BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Part 5: Indexing for JOINs

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

05 p1

Almost never happens in real life, but

ONE JOIN

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

05 p2

Show everyone's comments

```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
     ON u.Id = c.UserId;
```

Don't run this. It'll take forever.
And in reality, you'd never write this query.
Just get the estimated plan.

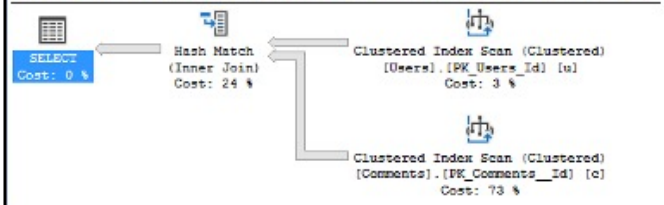
“I’m gonna scan both tables.”

```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
     ON u.Id = c.UserId;
```

150 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT u.DisplayName, c.CreationDate, c.Text   FROM dbo.Users u I
Missing Index (Impact 96.7066): CREATE NONCLUSTERED INDEX [<Name
```



Yes, it’s asking for an index, but not for selectivity.

It’s just a narrower copy of the table with only the fields we need.

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

05 p4

The index Clippy wants

```
Query 1: Query cost (relative to the batch): 100%
SELECT u.DisplayName, c.CreationDate, c.Text   FROM dbo.Users u INNER JOIN dbo.Comments c ON u.Id = c.UserId
Missing Index (Impact 96.7066): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Comments] ([UserId]) INCLUDE ([CreationDate],[Text])
```

- dbo.Comments
  - Columns
    - Id (PK, int, not null)
    - CreationDate (datetime, not null)
    - PostId (int, not null)
    - Score (int, null)
    - Text (nvarchar(700), not null)
    - UserId (int, null)

The Text is the comment itself.

This index is nearly the size of the entire table!

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

05 p5

A more realistic query:

```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
     ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar';
```

Run this, get the actual plan,
and add indexes to make it faster.

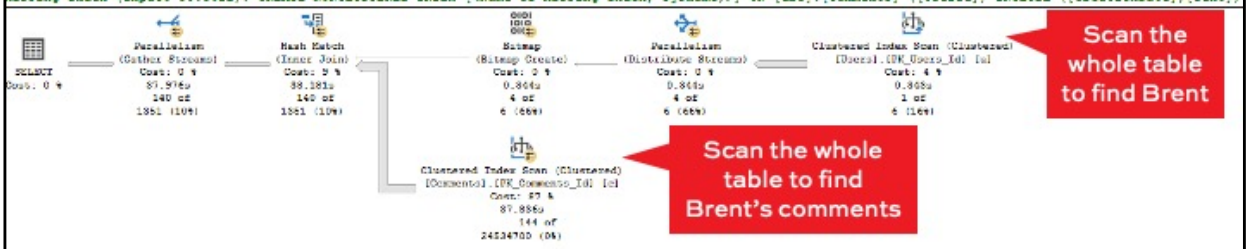
Logistics, chat, questions, recording info:
BrentOzar.com/training/live

We probably need 2 indexes. Users needs a filter, and Comments needs to be sorted by UserId to make it easier to find Brent's comments:

```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar'
GO
```

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT u.DisplayName, c.CreationDate, c.Text FROM dbo.Users u INNER JOIN dbo.Comments c ON u.Id = c.UserId WHERE u.DisplayName = 'Brent Ozar'
Missing Index (Impact 95.8622): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Comments] ([UserId]) INCLUDE ([CreationDate],[Text])
```



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

05 p7

These two indexes will help:

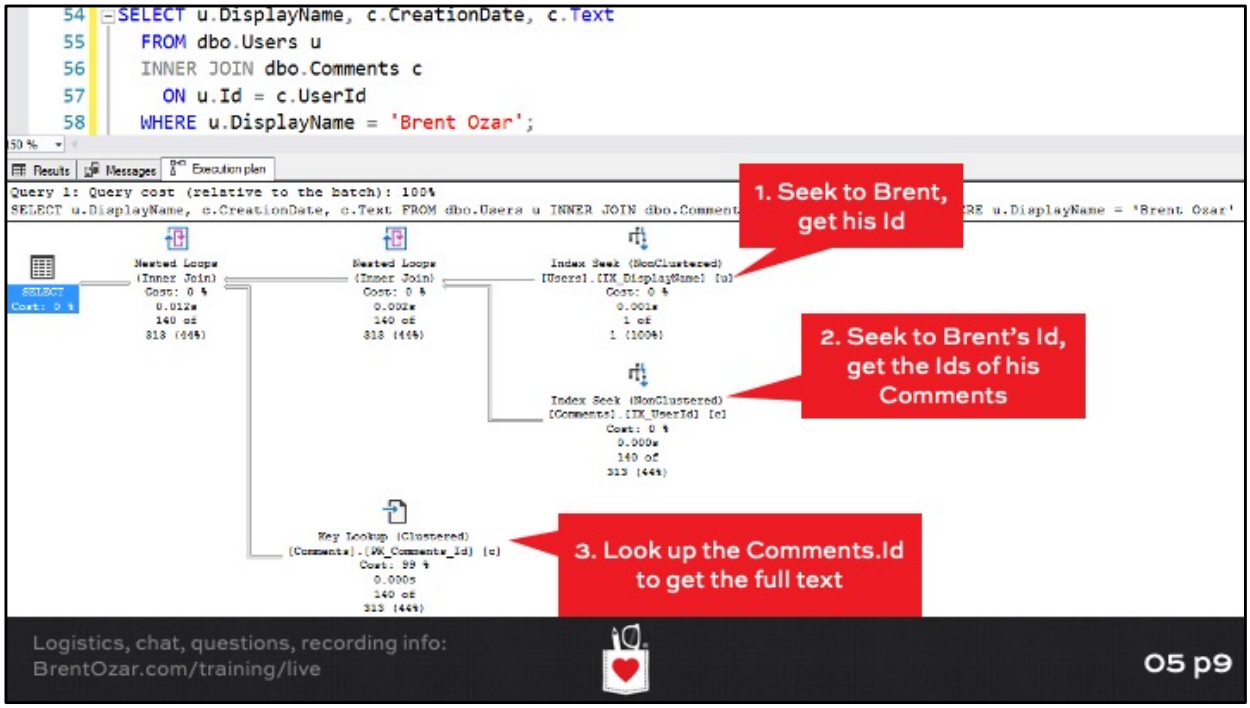
```
CREATE INDEX IX_DisplayName ON dbo.Users(DisplayName);

CREATE INDEX IX_UserId ON dbo.Comments(UserId);
```

Note that Clippy only suggested one index.
(He ain't perfect. More on that later.)

Also note that I didn't include Comments.Text.

```
54 SELECT u.DisplayName, c.CreationDate, c.Text
55   FROM dbo.Users u
56   INNER JOIN dbo.Comments c
57     ON u.Id = c.UserId
58   WHERE u.DisplayName = 'Brent Ozar';
```



Do we include Text in the index?

The more fields you include, the bigger your index is.

Here, the answer is no.

We analyze the tradeoffs in Mastering Index Tuning.

```
54  SELECT u.DisplayName, c.CreationDate, c.Text
55    FROM dbo.Users u
56    INNER JOIN dbo.Comments c
57      ON u.Id = c.UserId
58    WHERE u.DisplayName = 'Brent Ozar';
```

Results | Messages | Execution plan

```
(140 rows affected)
Table 'Comments'. Scan count 1, logical reads 825, physical reads 0,
Table 'Users'. Scan count 1, logical reads 3, physical reads 2, read-
```

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

That was a JOIN with a WHERE.

```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
     ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar';
```

But now let's make it a little trickier...

a one, and a two

JOIN + ORDER BY

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

05 p12

That was a JOIN with a WHERE.

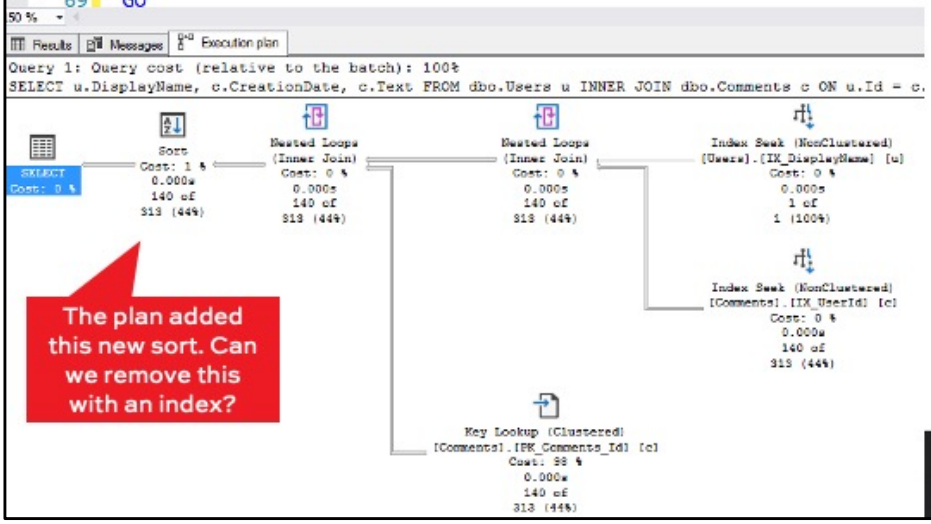
```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
     ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar'
  ORDER BY c.CreationDate;
```

What's the right index on Comments?

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

```sql
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar'
  ORDER BY c.CreationDate;
GO
```

With our current indexes...

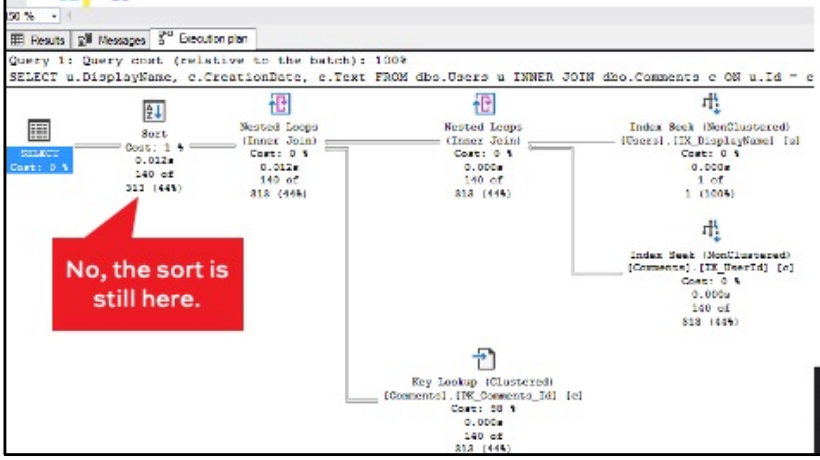


The plan added this new sort. Can we remove this with an index?

05 p14

```sql
/* Does a separate index on CreationDate help? */
CREATE INDEX IX_CreationDate ON dbo.Comments(CreationDate);
GO
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar'
  ORDER BY c.CreationDate;
GO
```

Can we just index CreationDate?



No, the sort is still here.

05 p15

SQL Server already has its data.

It already got all the data it needed from:

1. The index seek on Comments.UserId
2. The key lookup on the Comments clustered index

It won't go back later and use
another index to support a sort.
It's gotta already be sorted after we get it.

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

05 p16

Phone book example

"Find all the businesses that start with Smith%."

"Then, alphabetize them by business type."

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

05 p17

Phone book example

"Find all the businesses that start with Smith%."

First: use the white pages to find them.

"Then, alphabetize them by business type."

Second: use the yellow pages to sort them?

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

Instead, change the existing index.

```
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
     ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar'
  ORDER BY c.CreationDate;
```

We have an index on Comments.UserId.

What if we just add CreationDate as a second key?

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

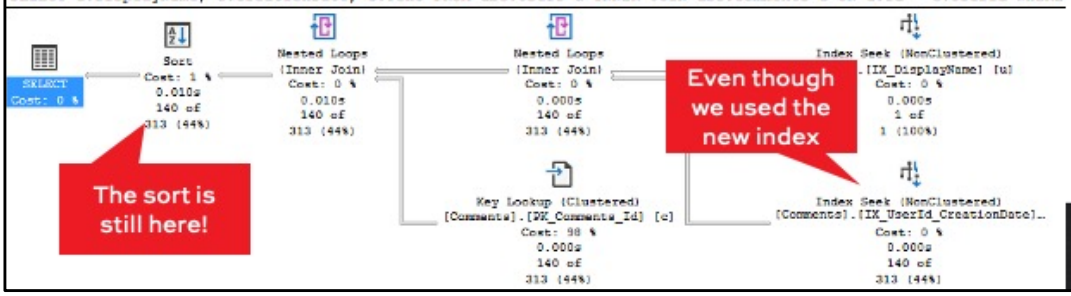
```sql
/* What if we widen up the index on UserId, CreationDate? */
CREATE INDEX IX_UserId_CreationDate ON dbo.Comments(UserId, CreationDate);
GO
SELECT u.DisplayName, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'Brent Ozar'
  ORDER BY c.CreationDate;
GO
```

Add this

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT u.DisplayName, c.CreationDate, c.Text FROM dbo.Users u INNER JOIN dbo.Comments c ON u.Id = c.UserId WHERE



Even though we used the new index

The sort is still here!

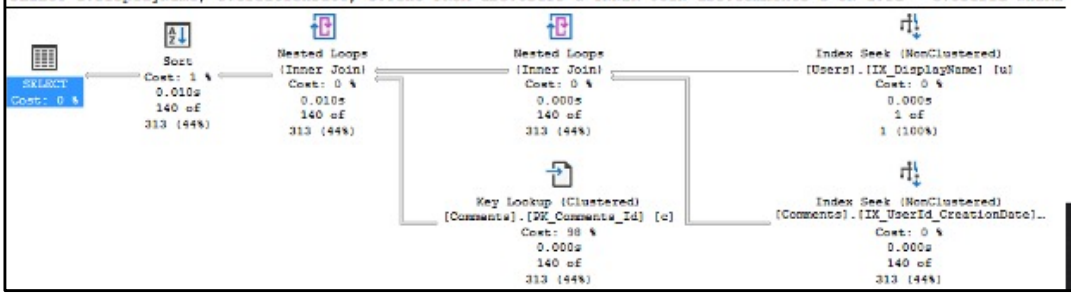
05 p20

```
84  /* What if we widen up the index on UserId, CreationDate? */
85  CREATE INDEX IX_UserId_CreationDate ON dbo.Comments(UserId, CreationDate);
86  GO
87  SELECT u.DisplayName, c.CreationDate, c.Text
88    FROM dbo.Users u
89    INNER JOIN dbo.Comments c
90      ON u.Id = c.UserId
91    WHERE u.DisplayName = 'Brent Ozar
92    ORDER BY c.CreationDate;
93  GO
94
```

There's no guarantee that this filter only matches one UserId.

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT u.DisplayName, c.CreationDate, c.Text FROM dbo.Users u INNER JOIN dbo.Comments c ON u.Id = c.UserId WHERE



05 p21

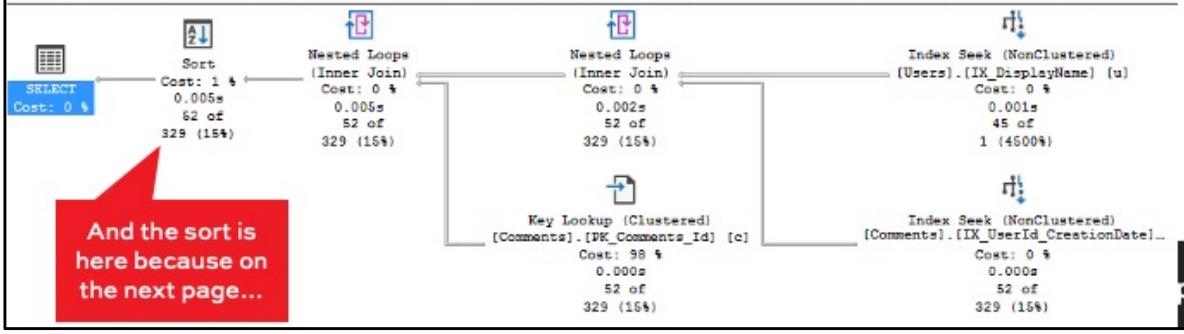
```
/* To understand why, use a different user name and show the User.Id: */
SELECT u.DisplayName, u.Id, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'JamesBrownIsDead'
  ORDER BY c.CreationDate;
GO
```

Try a more common user name...

150 %

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT u.DisplayName, u.Id, c.CreationDate, c.Text FROM dbo.Users u INNER JOIN dbo.Comments c ON u.Id = c.UserI



And the sort is here because on the next page...

```sql
/* To understand why, use a different user name and show the User.Id: */
SELECT u.DisplayName, u.Id, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'JamesBrownIsDead'
  ORDER BY c.CreationDate;
GO
```

50 %

Results | Messages | Execution pl

There are a few of 'em.

	DisplayName	Id	Creatio	
1	JamesBrownIsDead	193909	2009	ut Code Contracts should I look at?
2	JamesBrownIsDead	193909	2009-1	debugging do you see a thread's name?
3	JamesBrownIsDead	201949	2009-11-03 18:23:53.430	We're an international company with many users. U...
4	JamesBrownIsDead	193909	2009-11-04 03:39:57.293	How do you 'link to it in the debug configuration of...
5	JamesBrownIsDead	202125	2009-11-06 22:20:02.283	They are both unchecked.
6	JamesBrownIsDead	205456	2009-11-07 06:25:15.970	Updated question to answer this. (More files added...
7	JamesBrownIsDead	205456	2009-11-07 06:41:51.693	Wouldn't it be half a terabyte?
8	JamesBrownIsDead	207393	2009-11-10 19:55:51.813	Thanks for telling me how to find it!
9	JamesBrownIsDead	207393	2009-11-10 22:22:15.627	Yeah, it's not a matter of preference. ArgumentNull...
10	JamesBrownIsDead	207393	2009-11-10 22:40:12.160	Oh snap, nice work. These are two perfect edge c...
11	JamesBrownIsDead	208377	2009-11-11 04:09:29.250	I could, but I don't think anyone knows. I'm specific...
12	JamesBrownIsDead	212267	2009-11-16 18:11:43.793	Yeah, I'm sure there's a better way to do this, I'm ju...
13	JamesBrownIsDead	212457	2009-11-17 18:03:41.037	This is a terrible answer. Look, I'm not trying to use ...

I love this query.

It's a great, simple example of multiple challenges with indexing real-world queries:

- Filters
- Joins
- Ordering

```
SELECT u.DisplayName, u.Id, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
    ON u.Id = c.UserId
  WHERE u.DisplayName = 'JamesBrownIsDead'
  ORDER BY c.CreationDate;
```

It's about experimentation and compromise.

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

And keep things in perspective:

	Logical reads
Clustered indexes, filtering for Brent's comments, order by CreationDate	1,079,417
Add index on Comments.UserId	46,012
Add index on Users.DisplayName	583
Tweak Comments.UserId index to also include CreationDate	584

These are all huge improvements!

Don't get too hung up on the tiniest details.

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

Because people write crazy queries

MIXING JOINS AND FILTERS

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

05 p26

```sql
/* Does it matter where we put filters, the JOIN or the WHERE? */
SELECT u.DisplayName, u.Id, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
              ON u.Id = c.UserId
              AND c.Score > 0
  WHERE u.DisplayName = 'JamesBrownIsDead'
  ORDER BY c.CreationDate;

SELECT u.DisplayName, u.Id, c.CreationDate, c.Text
  FROM dbo.Users u
  INNER JOIN dbo.Comments c
              ON u.Id = c.UserId
  WHERE u.DisplayName = 'JamesBrownIsDead'
    AND c.Score > 0
  ORDER BY c.CreationDate;
```

Filter

Filter

Filter*

Does filter location matter?

Same query, but two different ways to check comment score.

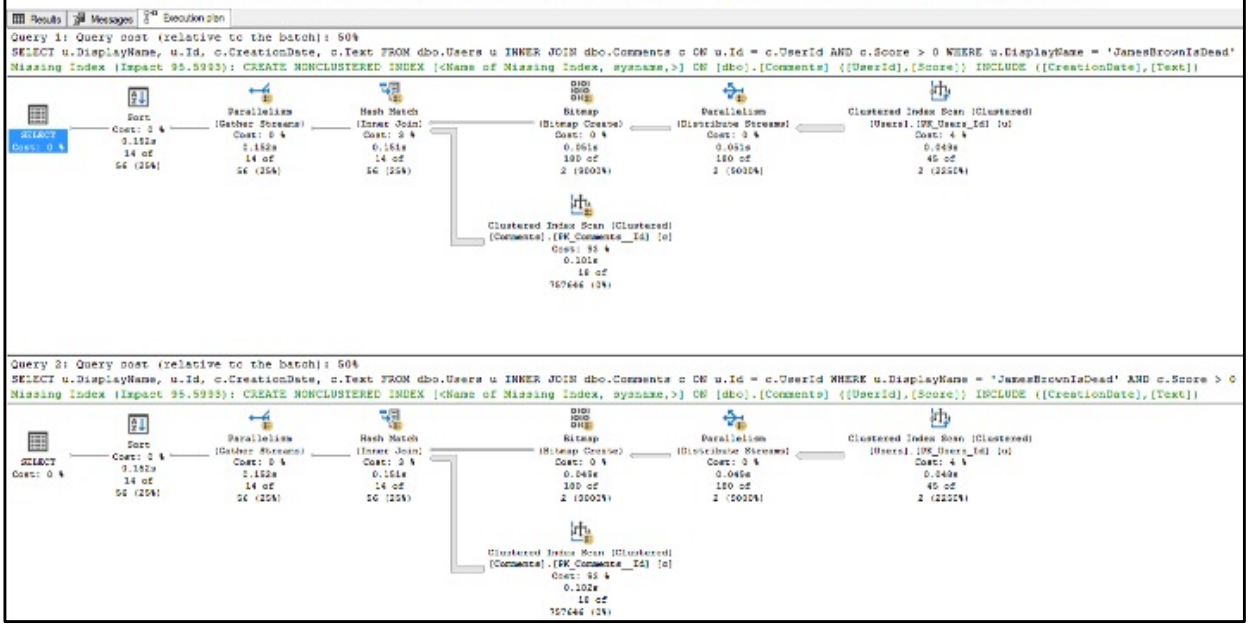
What index should we create?

recording info:

* Not dead, but singing about a dead guy. No, not Kurt Cobain: BrentOzar.com/go/filter

05 p27

In this case, no: same missing index hint, even.



But this opens a can of worms.

In theory, how you write your query shouldn't matter.

In practice, it does:
http://michaeljswart.com/2013/01/joins-are-commutative-and-sql-server-knows-it/

The more complex your query becomes, the harder it is to figure out which operations should be done first.

How the data comes out affects the next operation.

It's a party in the FROM clause and everyone's invited

LOTS OF JOINS

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

05 p30

https://stackoverflow.com/questions/3862045

Say I wanna find and render this comment at the bottom.

I need:

- The question
- The answer
- The comment

Questions & answers are both stored in dbo.Posts.

The query

```
SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text, c.Score
  FROM dbo.Users u
    INNER JOIN dbo.Comments c ON u.Id = c.UserId
    INNER JOIN dbo.Posts Answer ON c.PostId = Answer.Id
    INNER JOIN dbo.Posts Question ON Answer.ParentId = Question.Id
  WHERE u.DisplayName = 'Brent Ozar'
    AND Question.Title LIKE 'SQL Queries%';
```

I'm filtering at both ends of the join:

- Users named Brent Ozar
- Questions titled "SQL Queries"

```
133 DropIndexes;
134 GO
135 SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text, c.Score
136   FROM dbo.Users u
137     INNER JOIN dbo.Comments c ON u.Id = c.UserId
138     INNER JOIN dbo.Posts Answer ON c.PostId = Answer.Id
139     INNER JOIN dbo.Posts Question ON Answer.ParentId = Question.Id
140   WHERE u.DisplayName = 'Brent Ozar'
141     AND Question.Title LIKE 'SQL Queries%';
```

1. SQL Server decided to find the Brents first.

2. Then all his comments...

3. Then which answers he was commenting on...

4. And finally getting what questions they were on.

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

05 p33

Was it smart? Check selectivity.

At first it looks like a close race: they're both selective.

```
144 SELECT COUNT(*) FROM dbo.Users WHERE DisplayName = 'Brent Ozar';
145 SELECT COUNT(*) FROM dbo.Posts WHERE Title LIKE 'SQL Queries%';
146 GO
```

150 %

Results | Messages | Execution plan

	(No column name)
1	1

	(No column name)
1	94

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

05 p34

But check logical reads.

It was WAY easier to find the matching rows in Users.

```
144 SELECT COUNT(*) FROM dbo.Users WHERE DisplayName = 'Brent Ozar';
145 SELECT COUNT(*) FROM dbo.Posts WHERE Title LIKE 'SQL Queries%';
146 GO
```

150 %

Results | Messages | Execution plan

```
(1 row affected)
Table 'Users'. Scan count 5, logical reads 45184, physical reads 0, read-ahead reads 30356, lob

(1 row affected)

(1 row affected)
Table 'Posts'. Scan count 5, logical reads 4190270, physical reads 3, read-ahead reads 4080925,

(1 row affected)
```

SQL Server considers...

How big is the object we need to read?
(Think number of 8KB pages, not rows or columns)

How selective are the query filters on this object?

When we read data out of this object, what order will it be in? Does that help the next operation?

(And much, much more.)

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

05 p38

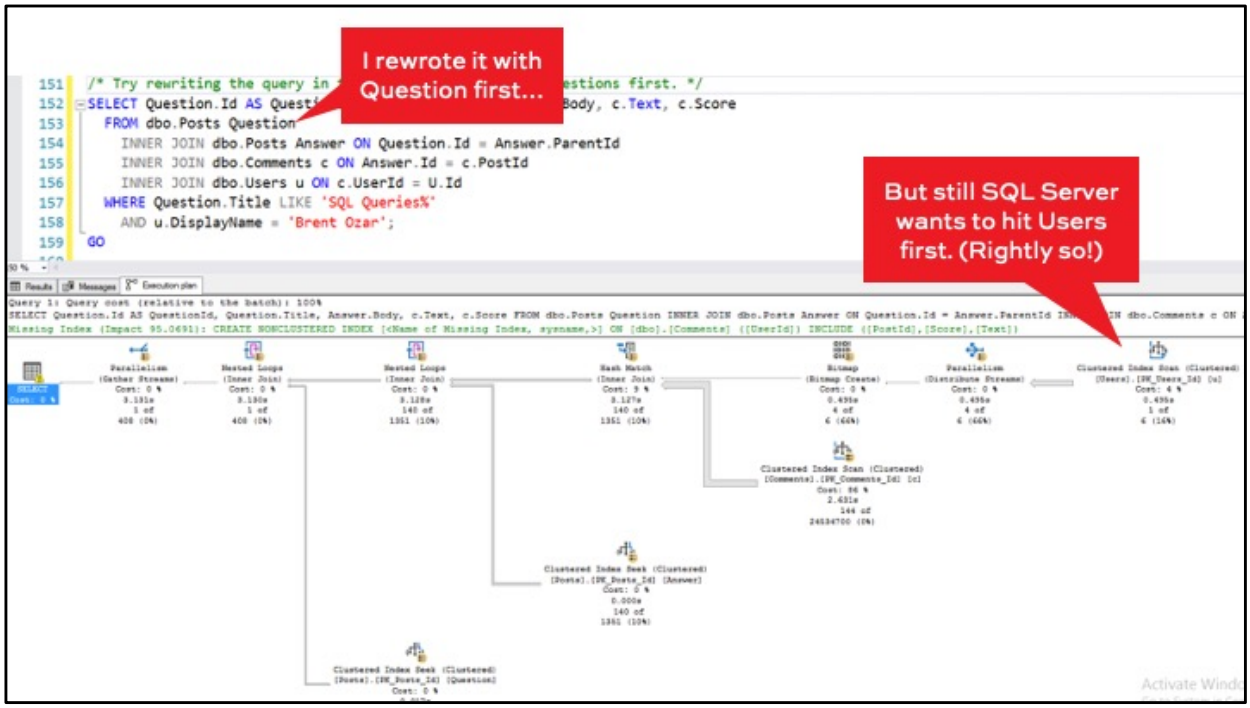
```
133 DropIndexes;
134 GO
135 SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text
136   FROM dbo.Users u
137     INNER JOIN dbo.Comments c ON u.Id = c.UserId
138     INNER JOIN dbo.Posts Answer ON c.PostId = Answer.Id
139     INNER JOIN dbo.Posts Question ON Answer.ParentId = Question.Id
140   WHERE u.DisplayName = 'Brent Ozar'
141     AND Question.Title LIKE 'SQL Queries%';
```

Right now, SQL Server is picking Users, the tiny object, to read first.

Is the query's design causing that?

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

05 p39



Let's throw him a curveball.

Create an index on Posts.Title to make that part of the filtering easier:

```
  WHERE Question.Title LIKE 'SQL Queries%'
    AND u.DisplayName = 'Brent Ozar';
GO

CREATE INDEX IX_Title ON dbo.Posts(Title);
GO
```

Then try the query again...

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

05 p41

```sql
/* Does that change which table gets processed first? */
SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text, c.Score
  FROM dbo.Posts Question
    INNER JOIN dbo.Posts Answer ON Question.Id = Answer.ParentId
    INNER JOIN dbo.Comments c ON Answer.Id = c.PostId
    INNER JOIN dbo.Users u ON c.UserId = U.Id
  WHERE Question.Title LIKE 'SQL Queries%'
    AND u.DisplayName = 'Brent Ozar';
GO
```

A-ha! First find the matching titles...

Then...wait... hang on...

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

05 p42

What you thought would happen

1. Find Questions where Title like '%SQL Queries%'
2. Find the Answers on those questions
3. Find the Comments on those answers
4. Look up the Users for each of those comments, and check to see if they're Brent Ozar

But that's not what happened.

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

What actually happened

1. Find Questions where Title like 'SQL Queries%'

Meanwhile, AT THE SAME TIME:

1. Find the users named Brent Ozar
2. Find the Comments they've left
3. Look up what Answers they were placed on
4. Then finally, join this to the SQL Query questions

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

05 p44

```sql
/* Does that change which table gets processed first? */
SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text, c.Score
  FROM dbo.Posts Question
    INNER JOIN dbo.Posts Answer ON Question.Id = Answer.ParentId
    INNER JOIN dbo.Comments c ON Answer.Id = c.PostId
    INNER JOIN dbo.Users u ON c.UserId = U.Id
  WHERE Question.Title LIKE 'SQL Queries%'
    AND u.DisplayName = 'Brent Ozar';
GO
```

If you ran into this query in production, what would you notice?

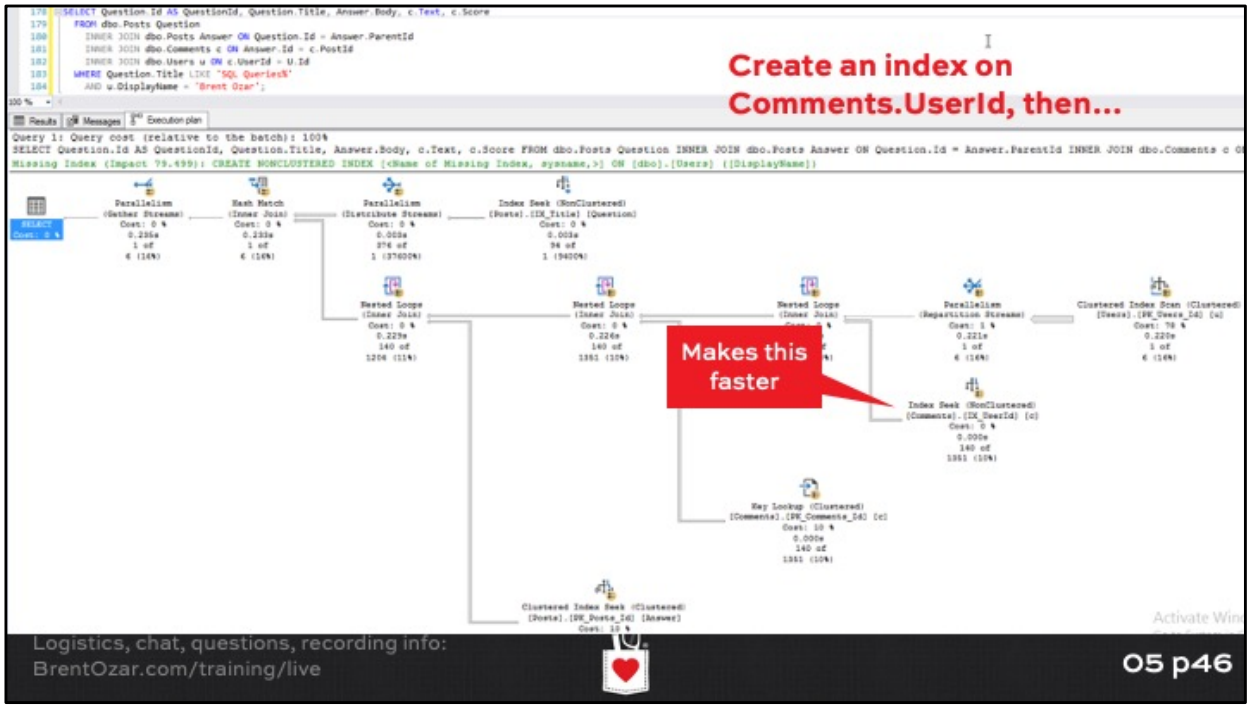
Results | Messages | Execution plan

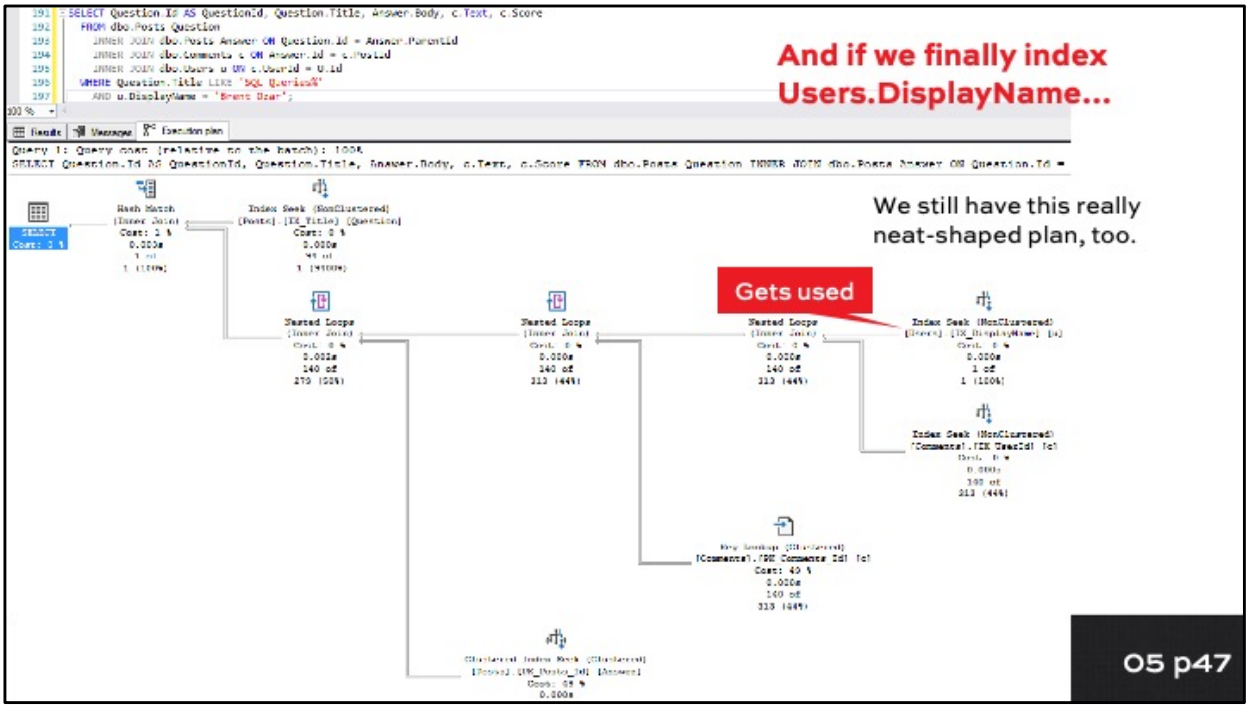
```
Query 1: Query cost (relative to the batch): 100%
SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text, c.Score FROM dbo.Posts Question INNER JOIN dbo.Posts Answer ON Question.Id = Answer.ParentId INNER JOIN dbo.Comments
Missing Index (Impact 95.3851): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Comments] ([UserId]) INCLUDE ([PostId],[Score],[Text])
```

HEY MAN CAN I GET SOME HELP HERE

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

05 p45

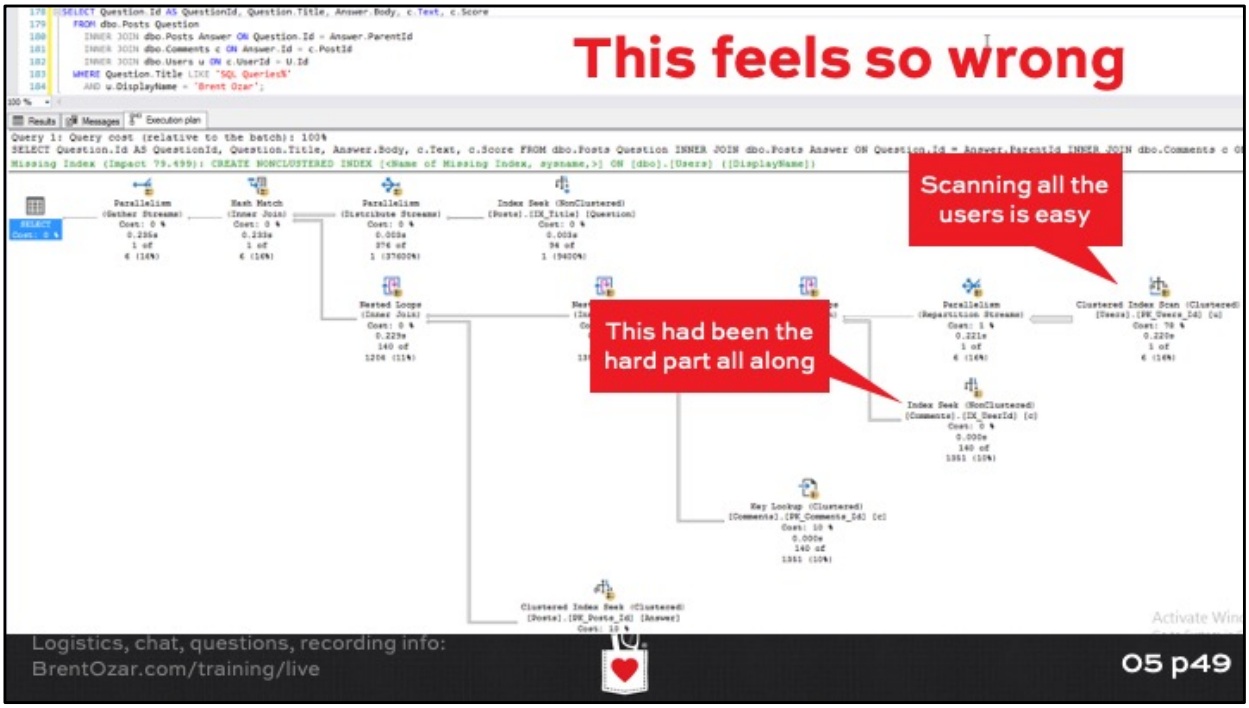




Looking at the big picture

	Logical Reads
Clustered index scans	1,077,747
Add index on Posts.Title	1,077,063
That, PLUS add index on Comments.UserId	46,600
That, PLUS add index on Users.DisplayName	1,165
Or what if we start over, and only add an index on Comments.UserId?	47,373

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



It's hard to tell that from the query

```
SELECT Question.Id AS QuestionId, Question.Title, Answer.Body, c.Text, c.Score
  FROM dbo.Posts Question
    INNER JOIN dbo.Posts Answer ON Question.Id = Answer.ParentId
    INNER JOIN dbo.Comments c ON Answer.Id = c.PostId
    INNER JOIN dbo.Users u ON c.UserId = U.Id
  WHERE Question.Title LIKE 'SQL Queries%'
    AND u.DisplayName = 'Brent Ozar';
```

When you look at that query, your first instinct is probably to index the stuff in the WHERE clause.

And that's totally okay. That helps.
But indexing to support joins is super important too.

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

05 p50

“Index your foreign keys”

That’s where this advice comes from.

It’s not just about making it easier for SQL Server to enforce foreign key relationships (which helps too.)

It’s also because you often join on these keys.

It’s a good starting point when you have no idea what indexes to build on a table.

It’s just not the finish line.

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

05 p51

Suddenly I feel all existential

WHERE EXISTS

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

05 p52

Exists is kinda like a join, too

```
SELECT *
  FROM dbo.Users u
  WHERE u.Location = 'Antarctica'
    AND EXISTS (SELECT 1/0 FROM dbo.Comments c WHERE u.Id = c.UserId)
```

What indexes do I need on these tables?

The suggested index isn't Location.

```
210 DropIndexes;
211 GO
212
213 SELECT *
214   FROM dbo.Users u
215   WHERE u.Location = 'Antarctica'
216   AND EXISTS (SELECT * FROM dbo.Comments c WHERE u.Id = c.UserId);
217 GO
```

It's on Comments – just like the join index.

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users u WHERE u.Location = 'Antarctica' AND EXISTS (SELECT * FROM dbo.Comments c WHERE u.Id = c.UserId)
Missing Index (Impact 95.8199): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Comments] ([UserId])
```

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

05 p54

SQL Server's thought process

"It's easy for me to scan the small Users table and find all the few people in Antarctica."

"However, once I've found their list of User Ids, it's gonna be painful for me to scan the giant Comments table to find their comments."

"The most efficient index would be on Comments.UserId."

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

05 p55

Give it to him, and he uses it...

```
CREATE INDEX IX_UserId ON dbo.Comments(UserId);
GO
SELECT *
  FROM dbo.Users u
  WHERE u.Location = 'Antarctica'
  AND EXISTS (SELECT * FROM dbo.Comments c WHERE u.Id = c.UserId);
GO
```

But now he has another idea.

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

05 p56

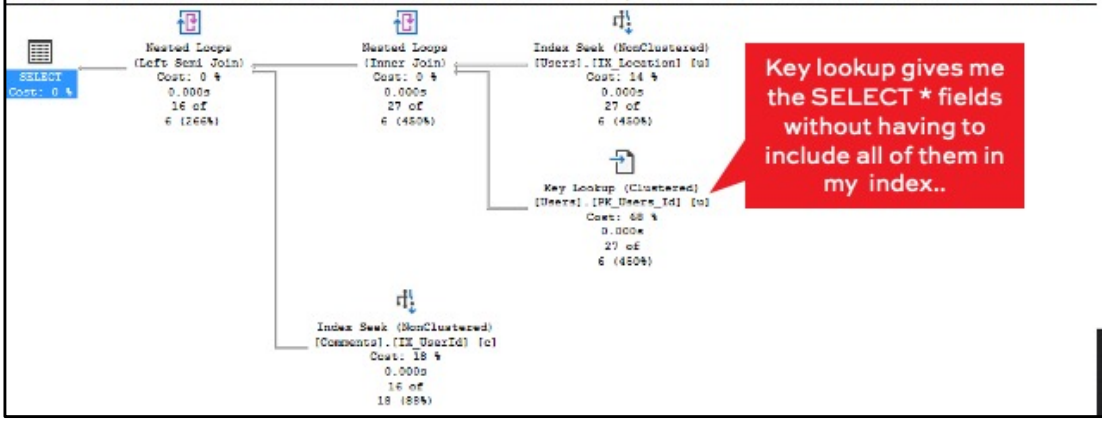
```
227  CREATE INDEX IX_Location ON dbo.Users(Location);
228  GO
229  SELECT *
230    FROM dbo.Users u
231    WHERE u.Location = 'Antarctica'
232    AND EXISTS (SELECT * FROM dbo.Comments c WHERE u.Id = c.UserId);
233  GO
```

Here's a better idea.

50 %

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users u WHERE u.Location = 'Antarctica' AND EXISTS (SELECT * FROM dbo.Comments c WHERE u.Id = c.UserId)



Key lookup gives me the SELECT * fields without having to include all of them in my index..

05 p57

Joins are interesting.

Joins are like filters:
only show me the rows from Table1
that have a matching partner in Table2.

Their selectivity isn't just about row count: also size.

Join operations can benefit from pre-sorting:
if I want to join two tables together,
it can help if they're already sorted in order.

Join-supporting indexes radically change plan shape.

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

BRENT OZAR
UNLIMITED®

Fundamentals of Index Tuning

Part 6: time to make a match and do some JOINs.

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

05 p60

Working through the lab

Read the first query, execute it, do your work inline, creating and dropping indexes where directed

45 minutes: you work through the rest, asking questions in Slack as you go, and take a bio break

30 minutes: I work through it onscreen