



BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Up first: intros & logistics.

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

O1 p1



BRENT OZAR UNLIMITED

99-05: dev, architect, DBA
05-08: DBA, VM, SAN admin
08-10: MCM, Quest Software
Since: consulting DBA

www.BrentOzar.com
Help@BrentOzar.com

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



01 p2

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



O1 p3

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.

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



O1 p4

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.

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



O1 p5

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



O1 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



O1 p7

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

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



01 p8

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)

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



O1 p9



BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Part 1: The WHERE Clause

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

01 p10

This module's agenda

Learn how to visualize an index's contents

Building indexes for the WHERE clause

Understanding how selectivity determines key order

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



O1 p11

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;
```

Results

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

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



O1 p15

WHERE with 1 equality search

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



O1 p16

Design an index for this query.

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

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



O1 p17

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 99.8528): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Users] ([DisplayName])

    graph TD
        A[Parallelism (Gather Streams)] --> B[Clustered Index Scan (Clustered)]
        B --> C[User1.IPK_Users_Id]
        C --> D[Cost: 99 %]
        D --> E[0.129s]
        E --> F[3488 of]
        F --> G[3336 (104%)]

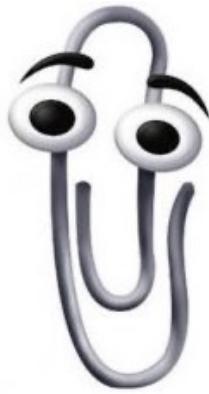
    SELECT
    Cost: 0 %
    Counts: 2 %
    0.017s
    3488 of
    3336 (104%)
```

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

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



O1 p18



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.

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



01 p19

I bet you could do better. Try it.

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

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



01 p20

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.



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

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

	DisplayName	Location	Id
1	dbo	London, United Kingdom	389099
2	jillo	California	9796
3	uled	Tehran, Iran	136691
4	0_	NULL	515054
5	0_o	NULL	380530
6	0_o	NULL	406168
7	0_o	NULL	413209
8	0_o	NULL	418884
9	0_o	NULL	438437
10	0_o	NULL	455360
11	000	NULL	276120
12	000	NULL	435238
13	0000	NULL	523417
14	00010000		238986
15	001	NULL	103264
16	001	Mumbai, India	558967
17	007	NULL	435590
18	007	NULL	546201
19	00jet	Baltimore, MD	459681

Create the index

And then visualize it by writing a query that returns the same data, in the same order.

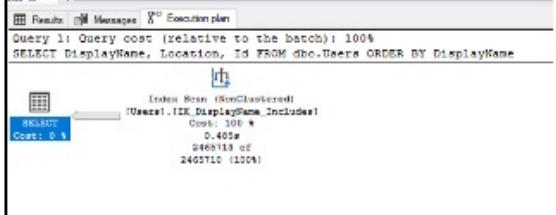
(Note: Id is included.)

To test your visualization, look at the actual query plan.



O1 p23

```
110 SELECT DisplayName, Location, Id  
111     FROM dbo.Users  
112 ORDER BY DisplayName;  
113
```



Is the index used?

Test your visualization query.

If you wrote it right, the newly created index will get used to render the visualization query.

Now, about the query we're trying to tune...

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



O1 p24

Ta-dah!

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

The screenshot shows a SQL query in the query editor:

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

The execution plan for this query is displayed below the query results. It shows an "Index Seek (NonClustered)" operation on the "[Users].[IX_DisplayName_Included]" index. The cost of the query is 100%, and the cost of the seek operation is 0.0000. The seek operation processes 3488 rows, which is 100% of the total.

```
Index Seek (NonClustered)  
[Users].[IX_DisplayName_Included]  
Cost: 100 %  
0.0000  
3488 of  
3488 (100%)
```

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



O1 p25

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';
```

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



01 p27

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);
```

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



O1 p28

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';
```

200 % Results Messages Execution plan

Query 1: Query cost (relative to the batch): 1000

SELECT [Id], [DisplayName], [Location] FROM [dbo].[Users] WHERE [DisplayName]=@1 AND [Location]=@2

Index Seek (NonClustered)

(1 row(s) affected)

Cost: 0.9

Cost: 0.9

Column	Type	Length	Nullability
[Id]	int	4	Yes
[DisplayName]	varchar	50	Yes
[Location]	varchar	50	Yes

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

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



01 p29

Index Seek (NonClustered)	
Scan a particular range of rows from a nonclustered index.	
Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	3488
Actual Number of Rows for All Executions	5
Actual Number of Batches	0
Estimated Operator Cost	0.0192425 (100%)
Estimated I/O Cost	0.0152487
Estimated Subtree Cost	0.0192425
Estimated CPU Cost	0.0039938
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows Per Execution	134.316
Estimated Number of Rows to be Read	3488
Estimated Row Size	42 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	0
Predicate	
[StackOverflow2013].[dbo].[Users].[Location]=[@2]	
Object	[StackOverflow2013].[dbo].[Users].[IX_DisplayName_Includes]
Output List	[StackOverflow2013].[dbo].[Users].Id, [StackOverflow2013].[dbo].[Users].DisplayName, [StackOverflow2013].[dbo].[Users].Location
Seek Predicates	
Seek Keys[1]: Prefix: [StackOverflow2013].[dbo].[Users].DisplayName = Scalar Operator(@1)]	

Decoding the popup

Seek predicate on DisplayName:
we're able to jump straight to the Alexes.

Predicate on Location: but note that it doesn't have the word "seek" in front of it. This is called a residual predicate.



01 p30

```

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

/* Visualize the index: */
SELECT DisplayName, Location
FROM dbo.Users
ORDER BY DisplayName;

```

Results

DisplayName	Location
subo	London, United Kingdom
pBio	California
plad	Tehran, Iran
0_	NULL
0_o	NULL
0_0	NULL
0_0	NULL
0_o	NULL
0_o	NULL
0_0	NULL
000	NULL
000	NULL
0000	NULL
00010000	NULL
001	NULL
001	Mumbai, India
007	NULL
007	NULL

Visualize the index

The index has both DisplayName and Location, but it's only ordered by DisplayName.

So when we use it for this query, think about what happens:

```

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

```



O1 p31

```
/* Visualize the index contents: */
SELECT DisplayName, Location
FROM dbo.Users
WHERE DisplayName = 'alex'
ORDER BY DisplayName;
```

	DisplayName	Location
1	Alex	Sandfield, MO
2	Alex	NULL
3	Alex	NULL
4	alex	NULL
5	Alex	Netherlands
6	Alex	NULL
7	Alex	Seattle, WA
8	Alex	Brooklyn, NY
9	Alex	Oberhausen, Germany
10	Alex	Toulouse, France
11	Alex	United States
12	alex	France
13	Alex	NULL
14	Alex	Wakefield, United Kingdom
15	Alex	Bucharest, Romania
16	Alex	Seattle, WA
17	Alex	Barcelona, Spain
18	Alex	United Kingdom
19	Alex	Seattle, WA
20	Alex	London, United Kingdom
21	Alex	NULL
22	alex	Chicago, IL
23	Alex	NULL
24	Alex	New York, NY, United St...

Visualizing the work

We're able to seek to Alex, but they're not in order of Location.

We have a residual predicate: another thing we have to figure out (Location = Seattle, WA) by scanning through the rows.

This index is good, but it isn't perfect.



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);
```

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



01 p33

```

/* Test 'em with index hints: */
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';

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

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

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

```

Test 'em

I don't like index hints for long-term usage because your query will simply fail if the index disappears or is renamed. Hints are great for checking logical reads though.

O1 p34

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!

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



01 p35

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 [
```

Index Seek (NonClustered)
[Users].[IX_Location_DisplayName]
Cost: 100 %
0.000s
6 of
773 (0%)

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.

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



01 p36

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

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



01 p38

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

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



01 p39

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.

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



01 p40

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.”

```
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';

SELECT Id, DisplayName, Location
  FROM dbo.Users WITH (INDEX = IX_DisplayName_Includes)
 WHERE DisplayName = N'alex'
   AND Location <> N'Seattle, WA';

SELECT Id, DisplayName, Location
  FROM dbo.Users WITH (INDEX = IX_DisplayName_Location)
 WHERE DisplayName = N'alex'
   AND Location <> N'Seattle, WA';

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

Test 'em

Note the <>.

O1 p44

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.

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

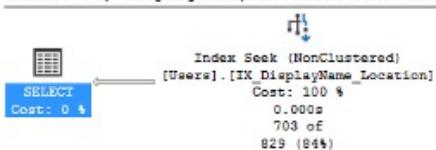


O1 p45

They look similar at first glance.

Query 1: Query cost (relative to the batch): 0%

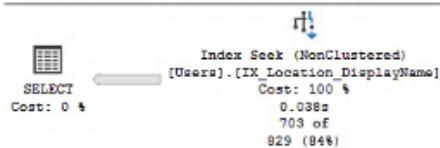
```
SELECT Id, DisplayName, Location FROM dbo.Users WITH (INDEX = IX_DisplayName_Location)
```



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)
```



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



01 p46

Index Seek (NonClustered)	
Scan a particular range of rows from a nonclustered index.	
Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	703
Actual Number of Rows	703
Actual Number of Batches	0
Estimated I/O Cost	0.006088
Estimated Operator Cost	0.0071571 (100%)
Estimated CPU Cost	0.0010691
Estimated Subtree Cost	0.0071571
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	829.184
Estimated Number of Rows to be Read	829.184
Estimated Row Size	42.8
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	0
Object	[StackOverflow2013].[dbo].[Users].[IX_DisplayName_Location]
Output List	[StackOverflow2013].[dbo].[Users].Id, [StackOverflow2013].[dbo].[Users].DisplayName, [StackOverflow2013].[dbo].[Users].Location
Seek Predicates	[1] Seek Keys[1]: Prefix: [StackOverflow2013].[dbo].[Users].DisplayName = Scalar Operator('alex'), End: [StackOverflow2013].[dbo].[Users].Location < Scalar Operator(N'Seattle, WA'), [2] Seek Keys[1]: Prefix: [StackOverflow2013].[dbo].[Users].DisplayName = Scalar Operator('alex'), Start: [StackOverflow2013].[dbo].[Users].Location > Scalar Operator(N'Seattle, WA')
Predicate	[StackOverflow2013].[dbo].[Users].[IX_Location_DisplayName]
Output List	[StackOverflow2013].[dbo].[Users].Id, [StackOverflow2013].[dbo].[Users].DisplayName, [StackOverflow2013].[dbo].[Users].Location
Seek Predicates	[1] Seek Keys[1]: End: [StackOverflow2013].[dbo].[Users].Location < Scalar Operator(N'Seattle, WA'), [2] Seek Keys[1]: Start: [StackOverflow2013].[dbo].[Users].Location > Scalar Operator(N'Seattle, WA')

Different:

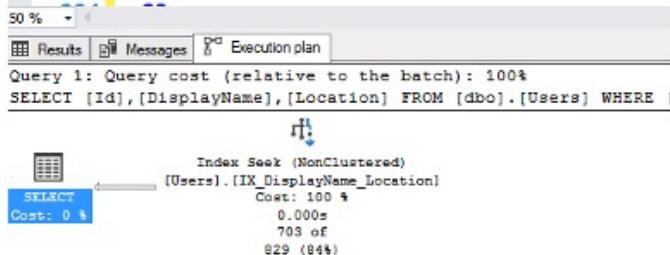
of rows read

The one on the right has a residual predicate (Alex) that we can't seek to.

O1 p47

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!)

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



O1 p48

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.

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



01 p49

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.

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



01 p50



Selectivity

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)



People think “selectivity” is the uniqueness of each value.

(They’re totally wrong.)



```
SELECT * FROM dbo.Booze  
WHERE BrandName LIKE '%e%';
```



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.

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



01 p56

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;
```

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



O1 p57



Re-cap

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.

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



01 p59

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

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



01 p60

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.

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



01 p61



BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Part 2: let's see what you learned about WHERE filters.

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

01 p62

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

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



01 p63

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

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



O1 p64