

Chicago Suburban SQL Server User Group

7 October 2014

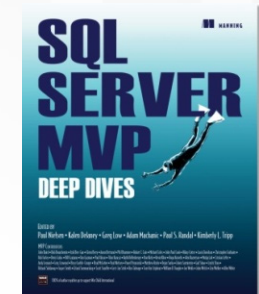
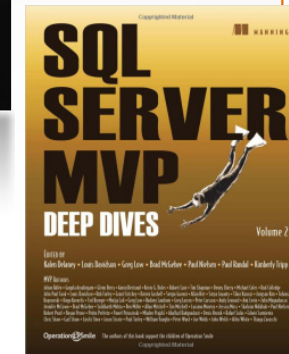
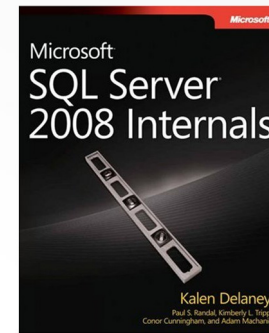
Session 3: Statement Execution/Caching

Kimberly L. Tripp
Kimberly@SQLskills.com



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 TechEd, SQL Connections, SQLPASS, ITForum, Owner & Manager of SQL Intersection
- 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 (two series and other individual webcasts)
- Co-author MPress Title: SQL Server 2008 Internals, the SQL Server MVP Project (1 & 2), and SQL Server 2000 High Availability
- Presenter/Technical Manager for SQL Server HOL DVDs
- I love this stuff... feel free to ask questions!



Overview

- Different ways to execute statements
- Some statements can be cached for reuse
- Statement auto-parameterization
- Dynamic string execution
- `sp_executesql`
- Stored procedures
- The life of a plan in cache
- Plan cache limits
- Demos

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)*

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

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 (check out the Pluralsight course on Statement Execution)
- **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**

Dynamic String Execution (DSE)

- **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

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_dbcmtlevel
 - **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 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, 2008, R2, 2012, 2014
 - 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?

Review

- Different ways to execute statements
- Some statements can be cached for reuse
- Statement auto-parameterization
- Dynamic string execution
- `sp_executesql`
- Stored procedures
- The life of a plan in cache
- Plan cache limits
- Demos



- Email paul@SQLskills.com with the subject line: **Chi Suburban UG Pluralsight code** to get a FREE 30-day trial of our SQLskills content on Pluralsight
- 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

Stored Procedure 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>

Thank you!

