SQLskills Immersion Event

IEPTO2: Performance Tuning and Optimization

Module 8: Statement Execution, Stored Procedures, and the Plan Cache

Kimberly L. Tripp

Kimberly@SQLskills.com

Overview

- **Understanding plan cache**
  - Statement execution
    - sp_executesql
    - Dynamic string execution
  - Query fingerprints ⇒ the cumulative effect
  - Reducing plan cache pollution
- **Understanding/optimizing stored procedures**
  - Stored procedure creation
  - Stored procedure coding best practices for performance
  - Stored procedure execution
  - Stored procedure recompilation/optimization

© SQLskills, All rights reserved.
https://www.sqlskills.com

Different Ways to Execute SQL Statements

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

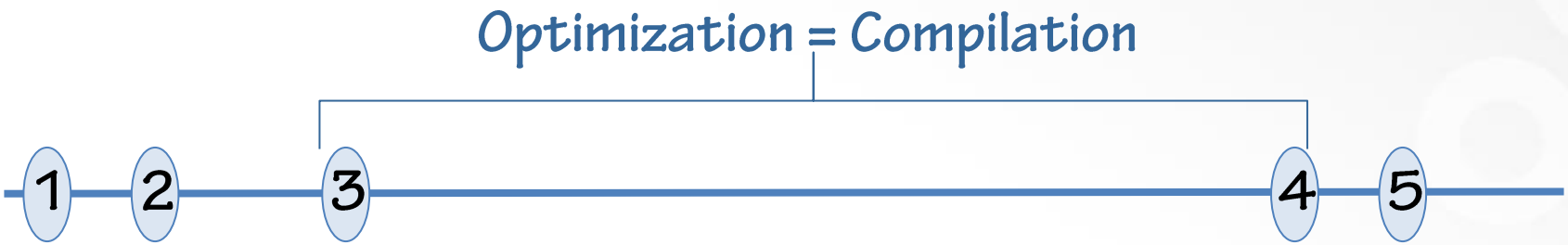
These two behave EXACTLY the same way!

- **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 (*more coming up on this*)

© SQLskills, All rights reserved.
https://www.sqlskills.com

Statement Execution Simplified



Optimization: this is really where you can have the greatest impact and where the most interesting events can occur…

1. Parse
2. Standardization/normalization/algebrization
   ⇒ Query tree (not T-SQL anymore)
3. Cost-based optimization (statistics are used to come up with optimization plan as well as lock granularity)
4. Compilation
5. Execution: at runtime, if the resources don't exist to support the lock level then escalation occurs to TABLE level (by default)

© SQLskills, All rights reserved.
https://www.sqlskills.com

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
```


© SQLskills, All rights reserved.
https://www.sqlskills.com

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

© SQLskills, All rights reserved.
https://www.sqlskills.com

Statement Auto-Parameterization

Much Ado About Nothing!

- **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 by using query_hash and query_plan_hash (more coming up)
- **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**

© SQLskills, All rights reserved.
https://www.sqlskills.com

Understanding sp_executesql

- **Usually used to help build statements from applications**
- **Parameters are typed explicitly**
- **Forces a plan in cache for the parameterized string – 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
- ***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)

© SQLskills, All rights reserved.
https://www.sqlskills.com

Dynamic String Execution (DSE)

- **String is NOT known / 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…**

© SQLskills, All rights reserved.
https://www.sqlskills.com

Query Plans/Plan Cache Problems

- **Plan cache pollution is created by "single-use plans" executing and being stored but you might not really benefit…**
- **Take the following identical query (except for the SARG):**

```
SELECT … FROM member WHERE lastname = 'Tripp'
SELECT … FROM member WHERE lastname = 'Tripped'
SELECT … FROM member WHERE lastname = 'Tripper'
SELECT … FROM member WHERE lastname = 'Falls'
```

- **Each plan takes a multiple of 8KB (the plan above is 24KB – for EACH statement)**
- **This "query class" is harder to track because each is listed in the cache… but, they will all have the same query_hash but possibly not the same query_plan_hash**
- **Query sys.dm_exec_query_stats and aggregate over query_hash to see how many distinct plans (query_plan_hash) a particular query_hash might have in the cache… if there's only one plan then the statement may be stable (even though SQL Server won't deem it as safe)**
  - However, this is completely data dependent (and, have you tried EVERY possibility?)

© SQLskills, All rights reserved.
https://www.sqlskills.com

The "Cumulative Effect" of Queries

- **SQL Server 2008 added query_hash and query_plan_hash to sys.dm_exec_query_stats**
  - Aggregate by query_hash to find similar queries
  - Aggregate by query_hash, query_plan_hash to find similar queries along with their plans (possibly more than one per query_hash which is why the statement wasn't safe)
- **SQL templatizes the parameters – similar to sp_get_query_template**
- **See the queries in the demo scripts…**
- **Check out BOL: Finding and Tuning Similar Queries by Using Query and Query Plan Hashes**
  - http://bit.ly/QfUmRY

© SQLskills, All rights reserved.
https://www.sqlskills.com

Plan Cache Size 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 2008+ and SQL Server 2005 SP2
    - 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?

© SQLskills, All rights reserved.
https://www.sqlskills.com

Verifying Plans in Cache NOW

- Use syscacheobjects (in SQL Server 2000 and 2005)
  - SELECT * FROM master.dbo.syscacheobjects
  - This lists plans in cache as well as parameterized plans
  - usecounts and refcounts are interesting in terms of frequency of execution
  - pagesused gives insight into the size of the plan
- **Use DMVs to see the same and more in SQL Server 2005+**
  - DMV: sys.dm_exec_query_stats – gives much of what you see from syscacheobjects
  - DMV: sys.dm_exec_procedure_stats – cumulative costs tied to frequency of executions!
  - DMF: sys.dm_exec_sql_text(sql_handle) – returns the actual text but only when needed (if just obj stats then it's faster not to include text)
  - DMF: sys.dm_exec_query_plan(plan_handle) – can give you the XML Showplan as well and there's no prior equivalent (might be better to use query store for history as this only gives LAST plan)

© SQLskills, All rights reserved.
https://www.sqlskills.com

Plan Cache

- **CACHESTORE_OBJCP = "Object Plans“**
  - Stored procedures, functions, triggers…
  - Generally, desirable
- **CACHESTORE_SQLCP = "SQL Plans"**
  - Ad hoc SQL statements (including parameterized ones)
  - Prepared statements
  - OK when highly re-used but often not reused…
- **CACHESTORE_PHDR = "Bound Trees"**
  - Views, constraints and defaults
  - Irrelevant here, usually OK
- **CACHESTORE_XPROC = extended procs**

© SQLskills, All rights reserved.
https://www.sqlskills.com

Clearing Plans from Cache

- **DBCC FREEPROCCACHE [({ plan_handle | sql_handle | pool_name })] [WITH NO_INFOMSGS]**
- **All plans**
  - DBCC FREEPROCCACHE
- **A single plan (using plan_handle)**
- **A single database's plans**
  - DBCC FLUSHPROCINDB(DBID)
  - New in SQL Server 2016:

```
ALTER DATABASE SCOPED CONFIGURATION
CLEAR PROCEDURE_CACHE;
```

- **Ad hoc plans**
  - In 2005+ use: DBCC FREESYSTEMCACHE('SQL Plans')
  - In 2008+ use: sp_configure 'optimize for ad hoc workloads', 1

© SQLskills, All rights reserved.
https://www.sqlskills.com

hidden slide
w/extra details

Plan Cache "Buckets / Entries" Limits

- **It's UNLIKELY (especially with cache management / clearing) but for some systems, you can run into an additional problem**
  - When too many concurrent inserts occur in the same hash bucket or **the ad hoc SQL Server plan cache hits its entry limit of 160,036**, severe contention on SOS_CACHESTORE spinlock occurs. In this situation, a high CPU usage occurs in Microsoft SQL Server 2012 or SQL Server 2014
- **Default number of buckets is 40,009**

```
SELECT [cc].[name], [cc].[type], [cc].[entries_count]
FROM [sys].[dm_os_memory_cache_counters] AS [cc]
WHERE [cc].[name] = 'SQL Plans'
```

- **SQL2012SP1CU12 & SQL2014CU6 -> Trace flags 174 & 8032 required to increase total buckets to 160,036**
- **See FIX: SOS_CACHESTORE spinlock contention on ad hoc SQL Server plan cache causes high CPU usage in SQL Server 2012 or 2014**
  - http://support.microsoft.com/kb/3026083

© SQLskills, All rights reserved.
https://www.sqlskills.com

Reducing Plan Cache Pollution

- **Server setting: Optimize for ad hoc workloads**
  - On first execution, only the query_hash will go into cache
    - For most queries, this is a matter of a few hundred bytes (compared to multiples of 8KB for the plan)
  - On second execution (if), the plan will be placed in cache
- **Create a single and more consistent plan with covering indexes; SQL Server might be able to see covered statements as safe!**
  - A lot of limitations to SQL Server detecting this as safe (see Appendix A of the *Plan Caching in SQL Server 2008* whitepaper) but if the plans are actually stable
    - Force stable statements into the cache using sp_executesql
- **Periodically, have a job that wakes up to check the single-use, plan cache bloat and then clears the "SQL Plans" cache (if over 2GB, for example)**
  - Be sure to review my blog category on Plan Cache: http://bit.ly/1eqNP9H
  - And, specifically, this post: http://bit.ly/Rj0MIP

© SQLskills, All rights reserved.
https://www.sqlskills.com

Multiple Plans (Tipping/Covering)

- **High-priority queries and queries that are executed often – need to reduce their cumulative effect on the server**
- **By covering a query you can reduce the number of possible plans that exist for a query and you might make a statement's plan STABLE**
  - Reduces I/Os required to access the data the query needs
  - Reducing [often, drastically] I/Os translates into:
    - Less time

      Important note: it's NOT just about I/Os – there are cases where plans with fewer I/Os are actually MORE expensive because of other operations (like sorts, worktables, etc.) so be careful not to get too wrapped up in just I/Os. Review the showplan and duration as well!
    - Fewer pages in cache results in better cache efficiency
  - SQL Server might deem the statement as safe (but, only when it meets all of the other requirements)
    - *Consider* forced parameterization (NOTE: I'd still recommend procedures as they're more granular than setting the database to forced parameterization)

© SQLskills, All rights reserved.
https://www.sqlskills.com

Forced Parameterization

- **Generally, not recommended**
  - Can investigate plan stability by using query_hash and query_plan_hash to determine if forced parameterization is possible
    - If every query_hash has only one distinct query_plan_hash then your plans are stable
    - There are really a couple of scenarios that you might be in – check out the next couple of slides!
- **Many more statements are forced to be cached**
  - Some statements cannot be forced with FORCED parameterization but could be "forced" with stored procedures. So, you really need to know what's the best option for that access method (depending on the stability of the plans)
  - If you have some statements that really aren't safe, you could end up executing bad plans… better to "get it right" from the start and control it yourself with stored procedures
- **NOTE: If you're going to consider the FORCED parameterization database setting – be sure to watch my Pluralsight course: SQL Server: Optimizing Ad Hoc Statement Performance: http://bit.ly/1nmYjHq**

© SQLskills, All rights reserved.
https://www.sqlskills.com

Reducing Plan Cache Pollution (2)

Two Primary Scenarios to Consider

- **Analyze the plan cache for the number of query plans per query hash (as well as the number of executions)**

```
SELECT qs.query_hash
    , COUNT(DISTINCT qs.query_plan_hash) AS [Distinct Plan Count]
    , SUM(qs.EXECUTION_COUNT) AS [Execution Total]
FROM sys.dm_exec_query_stats AS qs
   CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS st
   CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
WHERE st.text LIKE '%member%'
GROUP BY qs.query_hash
ORDER BY [Execution Total] DESC
```

Scenario 2

Generally, more likely…

Scenario 1

query_hash	# Plans	# Executions
0x04BB791B589774AD	1	6456456
0x1706E9EC3049A95B	6	276543
0x5BD9FF487079B335	1	124345
0x6604520C5200ABC0	1	78905
0x77BA5A89C7EBE605	1	14342
0xA078B4BC8768A9A6	1	4567
0xB81E270A58A79D16	1	6

Mostly stable plans (only 1 plan per hash)

Scenario 2

query_hash	# Plans	# Executions
0x04BB791B589774AD	34	6456456
0x1706E9EC3049A95B	1	276543
0x5BD9FF487079B335	8	124345
0x6604520C5200ABC0	3	78905
0x77BA5A89C7EBE605	2	14342
0xA078B4BC8768A9A6	9	4567
0xB81E270A58A79D16	6	24

Mostly UNstable plans (multiple plans per hash)

© SQLskills, All rights reserved.
https://www.sqlskills.com

Reducing Plan Cache Pollution (3)

- **Scenario 2: Default database parameterization mode: SIMPLE**
  - Use "templatized" plan guides to take the few statements that are stable and make them forced (reduced compilation/CPU)

```
EXEC sp_create_plan_guide
    Name_of_plan_guide
     templatized_version_of_stable_query,
     N'TEMPLATE',
     NULL,
     @Parameters,
     N'OPTION(PARAMETERIZATION FORCED)';
```

- **Scenario 1: Consider changing database parameterization to FORCED**
  - Use "templatized" plan guides to take the few statements that are NOT stable and make SIMPLE (recompiled)

```
EXEC sp_create_plan_guide
    Name_of_plan_guide
     templatized_version_of_unstable_query,
     N'TEMPLATE',
     NULL,
     @Parameters,
     N'OPTION(PARAMETERIZATION SIMPLE)';
```


© SQLskills, All rights reserved.
https://www.sqlskills.com

Stored Procedures & sp_executesql and the Cache

- **Stored procedures and sp_executesql work the same way**
  - Literals and parameters can be optimized
    - Literals inside the procedure CAN leverage features like filtered indexes and filtered statistics
    - Parameters can go through "sniffing" and optimization for the specific value but they cannot use filtered objects for fear of subsequent execution failures
  - Variables are deemed unknown
    - Variables are defined at runtime through the statements of the procedure and are unknown until runtime BUT SQL Server optimizes the statements at compilation… how?
      - The values are NOT sniffed
      - The histogram is NOT used
      - The "average" is used; the average comes from the density_vector portion of the statistics
- **Parameter sniffing is generally good but if ALL of the parameters don't look the same or work the same way – then it can be horribly bad!**

© SQLskills, All rights reserved.
https://www.sqlskills.com

Key Takeaways: Part I (1 of 2)

- **If the statement wildly varies:**
  - An ad hoc statement will work well
  - A procedure may offer more benefits/possibilities
- **If the statement produces a stable plan – regardless of parameter values:**
  - Use sp_executesql
  - Use a stored procedure
- **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 <u>stored procedures</u>**
  - Written by database developers that should:
    - Know the data/workload/requirements
    - Know how SQL Server works

NOTE: The same recompilation options are available for statements executed by sp_executesql but it's unlikely that the generation method using sp_executesql is "smart" enough, per se, to leverage these options.

© SQLskills, All rights reserved.
https://www.sqlskills.com

Key Takeaways: Part I (2 of 2)

- Analyze your plan cache to see if you have ad hoc "bloat"
- 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 fix the code!

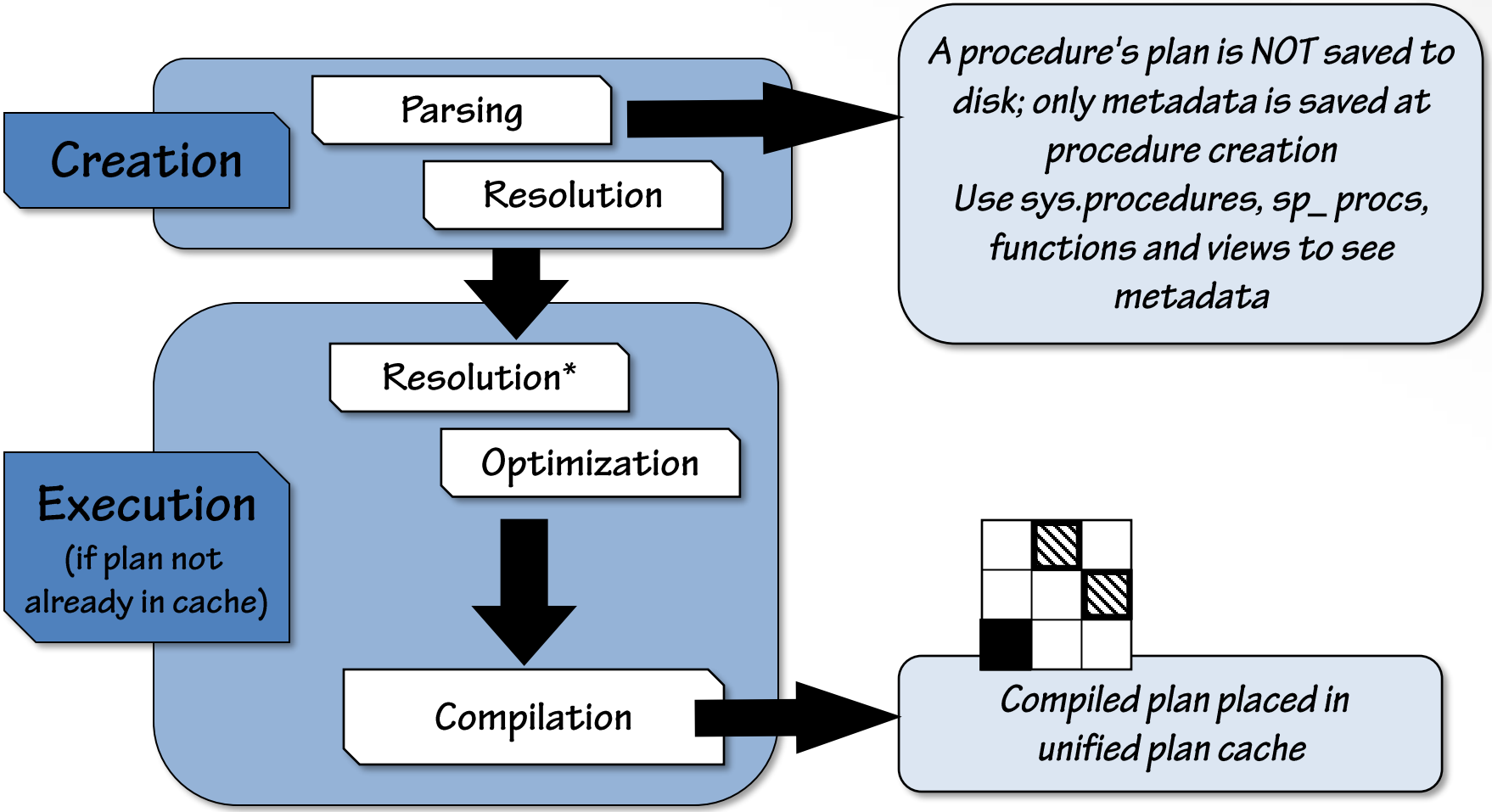
© SQLskills, All rights reserved.
https://www.sqlskills.com

Overview

- Understanding plan cache
  - Statement execution
    - sp_executesql
    - Dynamic string execution
  - Query fingerprints ⇒ the cumulative effect
  - Reducing plan cache pollution
- **Understanding/optimizing stored procedures**
  - Stored procedure creation
  - Stored procedure coding best practices for performance
  - Stored procedure execution
  - Stored procedure recompilation/optimization

© SQLskills, All rights reserved.
https://www.sqlskills.com

Processing Stored Procedures



© SQLskills, All rights reserved.
https://www.sqlskills.com

Stored Procedure Optimization

- **Plan is generated when no plan already exists in cache**
- **Plans are never saved on disk and can "fall out" of cache**
  - Forced out through:
    - Server restart
    - DBCC FREEPROCCACHE
    - DBCC FLUSHPROCINDB
    - sp_recompile
  - Aged out through non-use
  - Schema of base object changes (sp_recompile TNAME… SCH_M)
  - Statistics of base objects change
    - See next slide for details specific to version
- **When a plan exists in cache, all subsequent executions use that plan**
- **Plans are *not* regenerated for subsequent executions (even when statistics are added – use sp_recompile after adding statistics)**

© SQLskills, All rights reserved.
https://www.sqlskills.com

hidden slide
w/extra details

Plan Invalidation

A Painful History

- **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 thus increasing the importance for "*only when data has changed*" logic in maintenance routines.

© SQLskills, All rights reserved.
https://www.sqlskills.com

Procedure Caching

Isn't That the Point?

- **Reusing plans can be good**
  - When different parameters don't change the optimal plan, then saving and reusing is excellent!
  - SQL Server saves time in compilation
- **Reusing plans can be VERY bad**
  - When different parameters wildly change the size of the result set and the optimal plans vary, then reusing the plan can be horribly bad
    - Don't worry, I'll show you how to see this…
    - Don't worry, I'll show you the plethora of options to control/fix this!
  - If statistics are added to base tables, existing plans may not leverage them
    - Don't worry, there's a simple solution here:
      - `EXEC sp_recompile <tablename>`

NOTE: This requires a SCH_M lock and as a result, could create long blocking chains. Check out the `PreventLongBlockingChains.sql` script.

© SQLskills, All rights reserved.
https://www.sqlskills.com

SQL Server "Optimizing the Process of Optimization" Resulting in an Inoptimal Plan

- **To simplify the process of optimization, SQL Server optimizes all statements that are optimizable (at compilation)**
  - Parameters are known (because they initiated the execution)
  - Literals are known (because they're hard-coded in the logic of the procedure)
  - Variables won't be known until the logic of the procedure is executed (which is after we optimize / compile)
  - Conditional logic / branching is unknown until runtime
- **All statements that CAN be optimized, WILL be optimized – based on the information that they can interrogate**
  - Parameters and literals can be sniffed (SQL Server can use the histogram)
  - Variables cannot be sniffed (SQL Server has to use the density_vector)

© SQLskills, All rights reserved.
https://www.sqlskills.com

Optimization through Parameter Sniffing

- **Parameters and literals are known**
  - As a result, they are evaluated [e.g. "sniffed"]
- **The problem is that some procedures (in actuality, many) are "sensitive" to the different parameters passed**
  - If the optimal plan *should* change with different parameters then your procedure is said to be parameter sensitive – which leads to problems (PSP)
    - PSP = Parameter Sniffing Problems *or, sometimes* Parameter Sensitivity Problems
- **Disabling parameter sniffing has been an option for a while**
  - At the parameter-level
    - Using a variable in place of a parameter obfuscates the value passed and results in an "unknown" value (for that parameter) used for optimization
  - At the statement-level: `OPTION (OPTIMIZE for UNKNOWN)`
    - Tells SQL Server NOT to use the parameters coming in and instead optimize for the "average" density using the density_vector portion of the statistic
  - Server-wide trace flag: 4136
    - http://support.microsoft.com/kb/980653

© SQLskills, All rights reserved.
https://www.sqlskills.com

Disabling Parameter Sniffing in SQL Server 2016

- **Database-level configuration options**
- **Executed in the context of the database that you're changing – means that it's SCOPED to that database**

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

- **If the database in an Availability Group then you can set the secondary with the same or different configuration from the primary**
  - Enable or disable option for secondary

```
ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY
SET PARAMETER_SNIFFING = ON;
```

  - Set option for secondary to be the same as primary

```
ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY
SET PARAMETER_SNIFFING = PRIMARY;
```


© SQLskills, All rights reserved.
https://www.sqlskills.com

Recompilation Issues

RECOMPILATION = OPTIMIZATION

OPTIMIZATION = RECOMPILATION

- When do you want to recompile?
- What options do you have for recompilation – and at what granularity?
- How do you know you need to recompile?
- Do you want to recompile the entire procedure or only part of it?
- Can you test it?

© SQLskills, All rights reserved.
https://www.sqlskills.com

When to Recompile?

- **When the plan for a given statement within a procedure is not consistent in execution plan, due to parameter changes**
- **Cost of recompilation might be significantly less than the execution cost of a bad plan!**
- **Why?**
  - MUCH faster execution with a better plan
  - Some plans just don't work for a wide variety of your execution cases, in fact, some plans should NEVER be saved
- **Do you want to do this for every procedure?**
  - **No, but start with the highest priority/expensive procedures that aren't performing well first – and test!!**

© SQLskills, All rights reserved.
https://www.sqlskills.com

Options for Recompilation

- `CREATE … WITH RECOMPILE`
- `EXECUTE … WITH RECOMPILE`
- `sp_recompile objname`
- **Statement-level recompilation**
  - The 2000+ way (still has benefits)
    - Dynamic string execution (statement isn't stored with proc plan)
    - Modularized code (breaks the statement/block into its own proc)
  - The new way
    - 2005+: `OPTION(RECOMPILE)`
    - 2005+: `OPTION (OPTIMIZE FOR (@variable_name = constant, ...) )`
    - 2008+: `OPTION (OPTIMIZE FOR UNKNOWN)`
    - Special case where you want some values to be unknown
      - 2008+: `OPTION (OPTIMIZE FOR (@p1 UNKNOWN, @p2 UNKNOWN))`

© SQLskills, All rights reserved.
https://www.sqlskills.com

CREATE … WITH RECOMPILE

- **Causes the entire procedure's plan to be recompiled on every execution**
- **Since SQL Server 2005 introduced statement-level recompilation, you should only use this for SMALL procedures**
- **When procedure returns widely-varying results**
- **When the plan is not consistent**
- **For backward compatibility**
  - 2000: has only complete procedure recompiles (KB 263889 Compile Locks)
  - 2005: statement-level recompilation
- **Always target the smallest amount possible to recompile!**
- **NOTE: If a procedure is created WITH RECOMPILE, the statement(s) won't show up in dm_exec_query_stats OR dm_exec_procedure_stats (it does place the query in query store but overall, very limited troubleshooting capabilities; avoid using)**

© SQLskills, All rights reserved.
https://www.sqlskills.com

EXECUTE WITH RECOMPILE

- **Recompiles the plan only for the execution**
  - Does not affect the plan in cache
  - Does not place this plan in cache for subsequent executions
  - Does put the statement in query store!
- **Excellent for testing**
- **Possibly for "atypical" execution scenarios**
- **Verify plans for a variety of test cases**

```
EXEC dbo.GetMemberInfo 'Tripp' WITH RECOMPILE
EXEC dbo.GetMemberInfo 'T%' WITH RECOMPILE
EXEC dbo.GetMemberInfo '%T%' WITH RECOMPILE
```

- **Do the execution plans match?**
  - Yes: create the procedure normally
  - No: determine what should be recompiled

© SQLskills, All rights reserved.
https://www.sqlskills.com

Statement-Level Recompilation

- **In 2005 and 2008+: "inline" recompilation for statements**
  - `OPTION (RECOMPILE)`
    - Excellent when parameters cause the execution plan to widely vary
    - Bad because EVERY execution will recompile – but only for the statements
  - `OPTION (OPTIMIZE FOR (@variable = literal, ...))`
    - Excellent when large majority of executions generate the same optimization time
    - You don't care that the minority may run slower with a less than optimal plan?
  - 2008+ only: `OPTION (OPTIMIZE FOR UNKNOWN)`
    - Use the "all density" (average) instead of the histogram
- **The old way: modularizing your code**
  - Doesn't hurt!
  - Better for statement blocks/code reuse
  - SQL Server never steps into a procedure until it's executed

© SQLskills, All rights reserved.
https://www.sqlskills.com

Modularization

- **Be careful of block statements/conditional logic**
  - SQL Server optimizes the process of optimization and this may lead to an less-than-optimal plan (no, really!)
- **Breaking the stored procedure into smaller chunks**
  - Can handle the block of statements on a more granular basis
  - Subprocedures can be optimized/compiled
    - AVOID: `CREATE subprocedure WITH RECOMPILE`
    - CONSIDER: `EXECUTE subprocedure WITH RECOMPILE`
    - BEST: Statement-level execution options
      - More options for control
      - Better monitoring / tracking: these **can** be tracked in the DMVs through sys.dm_exec_query_stats

© SQLskills, All rights reserved.
https://www.sqlskills.com

When Is Recompilation Required

- **Specific types of code cannot support a single plan or perform well with one that's cached**
  - Filtered Objects (indexes and statistics)
    - SQL Server cannot save a single plan based on filtered objects
    - SQL Server won't use a filtered index /stats unless recompilation is performed:
      - Using OPTION (RECOMPILE): this is the most safe as it forces the statement to be recompiled and is not affected by the database setting for parameterization
      - Using a dynamically constructed string (unless forced parameterization is on)
  - Partitioned Objects (tables and views)
    - SQL Server CAN do partition elimination correctly (even if the first execution is only against one partition); you will get the correct data
    - However, the first execution will set the "statistics" for the plan – which could vary across the sets so the plan might not be optimal
  - CHOOSE
    - This is similar in that you will get the correct data but subsequent executions will have estimation issues

© SQLskills, All rights reserved.
https://www.sqlskills.com

Multi-Purpose Procedures

- Stored procedures with *n* parameters where any combination of these parameters can be supplied
- Often the code looks like this:

```
WHERE (column1 = @variable1 OR @variable1 IS NULL)
AND (column2 = @variable2 OR @variable2 IS NULL)
 …
AND (columnN = @variableN OR @variableN IS NULL)
```

- These are very difficult for the optimizer to "generalize" and what often happens is a generally bad plan
- How do you fix it?
  - Concatenate ONLY when the variable is NOT NULL
  - Define the data type of the variable in the string. Use: column = convert(datatype, @variable) in actual string (to reduce implicit conversions and plan cache bloat)
  - Use `sp_executesql` to force the statement caching but ADD conditional logic to add `OPTION (RECOMPILE)` for parameters that are not stable!

© SQLskills, All rights reserved.
https://www.sqlskills.com

Ad Hoc Statements and Statement Auto-Parameterization

- **EXEC (@str) = ad hoc statements, these can:**
  - Be cached when there are no parameters: subsequent executions must be an exact textual match (case-sensitive regardless of db setting and spaces)
  - Be parameterized and deemed safe
    - Subsequent executions will use the plan in cache
  - Be parameterized and deemed unsafe (MOST LIKELY)
- **sp_executesql = forced statement caching**
  - The first execution will be "sniffed" and the plan will be placed in cache for subsequent executions, regardless of whether or not the plan is good/consistent for all values
  - Great when the plan is consistent!
  - Can be horrible when it's not!!

© SQLskills, All rights reserved.
https://www.sqlskills.com

Summary: Stored Procedure Pitfalls/Performance

- **Stored procedures and sp_executesql have the same potential for executing a bad plan but stored procedures have more options for *centralized* control**
- **Forcing a recompile can be warranted/justified**
- **Always recompile the smallest amount possible!**
- **But, can you have too many recompiles?**
  - Yes: however, it's not as bad as 2000 because only the statement is recompiled (as of 2005+), not the entire procedure
    - What was the reason for recompilation?
      - Check out Pinal Dave's post: http://bit.ly/1pNloHr
  - Yes: however, sticking to these best practices can help to **DRASTICALLY** minimize that!

© SQLskills, All rights reserved.
https://www.sqlskills.com

Key Takeaways: Part III

- **Teach your developers that stored procedures SHOULD be used BUT that they have pros and cons**
- **Some stored procedures can work well without any special coding practices or strategies; however, for most environments this is the MINORITY of procedures**
- **Procedures should be tested with a two-phase approach**
  - Phase I: code coverage
  - Phase II: plan stability
- **Analyze cumulative the cost of your procedures to prioritize code changes in an existing environment**
  - Use sys.dm_exec_procedure_stats focusing on:
    - TIME: total_elapsed_time
    - CPU: total_worker_time

© SQLskills, All rights reserved.
https://www.sqlskills.com

Review

- **Understanding plan cache**
  - Statement execution
    - sp_executesql
    - Dynamic string execution
  - Query fingerprints ⇒ the cumulative effect
  - Reducing plan cache pollution
- **Understanding/optimizing stored procedures**
  - Stored procedure creation
  - Stored procedure coding best practices for performance
  - Stored procedure execution
  - Stored procedure recompilation/optimization

© SQLskills, All rights reserved.
https://www.sqlskills.com

- **Watch this first – SQL Server: Optimizing Ad Hoc Statement Performance**
  - http://pluralsight.com/training/Courses/Description/sqlserver-optimizing-adhoc-statement-performance
  - Just over 7 hours on ad hoc statement execution and plan caching
- **Then, watch this – SQL Server: Optimizing Stored Procedure Performance**
  - http://pluralsight.com/training/Courses/Description/sqlserver-optimizing-stored-procedure-performance
  - Just over 7 hours on procedure caching and recompilation
- **Then, watch part 2 – SQL Server: Optimizing Stored Procedure Performance**
  - Just over 3 hours… session settings and execution context!

© SQLskills, All rights reserved.
https://www.sqlskills.com

Optimizing Procedural Code Posts and PASStv

- **Start with this blog post (and sample code):**
  - Building High Performance Stored Procedures
  - https://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/

© SQLskills, All rights reserved.
https://www.sqlskills.com

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.mspx
- PSS SQL Server Engine Blog
  - http://blogs.msdn.com/psssql/default.aspx
- KB 243586: Troubleshooting Stored Procedure Recompilation

© SQLskills, All rights reserved.
https://www.sqlskills.com

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
    - https://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

© SQLskills, All rights reserved.
https://www.sqlskills.com

Questions?