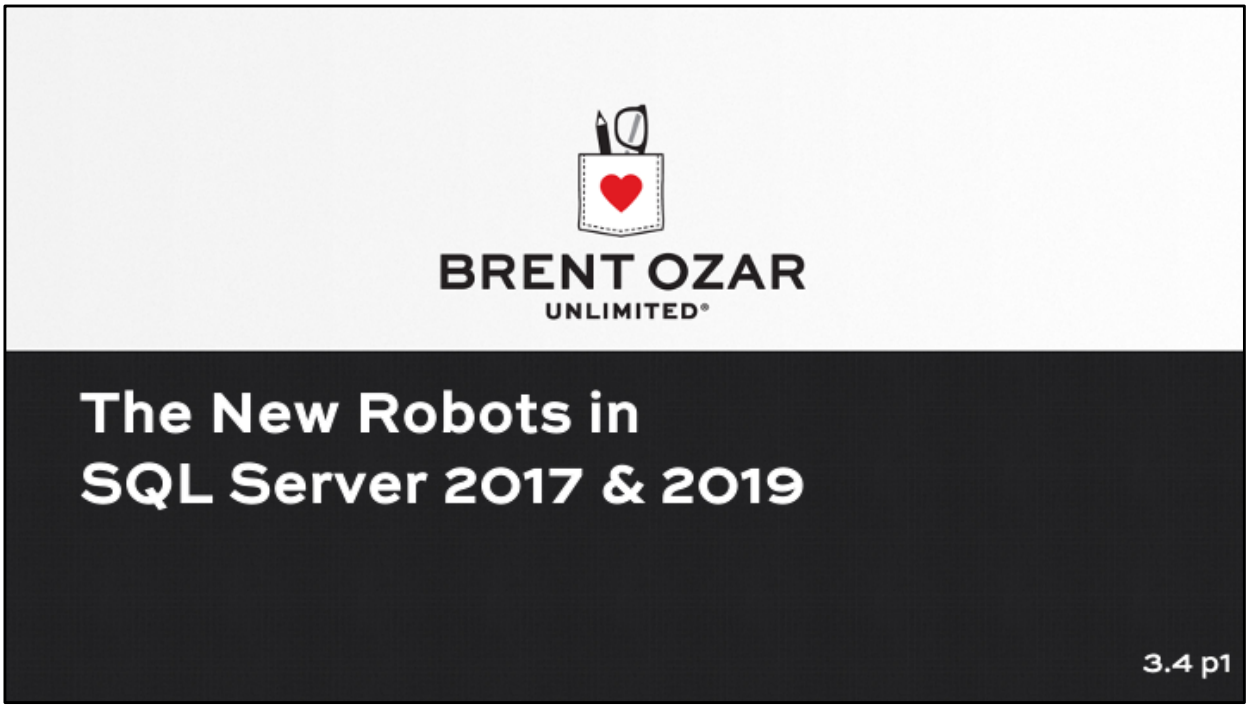
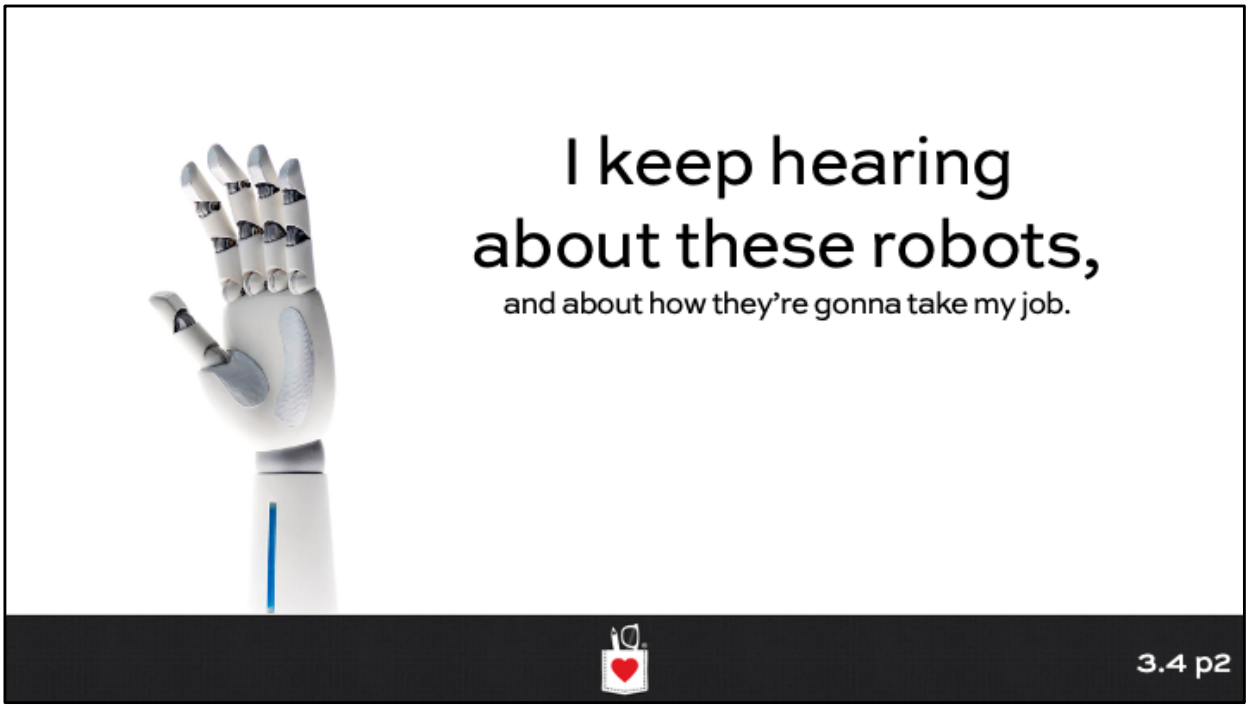
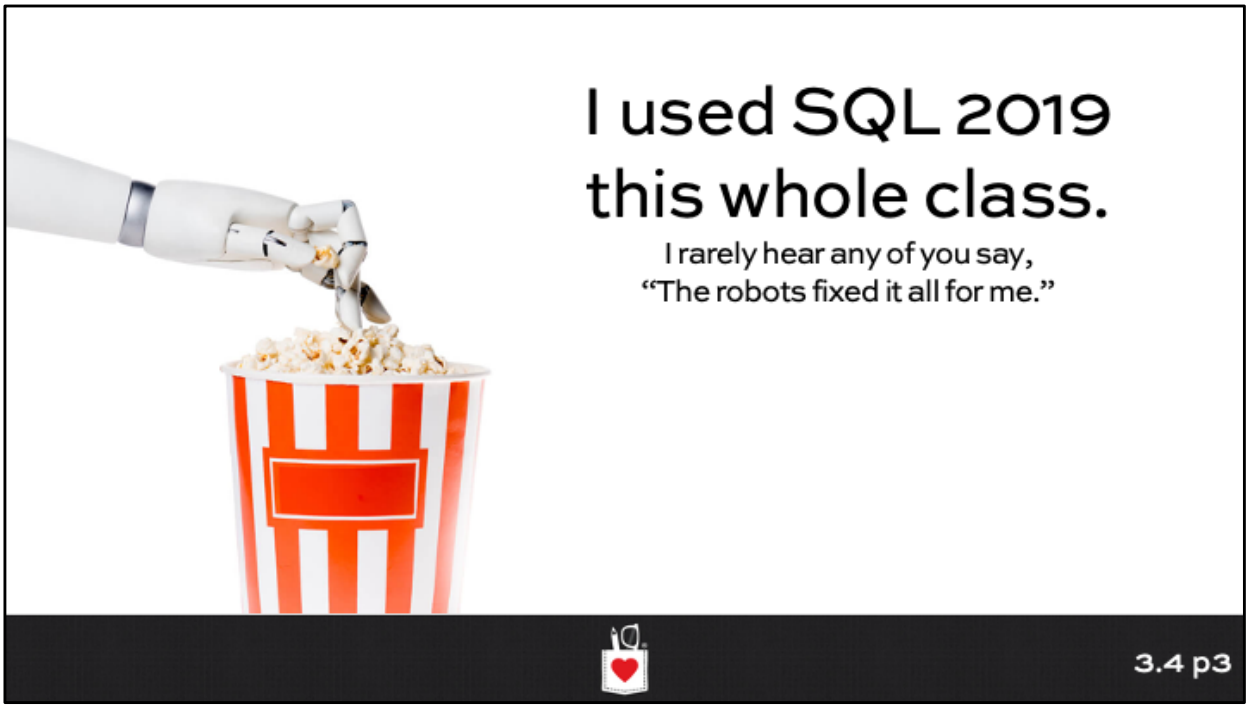






2017 & 2019's robots are kinda like your phone's AutoCorrect.

You're probably not going to disable them, but...

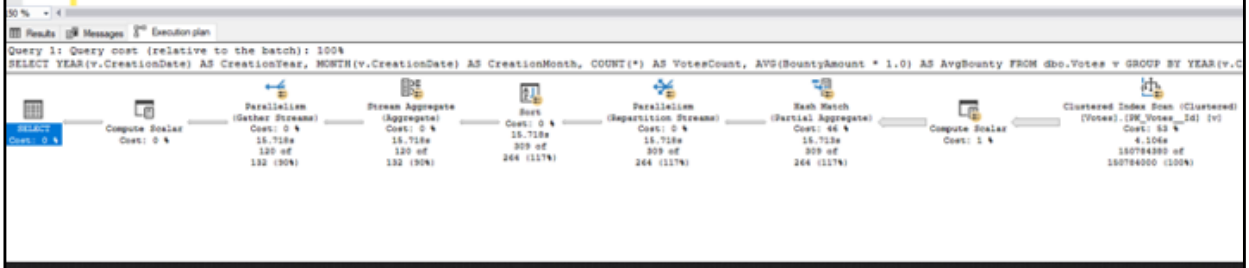
3.4 p4

	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Automatic tuning										
Query Store automatic plan correction	Yes	Yes	Enterprise	140	Yes	-	-	Yes	-	OK
Automatic index management	-	-	-	140	Yes	-	-	Yes	-	OK
Batch mode execution										
For columnstore indexes	Yes	Yes	Standard	140	Yes	-	Kinda	Yes	-	Great!
For rowstore indexes	-	Yes	Enterprise	150	Yes	-	Kinda	Yes	-	Great!
Functions										
TVF interleaved execution	Yes	Yes	Standard	140	Yes	-	Yes	Yes	-	OK
Scalar function inlining	-	Yes	Standard	150	-	-	Yes	Yes	Yes	Test it hard
Memory grant feedback (adaptive grants)	-									
For columnstore indexes	Yes	Yes	Enterprise	140	Yes	-	Kinda	Yes	-	OK
For rowstore indexes	-	Yes	Enterprise	150	Yes	-	Kinda	Yes	Yes	Test it hard
Adaptive joins										
For plans with columnstore indexes	Yes	Yes	Enterprise	140	Yes	-	Yes	Yes	-	OK
For all plans	-	Yes	Enterprise	150		-	Yes	Yes	-	OK
Others										
Approximate Count Distinct	-	Yes	Standard	150	Yes	Yes	Yes	Yes	-	OK
Table variable deferred compilation	-	Yes	Standard	150	Yes	-	Yes	Yes	-	OK

3.4 p5

Reporting-style query on 2017:

```
72  SET STATISTICS IO, TIME ON;
73  ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140;
74  GO
75  SELECT YEAR(v.CreationDate) AS CreationYear, MONTH(v.CreationDate) AS CreationMonth,
76      COUNT(*) AS VotesCount,
77      AVG(BountyAmount * 1.0) AS AvgBounty
78    FROM dbo.Votes v
79    GROUP BY YEAR(v.CreationDate), MONTH(v.CreationDate)
80    ORDER BY YEAR(v.CreationDate), MONTH(v.CreationDate)
81  GO
```



3.4 p7

Lots of reads, lots of CPU work.

```
72  SET STATISTICS IO, TIME ON;
73  ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140;
74  GO
75  SELECT YEAR(v.CreationDate) AS CreationYear, MONTH(v.CreationDate) AS CreationMonth,
76      COUNT(*) AS VotesCount,
77      AVG(BountyAmount * 1.0) AS AvgBounty
78    FROM dbo.Votes v
79    GROUP BY YEAR(v.CreationDate), MONTH(v.CreationDate)
80    ORDER BY YEAR(v.CreationDate), MONTH(v.CreationDate)
81  GO
```

150 %

Results | Messages | Execution plan

```
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead
Table 'Votes'. Scan count 5, logical reads 693459, physical reads 0, page server reads 0, read-ahea
Warning: Null value is eliminated by an aggregate or other SET operation.

(1 row affected)

SQL Server Execution Times:
  CPU time = 60969 ms,  elapsed time = 15794 ms.
```

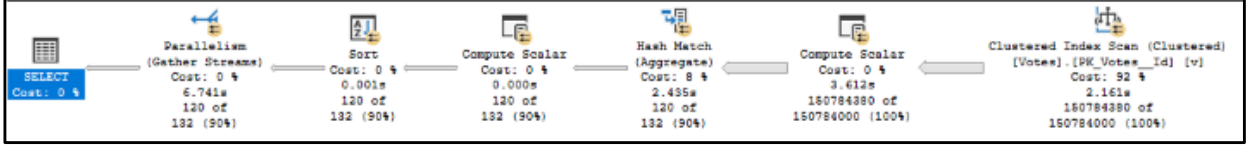
Change the compat level to 2019:

```
84  /* Then try in 2019: */
85  ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 150;
86  GO
87  SELECT YEAR(v.CreationDate) AS CreationYear, MONTH(v.CreationDate) AS CreationMonth,
88      COUNT(*) AS VotesCount,
89      AVG(BountyAmount * 1.0) AS AvgBounty
90    FROM dbo.Votes v
91    GROUP BY YEAR(v.CreationDate), MONTH(v.CreationDate)
92    ORDER BY YEAR(v.CreationDate), MONTH(v.CreationDate)
93  GO
```

150 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT YEAR(v.CreationDate) AS CreationYear, MONTH(v.CreationDate) AS CreationMonth, COUNT(*) AS VotesCount, AVG(BountyAmount * 1.0)
```



Same reads, but less CPU & time!

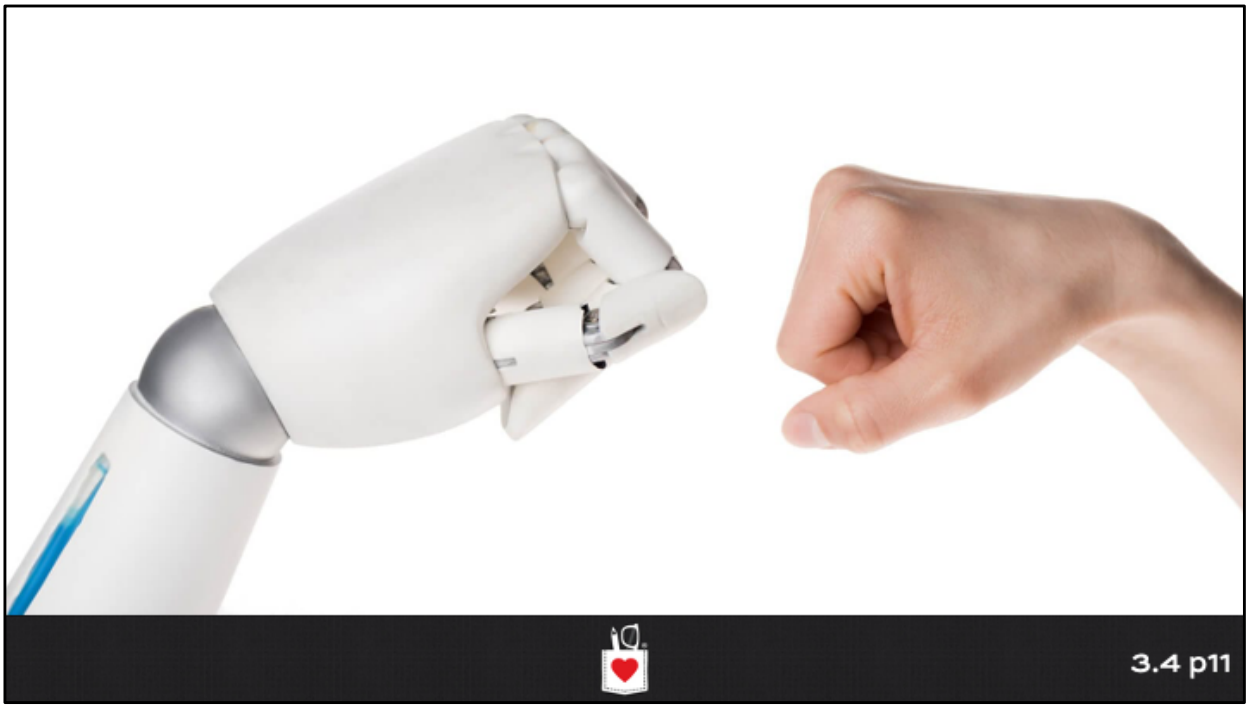
2017 was:

```
Table 'Votes'. Scan count 5, logical reads 693459,
Warning: Null value is eliminated by an aggregate o

(1 row affected)

SQL Server Execution Times:
  CPU time = 60969 ms,  elapsed time = 15794 ms
```

2019:

```
Table 'Votes'. Scan count 5, logical reads 693884,
Table 'Worktable'. Scan count 0, logical reads 0, p
Warning: Null value is eliminated by an aggregate o

(1 row affected)

SQL Server Execution Times:
  CPU time = 22313 ms,  elapsed time = 6823 ms.
```

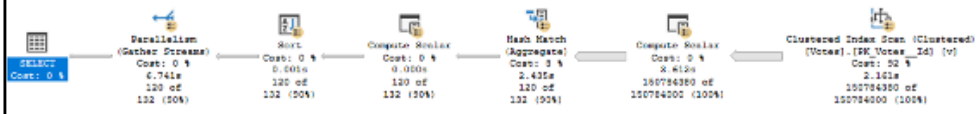
3.4 p10



It's not just a simpler plan:

2017 plan has two parallelism zones:

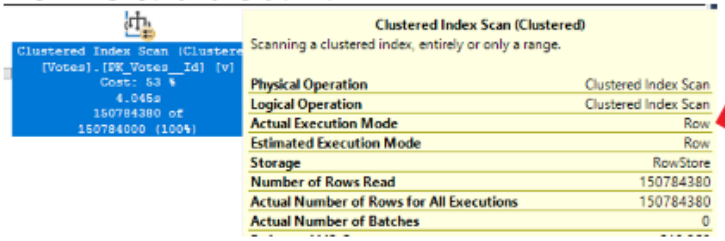
2019 plan – only one parallelism zone:



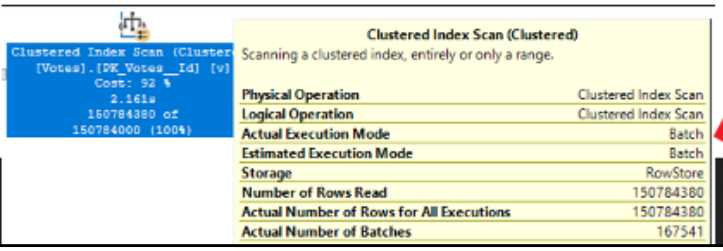
3.4 p12

It's different operators, really:

2017's table scan:



2019's:



3.4 p13

2017 & prior: row mode execution

Each plan operator is a mini-program

It runs in isolation

It passes data to the next operator in the form of rows

3.4 p14

Then came columnstore indexes.

The data is stored differently.

I describe this in Fundamentals of Columnstore.

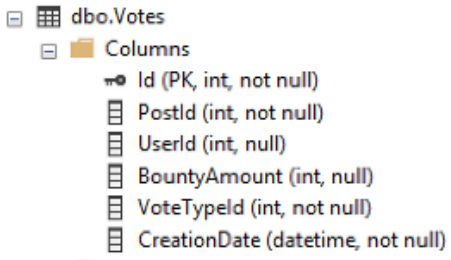
Paper: Query Execution in Column-Oriented Database Systems by Daniel J. Abadi:
http://www.cs.umd.edu/~abadi/papers/abadiphd.pdf

Niko Neugebauer 100+part blog series:
http://www.nikoport.com/columnstore/

3.4 p15

Why this mattered

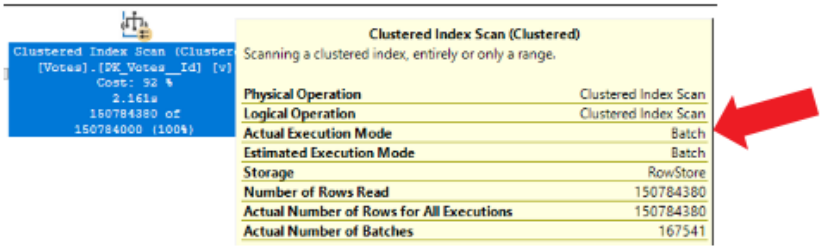
Think of it as a separate index on each column:



But here's the trick question:
when do you reassemble these columns
into their original rows?

Batch mode execution means

This operator isn't rebuilding the rows yet:
it's passing groups of columns to the next operator.



3.4 p17

Like an index on every column

Think of it as a separate index on each column:

- dbo.Votes
  - Columns
    - Id (PK, int, not null)
    - PostId (int, not null)
    - UserId (int, null)
    - BountyAmount (int, null)
    - VoteTypeId (int, not null)
    - CreationDate (datetime, not null)

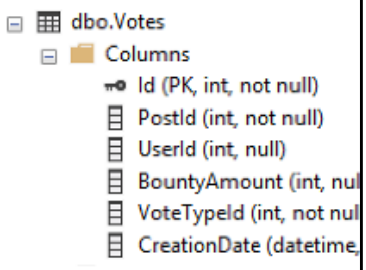
But here's the trick question:
when do you reassemble these columns
into their original rows?

So many cool implications

Batches of columnar data can be:

- Way smaller due to duplication (only unique values are stored)
- Way easier to group/sum/sort on (since so much is duplicated)

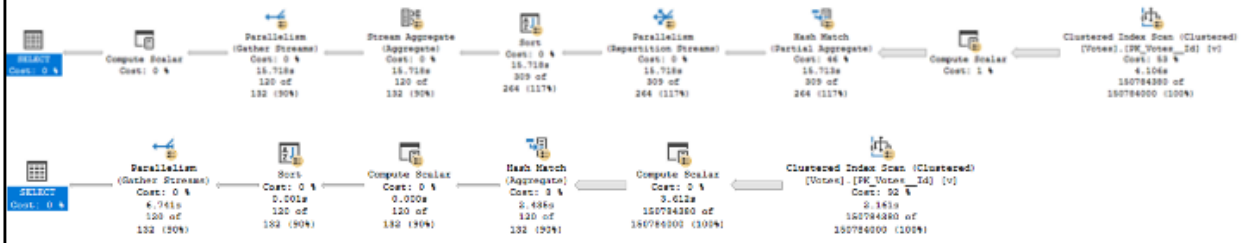
Result: massively less CPU work for reporting queries (table scans with grouping, summing, formulas.)



3.4 p19

The first challenge

Which of these plans is getting batch mode?



And which of the operators are getting it?

3.4 p20

More challenges

If you got batch mode on an operator:

- Is it working out in your favor?
- Why was it chosen?
- How can you change it?

If you didn't get batch mode:

- Why not?
- Can you change things to get it?

	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Batch mode execution										
For columnstore indexes	Yes	Yes	Standard	140	Yes	-	Kinda	Yes	-	Great!
For rowstore indexes	-	Yes	Enterprise	150	Yes	-	Kinda	Yes	-	Great!

Columnstore gets it in Standard, but it just works on columnstore indexes, period. No big gotchas there.

Rowstore tables only get it in Enterprise Edition.

I think that's a perfect fit for batch mode:

- It makes the most sense in reports
- It makes the least sense in OLTP

3.4 p23

Multi-Statement Table-Valued Functions

We covered why these suck:

- Hard to see impact in the query plan, stats IO
- Can run row-by-row
- Estimates are hard-coded garbage (100 or 1)

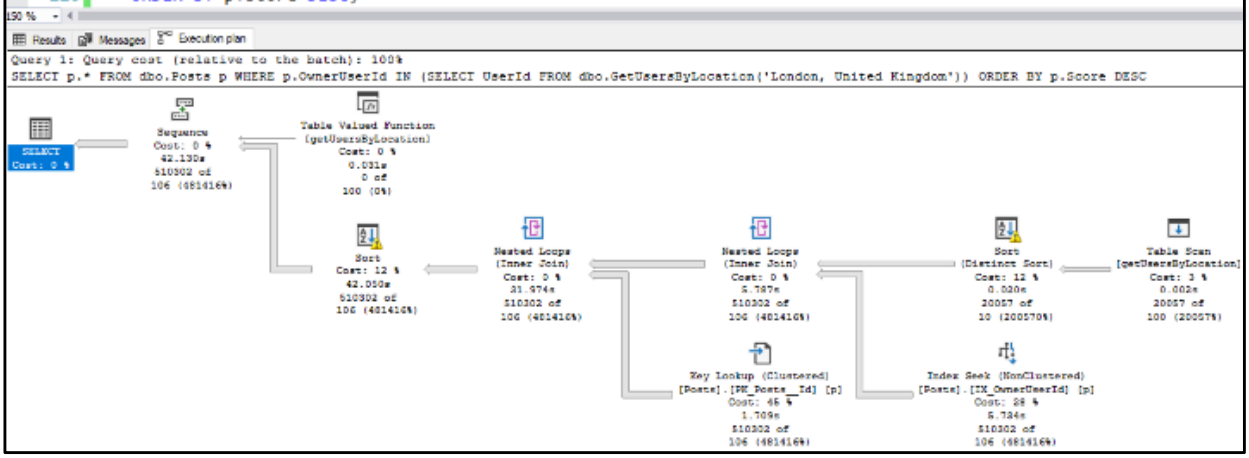
2017 can fix the latter by running the MSTVF first, then passing the row estimates to the rest of the plan.

3.4 p25

```
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 130; /* 2016 at first */
GO
CREATE OR ALTER FUNCTION dbo.getUsersByLocation ( @Location NVARCHAR(100) )
RETURNS @Out TABLE ( UserId INT )
    WITH SCHEMABINDING
AS
    BEGIN
        INSERT  INTO @Out(UserId)
        SELECT  Id
        FROM    dbo.Users
        WHERE   Location = @Location;
        RETURN;
    END;
GO


/* List the posts for the users in a specific location: */
SELECT p.*
  FROM dbo.Posts p
  WHERE p.OwnerUserId IN (SELECT UserId FROM dbo.GetUsersByLocation('London, United Kingdom'))
  ORDER BY p.Score DESC;
```

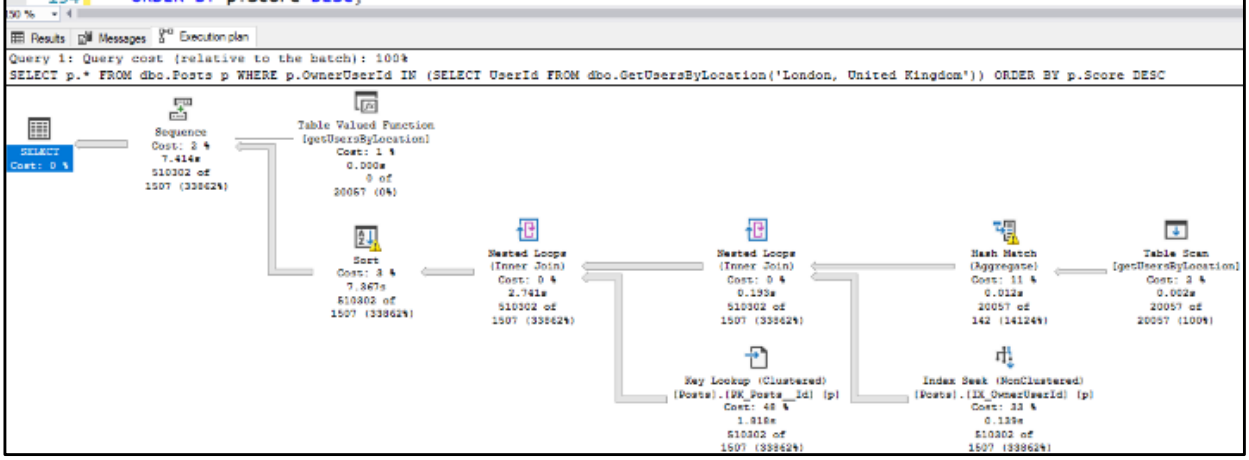
It takes 42 seconds

```
122  /* List the posts for the users in a specific location: */
123  SELECT p.*
124    FROM dbo.Posts p
125    WHERE p.OwnerUserId IN (SELECT UserId FROM dbo.GetUsersByLocation('London, United Kingdom'))
126    ORDER BY p.Score DESC;
```



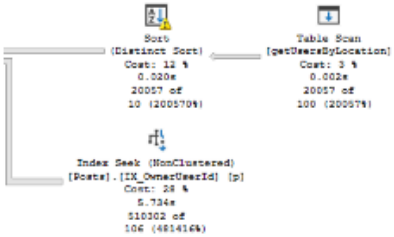
2017 compat mode: 7 seconds!

```
129  ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140; /* Now try 2017 */
130  GO
131  SELECT p.*
132    FROM dbo.Posts p
133    WHERE p.OwnerUserId IN (SELECT UserId FROM dbo.GetUsersByLocation('London, United Kingdom'))
134    ORDER BY p.Score DESC;
```

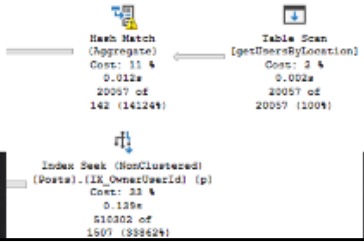


The difference

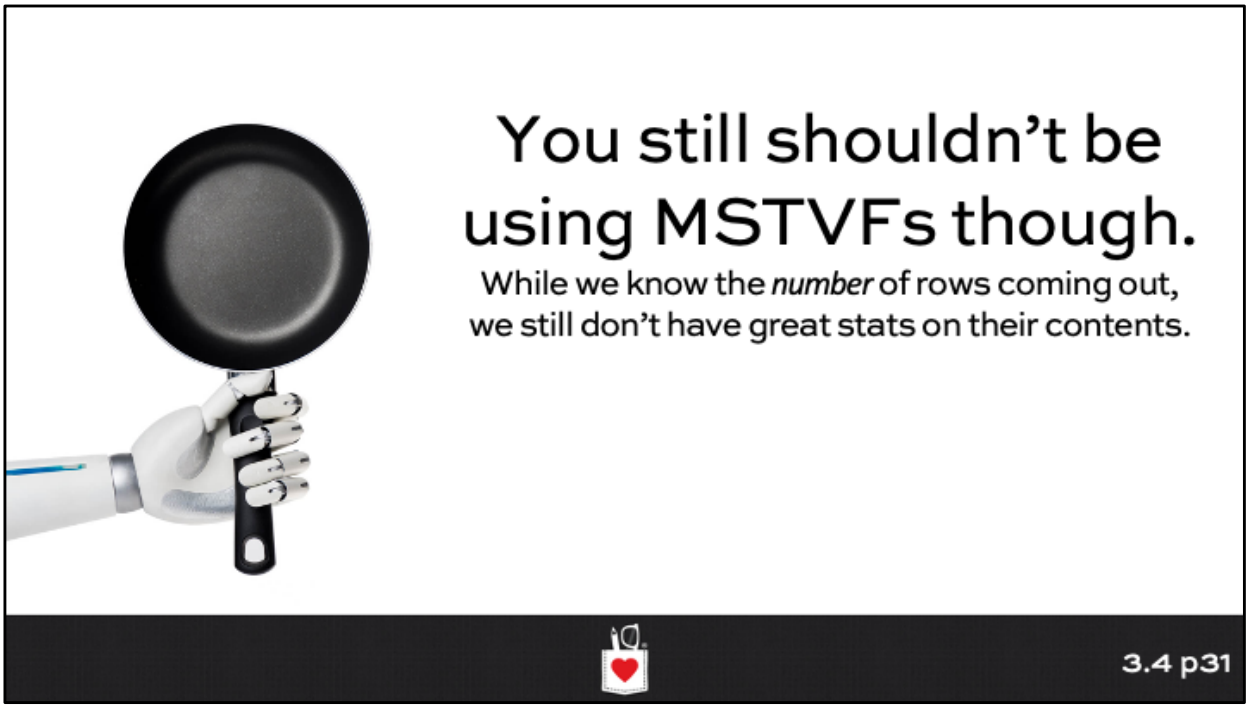
2016 estimates a hard-coded 100 rows from the TVF:



But 2017+ executes it, and knows 20,057 rows will come out:

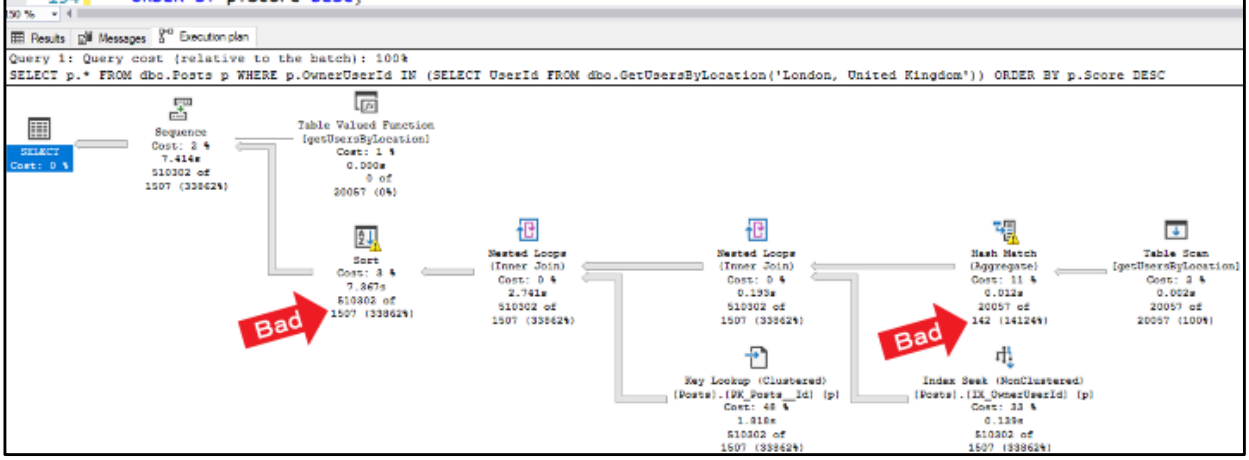


3.4 p29



The sort & hash match both spill.

```
129  ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140; /* Now try 2017 */
130  GO
131  SELECT p.*
132    FROM dbo.Posts p
133    WHERE p.OwnerUserId IN (SELECT UserId FROM dbo.GetUsersByLocation('London, United Kingdom'))
134    ORDER BY p.Score DESC;
```



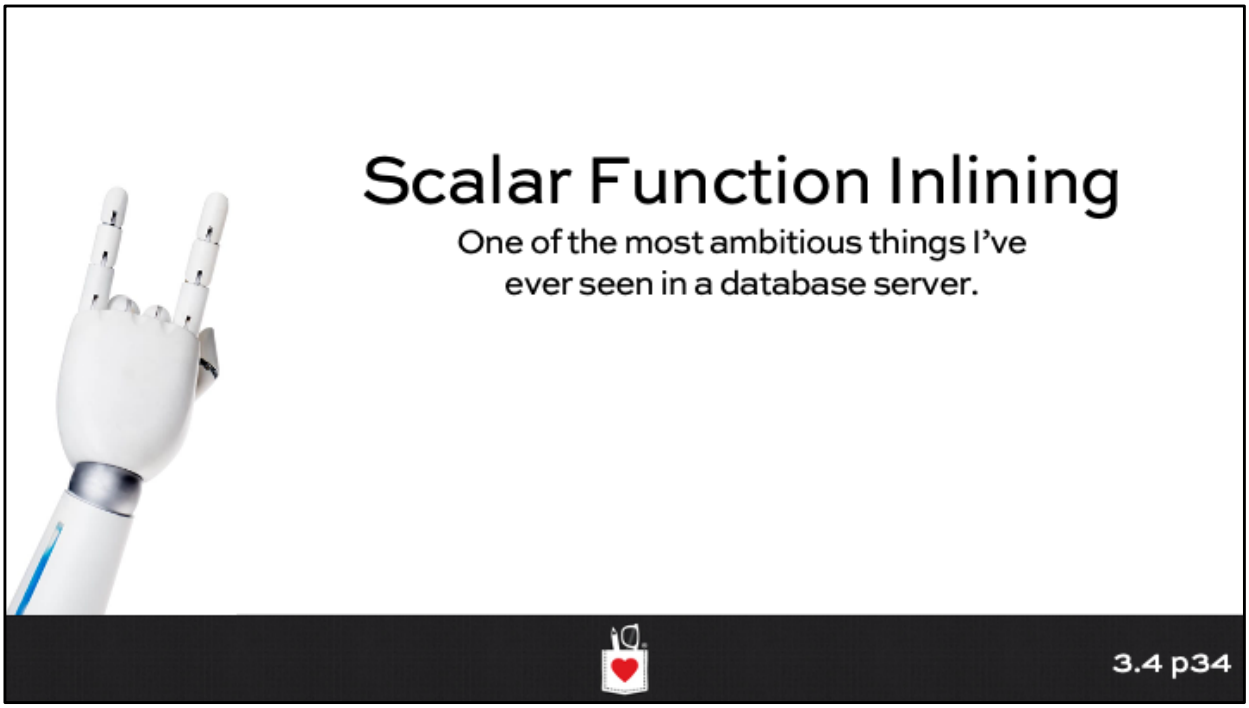
	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Functions										
TVF interleaved execution	Yes	Yes	Standard	140	Yes	-	Yes	Yes	-	OK
Scalar function inlining	-	Yes	Standard	150	-	-	Yes	Yes	Yes	Test it hard

TVF interleaved execution is fine.

It makes a bad thing (TVFs) suck less.

Scalar function inlining, that's different.

3.4 p33



The idea

Take scalar user-defined functions that have always been designed to run on a row-by-row basis

1. Rewrite the UDFs automatically into set-based functions (think: turn them into a view you can join to)
2. Rewrite queries calling the UDFs so that they can leverage the set-based version

White paper: https://arxiv.org/abs/1712.00498
(Click the PDF link in the Download box)

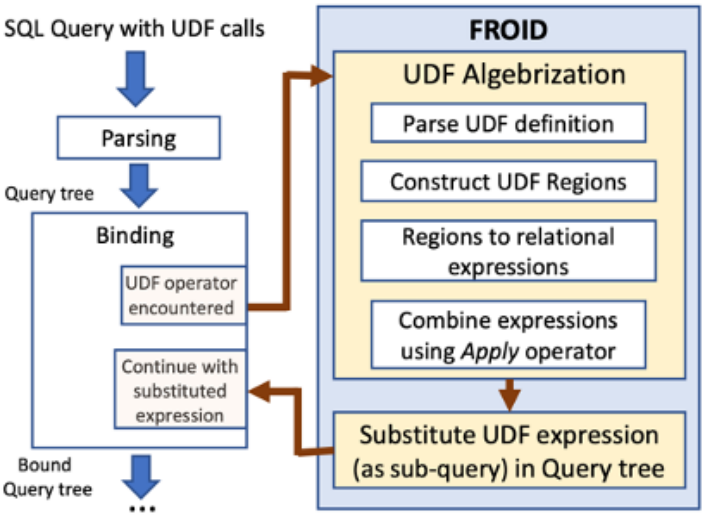


Figure 3: Overview of the Froid framework

3.4 p36

This paper is fun (for me) to read

Imperative Statement (T-SQL)	Relational expression (T-SQL)
DECLARE {@*var data_type* [= *expr*]}[, . . . *n*];	SELECT {*expr*|*null* AS *var*}[, . . . *n*];
SET {@*var* = *expr*}[, . . . *n*];	SELECT {*expr* AS *var*}[, . . . *n*];
SELECT {@*var*1 = *prj_expr*1}[, . . . *n*] FROM *sql_expr*;	{SELECT *prj_expr*1 AS *var*1 FROM *sql_expr*}; [, . . . *n*]
IF (*pred_expr*) {*t_stmt*; [. . . *n*]} ELSE {*f_stmt*; [, . . . *n*]}	SELECT CASE WHEN *pred_expr* THEN 1 ELSE 0 END AS *pred_val*; {SELECT CASE WHEN *pred_val* = 1 THEN *t_stmt* ELSE *f_stmt*; }[. . . *n*]
RETURN expr;	SELECT *expr* AS *returnVal*;

Table 1: Relational algebraic expressions for imperative statements (using standard T-SQL notation from [34])

3.4 p37

I was amazed that this shipped.

It's the most ambitious thing I've seen in SQL Server.

Other features were harder and more complex:
CLR, Hekaton, Polybase, Big Data Clusters.

But those had a limited blast radius because they required you to change your code, tables, indexes.

This feature was supposed to work without you changing anything at all.

3.4 p38

Well, that was the theory.

But they keep finding bugs with it, and disabling it:

FIX: Scalar UDF Inlining issues in SQL Server 2019

Applies to: SQL Server 2019 on Linux, SQL Server 2019 on Windows

Symptoms

User-Defined Functions (UDFs) that are implemented in Transact-SQL and that return a single data value are referred to as T-SQL Scalar User-Defined Functions (UDFs).

Microsoft SQL Server 2019 introduces the Scalar UDF Inlining feature that can improve the performance of queries that invoke T-SQL Scalar UDFs, where UDF execution is the main bottleneck. T-SQL Scalar UDF Inlining automatically transforms inlineable UDFs into relational expressions.

3.4 p39

At first, they fixed bugs...

This cumulative update includes several **fixes** across the following areas for scenarios in which a query that uses Scalar UDF Inlining may return an error or unexpected results:

- Type mismatch error if the return type of the UDF is a sql_variant (added in Microsoft SQL Server 2019 CU2)
- UDF invocation from sp_executesql aborts execution (added in Microsoft SQL Server 2019 CU2)
- UDFs referencing labels without an associated GOTO command return incorrect results (added in Microsoft SQL Server 2019 CU2)
- Out-of-memory conditions and memory leaks occur because of very large scalar UDFs (added in Microsoft SQL Server 2019 CU2)
- Uninitialized variables used in condition (IF-ELSE) statements cause errors (added in Microsoft SQL Server 2019 CU2)

CU2 had bug fixes, but after that, they just started disabling scalar function inlining in more scenarios...

3.4 p40

This cumulative update also **blocks** inlining in the following scenarios:

- If the UDF references certain intrinsic functions (for example, @@ROWCOUNT) that may alter the results when inlined (added in Microsoft SQL Server 2019 CU2)
- When aggregate functions are passed as parameters to a scalar UDF (added in Microsoft SQL Server 2019 CU2)
- If the UDF references built-in views (for example: OBJECT_ID) (added in Microsoft SQL Server 2019 CU2)
- If the UDF uses XML methods (added in Microsoft SQL Server 2019 CU4)
- If the UDF contains a SELECT with ORDER BY without a "TOP 1" (added in Microsoft SQL Server 2019 CU4)
- If the SELECT query performs an assignment in conjunction with the ORDER BY clause (for example, SELECT @x = @x +1 FROM table ORDER BY *column_name) (added in Microsoft SQL Server 2019 CU4)*
- If the UDF contains multiple RETURN statements (added in Microsoft SQL Server 2019 CU5)
- If the UDF is called from a RETURN statement (added in Microsoft SQL Server 2019 CU5)
- If the UDF references the STRING_AGG function (added in Microsoft SQL Server 2019 CU5)
- If the UDF definition references remote tables (added in Microsoft SQL Server 2019 CU6)
- If the UDF-calling query uses GROUPING SETS, CUBE, or ROLLUP (added in Microsoft SQL Server 2019 CU6)
- If the UDF-calling query contains a variable that is used as a UDF parameter for assignment (for example, SELECT @y=2, @x=UDF(@y)) (added in Microsoft SQL Server 2019 CU6)

And even when it “works”

It doesn’t always speed up queries, as we saw earlier.

Thus my verdict:

	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Functions										
TVF interleaved execution	Yes	Yes	Standard	140	Yes	-	Yes	Yes	-	OK
Scalar function inlining	-	Yes	Standard	150	-	-	Yes	Yes	Yes	Test it hard

It might work for you, but…
I’ve seen it backfire, HARD.

3.4 p42

Adaptive Memory Grants

3.4 p44

Query workspace memory grants

It's really hard for SQL Server to get estimates right.

It's not SQL Server's fault:
it's usually our crappy queries and indexes.

SQL Server does track how often spills occur for a given query plan, but it just doesn't do anything with that data.

3.4 p45

https://docs.microsoft.com/en-us/sql/relational-databases/performance/adaptive-query-processing

Memory grant feedback sizing

For an excessive memory grant condition, if the granted memory is more than two times the size of the actual used memory, memory grant feedback will recalculate the memory grant and update the cached plan. Plans with memory grants under 1 MB will not be recalculated for overages. For an insufficiently sized memory grant condition, that result in a spill to disk for batch mode operators, memory grant feedback will trigger a recalculation of the memory grant. Spill events are reported to memory grant feedback and can be surfaced via the *spilling_report_to_memory_grant_feedback* xEvent. This event returns the node id from the plan and spilled data size of that node.

Memory grant feedback and parameter sensitive scenarios

Different parameter values may also require different query plans in order to remain optimal. This type of query is defined as "parameter-sensitive." For parameter-sensitive plans, memory grant feedback will disable itself on a query if it has unstable memory requirements. The plan is disabled after several repeated runs of the query and this can be observed by monitoring the *memory_grant_feedback_loop_disabled* xEvent. For more information about parameter sniffing and parameter sensitivity, refer to the Query Processing Architecture Guide.

3.4 p46

Memory grant feedback caching

Feedback can be stored in the cached plan for a single execution. It is the consecutive executions of that statement, however, that benefit from the memory grant feedback adjustments. This feature applies to repeated execution of statements. Memory grant feedback will change only the cached plan. Changes are currently not captured in the Query Store. Feedback is not persisted if the plan is evicted from cache. Feedback will also be lost if there is a failover. A statement using `OPTION (RECOMPILE)` creates a new plan and does not cache it. Since it is not cached, no memory grant feedback is produced and it is not stored for that compilation and execution. However, if an equivalent statement (that is, with the same query hash) that did **not** use `OPTION (RECOMPILE)` was cached and then re-executed, the consecutive statement can benefit from memory grant feedback.

Tracking memory grant feedback activity

You can track memory grant feedback events using the *memory_grant_updated_by_feedback* xEvent. This event tracks the current execution count history, the number of times the plan has been updated by memory grant feedback, the ideal additional memory grant before modification and the ideal additional memory grant after memory grant feedback has modified the cached plan.

3.4 p47

Seeing it in action

Build our stored proc that is vulnerable to sniffing:

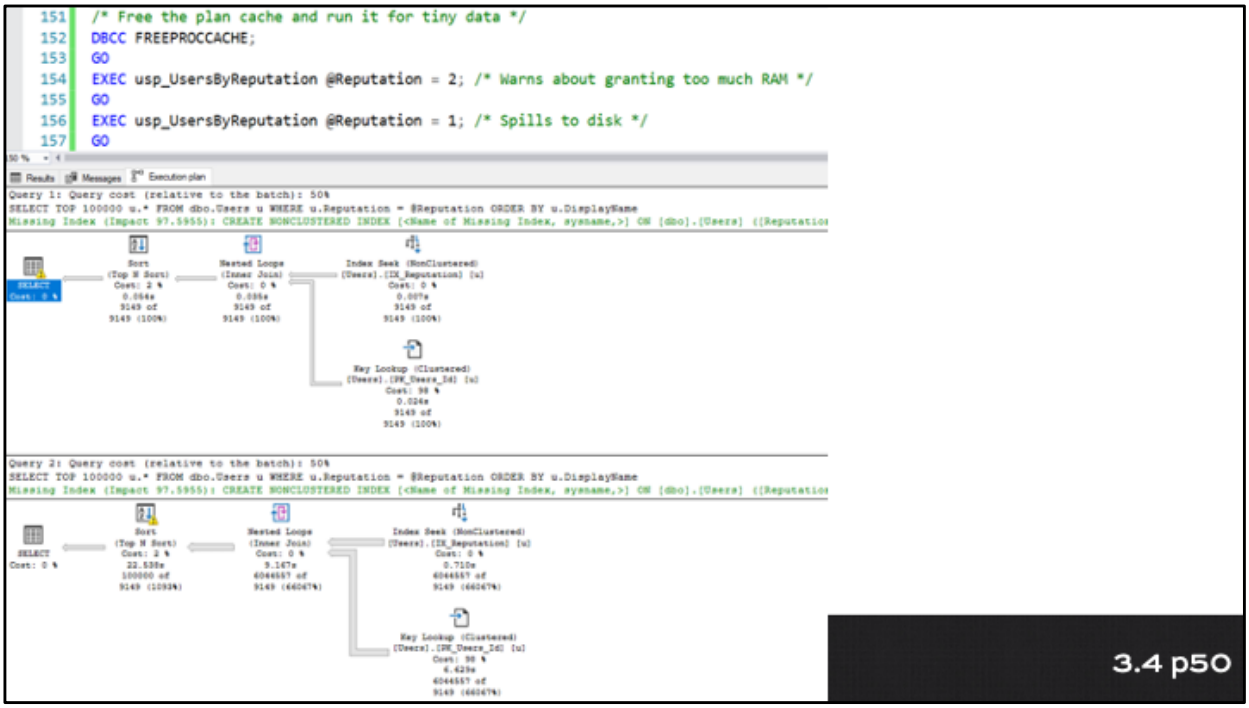
```
/* Adaptive memory grants for rowstore tables */
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140; /* 2017 */
GO
CREATE OR ALTER PROC dbo.usp_UsersByReputation @Reputation INT AS
    SELECT TOP 100000 u.*
        FROM dbo.Users u
        WHERE u.Reputation = @Reputation
        ORDER BY u.DisplayName;
GO
```

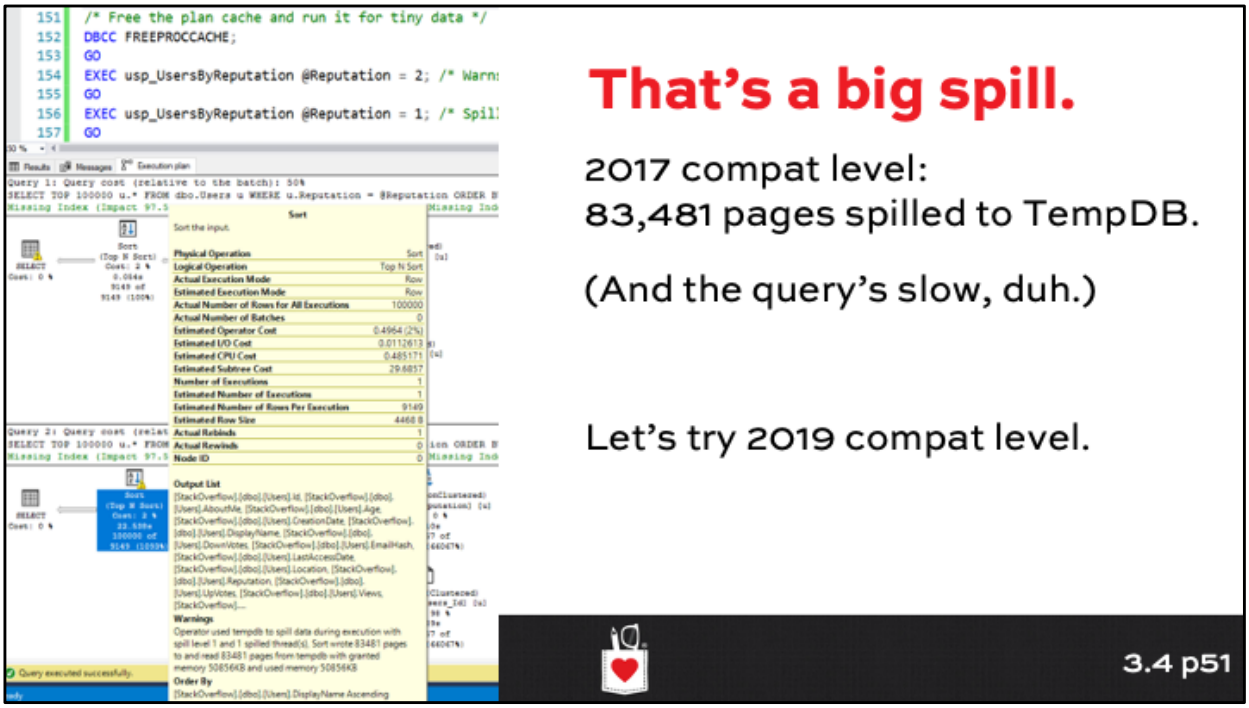
Then run it back & forth...

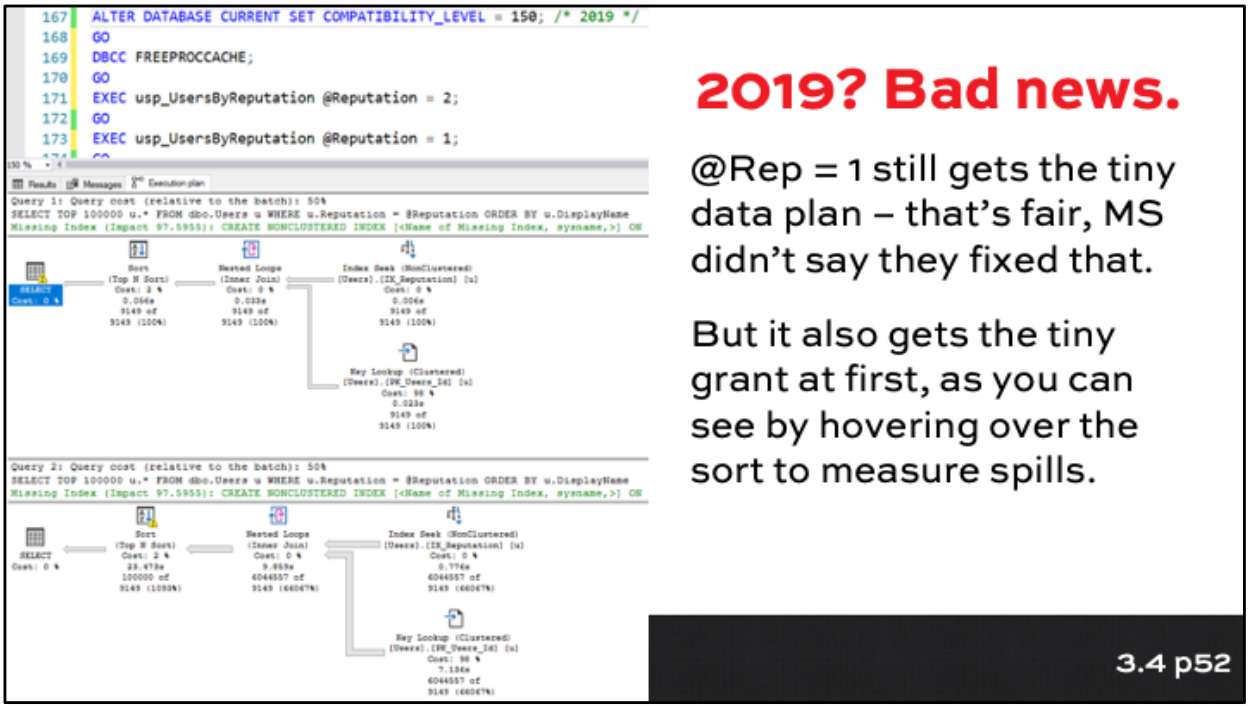
3.4 p48

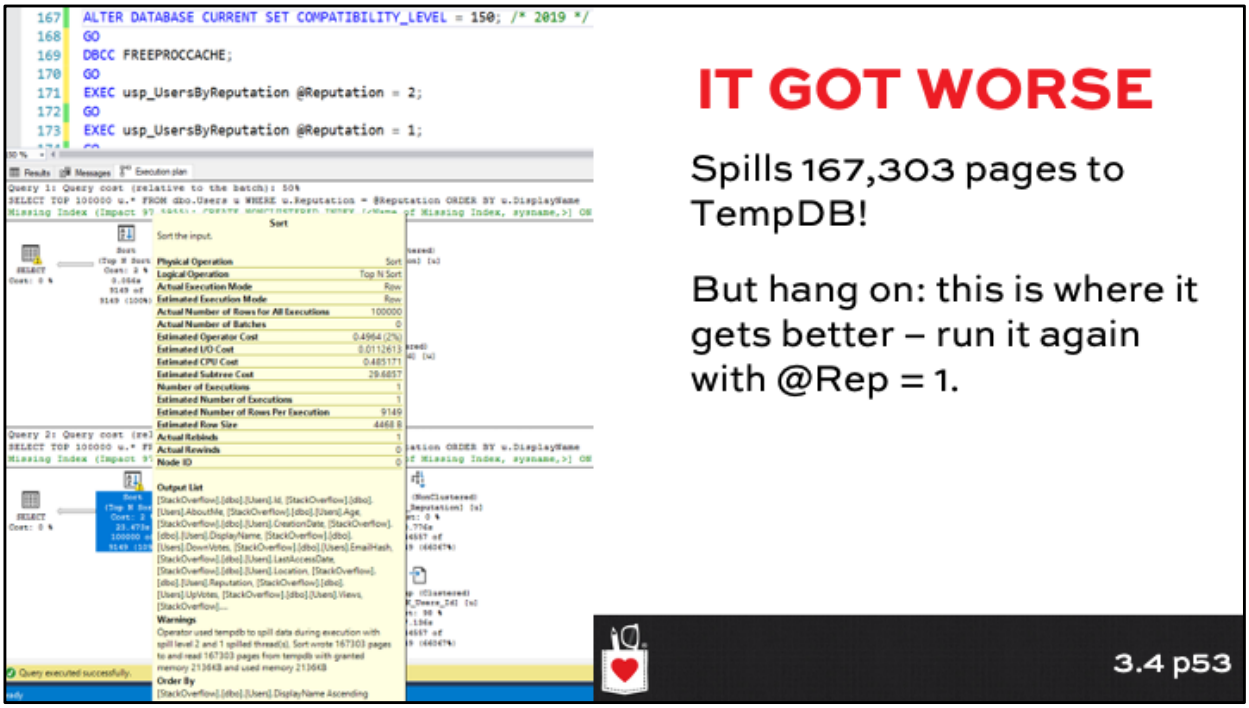
```sql
/* Adaptive memory grants for rowstore tables */
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140; /* 2017 */
GO
CREATE OR ALTER PROC dbo.usp_UsersByReputation @Reputation INT AS
    SELECT TOP 100000 u.*
        FROM dbo.Users u
        WHERE u.Reputation = @Reputation
        ORDER BY u.DisplayName;
GO

/* Free the plan cache and run it for tiny data */
DBCC FREEPROCCACHE;
GO
EXEC usp_UsersByReputation @Reputation = 2; /* Warns about granting too much RAM */
GO
EXEC usp_UsersByReputation @Reputation = 1; /* Spills to disk */
GO
/* Now do it in the opposite order: */
DBCC FREEPROCCACHE;
GO
EXEC usp_UsersByReputation @Reputation = 1; /* Grants a ton of memory */
GO
EXEC usp_UsersByReputation @Reputation = 2; /* Leaves a ton of unused memory on the floor */
GO
```

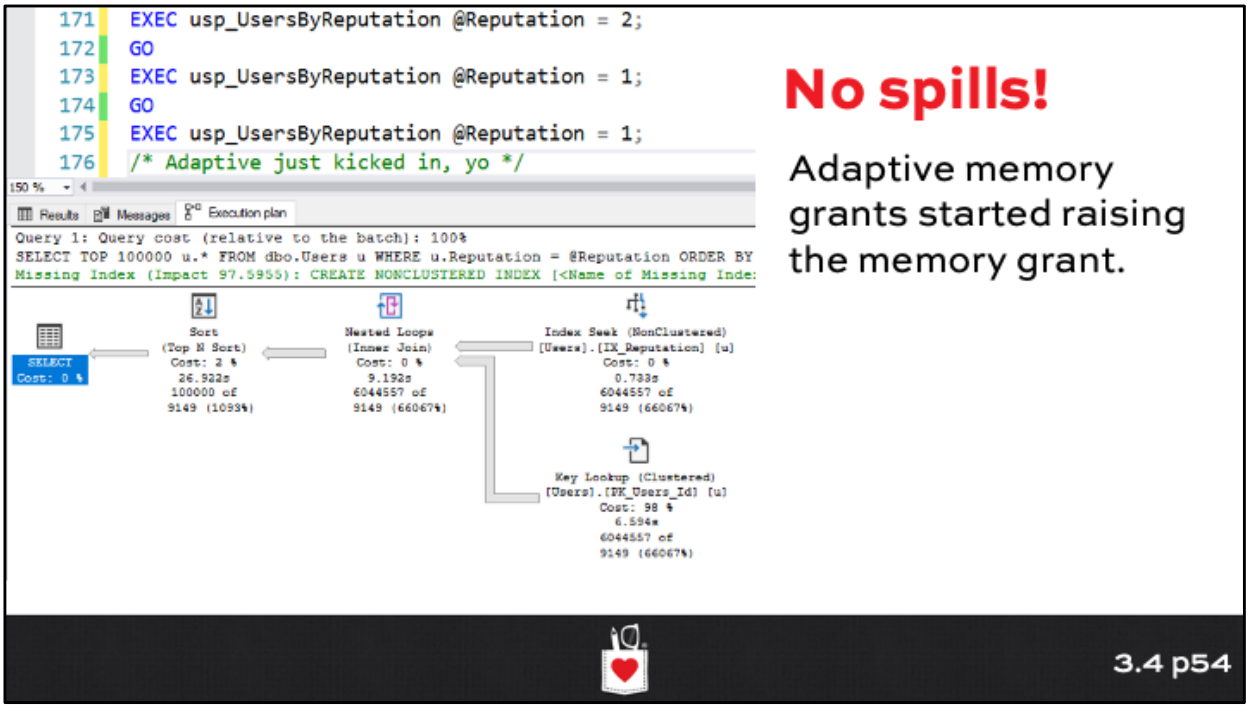
p49











How it works

After a query finishes, SQL reviews its memory use.

- If it spilled to disk, it didn't get enough RAM.
  Raise its desired memory grant for next time.
- If it left a lot of RAM unused, it got too much.
  Lower its desired memory grant for next time.

3.4 p55

How it works, translated

The amount of memory your query gets is based on the parameters that the last person used.

If the last person asked for big data, you get big RAM.

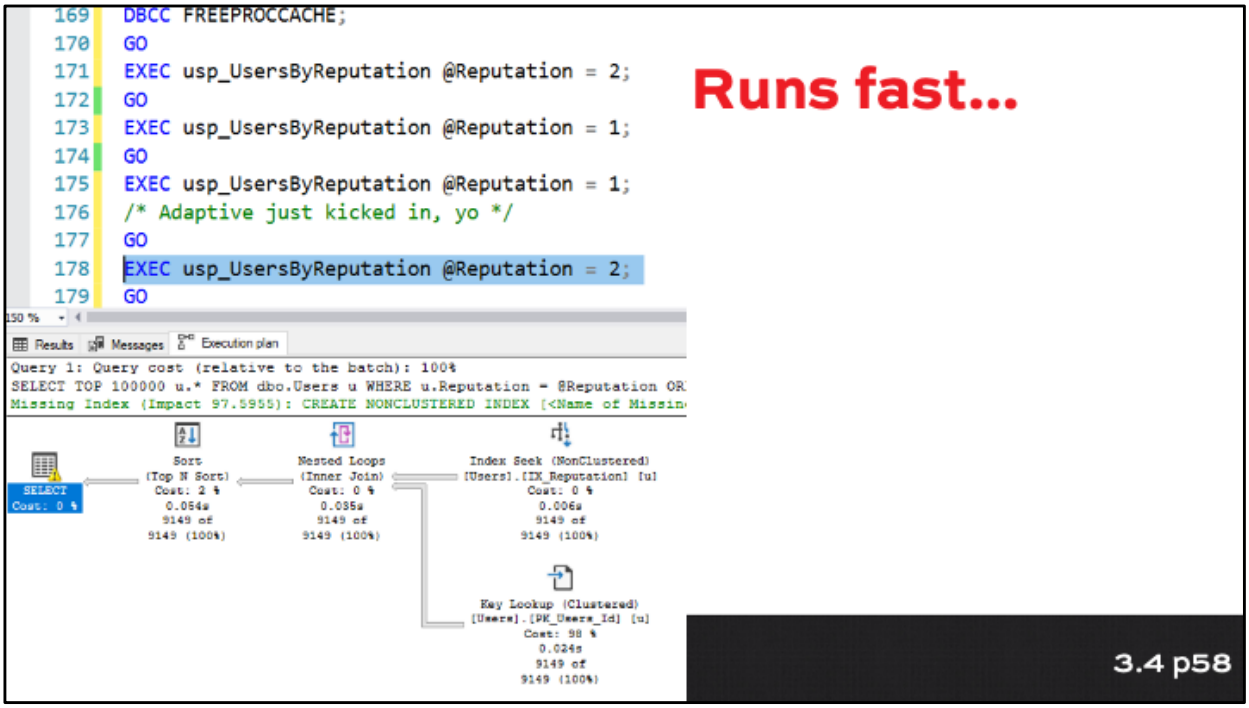
If they asked for small data, you get small RAM.

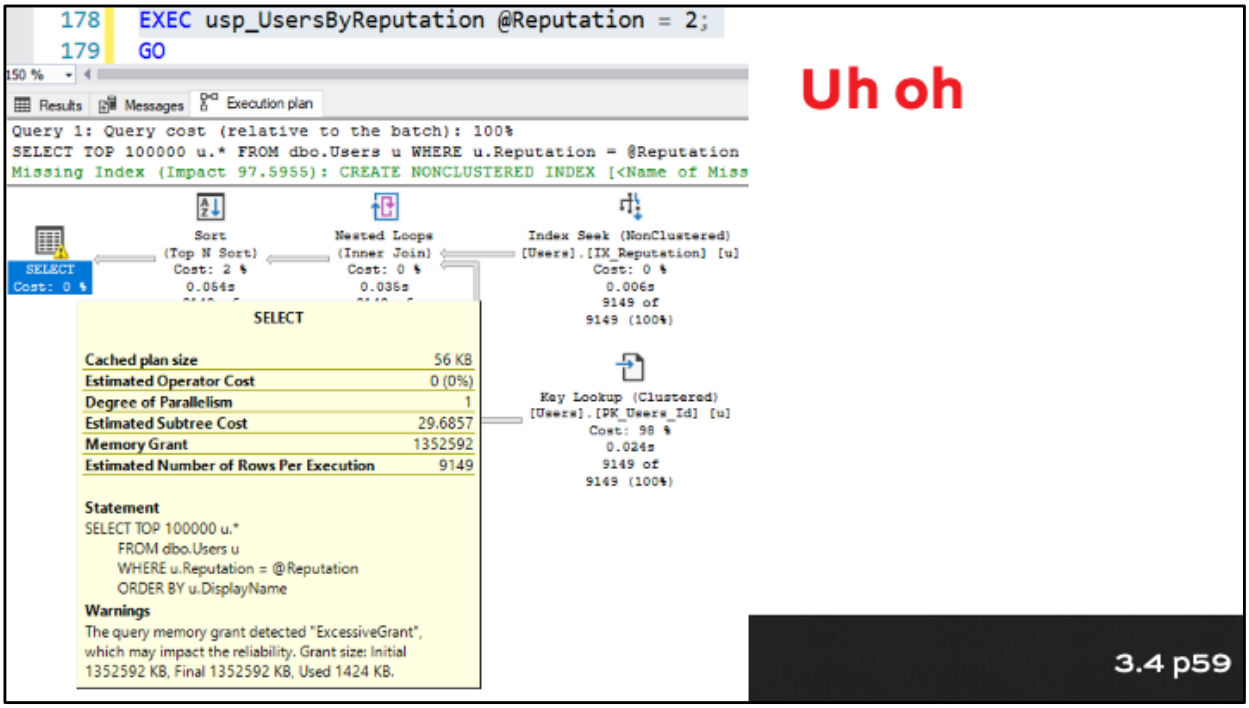
3.4 p56

So what's gonna happen now?

I'm about to run it for reputation = 2. What happens?

```
169  DBCC FREEPROCCACHE;
170  GO
171  EXEC usp_UsersByReputation @Reputation = 2;
172  GO
173  EXEC usp_UsersByReputation @Reputation = 1;
174  GO
175  EXEC usp_UsersByReputation @Reputation = 1;
176  /* Adaptive just kicked in, yo */
177  GO
178  EXEC usp_UsersByReputation @Reputation = 2;
179  GO
```

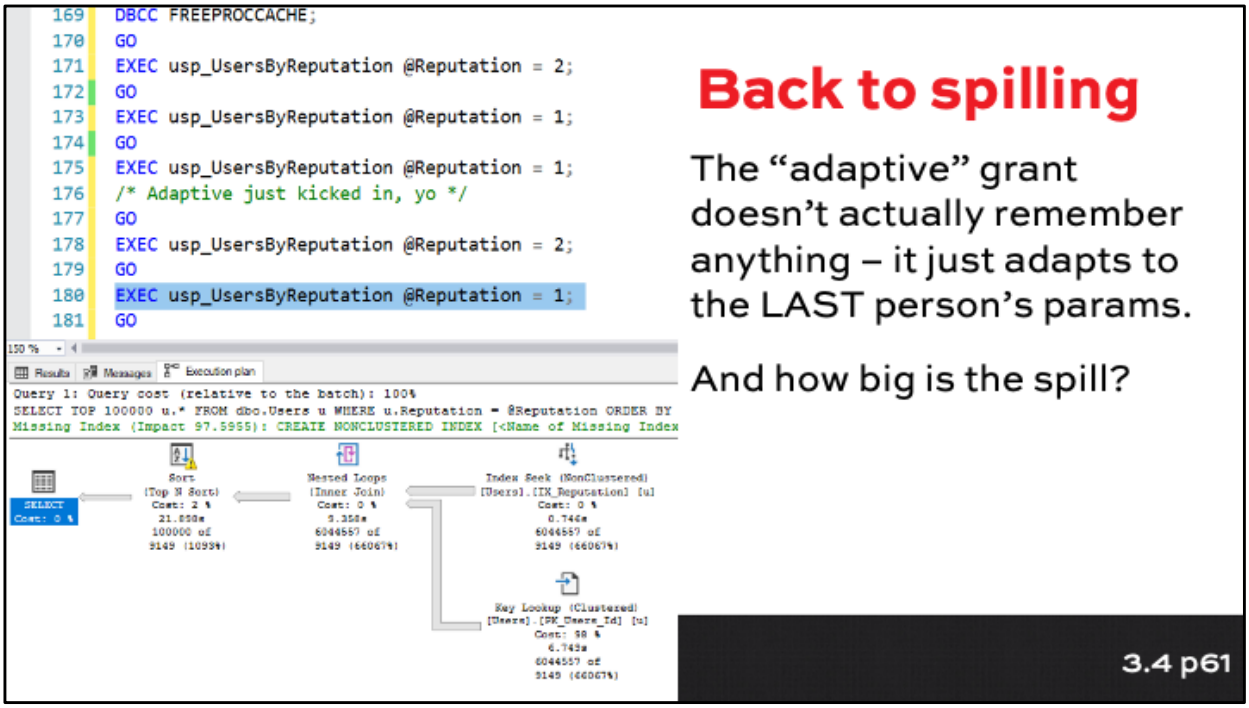


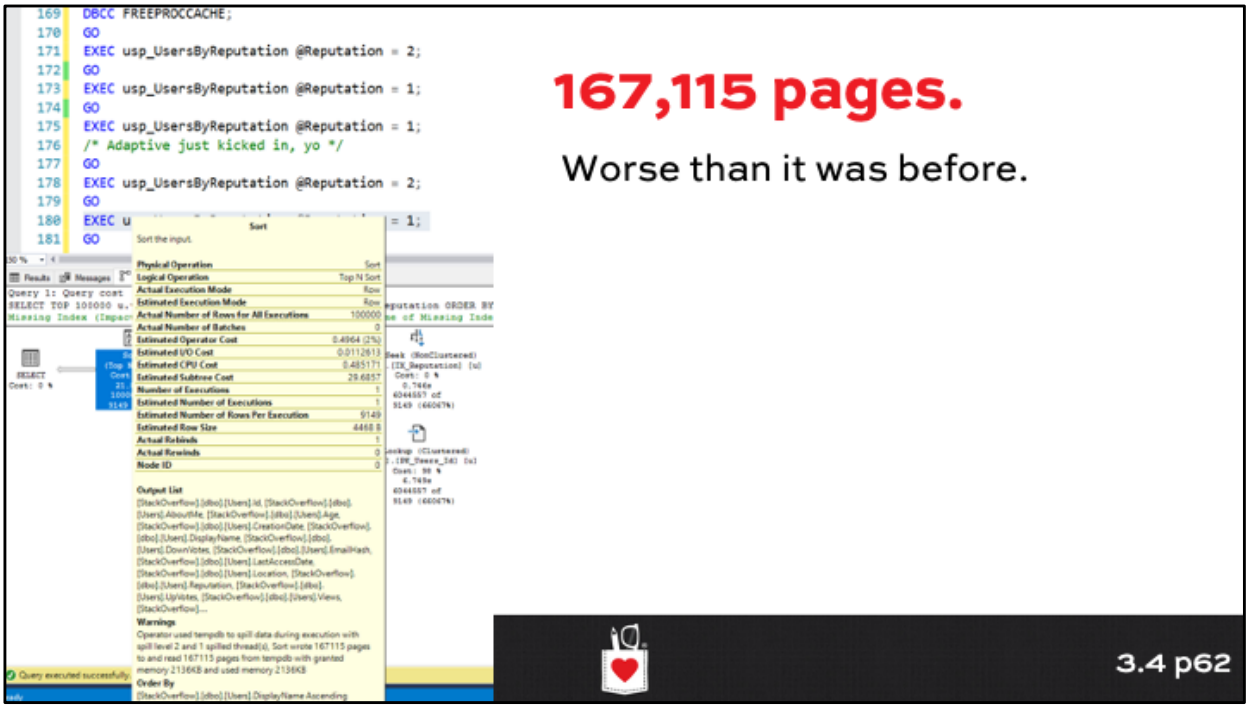


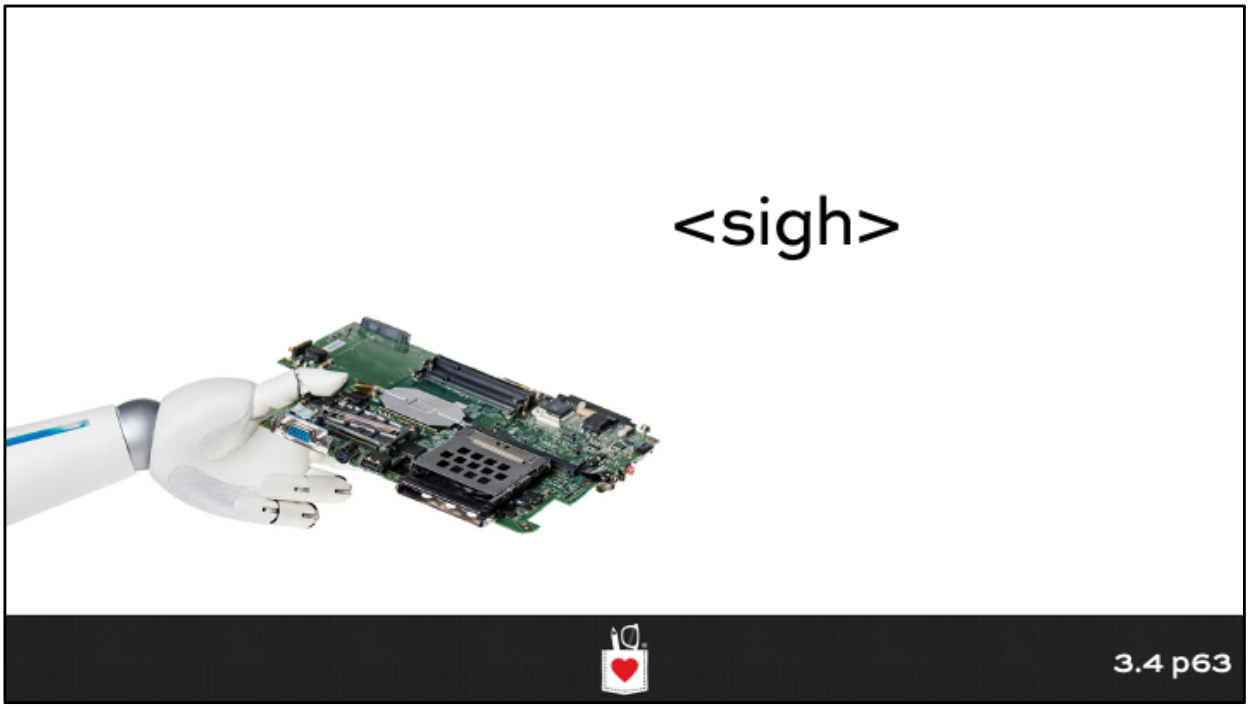
You see this coming, don't you?

```
169  DBCC FREEPROCCACHE;
170  GO
171  EXEC usp_UsersByReputation @Reputation = 2;
172  GO
173  EXEC usp_UsersByReputation @Reputation = 1;
174  GO
175  EXEC usp_UsersByReputation @Reputation = 1;
176  /* Adaptive just kicked in, yo */
177  GO
178  EXEC usp_UsersByReputation @Reputation = 2;
179  GO
180  EXEC usp_UsersByReputation @Reputation = 1;
181  GO
```

3.4 p60







	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Memory grant feedback (adaptive grants)	-									
For columnstore indexes	Yes	Yes	Enterprise	140	Yes	-	Kinda	Yes	-	OK
For rowstore indexes	-	Yes	Enterprise	150	Yes	-	Kinda	Yes	Yes	Test it hard

Adaptive grants for rowstore are rough right now:

- No proactive warning when it's happening
- Minimal instrumentation
- Monitoring tools don't cover it
- Resets constantly on index rebuilds, stats updates, failovers, restarts

Yet they're on by default in compat 150.

3.4 p64

Because I know you're gonna ask

```
ALTER DATABASE SCOPED CONFIGURATION
SET BATCH_MODE_MEMORY_GRANT_FEEDBACK = OFF;

ALTER DATABASE SCOPED CONFIGURATION
SET ROW_MODE_MEMORY_GRANT_FEEDBACK = OFF;
```

Adaptive join example

You need to join between two tables.
(Not 2 indexes: this doesn't fix key lookups.)

When the driver table returns just a few rows,
then it makes sense to do nested loops:
one operation per row that it finds.

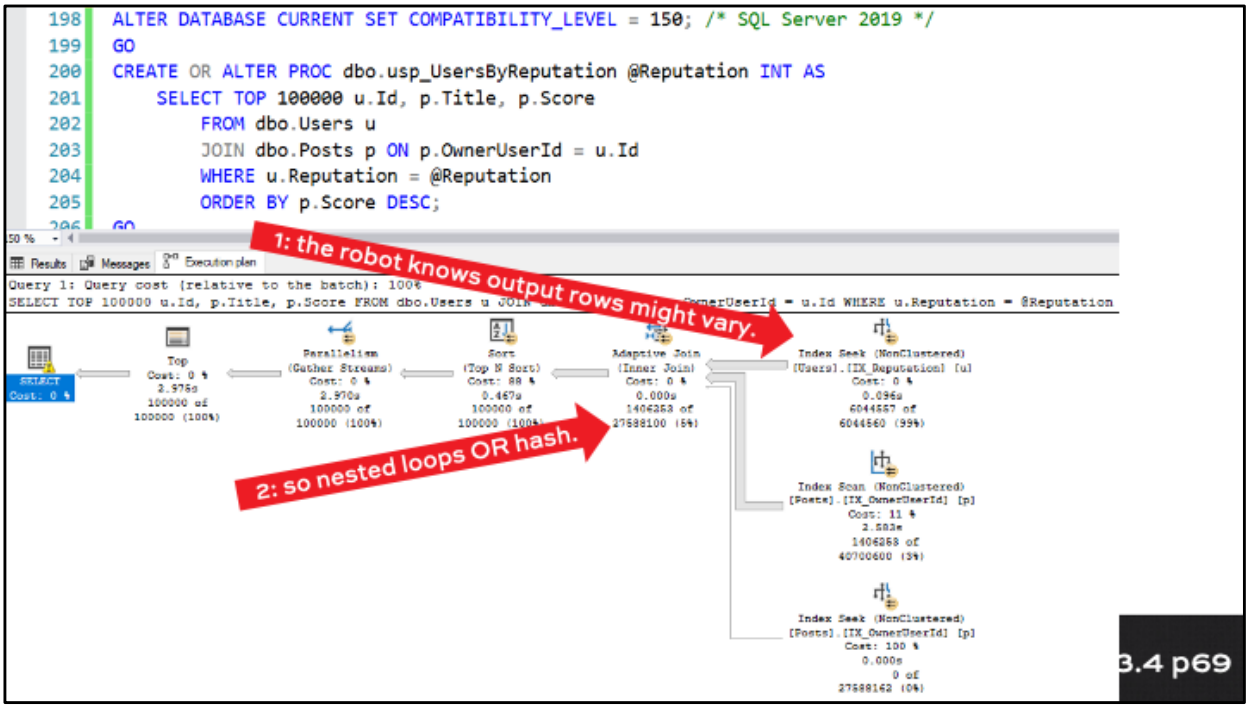
When the driver table returns a lot of rows,
then it makes sense to do a hash join:
scan the other table and check all the rows.

3.4 p67

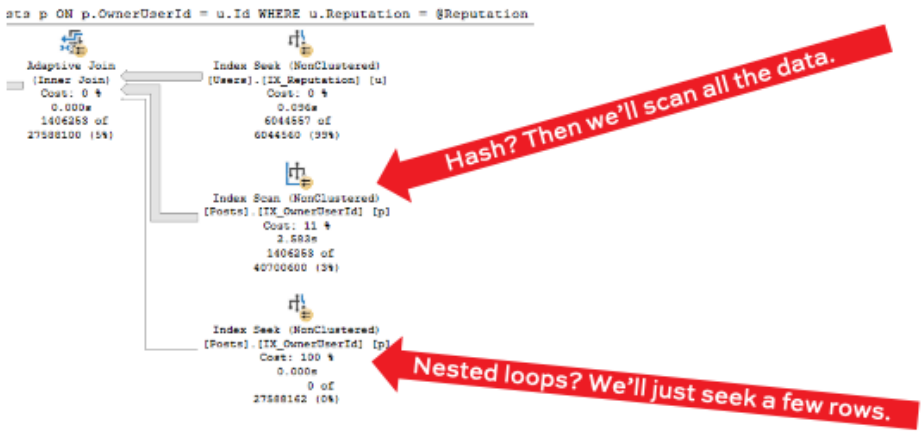
Sample query

```
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 150; /* SQL Server 2019 */
GO
CREATE OR ALTER PROC dbo.usp_UsersByReputation @Reputation INT AS
    SELECT TOP 100000 u.Id, p.Title, p.Score
        FROM dbo.Users u
        JOIN dbo.Posts p ON p.OwnerUserId = u.Id
        WHERE u.Reputation = @Reputation
        ORDER BY p.Score DESC;
GO

/* And run it: */
EXEC usp_UsersByReputation @Reputation = 1;
GO
```

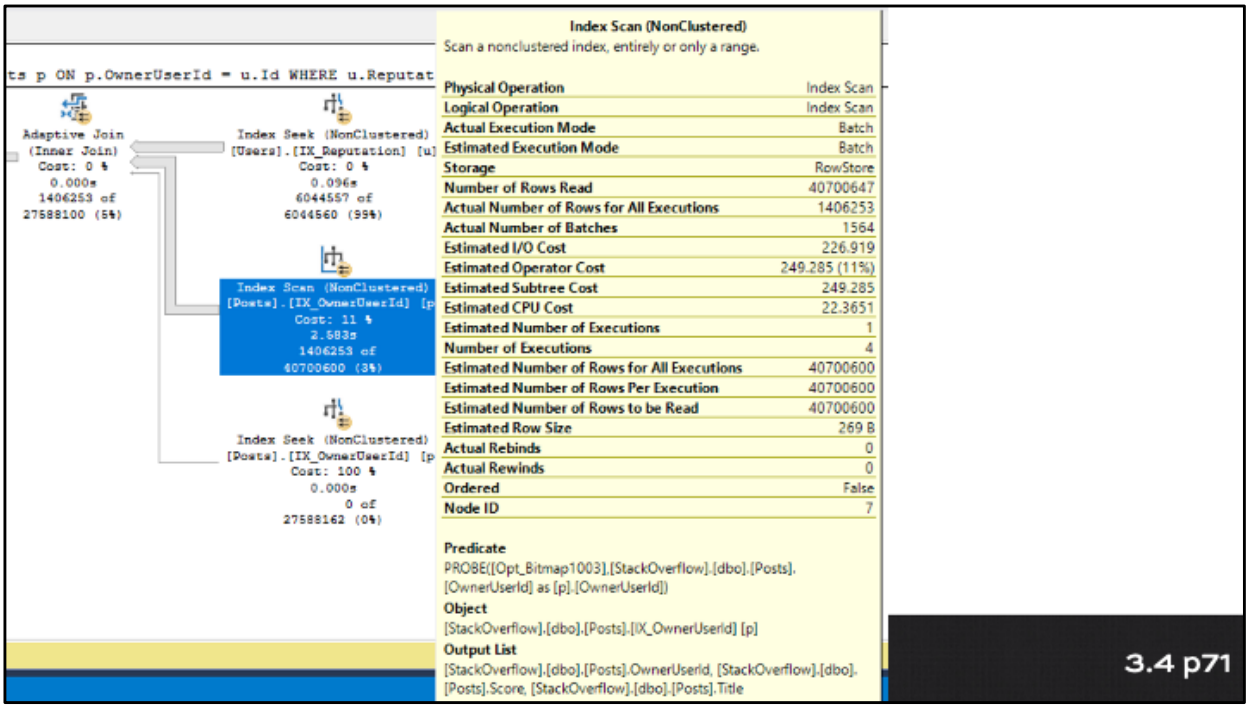
3.4 p68

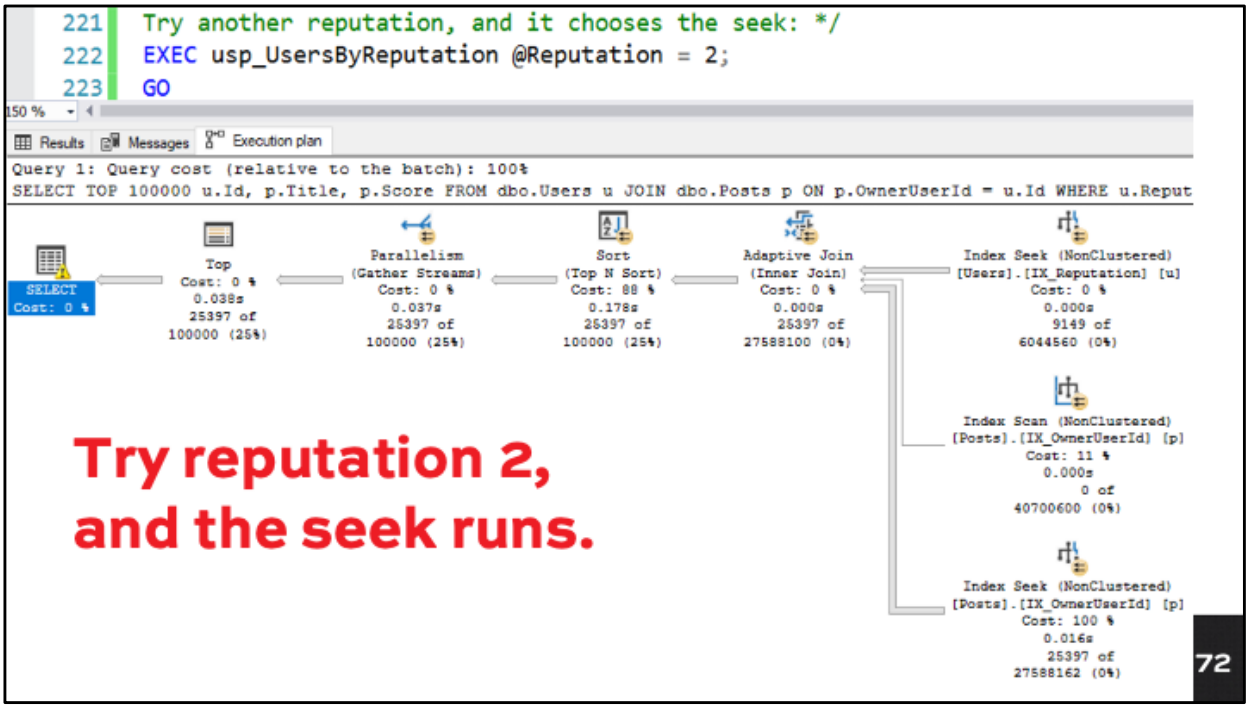


Both operations are shown, but only one will be executed.



3.4 p70





Adaptive joins are rarely seen.

In 2017: requires a columnstore index, compat 140

In 2019: works with rowstore indexes, compat 150

Requires batch mode execution (which in 2017 means a columnstore index somewhere in your query, anywhere, even an empty one)

SELECT queries only (no DUIs)

A join that can work with nested loops or hash

3.4 p73

	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Adaptive joins										
For plans with columnstore indexes	Yes	Yes	Enterprise	140	Yes	-	Yes	Yes	-	OK
For all plans	-	Yes	Enterprise	150		-	Yes	Yes	-	OK

Adaptive Joins are OK.

They're really hard to get today, and by themselves, they don't usually get you across the finish line with no other query changes.

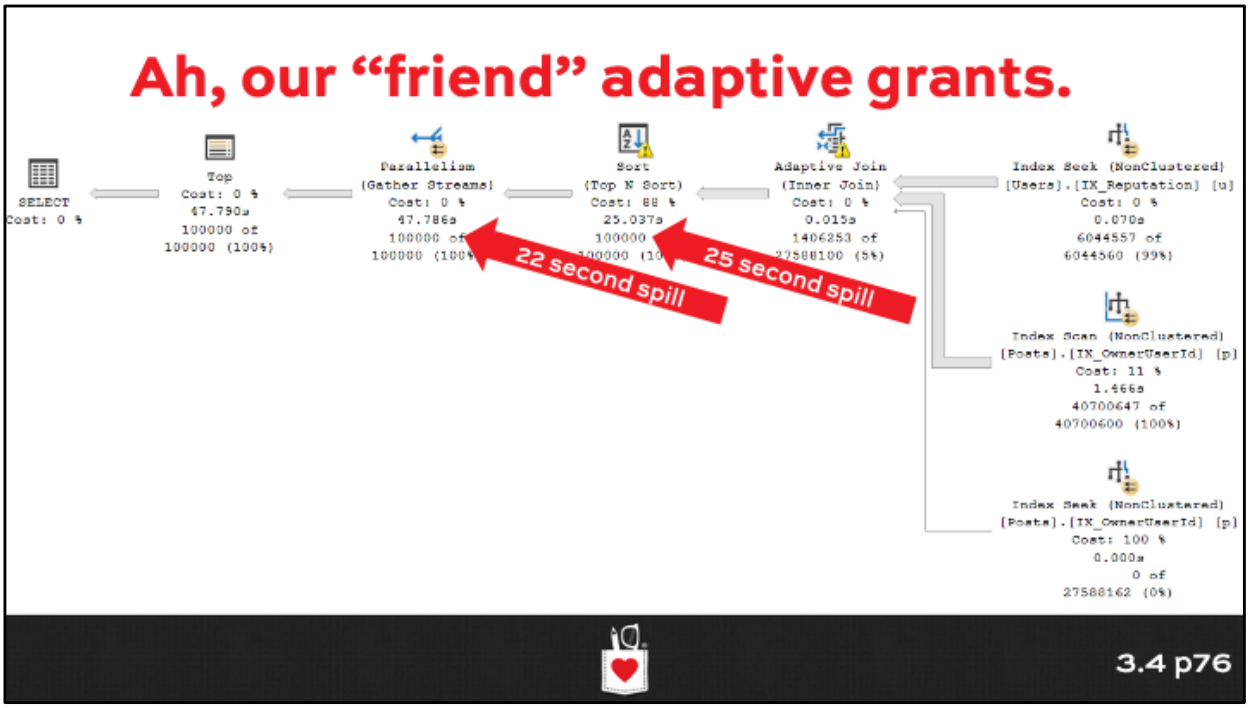
But they're OK.

3.4 p74

But run them back to back...

Action	Duration
DBCC FREEPROCCACHE	
Run it for @Reputation = 1	1 second
Run it for @Reputation = 2	0 seconds
Run it for @Reputation = 1 again	**1 minute 15 seconds**
Why do you suppose that is?	

3.4 p75



Parameter sniffing is still hard.

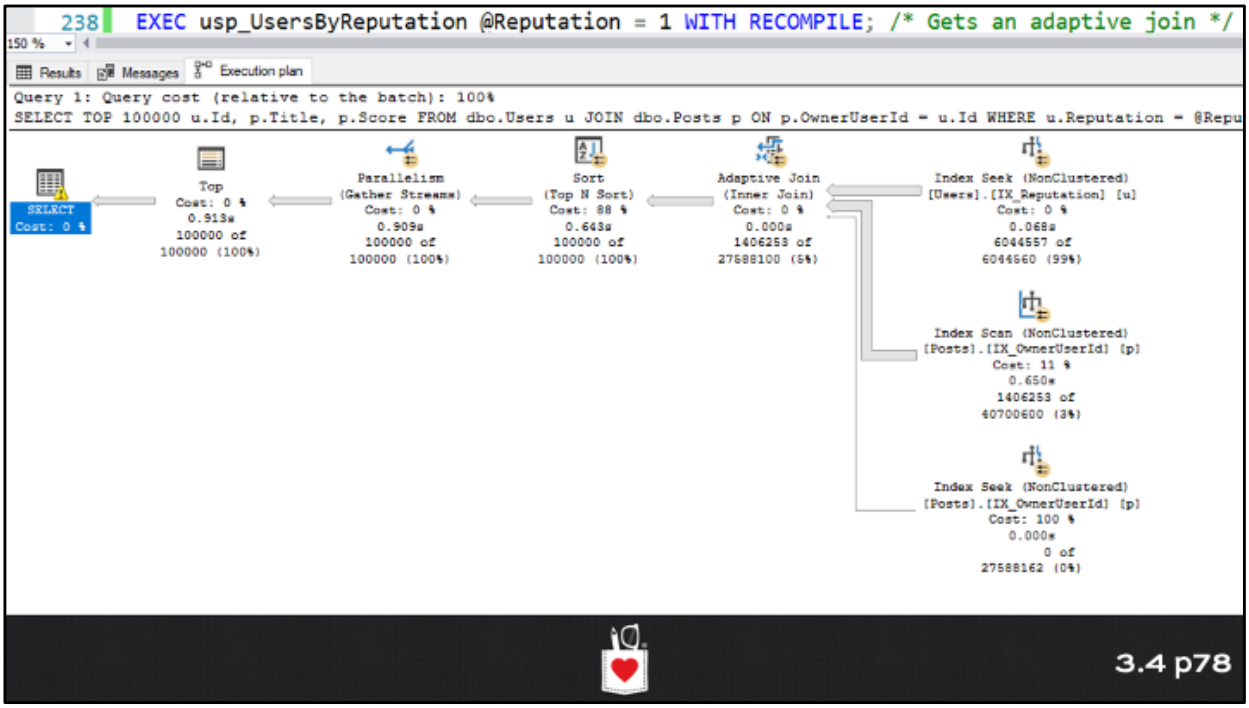
One new feature (adaptive grants)
can wreak havoc on other new features.

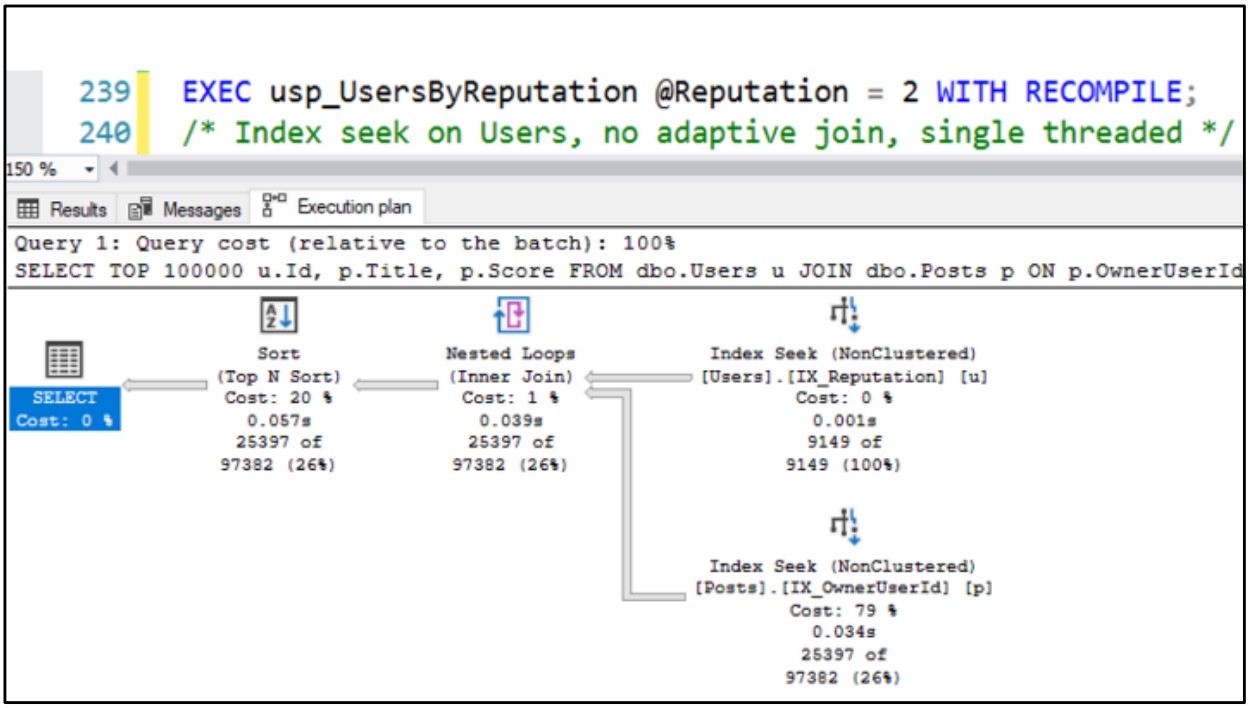
This is called “feature interoperability.”

You’re lucky if new features are tested in isolation,
let alone combined with other features.

Plus, we still have good ol’ parameter sniffing:
adaptive joins may not show up for all params.

3.4 p77





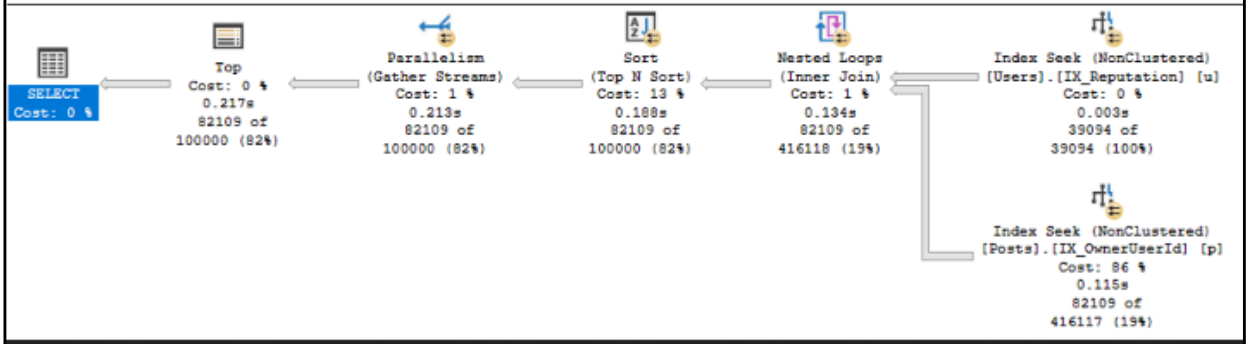
And there's yet another plan:

```
241  EXEC usp_UsersByReputation @Reputation = 4 WITH RECOMPILE;
242  /* Index seek on Users, no adaptive join, parallel */
```

150 %

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 100000 u.Id, p.Title, p.Score FROM dbo.Users u JOIN dbo.Posts p ON p.OwnerUserId = u.Id WHERE u.Reput



3.4 p80

Parameter sniffing is HARD now.

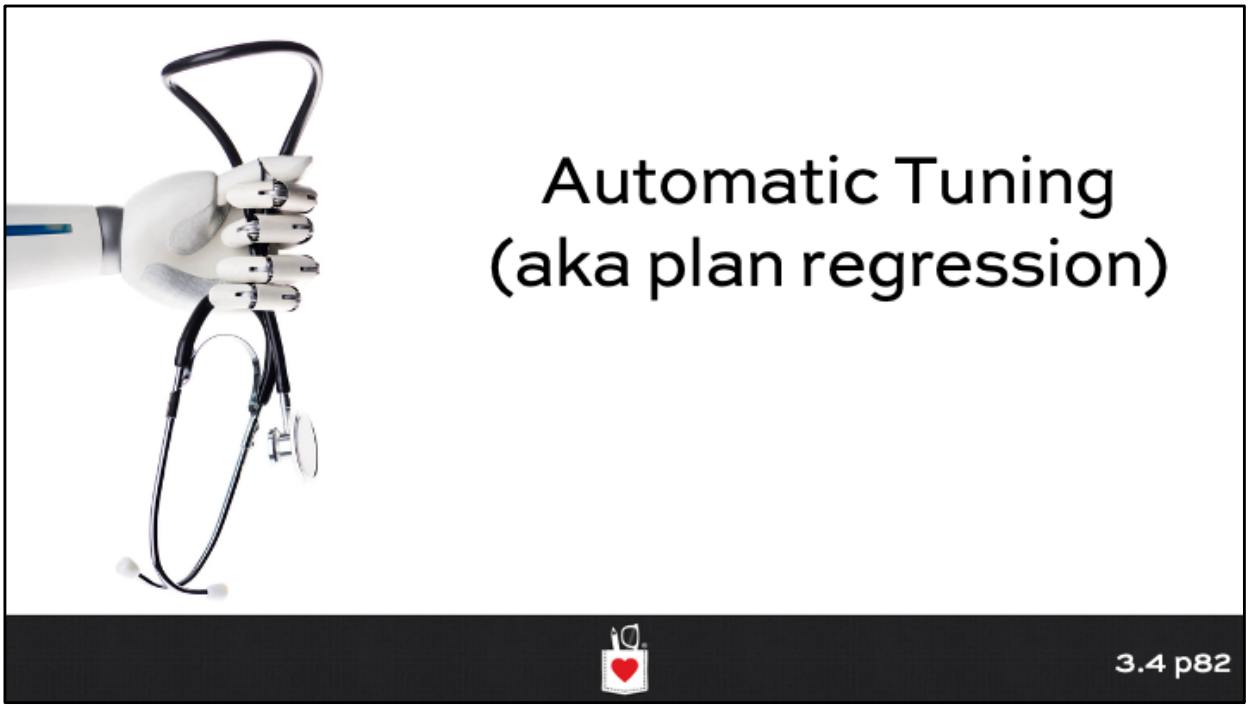
This is a very simple stored procedure, but:

Depending on which @Reputation it's called with first (1, 2, or 4), I can get 3 different plans in cache.

And then every single time it runs, it can set different memory grants for the very next execution.

Plan changes are a huge problem in 2019.

3.4 p81



This feature name is a flat out lie.

Automatic Tuning implies that SQL Server is actively doing something to make your query better.

It's not even close.

The sales pitch

Turn on Query Store (which is a story to begin with.)

Get a good query plan.

Have something go wrong: stats update, data change.

Get a bad query plan.

After a while, SQL Server will realize it's a bad plan, and try to revert to the good plan that was stored in Query Store. If it's faster, keep it permanently.

3.4 p84

Scenarios it doesn't help

If you never get a good plan in the first place

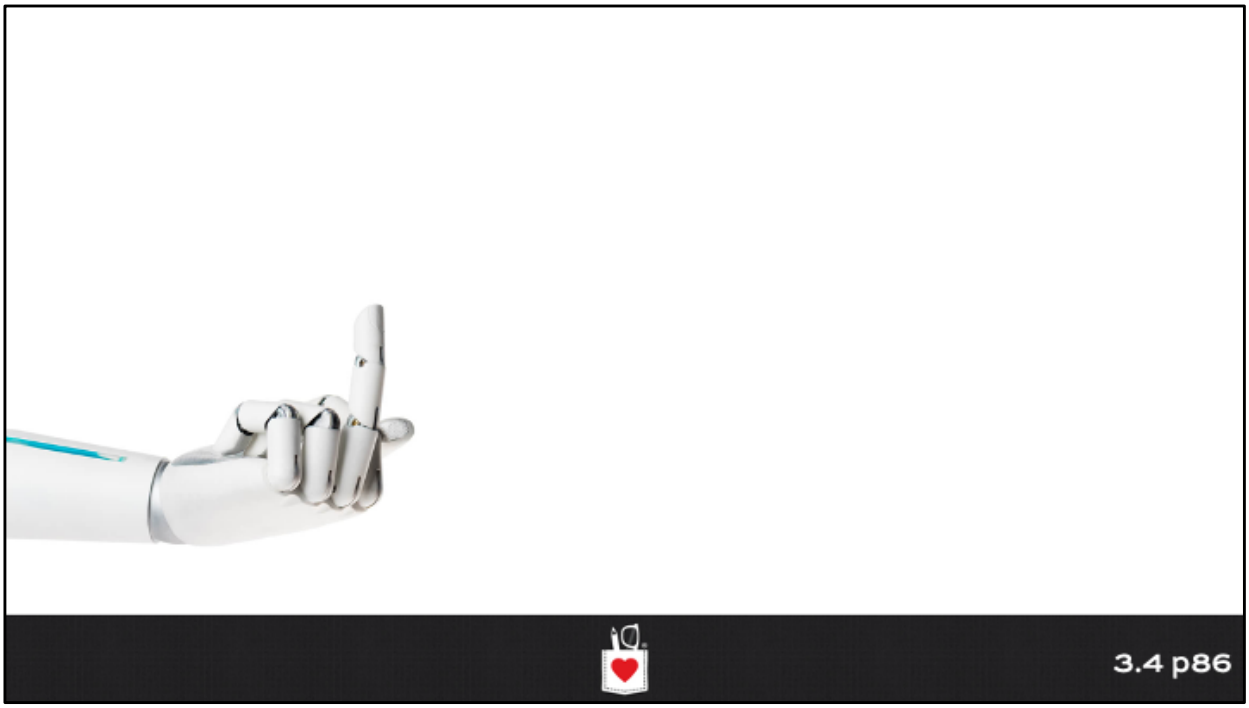
If you didn't have Query Store already enabled when the good plan was in effect

If the query changes (even by one space, like if you do a deployment and just changed a comment)

If there's no one good plan for all parameters (common)

This is not Automatic Tuning
in any way, shape, or form.

3.4 p85



Anyway, Microsoft is off to a good start.

3.4 p87

	SQL Server 2017	SQL Server 2019	Minimum Edition	Minimum Compat Level	Works in Azure SQL DB	T-SQL Changes Required	Can be seen in plan	Can make things better	Can make things much worse	Brent's Verdict
Automatic tuning										
Query Store automatic plan correction	Yes	Yes	Enterprise	140	Yes	-	-	Yes	-	OK
Automatic index management	-	-	-	140	Yes	-	-	Yes	-	OK
Batch mode execution										
For columnstore indexes	Yes	Yes	Standard	140	Yes	-	Kinda	Yes	-	Great!
For rowstore indexes	-	Yes	Enterprise	150	Yes	-	Kinda	Yes	-	Great!
Functions										
TVF interleaved execution	Yes	Yes	Standard	140	Yes	-	Yes	Yes	-	OK
Scalar function inlining	-	Yes	Standard	150	-	-	Yes	Yes	Yes	Test it hard
Memory grant feedback (adaptive grants)	-									
For columnstore indexes	Yes	Yes	Enterprise	140	Yes	-	Kinda	Yes	-	OK
For rowstore indexes	-	Yes	Enterprise	150	Yes	-	Kinda	Yes	Yes	Test it hard
Adaptive joins										
For plans with columnstore indexes	Yes	Yes	Enterprise	140	Yes	-	Yes	Yes	-	OK
For all plans	-	Yes	Enterprise	150		-	Yes	Yes	-	OK
Others										
Approximate Count Distinct	-	Yes	Standard	150	Yes	Yes	Yes	Yes	-	OK
Table variable deferred compilation	-	Yes	Standard	150	Yes	-	Yes	Yes	-	OK

3.4 p88

I have high hopes. Done right:

This can reduce Microsoft's hardware spend in Azure.

They might cut hosting costs to be more competitive with Amazon RDS Aurora MySQL & PostgreSQL.

This can give Microsoft a very competitive advantage over other on-premises databases, too.

Done right, there's really no downside (other than Microsoft's costs to build it.)

3.4 p89