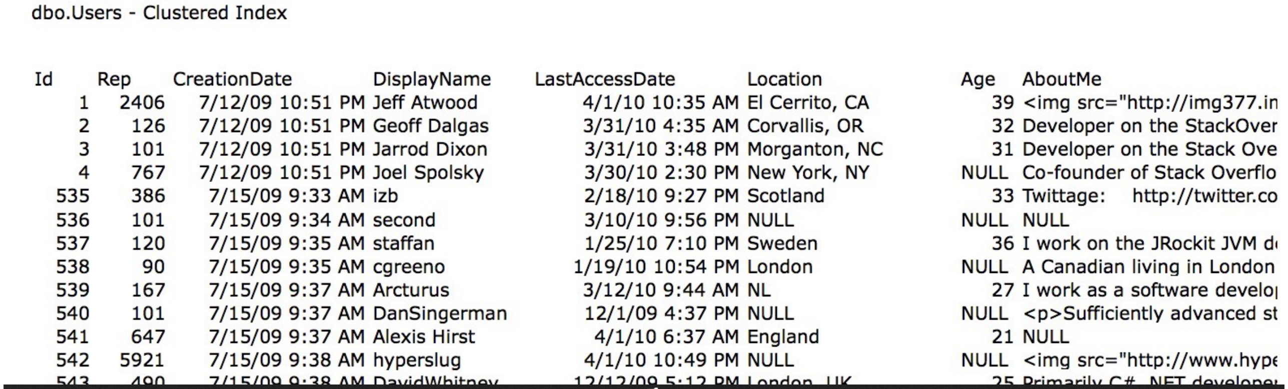
1. How heaps & their indexes are organized on disk
2. The benefit of heaps
3. The drawbacks
4. How to design good clustering keys with the SUN-E guidelines

© Brent Ozar Unlimited®. All rights reserved.

How heaps are organized on disk

We talk about the clustered index.

This is what your pages look like when you set Id as the clustering key:



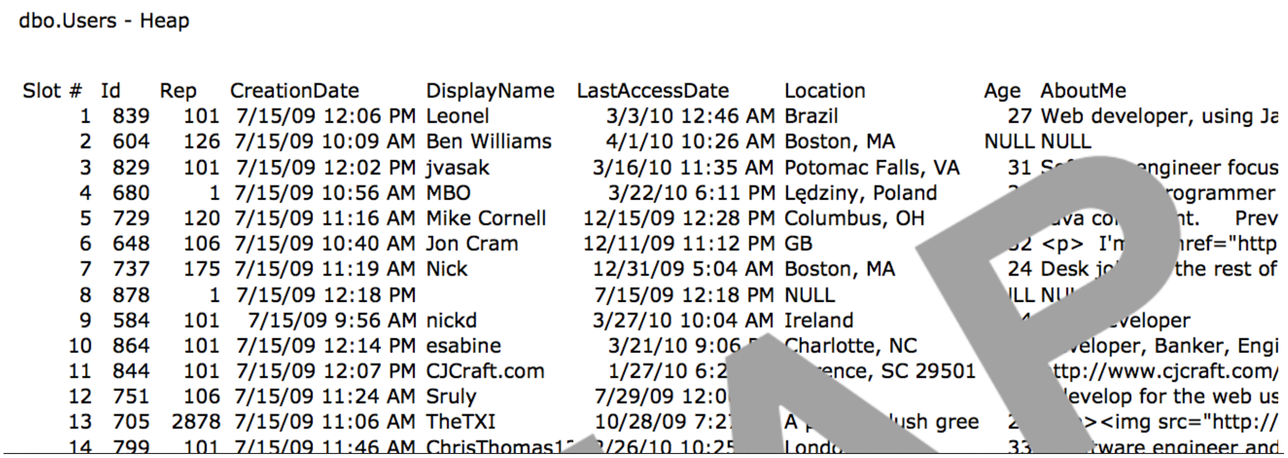
dbo.Users - Clustered Index

Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	2406	7/12/09 10:51 PM	Jeff Atwood	4/1/10 10:35 AM	El Cerrito, CA	39	<img src="http://img377.in
2	126	7/12/09 10:51 PM	Geoff Dalgas	3/31/10 4:35 AM	Corvallis, OR	32	Developer on the StackOver
3	101	7/12/09 10:51 PM	Jarrod Dixon	3/31/10 3:48 PM	Morganton, NC	31	Developer on the Stack Ove
4	767	7/12/09 10:51 PM	Joel Spolsky	3/30/10 2:30 PM	New York, NY	NULL	Co-founder of Stack Overflo
535	386	7/15/09 9:33 AM	izb	2/18/10 9:27 PM	Scotland	33	Twittage: http://twitter.co
536	101	7/15/09 9:34 AM	second	3/10/10 9:56 PM	NULL	NULL	NULL
537	120	7/15/09 9:35 AM	staffan	1/25/10 7:10 PM	Sweden	36	I work on the JRockit JVM d
538	90	7/15/09 9:35 AM	cgreeno	1/19/10 10:54 PM	London	NULL	A Canadian living in London
539	167	7/15/09 9:37 AM	Arcturus	3/12/10 9:44 AM	NL	27	I work as a software develo
540	101	7/15/09 9:37 AM	DanSingerman	12/1/09 4:37 PM	NULL	NULL	<p>Sufficiently advanced st
541	647	7/15/09 9:37 AM	Alexis Hirst	4/1/10 6:37 AM	England	21	NULL
542	5921	7/15/09 9:38 AM	hyperslug	4/1/10 10:49 PM	NULL	NULL	<img src="http://www.hype
543	490	7/15/09 9:38 AM	DavidWhitney	12/12/09 5:12 PM	London, UK	25	Primarily C#, .NET developer

© Brent Ozar Unlimited®. All rights reserved.

If you don’t set a clustered index…

SQL Server still has to store all the data – it’s just not ordered (in a way that’s useful to you or me.)

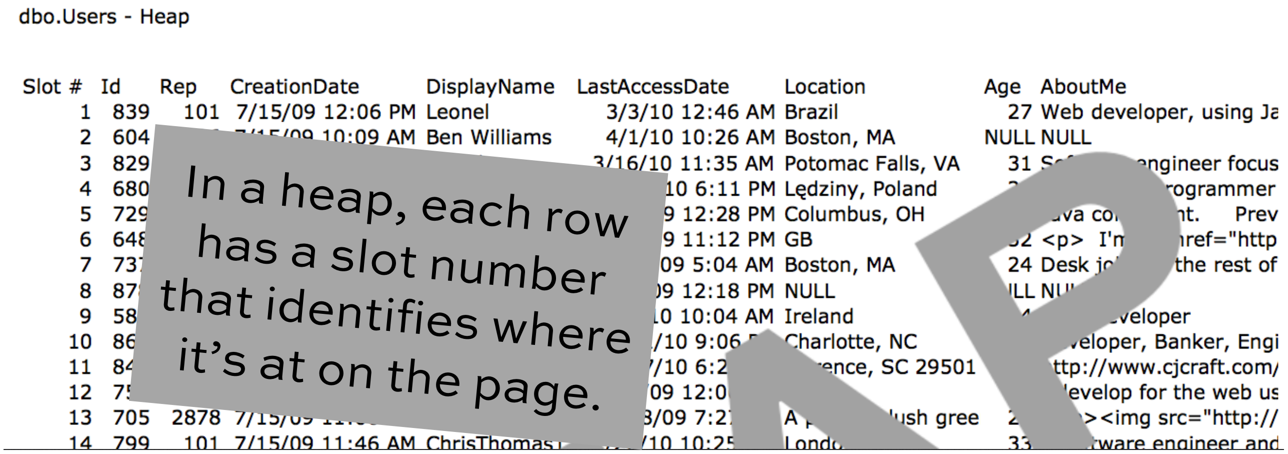


dbo.Users - Heap

Slot #	Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	839	101	7/15/09 12:06 PM	Leonel	3/3/10 12:46 AM	Brazil	27	Web developer, using Ja
2	604	126	7/15/09 10:09 AM	Ben Williams	4/1/10 10:26 AM	Boston, MA	NULL	NULL
3	829	101	7/15/09 12:02 PM	jvasak	3/16/10 11:35 AM	Potomac Falls, VA	31	S engineer focus
4	680	1	7/15/09 10:56 AM	MBO	3/22/10 6:11 PM	Lędziny, Poland		rogrammer
5	729	120	7/15/09 11:16 AM	Mike Cornell	12/15/09 12:28 PM	Columbus, OH		va co nt. Prev
6	648	106	7/15/09 10:40 AM	Jon Cram	12/11/09 11:12 PM	GB	2	<p> I'n href="http
7	737	175	7/15/09 11:19 AM	Nick	12/31/09 5:04 AM	Boston, MA	24	Desk jo the rest of
8	878	1	7/15/09 12:18 PM		7/15/09 12:18 PM	NULL	LL	NU
9	584	101	7/15/09 9:56 AM	nickd	3/27/10 10:04 AM	Ireland		veloper
10	864	101	7/15/09 12:14 PM	esabine	3/21/10 9:06	Charlotte, NC		veloper, Banker, Engi
11	844	101	7/15/09 12:07 PM	CJCraft.com	1/27/10 6:2	ence, SC 29501		ttp://www.cjcraft.com/
12	751	106	7/15/09 11:24 AM	Sruly	7/29/09 12:0			evelop for the web us
13	705	2878	7/15/09 11:06 AM	TheTXI	10/28/09 7:2	A ush gree	2	><img src="http://
14	799	101	7/15/09 11:46 AM	ChrisThomas1	2/26/10 10:25	Londo	33	ware engineer and

If you don’t set a clustered index…

SQL Server still has to store all the data – it’s just not ordered (in a way that’s useful to you or me.)



dbo.Users - Heap

Slot #	Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	839	101	7/15/09 12:06 PM	Leonel	3/3/10 12:46 AM	Brazil	27	Web developer, using Ja
2	604		7/15/09 10:09 AM	Ben Williams	4/1/10 10:26 AM	Boston, MA	NULL	NULL
3	829				3/16/10 11:35 AM	Potomac Falls, VA	31	S engineer focus
4	680				10 6:11 PM	Lędziny, Poland		rogrammer
5	729				9 12:28 PM	Columbus, OH		va co nt. Prev
6	648				9 11:12 PM	GB	2	<p> I'n href="http
7	73				09 5:04 AM	Boston, MA	24	Desk jo the rest of
8	87				09 12:18 PM	NULL	LL	NU
9	58				10 10:04 AM	Ireland		veloper
10	86				/10 9:06	Charlotte, NC		veloper, Banker, Engi
11	84				/10 6:2	ence, SC 29501		ttp://www.cjcraft.com/
12	75				09 12:0			evelop for the web us
13	705	2878			3/09 7:2	A ush gree	2	><img src="http://
14	799	101	7/15/09 11:46 AM	ChrisThomas	/10 10:25	Londo	33	ware engineer and

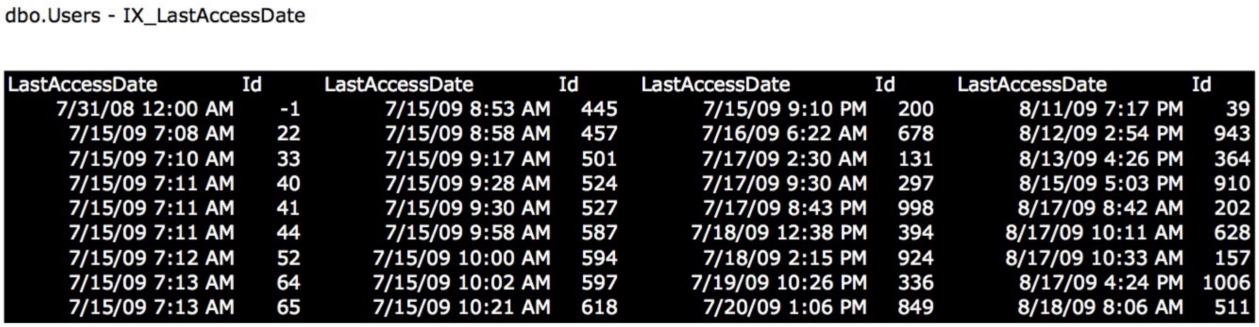
In a heap, each row has a slot number that identifies where it’s at on the page.

© Brent Ozar Unlimited®. All rights reserved.

Because we need to find rows.

Before, when we had a clustered index, the rows were sorted by their clustering key (ID).

Each nonclustered index row included the ID so we could jump back to the matching clustered index row.



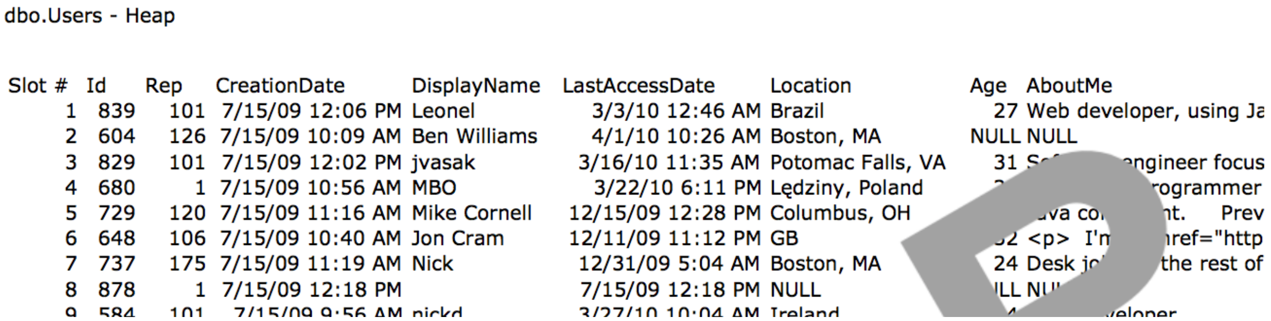
dbo.Users - IX_LastAccessDate

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

So what do heaps do?

It wouldn't make sense to put the ID # on each nonclustered index row.

If you wanted to find user ID # 26837, how would you quickly seek to his clustered index row?



dbo.Users - Heap

Slot #	Id	Rep	CreationDate	DisplayName	LastAccessDate	Location	Age	AboutMe
1	839	101	7/15/09 12:06 PM	Leonel	3/3/10 12:46 AM	Brazil	27	Web developer, using Ja
2	604	126	7/15/09 10:09 AM	Ben Williams	4/1/10 10:26 AM	Boston, MA	NULL	NULL
3	829	101	7/15/09 12:02 PM	jvasak	3/16/10 11:35 AM	Potomac Falls, VA	31	S engineer focus
4	680	1	7/15/09 10:56 AM	MBO	3/22/10 6:11 PM	Lędziny, Poland		rogrammer
5	729	120	7/15/09 11:16 AM	Mike Cornell	12/15/09 12:28 PM	Columbus, OH		va co nt. Prev
6	648	106	7/15/09 10:40 AM	Jon Cram	12/11/09 11:12 PM	GB	2	<p> I'n href="http
7	737	175	7/15/09 11:19 AM	Nick	12/31/09 5:04 AM	Boston, MA	24	Desk jo the rest of
8	878	1	7/15/09 12:18 PM		7/15/09 12:18 PM	NULL	LL	NU
9	584	101	7/15/09 9:56 AM	nickd	3/27/10 10:04 AM	Ireland		veloper

© Brent Ozar Unlimited®. All rights reserved.

Heaps use the Row Identifier.

This combination of data helps you jump directly to the row you're looking for:

- File number
- Page number
- Slot number

Together, they're called the RID:
Row Identifier.

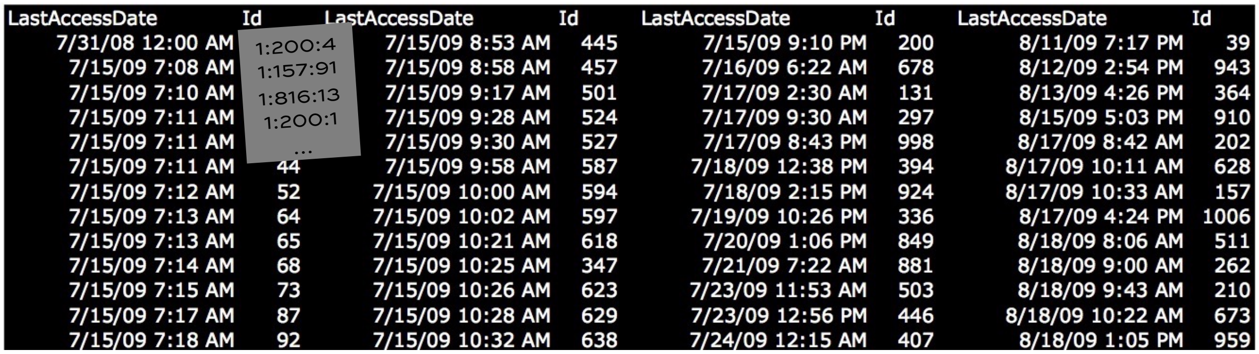
2.4 p11

Then the RID is on the NC index.

Instead of Id, you would see File:Page:SlotNumber.

I didn't fully illustrate that for you because you're imaginative enough to figure that out on your own. (Also, I'm lazy.)

dbo.Users - IX_LastAccessDate



LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id
7/31/08 12:00 AM	1:200:4	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	1:157:91	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	1:816:13	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	1:200:1	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	...	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
7/15/09 7:15 AM	73	7/15/09 10:26 AM	623	7/23/09 11:53 AM	503	8/18/09 9:43 AM	210
7/15/09 7:17 AM	87	7/15/09 10:28 AM	629	7/23/09 12:56 PM	446	8/18/09 10:22 AM	673
7/15/09 7:18 AM	92	7/15/09 10:32 AM	638	7/24/09 12:15 AM	407	8/18/09 1:05 PM	959

© Brent Ozar Unlimited®. All rights reserved.

Benefits of heaps

2.4 p13

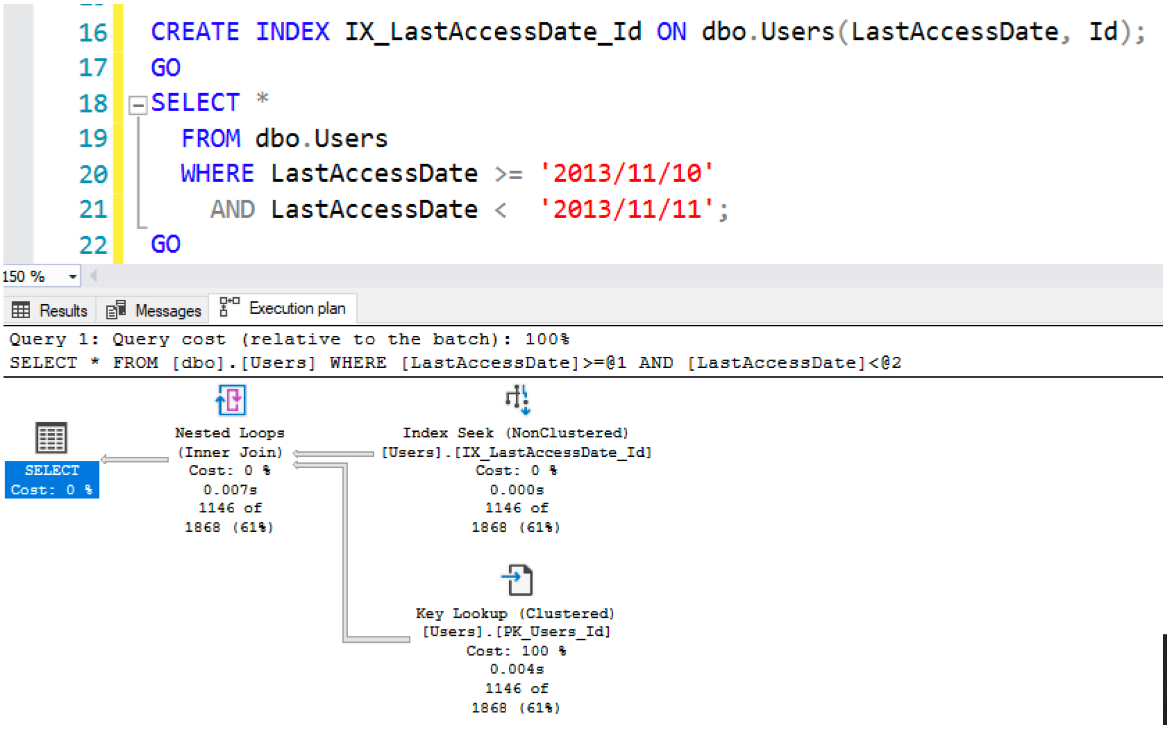
Key lookups are faster.

This is a really weird edge case, but it’s neat.

I’ll start with the Users table, with its normal clustered index on Id.

2.4 p14

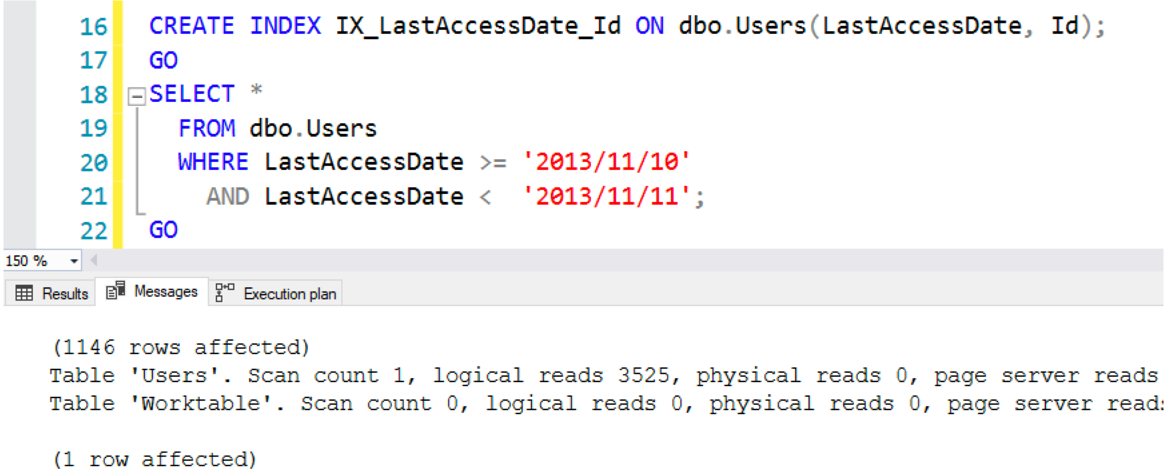
© Brent Ozar Unlimited®. All rights reserved.



```
CREATE INDEX IX_LastAccessDate_Id ON dbo.Users(LastAccessDate, Id);
GO
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

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

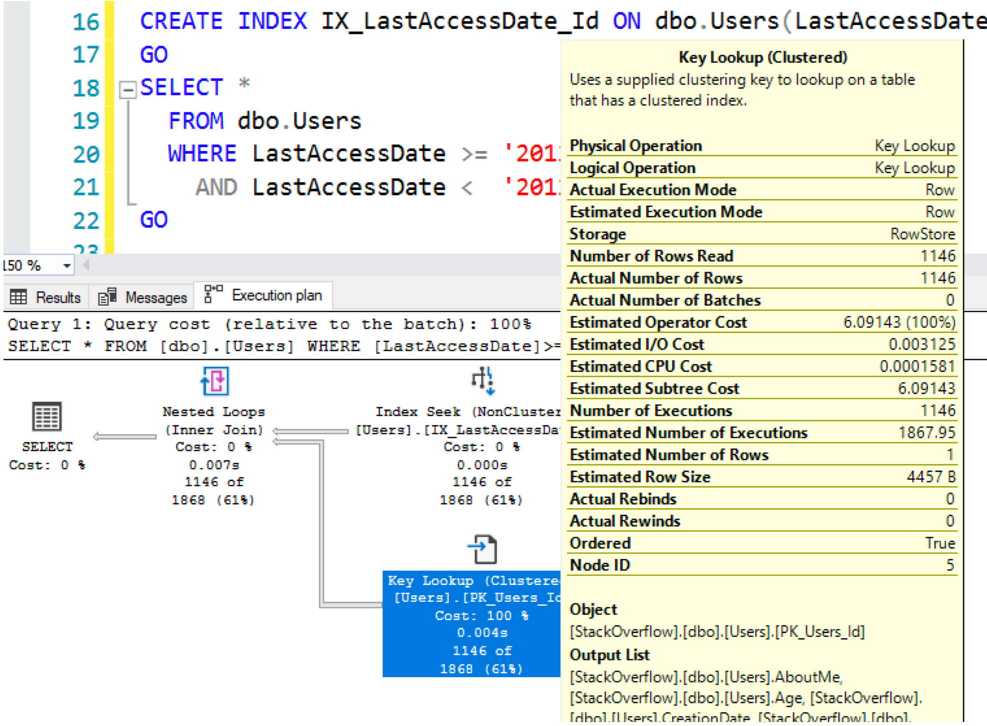
Seek + key lookup = 3,525 reads.



```
CREATE INDEX IX_LastAccessDate_Id ON dbo.Users(LastAccessDate, Id);
GO
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

```
(1146 rows affected)
Table 'Users'. Scan count 1, logical reads 3525, physical reads 0, page server reads
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server read:

(1 row affected)
```

© Brent Ozar Unlimited®. All rights reserved.



```sql
CREATE INDEX IX_LastAccessDate_Id ON dbo.Users(LastAccessDate, Id);
GO
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '201
    AND LastAccessDate <  '201
GO
```

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

Key Lookup (Clustered)

Uses a supplied clustering key to lookup on a table that has a clustered index.

Physical Operation	Key Lookup
Logical Operation	Key Lookup
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	1146
Actual Number of Rows	1146
Actual Number of Batches	0
Estimated Operator Cost	6.09143 (100%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001581
Estimated Subtree Cost	6.09143
Number of Executions	1146
Estimated Number of Executions	1867.95
Estimated Number of Rows	1
Estimated Row Size	4457 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	5

Object

[StackOverflow].[dbo].[Users].[PK_Users_Id]

Output List

[StackOverflow].[dbo].[Users].AboutMe, [StackOverflow].[dbo].[Users].Age, [StackOverflow].[dbo].[Users].CreationDate, [StackOverflow].[dbo].

The key lookup is executed 1,146 times.

Each time we do a key lookup:

SQL Server knows the table and the clustering key's value (the Id.)

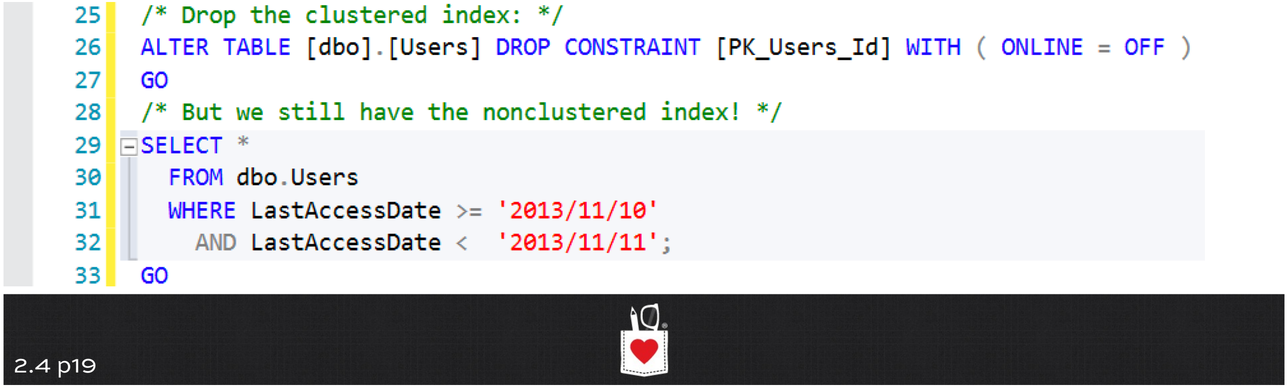
It *doesn't* know where that Id physically lives, so it has to figure out:

- What 8KB page(s) hold the clustered index
- Look up what physical page holds that Id
- Open up the physical page for that Id

© Brent Ozar Unlimited®. All rights reserved.

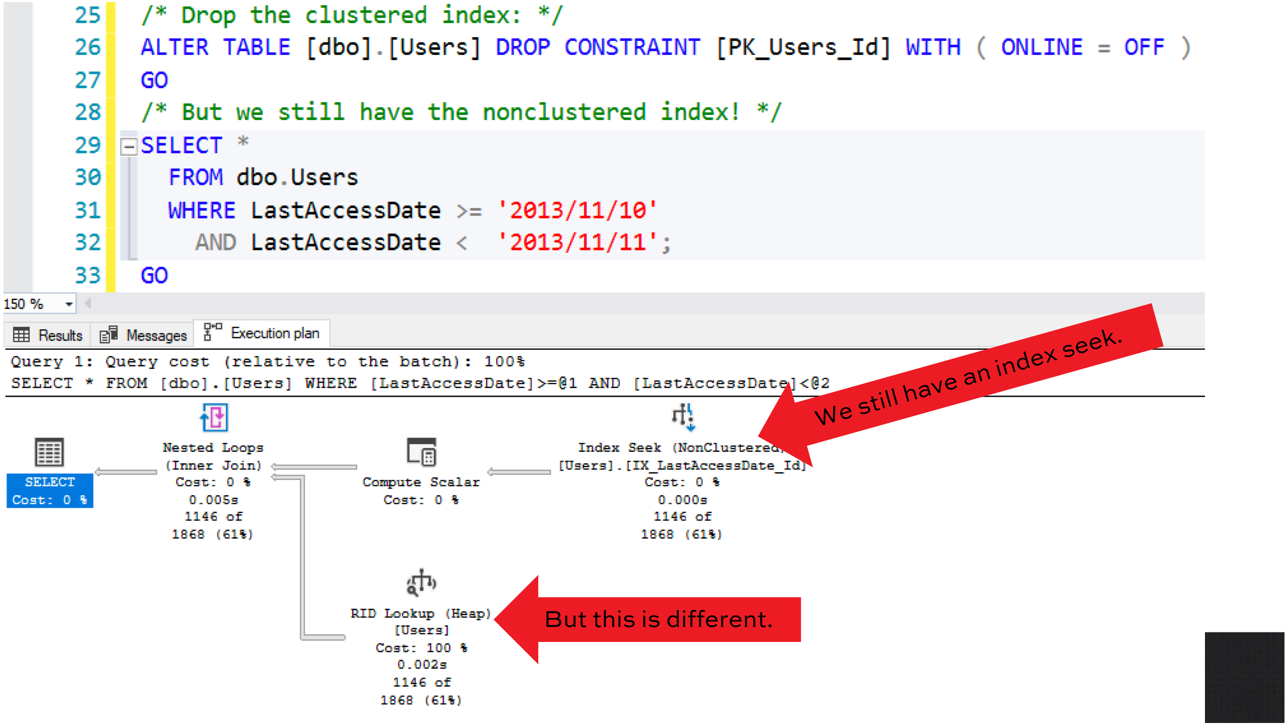
But heaps are different.

I’m going to drop the clustered primary key: that creates a heap. The Users table is still there, but it’s just now stored as a heap – aka, random order.



```sql
/* Drop the clustered index: */
ALTER TABLE [dbo].[Users] DROP CONSTRAINT [PK_Users_Id] WITH ( ONLINE = OFF )
GO
/* But we still have the nonclustered index! */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

2.4 p19



```sql
/* Drop the clustered index: */
ALTER TABLE [dbo].[Users] DROP CONSTRAINT [PK_Users_Id] WITH ( ONLINE = OFF )
GO
/* But we still have the nonclustered index! */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

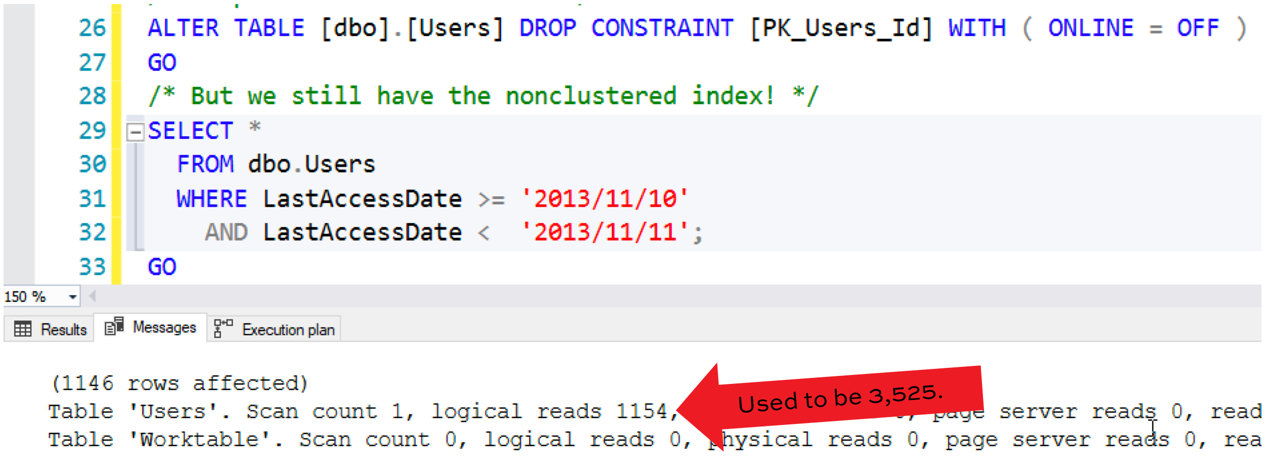
```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[Users] WHERE [LastAccessDate]>=@1 AND [LastAccessDate]<@2
```

We still have an index seek.

But this is different.

© Brent Ozar Unlimited®. All rights reserved.

It’s less logical reads.



```
ALTER TABLE [dbo].[Users] DROP CONSTRAINT [PK_Users_Id] WITH ( ONLINE = OFF )
GO
/* But we still have the nonclustered index! */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

```
(1146 rows affected)
Table 'Users'. Scan count 1, logical reads 1154,                  page server reads 0, read
Table 'Worktable'. Scan count 0, logical reads 0,  hysical reads 0, page server reads 0, rea
```

Used to be 3,525.

2.4 p21

Each time we do a key lookup:

When we have a clustered index, a key lookup has to:

- What 8KB page(s) hold the clustered index
- Look up what physical page holds that Id
- Open up the physical page for that Id

But when we have a heap, each nonclustered index row flat out tells you which page number and slot number the row is on, so we just have to:

- ~~Find what 8KB page(s) hold the clustered index~~
- ~~Look up what physical page holds that Id~~
- Open up the physical page for that Id

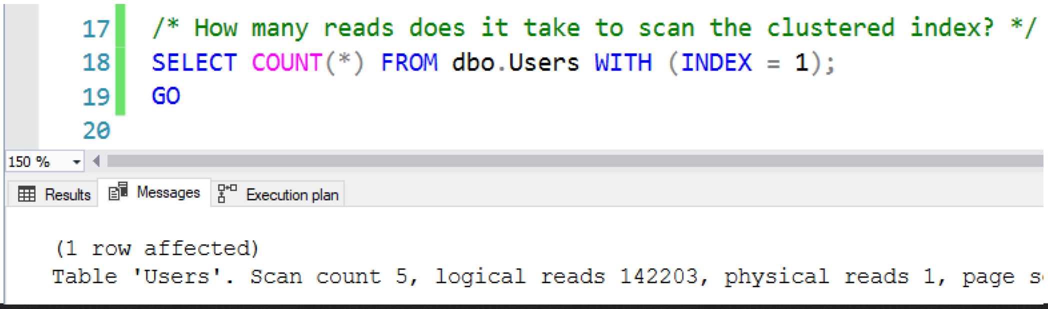
2.4 p22

© Brent Ozar Unlimited®. All rights reserved.

Table scans are faster, too.

With a normal clustered index, SQL Server may use the B-tree to navigate through all of the rows.

Scanning the clustered index = 142,203 page reads.



```
/* How many reads does it take to scan the clustered index? */
SELECT COUNT(*) FROM dbo.Users WITH (INDEX = 1);
GO
```

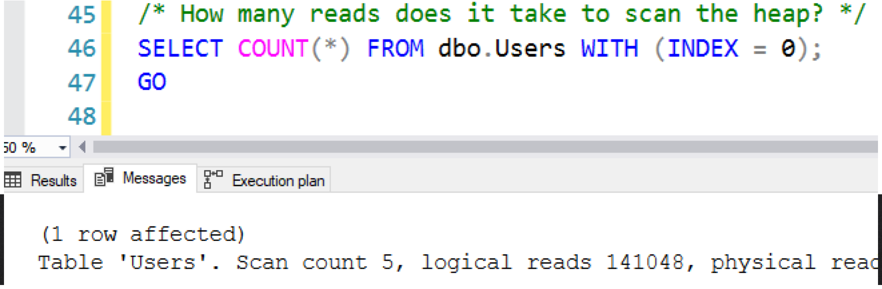
```
(1 row affected)
Table 'Users'. Scan count 5, logical reads 142203, physical reads 1, page s
```

The heap does <1% less reads.

The heap scans through the pages in the order they're physically allocated, without hassling with the B-tree.

This isn't a huge savings: it's just 1,155 less page reads in this case.

I dropped the clustered key to show it:



```
/* How many reads does it take to scan the heap? */
SELECT COUNT(*) FROM dbo.Users WITH (INDEX = 0);
GO
```

```
(1 row affected)
Table 'Users'. Scan count 5, logical reads 141048, physical reac
```

© Brent Ozar Unlimited®. All rights reserved.

At first, these sound compelling.

- 67% less reads for key lookups
- 1% less reads for table scans
- Possibly faster load times (but this is super-debatable, depends on your ETL)

And there *are* cases where they make sense.

2.4 p25

Possibly good use cases for heaps

Staging tables in data warehouses:

- Shove all the data in quickly
- Scan it back out once
- Truncate it every night

Scan-only tables like data warehouse fact tables:

- Write the data once in an optimized load
- Read only thereafter
- Read pattern is scans

2.4 p26

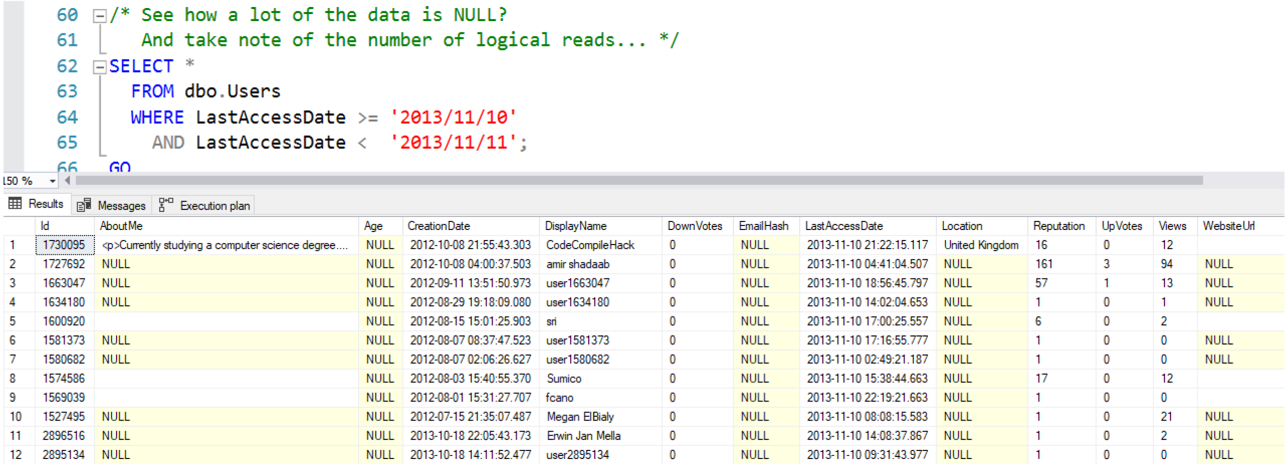
© Brent Ozar Unlimited®. All rights reserved.

But they have drawbacks.

2.4 p27

Continuing with Users...

A lot of our columns are null (empty).



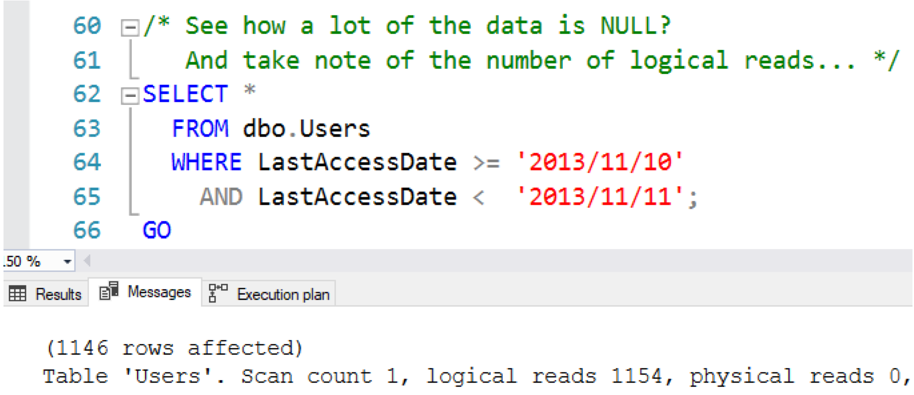
```
/* See how a lot of the data is NULL?
   And take note of the number of logical reads... */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

	Id	AboutMe	Age	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate	Location	Reputation	UpVotes	Views	WebsiteUrl
1	1730095	<p>Currently studying a computer science degree....	NULL	2012-10-08 21:55:43.303	CodeCompileHack	0	NULL	2013-11-10 21:22:15.117	United Kingdom	16	0	12	
2	1727692	NULL	NULL	2012-10-08 04:00:37.503	amir shadaab	0	NULL	2013-11-10 04:41:04.507	NULL	161	3	94	NULL
3	1663047	NULL	NULL	2012-09-11 13:51:50.973	user1663047	0	NULL	2013-11-10 18:56:45.797	NULL	57	1	13	NULL
4	1634180	NULL	NULL	2012-08-29 19:18:09.080	user1634180	0	NULL	2013-11-10 14:02:04.653	NULL	1	0	1	NULL
5	1600920		NULL	2012-08-15 15:01:25.903	sri	0	NULL	2013-11-10 17:00:25.557	NULL	6	0	2	
6	1581373	NULL	NULL	2012-08-07 08:37:47.523	user1581373	0	NULL	2013-11-10 17:16:55.777	NULL	1	0	0	NULL
7	1580682	NULL	NULL	2012-08-07 02:06:26.627	user1580682	0	NULL	2013-11-10 02:49:21.187	NULL	1	0	0	NULL
8	1574586		NULL	2012-08-03 15:40:55.370	Sumico	0	NULL	2013-11-10 15:38:44.663	NULL	17	0	12	
9	1569039		NULL	2012-08-01 15:31:27.707	fcano	0	NULL	2013-11-10 22:19:21.663	NULL	1	0	0	
10	1527495	NULL	NULL	2012-07-15 21:35:07.487	Megan ElBialy	0	NULL	2013-11-10 08:08:15.583	NULL	1	0	21	NULL
11	2896516	NULL	NULL	2013-10-18 22:05:43.173	Erwin Jan Mella	0	NULL	2013-11-10 14:08:37.867	NULL	1	0	2	NULL
12	2895134	NULL	NULL	2013-10-18 14:11:52.477	user2895134	0	NULL	2013-11-10 09:31:43.977	NULL	1	0	0	NULL

© Brent Ozar Unlimited®. All rights reserved.

It takes 1,154 logical reads.

Not a lot – just making a note.



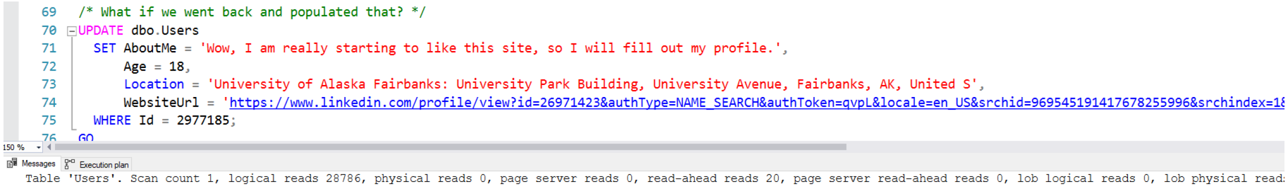
```sql
/* See how a lot of the data is NULL?
   And take note of the number of logical reads... */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

```
(1146 rows affected)
Table 'Users'. Scan count 1, logical reads 1154, physical reads 0,
```

2.4 p29

Let’s update one user’s profile.

And we’re going to fill in the nulls with long values.



```sql
/* What if we went back and populated that? */
UPDATE dbo.Users
  SET AboutMe = 'Wow, I am really starting to like this site, so I will fill out my profile.',
      Age = 18,
      Location = 'University of Alaska Fairbanks: University Park Building, University Avenue, Fairbanks, AK, United S',
      WebsiteUrl = 'https://www.linkedin.com/profile/view?id=26971423&authType=NAME_SEARCH&authToken=qvpL&locale=en_US&srchid=969545191417678255996&srchindex=1
  WHERE Id = 2977185;
GO
```

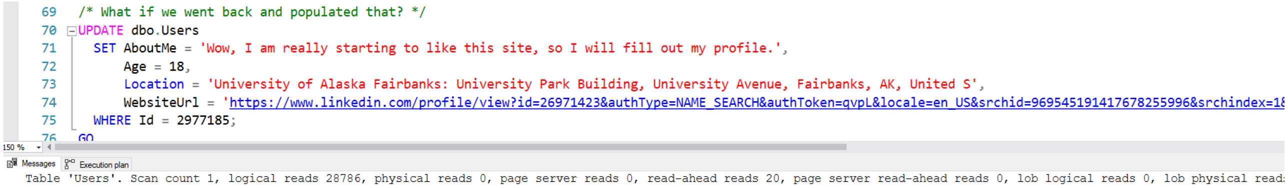
```
Table 'Users'. Scan count 1, logical reads 28786, physical reads 0, page server reads 0, read-ahead reads 20, page server read-ahead reads 0, lob logical reads 0, lob physical read:
```

Check out those logical reads.
Why did it take 28,786 reads to do this?

2.4 p30

© Brent Ozar Unlimited®. All rights reserved.

We have to find User Id 2977185.



```
/* What if we went back and populated that? */
UPDATE dbo.Users
  SET AboutMe = 'Wow, I am really starting to like this site, so I will fill out my profile.',
      Age = 18,
      Location = 'University of Alaska Fairbanks: University Park Building, University Avenue, Fairbanks, AK, United S',
      WebsiteUrl = 'https://www.linkedin.com/profile/view?id=26971423&authType=NAME_SEARCH&authToken=qvpL&locale=en_US&srchid=969545191417678255996&srchindex=1
  WHERE Id = 2977185;
GO
```

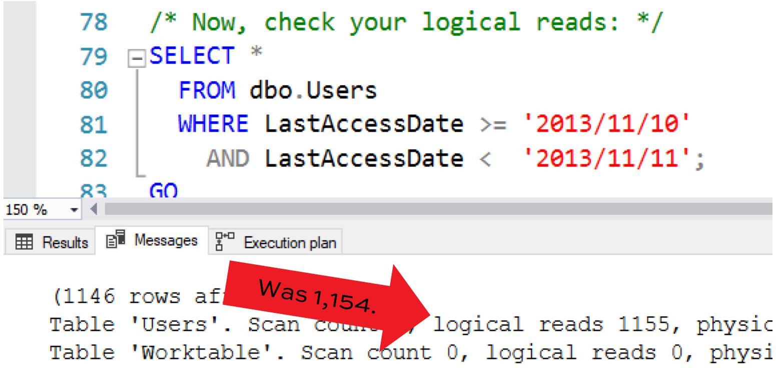
```
Table 'Users'. Scan count 1, logical reads 28786, physical reads 0, page server reads 0, read-ahead reads 20, page server read-ahead reads 0, lob logical reads 0, lob physical read
```

And since the data isn't organized by Id, we have to scan the entire heap to find 'em.

You could create an index on Id, but…
you're wasting one of the 5 & 5.

Now run our SELECT again.

It does 1,155 logical reads. It went up by 1. Why?



```
/* Now, check your logical reads: */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

```
(1146 rows af
Table 'Users'. Scan coun          logical reads 1155, physic
Table 'Worktable'. Scan count 0, logical reads 0, physi
```

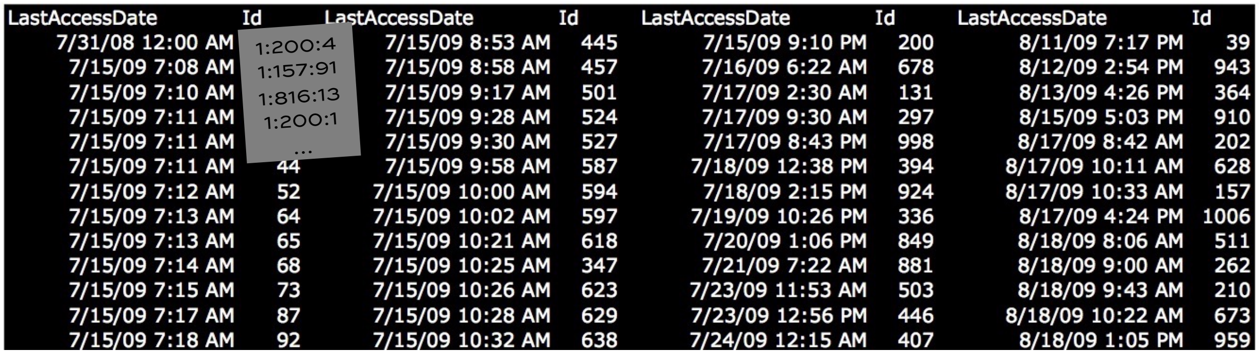
Was 1,154.

© Brent Ozar Unlimited®. All rights reserved.

Think back to our indexes.

Remember how each nonclustered index points back to the full row using the File:Page:SlotNumber?

dbo.Users - IX_LastAccessDate



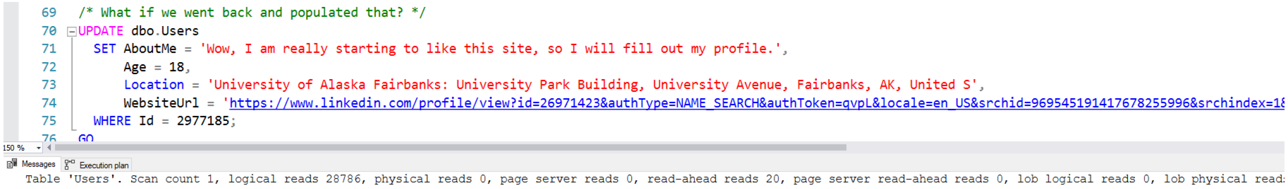
LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id	LastAccessDate	Id
7/31/08 12:00 AM		7/15/09 8:53 AM	445	7/15/09 9:10 PM	200	8/11/09 7:17 PM	39
7/15/09 7:08 AM		7/15/09 8:58 AM	457	7/16/09 6:22 AM	678	8/12/09 2:54 PM	943
7/15/09 7:10 AM		7/15/09 9:17 AM	501	7/17/09 2:30 AM	131	8/13/09 4:26 PM	364
7/15/09 7:11 AM		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		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
7/15/09 7:15 AM	73	7/15/09 10:26 AM	623	7/23/09 11:53 AM	503	8/18/09 9:43 AM	210
7/15/09 7:17 AM	87	7/15/09 10:28 AM	629	7/23/09 12:56 PM	446	8/18/09 10:22 AM	673
7/15/09 7:18 AM	92	7/15/09 10:32 AM	638	7/24/09 12:15 AM	407	8/18/09 1:05 PM	959

1:200:4
1:157:91
1:816:13
1:200:1
...

That's the *original* file:page:slot.

When you update a narrow field (like an empty/null), and you populate it with wider values, there may not be enough empty space on the page.

I purposely used wide values to force this to happen:



```
/* What if we went back and populated that? */
UPDATE dbo.Users
  SET AboutMe = 'Wow, I am really starting to like this site, so I will fill out my profile.',
      Age = 18,
      Location = 'University of Alaska Fairbanks: University Park Building, University Avenue, Fairbanks, AK, United S',
      WebsiteUrl = 'https://www.linkedin.com/profile/view?id=26971423&authType=NAME_SEARCH&authToken=qvpL&locale=en_US&srchid=969545191417678255996&srchindex=1
  WHERE Id = 2977185;
GO
```

```
Table 'Users'. Scan count 1, logical reads 28786, physical reads 0, page server reads 0, read-ahead reads 20, page server read-ahead reads 0, lob logical reads 0, lob physical read
```

© Brent Ozar Unlimited®. All rights reserved.

Not enough space? A row moves.

It moves to a new physical page.

Doesn’t really matter which one – any one with enough empty space will do.

But SQL Server doesn’t go update all the nonclustered indexes for that row with the new F:P:S.

It just leaves a “forwarding pointer” behind at the old F:P:S location saying, “I’ve moved to this new F:P:S.”

2.4 p35

So now a key lookup means:

Use the index to find the row you want

Look up its original page by F:P:S

Find a forwarding pointer

Jump over to the new F:P:S and do another read

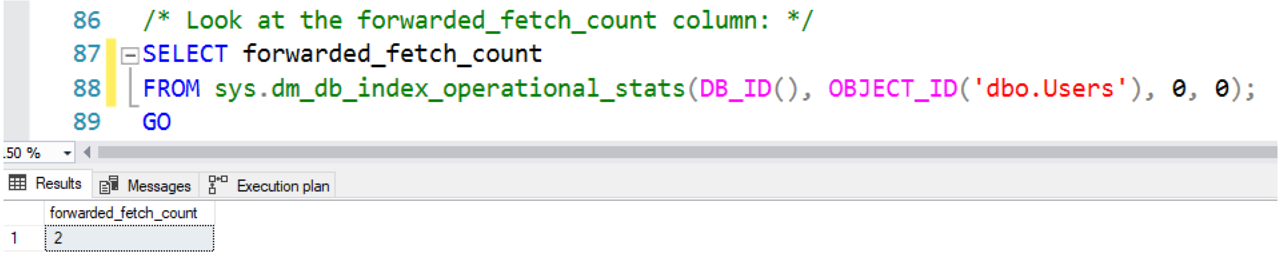
This is called a forwarded fetch.

2.4 p36

© Brent Ozar Unlimited®. All rights reserved.

You can track it in the DMVs.

And we surface this in sp_BlitzIndex, too:



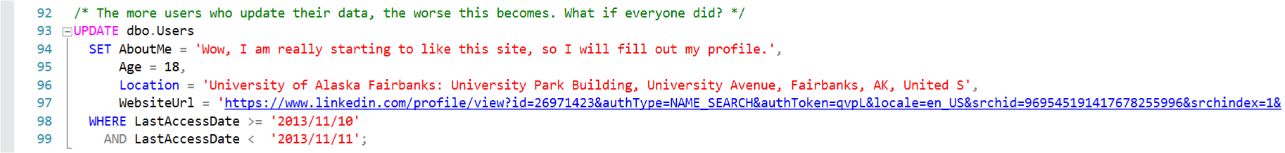
```
/* Look at the forwarded_fetch_count column: */
SELECT forwarded_fetch_count
FROM sys.dm_db_index_operational_stats(DB_ID(), OBJECT_ID('dbo.Users'), 0, 0);
GO
```

```
   forwarded_fetch_count
1  2
```

Forwarded fetches means we're
doing more reads than really necessary.

Let's update the rest of the rows

Update everyone in our LastAccessDate range:

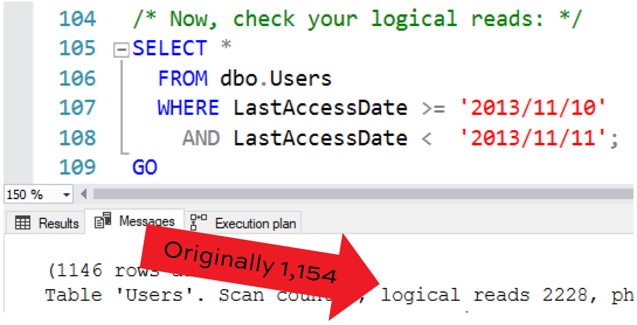


```
/* The more users who update their data, the worse this becomes. What if everyone did? */
UPDATE dbo.Users
  SET AboutMe = 'Wow, I am really starting to like this site, so I will fill out my profile.',
      Age = 18,
      Location = 'University of Alaska Fairbanks: University Park Building, University Avenue, Fairbanks, AK, United S',
      WebsiteUrl = 'https://www.linkedin.com/profile/view?id=26971423&authType=NAME_SEARCH&authToken=qvpL&locale=en_US&srchid=969545191417678255996&srchindex=1&
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
```

And then run the SELECT again.

© Brent Ozar Unlimited®. All rights reserved.

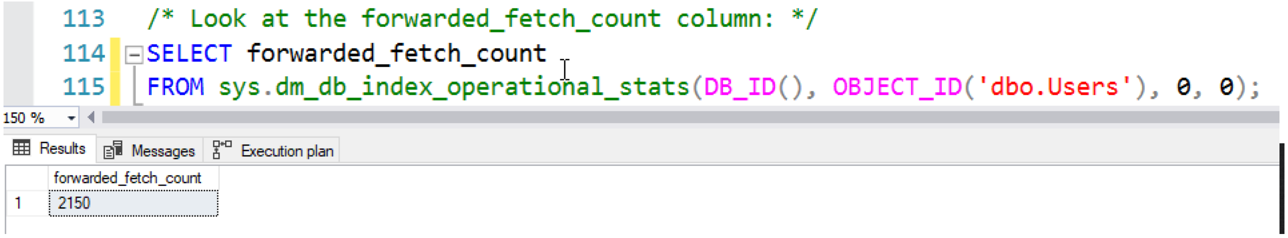
The numbers keep going up



```
/* Now, check your logical reads: */
SELECT *
  FROM dbo.Users
  WHERE LastAccessDate >= '2013/11/10'
    AND LastAccessDate <  '2013/11/11';
GO
```

```
 (1146 rows a
Table 'Users'. Scan coun  , logical reads 2228, ph
```

Originally 1,154



```
/* Look at the forwarded_fetch_count column: */
SELECT forwarded_fetch_count
FROM sys.dm_db_index_operational_stats(DB_ID(), OBJECT_ID('dbo.Users'), 0, 0);
```

	forwarded_fetch_count
1	2150

Is this a problem with heaps?

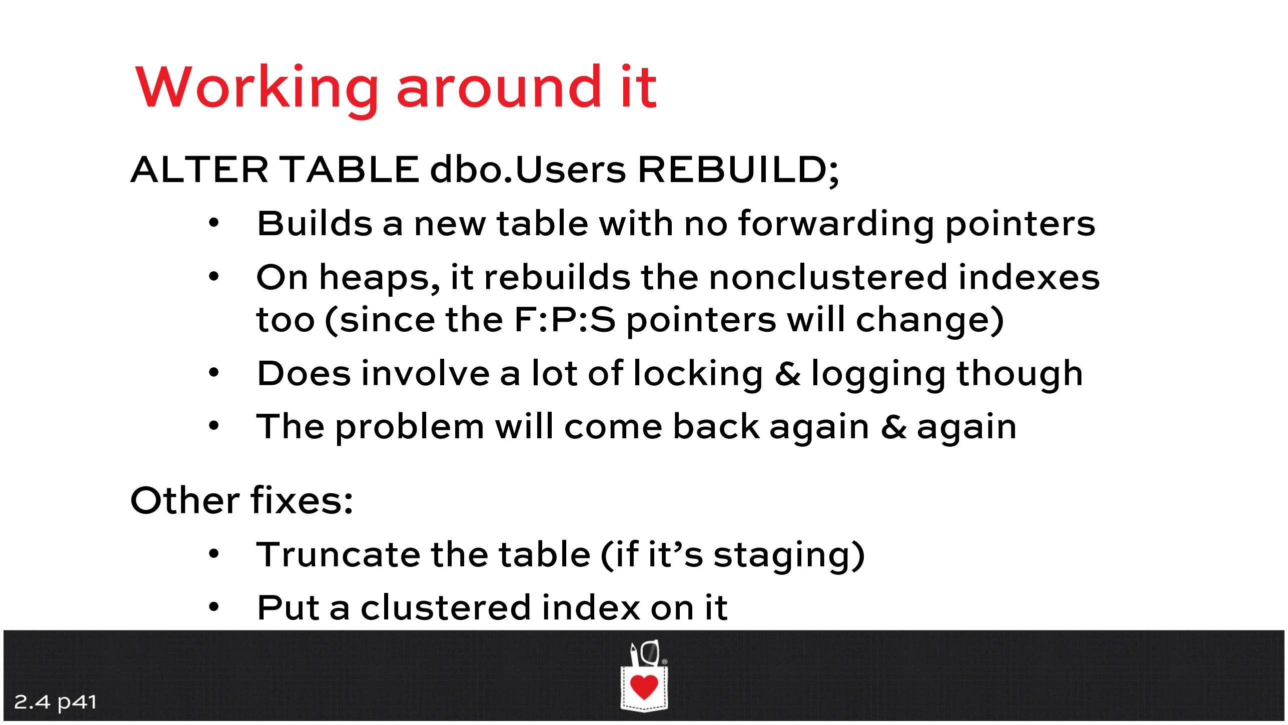
Technically, no. Microsoft *could* choose to fix this.

The decision not to update the F:P:S on each nonclustered index is an implementation decision.

Microsoft’s design makes for faster updates, but slower reads. It’s a design tradeoff.

If you do updates, especially on variable-length columns, heaps are usually a bad idea.

© Brent Ozar Unlimited®. All rights reserved.



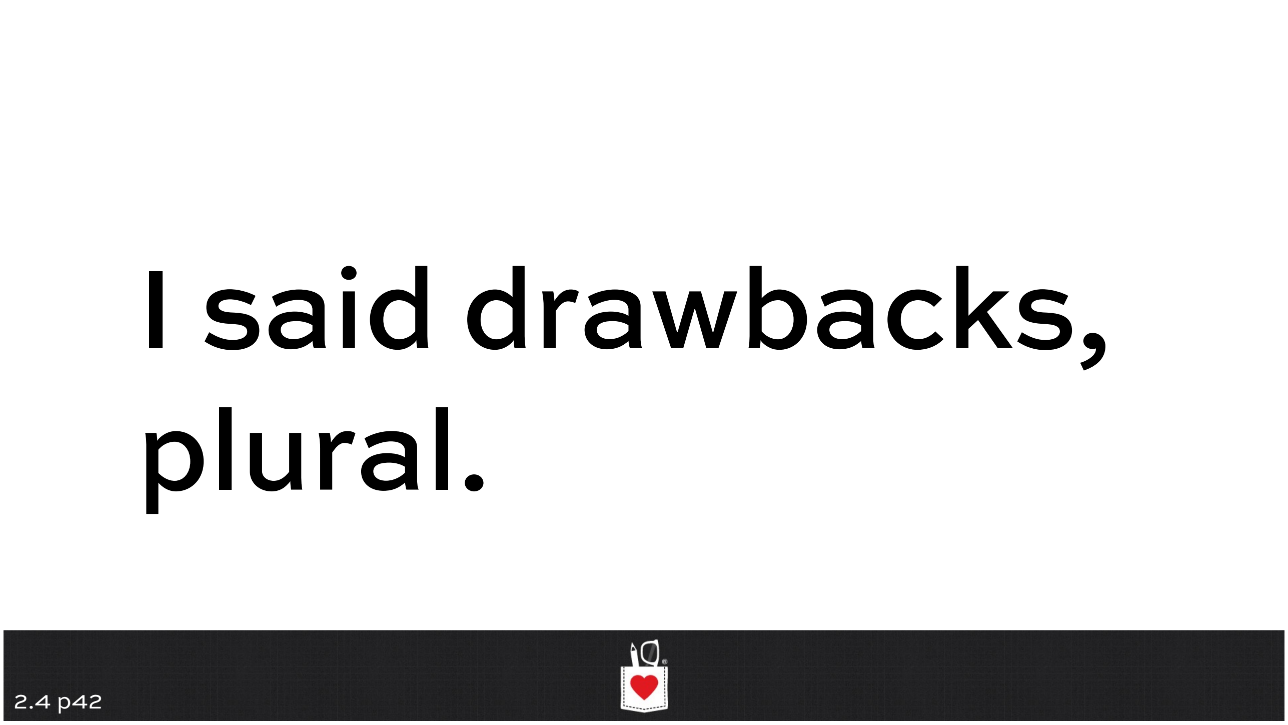
Working around it

ALTER TABLE dbo.Users REBUILD;

- Builds a new table with no forwarding pointers
- On heaps, it rebuilds the nonclustered indexes too (since the F:P:S pointers will change)
- Does involve a lot of locking & logging though
- The problem will come back again & again

Other fixes:

- Truncate the table (if it’s staging)
- Put a clustered index on it

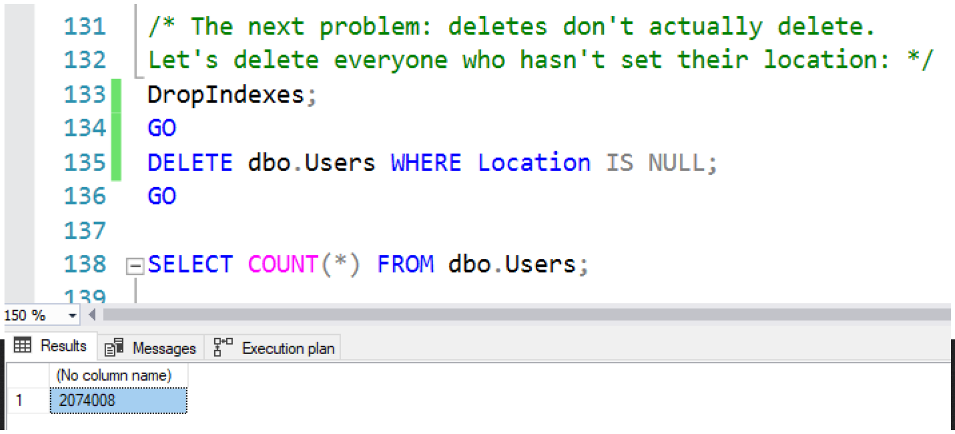


I said drawbacks, plural.

© Brent Ozar Unlimited®. All rights reserved.

Next drawback: deletes don’t.

Drop the nonclustered indexes, delete a bunch of users, then run a COUNT(*). 2M rows are left:



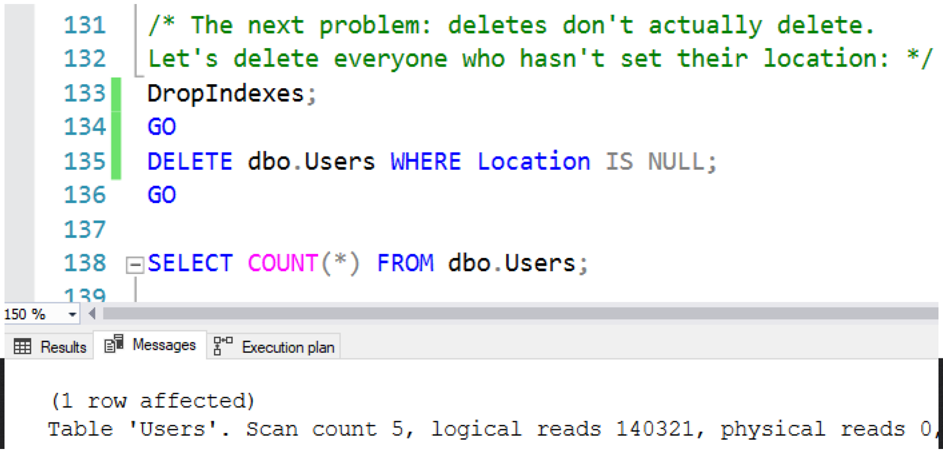
```
/* The next problem: deletes don't actually delete.
Let's delete everyone who hasn't set their location: */
DropIndexes;
GO
DELETE dbo.Users WHERE Location IS NULL;
GO

SELECT COUNT(*) FROM dbo.Users;
```

	(No column name)
1	2074008

2.4 p43

How many reads does it do?

140K reads to read 2M rows. Is that a lot?



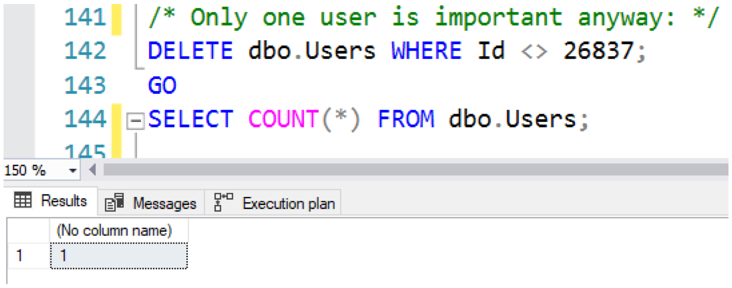
```
/* The next problem: deletes don't actually delete.
Let's delete everyone who hasn't set their location: */
DropIndexes;
GO
DELETE dbo.Users WHERE Location IS NULL;
GO

SELECT COUNT(*) FROM dbo.Users;
```

```
(1 row affected)
Table 'Users'. Scan count 5, logical reads 140321, physical reads 0,
```

2.4 p44

© Brent Ozar Unlimited®. All rights reserved.

Delete all but 1 row.

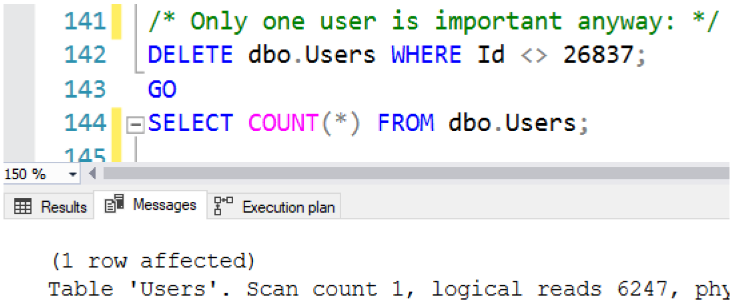


```
/* Only one user is important anyway: */
DELETE dbo.Users WHERE Id <> 26837;
GO
SELECT COUNT(*) FROM dbo.Users;
```

	(No column name)
1	1

And then check your logical reads…

6,247 page reads for 1 row?!?



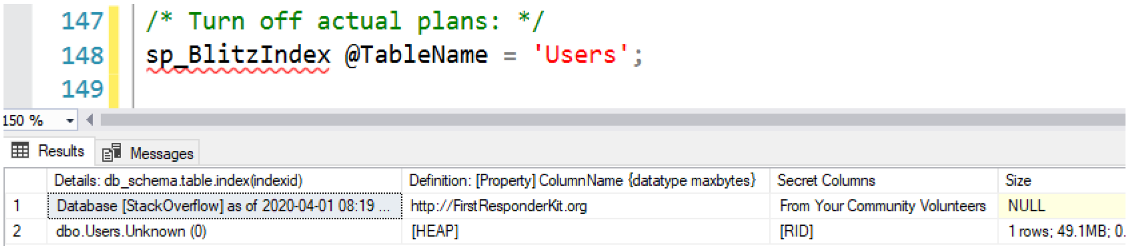
```
/* Only one user is important anyway: */
DELETE dbo.Users WHERE Id <> 26837;
GO
SELECT COUNT(*) FROM dbo.Users;
```

```
(1 row affected)
Table 'Users'. Scan count 1, logical reads 6247, phy
```

How can one user span 6,247 pages?
(I don’t have a big AboutMe, either.)

© Brent Ozar Unlimited®. All rights reserved.

How much space is Users taking?



```
/* Turn off actual plans: */
sp_BlitzIndex @TableName = 'Users';
```

	Details: db_schema.table.index(indexid)	Definition: [Property] ColumnName {datatype maxbytes}	Secret Columns	Size
1	Database [StackOverflow] as of 2020-04-01 08:19 ...	http://FirstResponderKit.org	From Your Community Volunteers	NULL
2	dbo.Users.Unknown (0)	[HEAP]	[RID]	1 rows; 49.1MB; 0.

The table has 49MB allocated for just one row!

What’s going on

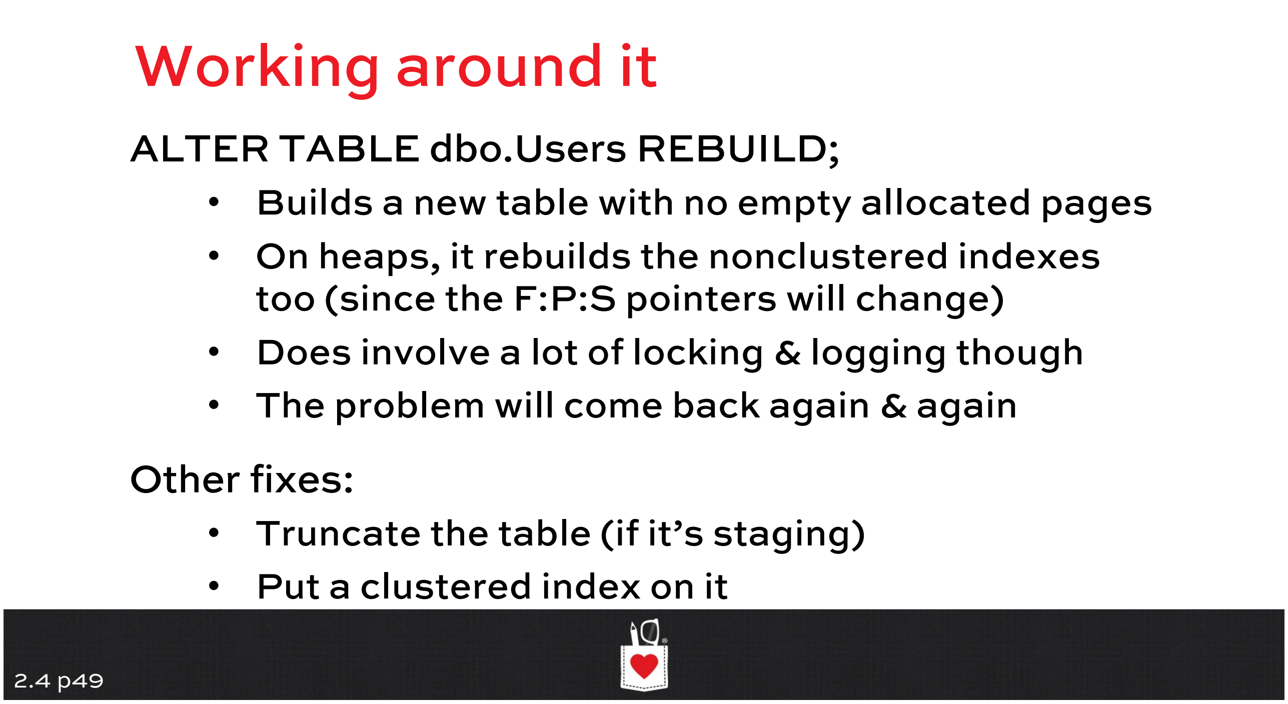
In a heap, deletes don’t deallocate all empty pages.

Heaps are optimized for fast loads.

SQL Server assumes you’re still going to want to load data again soon, so it leaves the pages allocated.

The bad scenario for heaps:
if you do deletes, and then
select from the same table.
Extra reads are incurred.

© Brent Ozar Unlimited®. All rights reserved.



Working around it

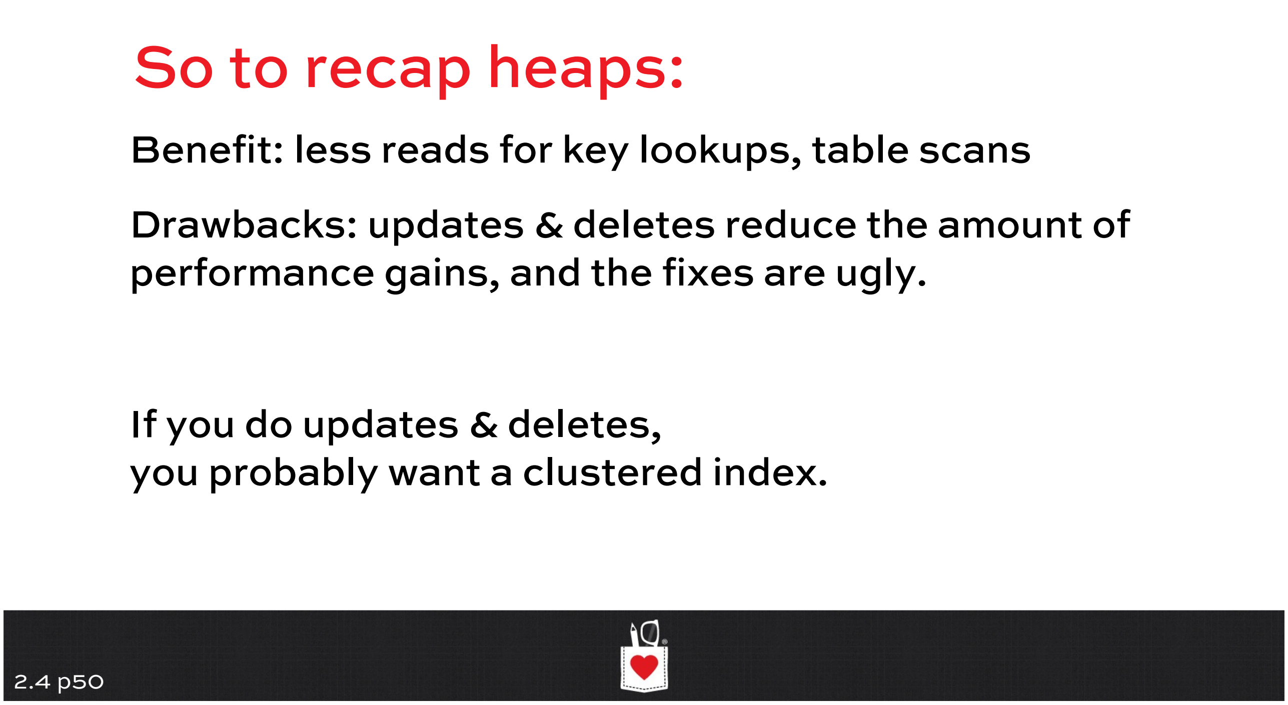
ALTER TABLE dbo.Users REBUILD;

- Builds a new table with no empty allocated pages
- On heaps, it rebuilds the nonclustered indexes too (since the F:P:S pointers will change)
- Does involve a lot of locking & logging though
- The problem will come back again & again

Other fixes:

- Truncate the table (if it's staging)
- Put a clustered index on it

2.4 p49



So to recap heaps:

Benefit: less reads for key lookups, table scans

Drawbacks: updates & deletes reduce the amount of performance gains, and the fixes are ugly.

If you do updates & deletes,
you probably want a clustered index.

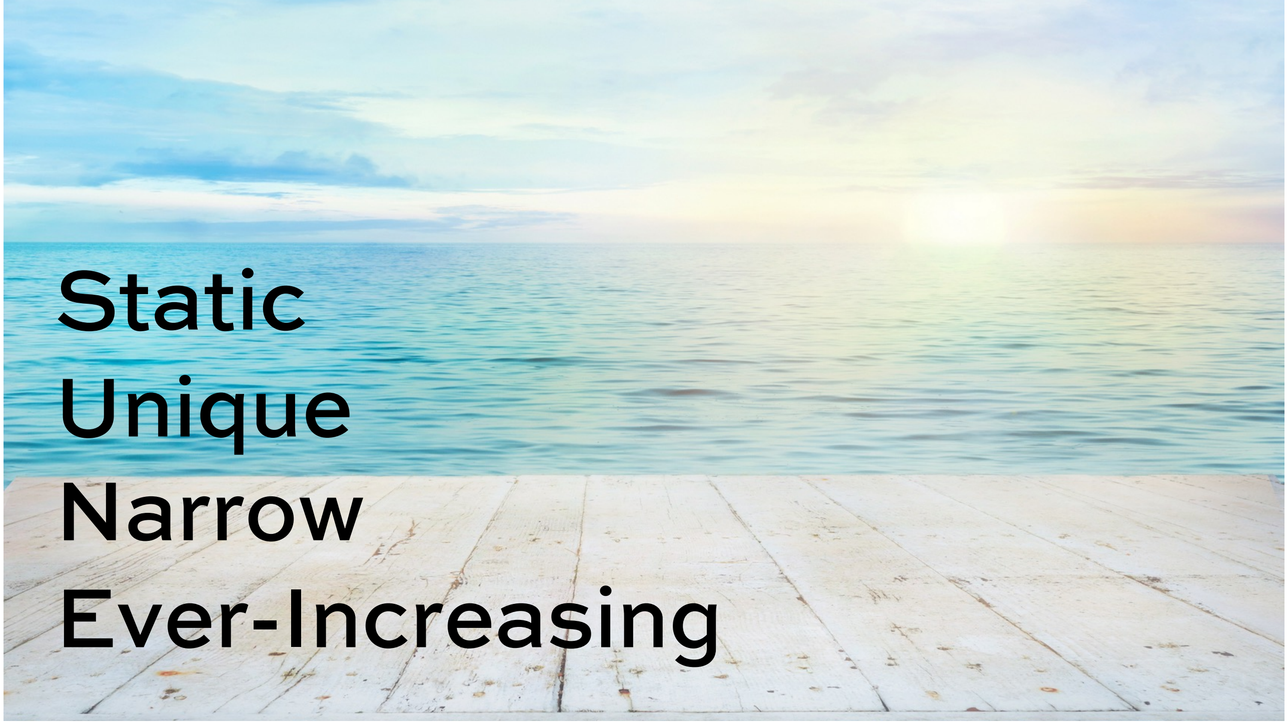
2.4 p50

© Brent Ozar Unlimited®. All rights reserved.

AdventureWorks' b-tree is bad at the roots

Designing good clustering keys

2.4 p51



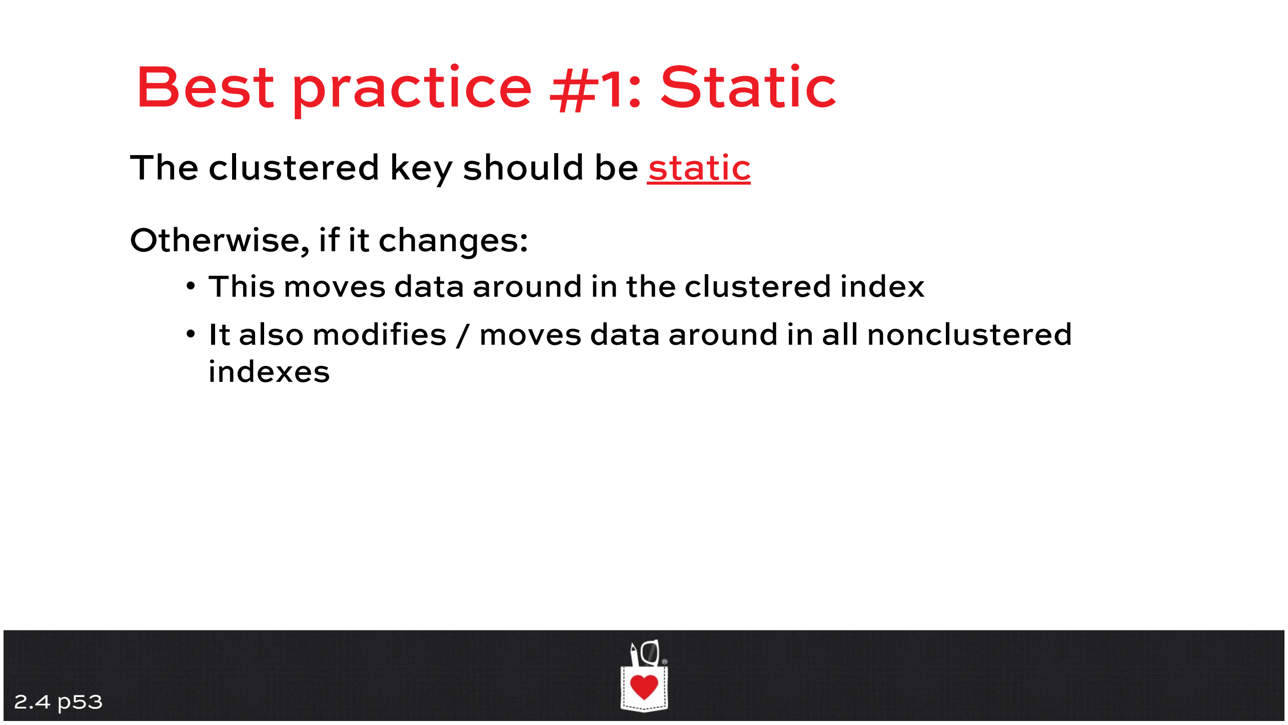
Static

Unique

Narrow

Ever-Increasing

© Brent Ozar Unlimited®. All rights reserved.

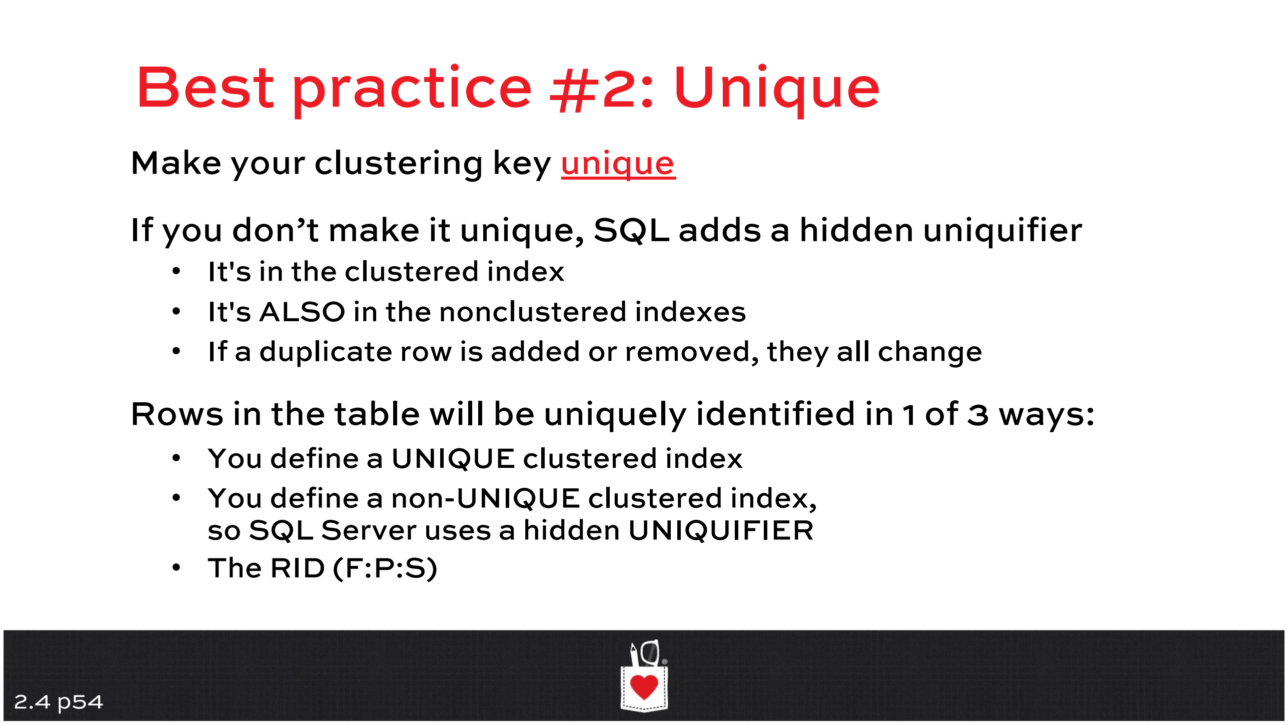


Best practice #1: Static

The clustered key should be static

Otherwise, if it changes:

- This moves data around in the clustered index
- It also modifies / moves data around in all nonclustered indexes



Best practice #2: Unique

Make your clustering key unique

If you don’t make it unique, SQL adds a hidden uniquifier

- It's in the clustered index
- It's ALSO in the nonclustered indexes
- If a duplicate row is added or removed, they all change

Rows in the table will be uniquely identified in 1 of 3 ways:

- You define a UNIQUE clustered index
- You define a non-UNIQUE clustered index, so SQL Server uses a hidden UNIQUIFIER
- The RID (F:P:S)

© Brent Ozar Unlimited®. All rights reserved.

Best practice #3: Narrow

Keep the clustered key as narrow as you can. Consider both:

- Data type
- Number of columns

Why?

- The wider it is, the wider your nonclustered indexes
- More space on memory, more space on disk
- More IO

How "narrow" is narrow?

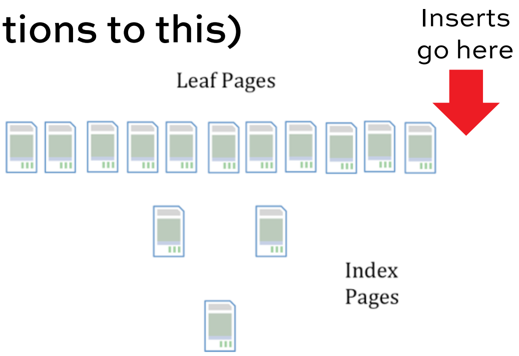
2.4 p55

Best practice #4: Ever-Increasing

The clustered key should be ever-increasing

- This keeps it from getting fragmented quickly

(There are often exceptions to this)



2.4 p56

© Brent Ozar Unlimited®. All rights reserved.

Secret columns...

...are the reason for these three best practices:

1. Static: Or you do secret writes on EVERY index
2. Unique: or you get uniquifier overhead
3. Narrow: or you add bloated size to all your NC indexes

2.4 p57

Overheard

“Always use an identity column as your clustering key”

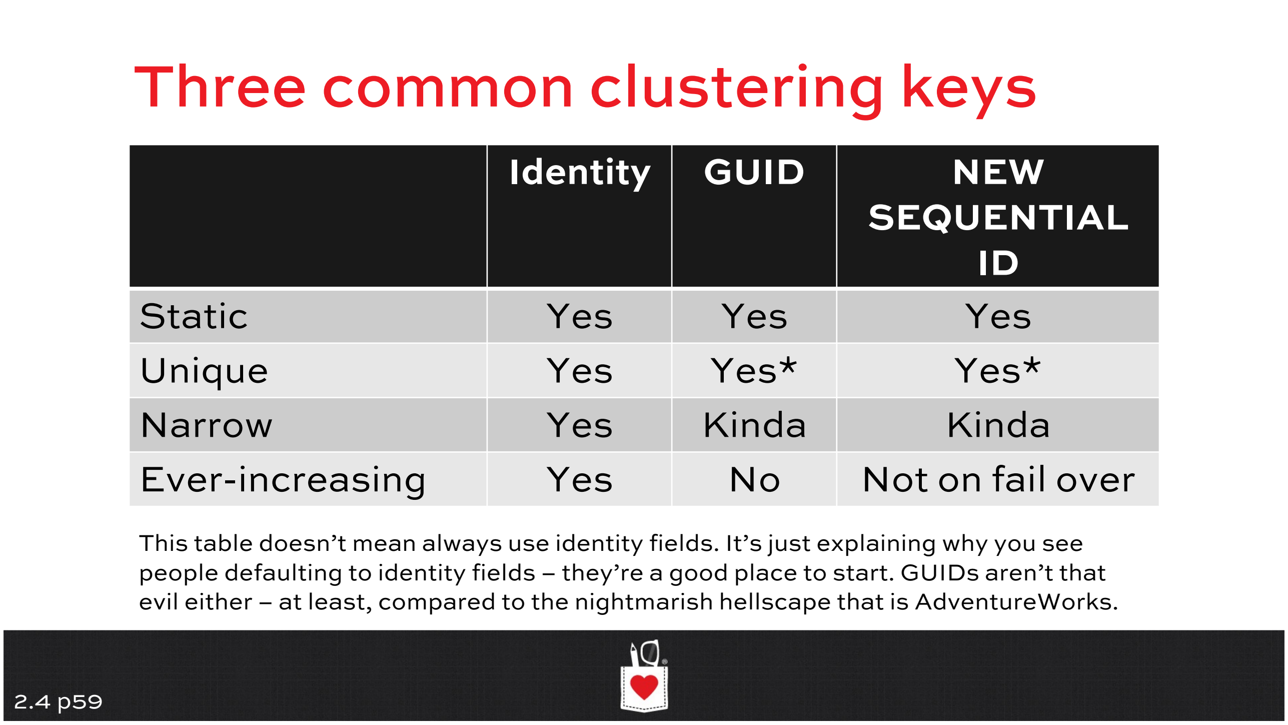
“Always use a surrogate key”

“Always use a one column key”

“Never use GUIDs”

2.4 p58

© Brent Ozar Unlimited®. All rights reserved.

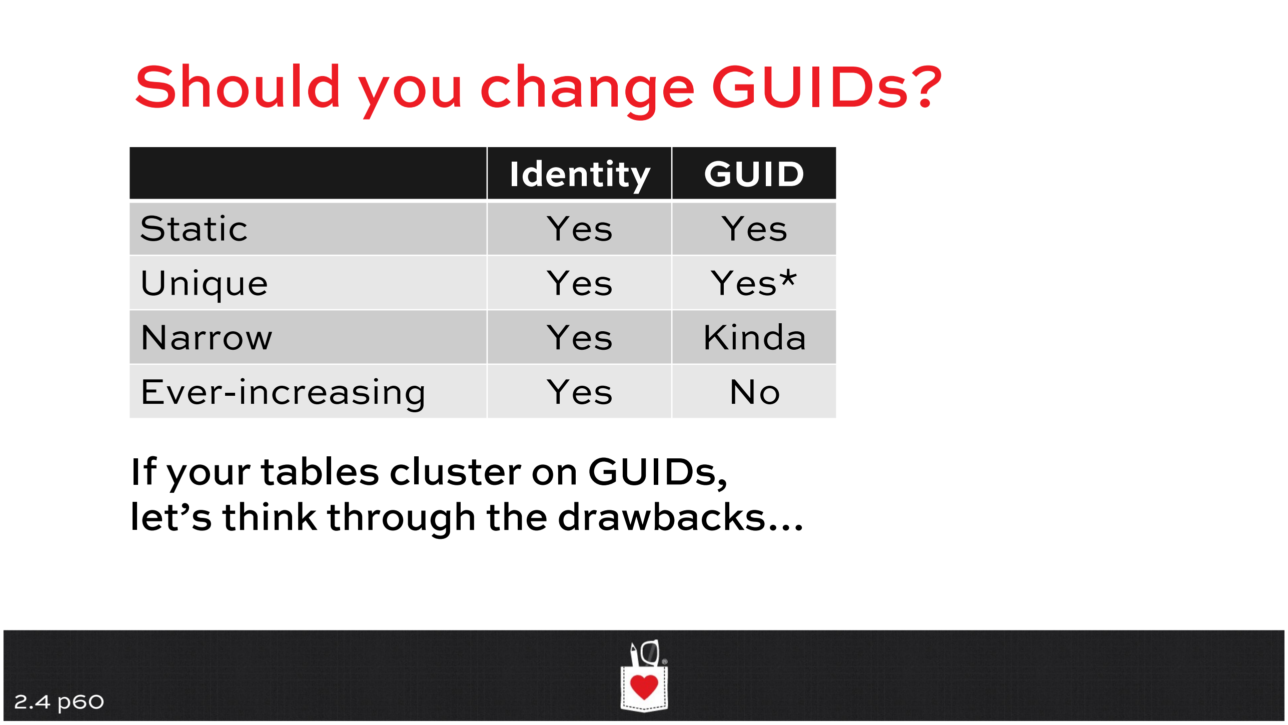


Three common clustering keys

	Identity	GUID	NEW SEQUENTIAL ID
Static	Yes	Yes	Yes
Unique	Yes	Yes*	Yes*
Narrow	Yes	Kinda	Kinda
Ever-increasing	Yes	No	Not on fail over

This table doesn't mean always use identity fields. It's just explaining why you see people defaulting to identity fields – they're a good place to start. GUIDs aren't that evil either – at least, compared to the nightmarish hellscape that is AdventureWorks.

2.4 p59



Should you change GUIDs?

	Identity	GUID
Static	Yes	Yes
Unique	Yes	Yes*
Narrow	Yes	Kinda
Ever-increasing	Yes	No

If your tables cluster on GUIDs,
let's think through the drawbacks…

2.4 p60

© Brent Ozar Unlimited®. All rights reserved.

GUIDs aren’t narrow.

GUID: 16 bytes per row. BIGINT: 8. INT: 4.

So you could save 8-12 bytes per row, per index, by changing from a GUID to a smaller data type.

For a 1 million row table,
that’s an 8MB savings per index.

For a 1 billion row table with 10 indexes,
that’s an 80GB savings – but painful to get.

2.4 p61

GUIDs aren’t ever-increasing.

Inserts will be randomly scattered through the table.

That leads to fragmentation. But… it’s not a big deal:

https://www.brentozar.com/archive/2022/08/video-fragmentation-explained-in-20-minutes-at-sqlbits/

https://www.youtube.com/watch?v=iEa6_QnCFMU

2.4 p62

© Brent Ozar Unlimited®. All rights reserved.

So should you “fix” GUID keys?

On smaller tables… nobody’s going to notice the fix.

On larger tables, you can save space, but the fix is going to suck:

- Dropping foreign keys, primary keys
- Breaking replication
- Tons of logged disk activity, AG traffic
- Long periods of blocking

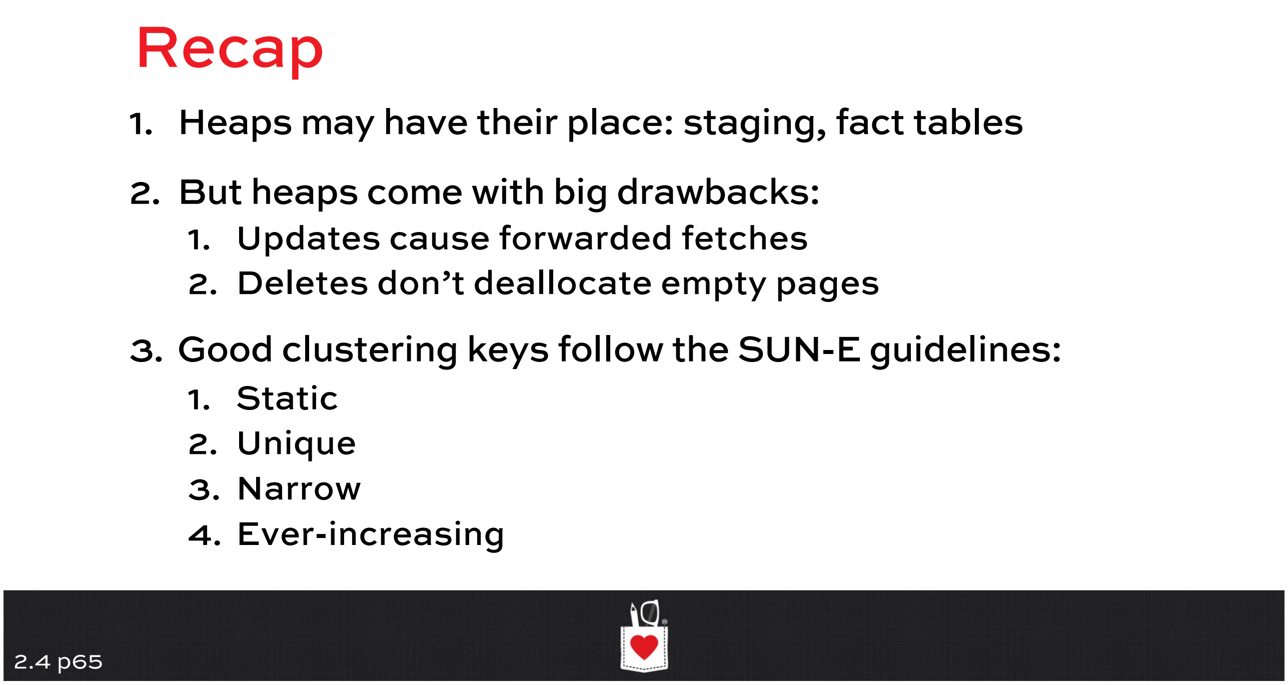
So fix growing tables that will be large in a few years.

2.4 p63

Three things we learned

2.4 p64

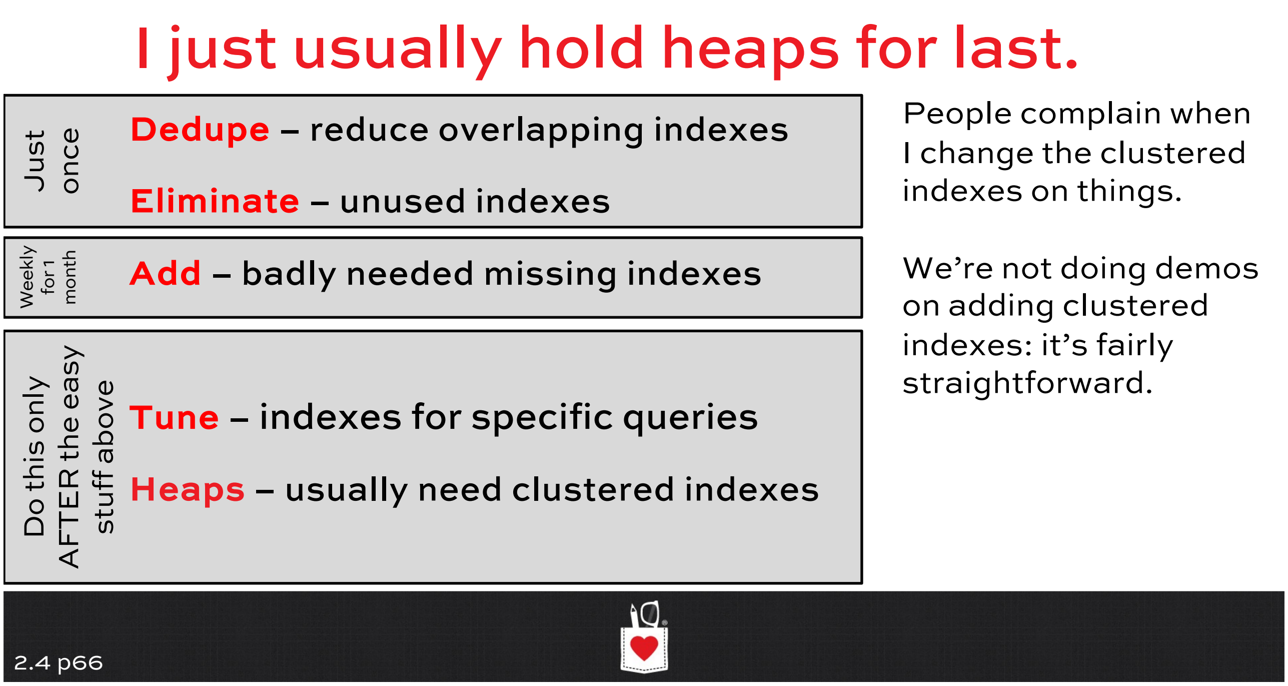
© Brent Ozar Unlimited®. All rights reserved.



Recap

1. Heaps may have their place: staging, fact tables
2. But heaps come with big drawbacks:
   1. Updates cause forwarded fetches
   2. Deletes don't deallocate empty pages
3. Good clustering keys follow the SUN-E guidelines:
   1. Static
   2. Unique
   3. Narrow
   4. Ever-increasing

2.4 p65



I just usually hold heaps for last.

Just once	**Dedupe** – reduce overlapping indexes **Eliminate** – unused indexes
Weekly for 1 month	**Add** – badly needed missing indexes
Do this only AFTER the easy stuff above	**Tune** – indexes for specific queries **Heaps** – usually need clustered indexes

People complain when I change the clustered indexes on things.

We're not doing demos on adding clustered indexes: it's fairly straightforward.

2.4 p66

© Brent Ozar Unlimited®. All rights reserved.