Creation, Compilation, and Invalidation

Kimberly L. Tripp
SQLskills.com
@KimberlyLTripp

pluralsight
hardcore dev and IT training

Plan Invalidation

- **From SQL Server 2012 onward, statistics updates only cause plan invalidation IF the data has changed**
- **In prior versions (SQL Server 2005, 2008, 2008 R2)**
  - If database option auto_update_stats is ON
    - Updating statistics runs and always caused plan invalidation
  - If database option auto_update_stats is OFF
    - Updating statistics always runs but does NOT cause plan invalidation
- **Recommendation**
  - In earlier versions, programmatically verify (using DATABASEPROPERTYEX) the database setting for auto_update_stats and where OFF, use sp_recompile after updating statistics against the table
  - In all versions, only update statistics IF data has changed

NOTE: This DMV is available in SQL 2008 R2 SP2 and higher OR SQL Server 2012 SP1 and higher

```
SELECT [stats].*
FROM [sys].[dm_db_stats_properties]
   (object_id, stats_id) AS [stats]
```

Plan Invalidation Due To Statistics Updates

- **How much data has to change to cause statistics to be updated and therefore plan invalidation**
  - Without trace flag 2371:
    - If table row count <= 500 then threshold is 500 (minimum)
    - If table row count > 500 then threshold is 500 + (20% table rows)
  - With trace flag 2371:
    - If table row count <= 25,000 then threshold is 500 + (20% table rows)
    - If table row count > 25,000 then "dynamic threshold" rules apply:
      - If table row count = 100,000 rows then ~10%
      - If table row count = 1,000,000 rows then ~3.2%
      - If table row count = 10,000,000 rows then ~1%
      - If table row count = 50,000,000 rows then ~0.5%
      - If table row count = 100,000,000 rows then ~0.31%
- **When statistics are updated manually then at least one row has to have changed (in SQL Server 2012 onward) to trigger plan invalidation**

Updates To Statistics May Not Invalidate Bad Plans

- **From SQL Server 2012 onward, at least one row has to have changed to trigger plan invalidation when you update statistics manually**
  - Implemented to reduce "false positives" for parameter sniffing problems
- **Pre-SQL Server 2012 scenario (and, 2012+ if only a few rows modified)**
  - Users complain about poor performance for a stored procedure
  - During troubleshooting someone reviews the showplan output for the stored procedure
    - *Estimated number of rows* is way off from *actual number of rows*
  - Someone says "oh, it must be bad statistics"
  - So… statistics get updated manually
  - The problem seems to go away!
    - Why? Not because the statistics changed and made SQL Server choose a new plan but often instead because of the plan invalidation!
  - False positive!
    - The REAL problem is likely to be **parameter sniffing!**

Plan Invalidation / Recompilation Causes

```
SELECT [xemv].[map_key] AS [Recompile Reason ID]
    , [xemv].[map_value] AS [Recompile Reason Text]
FROM [sys].[dm_xe_map_values] AS [xemv]
WHERE [xemv].[name] = N'statement_recompile_cause'
```

1 = Schema changed
2 = Statistics changed
3 = Deferred compile
4 = Set option change
5 = Temp table changed
SQL 2005
6 = Remote rowset changed
7 = For browse permissions changed
8 = Query notification environment changed
9 = PartitionView changed
10 = Cursor options changed
11 = Option (recompile) requested
SQL 2008
12 = Parameterized plan flushed
13 = Test plan linearization
14 = Plan affecting database version changed
SQL 2012
15 = QDS plan forcing policy changed
16 = QDS plan forcing failed
SQL 2014

Stored Procedure Caching

- **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
    - 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!

Compilation Concerns

- **When is having a plan in cache a bad thing?**
  - 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?

RECOMPILATION = OPTIMIZATION

OPTIMIZATION = RECOMPILATION

- **If a plan is not stable then it might be best NOT to save it…**

When Should You Recompile?

- **When the optimal plan for a given statement within a procedure is not consistent / stable due to parameter changes**
  - You won't actually know this without testing (or, users complaining)
- **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!!**

Summary: Creation and Compilation

- **Stored procedures NEVER have their plans saved on disk**
  - Creation is solely for parsing and storing the metadata information
- **Stored procedure plans are transient objects that exist in memory**
  - They're not stored in the database (except when an object plan guide exists)
  - They're not guaranteed to be the same for every execution
    - There are many reasons for this (and no, it's NOT always statistics)
- **Re-using *some* compiled plans can be a good thing**
  - When the plans are stable
- **Re-using *some* compiled plans might be horribly bad**
  - When the plans are unstable / inconsistent, based on parameters supplied
- **Recompiling is NOT always a bad thing!**