SQLintersection

Session: Wednesday, 1:30pm-2:30pm

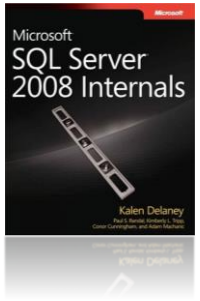
Statement Execution and the Plan Cache

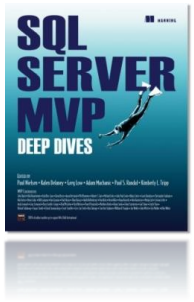
Kimberly L. Tripp
President / Founder, SQLskills.com
Kimberly@SQLskills.com
@KimberlyLTripp

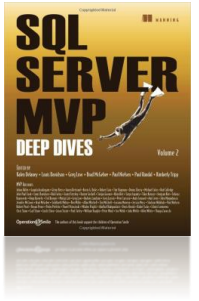
SQL *intersection*

SQLskills
immerse yourself in sql server

Author/Instructor: Kimberly L. Tripp







- Consultant/Trainer/Speaker/Writer
- President/Founder, SYSolutions, Inc.
  - e-mail: Kimberly@SQLskills.com
  - blog: http://www.SQLskills.com/blogs/Kimberly
  - Twitter: @KimberlyLTripp
- Author/Instructor for SQL Server Immersion Events
- Instructor for multiple rotations of both the SQL MCM & Sharepoint MCM
- Author/Manager of SQL Server 2005 & 2008 Launch Content
- Author/Speaker at Microsoft TechEd, SQLPASS, ITForum, TechDays, SQLIntersection
- Author of several SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability, SQLCLR for DBAs
- Author/Presenter for more than 25 online webcasts on MSDN and TechNet
- Author/Presenter for multiple online courses at Pluralsight
- Co-author MSPress Title: SQL Server 2008 Internals, the SQL Server MVP Project (1 & 2), and SQL Server 2000 High Availability
- Owner and Technical Content Manager of the SQLintersection conference

SQL intersection

Overview

- Different ways to execute SQL statements
- Some statements can be cached for better reuse
- Plan cache usage / limits
- Query plans / plan cache problems
- Forced parameterization (hidden slide)
- Understanding sp_executesql
- sp_executesql and the cache
- Reducing plan cache pollution
- But, what should we do????

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
  - With literals, parameters, variables
  - Including dynamic strings
  - Including sp_executesql

These behave the same way but some exceptions exist with certain statement types *inside* stored procedures (*more coming up on this*)

Demo

Statement Execution Methods

SQL *intersection*

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

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 SQL Server 2008+
    - 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?

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?)

Demo

Analyzing Query Hash

SQL intersection

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)

sp_executesql and the Cache

- sp_executesql works the same way as stored procedures do
  - Literals and parameters can be optimized
    - Literals inside the procedure CAN leverage features like filtered indexes and filtered statements
    - 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!

hidden slide w/extra details

Forced Parameterization

- 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 "get it 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 to determine if forced parameterization is possible

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

Demo

Plan Cache Pollution

SQL intersection

Reducing Plan Cache Pollution

- Server setting: Optimize for ad hoc workloads
  - On first execution, only the query_hash will go into cache. For the prior query this is only 380 bytes (compared to the 24KB of the plan)
  - On second execution (if), the plan will be placed in cache
- Create a single and more consistent plan with covering indexes – might make the plan more stable!
  - 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…
- For statements that are stable, force the statement into the cache with 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/RjOMIP

What do we do?

- 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 stored procedures
  - 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 in sp_executesql but it's unlikely that the generation method using sp_executesql is "smart" enough, per se, to leverage them

Review

- Different ways to execute SQL statements
- Some statements can be cached for better reuse
- Plan cache usage / limits
- Query plans / plan cache problems
- Forced parameterization (hidden slide)
- Understanding sp_executesql
- sp_executesql and the cache
- Reducing plan cache pollution
- But, what should we do????

Resources

- Pluralsight courses:
  - SQL Server: Optimizing Ad Hoc Statement Performance
    - http://bit.ly/1nmYjHq
  - SQL Server: Optimizing Stored Procedure Performance
    - http://bit.ly/1xwH7F6
- Whitepaper: Plan Caching and Recompilation in SQL Server 2012
  - http://bit.ly/1gWKmKX
- Demo code/samples:
  - SQLskills, Resources, Demo Scripts and Sample Databases
- Questions:
  - You can always reach me at: Kimberly@SQLskills.com

Session Settings and Client Connectivity

From **new** Pluralsight course, Optimizing Stored Procedures – Part 2

SET Options	Required for Perf Features	Default Server Value	SSMS	SQLCMD	SQL Server Agent	Default OLE DB and ODBC Value	Default DB-Library Value	.NET / Your App
ANSI_DEFAULTS [2]	NO	OFF	OFF	OFF	OFF	OFF	OFF	?
ANSI_NULL_DFLT_ON [2]	NO	**OFF**	ON	ON	ON	ON	**OFF**	?
ANSI_NULLS [1, 3]	YES = ON	**OFF**	ON	ON	ON	ON	**OFF**	?
ANSI_PADDING [2, 3]	YES = ON	ON	ON	ON	ON	ON	**OFF**	?
ANSI_WARNINGS [2]	YES = ON	**OFF**	ON	ON	ON	ON	**OFF**	?
ARITHABORT [2]	YES = ON	ON	ON	**OFF**	**OFF**	**OFF**	**OFF**	?
CONCAT_NULL_YIELDS_NULL [2, 3]	YES = ON	**OFF**	ON	ON	ON	ON	**OFF**	?
NUMERIC_ROUNDABORT [2]	YES = OFF	OFF	OFF	OFF	OFF	OFF	OFF	?
QUOTED_IDENTIFIER [1]	YES = ON	**OFF**	ON	**OFF**	**OFF**	ON	**OFF**	?

(1) The only state that's important is how it's set when the stored procedure is CREATED; the runtime setting is irrelevant.

(2) If different than existing plan in cache, will be recompiled and added to the plan cache; performance may vary at execution.

(3) In a future version of SQL Server, these options will always be ON and any applications that explicitly set the option to OFF will produce an error. Avoid using this feature in new development work and plan to modify applications that currently use this feature.

Questions?

SQLskills
immerse yourself in sql server

Don’t forget to complete an online evaluation on EventBoard!

Statement Execution and the Plan Cache

Session by Kimberly L. Tripp

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

SQL *intersection*

Thank you!