

**sqlbits**

Session: Friday, October 2, 2020

# Troubleshooting Query Plans: Part 1

## Root Cause Analysis and Caching Problems

Kimberly L. Tripp

President / Founder, SQLskills.com

Kimberly@SQLskills.com

@KimberlyLTripp

**sqlbits**





## Consultant / Trainer / Speaker / Writer

- Author / instructor for SQL Server Immersion Events: IEPT01, IEPT02, and IEHADR
- Author / presenter for Pluralsight
- Instructor and exam author for SQL Server and Sharepoint MCMs
- Author / manager of SQL Server 2005 and 2008 Launch Content
- Author / speaker at Microsoft TechEd, SQLPASS, ITForum, TechDays, and SQLintersection
- Author of SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability, SQLCLR for DBAs
- Author / presenter for MSDN and TechNet (25+)
- Instructor / SME for Microsoft University

## Data Platform MVP

17 years: 2002-2019



## When I'm not working with SQL

- Traveler/adventurer and avid diver/photographer: [www.BlueWaterImages.com](http://www.BlueWaterImages.com)
- @KimberlyLTripp on Insta



# Kimberly L. Tripp

Founder / President, SQLskills



[Kimberly@SQLskills.com](mailto:Kimberly@SQLskills.com)



[KimberlyLTripp](https://twitter.com/KimberlyLTripp)



[www.sqlskills.com/blogs/Kimberly](http://www.sqlskills.com/blogs/Kimberly)

# Overview

- **Root cause analysis**
  - Testing / troubleshooting – is it a CACHING problem?
- **Statement Execution and Caching**
  - Different ways to execute statements
  - Statements caching for reuse
  - Statement auto-parameterization
  - Dynamic string execution
  - sp\_executesql (and stored procedures)
  - The life of a plan in cache
  - Plan cache limits
  - Bringing it all together

# Performance Problems

- Query performance inconsistencies
  - Execution time varies
  - Statement's execution plan varies
  - IO / CPU metrics vary
- Variations due to:
  - Statement execution method
  - Parameters
  - Time (fragmentation, statistics not current, plan not valid)
- Sledgehammer approaches
  - Updating statistics
  - Rebuilding indexes
  - Clearing cache

# BETTER: Troubleshooting Methodologies (1 of 2)

- Is this a CACHED plan method?
  - Stored procedure
  - `sp_executesql`
- Test to see if the optimal plan varies from the cached plan?
  - `EXECUTE <procedure> <params>` or `sp_executesql` (same way)
  - VS. recompiling: `EXECUTE <procedure> <params> WITH RECOMPILE`
  - NOTE: execute with recompile does NOT allow the parameterization embedding optimization [doesn't optimize as effectively as `OPTION (RECOMPILE)` so there are features that might not be leveraged EXCEPT when using `OPTION (RECOMPILE)` at the statement level but that's not a problem here...]
- Do you get a different plan for the second?
  - Parameter sensitivity ("sniffing") problem
    - Long term solution is to changing the code

# BETTER: Troubleshooting Methodologies (2 of 2)

- Is this a poorly performing but NEW (non-cached) plan?
  - AdHoc statement
  - First execution
  - Plan doesn't change when using WITH RECOMPILE
- Use **ACTUAL EXECUTION**
  - Estimated plan vs. actual plan
    - Be sure to check executions \* rows (not just estimated rows vs. actual rows)
  - Problems / solutions that can be exposed by incorrect row estimations
    - Statistics out-of-date -> updating scenarios
    - Estimate is incorrect because of skewed data -> filtered statistics scenarios
    - Estimate is incorrect because of estimation algorithm -> cardinality estimation scenarios

# Overview

- Root cause analysis
  - Testing / troubleshooting – is it a CACHING problem?
- Statement Execution and Caching
  - Different ways to execute statements
  - Statements caching for reuse
  - Statement auto-parameterization
  - Dynamic string execution
  - sp\_executesql (and stored procedures)
  - The life of a plan in cache
  - Plan cache limits
  - Bringing it all together

# Different Ways to Execute SQL Statements

- Ad hoc statements
  - Possibly, as auto-parameterized statements
- Dynamic string execution (DSE)
  - EXECUTE ( @string )

*These two behave  
EXACTLY the same way!*

- sp\_executesql (forced statement caching)
- Prepared queries (forced statement caching through “parameter markers”)
  - Client-side caching from ODBC and OLEDB (parameter via question mark)
  - Exposed via SQLPrepare / SQLExecute and ICommandPrepare
- Stored Procedures
  - Including statements with literals, parameters, and/or variables
  - Including dynamic strings
  - Including sp\_executesql

*In this section, these behave the same way but some exceptions  
exist with certain statement types inside stored procedures  
(a bit more on this is coming up)*



# Why is this Such an Issue?

- EVIL ORMS!
- ORM = Object Relational Mapper
- Pro – Rapid Application Development
  - Fast **IO** production
  - Slow **IN** production
- How **NOT** to structure your database-backed web applications: a study of performance bugs in the wild
- <https://bit.ly/2lxpk14>

The screenshot shows a Twitter thread with three tweets. The first tweet, by Caitie McCaffrey (@caitie), is the top tweet and discusses a paper on ORM performance. The second tweet, by Adrian Colyer (@adriancolyer), is a reply to the first and discusses the paper's findings. The third tweet, by Caitie McCaffrey (@caitie), is a reply to the second and discusses the paper's findings. The third tweet is highlighted with an orange border.

**Caitie McCaffrey** @caitie 54m  
Great read on some performance pit falls in ORMs

**Adrian Colyer** @adriancolyer  
ORM could also stand for "Opaque Relational Mapper". Yang et al., investigate common ORM-related performance issues and a build a tool to help you find them in your own (Rails) applications. [blog.acolyer.org/2018/06/28/how...](http://blog.acolyer.org/2018/06/28/how...) Just a few lines of code changed can make a huge difference!

8:54am · 28 Jun 2018 · Twitter Web Client

2 REPLIES 8 RETWEETS 11 LIKES

Reply to @caitie

**Caitie McCaffrey** @caitie 31m  
Also this paper doesn't discuss consistency issues. I think this is because the studied ORMs are all on top of relational databases.

**Caitie McCaffrey** @caitie 30m  
But in my experience ORMs on top of Eventually Consistent or NoSql data stores also have a ton of consistency challenges because you need to understand the implementation to get the consistency correct.

**Caitie McCaffrey** @caitie 27m  
Basically ORMs are very leaky storage abstractions, and I would strongly caution against using them.

I totally agree!

# Some Statements can be Cached for Reuse (1 of 2)

- Ad hoc statements and dynamic strings are evaluated at runtime
  - If a statement is very simple ("safe"), then it can be parameterized and cached
    - It's generally a good thing that the plans are cached
      - Saves CPU/time
      - Reduced footprint in the cache
    - This can lead to a small amount of prepared plan cache bloat when the parameters are typed per execution:

```
SELECT ... WHERE member_no = 12
```

```
↳ (@1 tinyint)SELECT ... WHERE [member_no]=@1
```

```
SELECT ... WHERE member_no = 278
```

```
↳ (@1 smallint)SELECT ... WHERE [member_no]=@1
```

```
SELECT ... WHERE member_no = 62578
```

```
↳ (@1 int)SELECT ... WHERE [member_no]=@1
```

# Some Statements can be Cached for Reuse (2 of 2)

- Ad hoc statements and dynamic strings are evaluated at runtime
  - And, unfortunately, most statements won't be safe:
    - Many query limitations:
      - FROM clause cannot have more than one table
      - WHERE clause cannot have expressions joined by OR
      - WHERE clause cannot have an IN clause
      - Statement cannot contain a sub-query
      - VERY restrictive (see Appendix A in [whitepaper](#) for complete list)
    - Parameters do not change plan choice
  - Even when a statement's NOT safe, the un-parameterized statement (and the specific literal values) will be placed in the ad hoc plan cache
    - Used for later "exact textual matching" cases
    - Eats up the cache quickly because:
      - Most statements aren't safe
      - Lots of statements are executing

# Statement Auto-Parameterization: Keep It Simple!

- How does parameterization work by default: SIMPLE
  - Most statements are probably NOT deemed safe
- Database option: parameterization FORCED
  - Generally, not recommended
  - Many more statements are forced to be cached
    - PRO: if you have stable plans from a lot of adhoc clients then this might help to significantly reduce CPU
    - CON: If you have some statements that really aren't safe, you could end up executing bad plans... better to do this right from the start and control it yourself with stored procedures (much more difficult!)
    - Recommendation: can investigate plan stability BEFORE changing to forced by using query\_hash and query\_plan\_hash (check out the Pluralsight course on [SQL Server: Optimizing Ad Hoc Statement Performance](#), Section: Plan Cache Pollution)
- If statement is parameterized and safe (or forced), SQL Server places statement in cache for subsequent executions
- If statement is unsafe, SQL Server places statement in cache for subsequent executions through TEXTUAL MATCHING ONLY



# Ad Hoc Statement / Dynamic String Execution (DSE)

- Statement / string is NOT evaluated until runtime (execution)
- Parameters allow virtually any statement to be built “dynamically”
- This statement is then treated as an ad hoc statement
  - If it's safe – it will be parameterized, saved in cache, and reused
  - If it's unsafe – it will be recompiled for each/every execution
    - Just to be clear – DSE does not automatically mean it's compiled for every execution
      - Textual matching can occur (with statements in the ad hoc plan cache)
      - Parameterization can occur (again, only if it's safe)
- String can be up to 2GB in size
  - 2005+: Can declare variable of type (n)varchar(max)
  - sp\_executesql only allows parameters where a typical SQL statement would allow them; however, these two can be combined!
- Can be complex to write, read, perform
- And there's a whole discussion about security...

# Understanding sp\_executesql

- Usually used to help build statements from applications
- Parameters are typed explicitly
- Forces a plan in cache for the parameterized string (when one doesn't already exist) – subsequent executions will use this plan
  - Can be EXCELLENT if the statement's plan is stable even with different parameters
  - Can be horrible if the statement's most optimal plan varies from execution to execution (because of the parameters)
- Almost like dynamic string execution, but it's not!
  - Often compared to DSE [where you EXEC (@ExecStr)] but they're not the same
    - sp\_executesql is a parameterized statement that works JUST like a stored procedure
    - DSE is just a way of building an ad hoc statement that's not evaluated until runtime
      - If it's safe – it's parameterized and reused
      - If it's not safe – then it's not (meaning, it will be in the ad hoc plan cache but not the compiled plan cache AND it will need to be compiled for each execution)

# Stored Procedure Caching Just Like sp\_executesql

- Places a plan in cache for the stored procedure (when one doesn't already exist) – subsequent executions will use this plan
- Reusing plans can be good
  - When different parameters don't change the optimal plan, then caching / saving and reusing is excellent!
  - SQL Server saves CPU and time in compilation
- Reusing plans can be VERY bad
  - When different parameters are used and the optimal plans vary by parameter set, then reusing the plan can be horribly bad

# Literals, Variables, and Parameters (1 of 2)



Check out the Pluralsight  
courses for in-depth information  
about these options!

## ■ Literals

```
SELECT ... WHERE member_no = 12
```

- member\_no is being compared to a specific, defined value
- Literals are “known” values at the time of optimization (compilation)
- They provide the best information for optimization

## ■ Variables

```
DECLARE @mno int = 12  
SELECT ... WHERE member_no = @mno
```

- @mno is being defined and set in the declaration line
- Variables are “unknown” values at the time of optimization (compilation) as the assignment does not occur until runtime
- They provide NO useful information for optimization
- The optimizer has to use “averages” instead of real values
  - This can be both good and bad...



# Literals, Variables, and Parameters (2 of 2)



*hidden slide  
w/extra details*

**Check out the Pluralsight  
courses for in-depth information  
about these options!**

- Parameters look like variables but they're not...
- Variables CAN be defined within a stored procedure

```
CREATE PROCEDURE procname  
    ( @parameter1    int )  
AS  
DECLARE @mno int = 12  
SELECT ... WHERE member_no = @mno  
SELECT ... WHERE member_no = @parameter1
```

- @mno is being defined and set inside the stored procedure; this is a variable
- @parameter1 is being defined as a parameter and the value will be KNOWN at execution
- If this procedure does not yet have a plan defined in cache, then SQL Server will evaluate PARAMETER values during optimization (not VARIABLES)
- This leads to the phrase: parameters can be "sniffed," variables are unknown.

# The Life of a Plan in Cache

- A plan is generated when no plan already exists in cache
- Plans are never saved on disk and may not persist within the cache
  - Some operations completely remove / evict the plan from the cache
    - Server-level: Server restart | DBCC FREEPROCCACHE | some RECONFIGURE changes
      - See Pluralsight recordings starting with Side Effect: Plan Cache Flush for version-specific details and demo (Optimizing Procedural Code course)
    - Database-level: DBCC FLUSHPROCINDB (undocumented) | sp\_dbcmptlevel
    - Procedure-level: sp\_recompile or they're aged out through non-use
  - Some operations cause the plan to be invalidated (but it still remains an object in the cache); these can be tracked
    - Schema of base object changes (ALTER TABLE / ALTER <object>)
      - Including when indexes are added to the base object
    - Statistics of base objects change
      - See recordings starting with Plan Invalidation for version-specific details and demos
- When a plan exists in cache, all subsequent executions use that plan

# Plan Invalidation



Check out the **Pluralsight**  
courses for in-depth information  
about these options!

## ■ Versions prior to 2012

- If database option: auto\_update\_stats is ON
  - Updating statistics causes plan invalidation
- If database option: auto\_update\_stats is OFF
  - Updating statistics does NOT cause plan invalidation
- If you manually update statistics and have set auto\_update\_statistics to OFF, add sp\_recompile @tname to your stats scripts (remembering SCH\_M problems)
- For more info: Erin Stellato's links about auto update stats/plan invalidation:
  - Statistics and Recompilations: <http://erinstellato.com/2012/01/statistics-recompilations/>
  - Statistics and Recompilations, Part II: <http://erinstellato.com/2012/02/statistics-recompilations-part-ii/>

## ■ SQL Server 2012+

- Plan invalidation is NOT affected by the setting of auto update stats
- Plan invalidation does NOT occur if data has not changed
  - Only for UPDATE STATISTICS
  - An index rebuild will update statistics as well as cause plan invalidation, even if no data has changed

# Plan Cache Usage/Limits

- The “plan cache” (a.k.a. “procedure cache”)
- Uses “stolen” pages from the buffer pool (data pages)
- View cached plans: sys.dm\_exec\_cached\_plans
- Plan cache memory limits:
  - SQL Server 2005 SP2 and higher
    - 75% of visible target memory from 0-4GB
    - + 10% of visible target memory from 4Gb-64GB
    - + 5% of visible target memory > 64GB
  - SQL Server 2005 RTM and SQL Server 2005 SP1
    - 75% of visible target memory from 0-8GB
    - + 50% of visible target memory from 8Gb-64GB
    - + 25% of visible target memory > 64GB
  - SQL Server 2000
    - SQL Server 2000 4GB upper cap on the plan cache
  - 32-bit? AWE memory is not “visible” to the plan cache (plan cache has to live in “real memory”)

2005 SP2 +	
Memory	Plan Cache
4GB	3.0GB
8GB	3.5GB
16GB	4.2GB
32GB	5.8GB
64GB	9.0GB
128GB	12.2GB
256GB	18.6GB
512GB	31.4GB

## Consolidation Note

If you've consolidated / virtualized multiple servers then the real question is – how much memory have you given to each SQL Server?

# Reducing Plan Cache Bloat & Optimizing Statement Execution

- Analyze your plan cache to see if you have ad hoc “bloat”
  - See plan cache blog post slide for query/details to analyze your cache
- Turn on the server-wide configuration “optimize for ad hoc workloads”
- Setup a job to regularly monitor for plan cache bloat and clear the SQL Plans portion of the cache using DBCC FREESYSTEMCACHE
- Work to change the way that ad hoc requests are made
  - Use ad Hoc for statements with unstable plans
  - Use sp\_executesql for statements with stable plans
  - If you want centralized logic, code reuse, and compiled / cached plans (when they’re stable) and lots of other options (for when the plans are not stable), use stored procedures
    - Written by database developers that should
      - Know the data / workload / requirements
      - Know how SQL Server works
    - Can be written to balance CPU / caching for optimal performance!

# Review

- **Root cause analysis**
  - Testing / troubleshooting – is it a CACHING problem?
- **Statement Execution and Caching**
  - Different ways to execute statements
  - Statements caching for reuse
  - Statement auto-parameterization
  - Dynamic string execution
  - sp\_executesql (and stored procedures)
  - The life of a plan in cache
  - Plan cache limits
  - Bringing it all together



- Demo code/samples: SQLskills, Resources, Demo Scripts and Sample Databases
- Courses on Pluralsight: [www.pluralsight.com](https://www.pluralsight.com)
  - SQL Server: Indexing for Performance (7 hr., 11 min)
  - SQL Server: Why Physical Database Design Matters
  - **SQL Server: Optimizing Ad Hoc Statement Performance**
    - More details on plan cache bloat, analyzing single-use plans, determining if you can leverage "forced" parameterization (which is probably NOT), and lots of samples/demos!
  - SQL Server: Optimizing Stored Procedure Performance (Parts 1 and 2)
    - Part 2 has an entire section on troubleshooting session settings (in general, and for performance-related features)

# Plan Caching and Recompilation Resources

- Plan Caching and Recompilation in SQL Server 2012
  - <http://msdn.microsoft.com/en-us/library/dn148262.aspx>
- Plan Caching in SQL Server 2008
  - <http://msdn.microsoft.com/en-us/library/ee343986.aspx>
- Batch Compilation, Recompilation, and Plan Caching Issues in SQL Server 2005
  - <http://www.microsoft.com/technet/prodtechnol/sql/2005/recomp.msp>
- PSS SQL Server Engine Blog
  - <http://blogs.msdn.com/psssql/default.aspx>
- KB 243586: Troubleshooting Stored Procedure Recompilation



# Plan Cache Pollution Resources

- Blog posts:
  - [Plan cache and optimizing for adhoc workloads](#)
  - [Plan cache, adhoc workloads and clearing the single-use plan cache bloat](#)
  - [Clearing the cache - are there other options?](#)
  - [Statement execution and why you should use stored procedures](#)
  - Category: Optimizing Procedural Code
    - <http://www.sqlskills.com/BLOGS/KIMBERLY/category/Optimizing-Procedural-Code.aspx>
- MSDN Article: How Data Access Code Affects Database Performance, Bob Beauchemin
  - <http://msdn.microsoft.com/en-us/magazine/ee236412.aspx>

# Optimizing Procedural Code Posts and PASStv

- Start with this blog post (and sample code):
  - Building High Performance Stored Procedures
  - <http://www.sqlskills.com/blogs/kimberly/high-performance-procedures/>
- Then, watch my PASS Summit 2014 presentation through PASStv
  - <http://www.sqlpass.org/summit/2014/passtv.aspx>
  - See, Day 1: 4:30pm
    - *Dealing with Multipurpose Procs and PSP the RIGHT Way!* - Kimberly Tripp
- Then, check out this post: Stored Procedure Execution with Parameters, Variables, and Literals
  - <https://www.sqlskills.com/blogs/kimberly/stored-procedure-execution-with-parameters-variables-and-literals/>

# Questions?



Don't forget to complete an online evaluation!

## Troubleshooting Query Plans: Part 1

### Root Cause Analysis and Caching Problems

Kimberly L. Tripp

Your evaluation helps organizers build better conferences  
and helps speakers improve their sessions.

# sqlbits

# Thank you!