SQL Server: Optimizing Ad Hoc Statement Performance

Module 6: Statement Execution Summary

Kimberly L. Tripp
Kimberly@SQLskills.com
http://www.SQLskills.com/blogs/Kimberly

pluralsight
hardcore developer training

Course Overview

- **Statement execution methods**
- **Estimates and selectivity**
- **Statement caching**
- **Plan cache pollution**
- **Statement execution summary**
  - Statement execution, estimates, and caching
  - Statement execution methods, caching, and concerns
  - Statement execution solutions

Statement Execution, Estimates, and Caching (1)

- **Ultimately it comes down to these points:**
  - Some statements have a consistent, stable plan and that plan should be cached for reuse
    - This reduces execution time and eliminates compilation (reducing CPU)
    - But, SQL Server will NOT cache all statements (even if they are stable)
  - Some statements do not have a consistent, stable plan and reusing a cached plan might be worse than compiling a new plan
    - This requires additional CPU/compilation but might result in a significantly reduced execution time
    - But, SQL Server will NOT know to kick a plan out of cache (even if it's unstable)
- **The key point will be:**
  - Knowing when to use each method
  - Knowing how to rewrite code to benefit
  - Knowing how to control an environment where you can't rewrite the code

Statement Execution, Estimates, and Caching (2)

- **Knowing how these things all work together is critical**
- **How you execute SQL statements has an impact on performance**
  - Do you just send the statement to SQL Server?
  - Do you pass it to *sp_executesql*
  - Do you use *EXEC (@string)*
  - Does the statement have:
    - Literals?
    - Variables?
    - Parameters?
- **When is the plan for a statement defined/chosen?**
  - Can the values be “sniffed”
  - Or, are they unknown?
  - Or, are they defined by a prior execution because we’re using a cached plan

Bringing It All Together

- **Ad hoc cache management must be done (no matter what)**
- **Ad hoc/generated statements are not ideal**
  - Can create plan cache pollution
  - Requires compilation/CPU due to lack of parameterization (most statements are not deemed safe)
  - Different parameters often warrant different plans – for the best performance
  - Only guaranteed reuse are exact textual matches
- **For better plan re-use consider:**
  - Forced statement caching through *sp_executesql*
    - Do your application developers know when and how to use this appropriately?
  - Stored procedures
    - For better plan re-use, flexibility, and centralized logic/control use procedures
    - With statement-level recompilation techniques based on server-side knowledge, testing, and the characteristics of the data (this is best from back-end, database developer experience and knowledge)

Statement Execution Methods, Caching, and Concerns

Statement Execution Method	Database Setting	Plan Categorization	Ad Hoc Plan Cache	Compiled Plan Cache	Ad Hoc Plan Cache Pollution*	Prepared Plan Cache Bloat	PSP	SQL Injection
Ad hoc	SIMPLE	unsafe	each individual execution	no	yes			no
Ad hoc	SIMPLE	safe	each individual execution	parameterized (auto)	yes	yes, plan per data type**		no
Dynamic string execution	SIMPLE	unsafe	each individual execution	no	yes			possible
Dynamic string execution	SIMPLE	safe	each individual execution	parameterized (auto)	yes	yes, plan per data type**		possible
sp_executesql	n/a	unstable	none	parameterized (forced)	no	no, strongly-typed	yes	less likely; depends on how you build the statement
sp_executesql	n/a	stable	none	parameterized (forced)	no	no, strongly-typed		less likely; depends on how you build the statement
Ad hoc	FORCED	still unsafe (not forced)	each individual execution	no	yes			no
Ad hoc	FORCED	unsafe but forced	each individual execution	parameterized (forced)	yes	no***	yes	no
Ad hoc	FORCED	safe	each individual execution	parameterized (forced)	yes	no***		no
Dynamic string execution	FORCED	still unsafe (not forced)	each individual execution	no	yes			possible
Dynamic string execution	FORCED	unsafe but forced	each individual execution	parameterized (forced)	yes	no***	yes	possible
Dynamic string execution	FORCED	safe	each individual execution	parameterized (forced)	yes	no***		possible

* Ad hoc plan cache pollution is reduced by the server setting for ‘*optimize for ad hoc workloads*’

** Can reduce prepared plan cache pollution by explicitly converting the parameters to the column type

*** See the Books Online topic: Forced Parameterization (Data Types of Parameters)

Statement Execution Solutions (1)

- **First step: set *optimize for ad hoc workloads***
- **Second step: evaluate your plan cache for ad hoc statements and take action based how the optimization plan is categorized:**
    - Statement is deemed “safe”
    - Statement is deemed “unsafe” and has a stable plan
    - Statement is deemed “unsafe” and has an unstable plan
- **Final step: set up a job to clear the ‘SQL Plans’ plan cache when more than 2GB is being used for single-use plans**
    - Scanning the compiled plan stubs can be expensive, don’t let the chain get too large
    - Even with all of the optimizations, it is likely that you will still have some plan cache pollution (if not, the job won’t have to do anything)

Statement Execution Solutions (2)

- **If SQL Server defines the statement as "safe":**
  - Nothing do to here as SQL Server parameterizes it
  - Better to use *sp_executesql* to directly reduce ad hoc plan cache pollution
  - If you stabilize plans with better indexes, SQL Server might pick up more (of the simple) statements as "safe"
- **If SQL Server defines the statement as "unsafe", but the parameters do not require a plan change (i.e. a stable plan):**
  - Covering indexes often lead to better plan stability…
  - If you can change the code – use *sp_executesql*
  - If you can't change the code – consider a plan guide template to force the plan (*PARAMETERIZATION FORCED*)
    - **NOTE:** Not all statements can be used in a plan guide template (for example, *LIKE*) even if the parameters are consistent enough to generate the same plan.
  - Generally, avoid changing the database setting for *PARAMETERIZATION* unless the large majority of statements over your business cycle are stable plans with large numbers of executions (test, test, test!)

Statement Execution Solutions (3)

- **If SQL Server defines the statement as "unsafe" and the parameters supplied require plan changes (i.e. unstable plan):**
  - Do not use *sp_executesql*, leave it as ad hoc
  - Or, if using *sp_executesql* – consider adding *OPTION (RECOMPILE)*
    - **NOTE:** There are other methods for dealing with various PSP patterns but they're a bit beyond the scope of this course. I will discuss in great detail how to deal with PSP in my next course: *SQL Server: Optimizing Stored Procedure Performance*
  - For ad hoc statements, nothing to do unless the database is set to *FORCED*, if so:
    - Change the code – do not use an ad hoc statement, use *sp_executesql* and *OPTION (RECOMPILE)*
    - If you can't change the code – use a plan guide template to recompile the plan (*PARAMETERIZATION SIMPLE*)
    - **NOTE:** If you turn on the server setting: *optimize for ad hoc workloads* as well as the database option for parameterization: *FORCED* and you don't have a lot of executions you can create more plan cache pollution by placing the forced plan at first execution as well as the compiled plan stub

Summary: Statement Execution

- **Not every statement is created equal**
- **Not every statement should be executed exactly the same way**
- **Knowing the different ways to execute a statement might:**
    - Reduce CPU and compilation time
        - When the statement is stable, parameterized, and/or "safe"
    - Increase CPU and compilation time
        - When it must be recompiled every time (regardless of whether or not it's safe and stable or unstable and unpredictable)
    - Reuse a plan and get better performance
        - No compilation – just execute the plan and go!
    - Reuse a plan and get worse performance
        - Not all plans work well for all executions…
- **Sometimes a statement is best as an ad hoc / dynamic statement**
- **Sometimes a statement is best passed through *sp_executesql***
- **Neither is ALWAYS best; it depends!**

Just the Tip of the Iceberg

- **Data types: column size / row size / consistency**
  - **Released course:** *SQL Server: Why Physical Database Design Matters*
- **Ad hoc statements: plan cache / parameter sniffing**
  - **THIS COURSE**
- **Stored procedures: parameter sniffing / recompilation**
  - **Coming next:** *SQL Server: Optimizing Stored Procedure Performance*
- **Indexes: creation / overhead / maintenance**
  - **Coming soon:** *multiple courses on index internals, data access patterns, and indexing strategies*
- **Queries: predicates / functions / *WHERE* clause vs. *FROM* clause / isolating expressions**
- **Statistics: accuracy / cardinality estimation / skewed data / histogram limitations / uneven distribution and correlated columns**

Where To Go Next?

- **Check for new Pluralsight courses from me**
  - I'm going to stay within the developer/database development area for my first few courses
    - Targeting best practices and typically using a "problem/solution" approach
- **Check out these SQLskills courses on Pluralsight that are the most appropriate courses for you to consider next:**
  - *Developing and Deploying SQL Server ISV Applications*
  - *SQL Server: Common Performance Issue Patterns*
  - *SQL Server: Troubleshooting Query Plan Quality Issues*
- **Everyone using SQL Server should watch Paul Randal's course: *SQL Server: Myths and Misconceptions***
  - You'd be surprised at how many of these you might think you know
  - It gives you all sorts of great advice – across the entire product!

Course Summary

- **Performance doesn't just "happen"**
- **Do not just expect the SQL Server defaults to perfectly support every environment**
    - It's not one-size-fits-all!
- **The effect on performance of using only one method to submit all of your data requests can be huge**
    - Use the right method for the right request (and right data pattern)
    - Caching isn't ***always*** good… it isn't ***always*** bad either
    - Knowing what works and how to test it is the key to good statement execution as well as reducing plan cache pollution and CPU!
- **Knowing this information about ad hoc statements, caching, and estimates (statistics and heuristics) will help you write better stored procedures**
- **Thanks for watching!**