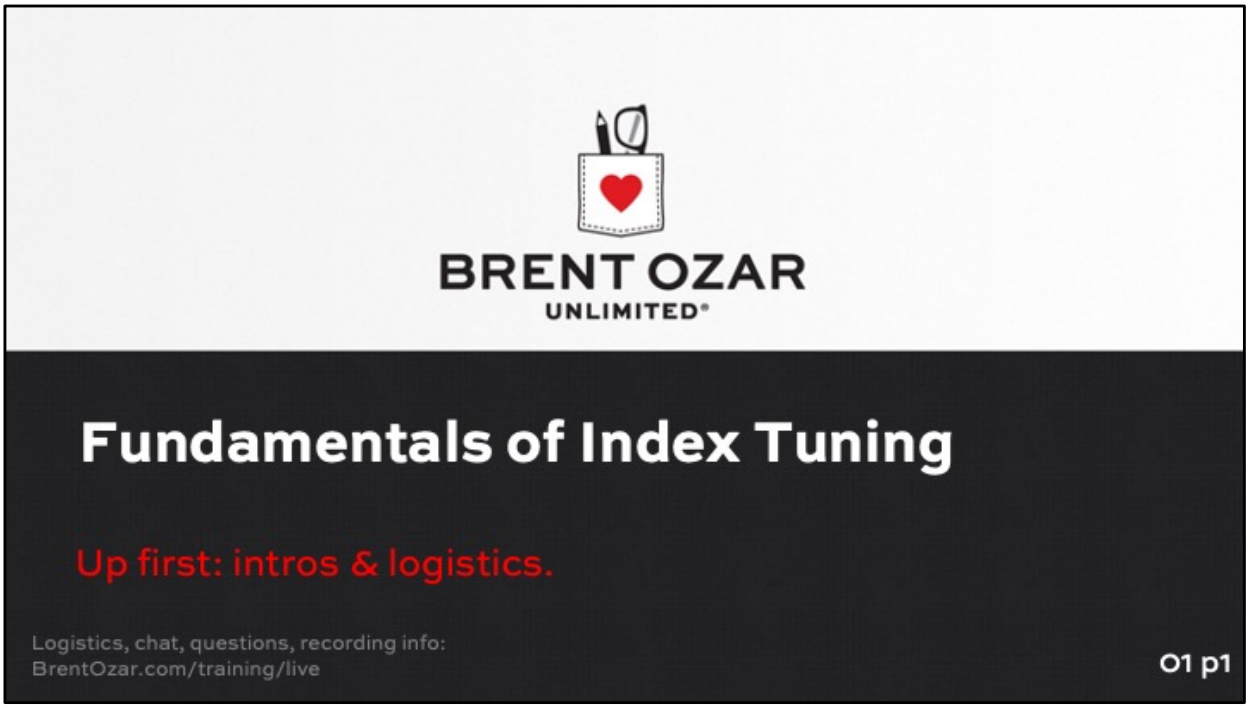
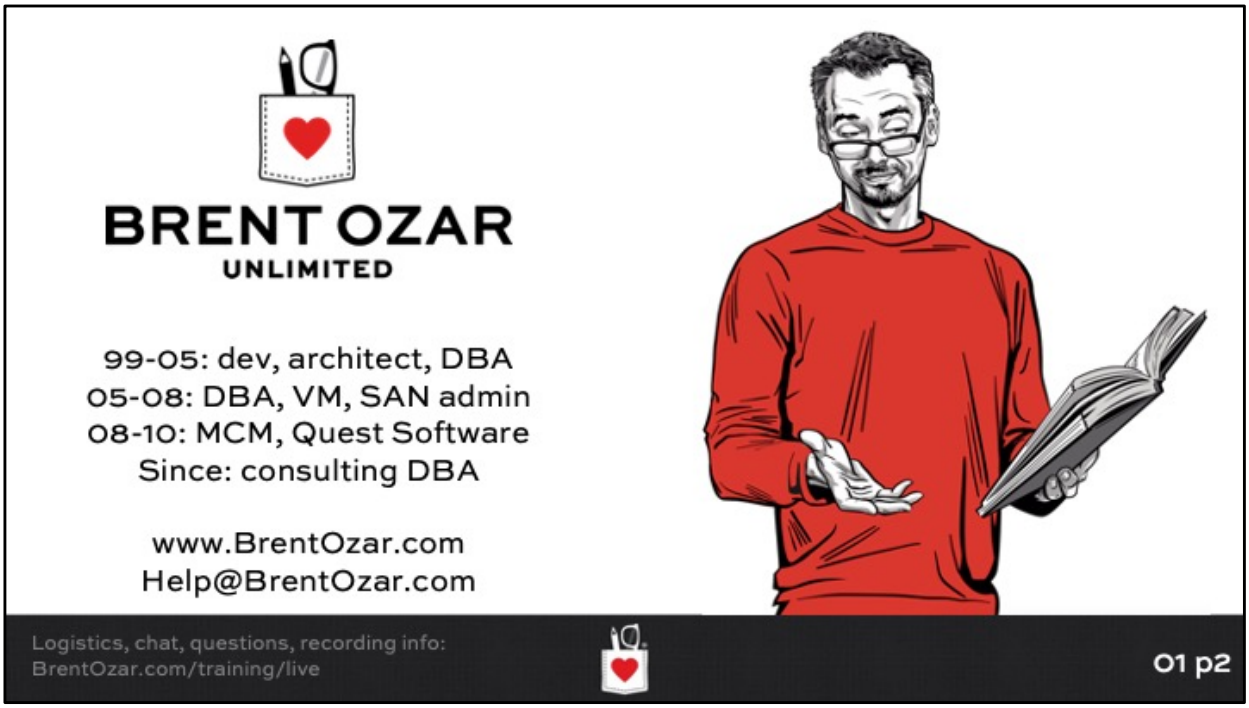




Introduce yourself in Slack:

	Developer	Development DBA	Production DBA
Write C#, Java code	Daily		
Build queries, tables	Daily	Sometimes	
Tune queries	Sometimes	Daily	
Design indexes		Daily	
Monitor performance		Daily	Sometimes
Troubleshoot outages			Daily
Manage backups, jobs			Daily
Install, config SQL			Sometimes
Install, config OS			Sometimes

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

Slack pro tips

Accidentally close your browser? Want to share screenshots? Lots of pro tips: BrentOzar.com/slack

To share code or T-SQL, click the + sign next to where you type text in, and choose “code or text snippet.”

To share files, upload at Imgur.com, paste URL here.

No direct messages please.

Keep the questions on-topic.

Instant Replay & lab scripts

For a year from your purchase, paid students can:

- Watch the class recordings
- Download the scripts
- Re-run the labs on your local machine

Problems or questions?
Leave a comment on the relevant module.

Live streaming tips & tricks

You can adjust video quality at the bottom right

You can pause if you need to walk away, but once you pause, you'll be behind the other students, and the live Q&A, polls, etc won't make sense

If your video stream seems 20-30 seconds behind, or you lose connection, try refreshing your browser

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

01 p6

Today, we'll cover:

- How to design indexes for a query without a plan
- How to pick order of keys in an index
- How parameter values can change index needs
- Where to find index recommendations in plans, DMVs, and sp_BlitzIndex
- How SQL Server's index recommendations are built, and why they're often wrong

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

We're using Stack Overflow data.

Open source, licensed with Creative Commons

XML dump: archive.org/details/stackexchange

SQL Server db: BrentOzar.com/go/querystack

I'm using the StackOverflow2013 database (50GB)

- You can use smaller or larger ones depending on your hardware
- Index creations may just take longer
- Exact logical page reads will be different

About my setup

SQL Server 2019, currently patched

SQL Server Management Studio 18

StackOverflow2013 50GB database

4 cores, 30GB RAM, SSDs
(to make index creation fast)

This module's agenda

Learn how to visualize an index's contents

Building indexes for the WHERE clause

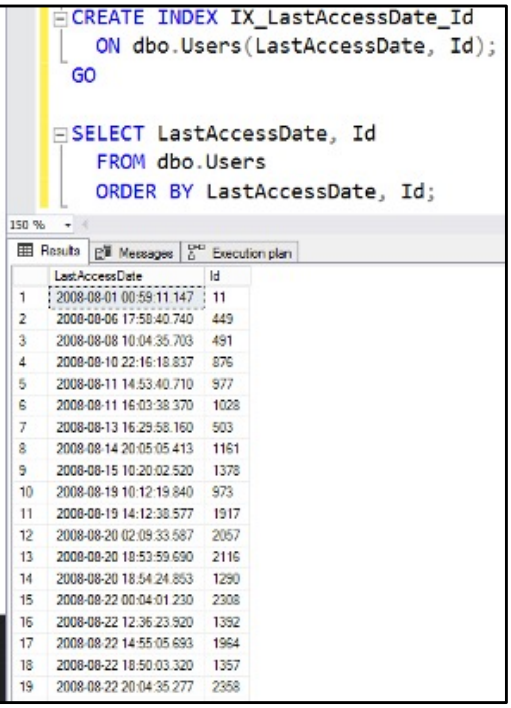
Understanding how selectivity determines key order

To visualize an index, select its contents.

In “How to Think Like the Engine,” we created the black pages: an index on LastAccessDate, Id.

To understand what the data looks like on disk, write a SELECT to match the index: same fields in the SELECT and in the ORDER BY.

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



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

SELECT LastAccessDate, Id
  FROM dbo.Users
  ORDER BY LastAccessDate, Id;
```

150 %

Results | Messages | Execution plan

	LastAccessDate	Id
1	2008-08-01 00:59:11.147	11
2	2008-08-06 17:50:40.740	449
3	2008-08-08 10:04:35.703	491
4	2008-08-10 22:16:18.837	876
5	2008-08-11 14:53:40.710	977
6	2008-08-11 16:03:38.370	1028
7	2008-08-13 16:29:58.160	503
8	2008-08-14 20:05:05.413	1161
9	2008-08-15 10:20:02.520	1378
10	2008-08-19 10:12:19.840	973
11	2008-08-19 14:12:38.577	1917
12	2008-08-20 02:09:33.587	2057
13	2008-08-20 18:53:59.690	2116
14	2008-08-20 18:54:24.853	1290
15	2008-08-22 00:04:01.230	2308
16	2008-08-22 12:36:23.920	1392
17	2008-08-22 14:55:05.693	1964
18	2008-08-22 18:50:03.320	1357
19	2008-08-22 20:04:35.277	2358

That's how I built the handouts.

I simply took that data, copy/pasted it into Excel, and formatted it into columns.

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

```
CREATE INDEX IX_LastAccessDate_Id_DisplayName_Age
ON dbo.Users (LastAccessDate, Id) INCLUDE (DisplayName, Age)

Becomes:

SELECT LastAccessDate, Id, DisplayName, Age
FROM dbo.Users
ORDER BY LastAccessDate, Id
```

dbo.Users - IX_LastAccessDate_ID_DisplayName_Age

LastAccessDate	Id	DisplayName	Age	LastAccessDate	Id	DisplayName	Age
7/31/08 12:00 AM	-1	Community	1	8/18/09 10:22 AM	673		NULL
7/15/09 9:58 AM	587		NULL	8/18/09 1:14 PM	643	Sarcastic	30
7/15/09 10:00 AM	594	Louis Haußknechl	NULL	8/18/09 2:28 PM	690	msh	31
7/15/09 10:02 AM	597	elo80ka	29	8/19/09 9:22 AM	750	Cyphus	30
7/15/09 10:21 AM	618	Marien	30	8/20/09 3:45 AM	749		NULL
7/15/09 10:26 AM	623	Fionn	NULL	8/20/09 12:05 PM	752	Sean Taylor	27
7/15/09 10:28 AM	629	Zoomzoom83	26	8/22/09 11:58 AM	580	n00ki3	22
7/15/09 10:32 AM	638		NULL	8/26/09 2:43 AM	851	Tim Howland	41
7/15/09 10:36 AM	642	aherrick	23	8/26/09 9:59 AM	546	hitec	27
7/15/09 10:46 AM	661		NULL	8/28/09 7:51 PM	668	mtruesdell	34

Use this process as you work.

When you're designing indexes, write a query with a matching SELECT & WHERE.

Review the data that comes out.

Think like the engine: try to use the query results as if they're 8KB pages. Will they help execute the query you're trying to tune?

WHERE
with 1 equality search

Design an index for this query.

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex';
```

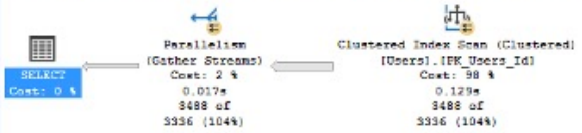
Clippy has an idea:

```
101  /* Design an index for this: */
102  SELECT Id, DisplayName, Location
103    FROM dbo.Users
104    WHERE DisplayName = 'alex';
```

150 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT [Id],[DisplayName],[Location] FROM [dbo].[Users] WHERE [DisplayName]=@1
Missing Index (Impact 99.8523): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Users] ([DisplayName])
```



I'm not cutting that screenshot off, either:
Clippy didn't suggest that we include Location.

Clippy's hints are a gift.

But they're just a lucky byproduct of query plan optimization.

They're not Clippy's main job.

Later today, we'll explain how he builds them and why they're usually wrong.

For now, focus on building your own.

I bet you could do better. Try it.

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex';
```

Did you come up with this?

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex';

CREATE INDEX IX_DisplayName_Includes
  ON dbo.Users(DisplayName)
  INCLUDE (Location);
```

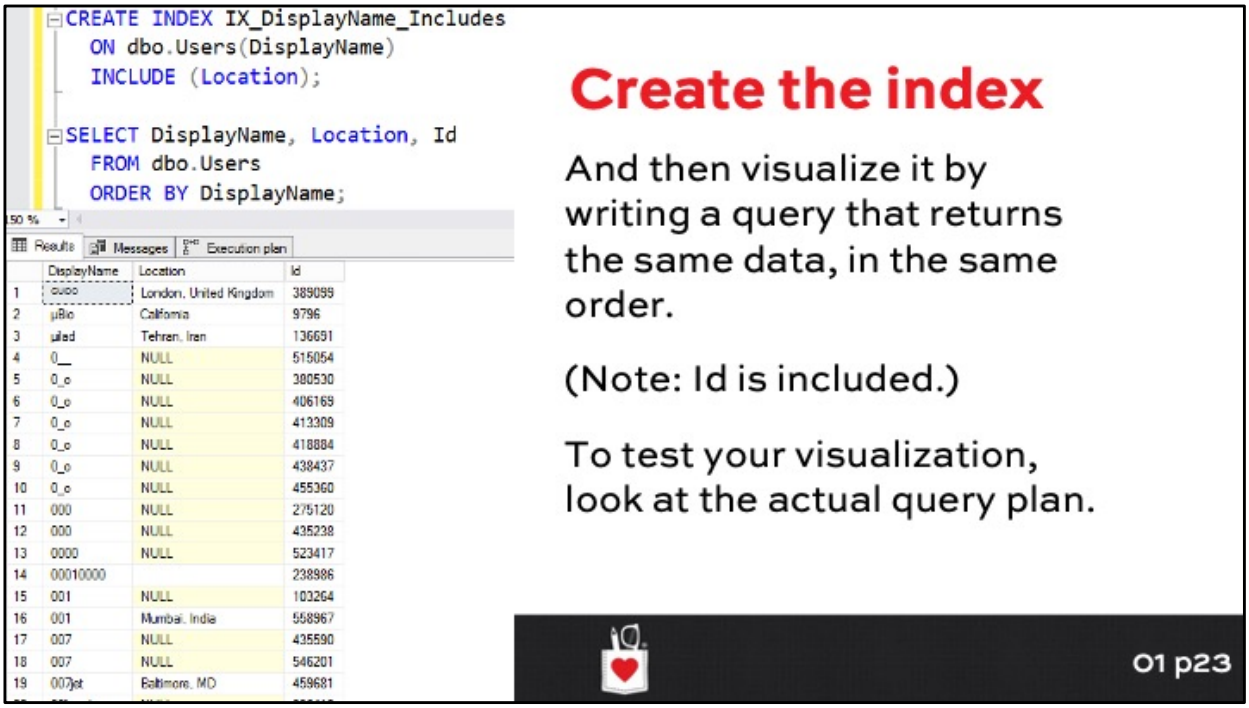
Should you include Id?

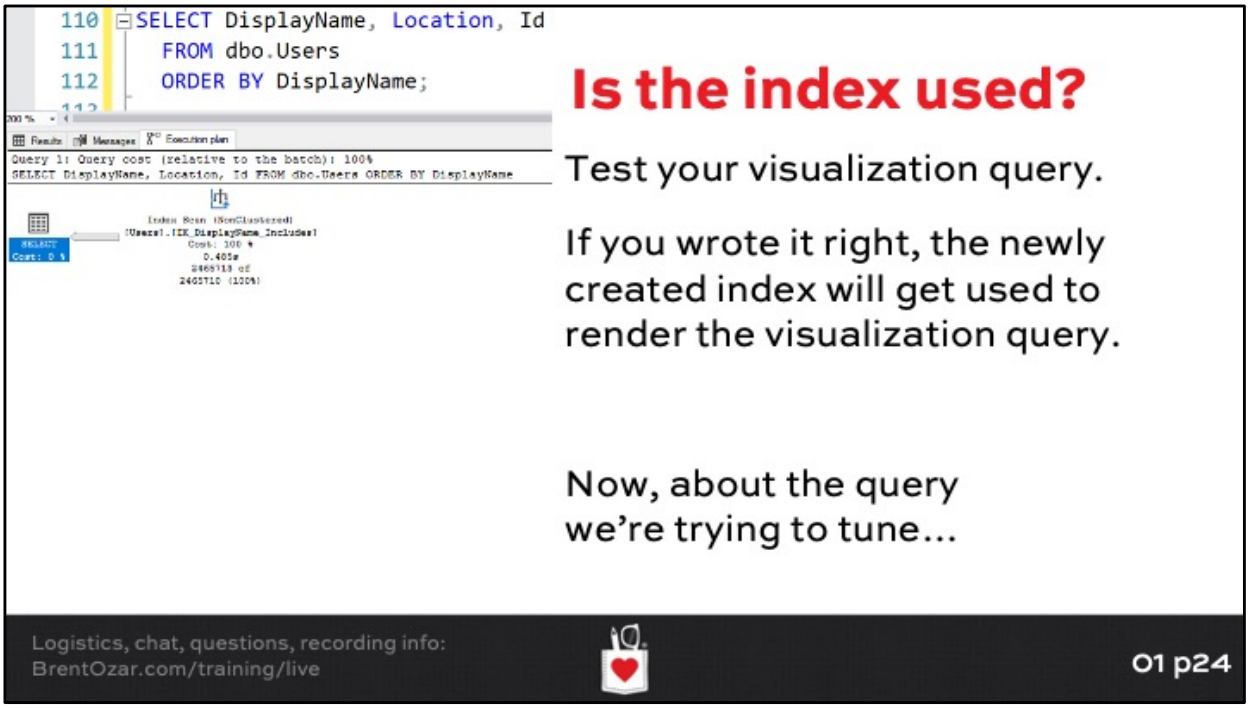
How to Think Like the Engine explained that the clustering key on a table is always included.

There's no extra cost whether you include it or not: it doesn't get stored twice.

I only include it if my query needs it in the output, and I suspect somebody's gonna come behind me and change the clustering key later.

Here, I'm fine either way.





Ta-dah!

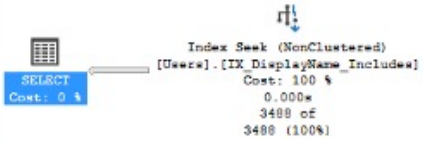
If we created the right index, the optimizer uses it.

```
114 SELECT Id, DisplayName, Location
115   FROM dbo.Users
116   WHERE DisplayName = 'alex';
117
```

150 %

Results | Messages | Execution plan

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



WHERE
with 2 equality searches

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

01 p26

Now try this query.

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

A couple of common solutions

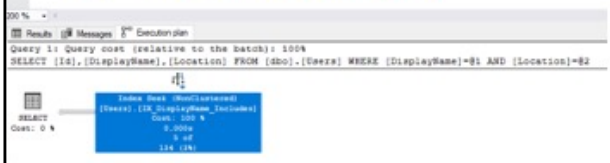
```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex'
    AND Location = 'Seattle, WA';

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

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

But remember that last index?

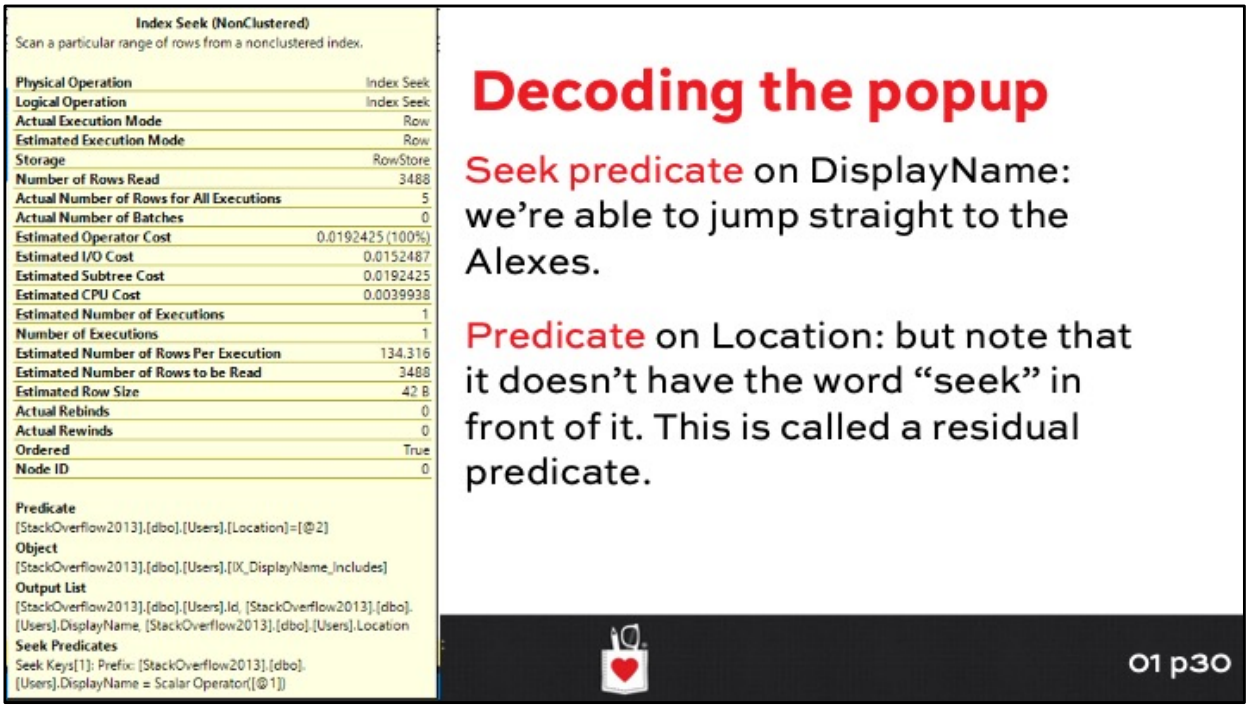
```
1  CREATE INDEX IX_DisplayName_Includes
2    ON dbo.Users(DisplayName) INCLUDE (Location);
3  GO
4  SELECT Id, DisplayName, Location
5    FROM dbo.Users
6    WHERE DisplayName = N'alex'
7    AND Location = N'Seattle, WA';
```

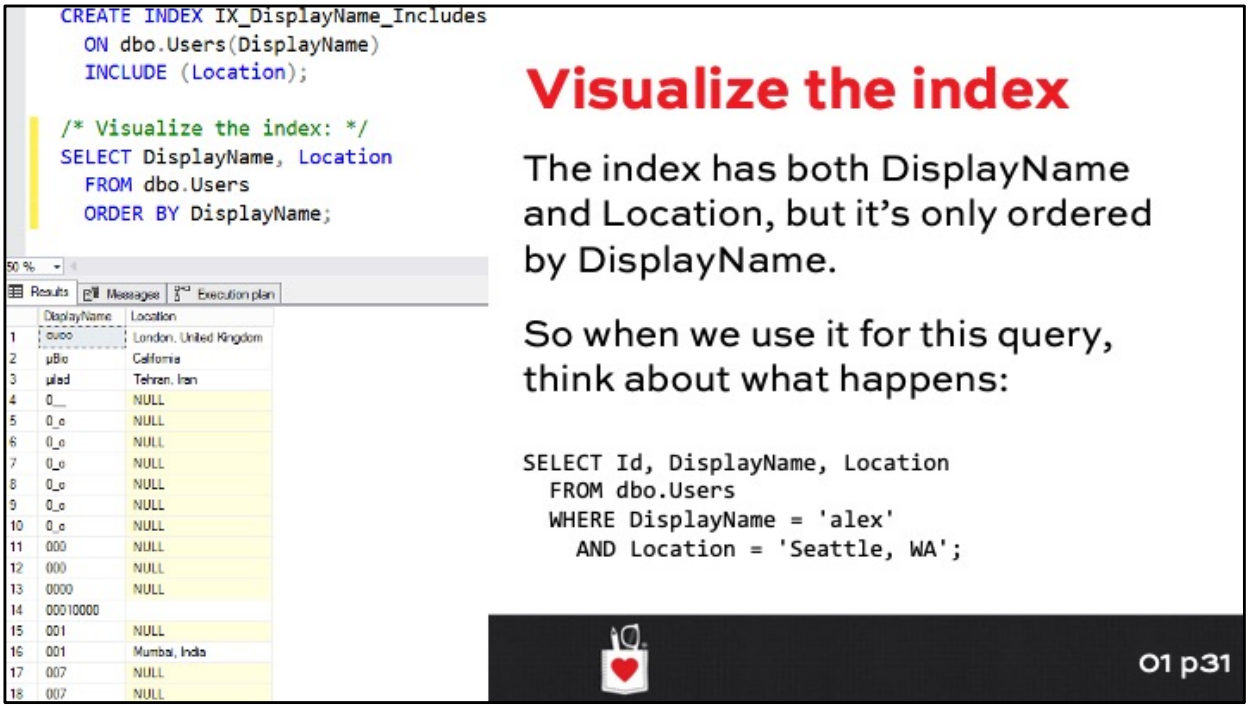


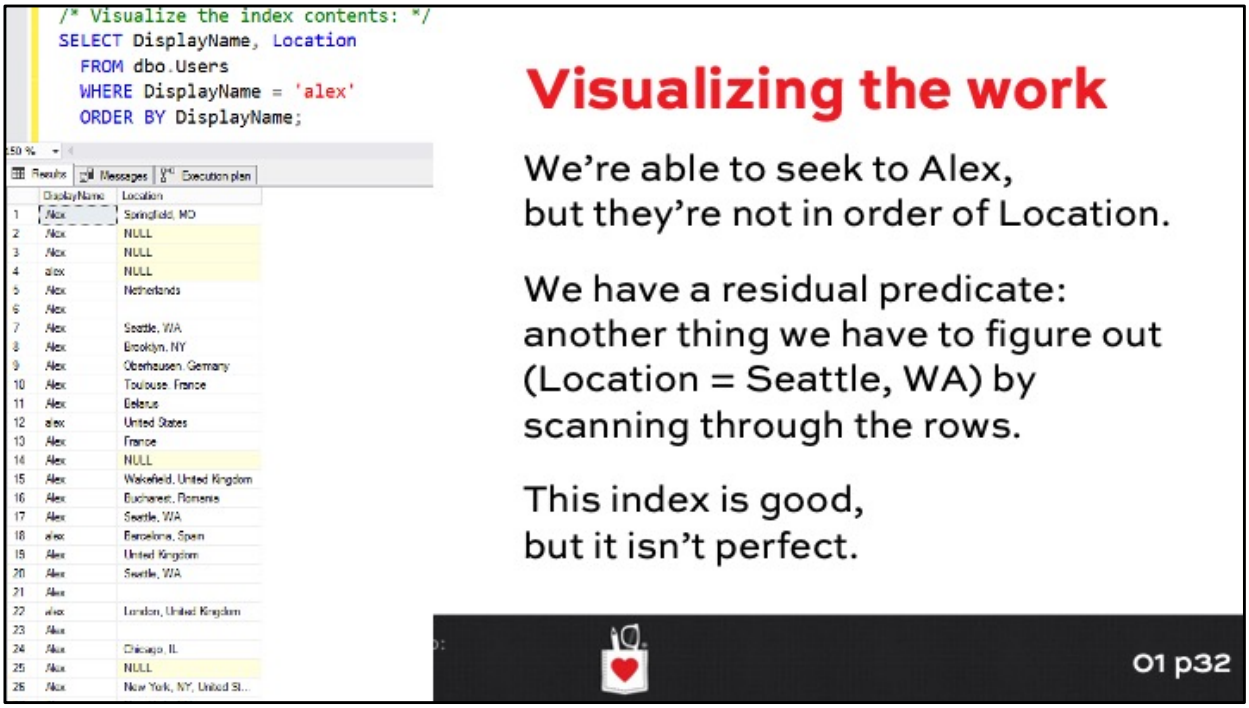
SQL Server can use that last index we created.

Sure, Location isn't sorted in order, but we can still:

1. Seek to Alex
2. Scan through them, checking their locations





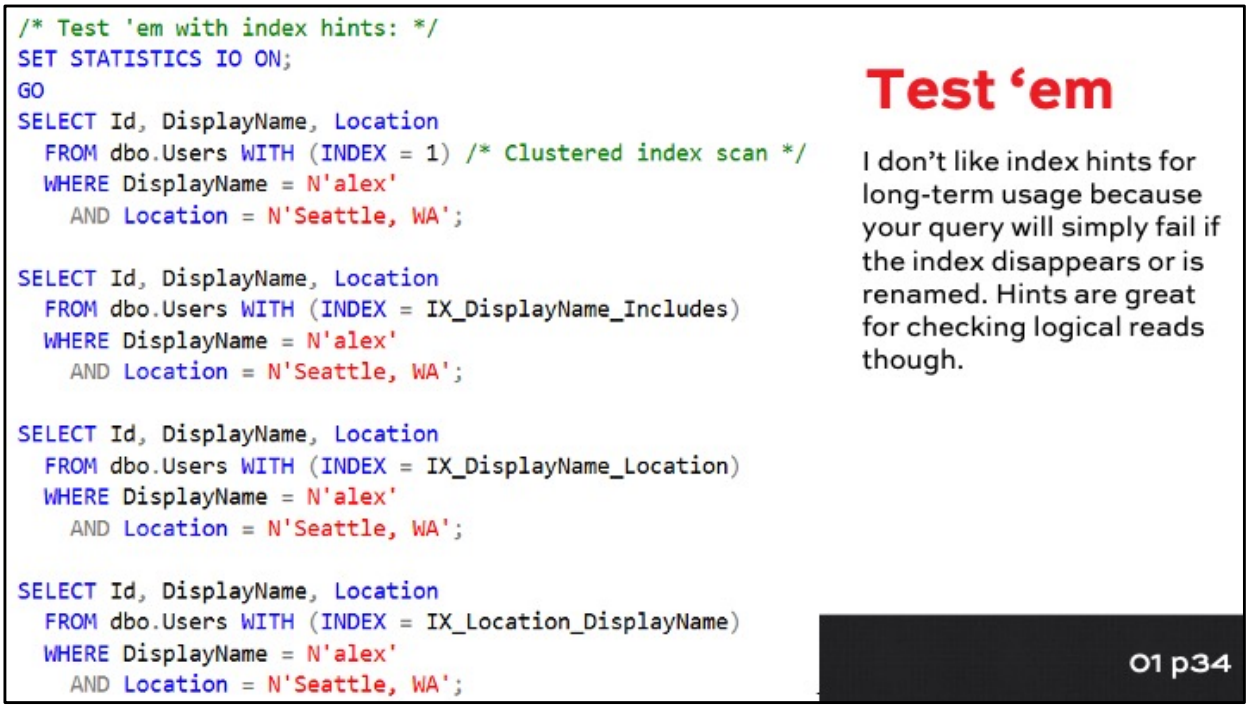


Maybe your indexes are better. Let's create 'em and try 'em.

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

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

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



Survey says...

Index	Logical Reads
Clustered index (white pages)	45,184
IX_DisplayName_Includes	16
IX_DisplayName_Location	4
IX_Location_DisplayName	5

But don't quibble over a handful of logical reads.

All of the indexes are pretty good!

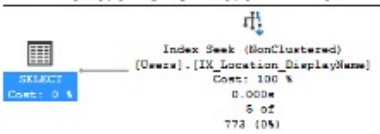
Without a hint, what gets used?

```
173  /* Which one does SQL Server pick? */
174 SELECT Id, DisplayName, Location
175   FROM dbo.Users
176   WHERE DisplayName = N'alex'
177     AND Location = N'Seattle, WA';
178  GO
```

150 %

Results | Messages | Execution plan

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



SQL Server uses the Location, DisplayName index.

Don't read too much into that yet though: it doesn't mean it's the one you should keep.

Before you pick a winner...

You never a SQL Server that only runs one query.

What if this query still runs sometimes?

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex';
```

You wouldn't want one index for this query, and a different index for Location, DisplayName.

WHERE
with both equality and inequality searches

Because I know you're gonna ask

Equality searches	Inequality searches
=	<>, >, <, >=, <=
IS NULL	IS NOT NULL
IN (one value)	IN (two or more values)
	LIKE, NOT LIKE

Now try this query.

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

The <> is really important: it changes the game.

Think back to your 2 earlier indexes.

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

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

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

Your execution plan...

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

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

"I'll seek to DisplayName = Alex,
then read through them, looking at their locations,
returning all the ones who aren't in Seattle."

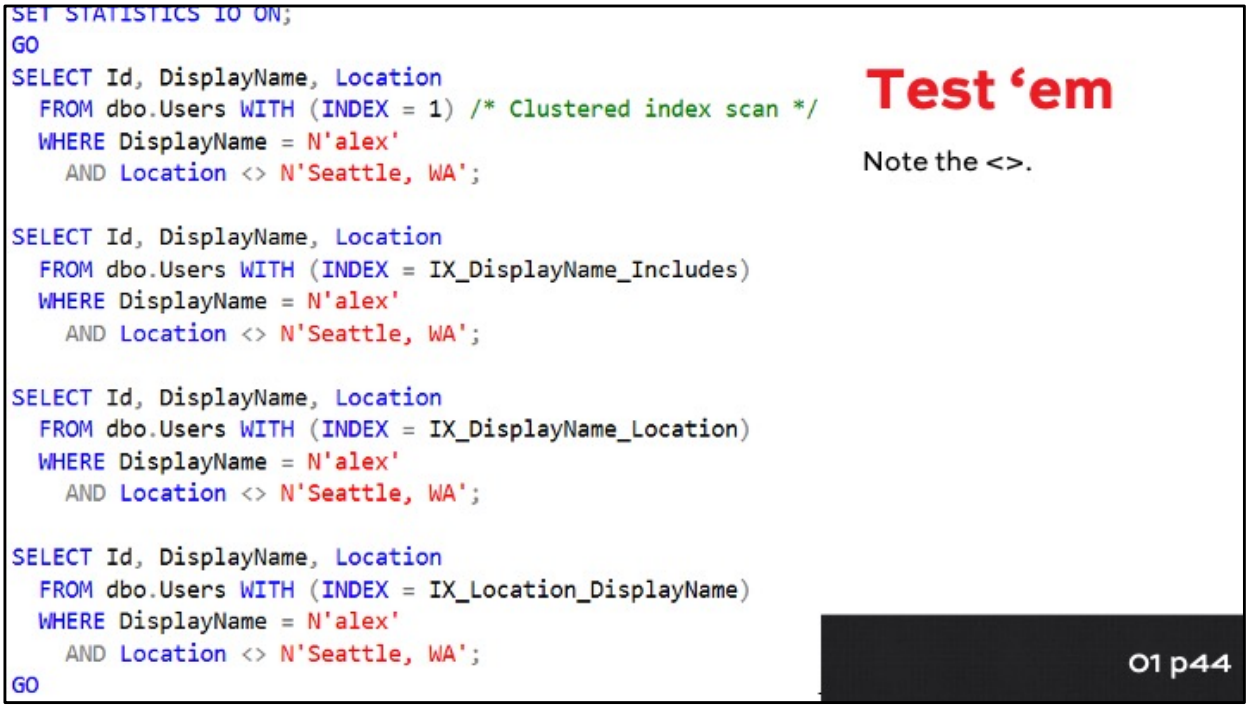
But if the index starts with Location?

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

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

"I can't just jump to the Alexes."

"I'm gonna have to scan most of the index."



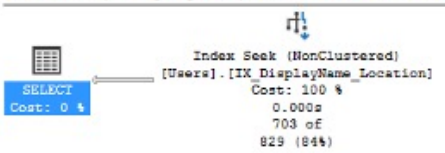
Survey says...

Index	Logical Reads	Total Pages in the Index
Clustered index (white pages)	45,184	45,184
IX_DisplayName_Includes	16	12,577
IX_DisplayName_Location	13	12,701
IX_Location_DisplayName	4,566	13,183

Let's compare the actual execution plans for the bottom two.

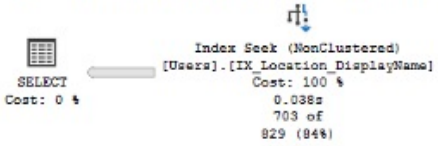
They look similar at first glance.

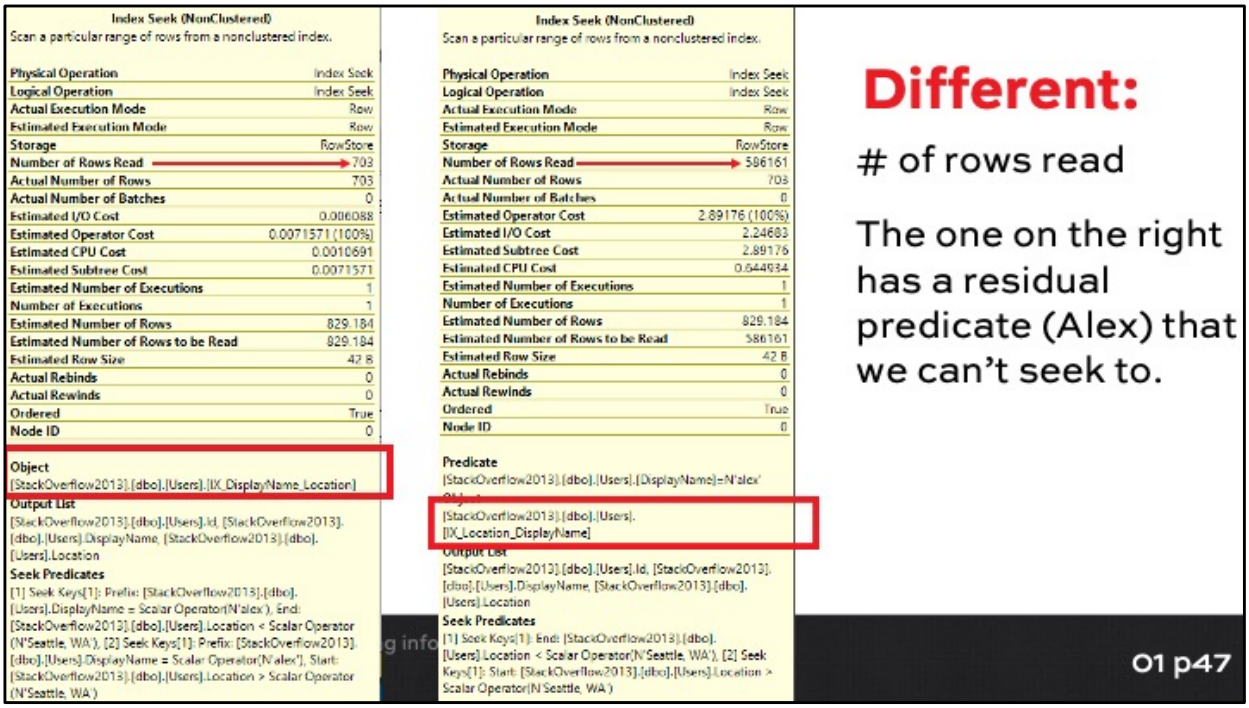
```
Query 1: Query cost (relative to the batch): 0%
SELECT Id, DisplayName, Location FROM dbo.Users WITH (INDEX = IX_DisplayName_Location)
```



But hover your mouse over each seek to see details...

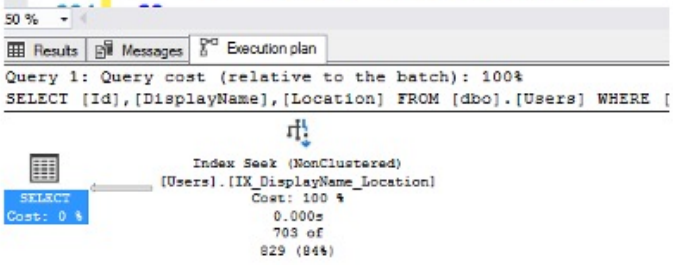
```
Query 2: Query cost (relative to the batch): 100%
SELECT Id, DisplayName, Location FROM dbo.Users WITH (INDEX = IX_Location_DisplayName)
```





Which one did SQL Server pick?

```
219  /* Which one does SQL Server pick? */
220 SELECT Id, DisplayName, Location
221   FROM dbo.Users
222   WHERE DisplayName = N'alex'
223     AND Location <> N'Seattle, WA';
```



When not hinted, it picked the one on DisplayName, Location.

(That's the right one!)

So what's the lesson?

When you have both equality and inequality searches, you might think it's important to put the equality fields first in the index key order so that you can seek directly to the rows you want.

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

But that's not necessarily true. Hold that thought.

Field order isn't about equality.

Field order is about selectivity, the ability to reduce the amount of work we're about to do.

Sometimes that's about reducing row counts by filtering down the number of rows we're going to pass on to the next operator in a plan.

Other times, it's about pre-sorting data to avoid sorts.

How selective is Alex?

```
SELECT Id, DisplayName, Location
  FROM dbo.Users
  WHERE DisplayName = 'alex';
```

Total rows in table: 2,465,713

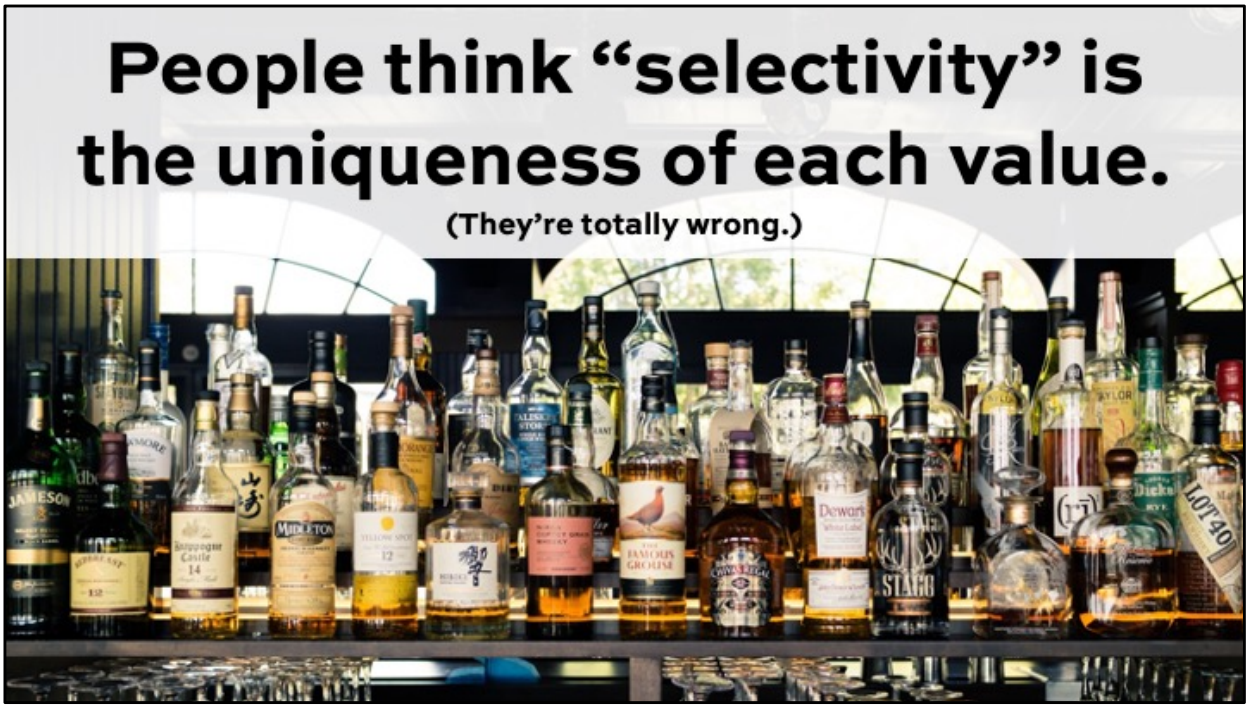
Rows where DisplayName = alex: 3,488 (0.14%)

Comparing selectivity

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

Total rows in table: 2,465,713

- DisplayName = alex: 3,488 (0.14%)
- Location = Seattle, WA: 3,345 (0.14%)
- Location <> Seattle, WA: 586,161 (24%)
  (remainder: nulls)



What selectivity really means

It's not about how unique each row is by itself.

It's about your query,
and how small a percentage of the table
you're searching for.

When evaluating column order for indexes,
don't think about how unique each column is.
Think about what percentage you're searching for.

Testing it out

When designing indexes for a query,
craft a separate SELECT query
for each filter in the WHERE clause,
and test to see how selective it is.

```
SELECT Id, LastAccessDate, DownVotes
  FROM dbo.Users
  WHERE LastAccessDate <= GETDATE()
    AND Reputation = 0;
```

Recap

If your queries only have equality searches, key field order isn't all that important.

When you have inequality searches, though, key field order matters a LOT. The first fields in the key need to help reduce the amount of rows you scan.

Just because you see a "seek" doesn't mean you're seeking to a specific row: residual predicates indicate a seek, followed by a scan of an area of the index.

Picking key order

Fields in the WHERE clause usually* need to go first

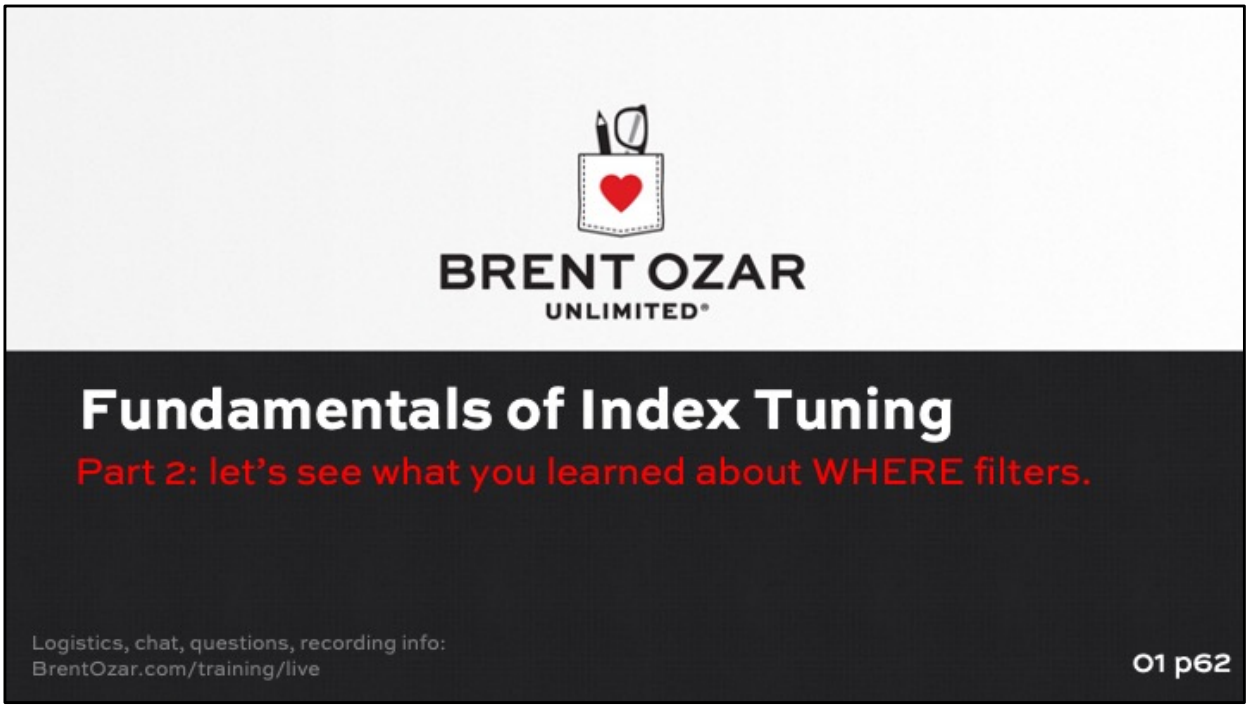
Selective ~~fields~~ query filters go first:
reduce the amount of data you're searching through

Commonly filtered-on fields go first:
maximize the number of queries that can use an index

* We'll break this rule in the next module

Testing it out

When designing indexes for a query,
craft a separate SELECT query
for each filter in the WHERE clause,
and test to see how selective it is.



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, creating and dropping indexes where directed

10 minutes: I'll start the lab with you

45 minutes: you work through the rest, asking questions in Slack as you go

30 minutes: I work through it onscreen