SQL Server: Optimizing Ad Hoc Statement Performance

Module 2: Statement Execution Methods

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

pluralsight
hardcore developer training

Course Overview

- **Statement execution methods**
  - Different ways to execute SQL statements
  - Understanding ad hoc statements
  - Understanding *sp_executesql*
  - Understanding dynamic string execution
  - Dynamic string execution and SQL injection
- **Estimates and selectivity**
- **Statement caching**
- **Plan cache pollution**
- **Statement execution summary**

Different Ways to Execute SQL Statements

- **Ad hoc statements**
  - Possibly, as auto-parameterized statements
- **Dynamic string execution (DSE)**
  - *EXECUTE (@string)*
- ***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*
- **Statements within procedural code**
  - Functions
  - Stored procedures (and triggers)

Understanding Ad Hoc Statements

- **Statement is submitted within a batch (using literals)**

```
SELECT [m].*
FROM [dbo].[member] AS [m]
WHERE [m].[member_no] = 258;
```

- **Statement is submitted within a batch (using variables)**

```
DECLARE @mno INT = 258;
SELECT [m].*
FROM [dbo].[member] AS [m]
WHERE [m].[member_no] = @mno;
```

- **Statements end up being categorized (by SQL Server) as either "safe" or "unsafe"**
- **Whether or not their plan is "stable" is another discussion**
  - Safe statements always have stable plans (but, only relatively simple statements are ever deemed safe)
  - Unsafe statements (the majority) can be stable or unstable plans
- **How SQL Server optimizes and caches them differs (*more on this coming up*)**

Understanding *sp_executesql*

- **Usually used to help build statements from applications**

```
DECLARE @ExecStr NVARCHAR (4000);
SELECT @ExecStr =
N'SELECT [m].*
FROM [dbo].[member] AS [m]
WHERE [m].[member_no] = @mno';
EXEC [sp_executesql] @ExecStr, N'@mno INT', 1234;
```

- **Parameters are explicitly/strongly typed**
- **Forces a plan in cache for the parameterized string**
  - Subsequent executions will *always* use this plan
  - Important note: this is not *always* a good thing…
- **Almost like dynamic string execution, but it's not!**

Understanding Dynamic String Execution

- **String is NOT evaluated until runtime (execution)**

```
DECLARE @ExecStr NVARCHAR (4000)
      , @mno     INT = 258;
SELECT @ExecStr =
N'SELECT [m].* FROM [dbo].[member] AS [m]
WHERE [m].[member_no] = ' + CONVERT(NVARCHAR, @mno);
EXEC (@ExecStr);
```

- **Parameters allow virtually any statement to be built "dynamically"**
- **String can be up to *(n)varchar(max)* in size**
  - SQL Server 2000: had to concatenate multiple variables
  - SQL Server 2005 onward: 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/review, and troubleshoot**
- **And, there's a whole discussion about security and SQL injection**

Dynamic String Execution and SQL Injection

- **First, the parameters**
  - If the parameters are identifiers
    - Consider using *QUOTENAME()* to properly delimit them
  - If the parameters are a simple string
    - Consider using *REPLACE* to properly delimit it
  - Can the parameters be validated?
    - Can you analyze, test, or otherwise guarantee the validity of the values passed?
    - Can you restrict them in the client?
    - Is there some other programmatic test that you can perform?
- **Second, the execution context**
  - Restrict the execution context of the procedure using *EXECUTE AS*
  - Set the context to a **low-privileged user** in the database (ideally, a login-less user that has only the rights needed for the expected string)
- **See my blog post: Little Bobby Tables, SQL Injection and EXECUTE AS**
  - http://bit.ly/V1R3l5

Summary: Statement Execution Methods

- **Ad hoc statements are:**
  - The easiest form of data request – build the statement in the client, submit
  - But, that's only part of the picture – ad hoc statements are the most expensive type of statement for SQL Server to deal with:
    - Usually require compilation on every execution
    - Very unlikely to be cached/reused
    - Data types are not strongly typed
    - Create plan cache pollution
- ***sp_executesql* is often used because:**
  - It's also fairly easy to create/build – build the statement in the client, submit
  - The fallacy is that the plan that gets compiled and reused is always a great thing
  - But, that's only part of the picture – this is what we're going to see/learn:
    - Subsequent executions might suffer
- ***EXEC (@string)* turns the statement into an ad hoc statement**
  - The entire statement is evaluated at runtime
  - The resulting statement behaves exactly as an ad hoc statement does
    - Might be parameterized and cached
    - Likely to be compiled for every execution