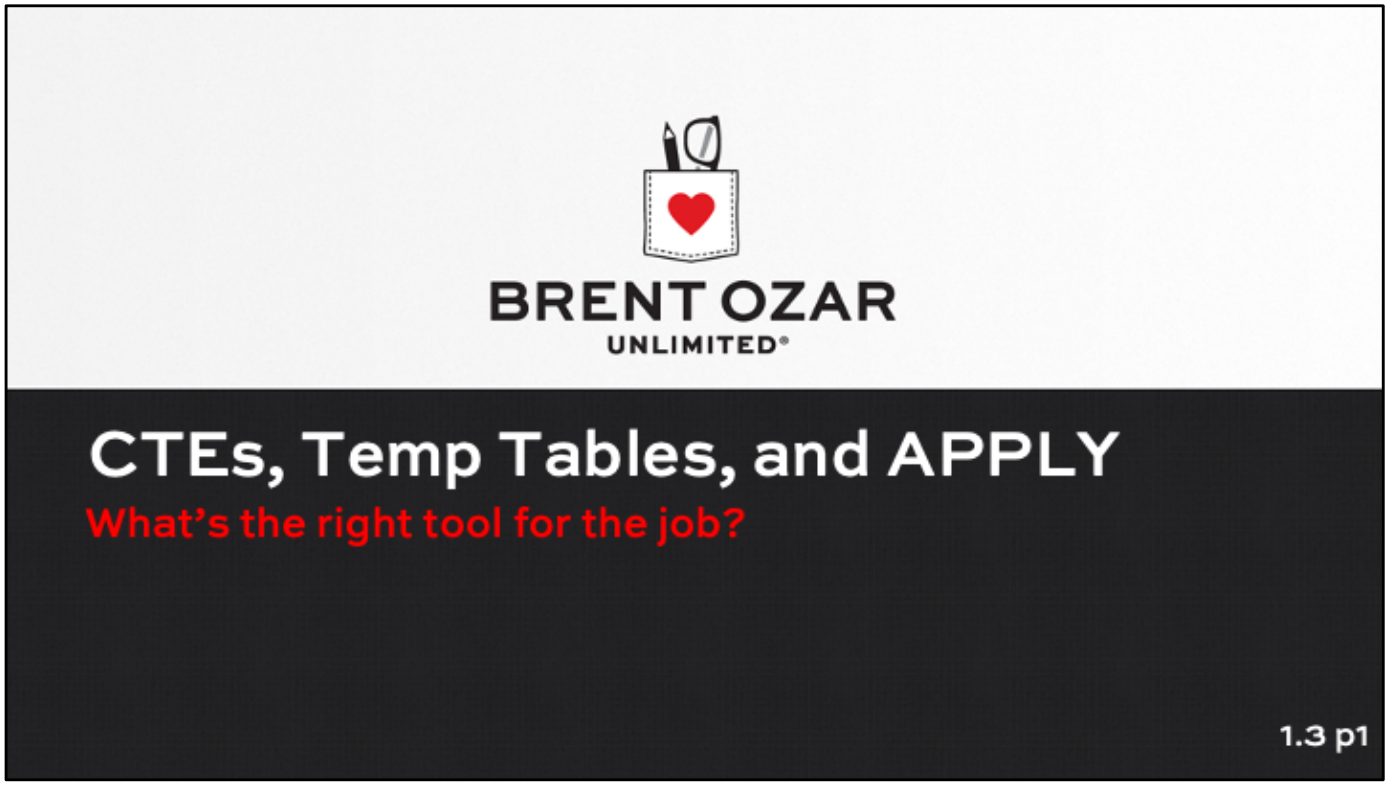
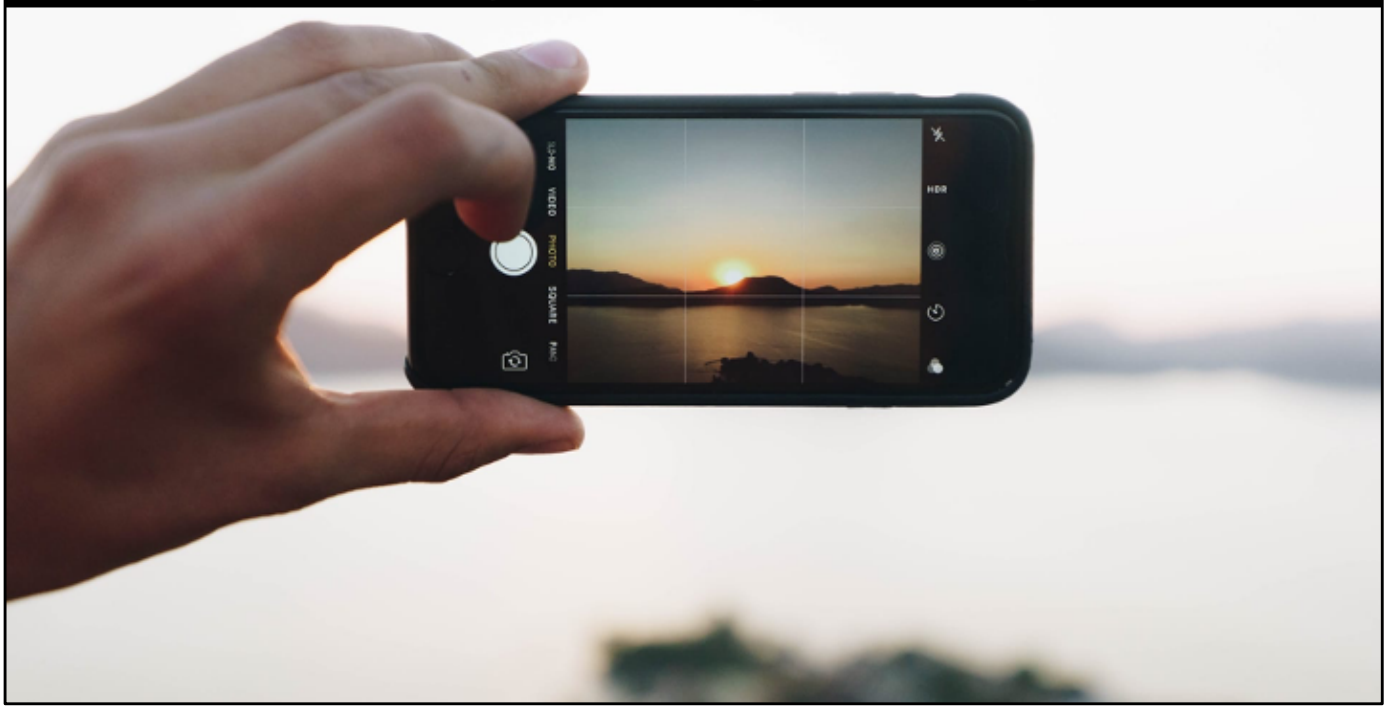


You take pictures.

Sometimes you just need good enough, quickly.



Use cases we'll explore

1. "I have a few different queries that I want to merge into one."
2. "I have a big query where the estimates go wrong halfway in, so I need to break it up into stages."
3. "I want to run a subquery for every row, getting the top N rows, but I want it to go fast."

"I have a big query with a lot of joins, and the estimates go wrong."

Table variables and temp tables

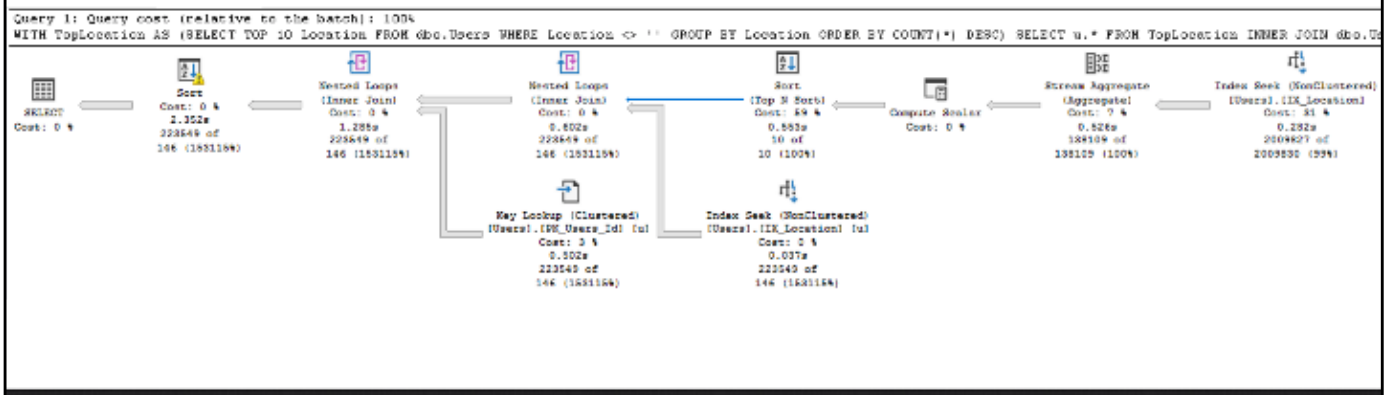
Users want us to tune this...

And we already added an index...

```
CREATE INDEX IX_Location ON dbo.Users (Location);
GO
CREATE OR ALTER PROC dbo.usp_UsersInTop10Locations AS
BEGIN
WITH TopLocation AS (SELECT TOP 10 Location
  FROM dbo.Users
  WHERE Location <> ''
  GROUP BY Location
  ORDER BY COUNT(*) DESC)
SELECT u.*
  FROM TopLocation
    INNER JOIN dbo.Users u ON TopLocation.Location = u.Location
  ORDER BY DisplayName;
END
```

The query plan: look familiar?

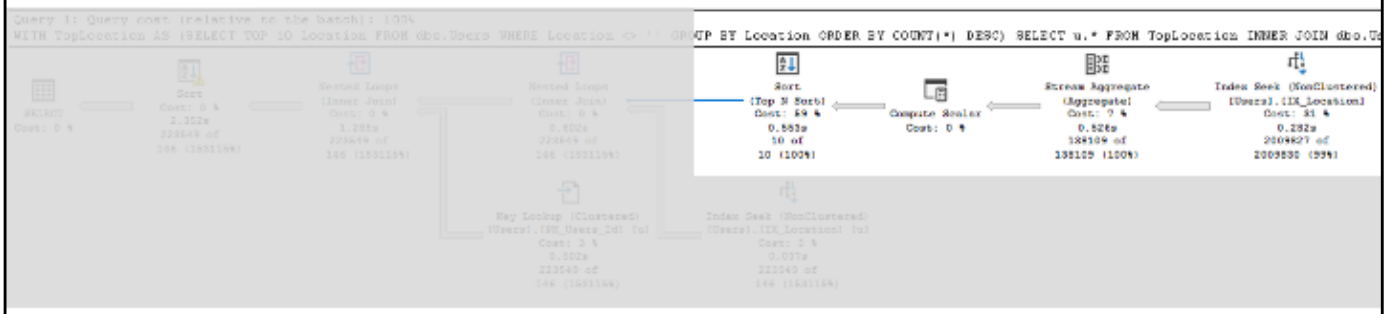
It's like our recent query where we got the top 1 location, but it's 10.



1.3 p8

In here, the estimates are right

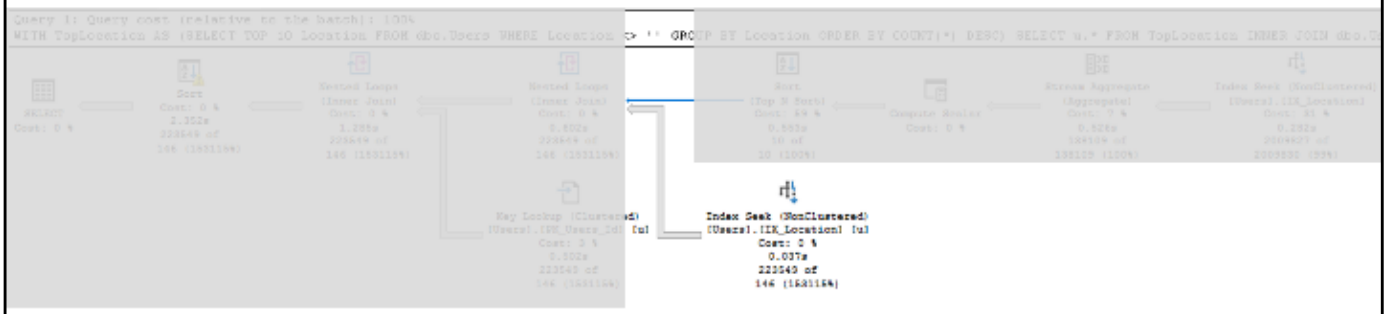
SQL Server groups the locations together to find the top 10, and it knows that there will be 10 values...just not which locations they'll be.



1.3 p9

Here, the estimates go wrong

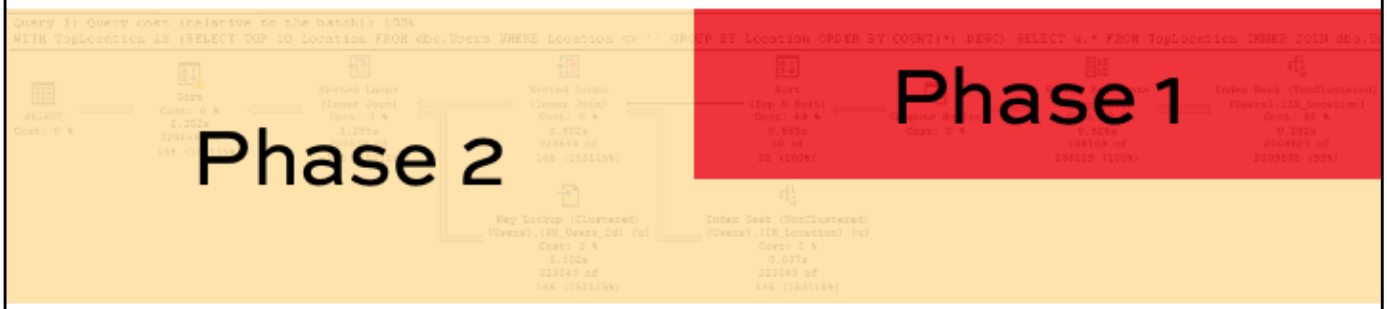
Because SQL Server underestimates how many rows will come back for each location – it's using the average number of rows per average location.



1.3 p10

We need to separate these parts

Because SQL Server underestimates how many rows will come back for each location – it's using the average number of rows per average location.



1.3 p11

Before...

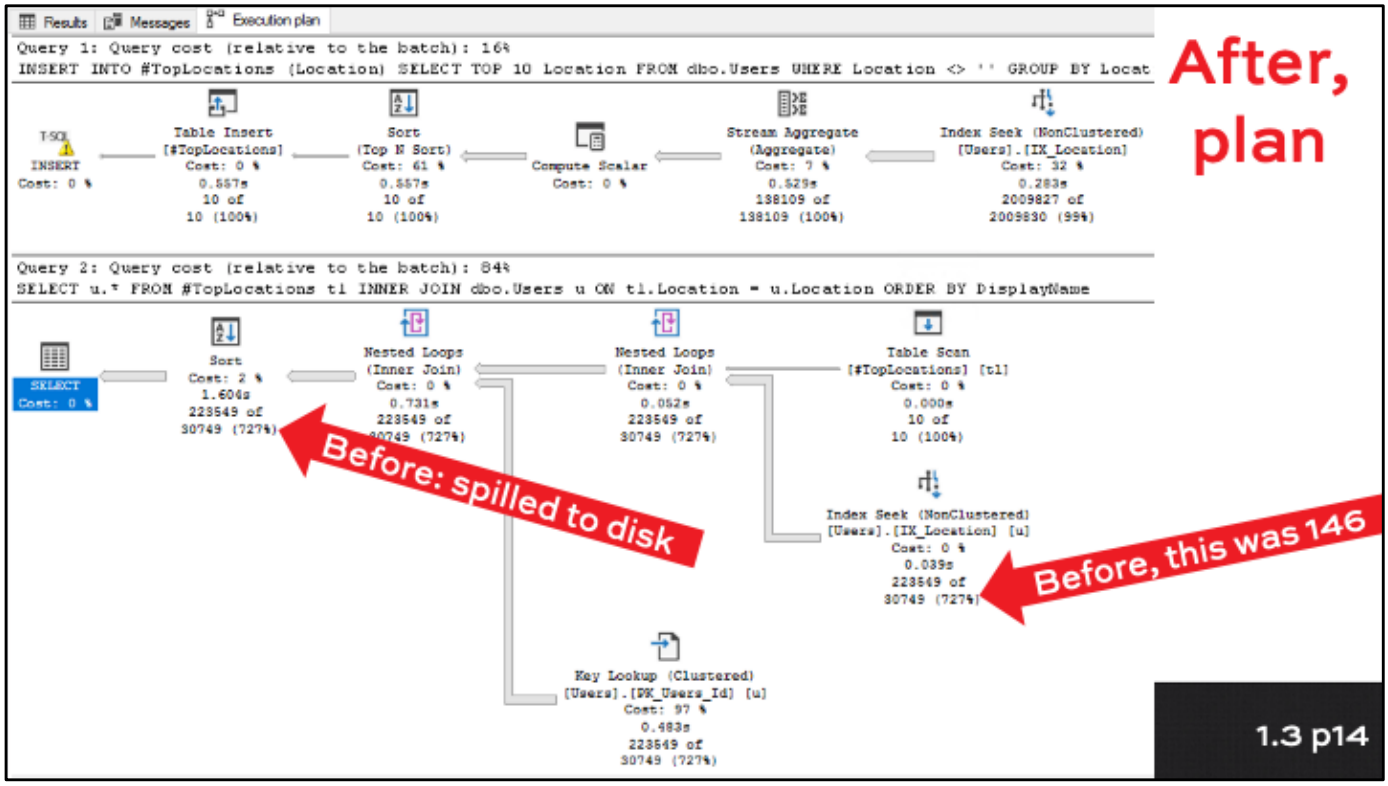
```
CREATE OR ALTER PROC dbo.usp_UsersInTop10Locations AS
BEGIN
WITH TopLocation AS (SELECT TOP 10 Location
  FROM dbo.Users
  WHERE Location <> ''
  GROUP BY Location
  ORDER BY COUNT(*) DESC)
SELECT u.*
  FROM TopLocation
    INNER JOIN dbo.Users u ON TopLocation.Location = u.Location
  ORDER BY DisplayName;
END
```

1.3 p12

After, with temp tables:

```
CREATE OR ALTER PROC dbo.usp_UsersInTop10Locations AS
BEGIN
/* Phase 1 */
CREATE TABLE #TopLocations (Location NVARCHAR(100));
INSERT INTO #TopLocations (Location)
SELECT TOP 10 Location
  FROM dbo.Users
  WHERE Location <> ''
  GROUP BY Location
  ORDER BY COUNT(*) DESC;

/* Phase 2 */
SELECT u.*
  FROM #TopLocations tl
    INNER JOIN dbo.Users u ON tl.Location = u.Location
  ORDER BY DisplayName;
END
GO
```

1.3 p13



To see how, leave the temp table

```
/* Seeing the statistics */
/* Phase 1 */
CREATE TABLE #TopLocations (Location NVARCHAR(100));
INSERT INTO #TopLocations (Location)
SELECT TOP 10 Location
  FROM dbo.Users
  WHERE Location <> ''
  GROUP BY Location
  ORDER BY COUNT(*) DESC;

/* Phase 2 */
SELECT u.*
  FROM #TopLocations tl
    INNER JOIN dbo.Users u ON tl.Location = u.Location
  ORDER BY DisplayName;
GO
```

1.3 p15

Get the temp table's name for sp_BlitzIndex

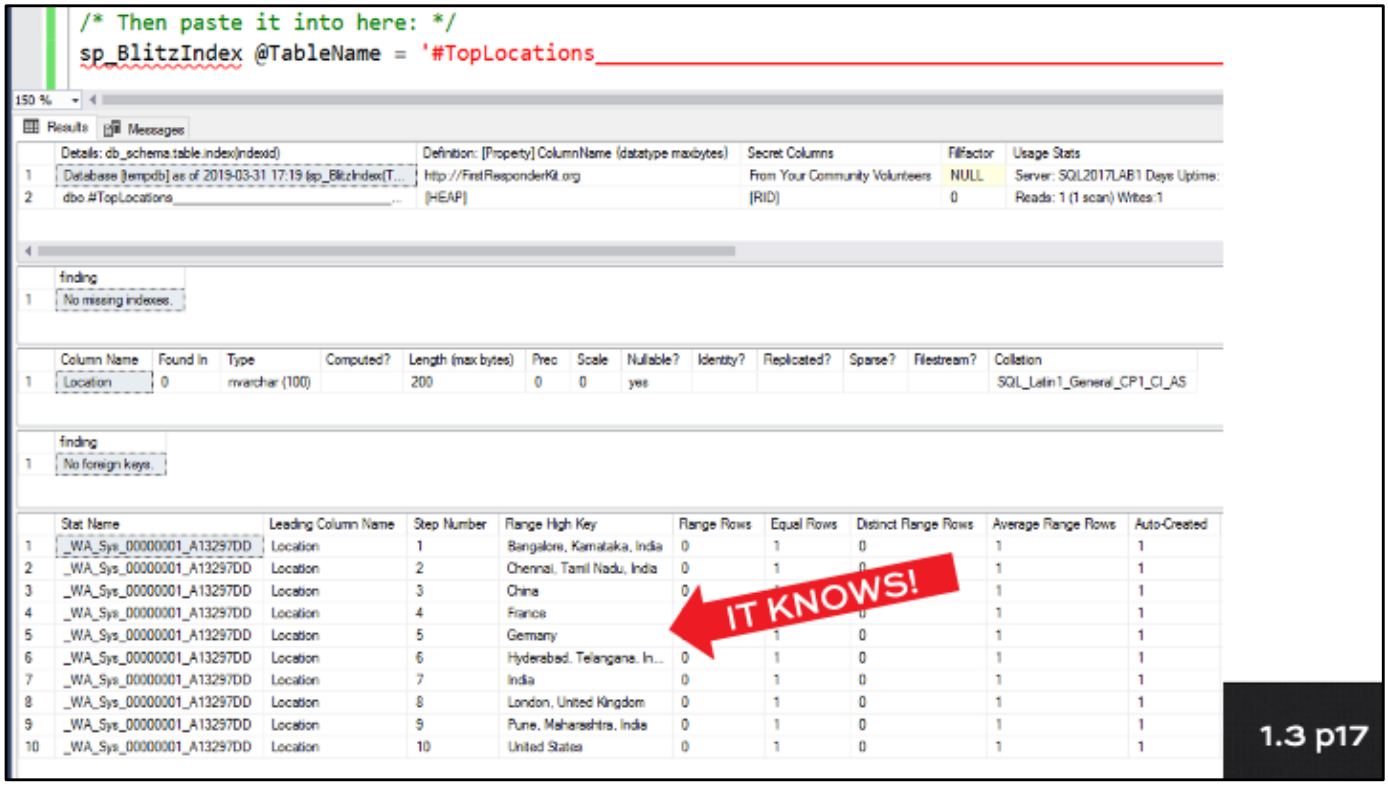
```
USE tempdb;
GO
/* Get the temp table's full name: */
SELECT * FROM sys.all_objects WHERE name LIKE '%TopLocations%'

/* Then paste it into here: */
sp_BlitzIndex @TableName = '#TopLocations
```

150 %

Results | Messages

	name	object_id
1	#TopLocations_____000000000C93	-1590519843

1.3 p16



#tables have auto-created stats

Just like regular tables (although they're harder to see, since they drop)

Auto-updated just like regular tables after data is loaded

Just like real table stats, they can go bad

They can get REALLY strange due to plan reuse and statistics reuse:
https://sqlperformance.com/2017/05/sql-performance/sql-server-temporary-object-caching

1.3 p18

When to use temp tables

If you need to refer to a result set multiple times in multiple queries

When you have a big query, and estimates go wrong partway in

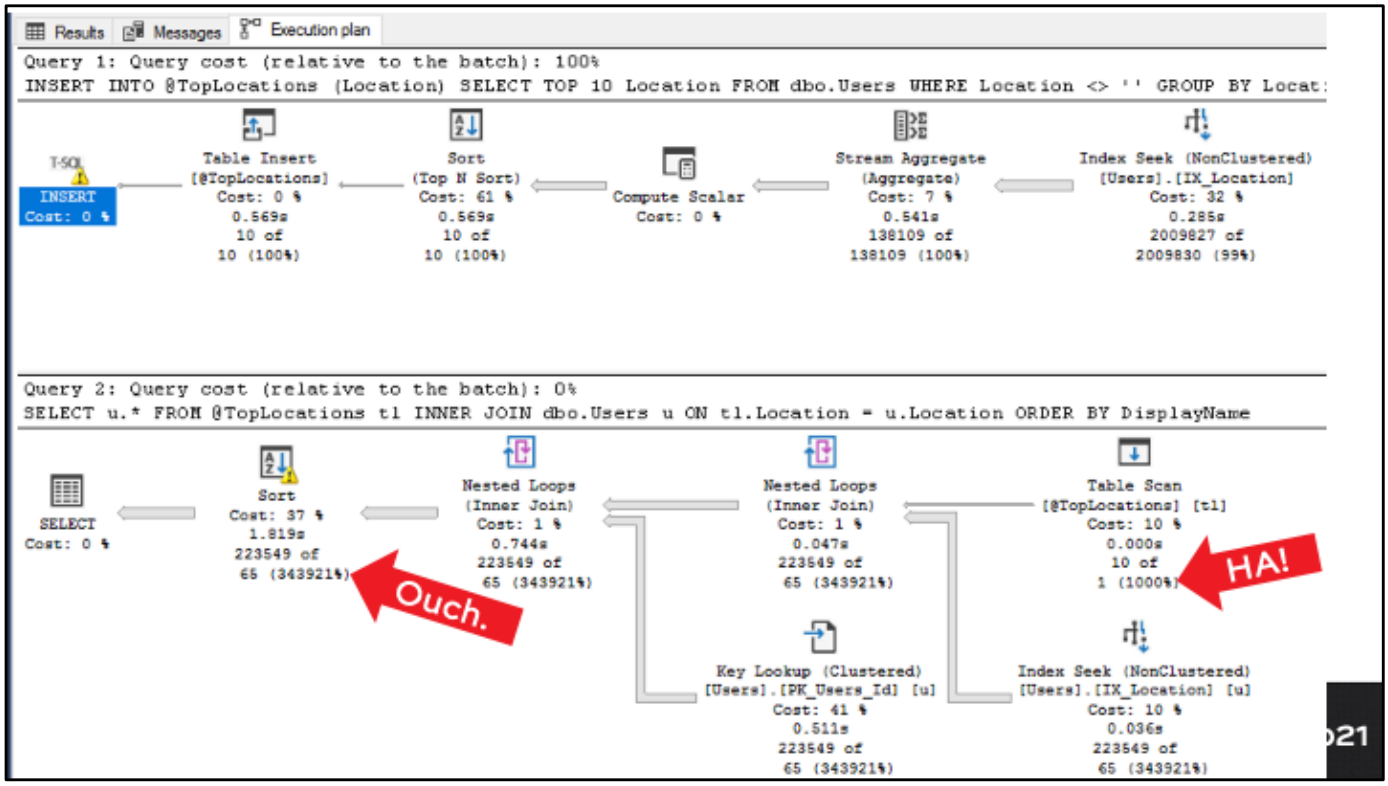
If knowing what rows come out of Phase 1 will dramatically change Phase 2's query plan

1.3 p19

What about table variables?

```
CREATE OR ALTER PROC dbo.usp_UsersInTop10Locations AS
BEGIN
/* Phase 1 */
DECLARE @TopLocations TABLE (Location NVARCHAR(100));
INSERT INTO @TopLocations (Location)
SELECT TOP 10 Location
  FROM dbo.Users
  WHERE Location <> ''
  GROUP BY Location
  ORDER BY COUNT(*) DESC;

/* Phase 2 */
SELECT u.*
  FROM @TopLocations tl
    INNER JOIN dbo.Users u ON tl.Location = u.Location
  ORDER BY DisplayName;
END
GO
```

1.3 p20



When to use table variables

Never because they don't have stats.

When to use table variables

~~Never because they don't have stats.~~

- When you purposely want an incorrect 1-row estimate.
- When you want an object that ignores transactions.
- When you want to reduce recompilations due to changing data.

SQL Server 2019:

- Gains deferred compilation for table variables
- But modifying table variable contents are still serial

More info in my Fundamentals of TempDB class.

I'm going to use a dumb example.

You would never write a query this terribad:

1. Total up all the Votes by UserId
2. Show the top voters in a specific location

But bear with me. It'll tell the story.

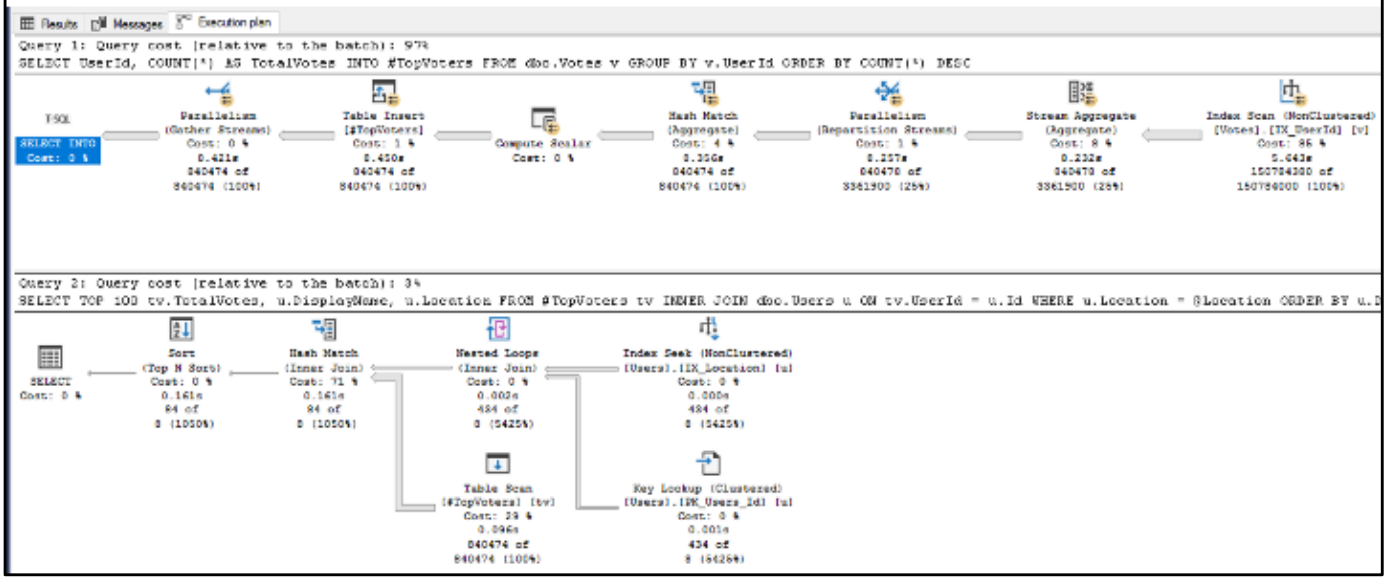
```sql
CREATE OR ALTER PROC dbo.usp_TopVotersInCity @Location NVARCHAR(40) AS
BEGIN
/* Find the users who have left the most votes */
SELECT UserId, COUNT(*) AS TotalVotes
  INTO #TopVoters
  FROM dbo.Votes v
  GROUP BY v.UserId
  ORDER BY COUNT(*) DESC;

SELECT TOP 100 tv.TotalVotes, u.DisplayName, u.Location
  FROM #TopVoters tv
  INNER JOIN dbo.Users u ON tv.UserId = u.Id
  WHERE u.Location = @Location
  ORDER BY u.DisplayName;
END
GO
```

Scan all the votes

Do some filtering

1.3 p26

Runs, but takes 8-10 seconds



Common Table Expression (CTE)

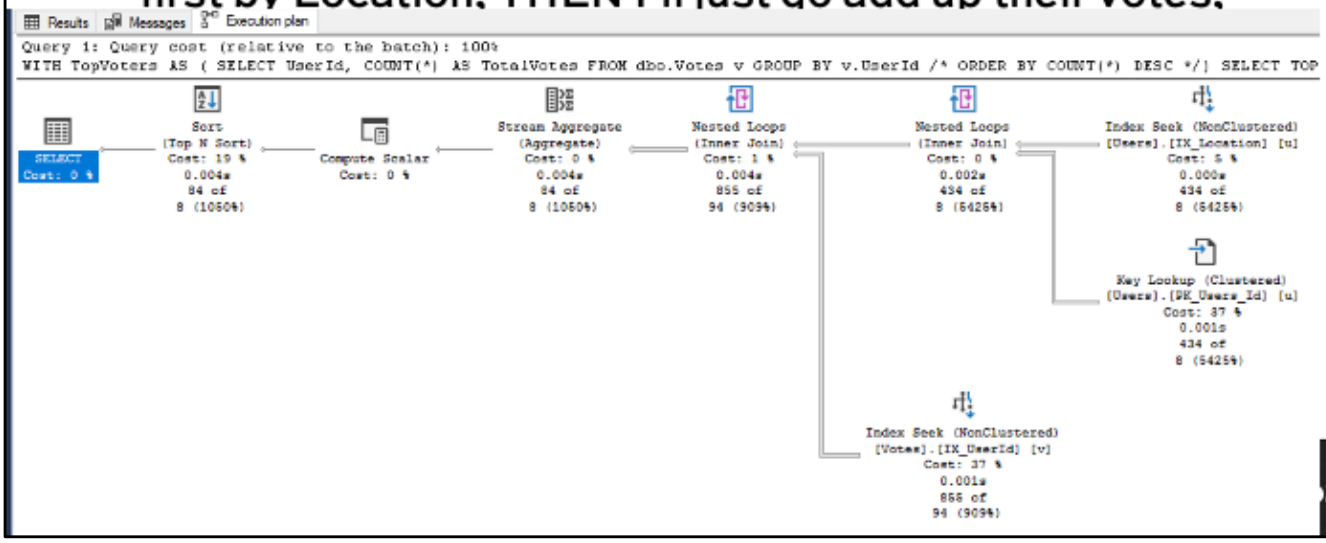
WITH Alias AS: lets SQL Server choose how to process this part

```
CREATE OR ALTER PROC dbo.usp_TopVotersInCity_CTE @Location NVARCHAR(40) AS
BEGIN
WITH TopVoters AS (
    SELECT UserId, COUNT(*) AS TotalVotes
      FROM dbo.Votes v
      GROUP BY v.UserId
      /* ORDER BY COUNT(*) DESC */)
SELECT TOP 100 tv.TotalVotes, u.DisplayName, u.Location
  FROM TopVoters tv
  INNER JOIN dbo.Users u ON tv.UserId = u.Id
  WHERE u.Location = @Location
  ORDER BY u.DisplayName;
END
```

1.3 p28

Runs instantly

Because SQL Server says, lemme find the right Users first by Location. THEN I'll just go add up their votes.



CTEs are great when...

You have multi-step queries

You want to break them up to be easier to read
(easier than subqueries in some cases)

You're not sure which parts of the processing should be done first

When the row estimates from each phase may not matter
(or when SQL Server can do a pretty good job at estimating them)

You don't need to refer to the results over and over in multiple steps
(because they disappear – they're not materialized)

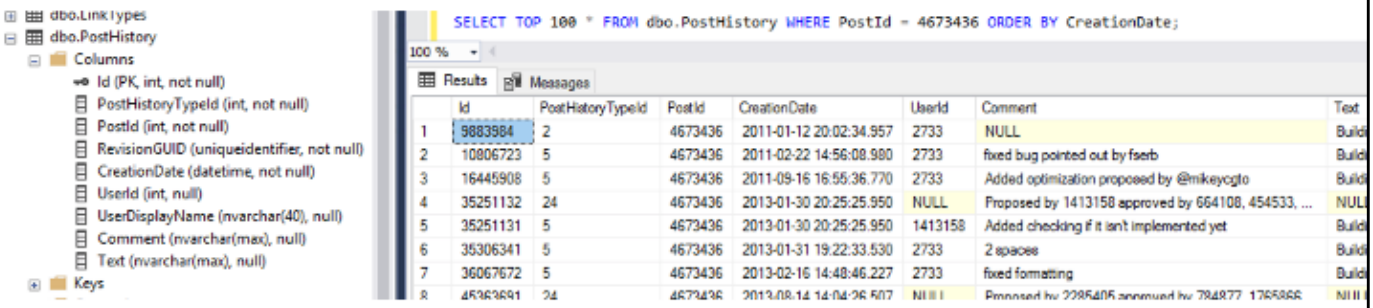
Advanced use case: recursive hierarchies

1.3 p30

Stack Overflow example

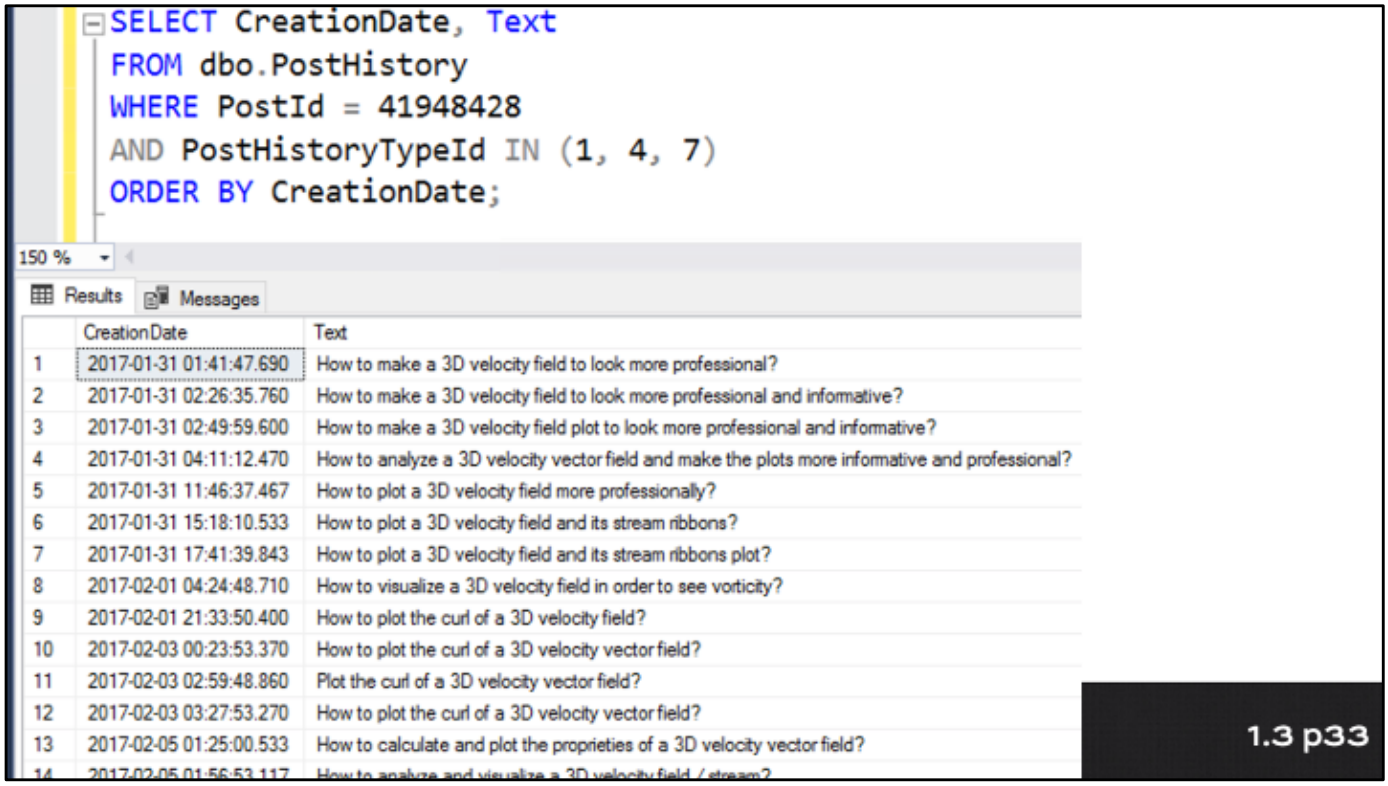
Questions & answers (Posts) can be edited

History stored in the PostHistory table, so you can see edits over time:



PostHistoryTypeId: title changed, post closed, protected, tweeted, etc.

1.3 p32



Our query requirements, v1

For a user, show their top 10 questions by score.

```
/* Go get a user's top 10 questions by score */

SELECT TOP 10 q.Id AS QuestionId, q.Title AS CurrentTitle, q.Score
  FROM dbo.Posts q
  WHERE q.OwnerUserId = 1031417
    AND q.PostTypeId = 1 /* Question */
  ORDER BY q.Score DESC;
```

150 %

Results | Messages

	QuestionId	CurrentTitle	Score
1	9959727	__proto__ VS. prototype in JavaScript	584
2	8450472	How to format print output into fixed width?	65
3	13786499	What is the difference between 'sh' and 'source'?	40
4	15065010	How do I use a for-each loop to iterate over file path...	39
5	14751973	What is "!!" in C?	37
6	13468286	How to read, understand, analyze and debug a Linu...	35
7	8393424	How to analyze twitters messages? (improving my sl...	21
8	15779185	List all the functions/symbols on the fly in C?	17
9	15553493	How to print only the hex values from hexdump with...	16
10	10089273	Why does this code enable me to detect a debugger?	15

1.3 p34

Our query requirements, v2

For a user, show their top 10 questions by score...

And add rows for the first 5 edited titles (PostHistoryTypeId = 4)

Our original query:

```
/* Go get a user's top 10 questions by score */

SELECT TOP 10 q.Id AS QuestionId, q.Title AS CurrentTitle, q.Score
  FROM dbo.Posts q
  WHERE q.OwnerUserId = 1031417
    AND q.PostTypeId = 1 /* Question */
  ORDER BY q.Score DESC;
```

But now we need to add joins to PostHistory,
but we only want to show the first 5 rows.

1.3 p35

Enter APPLY.

```
/* Add rows for the first 5 titles per post... */

SELECT q.Id AS QuestionId, q.Title AS CurrentTitle, q.Score,
    qTitles.CreationDate AS TitleEditDate, qTitles.Text AS TitleText
  FROM dbo.Posts q

  CROSS APPLY (SELECT TOP 5 ph.CreationDate, ph.Text
               FROM dbo.PostHistory ph
               WHERE ph.PostId = q.Id
                   AND ph.PostHistoryTypeId = 4
               ORDER BY ph.CreationDate) qTitles

  WHERE q.OwnerUserId = 1031417
    AND q.PostTypeId = 1 /* Question */
  ORDER BY q.Score DESC, q.Id, qTitles.CreationDate;
```

1.3 p36

Like subquery or join, but different.

```
CROSS APPLY (SELECT TOP 5 ph.CreationDate, ph.Text
             FROM dbo.PostHistory ph
             WHERE ph.PostId = q.Id
                 AND ph.PostHistoryTypeId = 4
             ORDER BY ph.CreationDate) qTitles
```

Subqueries can't return multiple fields – this can.

Joins require you to define relationships and take either one, or all of the rows – this lets you pick and choose.

Subqueries can struggle with correlation performance – this doesn't.*

* If you don't have indexes to support this, though, it'll be slow.

1.3 p37

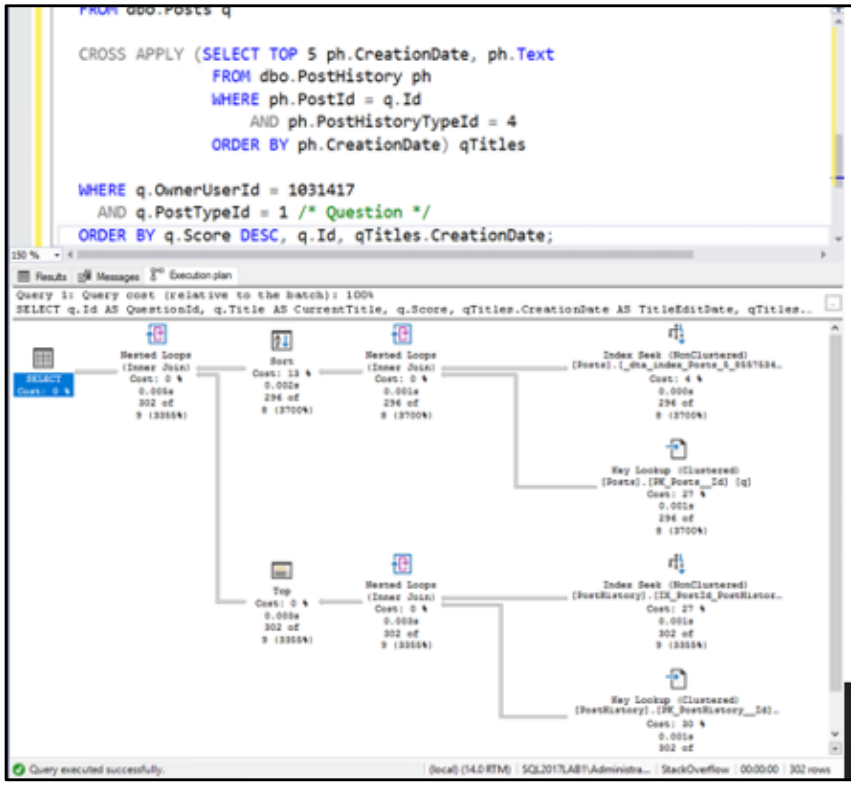
```sql
/* Add rows for the first 5 titles per post... */

SELECT q.Id AS QuestionId, q.Title AS CurrentTitle, q.Score,
    qTitles.CreationDate AS TitleEditDate, qTitles.Text AS TitleText
  FROM dbo.Posts q

  CROSS APPLY (SELECT TOP 5 ph.CreationDate, ph.Text
                 FROM dbo.PostHistory ph
                 WHERE ph.PostId = q.Id
                     AND ph.PostHistoryTypeId = 4
                 ORDER BY ph.CreationDate) qTitles

  WHERE q.OwnerUserId = 1031417
    AND q.PostTypeId = 1 /* Question */
  ORDER BY q.Score DESC, q.Id, qTitles.CreationDate;
```

150 %

Results | Messages

	QuestionId	CurrentTitle	Score	TitleEditDate	TitleText
1	9659727	__proto__ VS. prototype in JavaScript	584	2012-04-28 17:24:38.603	What is the difference between __proto__ and prototype?
2	9659727	__proto__ VS. prototype in JavaScript	584	2012-07-30 00:21:32.177	Java Script - What is the difference between __proto__ and prototype?
3	9659727	__proto__ VS. prototype in JavaScript	584	2013-04-10 16:11:05.233	What is the difference between __proto__ and prototype?
4	9659727	__proto__ VS. prototype in JavaScript	584	2013-04-10 16:29:24.743	What is the difference between __proto__ and prototype in Java script?
5	9659727	__proto__ VS. prototype in JavaScript	584	2013-06-13 13:08:50.883	What is the difference between __proto__ and prototype in JavaScript?
6	8450472	How to format print output into fixed width?	65	2012-10-30 21:23:01.470	How to format print to fixed width of strings in python?
7	8450472	How to format print output into fixed width?	65	2012-11-30 07:20:31.887	python - How to format the print call into a fixed width output?
8	8450472	How to format print output into fixed width?	65	2013-04-19 12:51:58.850	python - How to format print output into a fixed width output?
9	8450472	How to format print output into fixed width?	65	2013-04-20 21:36:29.567	python - How to format print output into fixed width?
10	8450472	How to format print output into fixed width?	65	2013-05-04 16:09:40.837	python 2.7 - How to format print output into fixed width?
11	13786499	What is the difference between 'sh' and 'source'?	40	2015-04-09 15:03:15.427	What is the difference in Linux between 'source' and 'sh'?
12	13786499	What is the difference between 'sh' and 'source'?	40	2015-12-25 23:37:30.307	What is the difference in a Linux shell between 'source' and 'sh'?

1.3 p38



Indexed, so it's fast.

PostHistory has an index on:

- PostId
- PostHistoryTypeId
- CreationDate

(But no includes)

1.3 p39

This beats a subquery when you need >1 col

```
SELECT u.DisplayName,
       (
         SELECT TOP 3 *
         FROM dbo.Posts AS p
         WHERE p.OwnerUserId = u.Id
        ) AS TopThreePosts
FROM   dbo.Users AS u;
```

131 %

Messages

```
Msg 116, Level 16, State 1, Line 10
Only one expression can be specified in the select list
when the subquery is not introduced with EXISTS.
```

1.3 p40

It's not all magic and fairies

When writing queries using Apply, this stuff generally works better

- The base table should be the smaller table
- If possible, use a Primary Key or other Unique index to correlate

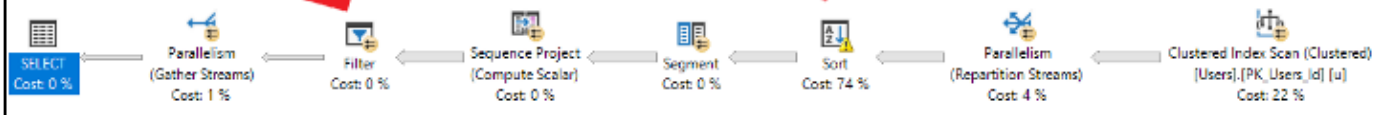
You still need indexes, pal

- The WHERE clause is obvious
- Support your ORDER BY
- Cover any select list columns

1.3 p41

CTEs also need indexes

```
WITH slow_cte
    AS
     (
      SELECT u.Id,
             u.DisplayName,
             ROW_NUMBER() OVER ( PARTITION BY u.Reputation
                                 ORDER BY u.LastAccessDate DESC ) AS rn
      FROM   dbo.Users AS u
     )
SELECT *
FROM   slow_cte
WHERE slow_cte.rn <= 3;
```



1.3 p42

P.O.C. – No, not like your car

Partition: key columns should lead with Partition by columns

Order: next should come any columns you order by

Cover: where sane and reasonable, cover selected columns

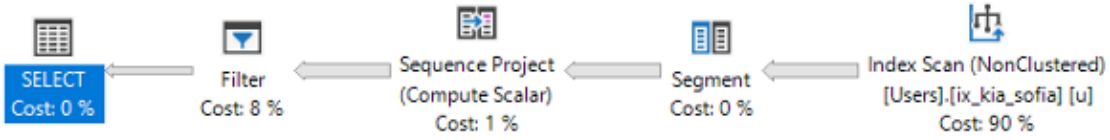
```
WITH slow_cte
    AS
     (
      SELECT u.Id,
             u.DisplayName,
             ROW_NUMBER() OVER ( PARTITION BY u.Reputation
                                 ORDER BY u.LastAccessDate DESC ) AS rn
      FROM   dbo.Users AS u
```

```
CREATE INDEX ix_kia_sofia
    ON dbo.Users ( Reputation, LastAccessDate DESC)
         INCLUDE ( DisplayName );
```

1.3 p43

Happy little plan

This plan does a whole lot less work



No need to sort data – all we have to do is assign row numbers as they come out of the index, and then filter

Our cameras: temp tables, CTEs, APPLY.

- Help to separate units of work (especially SELECT * soon)
- Make queries easier to read
- Avoid (or embrace) repeated processing

CTEs are also great for

- Recursing over hierarchies
- Paging queries

APPLY is also great for

- Top N per group
- Working with XML

1.3 p46

	Temp tables & table variables	CTEs & APPLY
Scope (lasts for)	Your session, with gotchas	Just one statement
Materialized to disk*	Yes	No
Executed	Just once, when you insert	Can be multiple times even in one query
Has statistics (which can be good or bad)	Temp tables: yes Table variables: no, til 2019	No

* Not every temp object gets written to disk, and CTE/APPLY can cause TempDB spills just like any other query operation

1.3 p47

These are more advanced tools

Like everything in SQL Server, they aren't fast or efficient by themselves – it's up to you to design queries and indexes to make them fast and efficient

These tools aren't always the best in every situation, either – but now that you know about them, it's also up to you to test them to see where they fit into tuning efforts

In the end, the optimizer may just laugh at your hard work

1.3 p48