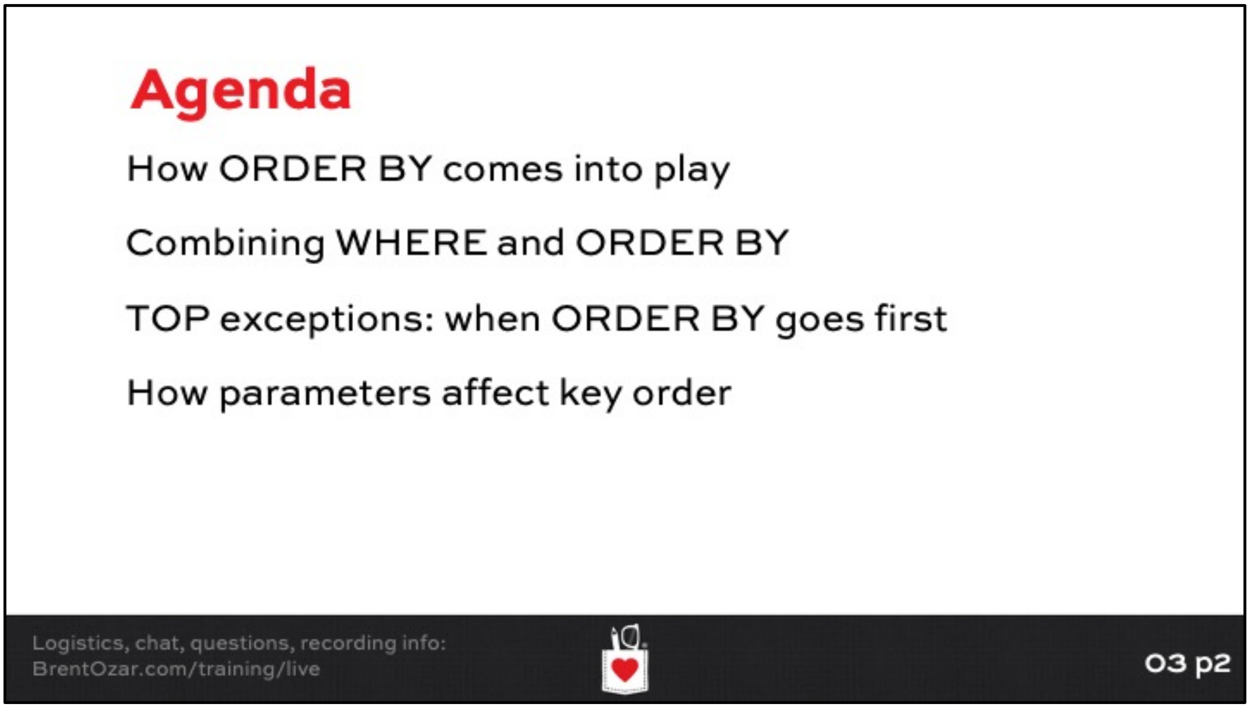
BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Part 3: Indexing for ORDER BY

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

03 p1



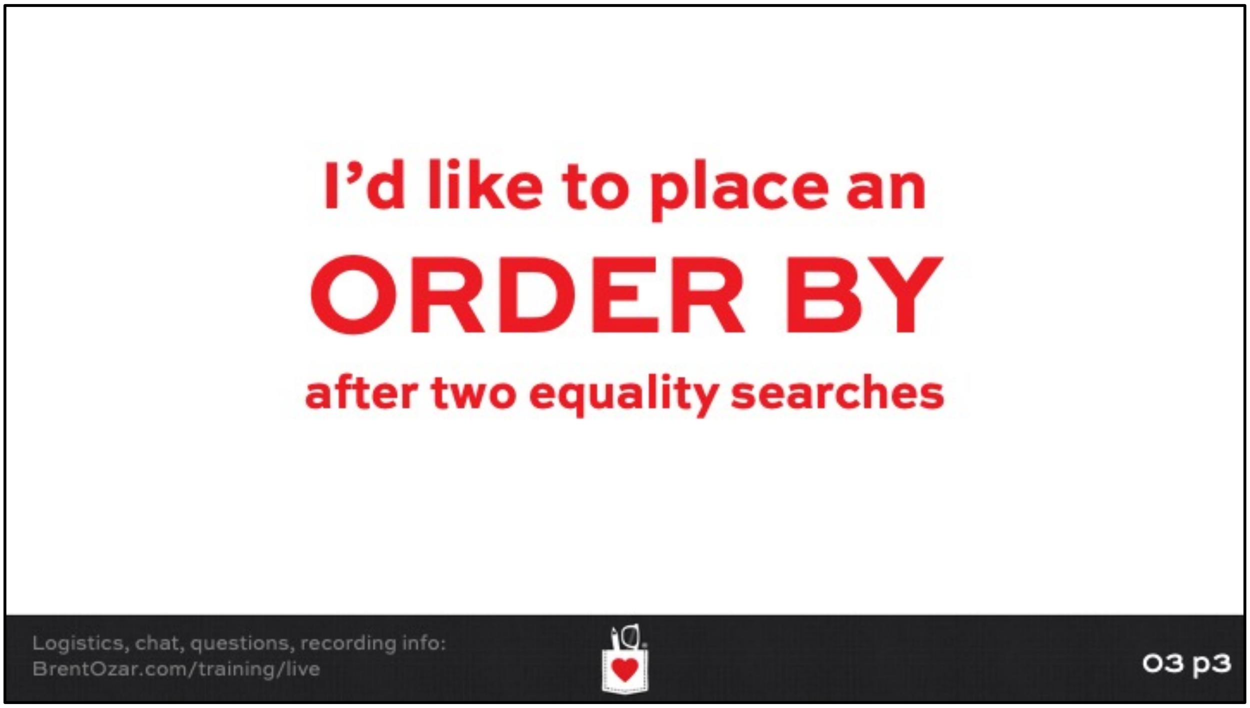
Agenda

How ORDER BY comes into play

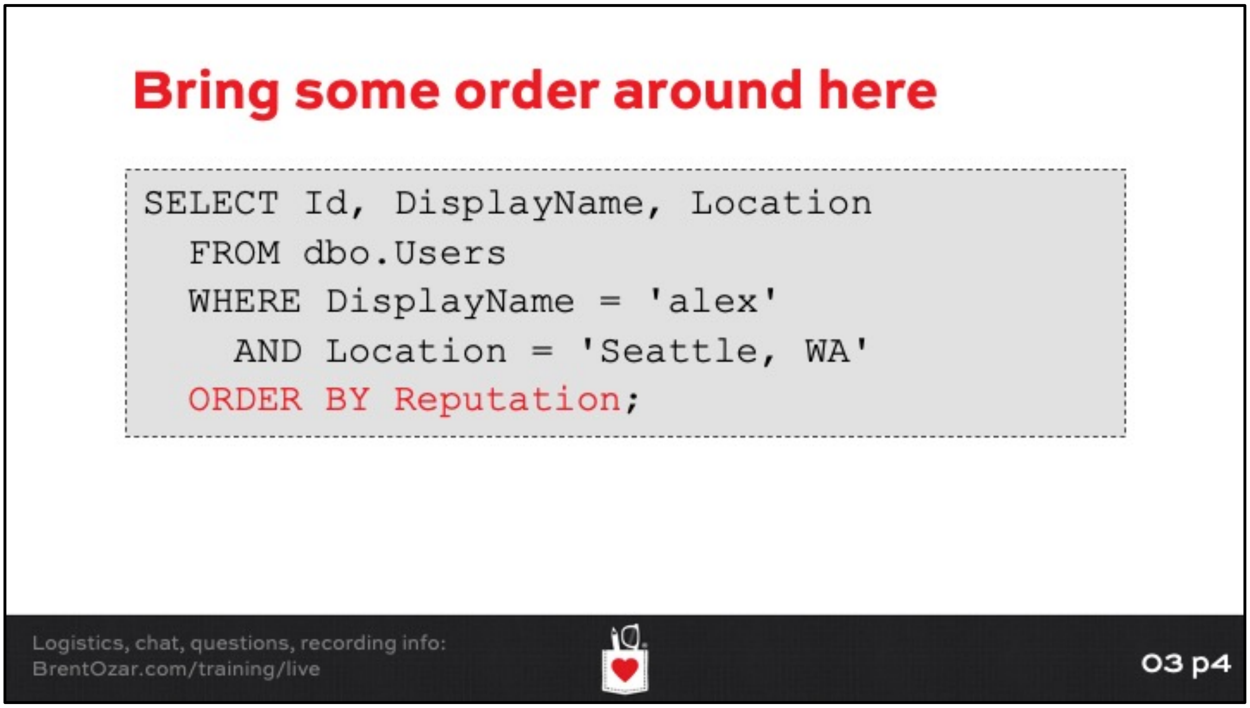
Combining WHERE and ORDER BY

TOP exceptions: when ORDER BY goes first

How parameters affect key order



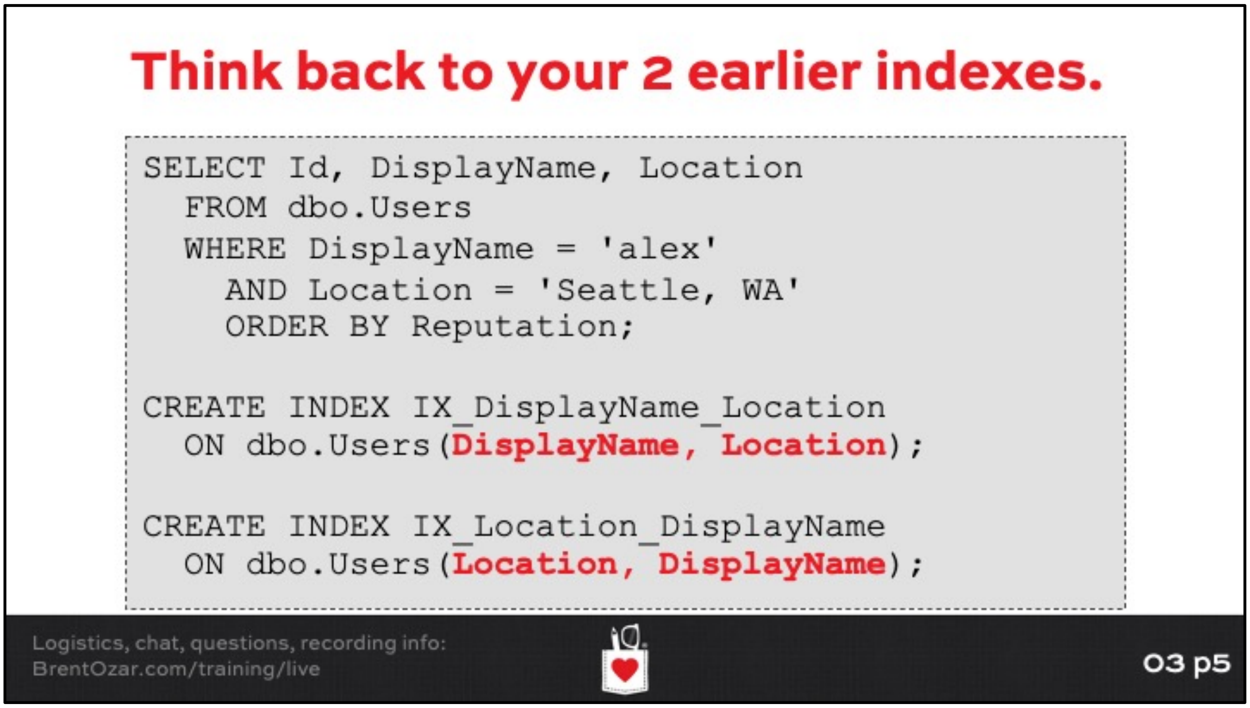
I’d like to place an ORDER BY after two equality searches



Bring some order around here

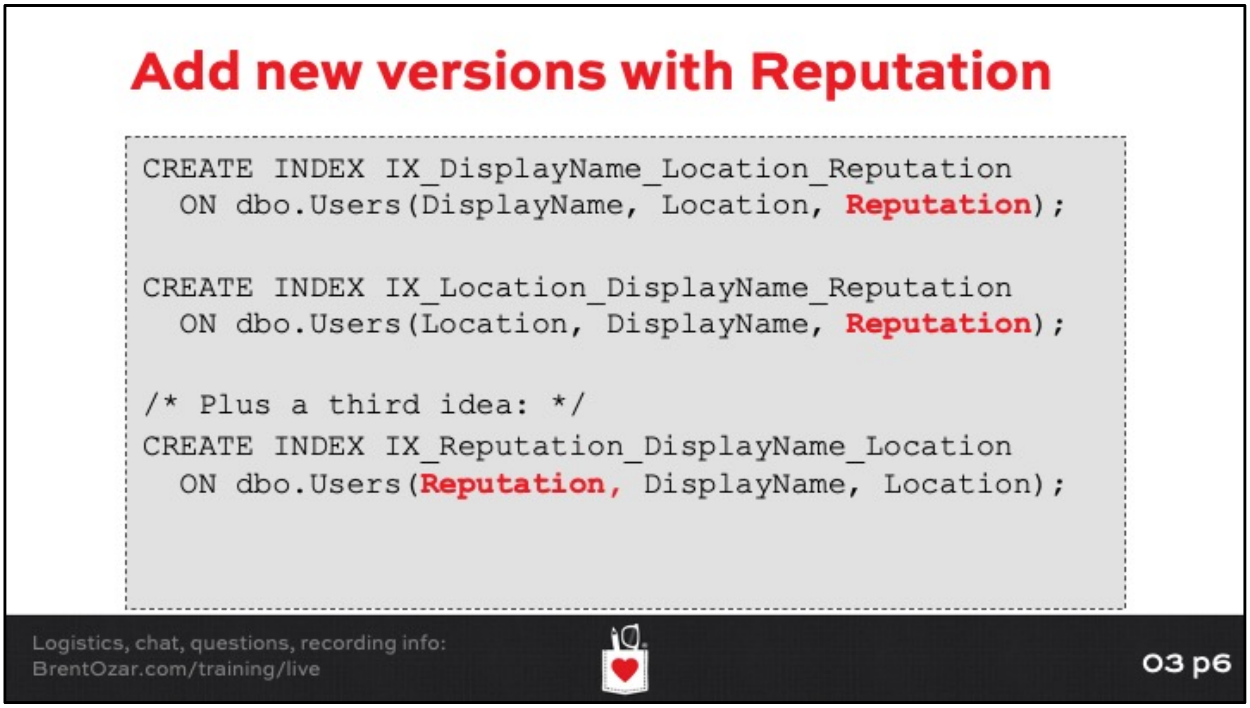
```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location = 'Seattle, WA'
  ORDER BY Reputation;
```

Logistics, chat, questions, recording info:
BrentOzar.com/training/live



Think back to your 2 earlier indexes.

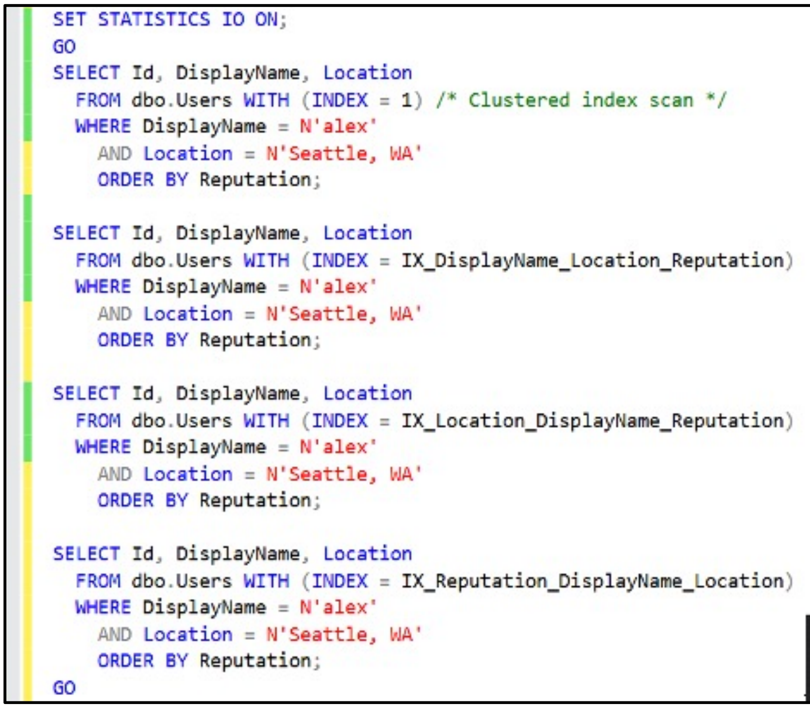
```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location = 'Seattle, WA'
    ORDER BY Reputation;

CREATE INDEX IX_DisplayName_Location
  ON dbo.Users(DisplayName, Location);

CREATE INDEX IX_Location_DisplayName
  ON dbo.Users(Location, DisplayName);
```



Add new versions with Reputation

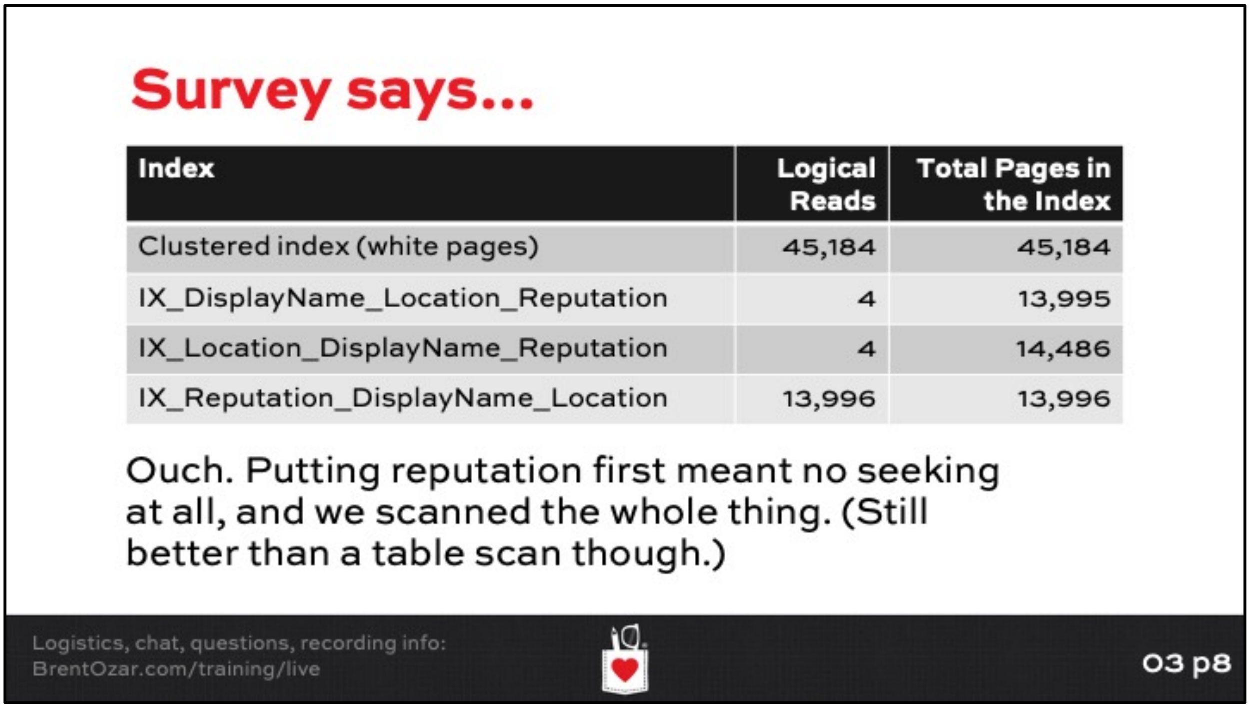
```
CREATE INDEX IX_DisplayName_Location_Reputation
  ON dbo.Users(DisplayName, Location, Reputation);

CREATE INDEX IX_Location_DisplayName_Reputation
  ON dbo.Users(Location, DisplayName, Reputation);

/* Plus a third idea: */
CREATE INDEX IX_Reputation_DisplayName_Location
  ON dbo.Users(Reputation, DisplayName, Location);
```

Test ‘em



```
SET STATISTICS IO ON;
GO
SELECT Id, DisplayName, Location
  FROM dbo.Users WITH (INDEX = 1) /* Clustered index scan */
  WHERE DisplayName = N'alex'
    AND Location = N'Seattle, WA'
    ORDER BY Reputation;

SELECT Id, DisplayName, Location
  FROM dbo.Users WITH (INDEX = IX_DisplayName_Location_Reputation)
  WHERE DisplayName = N'alex'
    AND Location = N'Seattle, WA'
    ORDER BY Reputation;

SELECT Id, DisplayName, Location
  FROM dbo.Users WITH (INDEX = IX_Location_DisplayName_Reputation)
  WHERE DisplayName = N'alex'
    AND Location = N'Seattle, WA'
    ORDER BY Reputation;

SELECT Id, DisplayName, Location
  FROM dbo.Users WITH (INDEX = IX_Reputation_DisplayName_Location)
  WHERE DisplayName = N'alex'
    AND Location = N'Seattle, WA'
    ORDER BY Reputation;
GO
```

03 p7

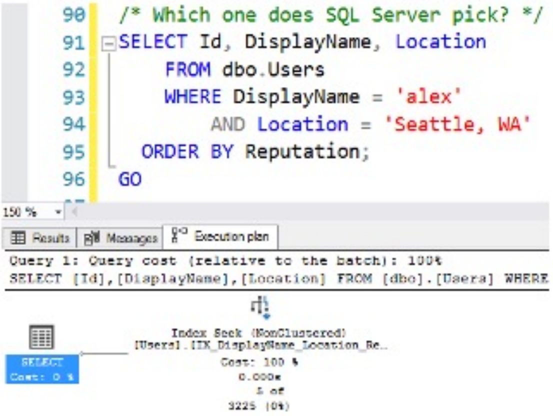


Survey says…

Index	Logical Reads	Total Pages in the Index
Clustered index (white pages)	45,184	45,184
IX_DisplayName_Location_Reputation	4	13,995
IX_Location_DisplayName_Reputation	4	14,486
IX_Reputation_DisplayName_Location	13,996	13,996

Ouch. Putting reputation first meant no seeking at all, and we scanned the whole thing. (Still better than a table scan though.)

Which one does SQL Server pick?



```
/* Which one does SQL Server pick? */
SELECT Id, DisplayName, Location
    FROM dbo.Users
    WHERE DisplayName = 'alex'
        AND Location = 'Seattle, WA'
  ORDER BY Reputation;
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT [Id],[DisplayName],[Location] FROM [dbo].[Users] WHERE
```

SELECT
Cost: 0 %
Index Seek (NonClustered)
[Users].[IX_DisplayName_Location_Re…
Cost: 100 %
0.000s
1 of
3225 (0%)

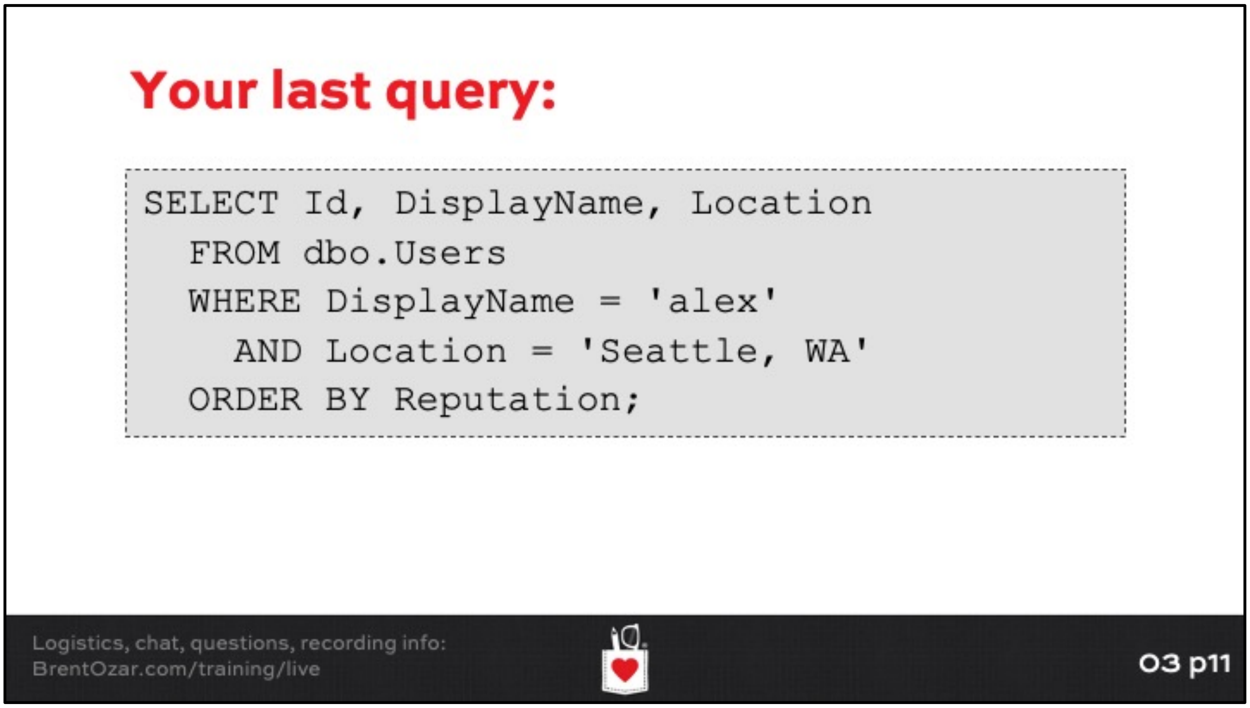
The one that leads with DisplayName.

ORDER BY

after an INequality search

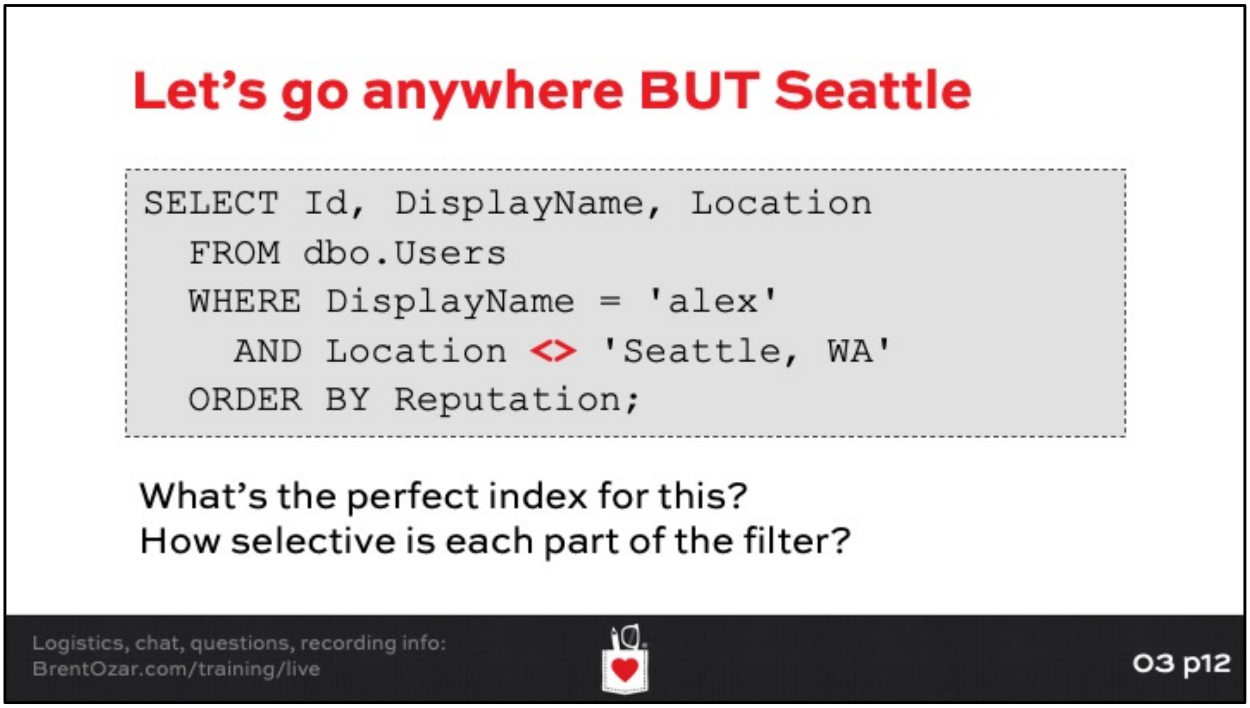
Logistics, chat, questions, recording info:
BrentOzar.com/training/live

03 p10



Your last query:

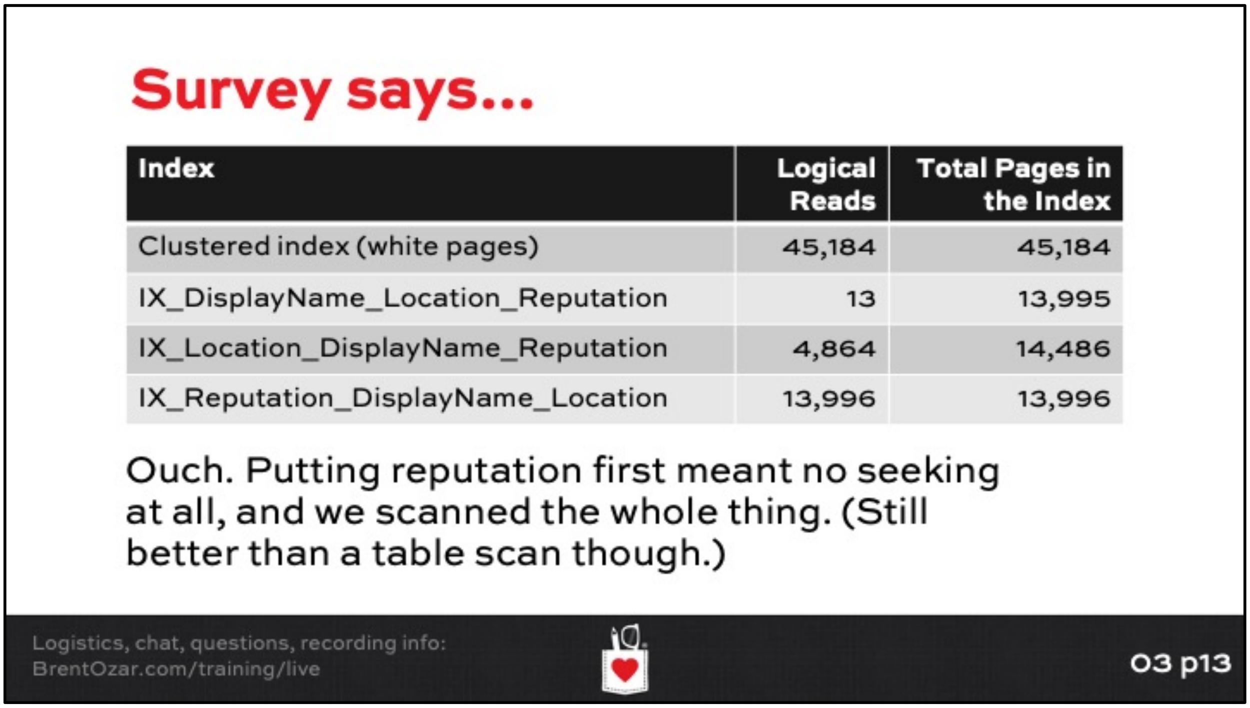
```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location = 'Seattle, WA'
  ORDER BY Reputation;
```



Let’s go anywhere BUT Seattle

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location <> 'Seattle, WA'
  ORDER BY Reputation;
```

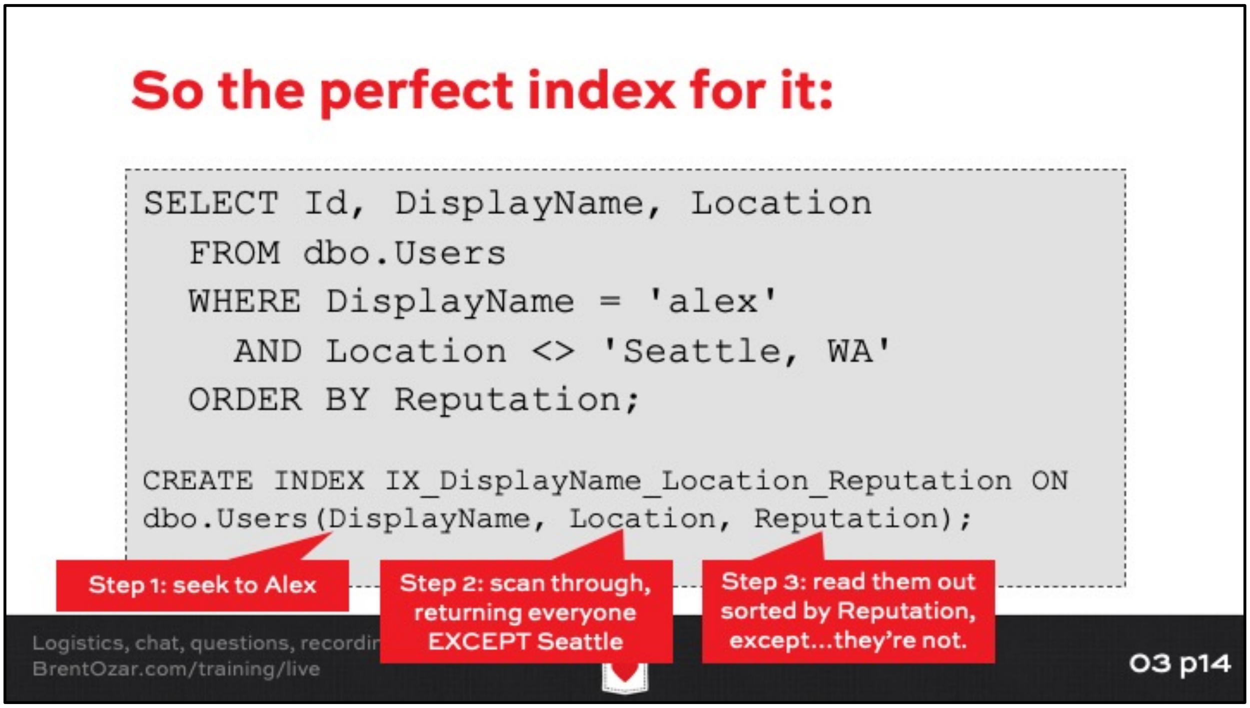
What’s the perfect index for this?
How selective is each part of the filter?



Survey says…

Index	Logical Reads	Total Pages in the Index
Clustered index (white pages)	45,184	45,184
IX_DisplayName_Location_Reputation	13	13,995
IX_Location_DisplayName_Reputation	4,864	14,486
IX_Reputation_DisplayName_Location	13,996	13,996

Ouch. Putting reputation first meant no seeking at all, and we scanned the whole thing. (Still better than a table scan though.)



So the perfect index for it:

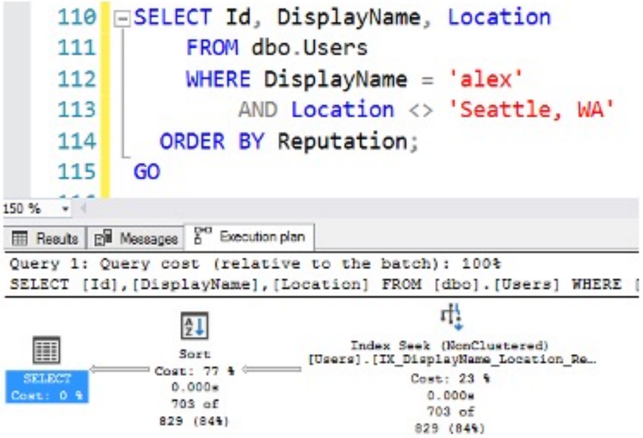
```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location <> 'Seattle, WA'
  ORDER BY Reputation;

CREATE INDEX IX_DisplayName_Location_Reputation ON
dbo.Users(DisplayName, Location, Reputation);
```

Step 1: seek to Alex

Step 2: scan through, returning everyone EXCEPT Seattle

Step 3: read them out sorted by Reputation, except…they're not.

Our index gets used, but...



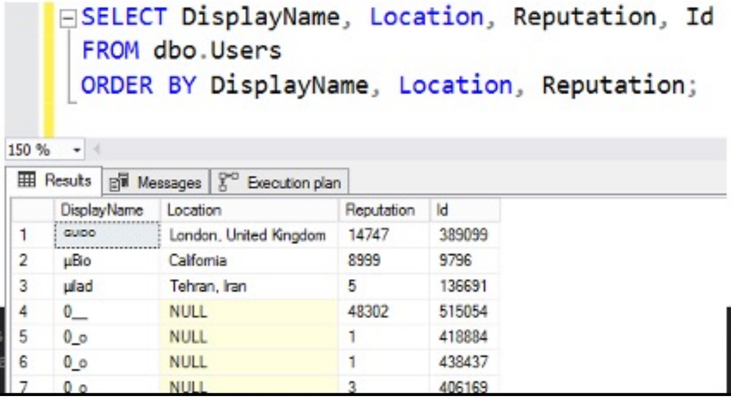
```
SELECT Id, DisplayName, Location
    FROM dbo.Users
    WHERE DisplayName = 'alex'
        AND Location <> 'Seattle, WA'
  ORDER BY Reputation;
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT [Id],[DisplayName],[Location] FROM [dbo].[Users] WHERE [
```

The plan has a Sort even though the data in the index is sorted in order – isn't it?

Write a query to visualize the index

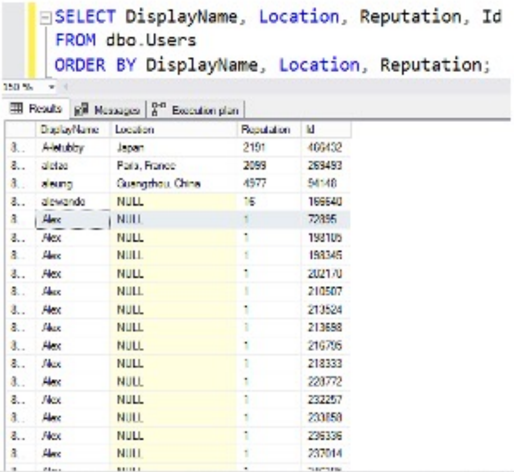
```
CREATE INDEX IX_DisplayName_Location_Reputation ON
dbo.Users(DisplayName, Location, Reputation);
```



```
SELECT DisplayName, Location, Reputation, Id
FROM dbo.Users
ORDER BY DisplayName, Location, Reputation;
```

	DisplayName	Location	Reputation	Id
1	ɢuᴅo	London, United Kingdom	14747	389099
2	µBio	Calfornia	8999	9796
3	µlad	Tehran, Iran	5	136691
4	0__	NULL	48302	515054
5	0_o	NULL	1	418884
6	0_o	NULL	1	438437
7	0_o	NULL	3	406169

Seek down to Alex

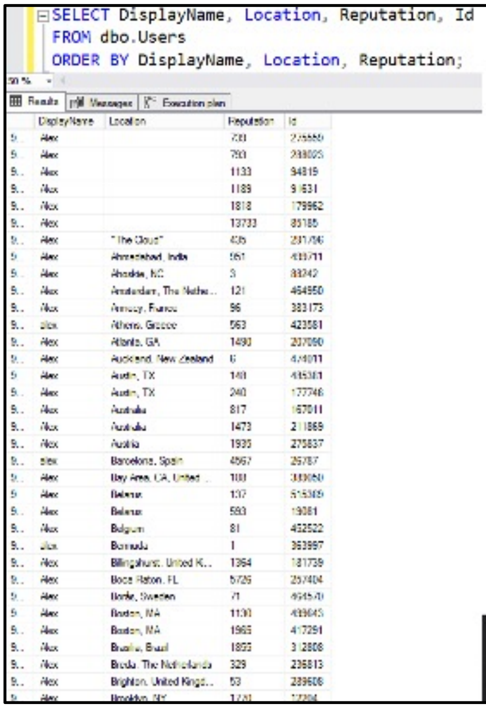


```sql
SELECT DisplayName, Location, Reputation, Id
FROM dbo.Users
ORDER BY DisplayName, Location, Reputation;
```

	DisplayName	Location	Reputation	Id
8..	A-letubby	Japan	2191	406432
8..	alotaz	Paris, France	2099	269493
8..	aleung	Guangzhou, China	4977	94148
8..	alewando	NULL	16	166640
8	Alex	NULL	1	72895
8..	Alex	NULL	1	198105
8..	Alex	NULL	1	198345
8..	Alex	NULL	1	202170
8..	Alex	NULL	1	210507
8..	Alex	NULL	1	213524
8..	Alex	NULL	1	213698
8..	Alex	NULL	1	216705
8..	Alex	NULL	1	218333
8..	Alex	NULL	1	220772
8..	Alex	NULL	1	232257
8..	Alex	NULL	1	233858
8..	Alex	NULL	1	236336
8	Alex	NULL	1	237014

Remember, we need them ordered by Reputation.

At first it looks like this will work, but…



```sql
SELECT DisplayName, Location, Reputation, Id
FROM dbo.Users
ORDER BY DisplayName, Location, Reputation;
```

	DisplayName	Location	Reputation	Id
9...	Alex		730	275559
9...	Alex		793	238025
9...	Alex		1133	94819
9...	Alex		1189	91631
9...	Alex		1818	179962
9...	Alex		13733	85185
9...	Alex	"The Cloud"	435	231786
9...	Alex	Ahmedabad, India	561	439711
9...	Alex	Ahoskie, NC	3	88242
9...	Alex	Amsterdam, The Nethe...	121	464950
9...	Alex	Annecy, France	96	383173
9...	alex	Athens, Greece	563	423581
9...	Alex	Atlanta, GA	1490	207090
9...	Alex	Auckland, New Zealand	6	474011
9...	Alex	Austin, TX	140	435381
9...	Alex	Austin, TX	240	177746
9...	Alex	Australia	817	167011
9...	Alex	Australia	1473	211869
9...	Alex	Austria	1935	275837
9...	alex	Barcelona, Spain	4567	25787
9...	Alex	Bay Area, CA, United ...	100	330060
9...	Alex	Belarus	137	515309
9...	Alex	Belarus	593	19081
9...	Alex	Belgium	81	452522
9...	alex	Bermuda	1	363997
9...	Alex	Billingshurst, United K...	1364	181739
9...	Alex	Boca Raton, FL	5726	257404
9...	Alex	Borås, Sweden	71	494570
9...	Alex	Boston, MA	1130	489645
9...	Alex	Boston, MA	1965	417291
9...	Alex	Brasilia, Brazil	1855	312808
9...	Alex	Breda, The Netherlands	329	236813
9...	Alex	Brighton, United Kingd...	53	289608
9...	Alex	Brooklyn, NY	1770	17704

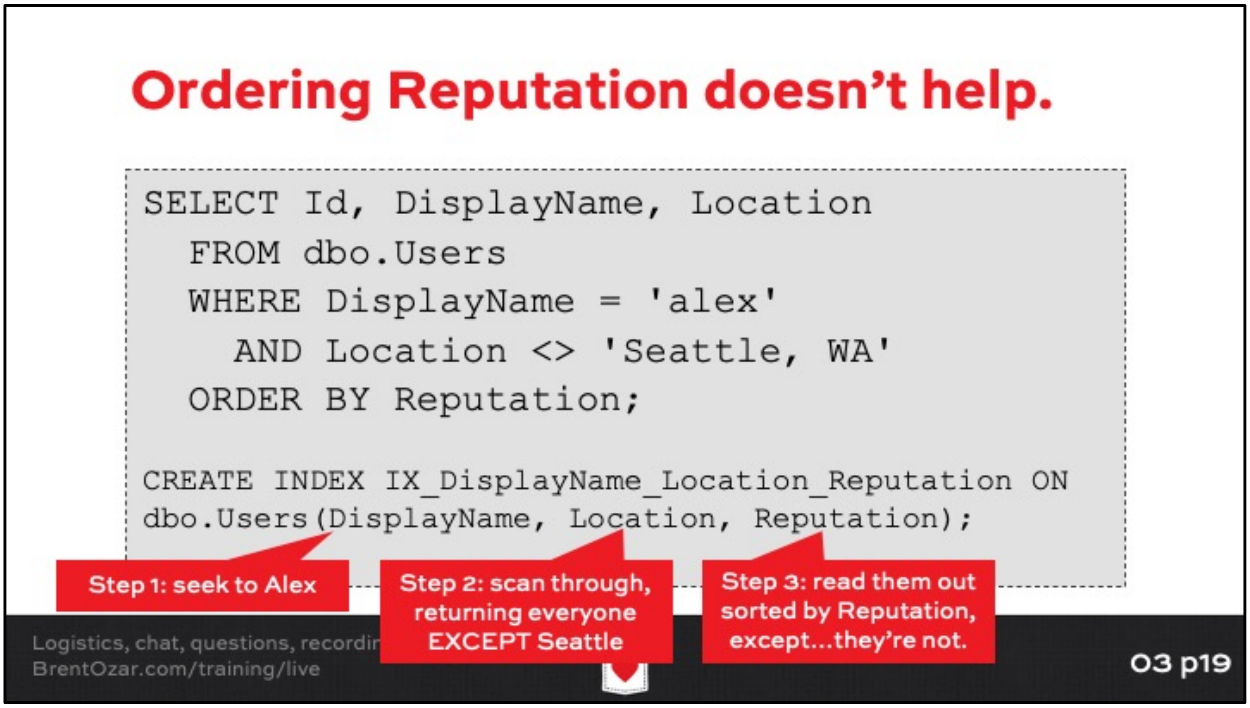
Reputation isn't sorted.

We're going to skip everyone who isn't in Seattle.

That means we need all the Alexes on this screen, plus more.

And they're not sorted by Reputation.

The fact that Reputation is "sorted" isn't helping here.



Ordering Reputation doesn't help.

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location <> 'Seattle, WA'
  ORDER BY Reputation;

CREATE INDEX IX_DisplayName_Location_Reputation ON
dbo.Users(DisplayName, Location, Reputation);
```

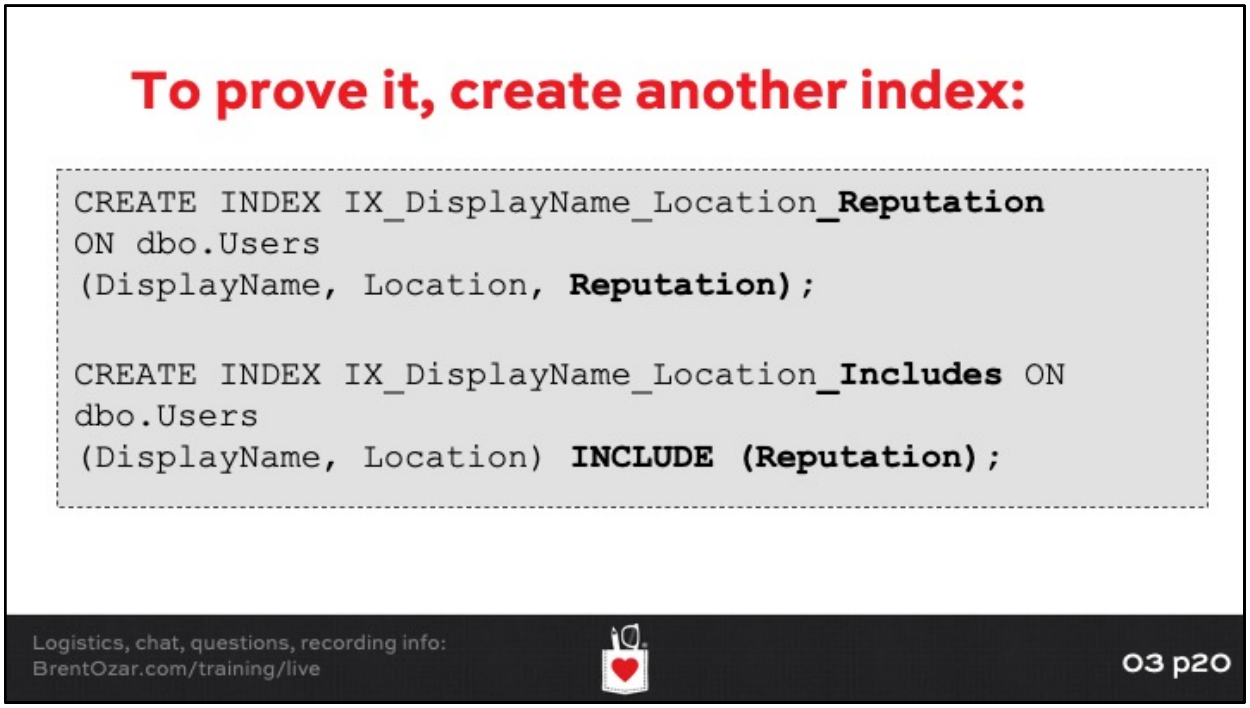
Step 1: seek to Alex

Step 2: scan through, returning everyone EXCEPT Seattle

Step 3: read them out sorted by Reputation, except…they're not.

Logistics, chat, questions, recordir
BrentOzar.com/training/live

03 p19



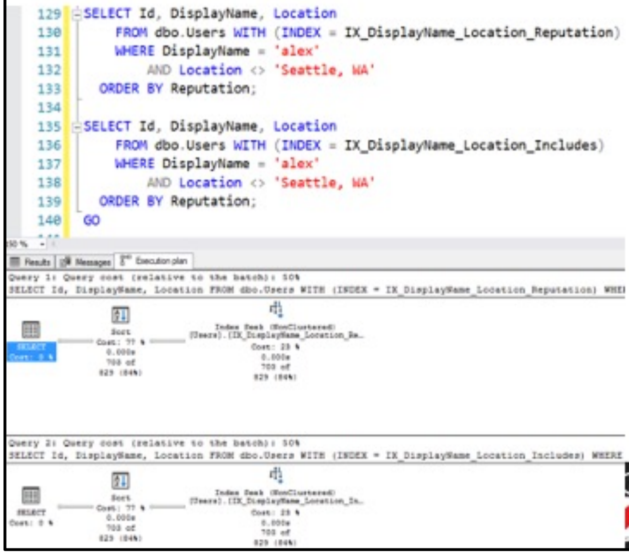
To prove it, create another index:

```
CREATE INDEX IX_DisplayName_Location_Reputation
ON dbo.Users
(DisplayName, Location, Reputation);

CREATE INDEX IX_DisplayName_Location_Includes ON
dbo.Users
(DisplayName, Location) INCLUDE (Reputation);
```

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

03 p20

They both get the same plan



```
SELECT Id, DisplayName, Location
    FROM dbo.Users WITH (INDEX = IX_DisplayName_Location_Reputation)
    WHERE DisplayName = 'alex'
        AND Location <> 'Seattle, WA'
  ORDER BY Reputation;

SELECT Id, DisplayName, Location
    FROM dbo.Users WITH (INDEX = IX_DisplayName_Location_Includes)
    WHERE DisplayName = 'alex'
        AND Location <> 'Seattle, WA'
  ORDER BY Reputation;
GO
```

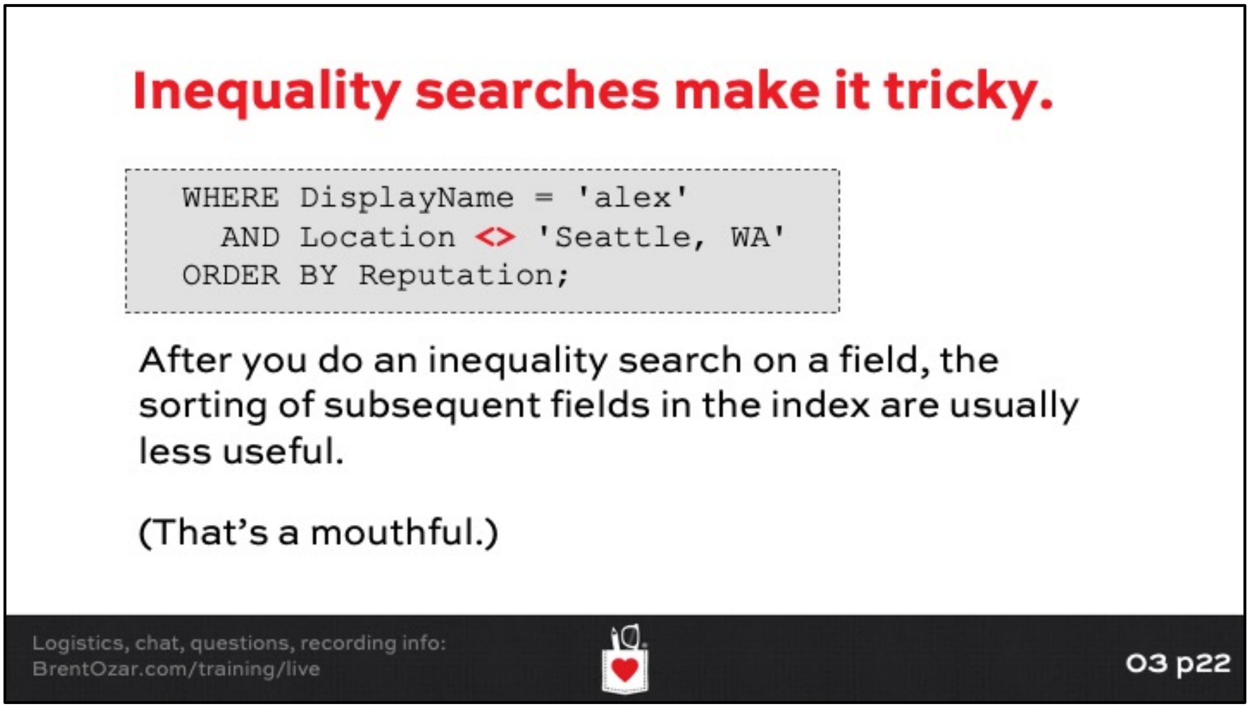
```
Query 1: Query cost (relative to the batch): 50%
SELECT Id, DisplayName, Location FROM dbo.Users WITH (INDEX = IX_DisplayName_Location_Reputation) WHE
```

```
Query 2: Query cost (relative to the batch): 50%
SELECT Id, DisplayName, Location FROM dbo.Users WITH (INDEX = IX_DisplayName_Location_Includes) WHERE
```

Use an index hint to test both indexes separately.

Both do the sort.

And both have the same number of logical reads.

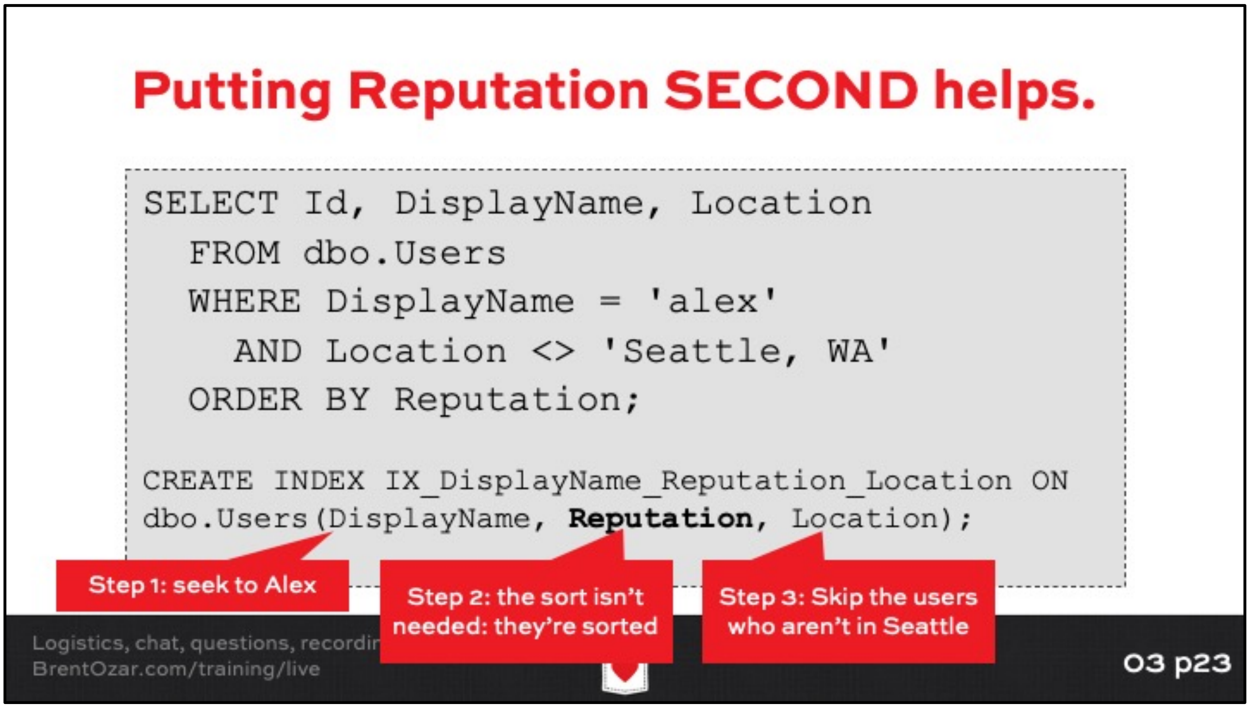


Inequality searches make it tricky.

```
  WHERE DisplayName = 'alex'
    AND Location <> 'Seattle, WA'
  ORDER BY Reputation;
```

After you do an inequality search on a field, the sorting of subsequent fields in the index are usually less useful.

(That's a mouthful.)



Putting Reputation SECOND helps.

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location <> 'Seattle, WA'
  ORDER BY Reputation;

CREATE INDEX IX_DisplayName_Reputation_Location ON
dbo.Users(DisplayName, Reputation, Location);
```

Step 1: seek to Alex

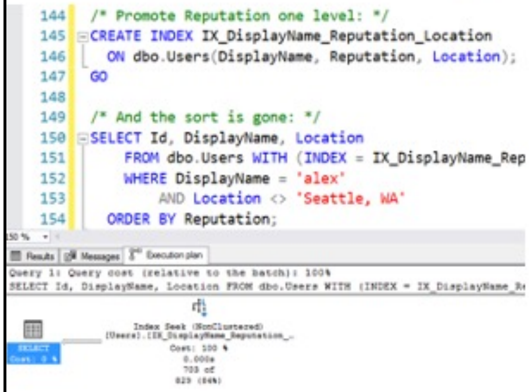
Step 2: the sort isn't needed: they're sorted

Step 3: Skip the users who aren't in Seattle

Logistics, chat, questions, recordin
BrentOzar.com/training/live

03 p23

The sort is gone with this trick.



```
/* Promote Reputation one level: */
CREATE INDEX IX_DisplayName_Reputation_Location
  ON dbo.Users(DisplayName, Reputation, Location);
GO

/* And the sort is gone: */
SELECT Id, DisplayName, Location
    FROM dbo.Users WITH (INDEX = IX_DisplayName_Rep
    WHERE DisplayName = 'alex'
        AND Location <> 'Seattle, WA'
  ORDER BY Reputation;
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT Id, DisplayName, Location FROM dbo.Users WITH (INDEX = IX_DisplayName_R
```

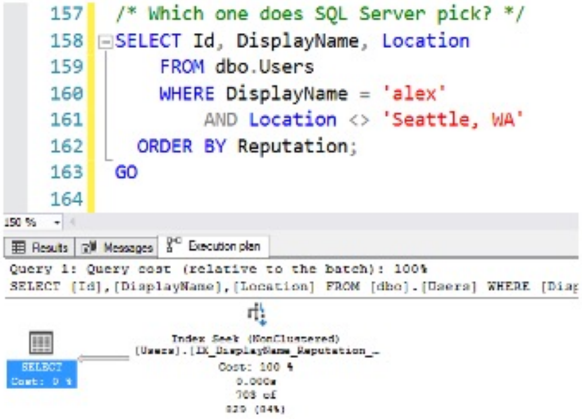
SELECT
Cost: 0 %
Index Seek (NonClustered)
[Users].[IX_DisplayName_Reputation_...
Cost: 100 %
0.000s
703 of
829 (84%)

Obscure trick. To get it, key on:

1. Equality fields, then
2. Sort fields, then
3. Inequality fields

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

SQL Server picks it, too.

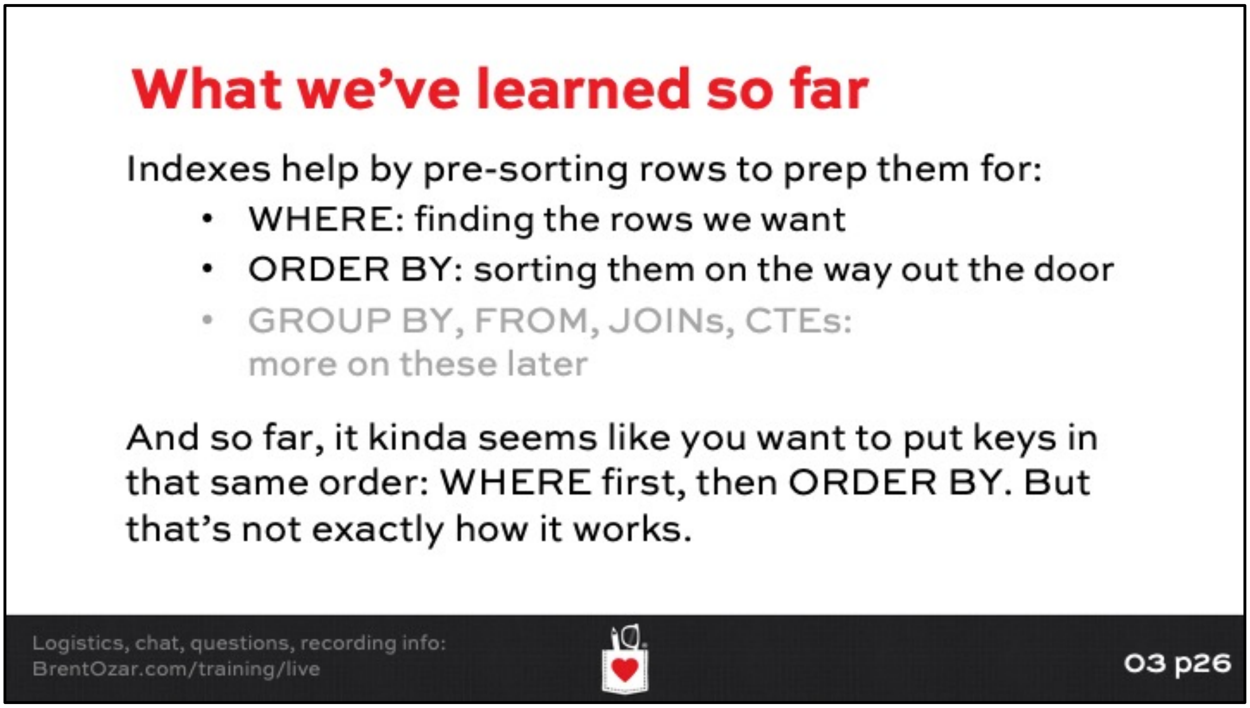


```
/* Which one does SQL Server pick? */
SELECT Id, DisplayName, Location
    FROM dbo.Users
    WHERE DisplayName = 'alex'
        AND Location <> 'Seattle, WA'
  ORDER BY Reputation;
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT [Id],[DisplayName],[Location] FROM [dbo].[Users] WHERE [Diag
```

SELECT
Cost: 0 %
Index Seek (NonClustered)
[Users].[IX_DisplayName_Reputation_…
Cost: 100 %
0.000s
708 of
829 (84%)

If we don't hint the query, here it picks the index that removes the sort.

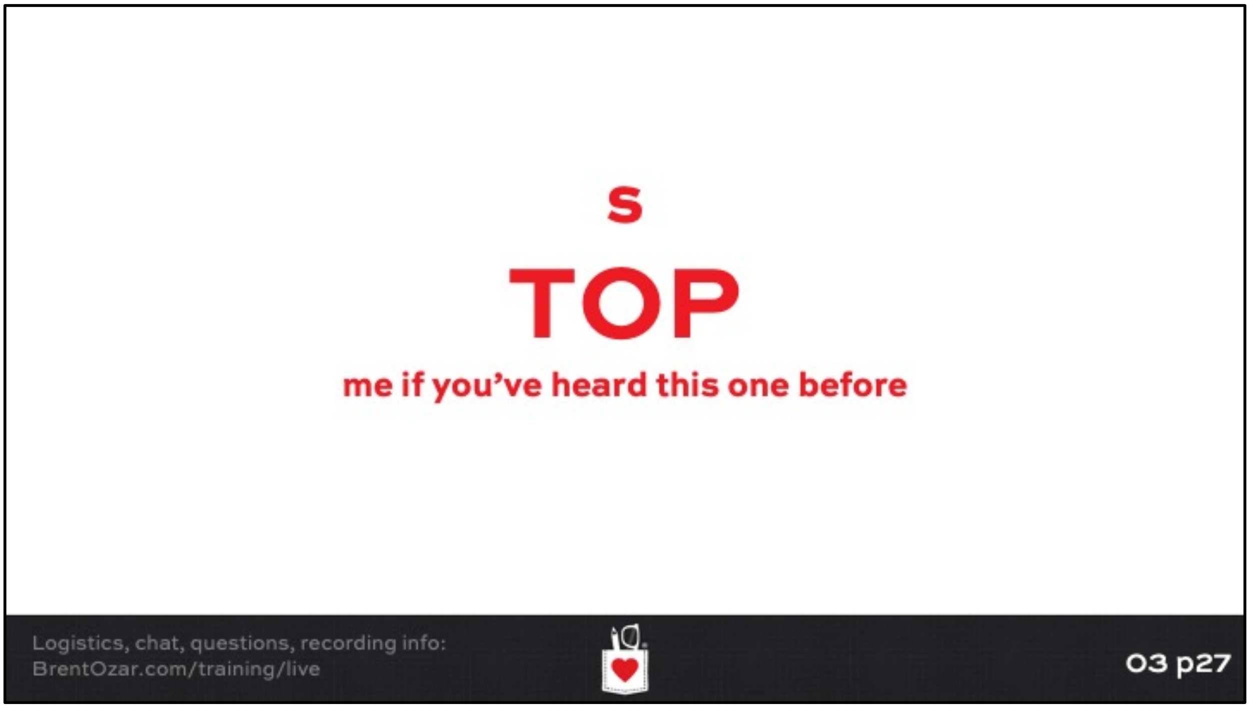


What we've learned so far

Indexes help by pre-sorting rows to prep them for:

- WHERE: finding the rows we want
- ORDER BY: sorting them on the way out the door
- GROUP BY, FROM, JOINs, CTEs:
  more on these later

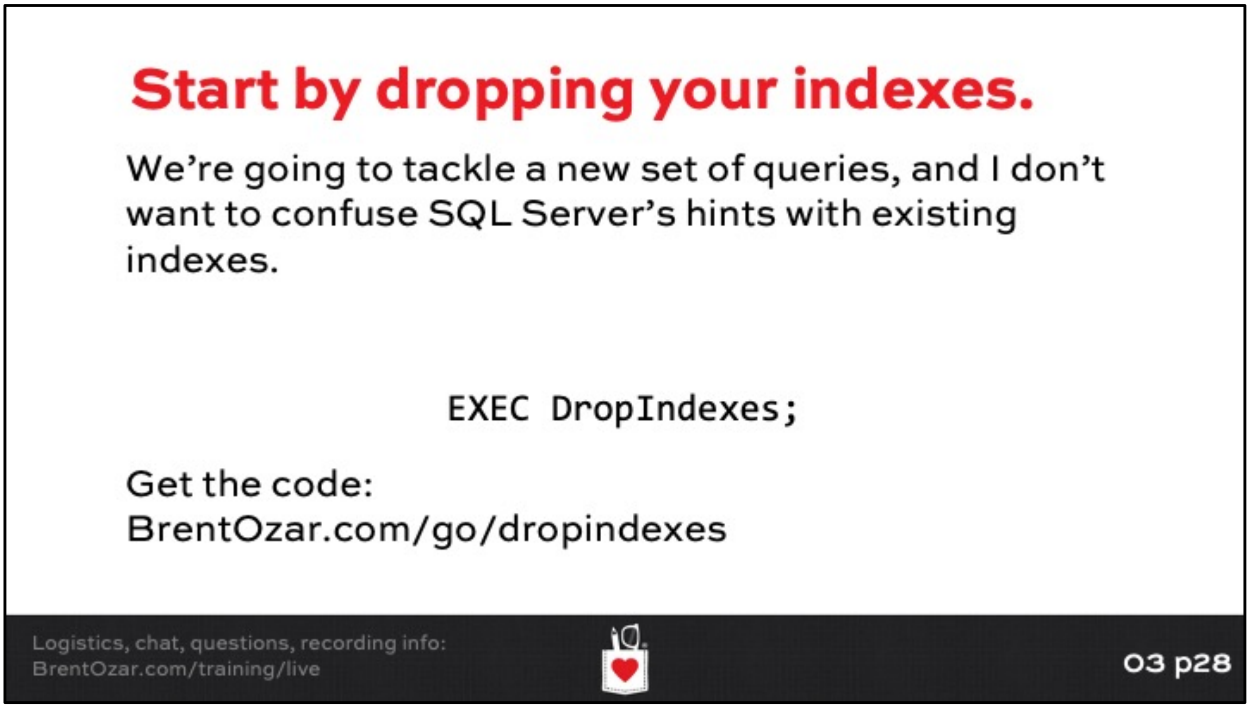
And so far, it kinda seems like you want to put keys in that same order: WHERE first, then ORDER BY. But that's not exactly how it works.



s

TOP

me if you’ve heard this one before



Start by dropping your indexes.

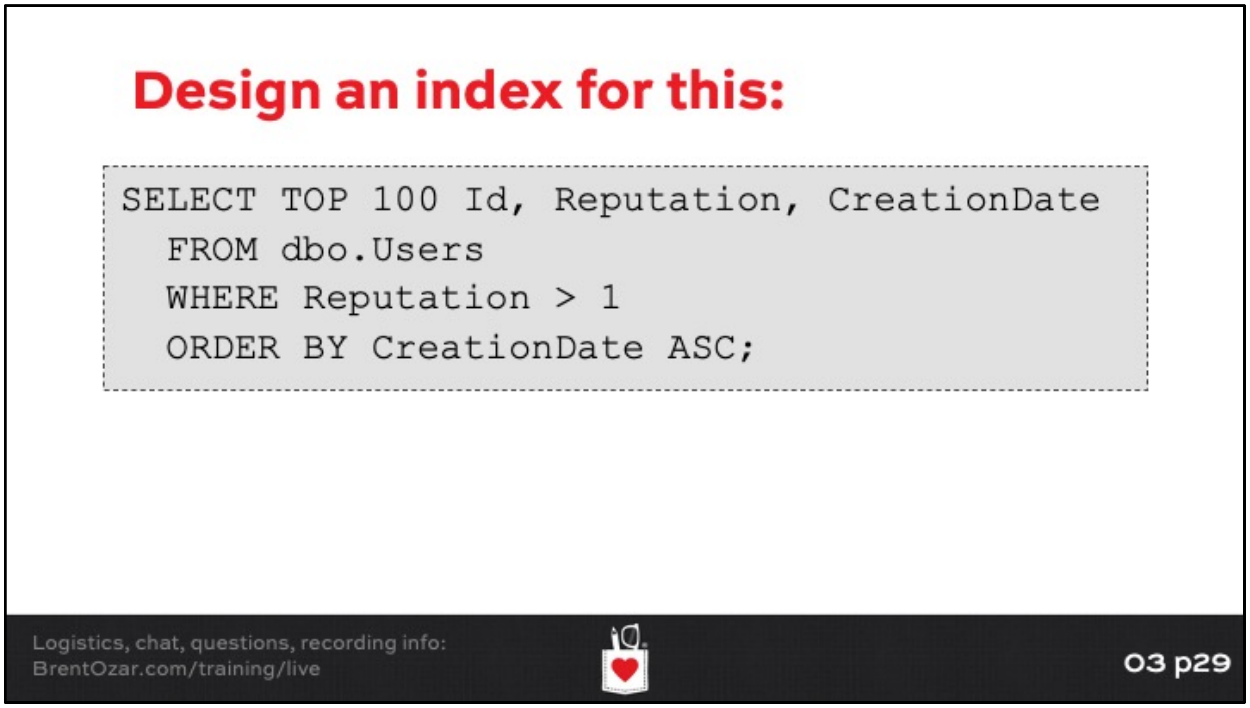
We’re going to tackle a new set of queries, and I don’t want to confuse SQL Server’s hints with existing indexes.

```
EXEC DropIndexes;
```

Get the code:
BrentOzar.com/go/dropindexes

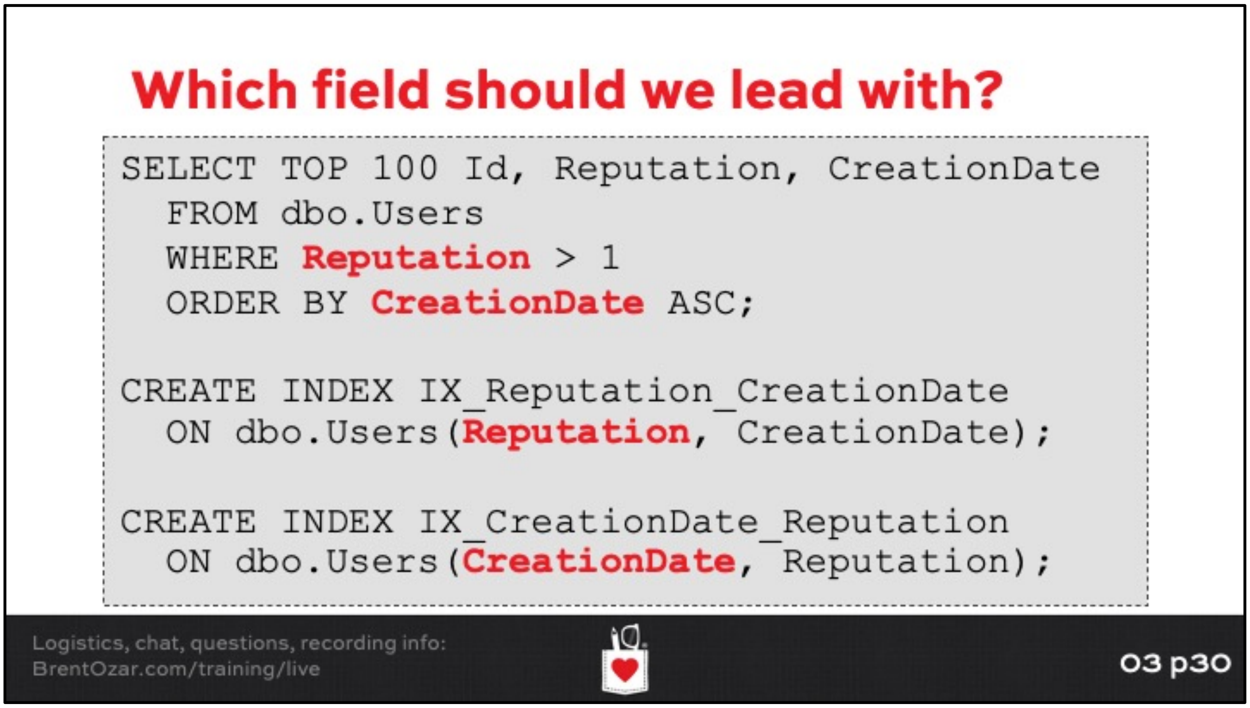
Logistics, chat, questions, recording info:
BrentOzar.com/training/live

03 p28



Design an index for this:

```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1
  ORDER BY CreationDate ASC;
```

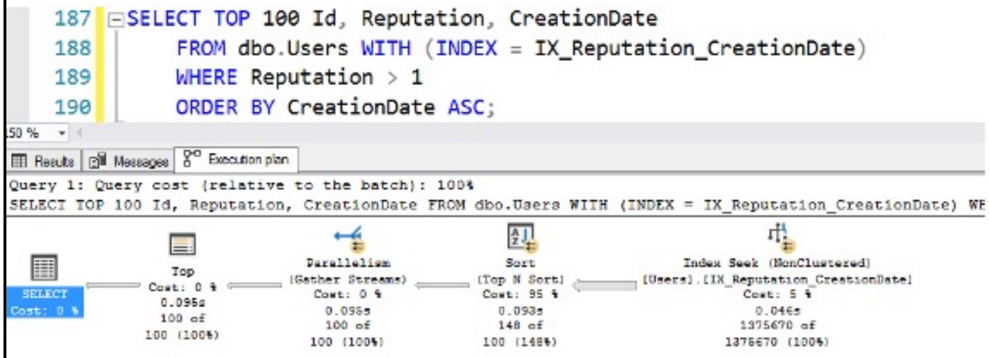


Which field should we lead with?

```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1
  ORDER BY CreationDate ASC;

CREATE INDEX IX_Reputation_CreationDate
  ON dbo.Users(Reputation, CreationDate);

CREATE INDEX IX_CreationDate_Reputation
  ON dbo.Users(CreationDate, Reputation);
```

If we lead with Reputation...

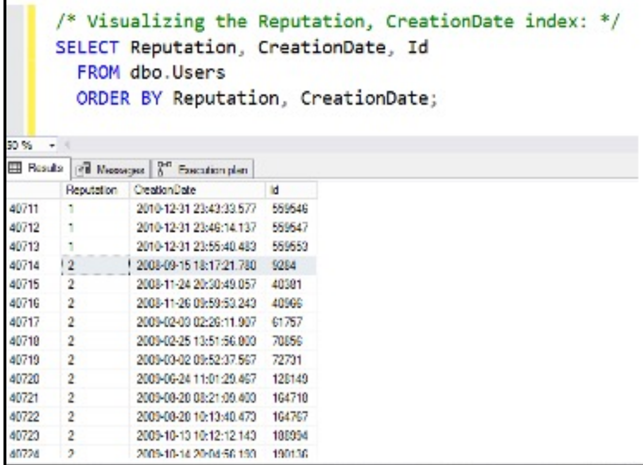
We seek to 2, but then we find 1.4M users that match!
We have to sort 'em all by CreationDate.



```
SELECT TOP 100 Id, Reputation, CreationDate
    FROM dbo.Users WITH (INDEX = IX_Reputation_CreationDate)
    WHERE Reputation > 1
    ORDER BY CreationDate ASC;
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 100 Id, Reputation, CreationDate FROM dbo.Users WITH (INDEX = IX_Reputation_CreationDate) WH
```

Visualize the index contents



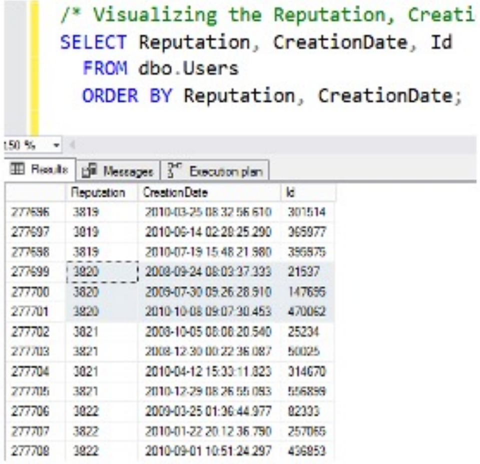
```
/* Visualizing the Reputation, CreationDate index: */
SELECT Reputation, CreationDate, Id
  FROM dbo.Users
  ORDER BY Reputation, CreationDate;
```

	Reputation	CreationDate	Id
40711	1	2010-12-31 23:43:33.577	559546
40712	1	2010-12-31 23:46:14.137	559547
40713	1	2010-12-31 23:55:40.483	559553
40714	2	2008-09-15 18:17:21.780	9284
40715	2	2008-11-24 20:30:49.057	40381
40716	2	2008-11-26 09:59:53.243	40966
40717	2	2009-02-03 02:26:11.907	61757
40718	2	2009-02-25 13:51:56.000	70856
40719	2	2009-03-02 09:52:37.567	72701
40720	2	2009-06-24 11:01:29.467	128149
40721	2	2009-08-20 08:21:09.400	164710
40722	2	2009-08-20 10:13:40.470	164767
40723	2	2009-10-13 10:12:12.143	188934
40724	2	2009-10-14 20:04:56.190	190136

When the index is on Reputation, CreationDate, we can seek to 2, but…are the first 10 users we find the lowest CreationDates overall?

Or just the lowest for Reputation = 2?

Easier way to see it



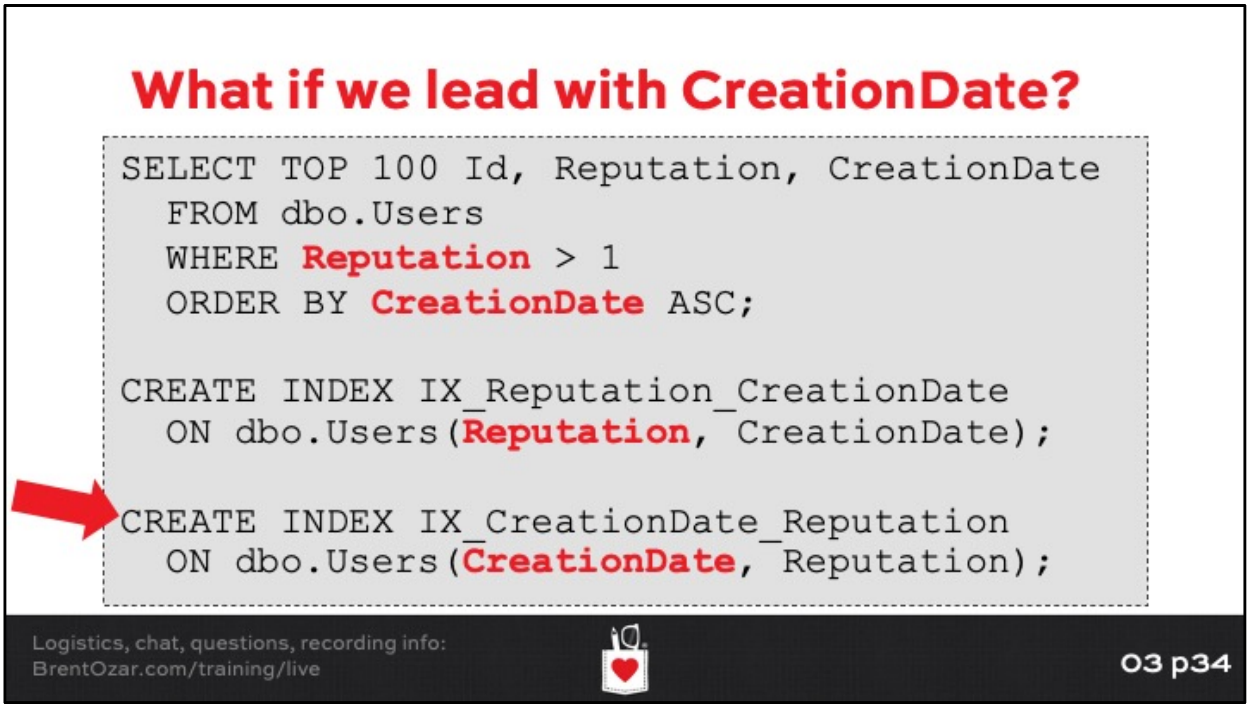
```
/* Visualizing the Reputation, Creati
SELECT Reputation, CreationDate, Id
  FROM dbo.Users
  ORDER BY Reputation, CreationDate;
```

	Reputation	CreationDate	Id
277696	3819	2010-03-25 08:32:56.610	301514
277697	3819	2010-06-14 02:28:25.290	365977
277698	3819	2010-07-19 15:48:21.980	395975
277699	3820	2008-09-24 08:03:37.333	21537
277700	3820	2009-07-30 09:26:28.910	147695
277701	3820	2010-10-08 09:07:30.453	470062
277702	3821	2008-10-05 08:08:20.540	25234
277703	3821	2008-12-30 00:22:36.087	50025
277704	3821	2010-04-12 15:33:11.823	314670
277705	3821	2010-12-29 08:26:55.093	556899
277706	3822	2009-03-25 01:36:44.977	82333
277707	3822	2010-01-22 20:12:36.790	257065
277708	3822	2010-09-01 10:51:24.297	436853

It's more obvious when we page down to higher Reputation numbers.

The CreationDate keeps resetting with each new Reputation.

The sort on the second field is less useful when we're scanning.



What if we lead with CreationDate?

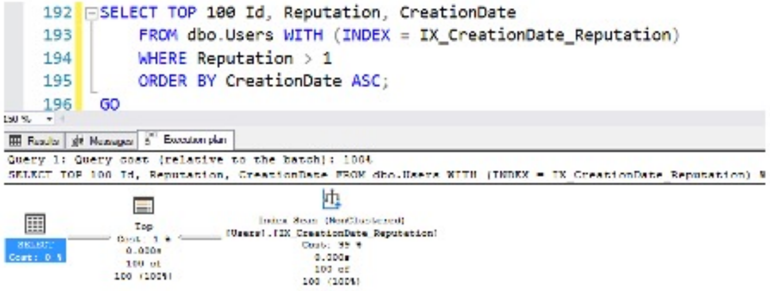
```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1
  ORDER BY CreationDate ASC;

CREATE INDEX IX_Reputation_CreationDate
  ON dbo.Users(Reputation, CreationDate);

CREATE INDEX IX_CreationDate_Reputation
  ON dbo.Users(CreationDate, Reputation);
```

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

We “scan” the index, but...

Remember from How to Think Like the Engine: scan just means we start at one end of the index, and we read until we find the rows that match.

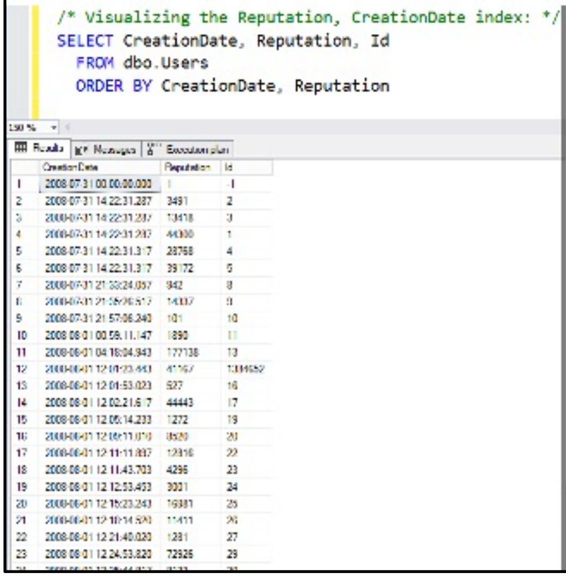
And there’s no sort! The data is already sorted.



```
SELECT TOP 100 Id, Reputation, CreationDate
    FROM dbo.Users WITH (INDEX = IX_CreationDate_Reputation)
    WHERE Reputation > 1
    ORDER BY CreationDate ASC;
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 100 Id, Reputation, CreationDate FROM dbo.Users WITH (INDEX = IX_CreationDate_Reputation) W
```

Visualize the index contents



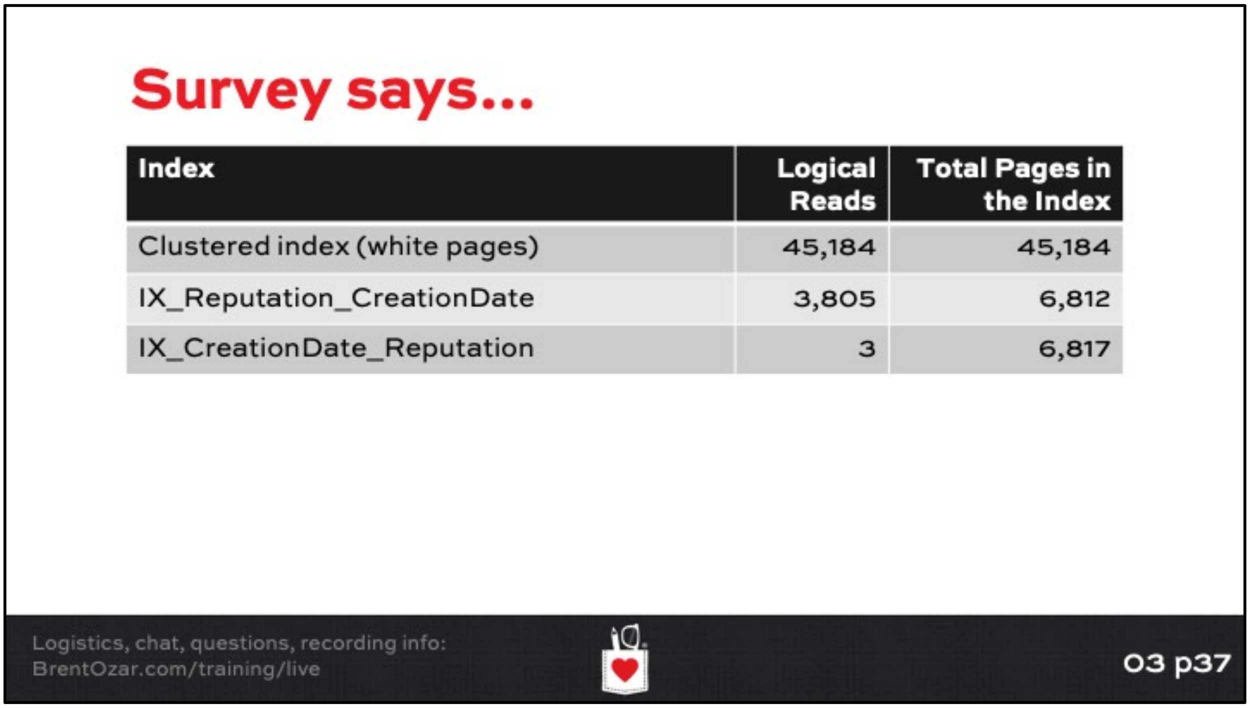
```sql
/* Visualizing the Reputation, CreationDate index: */
SELECT CreationDate, Reputation, Id
  FROM dbo.Users
  ORDER BY CreationDate, Reputation
```

	CreationDate	Reputation	Id
1	2008-07-31 00:00:00.000	1	-1
2	2008-07-31 14:22:31.287	3491	2
3	2008-07-31 14:22:31.287	13418	3
4	2008-07-31 14:22:31.287	44300	1
5	2008-07-31 14:22:31.317	28768	4
6	2008-07-31 14:22:31.317	39172	5
7	2008-07-31 21:33:24.057	942	8
8	2008-07-31 21:35:26.517	14337	9
9	2008-07-31 21:57:06.240	101	10
10	2008-08-01 00:59:11.147	1890	11
11	2008-08-01 04:18:04.943	177138	13
12	2008-08-01 12:01:23.443	41167	1384652
13	2008-08-01 12:01:53.023	527	16
14	2008-08-01 12:02:21.617	44443	17
15	2008-08-01 12:05:14.233	1272	19
16	2008-08-01 12:08:11.070	8520	20
17	2008-08-01 12:11:11.887	12316	22
18	2008-08-01 12:11:43.703	4296	23
19	2008-08-01 12:12:53.453	3001	24
20	2008-08-01 12:15:23.243	16381	25
21	2008-08-01 12:18:14.520	11411	26
22	2008-08-01 12:21:40.020	1281	27
23	2008-08-01 12:24:53.820	72926	29
24	[illegible]	[illegible]	[illegible]

When the index is on CreationDate, Reputation, we start reading, looking for 100 users with Reputation > 1.

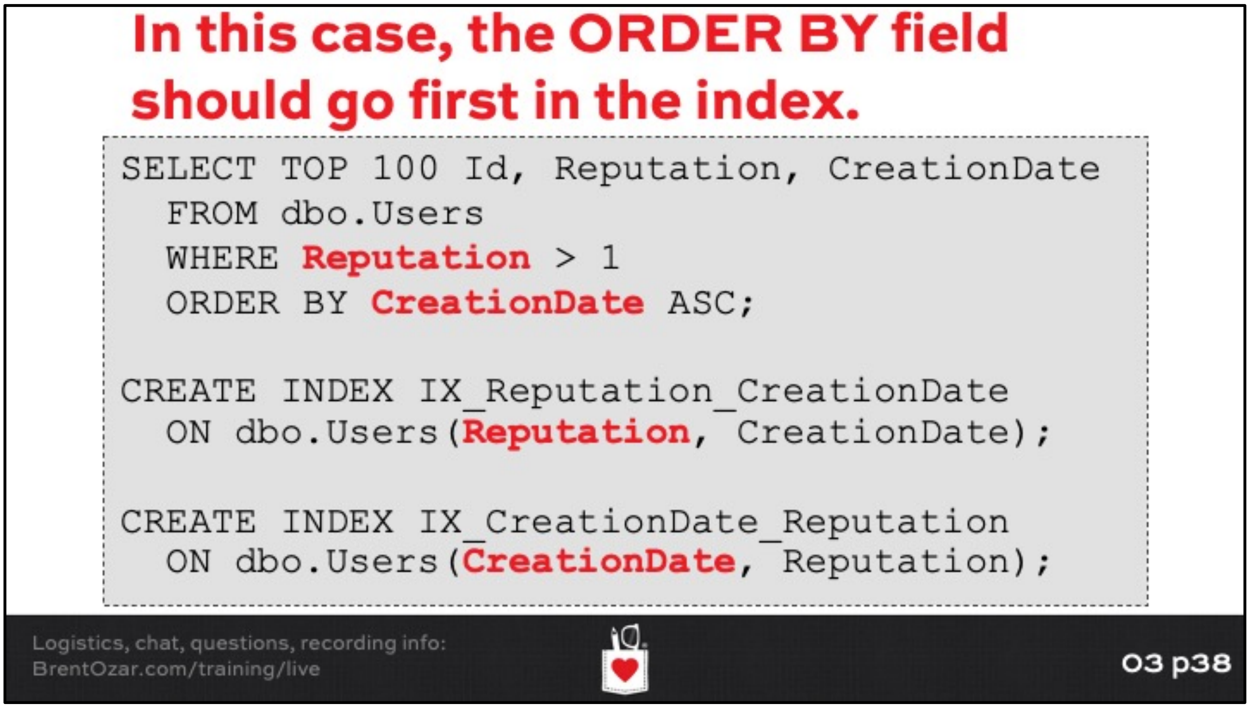
They almost all match!

As soon as we read 100 rows that match, we're done. No need to scan the whole index.



Survey says…

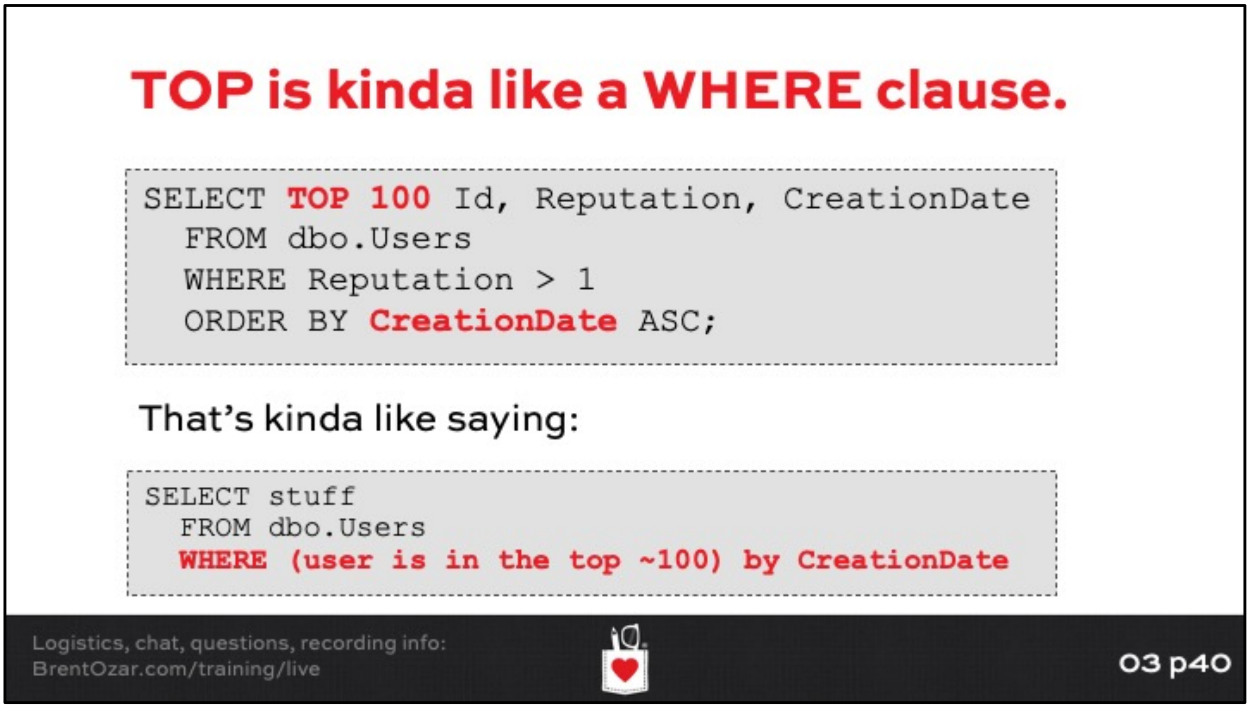
Index	Logical Reads	Total Pages in the Index
Clustered index (white pages)	45,184	45,184
IX_Reputation_CreationDate	3,805	6,812
IX_CreationDate_Reputation	3	6,817



In this case, the ORDER BY field should go first in the index.

```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1
  ORDER BY CreationDate ASC;

CREATE INDEX IX_Reputation_CreationDate
  ON dbo.Users(Reputation, CreationDate);

CREATE INDEX IX_CreationDate_Reputation
  ON dbo.Users(CreationDate, Reputation);
```

Remember selectivity?

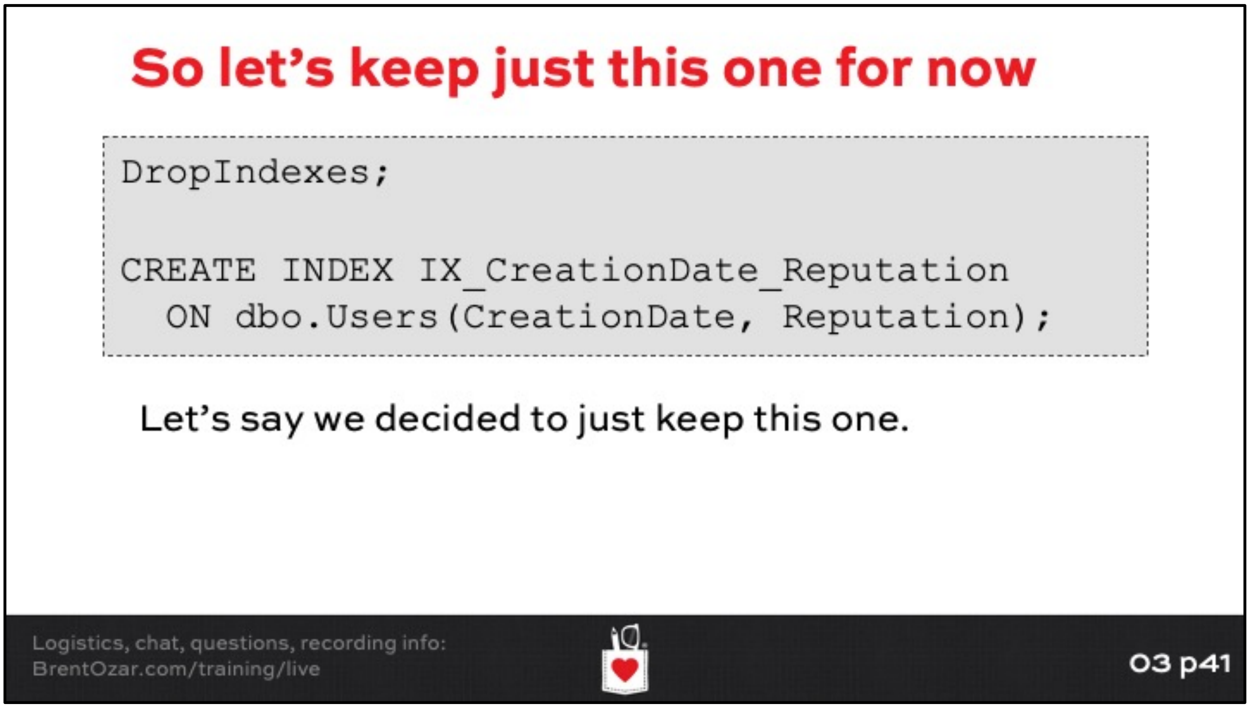


TOP is kinda like a WHERE clause.

```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1
  ORDER BY CreationDate ASC;
```

That's kinda like saying:

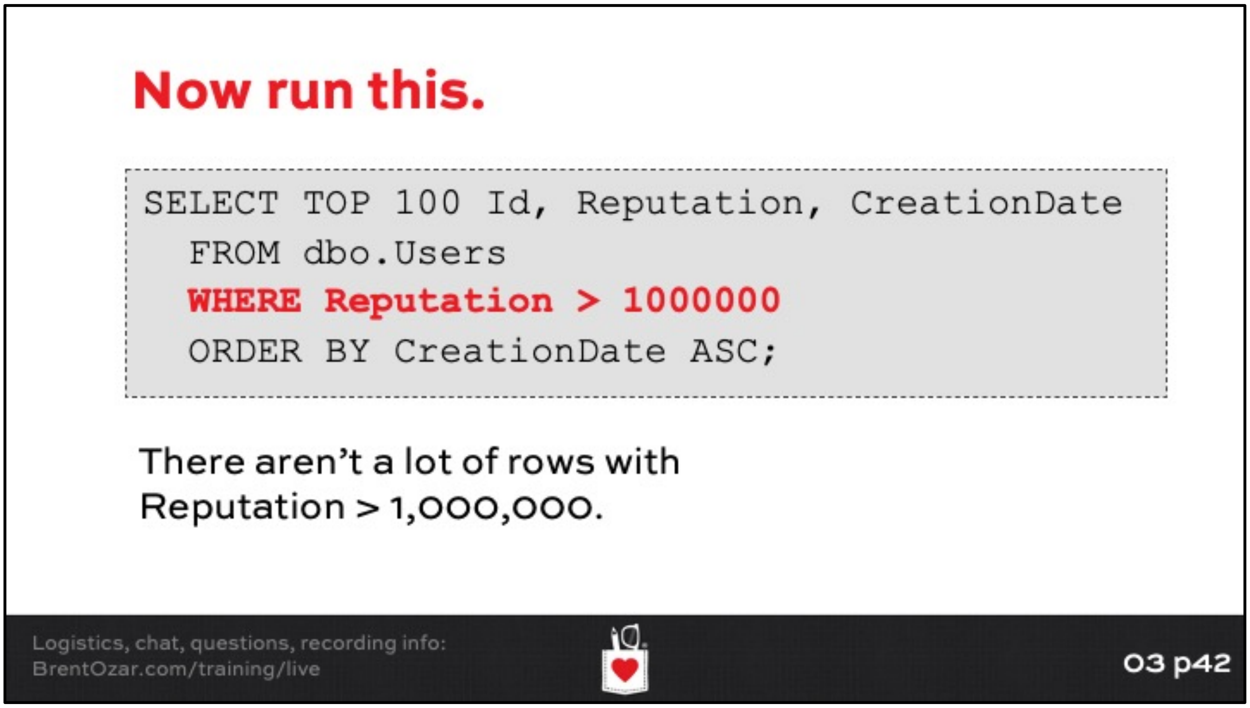
```
SELECT stuff
  FROM dbo.Users
  WHERE (user is in the top ~100) by CreationDate
```



So let’s keep just this one for now

```
DropIndexes;

CREATE INDEX IX_CreationDate_Reputation
  ON dbo.Users(CreationDate, Reputation);
```

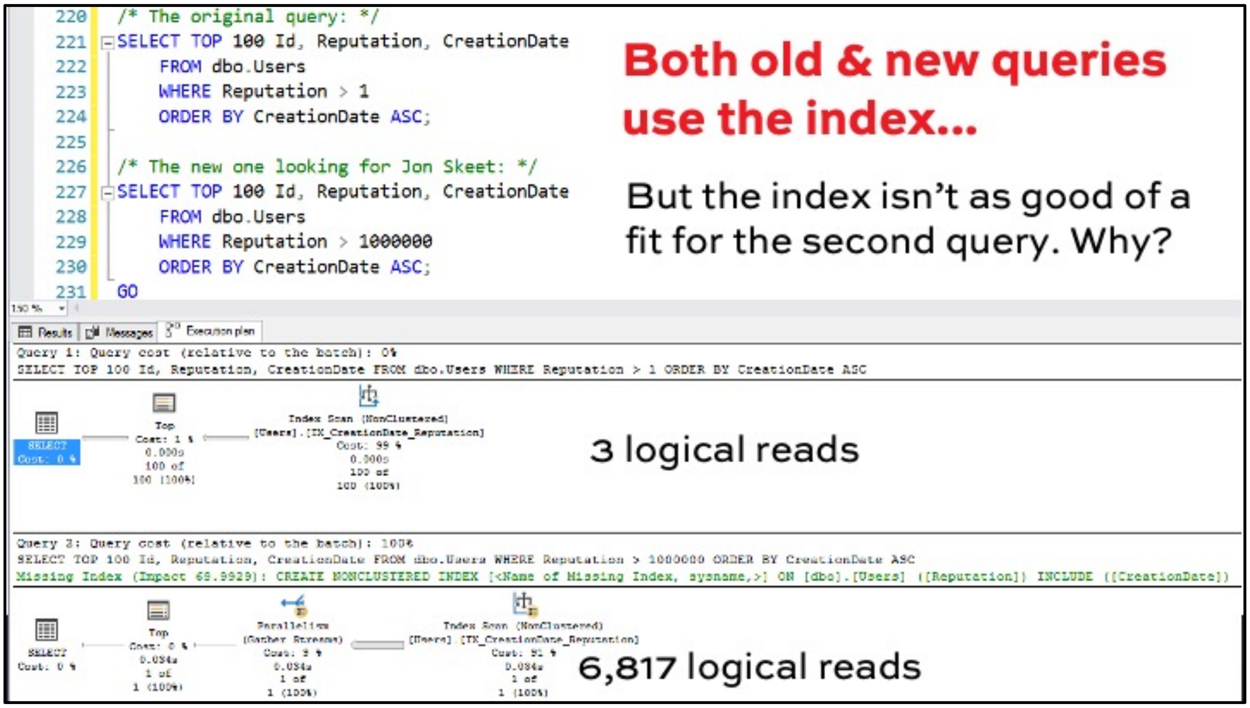
Let’s say we decided to just keep this one.



Now run this.

```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1000000
  ORDER BY CreationDate ASC;
```

There aren’t a lot of rows with
Reputation > 1,000,000.



```
/* The original query: */
SELECT TOP 100 Id, Reputation, CreationDate
    FROM dbo.Users
    WHERE Reputation > 1
    ORDER BY CreationDate ASC;

/* The new one looking for Jon Skeet: */
SELECT TOP 100 Id, Reputation, CreationDate
    FROM dbo.Users
    WHERE Reputation > 1000000
    ORDER BY CreationDate ASC;
GO
```

Both old & new queries use the index...

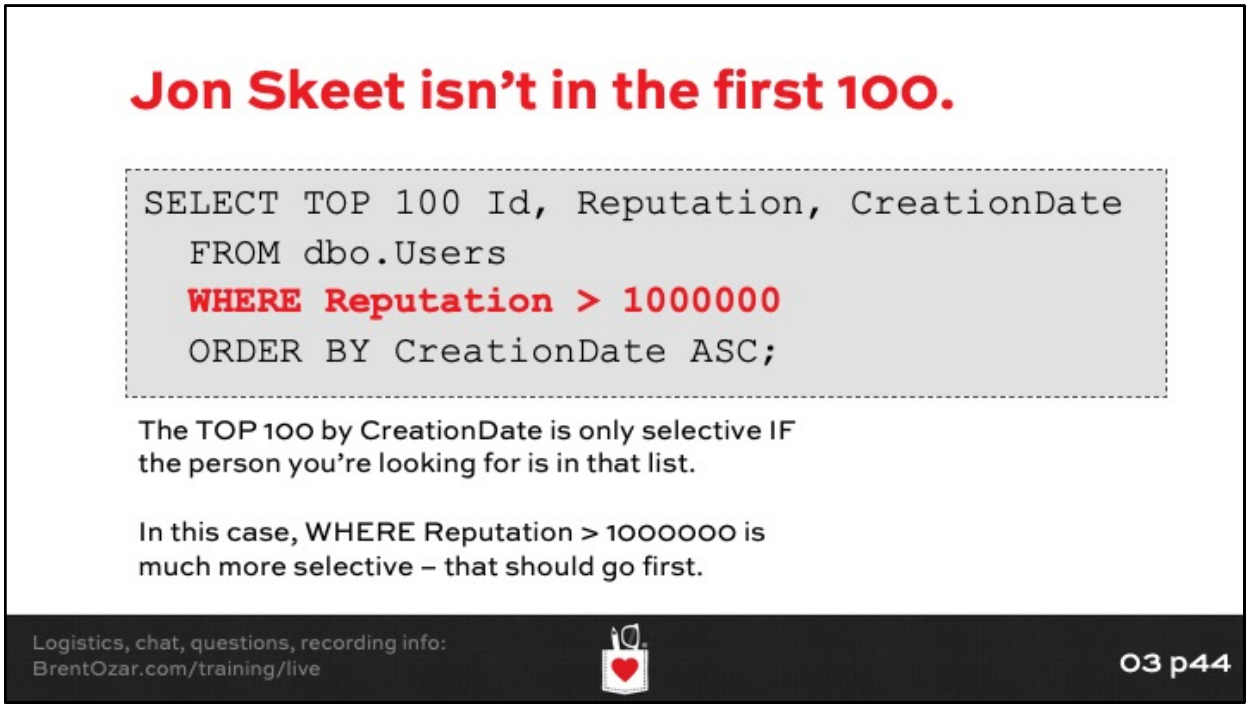
But the index isn't as good of a fit for the second query. Why?

```
Query 1: Query cost (relative to the batch): 0%
SELECT TOP 100 Id, Reputation, CreationDate FROM dbo.Users WHERE Reputation > 1 ORDER BY CreationDate ASC
```

3 logical reads

```
Query 2: Query cost (relative to the batch): 100%
SELECT TOP 100 Id, Reputation, CreationDate FROM dbo.Users WHERE Reputation > 1000000 ORDER BY CreationDate ASC
Missing Index (Impact 69.9929): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Users] ([Reputation]) INCLUDE ([CreationDate])
```

6,817 logical reads



Jon Skeet isn't in the first 100.

```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1000000
  ORDER BY CreationDate ASC;
```

The TOP 100 by CreationDate is only selective IF the person you're looking for is in that list.

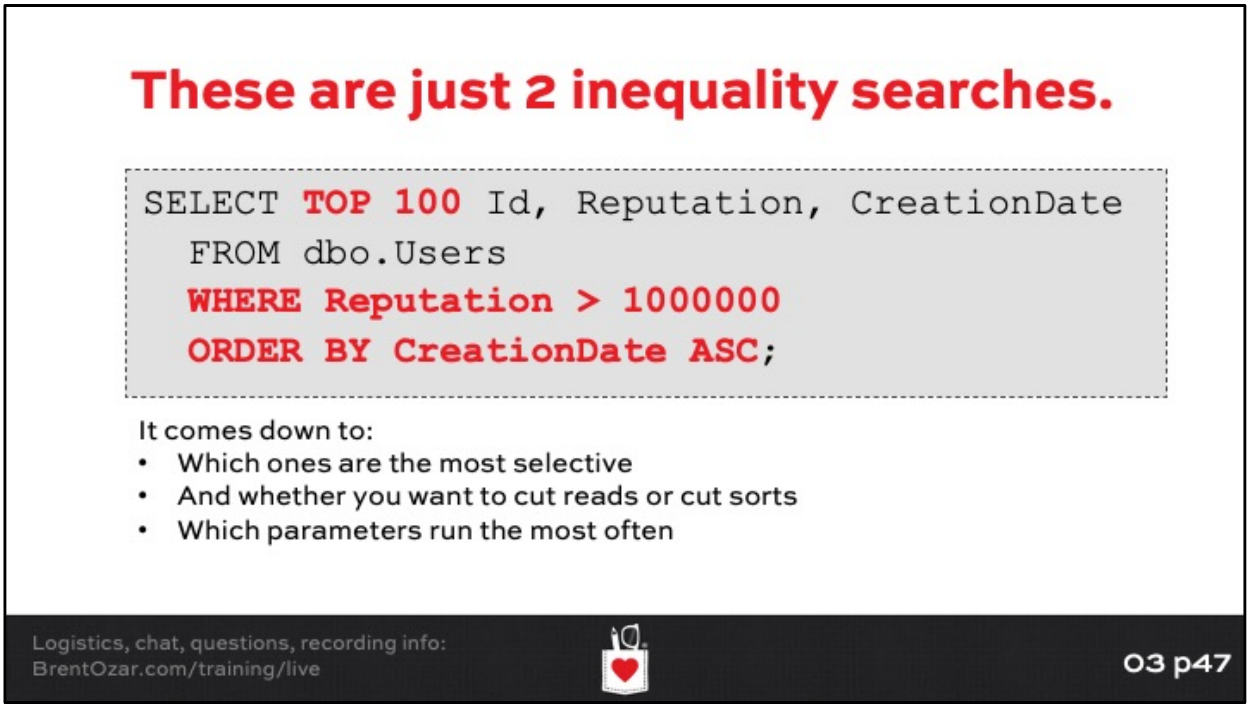
In this case, WHERE Reputation > 1000000 is much more selective – that should go first.

Indexing requires compromises and choices.

Index for the WHERE to reduce reads

Index for the ORDER BY to reduce sorts

Indexing requires compromises and choices.



These are just 2 inequality searches.

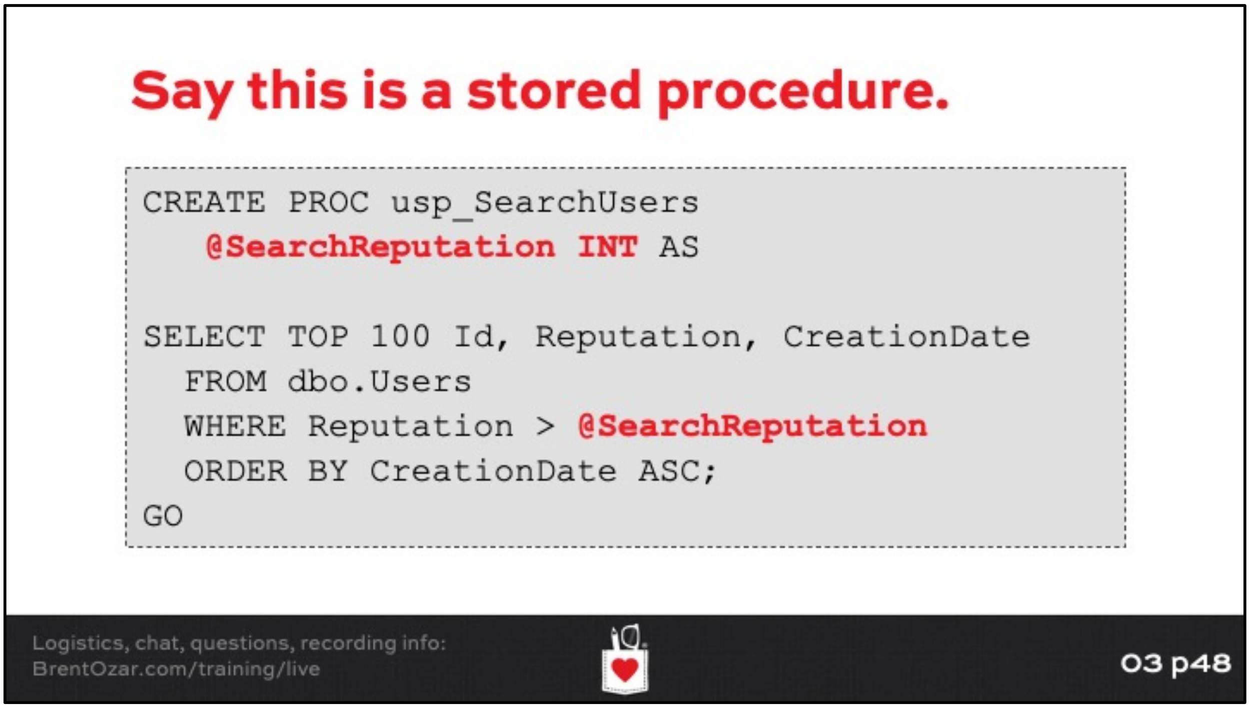
```
SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > 1000000
  ORDER BY CreationDate ASC;
```

It comes down to:

- Which ones are the most selective
- And whether you want to cut reads or cut sorts
- Which parameters run the most often

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

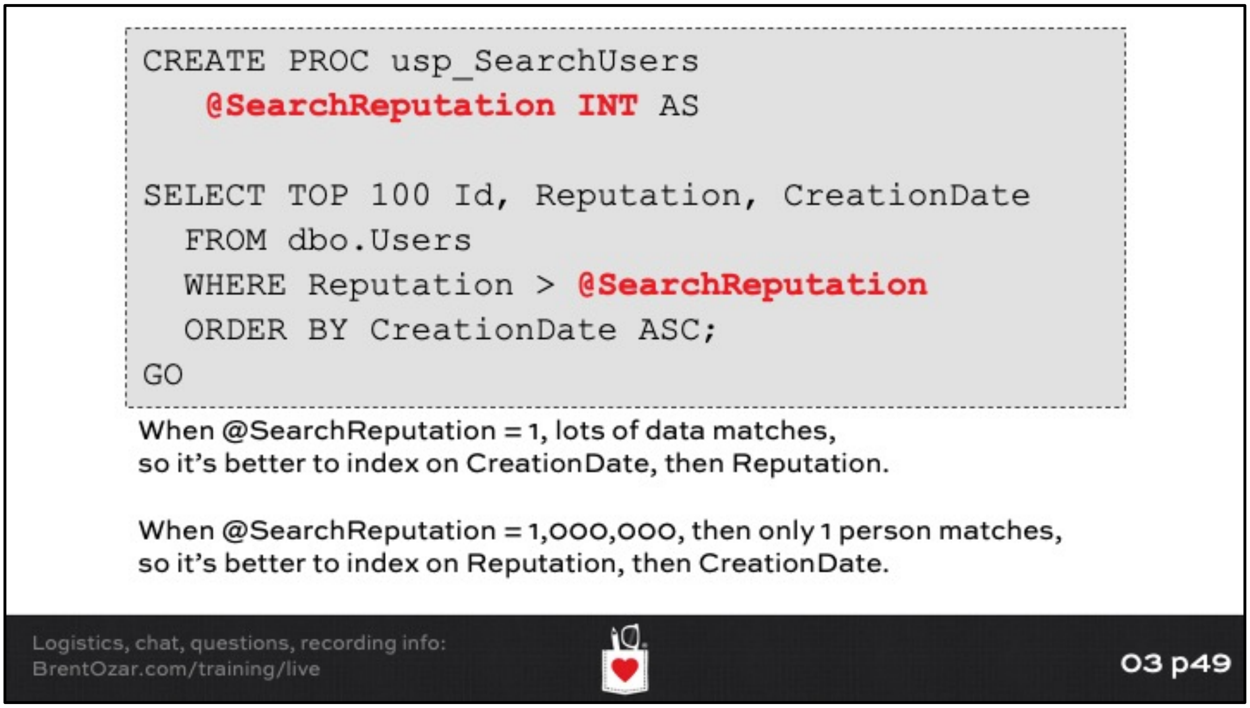
03 p47



Say this is a stored procedure.

```
CREATE PROC usp_SearchUsers
   @SearchReputation INT AS

SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > @SearchReputation
  ORDER BY CreationDate ASC;
GO
```

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

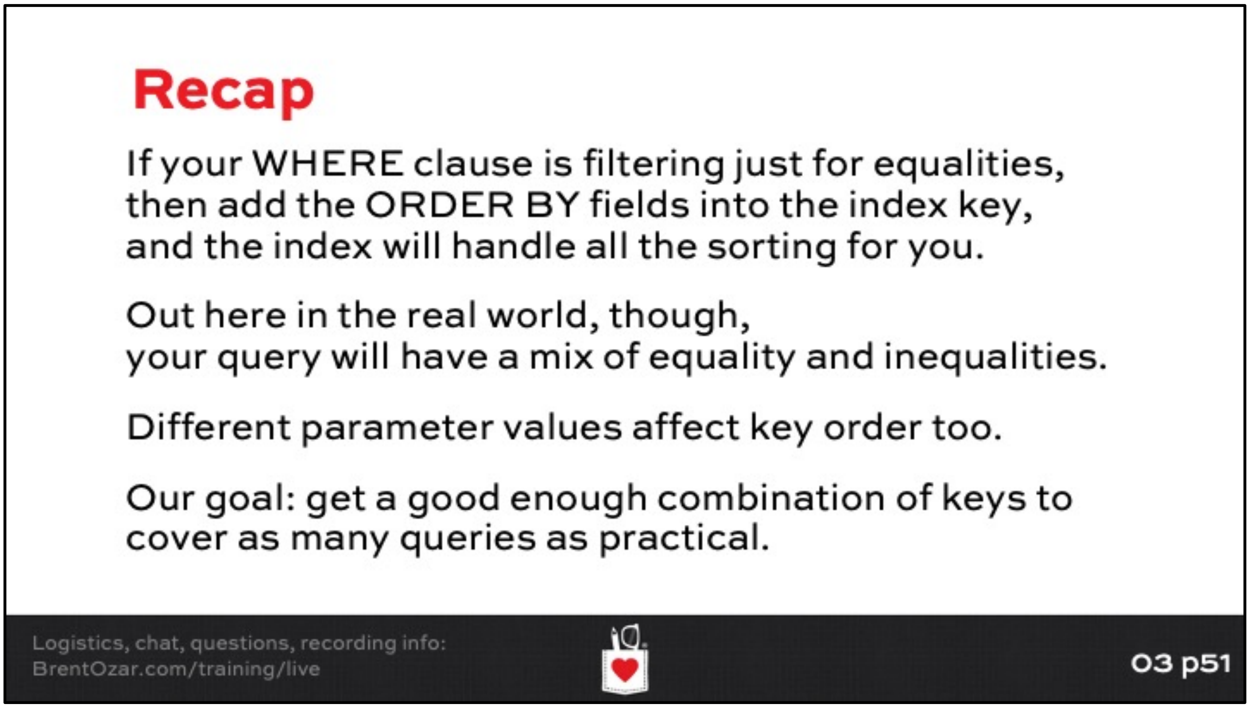


```
CREATE PROC usp_SearchUsers
   @SearchReputation INT AS

SELECT TOP 100 Id, Reputation, CreationDate
  FROM dbo.Users
  WHERE Reputation > @SearchReputation
  ORDER BY CreationDate ASC;
GO
```

When @SearchReputation = 1, lots of data matches,
so it's better to index on CreationDate, then Reputation.

When @SearchReputation = 1,000,000, then only 1 person matches,
so it's better to index on Reputation, then CreationDate.

Re-cap



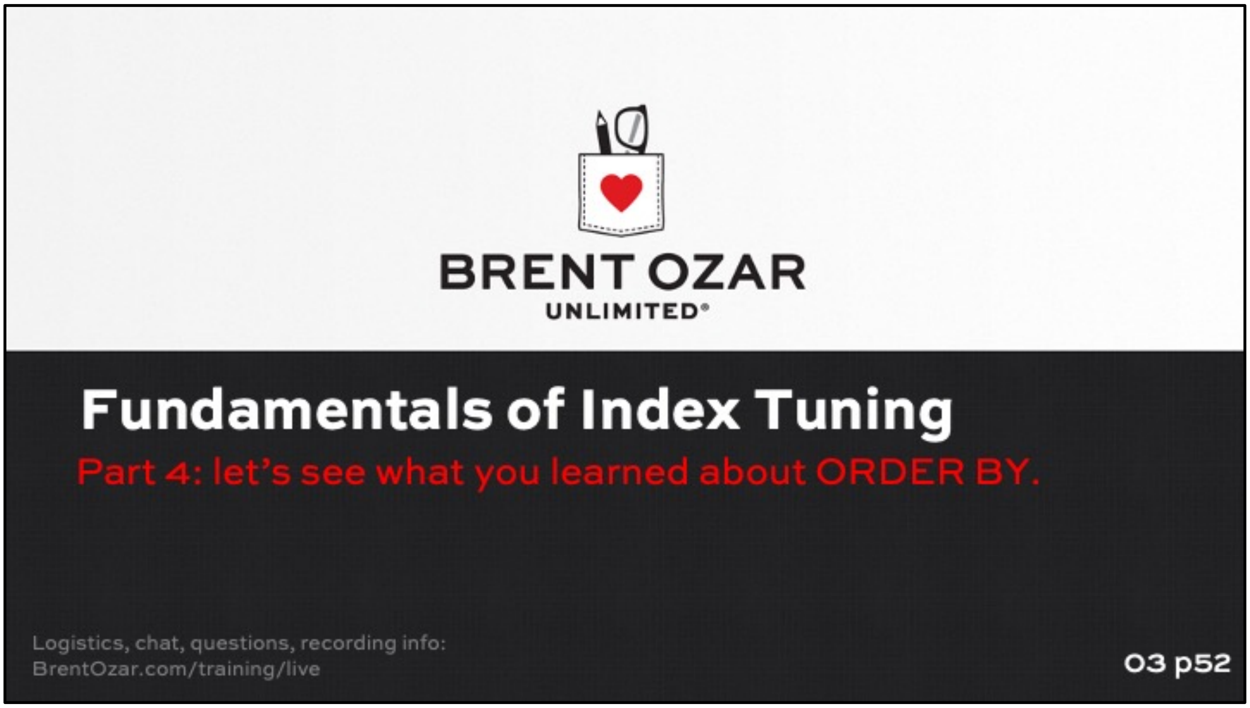
Recap

If your WHERE clause is filtering just for equalities, then add the ORDER BY fields into the index key, and the index will handle all the sorting for you.

Out here in the real world, though,
your query will have a mix of equality and inequalities.

Different parameter values affect key order too.

Our goal: get a good enough combination of keys to cover as many queries as practical.



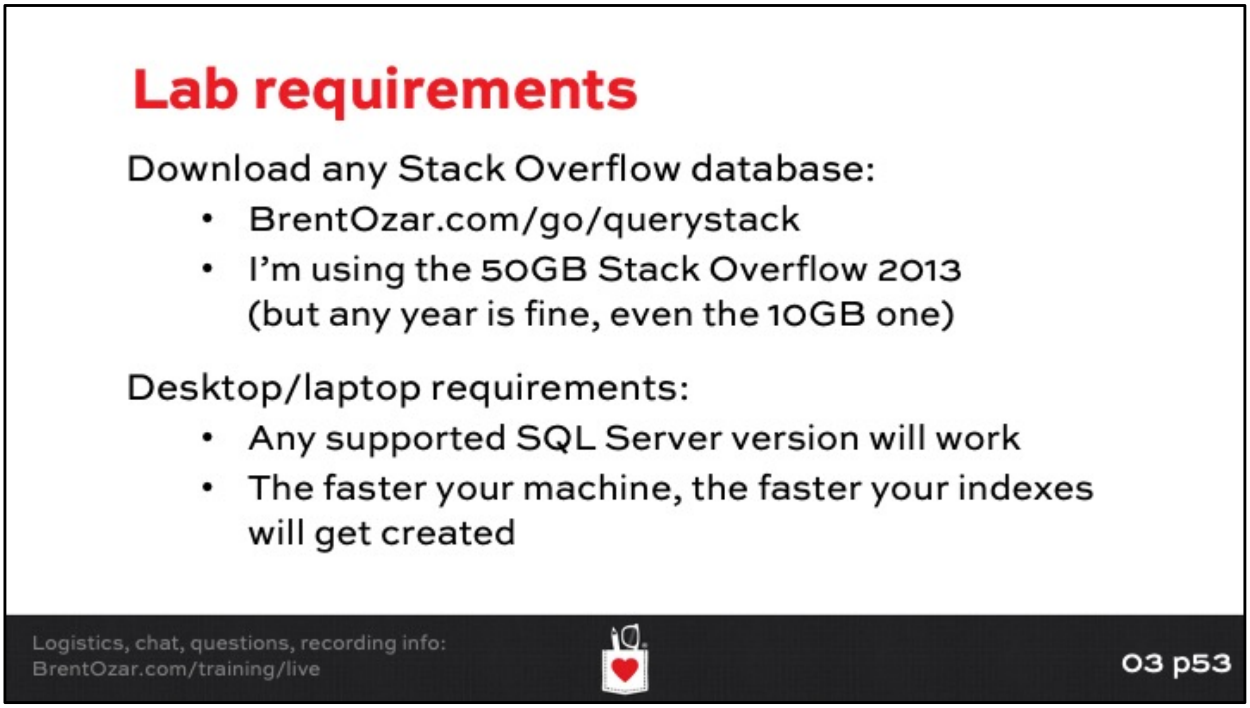
BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Part 4: let's see what you learned about ORDER BY.

Logistics, chat, questions, recording info:
BrentOzar.com/training/live

03 p52



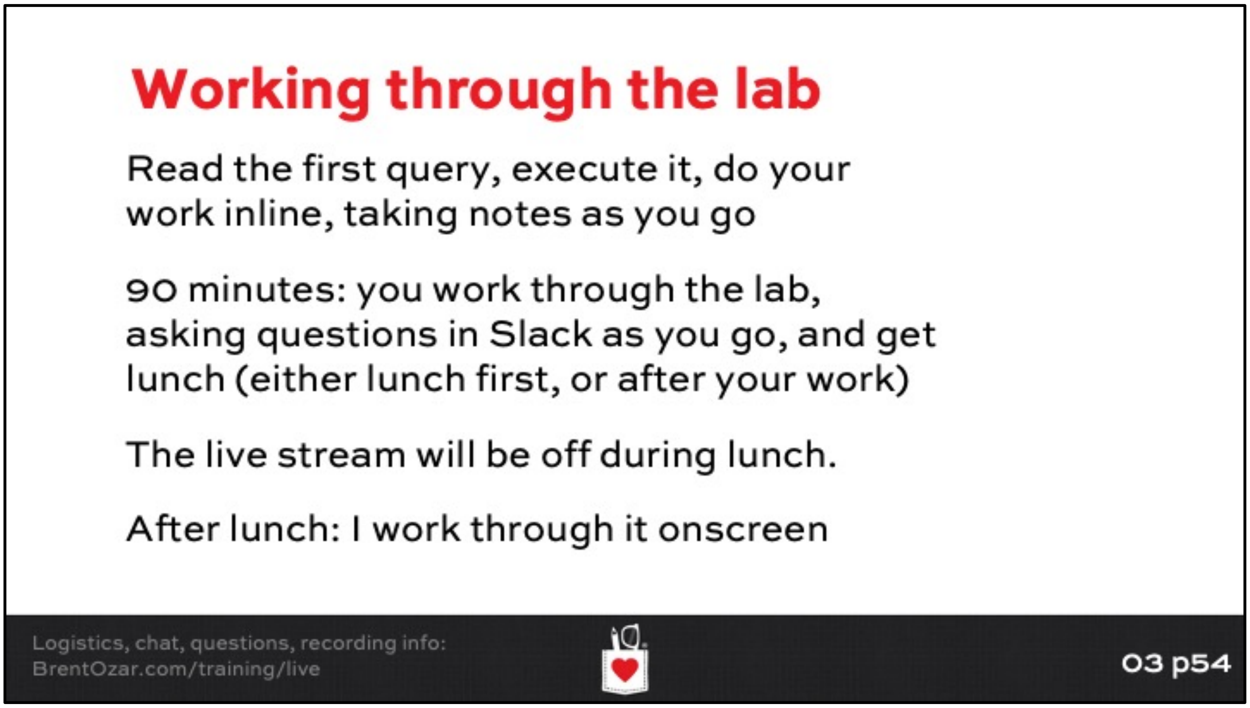
Lab requirements

Download any Stack Overflow database:

- BrentOzar.com/go/querystack
- I'm using the 50GB Stack Overflow 2013 (but any year is fine, even the 10GB one)

Desktop/laptop requirements:

- Any supported SQL Server version will work
- The faster your machine, the faster your indexes will get created



Working through the lab

Read the first query, execute it, do your work inline, taking notes as you go

90 minutes: you work through the lab, asking questions in Slack as you go, and get lunch (either lunch first, or after your work)

The live stream will be off during lunch.

After lunch: I work through it onscreen