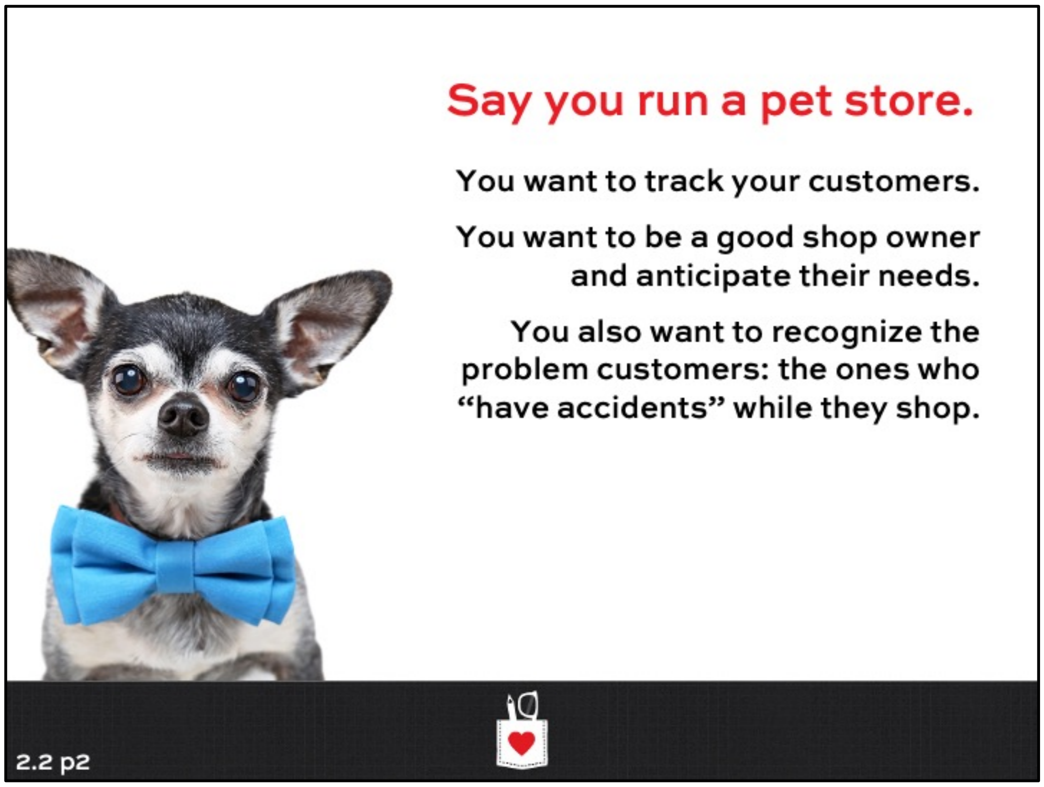
BRENT OZAR
UNLIMITED®

How to Fix Plan Caching Problems

2.2 p1



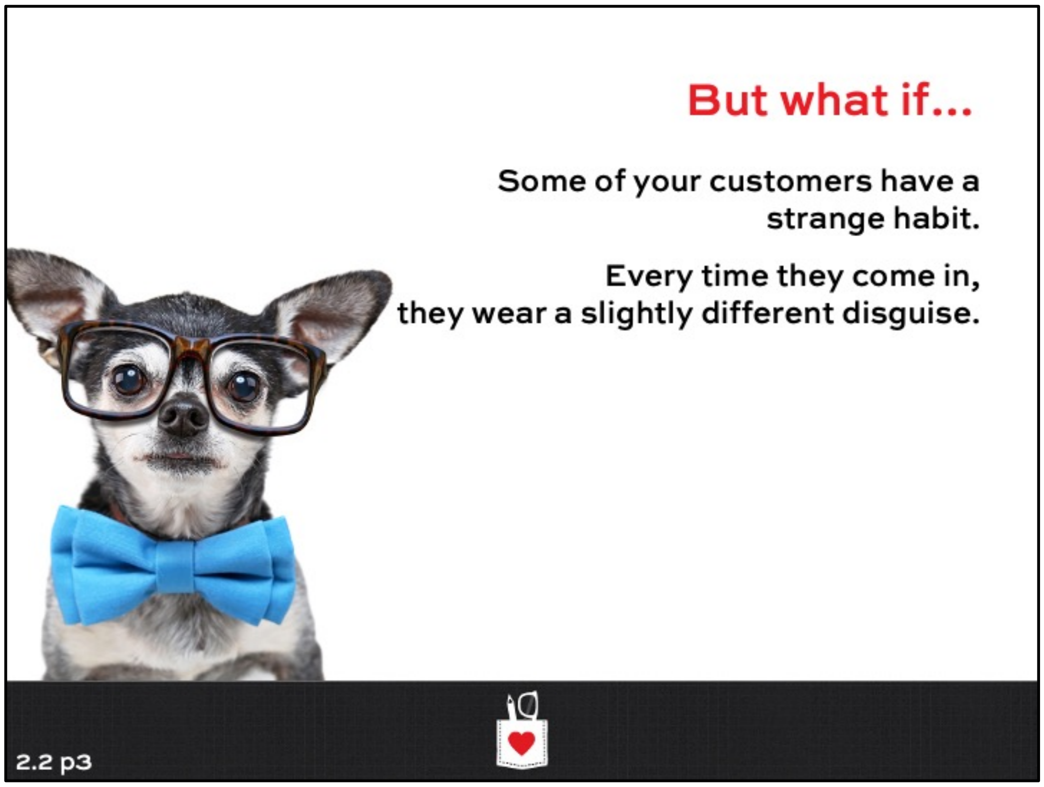
Say you run a pet store.

You want to track your customers.

You want to be a good shop owner and anticipate their needs.

You also want to recognize the problem customers: the ones who "have accidents" while they shop.

2.2 p2



But what if…

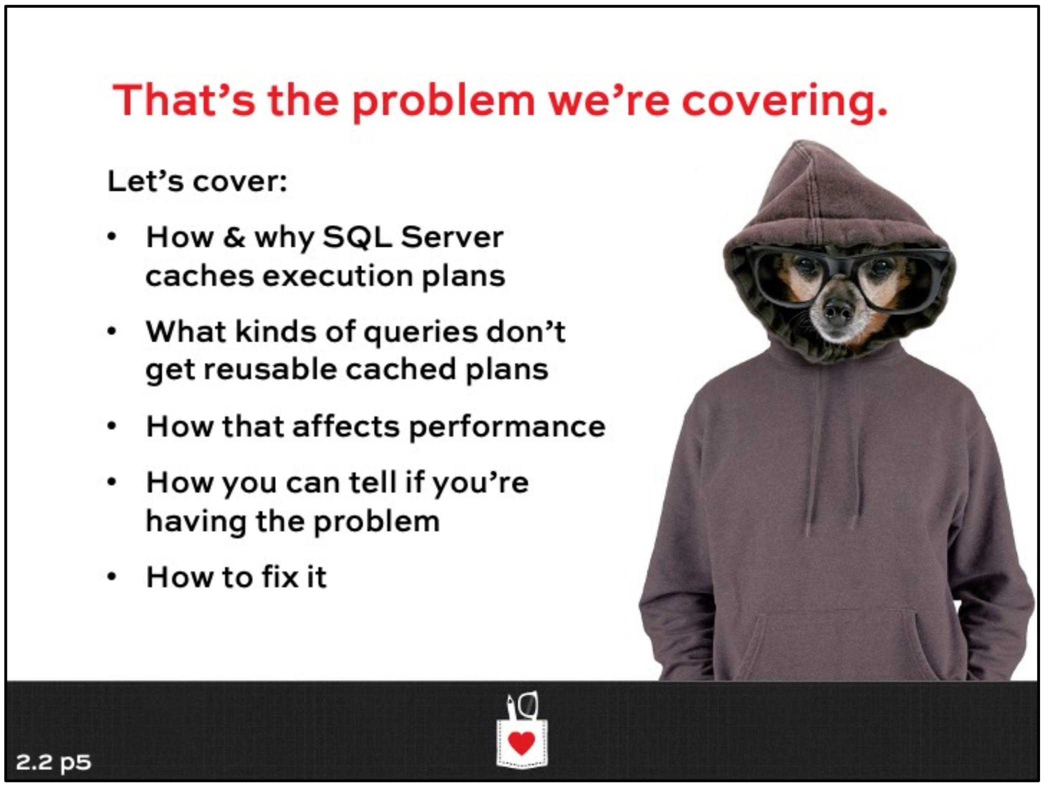
Some of your customers have a strange habit.

Every time they come in, they wear a slightly different disguise.

2.2 p3

“No, I’m not Bowser.
Totally different guy.
I never pee on the floor.”

2.2 p4

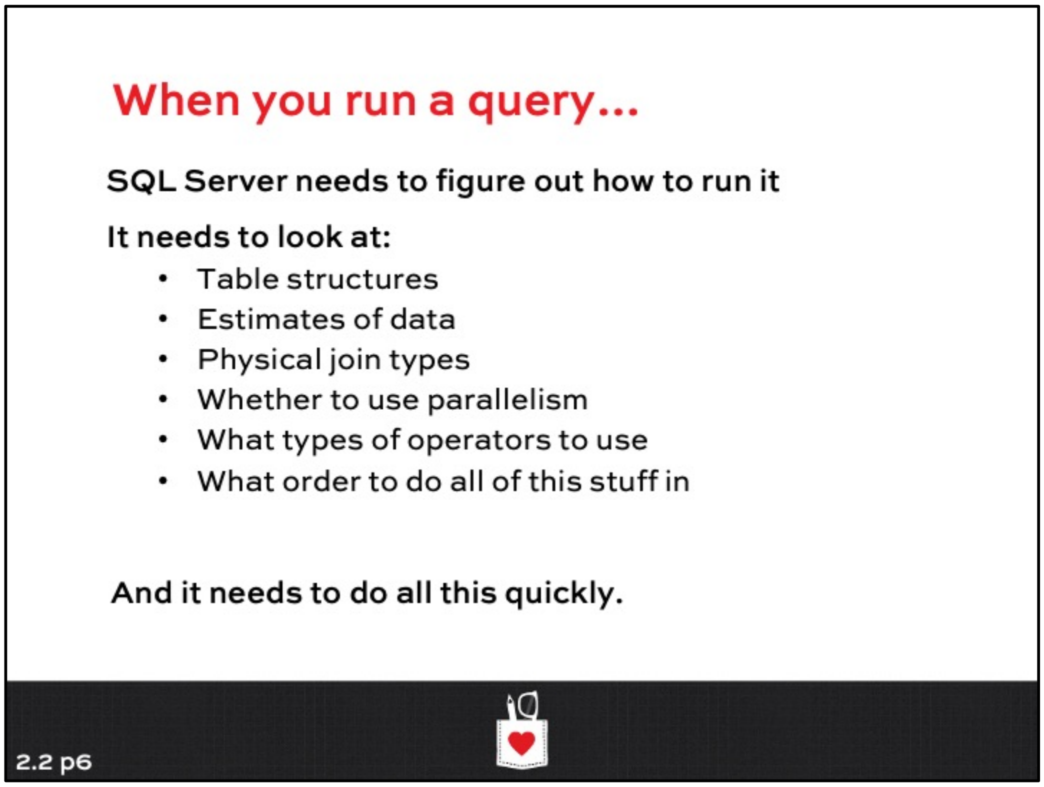


That's the problem we're covering.

Let's cover:

- How & why SQL Server caches execution plans
- What kinds of queries don't get reusable cached plans
- How that affects performance
- How you can tell if you're having the problem
- How to fix it

2.2 p5



When you run a query...

SQL Server needs to figure out how to run it

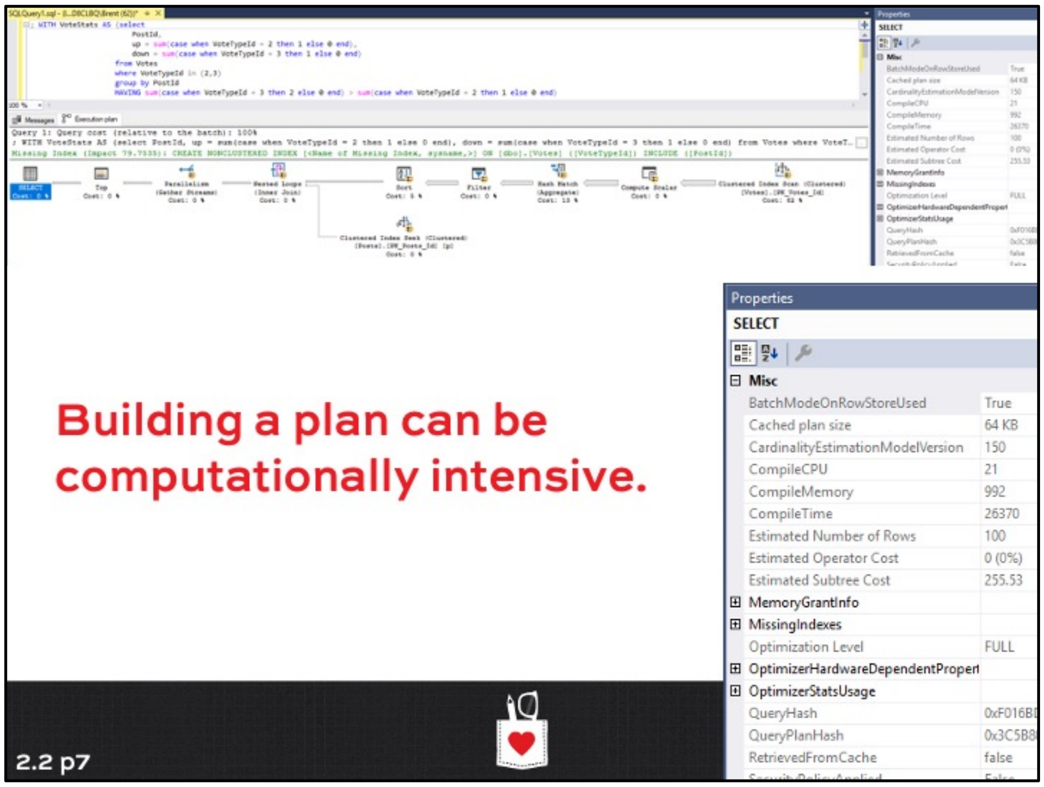
It needs to look at:

- Table structures
- Estimates of data
- Physical join types
- Whether to use parallelism
- What types of operators to use
- What order to do all of this stuff in

And it needs to do all this quickly.

2.2 p6

Lots of things need to be considered



```
; WITH VoteStats AS (select
            PostId,
            up = sum(case when VoteTypeId = 2 then 1 else 0 end),
            down = sum(case when VoteTypeId = 3 then 1 else 0 end)
        from Votes
        where VoteTypeId in (2,3)
        group by PostId
        HAVING sum(case when VoteTypeId = 3 then 2 else 0 end) > sum(case when VoteTypeId = 2 then 1 else 0 end)
```

```
Query 1: Query cost (relative to the batch): 100%
; WITH VoteStats AS (select PostId, up = sum(case when VoteTypeId = 2 then 1 else 0 end), down = sum(case when VoteTypeId = 3 then 1 else 0 end) from Votes where VoteT...
Missing Index (Impact 79.7535): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Votes] ([VoteTypeId]) INCLUDE ([PostId])
```

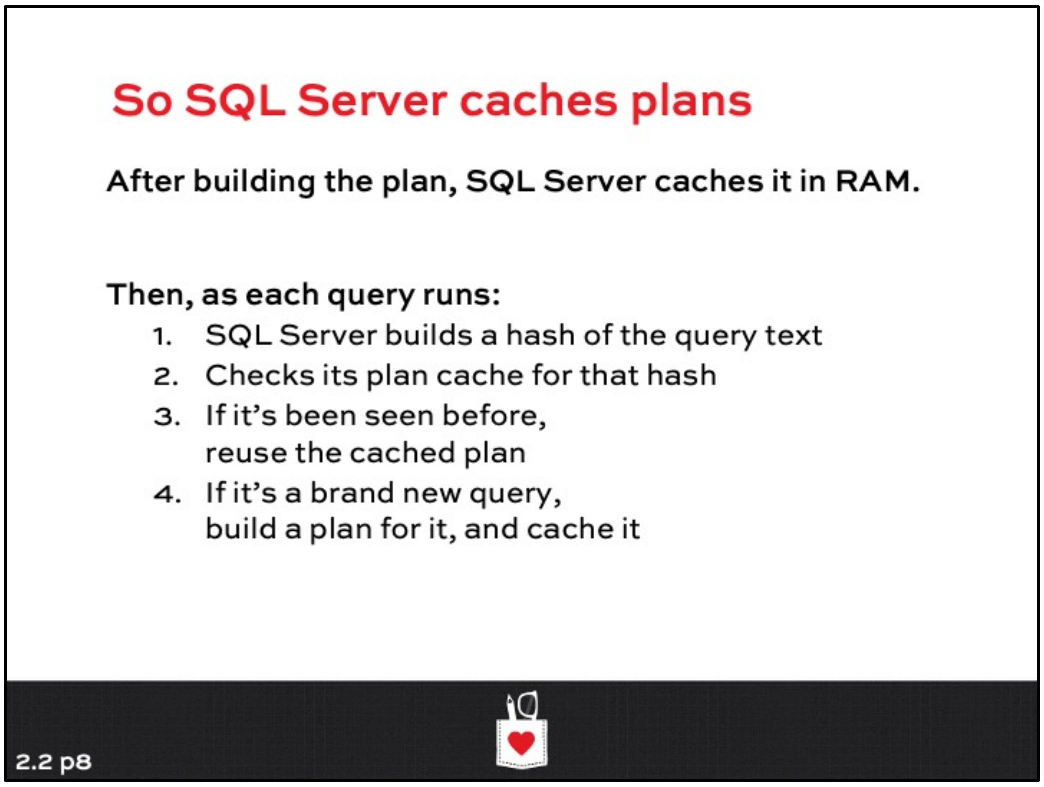
Properties

SELECT

Misc	
BatchModeOnRowStoreUsed	True
Cached plan size	64 KB
CardinalityEstimationModelVersion	150
CompileCPU	21
CompileMemory	992
CompileTime	26370
Estimated Number of Rows	100
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	255.53
MemoryGrantInfo	
MissingIndexes	
Optimization Level	FULL
OptimizerHardwareDependentPropert	
OptimizerStatsUsage	
QueryHash	0xF016BI
QueryPlanHash	0x3C5B8
RetrievedFromCache	false

Building a plan can be computationally intensive.

2.2 p7



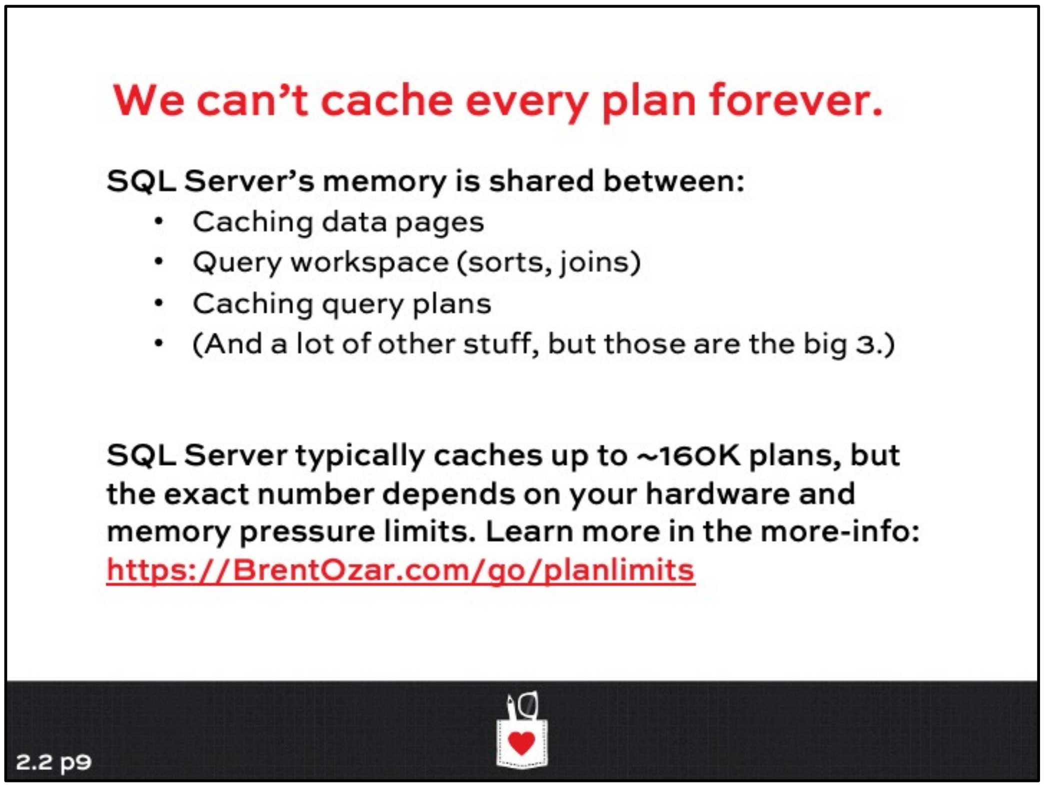
So SQL Server caches plans

After building the plan, SQL Server caches it in RAM.

Then, as each query runs:

1. SQL Server builds a hash of the query text
2. Checks its plan cache for that hash
3. If it's been seen before,
   reuse the cached plan
4. If it's a brand new query,
   build a plan for it, and cache it

2.2 p8



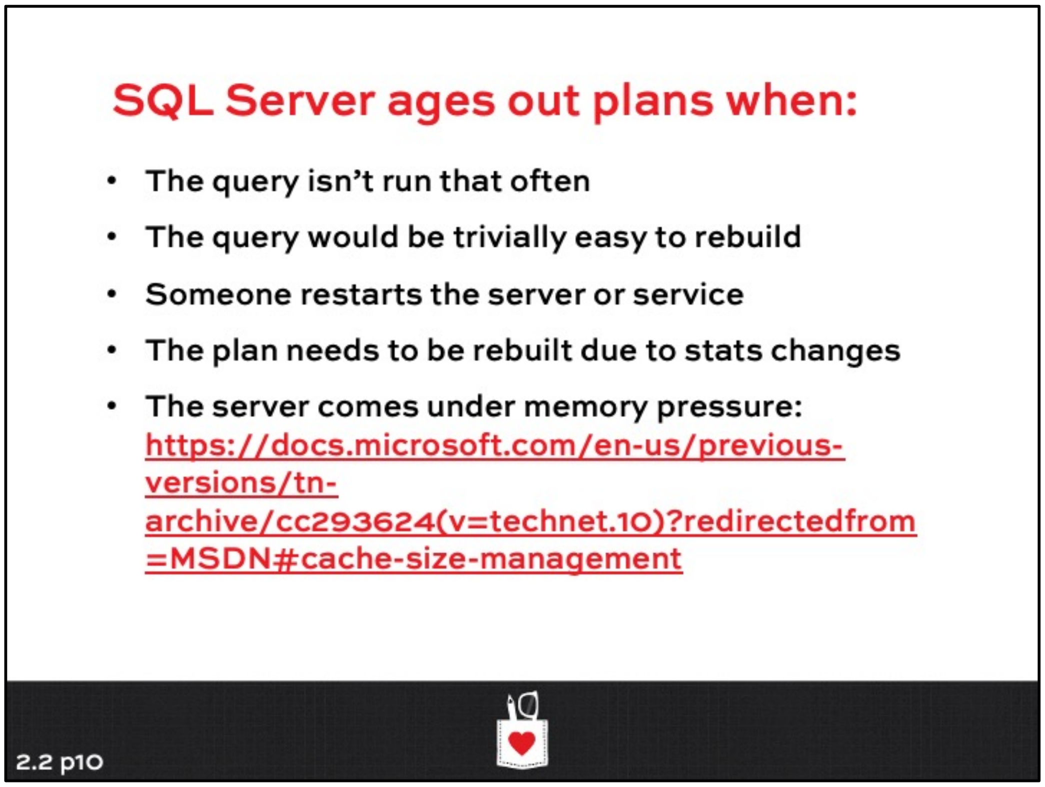
We can't cache every plan forever.

SQL Server's memory is shared between:

- Caching data pages
- Query workspace (sorts, joins)
- Caching query plans
- (And a lot of other stuff, but those are the big 3.)

SQL Server typically caches up to ~160K plans, but the exact number depends on your hardware and memory pressure limits. Learn more in the more-info: https://BrentOzar.com/go/planlimits

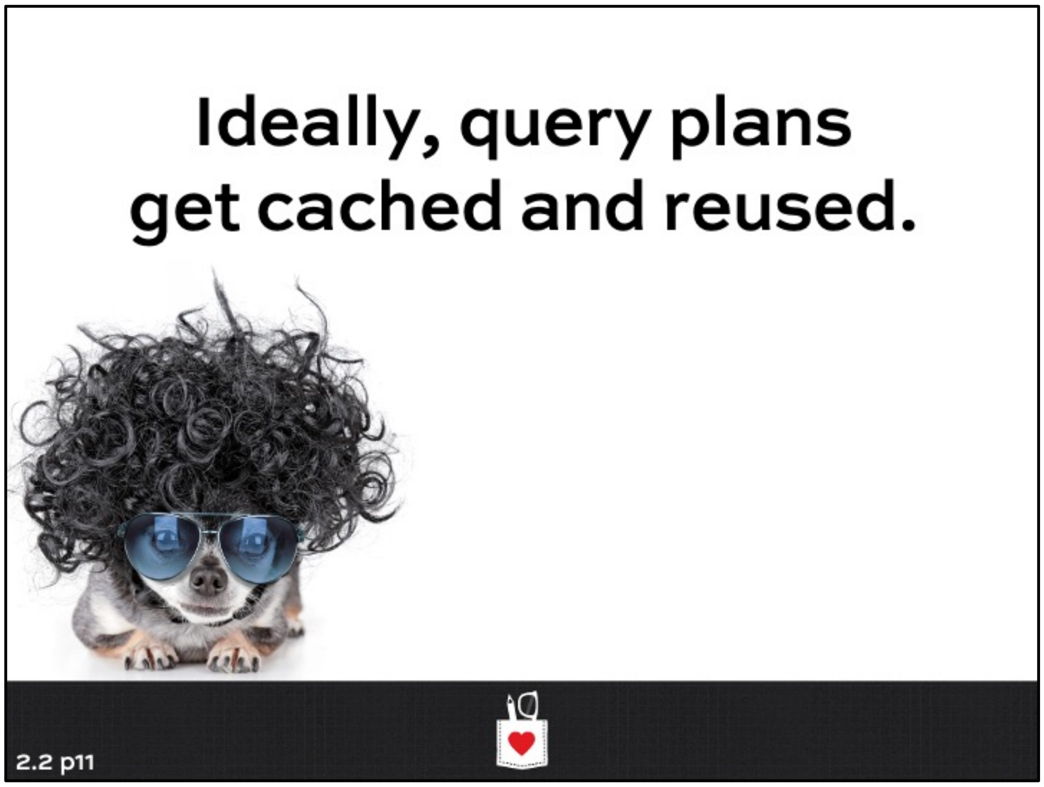
2.2 p9



SQL Server ages out plans when:

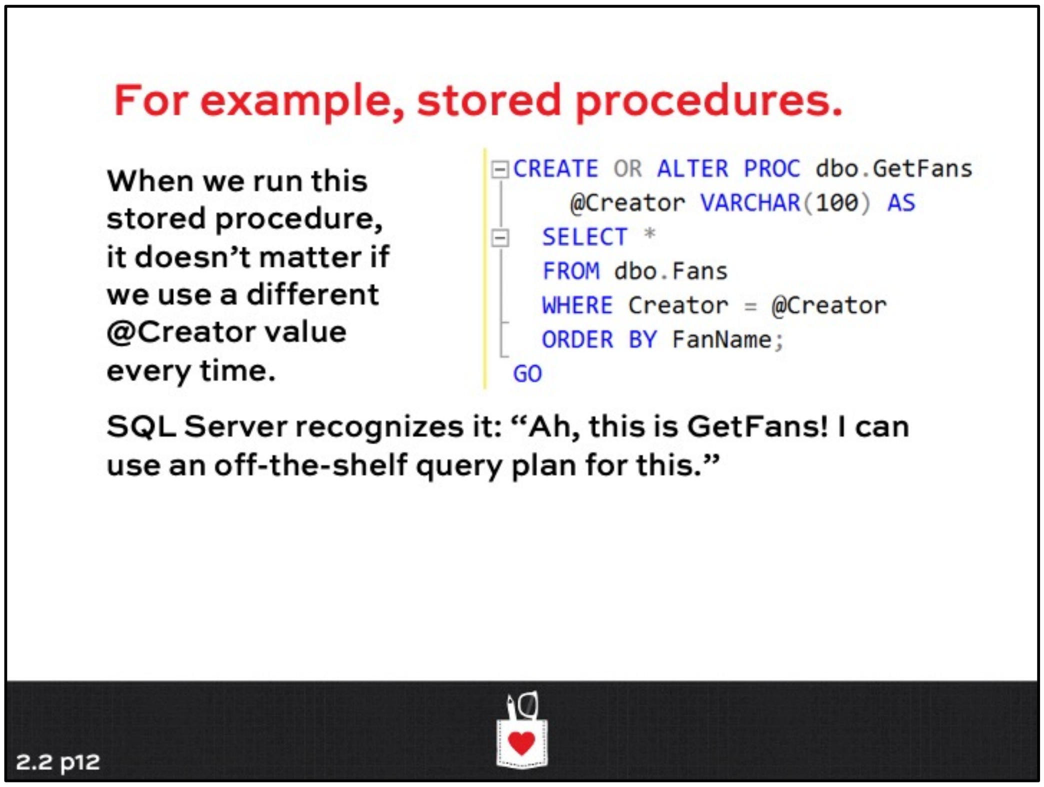
- The query isn't run that often
- The query would be trivially easy to rebuild
- Someone restarts the server or service
- The plan needs to be rebuilt due to stats changes
- The server comes under memory pressure: https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc293624(v=technet.10)?redirectedfrom=MSDN#cache-size-management

2.2 p10



Ideally, query plans get cached and reused.

2.2 p11



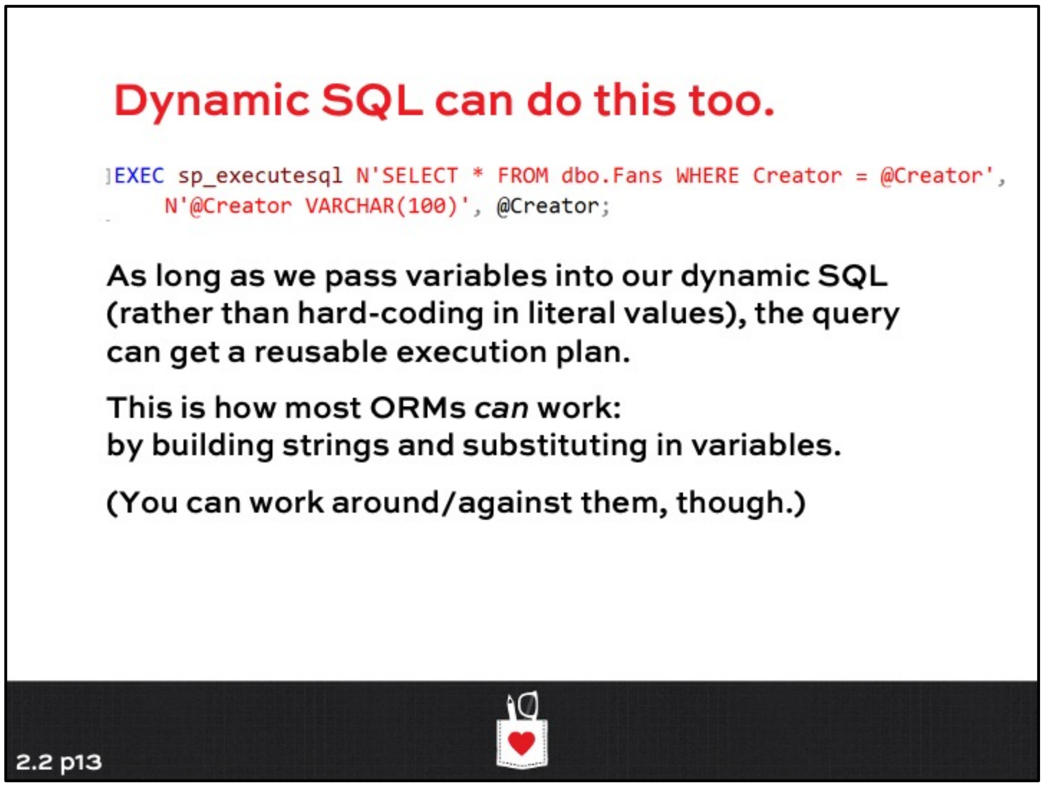
For example, stored procedures.

When we run this stored procedure, it doesn't matter if we use a different @Creator value every time.

```
CREATE OR ALTER PROC dbo.GetFans
    @Creator VARCHAR(100) AS
  SELECT *
  FROM dbo.Fans
  WHERE Creator = @Creator
  ORDER BY FanName;
GO
```

SQL Server recognizes it: "Ah, this is GetFans! I can use an off-the-shelf query plan for this."

2.2 p12



Dynamic SQL can do this too.

```
EXEC sp_executesql N'SELECT * FROM dbo.Fans WHERE Creator = @Creator',
    N'@Creator VARCHAR(100)', @Creator;
```

As long as we pass variables into our dynamic SQL (rather than hard-coding in literal values), the query can get a reusable execution plan.

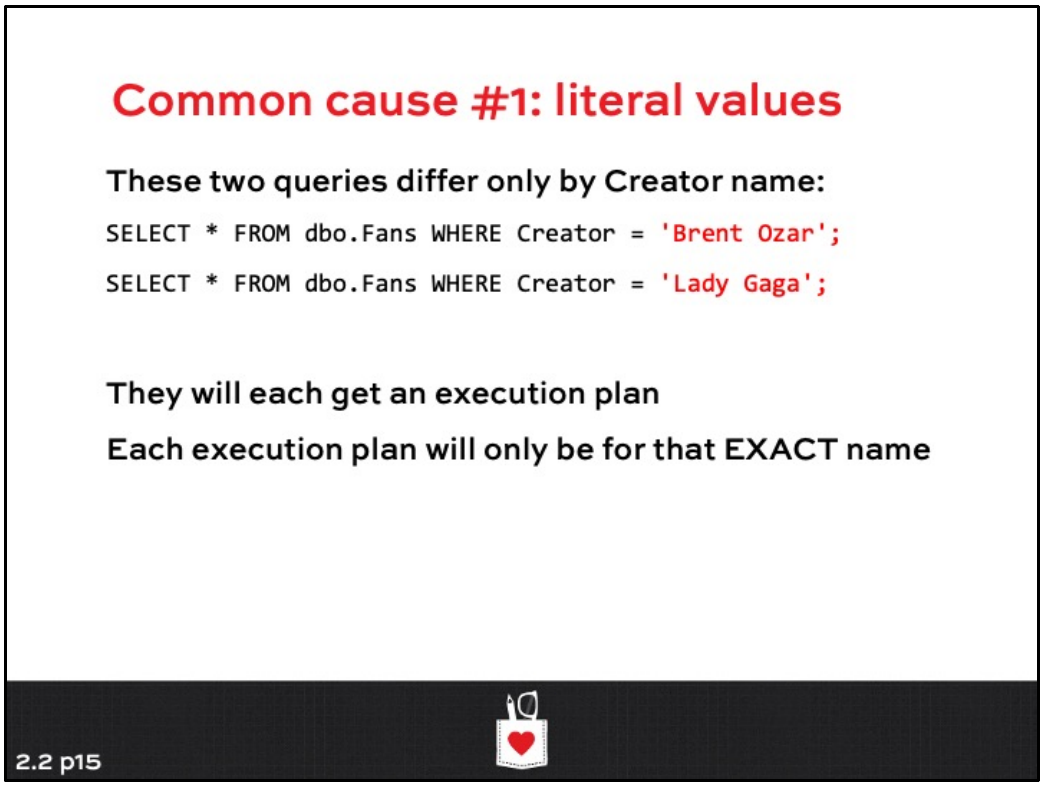
This is how most ORMs *can* work:
by building strings and substituting in variables.

(You can work around/against them, though.)

2.2 p13

Some queries don't get reusable plans.

2.2 p14


Common cause #1: literal values

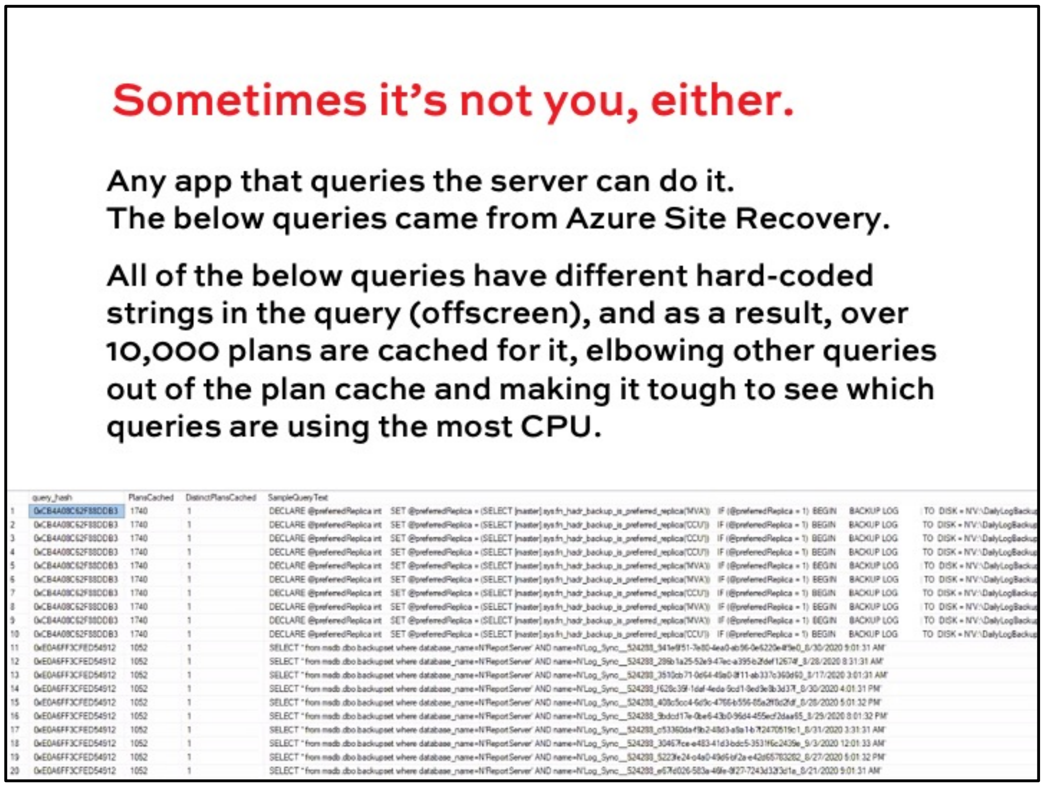
These two queries differ only by Creator name:

```
SELECT * FROM dbo.Fans WHERE Creator = 'Brent Ozar';
SELECT * FROM dbo.Fans WHERE Creator = 'Lady Gaga';
```

They will each get an execution plan

Each execution plan will only be for that EXACT name

2.2 p15



Sometimes it's not you, either.

Any app that queries the server can do it.
The below queries came from Azure Site Recovery.

All of the below queries have different hard-coded strings in the query (offscreen), and as a result, over 10,000 plans are cached for it, elbowing other queries out of the plan cache and making it tough to see which queries are using the most CPU.

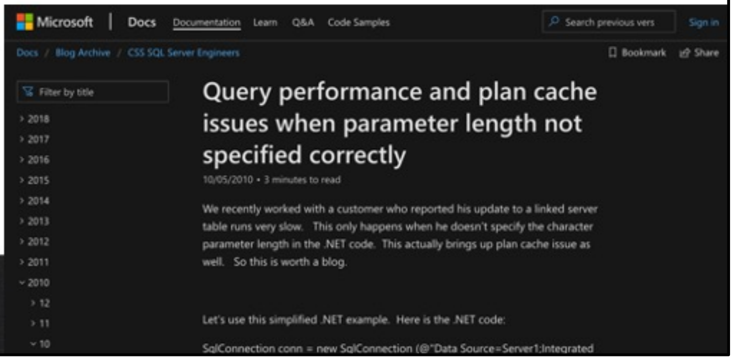
	query_hash	PlansCached	DistinctPlansCached	SampleQueryText
1	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('MVA')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
2	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('CCU')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
3	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('CCU')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
4	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('CCU')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
5	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('MVA')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
6	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('MVA')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
7	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('CCU')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
8	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('MVA')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
9	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('MVA')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
10	0xCB4A08C62F88DDB3	1740	1	DECLARE @preferredReplica int SET @preferredReplica = (SELECT [master].sys.fn_hadr_backup_is_preferred_replica('CCU')) IF (@preferredReplica = 1) BEGIN BACKUP LOG TO DISK = N'V:\DailyLogBackup
11	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_941e8f51-7e80-4ea0-ab96-0e6220e4f9e0_8/30/2020 9:01:31 AM'
12	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_286b1a25-52e9-47ec-a395-b2fdef12674f_8/28/2020 8:31:31 AM'
13	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_3510cb71-0d64-49a0-8f11-ab337c360d60_8/17/2020 3:01:31 AM'
14	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_f626c35f-1daf-4eda-9cd1-0ed3e8b3d37f_8/30/2020 4:01:31 PM'
15	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_408c5cc4-6d9c-4766-b556-85a2ff0d2fdf_8/28/2020 5:01:32 PM'
16	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_9bdcd17e-0be6-43b0-96d4-455ecf2daa65_8/29/2020 8:01:32 PM'
17	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_c53360da-f9b2-48d3-a9a1-b7f2470519c1_8/31/2020 3:31:31 AM'
18	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_30467fce-e483-41d3-bdc5-3531f6c2439e_9/3/2020 12:01:33 AM'
19	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_5223fe24-c4a0-49d6-bf2a-e42d65783282_8/27/2020 5:01:32 PM'
20	0xE0A6FF3CFED54912	1052	1	SELECT * from msdb.dbo.backupset where database_name=N'ReportServer' AND name=N'Log_Sync__524288_e67fd026-583a-46fe-8f27-7243d32f3d1e_8/21/2020 9:01:31 AM'

Common cause #2: param lengths

When you build a parameterized query in .NET, if you don't specify a parameter length, .NET uses whatever parameter value you're passing in as the length:

https://docs.microsoft.com/en-us/archive/blogs/psssql/query-performance-and-plan-cache-issues-when-parameter-length-not-specified-correctly

(Yep, known issue since 2010.)



Query performance and plan cache issues when parameter length not specified correctly

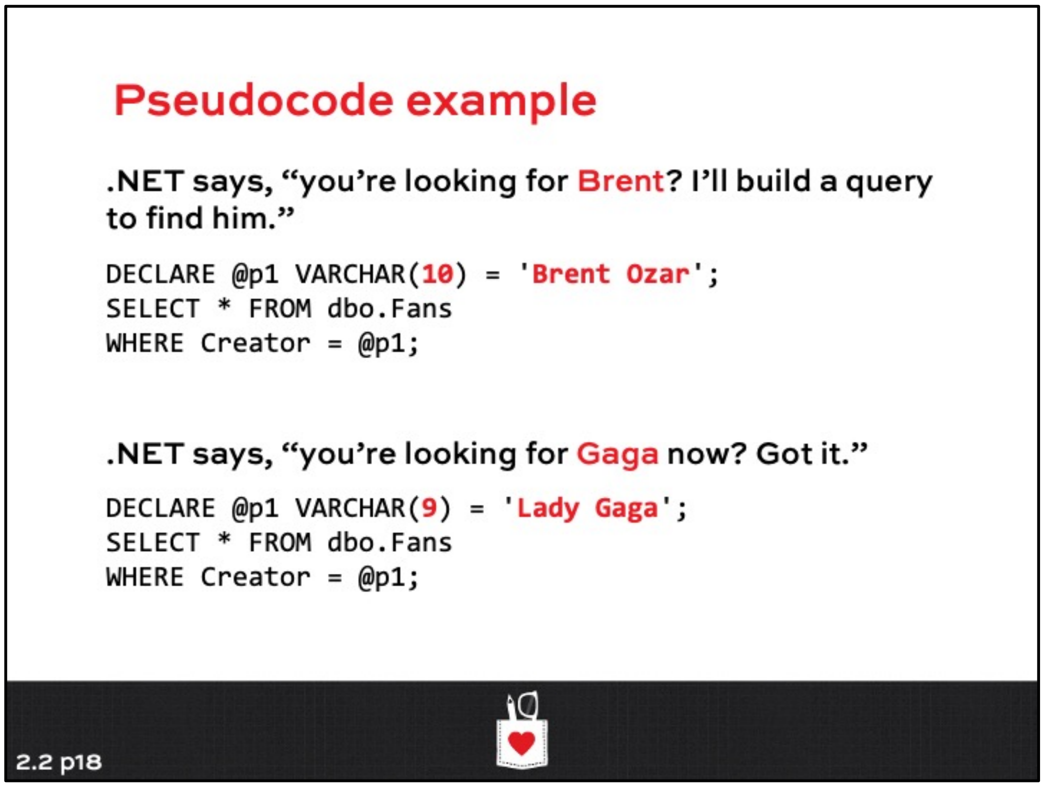
10/05/2010 • 3 minutes to read

We recently worked with a customer who reported his update to a linked server table runs very slow. This only happens when he doesn't specify the character parameter length in the .NET code. This actually brings up plan cache issue as well. So this is worth a blog.

Let's use this simplified .NET example. Here is the .NET code:

SqlConnection conn = new SqlConnection (@"Data Source=Server1;Integrated

2.2 p17



Pseudocode example

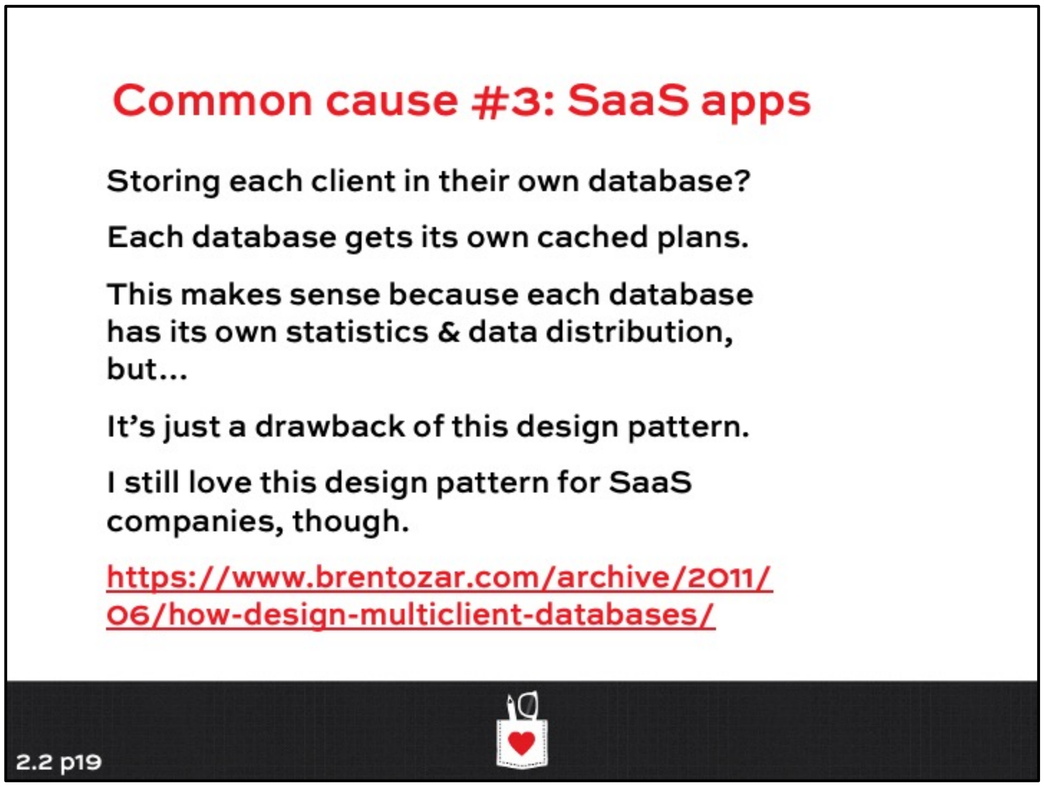
.NET says, "you're looking for Brent? I'll build a query to find him."

```
DECLARE @p1 VARCHAR(10) = 'Brent Ozar';
SELECT * FROM dbo.Fans
WHERE Creator = @p1;
```

.NET says, "you're looking for Gaga now? Got it."

```
DECLARE @p1 VARCHAR(9) = 'Lady Gaga';
SELECT * FROM dbo.Fans
WHERE Creator = @p1;
```

2.2 p18



Common cause #3: SaaS apps

Storing each client in their own database?

Each database gets its own cached plans.

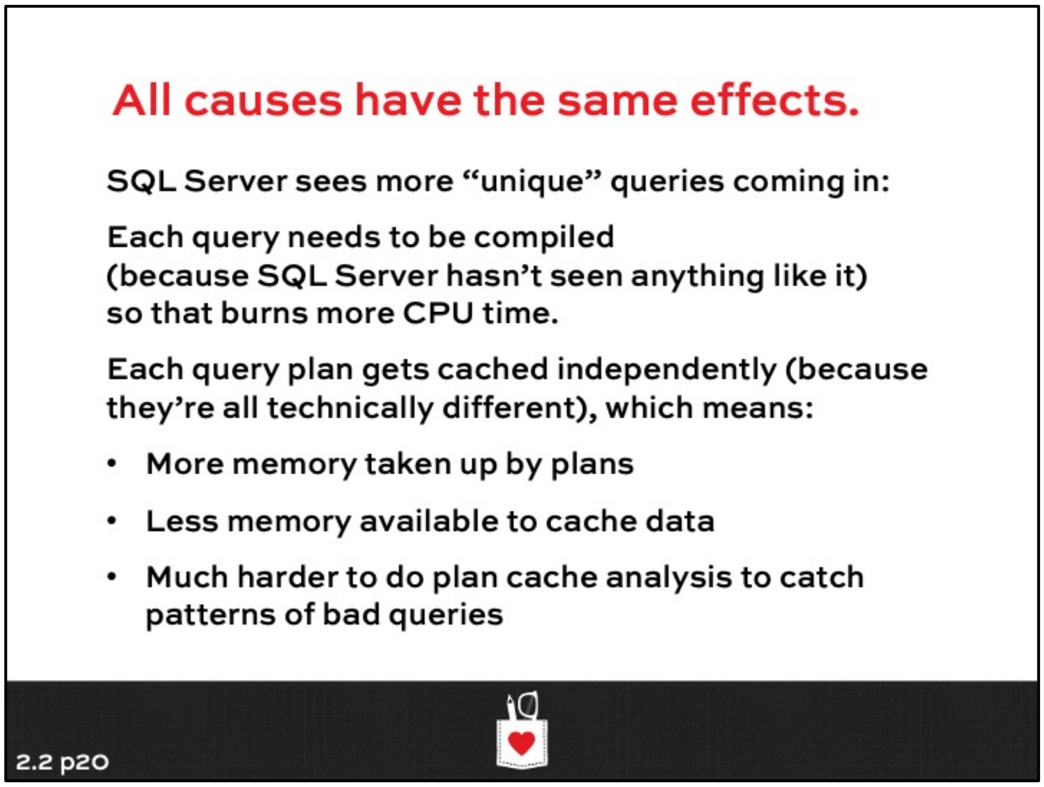
This makes sense because each database has its own statistics & data distribution, but…

It's just a drawback of this design pattern.

I still love this design pattern for SaaS companies, though.

https://www.brentozar.com/archive/2011/06/how-design-multiclient-databases/

2.2 p19



All causes have the same effects.

SQL Server sees more “unique” queries coming in:

Each query needs to be compiled
(because SQL Server hasn’t seen anything like it)
so that burns more CPU time.

Each query plan gets cached independently (because they’re all technically different), which means:

- More memory taken up by plans
- Less memory available to cache data
- Much harder to do plan cache analysis to catch patterns of bad queries

2.2 p20

How to tell if you have this problem

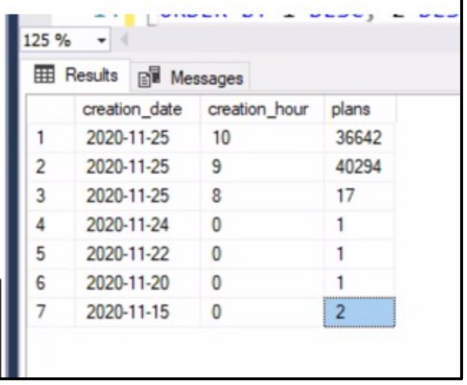
2.2 p21

1. Find out if you have the problem:

https://www.brentozar.com/archive/2018/07/tsql2sday-how-much-plan-cache-history-do-you-have/

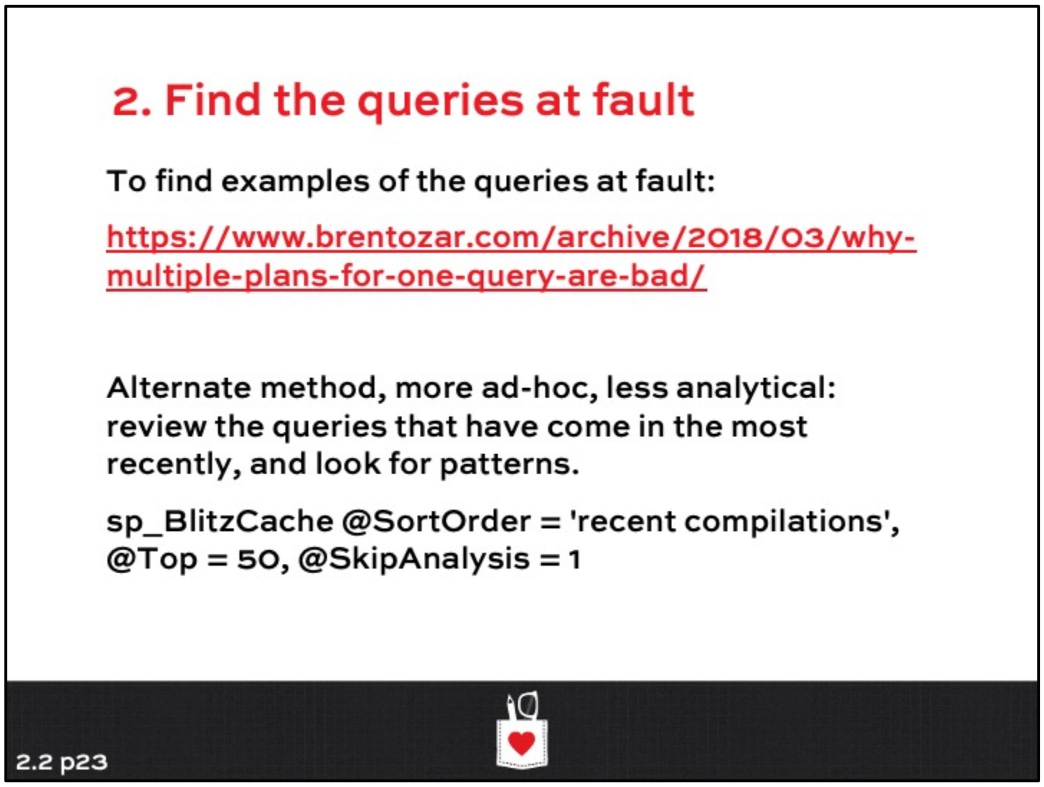
If your server has been up for a while (days/weeks), and yet most of your queries were compiled in the last ~24 hours, your query plans aren't getting reused.

Here's an example from a client I reviewed on Nov 25 at 10:45AM:



	creation_date	creation_hour	plans
1	2020-11-25	10	36642
2	2020-11-25	9	40294
3	2020-11-25	8	17
4	2020-11-24	0	1
5	2020-11-22	0	1
6	2020-11-20	0	1
7	2020-11-15	0	2

2.2 p22



2. Find the queries at fault

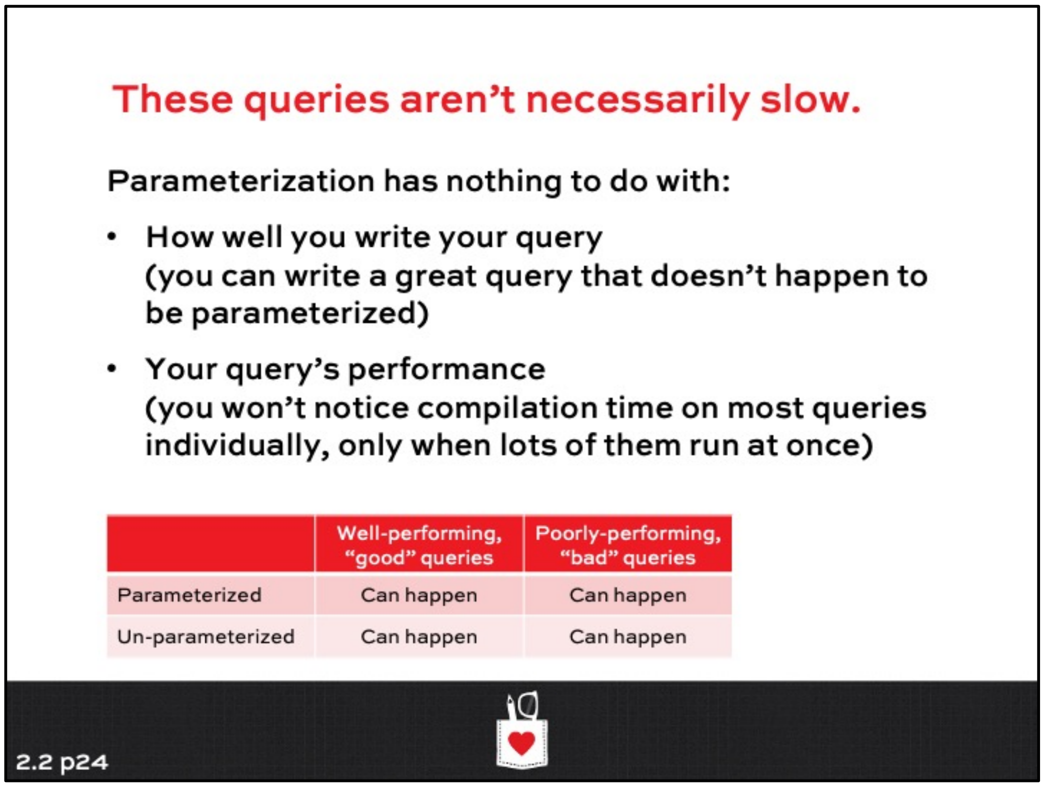
To find examples of the queries at fault:

https://www.brentozar.com/archive/2018/03/why-multiple-plans-for-one-query-are-bad/

Alternate method, more ad-hoc, less analytical: review the queries that have come in the most recently, and look for patterns.

sp_BlitzCache @SortOrder = 'recent compilations', @Top = 50, @SkipAnalysis = 1

2.2 p23



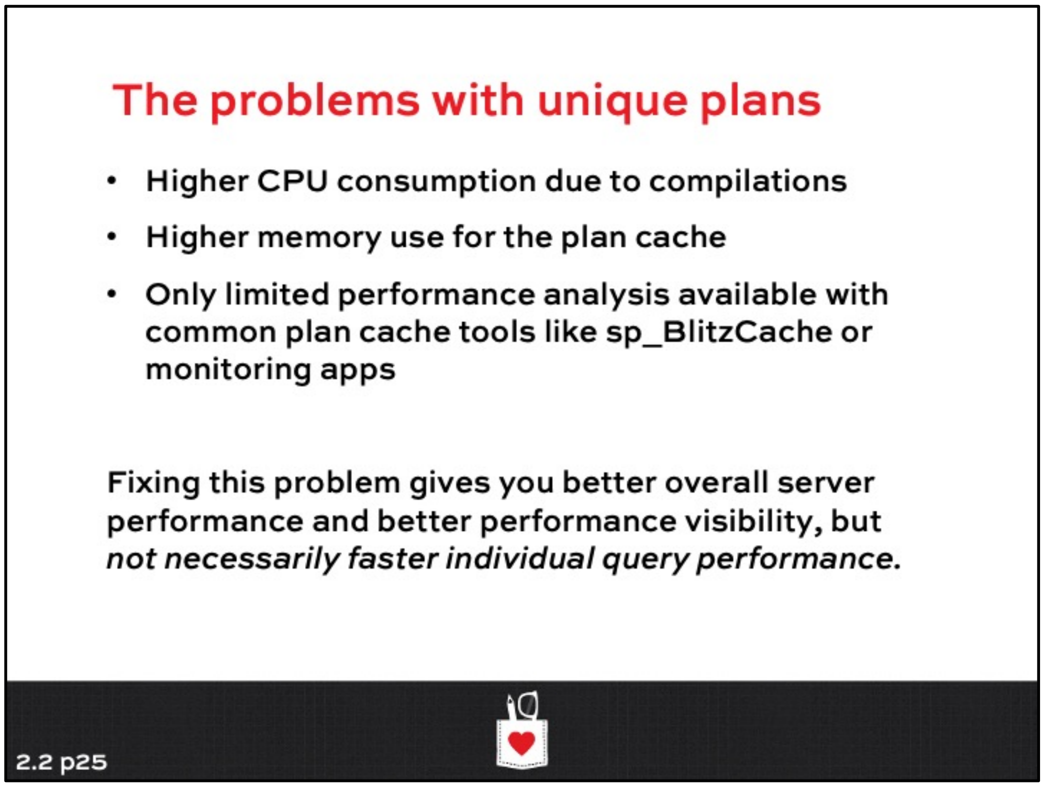
These queries aren't necessarily slow.

Parameterization has nothing to do with:

- How well you write your query (you can write a great query that doesn't happen to be parameterized)
- Your query's performance (you won't notice compilation time on most queries individually, only when lots of them run at once)

	Well-performing, "good" queries	Poorly-performing, "bad" queries
Parameterized	Can happen	Can happen
Un-parameterized	Can happen	Can happen

2.2 p24

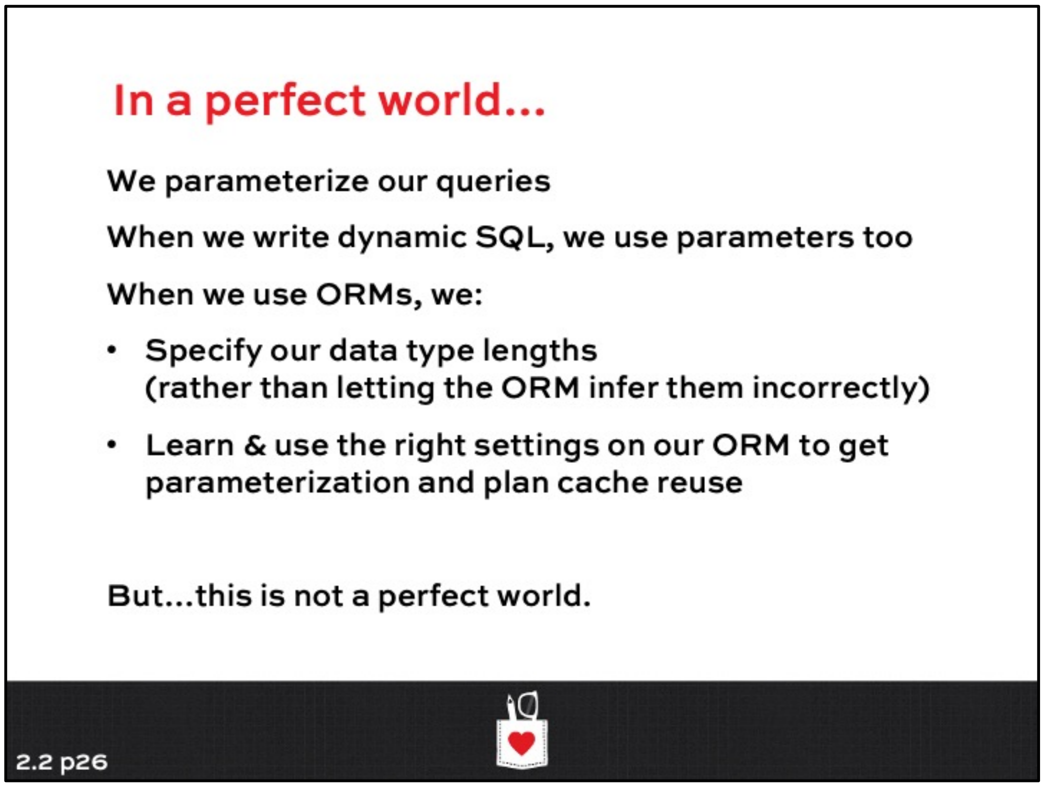


The problems with unique plans

- Higher CPU consumption due to compilations
- Higher memory use for the plan cache
- Only limited performance analysis available with common plan cache tools like sp_BlitzCache or monitoring apps

Fixing this problem gives you better overall server performance and better performance visibility, but *not necessarily faster individual query performance.*

2.2 p25



In a perfect world…

We parameterize our queries

When we write dynamic SQL, we use parameters too

When we use ORMs, we:

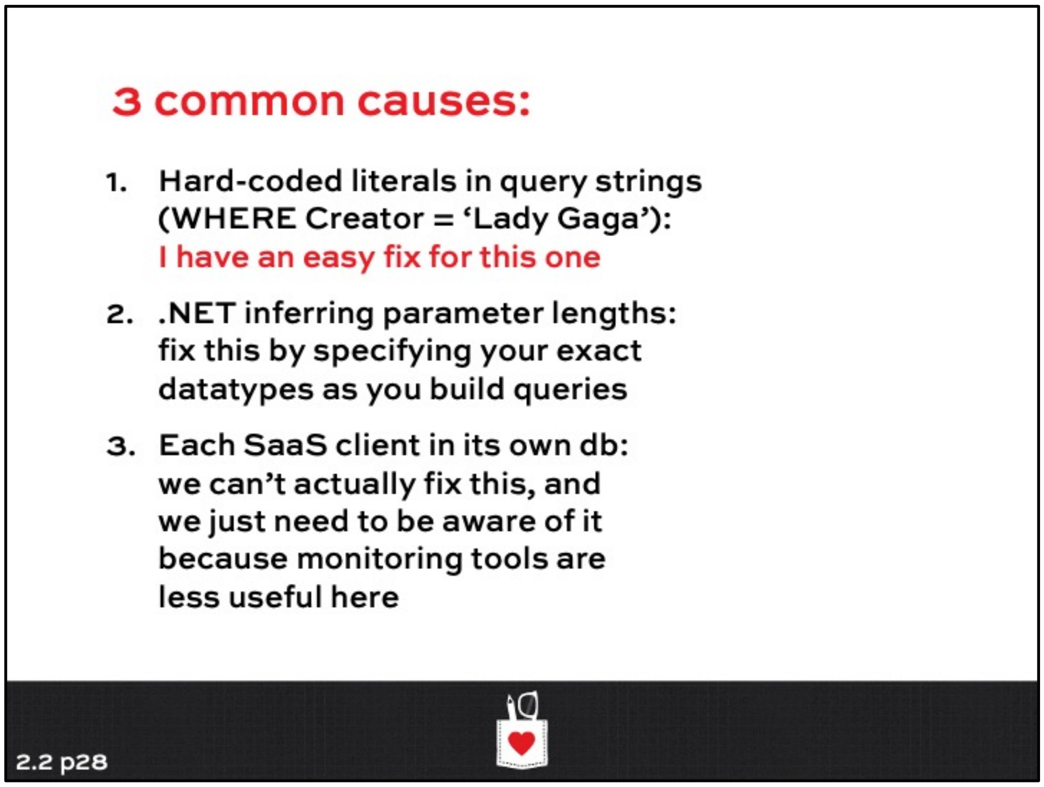
- Specify our data type lengths
  (rather than letting the ORM infer them incorrectly)
- Learn & use the right settings on our ORM to get parameterization and plan cache reuse

But…this is not a perfect world.

2.2 p26

How to work around the problem

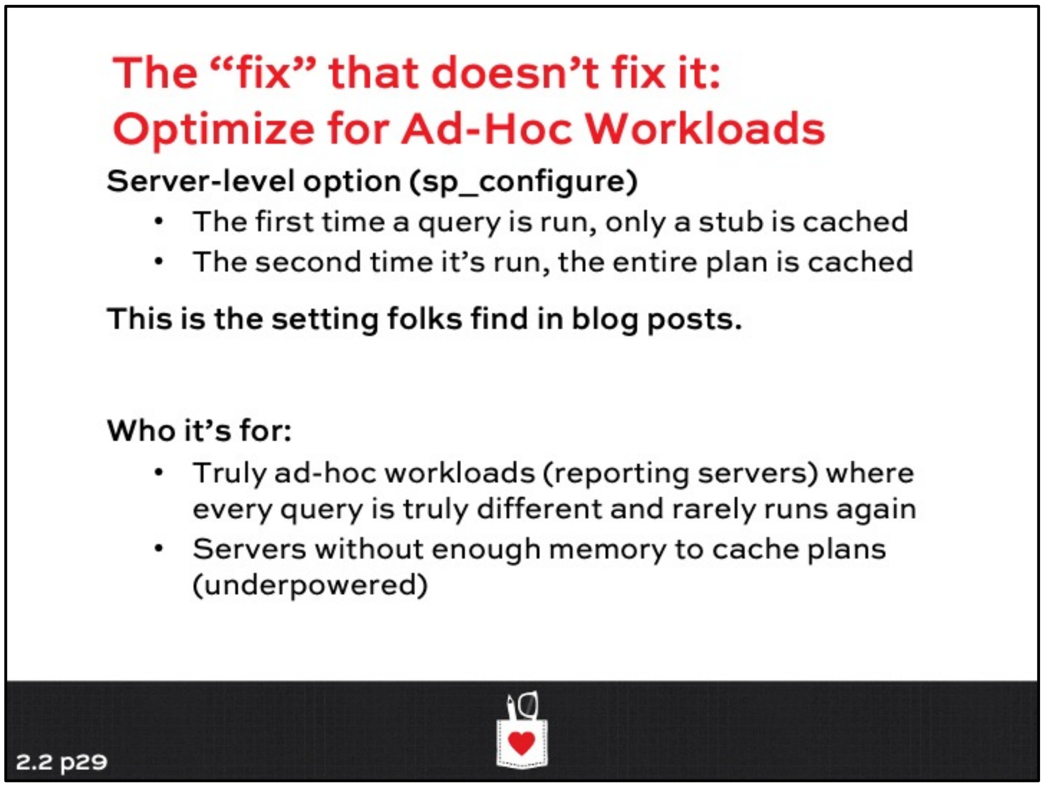
2.2 p27



3 common causes:

1. Hard-coded literals in query strings (WHERE Creator = 'Lady Gaga'): I have an easy fix for this one
2. .NET inferring parameter lengths: fix this by specifying your exact datatypes as you build queries
3. Each SaaS client in its own db: we can't actually fix this, and we just need to be aware of it because monitoring tools are less useful here

2.2 p28



The "fix" that doesn't fix it: Optimize for Ad-Hoc Workloads

Server-level option (sp_configure)

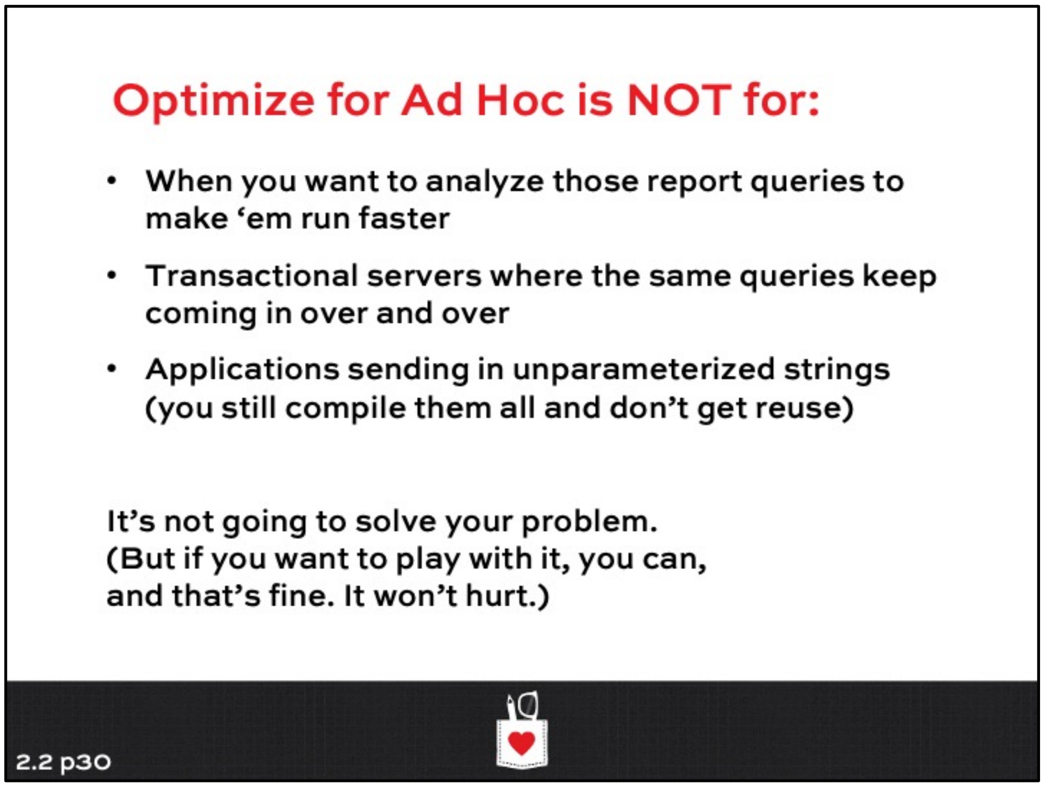
- The first time a query is run, only a stub is cached
- The second time it's run, the entire plan is cached

This is the setting folks find in blog posts.

Who it's for:

- Truly ad-hoc workloads (reporting servers) where every query is truly different and rarely runs again
- Servers without enough memory to cache plans (underpowered)

2.2 p29



Optimize for Ad Hoc is NOT for:

- When you want to analyze those report queries to make 'em run faster
- Transactional servers where the same queries keep coming in over and over
- Applications sending in unparameterized strings (you still compile them all and don't get reuse)

It's not going to solve your problem.
(But if you want to play with it, you can, and that's fine. It won't hurt.)

2.2 p30

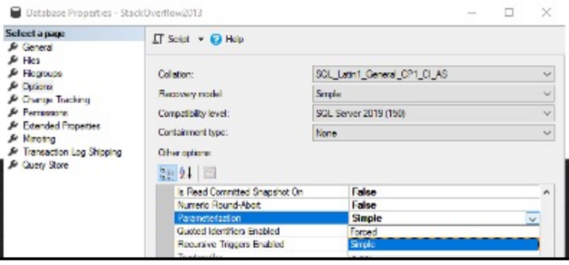
The real fix: Forced Parameterization

Database level setting that affects the context of where query is run – can be turned on/off at any time

Who it's for:

- Applications that send in unparameterized queries
- Solving high CPU usage due to compiles
- Solving a constantly-flushing plan cache

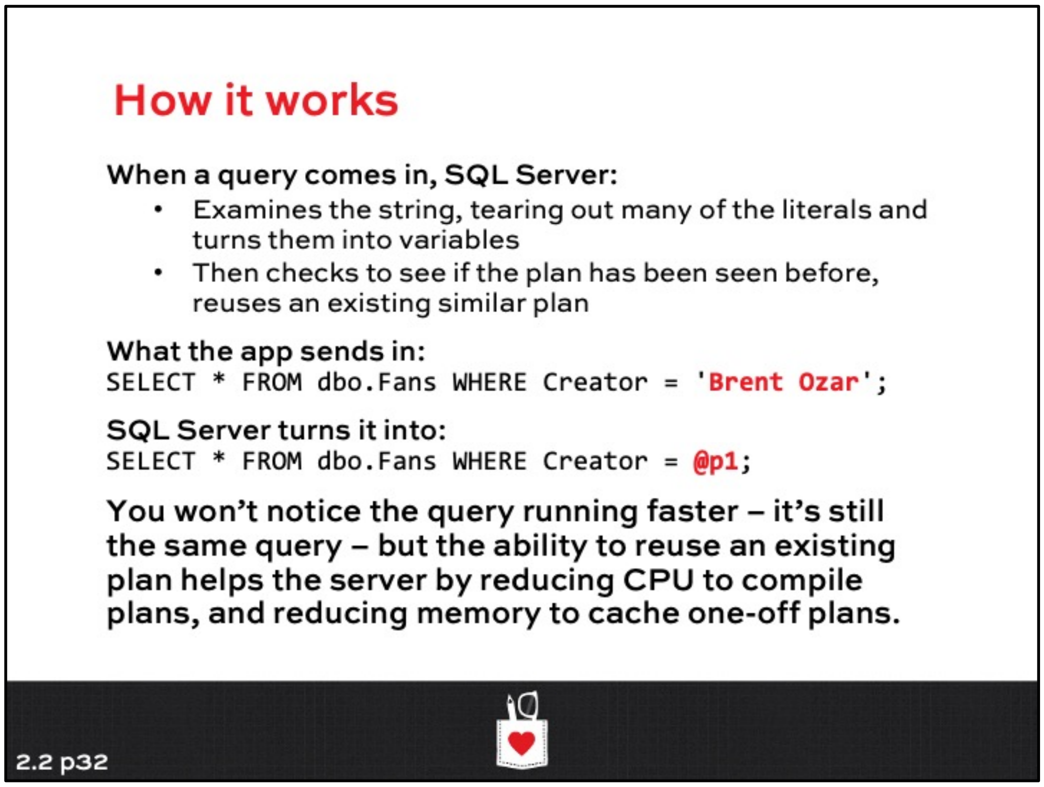
```
ALTER DATABASE [StackOverflow2013] SET
PARAMETERIZATION FORCED WITH NO_WAIT;
```



Collation: SQL_Latin1_General_CP1_CI_AS
Recovery model: Simple
Compatibility level: SQL Server 2019 (150)
Containment type: None

Is Read Committed Snapshot On	False
Numeric Round-Abort	False
Parameterization	Simple
Quoted Identifiers Enabled	Forced
Recursive Triggers Enabled	Simple

2.2 p31



How it works

When a query comes in, SQL Server:

- Examines the string, tearing out many of the literals and turns them into variables
- Then checks to see if the plan has been seen before, reuses an existing similar plan

What the app sends in:

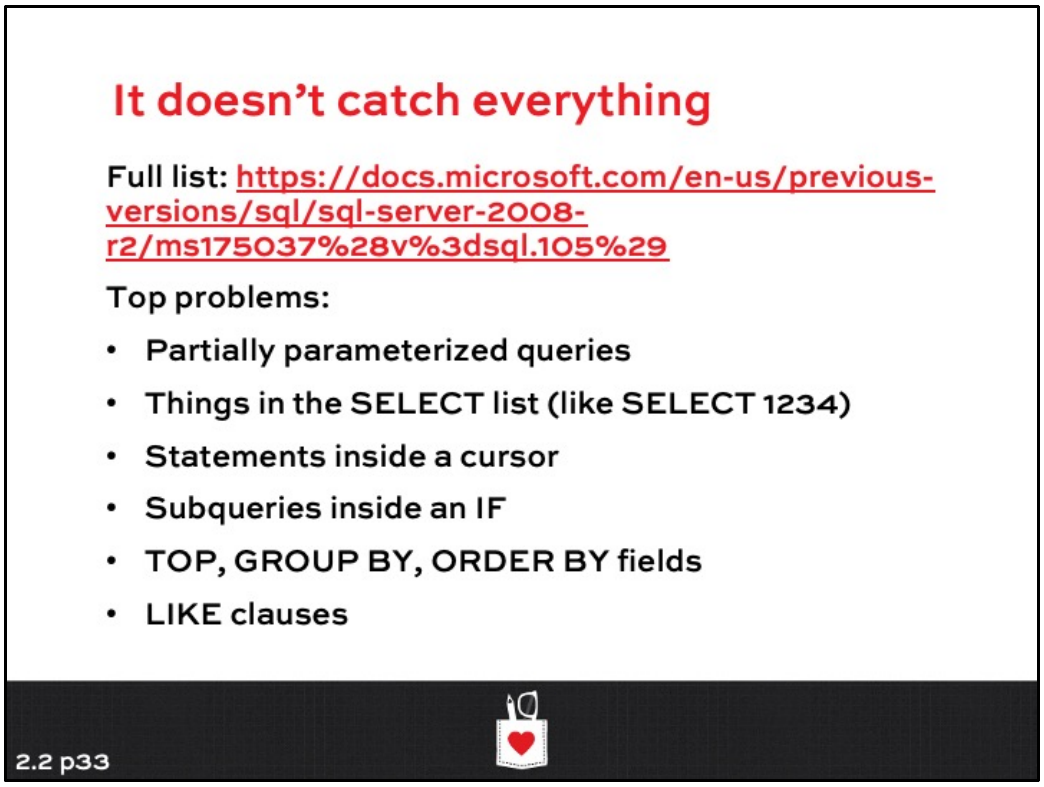
```
SELECT * FROM dbo.Fans WHERE Creator = 'Brent Ozar';
```

SQL Server turns it into:

```
SELECT * FROM dbo.Fans WHERE Creator = @p1;
```

You won't notice the query running faster – it's still the same query – but the ability to reuse an existing plan helps the server by reducing CPU to compile plans, and reducing memory to cache one-off plans.

2.2 p32



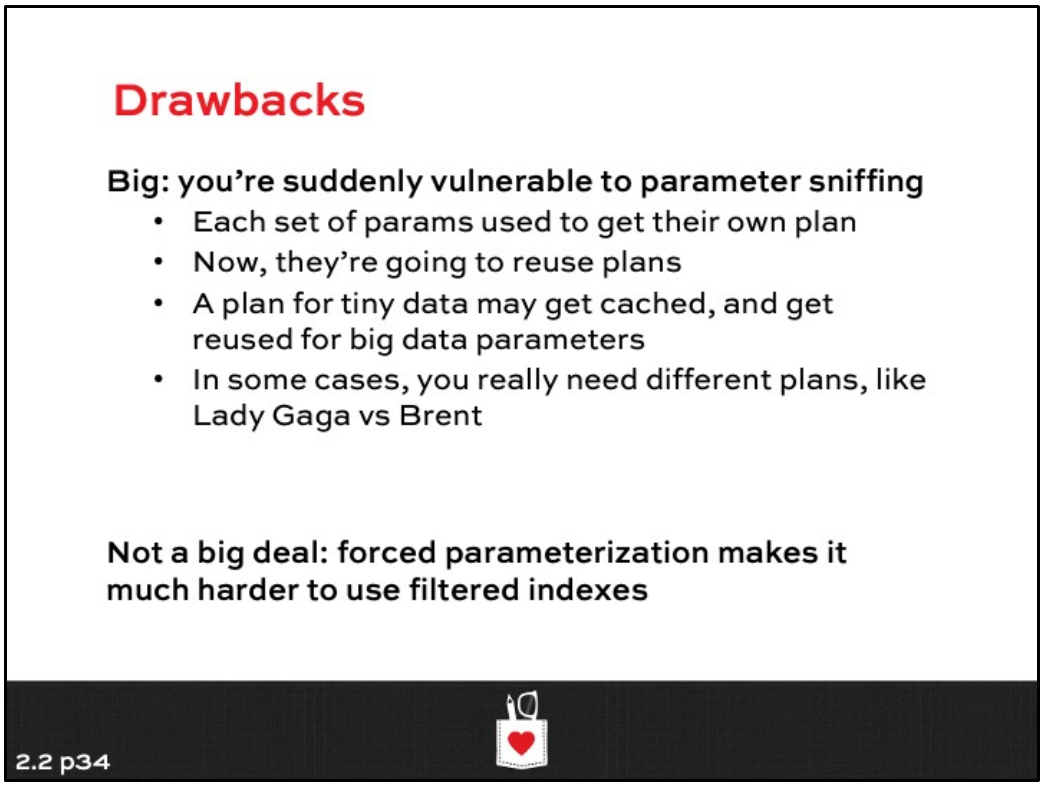
It doesn't catch everything

Full list: https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms175037%28v%3dsql.105%29

Top problems:

- Partially parameterized queries
- Things in the SELECT list (like SELECT 1234)
- Statements inside a cursor
- Subqueries inside an IF
- TOP, GROUP BY, ORDER BY fields
- LIKE clauses

2.2 p33



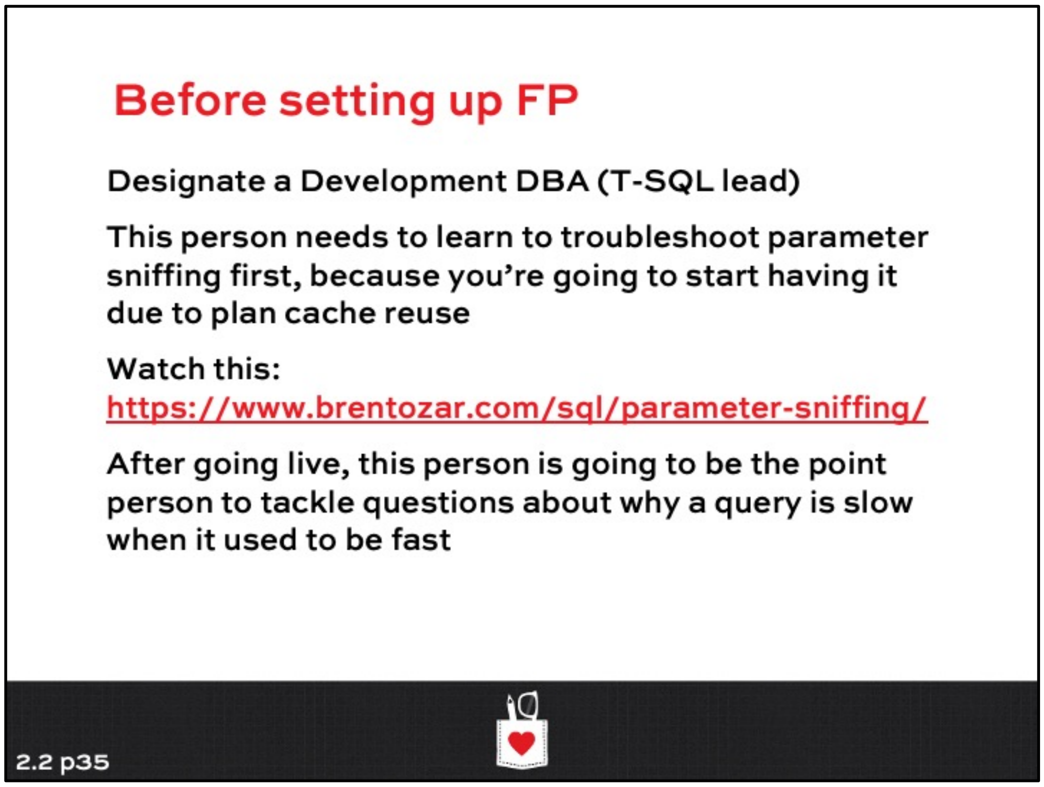
Drawbacks

Big: you're suddenly vulnerable to parameter sniffing

- Each set of params used to get their own plan
- Now, they're going to reuse plans
- A plan for tiny data may get cached, and get reused for big data parameters
- In some cases, you really need different plans, like Lady Gaga vs Brent

Not a big deal: forced parameterization makes it much harder to use filtered indexes

2.2 p34



Before setting up FP

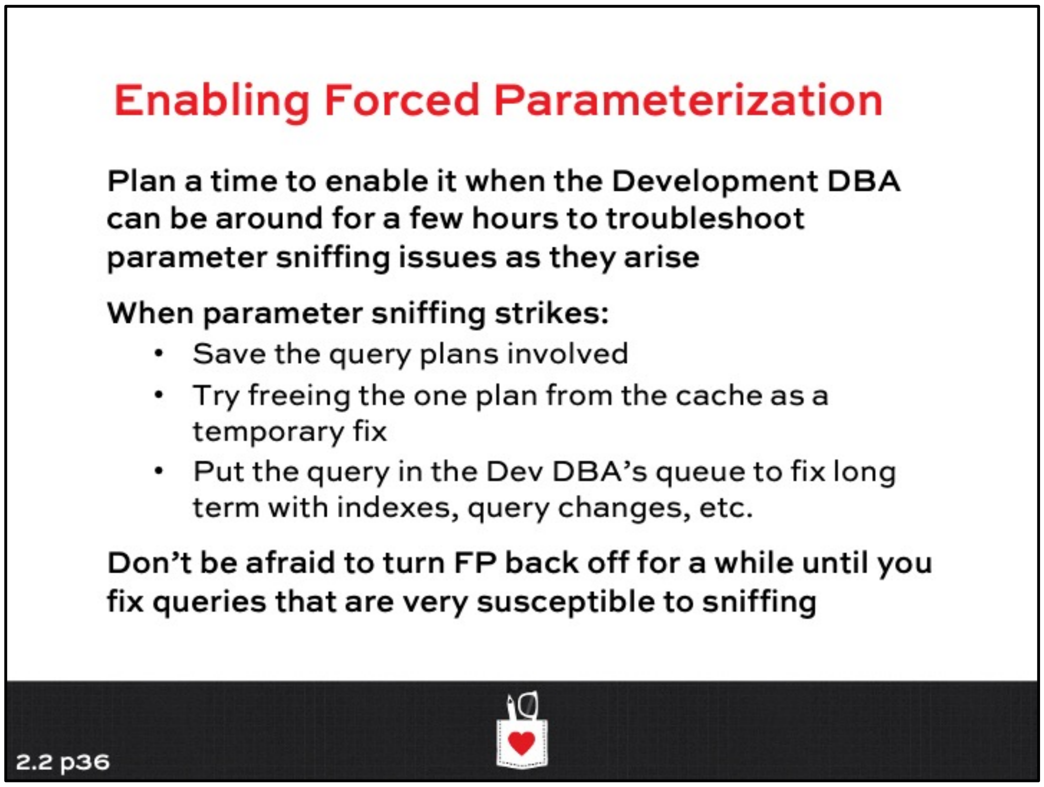
Designate a Development DBA (T-SQL lead)

This person needs to learn to troubleshoot parameter sniffing first, because you're going to start having it due to plan cache reuse

Watch this:
https://www.brentozar.com/sql/parameter-sniffing/

After going live, this person is going to be the point person to tackle questions about why a query is slow when it used to be fast

2.2 p35



Enabling Forced Parameterization

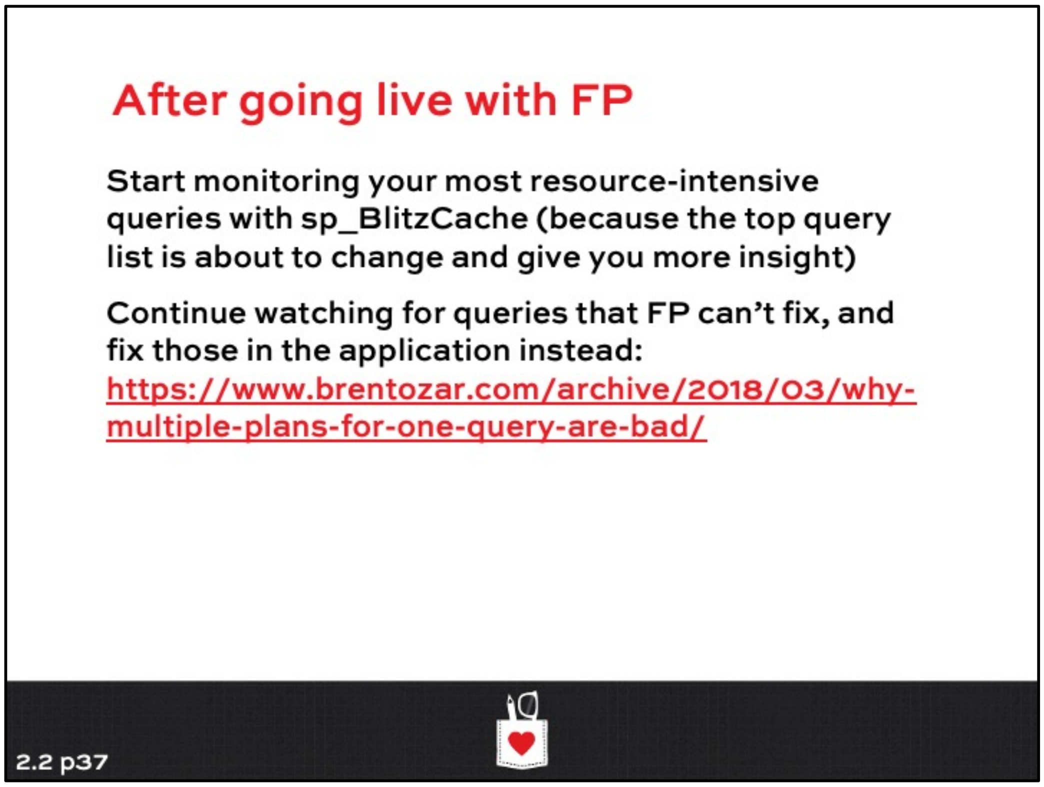
Plan a time to enable it when the Development DBA can be around for a few hours to troubleshoot parameter sniffing issues as they arise

When parameter sniffing strikes:

- Save the query plans involved
- Try freeing the one plan from the cache as a temporary fix
- Put the query in the Dev DBA's queue to fix long term with indexes, query changes, etc.

Don't be afraid to turn FP back off for a while until you fix queries that are very susceptible to sniffing

2.2 p36

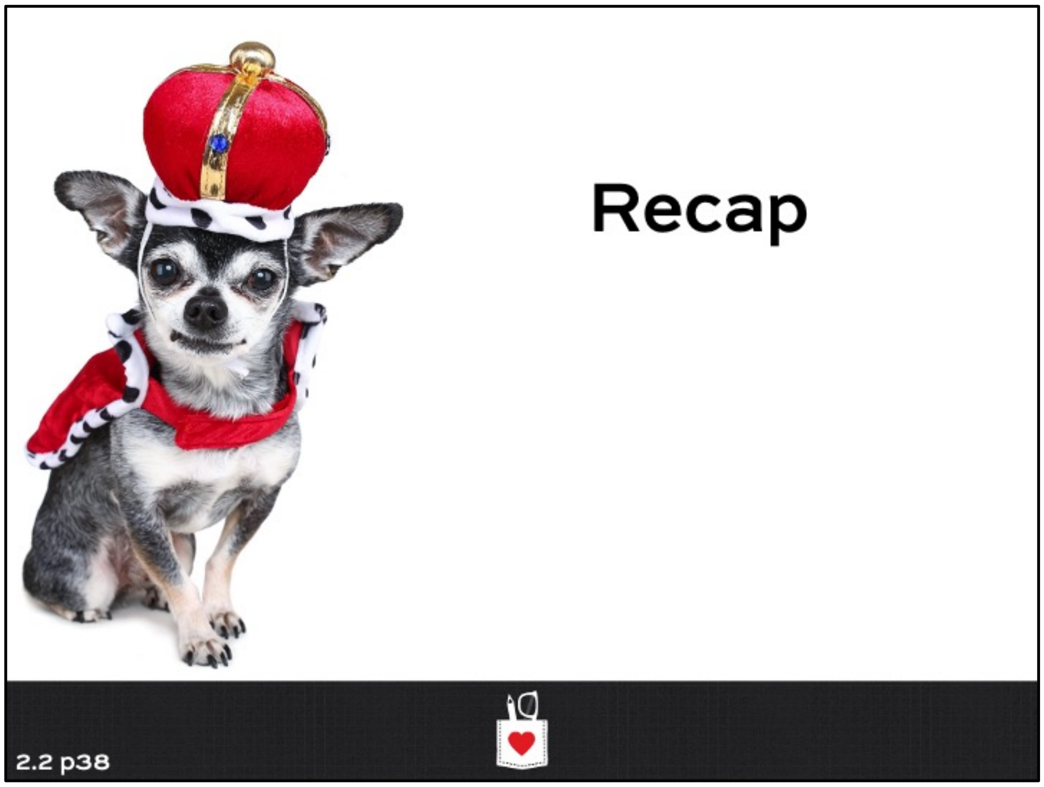


After going live with FP

Start monitoring your most resource-intensive queries with sp_BlitzCache (because the top query list is about to change and give you more insight)

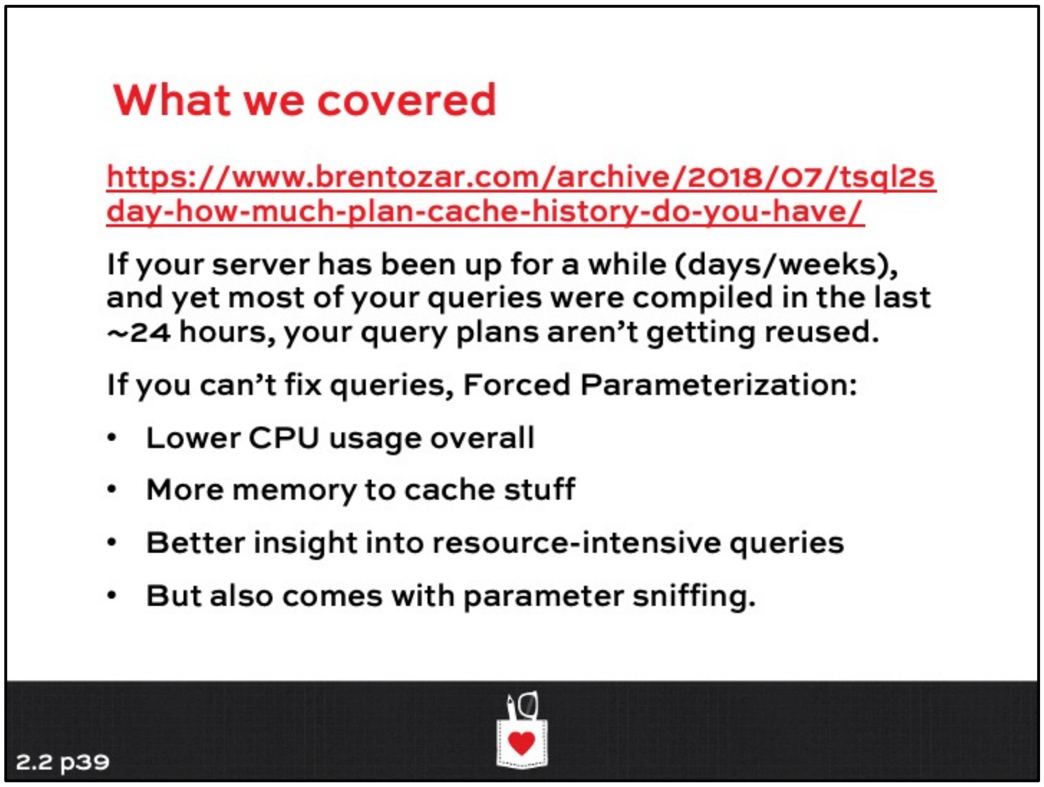
Continue watching for queries that FP can't fix, and fix those in the application instead: https://www.brentozar.com/archive/2018/03/why-multiple-plans-for-one-query-are-bad/

2.2 p37



Recap

2.2 p38



What we covered

https://www.brentozar.com/archive/2018/07/tsql2sday-how-much-plan-cache-history-do-you-have/

If your server has been up for a while (days/weeks), and yet most of your queries were compiled in the last ~24 hours, your query plans aren't getting reused.

If you can't fix queries, Forced Parameterization:

- Lower CPU usage overall
- More memory to cache stuff
- Better insight into resource-intensive queries
- But also comes with parameter sniffing.

2.2 p39