

SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 9: Statistics – Internals and Updates

Kimberly L. Tripp
Kimberly@SQLskills.com



Overview

- **Cost-based optimization**
- **Data access patterns**
- **Statistics**
 - What do they look like?
 - What are they telling us?
 - How do you see them
 - When/how do they get created
 - When/how do they get updated
- **Additional resources**



2

© SQLskills. All rights reserved.
<http://www.SQLskills.com>



Statement Execution Simplified

Optimization = Compilation



Optimization: this is really where you can have the greatest impact and where the most interesting events can occur...

1. Parse
2. Standardization/normalization/algebraization
⇒ query tree (not T-SQL anymore)
3. Cost-based optimization (statistics are used to come up with optimization plan as well as lock granularity)
4. Compilation
5. Execution: at runtime, if the resources don't exist to support the lock level then escalation occurs to TABLE level (by default)

Cost-Based Optimization

- Find a reasonable subset of possible algorithms to access data based on:
 - The query *Sometimes a rewrite helps (join rewritten as sub-query or vice versa)*
 - Any joins *Sometimes a derived table helps (sub-query in the FROM clause)*
 - Any SARGs *Sometimes rewriting SARGs helps (well-defined predicates)*
 - Data selectivity
 - Join density
- The more information the optimizer has the better...
- How do you provide the BEST information?
- One of the best ways to "influence" your query plans is through effective statistics (and better indexes)

Selectivity

- Not just based on the number of rows returned
- Always relative to the number of rows in the table (usually expressed as a percentage)
- Low number of rows = high selectivity
 - Any index is useful if even ONE condition is highly selective!
- High number of rows = low selectivity
 - What is considered low selectivity? 5%, 10%, 15%???
 - Remember the tipping point – it's a very low percentage (usually 1-2%) where SQL Server deems a result set to be not selective enough
 - Must consider some form of covering

Understanding Selectivity

How Would YOU Find This Data?

- Imagine a table of employee data, for a Chicago company
- The table is clustered by EmployeeID
- Imagine executing this query?

```
SELECT e.*  
FROM dbo.EmployeesPersonalAddresses AS e  
WHERE e.city = 'Chicago' not selective enough  
WHERE e.city = 'Glenview' not an easy answer  
WHERE e.city = 'Peoria' selective enough
```

- When is an index on "City" useful?
 - When the data is selective ENOUGH... *More importantly, how does SQL Server know?*

How Would SQL Server Know?

- In every example, knowing the number of rows is necessary to estimate I/Os
- How does SQL Server know how many rows without reading the rows?

Statistics

What Do They Look Like?

1		2		Statistics Header			
Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index
MemberName	Oct 10 2008 1:02AM	10000	10000	26	0	21.5526	YES

Density Vector

3		All density	Average Length	Columns
		0.03846154	5.6154	LastName
		0.0001	16.5526	LastName, Firstname
		0.0001	17.5526	LastName, Firstname, MiddleInitial
		0.0001	21.5526	LastName, Firstname, MiddleInitial, member_no

Histogram

4		RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
		ANDERSON	0	385	0	1
		BARR	0	385	0	1
		CHEN	0	385	0	1
		...	...	...	...	...
		ZUCKER	0	384	0	1

Overall,
this is
weird
data?!

What Are They Telling You?

- **Histogram has the best detail about the first column (sometimes referred to as the high-order element [LastName])**
 - Anderson 385 rows
 - Barr 385 rows
- **Secondary columns – DENSITY (#3)**
 - Always LEFT-based combinations
 - Rows * All density = average number of rows given that column or combination of columns
 - Index LastName, FirstName will have average for LastName alone and combo
 - Index FirstName, LastName will have average for FirstName alone and combo
 - The density vector value for the combination will be the same
- **Density information**
 - Density for LastName = 0.03846154 [10,000 Rows * 0.03846154 = 384.6154 returned on average]
 - Density for LastName, FirstName combo – what does that tell you?

Are They Really True?

 **hidden slide**
w/extra details

- **For LastName**

```
SELECT AVG(Counts.GroupCounts)
FROM (SELECT COUNT(*) AS GroupCounts
      FROM dbo.member AS m
      GROUP BY m.LastName) AS Counts
```

- **For LastName, FirstName**

```
SELECT AVG(Counts.GroupCounts)
FROM
  (SELECT COUNT(*) AS GroupCounts
   FROM dbo.member AS m
   GROUP BY m.LastName, m.FirstName)
  AS Counts
```

- **What does this tell you about FirstNames?**

What Are the Options?

```
SELECT m.LastName, m.FirstName,  
       m.MiddleInitial, m.Phone_no, m.City  
FROM   dbo.Member AS m  
WHERE  m.FirstName LIKE 'Kim%'
```

- Table scan (always an option)
- No indexes exist for SEEKING (no indexes with FirstName as the high-order element)
- What about SCANNing the NC index which has FirstNames in it...seems like a gamble?

How Do You See Them? (1)

DBCC SHOW_STATISTICS (tname, statname)

- Gives you ALL the statistical details
 - Number of rows and number of rows on which the statistics were based
 - Densities for all LEFT based subsets of column, including the CL key (last – if not already somewhere in the index)
 - Histogram for high order element

sp_autostats tname

Index Name	AUTOSTATS	Last Updated
[member_ident]	ON	2008-08-26 17:18:12.59
[member_corporation_link]	ON	2008-08-26 17:18:12.61
[member_region_link]	ON	2008-08-26 17:18:12.73
[MemberName]	ON	2008-10-29 11:13:29.21
[_WA_Sys_00000003_OCBAE8]	ON	2008-10-29 11:28:32.31

How Do You See Them? (2)

- **Query sys.indexes**
 - Use object_id and index_id as input to stats_date
- **Query sys.stats**
 - Use object_id and stats_id as input to stats_date
- **Use STATS_DATE ([object_id], [index_id]) in a query**
 - Nice quick way to see JUST the date the statistics were last updated
 - Nice to check periodically and automatically
- **Use sys.dm_db_stats_properties (2008R2 SP2+, 2012 SP1+)**

What Kinds of Statistics Exist?

- **Auto-created statistics**
 - Named _WA_SYS_
 - Paul's blog: *How are auto-created column statistics names generated?*
(<http://bit.ly/1giY28K>)
 - Created automatically when a missing statistics event is encountered
 - Permanent objects in the database, will get auto-updated if database option is set
- **User-created statistics**
 - Created by you...
 - Could have been recommended by DTA and named _dta_stat_
- **Hypothetical indexes**
 - Created during DTA's analysis phase
 - Dropped by letting DTA complete successfully
 - Can be created manually for "what if analysis using auto pilot"

DBCC AUTOPILOT

- Check out the Simple Talk article *Hypothetical Indexes on SQL Server* (<http://bit.ly/1fi472d>)

```
CREATE INDEX <name> ON <tablename> (<columns>)
WITH STATISTICS_ONLY = -1
GO
DBCC AUTOPILOT(0, <dbid>, <objectid>, <indexid>)
GO
SET AUTOPILOT ON
GO
RUN QUERY TO TEST INDEX USAGE? (output is showplan xml)
GO
SET AUTOPILOT OFF
```

How Do You See Them? (3)

- Programmatically pull tabular sets from DBCC SHOW_STATISTICS ...
 - STAT_HEADER
 - Number of rows v. rows sampled and number of steps
 - Last updated date
 - DENSITY_VECTOR
 - Left-based density subsets
 - Averages given left-based combinations of index key
 - STAT_HEADER JOIN DENSITY_VECTOR
 - Presents the details from STAT_HEADER and DENSITY_VECTOR as a single row
 - All Density = $\text{TABLE_CARDINALITY} / \text{TUPLE_CARDINALITY}$ (for each left-based subset starting ending at that ordinal position)
 - HISTOGRAM
 - Up to 201 total steps with each step's density information
 - Up to 200 distinct actual values from the table
 - 1 row for NULLs if the column allows NULL values

When Are They Created?

- **Automatically**
 - For all Indexes
 - When “auto create statistics” is ON AND when the optimizer thinks that statistics would be a good idea (often when a column is in a SARG or a join and does not have an index with that column as the high-order element)
- **Manually**
 - sp_createstats
 - Using CREATE STATISTICS

Tip: Leave Auto Create Statistics ON

Manually Creating Statistics

- For secondary columns of an index: can make SCAN choices more likely for highly selective secondary conditions that don't warrant an index (where only some values are selective and where query frequency doesn't warrant the additional index)
- For columns in search arguments and joins where some are selective and others aren't (and again, you've decided not to index)

sp_createstats

```
sp_createstats
@indexonly = 'indexonly'
, @fullscan = 'fullscan'
, @norecompute = 'norecompute'
```

- **@indexonly: only create statistics for secondary columns of indexes**
 - This can help to make non-clustered indexes more useful
- **@fullscan: requires more time but will create more accurate statistics**
 - If off-hours this is a good idea
- **@norecompute: the statistics will NOT get automatically updated as distribution of data changes**
 - Generally not recommended
- **Recommendation:**
`sp_createstats 'indexonly', 'fullscan'`

What If the Data Changes?

- **Automatically updated!**
 - If auto update statistics is ON (for both the DB and the statistic – didn't set "norecompute")
 - If roughly 500 + 20% of the data changes
 - *Complete details in TechNet whitepaper*
- **Manually update statistics**
 - For highly volatile tables where distribution isn't changing significantly and you see a lot of "statistics" events
 - Might want to increase the frequency of updating (and less than 20% change)
 - If so, consider turning off auto update stats at the statistic-level by adding STATISTICS_NORECOMPUTE on the index definition or NORECOMPUTE on the statistics definition
 - This is a lot better and offers more granular control than turning it off at the database level
 - Executing UPDATE STATISTICS

Auto Update Statistics

- **SQL Server 7.0**
 - Invalidated when sysindexes.rowmodctr reached
 - Updated when invalidated
- **SQL Server 2000**
 - Invalidated when sysindexes.rowmodctr reached
 - ✓ Updated when needed
- **SQL Server 2005**
 - ✓ Invalidated when sysrowsetcolumns.rcmodified reached
 - Updated when needed
- **SQL Server 2008+**
 - Invalidated when sysrscols.rcmodified reached
 - Updated when needed

Statement Execution: Statistics Events

Optimization = Compilation



Missing Statistics Event

If `auto_create_stats` is enabled then SQL Server (the QO) will WAIT while statistics are created

Invalidated Statistics Event

- If `auto_update_stats_async` is disabled AND `auto_update_statistics` is enabled then SQL Server will WAIT while statistics are updated
- If `auto_update_stats_async` is enabled then SQL Server will optimize based on the invalidated statistics AND kick off an update (which will be used by subsequent users)
- If both methods for updating statistics are disabled then the query will optimize using the invalidated statistic and a warning will be generated (visible in [xml] showplan)

Asynchronous Statistics Updates

- By default statistics are updated before query compilation (as part of compilation/optimization) and this can cause a delay in execution
- Can turn "Async Stats Update" on to have the current execution trigger the update but not wait for the update (but you could get into a vicious cycle where you optimize with old statistics, trigger the update and by the time you execute again, they're invalid)

```
ALTER DATABASE databasename  
SET AUTO_UPDATE_STATISTICS_ASYNC ON
```

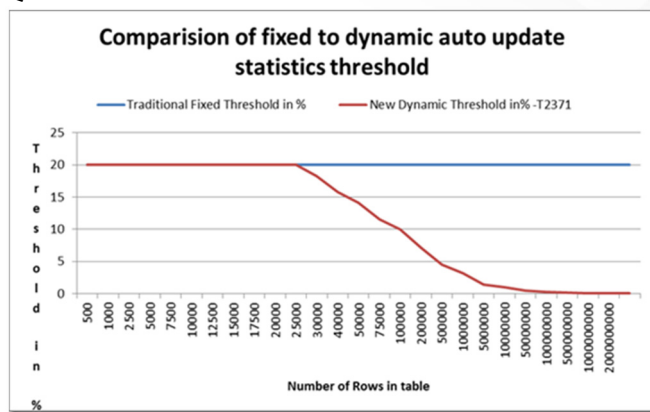
BUG: When you enable the Auto Update Statistics Asynchronously statistics option in a database of Microsoft SQL Server 2012, Microsoft SQL Server 2008, or Microsoft SQL Server 2008 R2, and then you run queries on the database, a memory leak occurs.

FIXED: Cumulative Update 2 for SQL Server 2012 SP1, Cumulative Update 5 for SQL Server 2012, Cumulative Update 4 for SQL Server 2008 R2 SP2

See KB article: 2778088 <http://support.microsoft.com/kb/2778088>

Dynamic Auto-Updating Threshold

- Server-wide trace flag 2371
- Released in SQL Server 2008 R2 SP1
- Blogged by:
Juergen
Thomas
(SQLCat)
7 Sep 2011



Incremental Stats Updates

- **New feature for SQL Server 2014 partitioned tables**
- **Statistics update triggered when 20% of the partition changes**
 - Build partition-level statistics
 - Compress table-level statistics and merge partition-level statistics in with table-level
 - Only table-level statistics are used for estimation
- **Positive: statistics updates are triggered a lot earlier for partitioned tables**
- **Negative: statistic is still limited to 200 steps and can become lossy for earlier partitions**
- **For more information**
 - **Article: SQL Server 2014 Incremental Statistics on SQLperformance.com**
 - <http://bit.ly/1uhEbr1>

Updating Statistics

- **Manually: but automated through a job**
 - Executing sp_updatestats
 - Sledgehammer maintenance. Only one row has to have been modified to execute this... not very granular
 - Ola Hallengren's code
 - <http://ola.hallengren.com/>
 - Roll your own?
 - Programmatically evaluate the stat_header (just an idea)
 - If stats_date older than 1 week OR sampled < rows then UPDATE STATS with FULLSCAN
- **Auto update stats: only as a safety measure**
 - As a fallback if your code doesn't catch EVERY statistic
- **Asynchronous update stats**
 - *Unlikely* to cause a problem

Plan Invalidation

- **Versions prior to 2012**
 - If database option: `auto_update_stats` is ON
 - Updating statistics causes plan invalidation
 - If database option: `auto_update_stats` is OFF
 - Updating statistics does NOT cause plan invalidation
 - If you manually update statistics and do not use `auto_update_statistics`, add `sp_recompile` to your stats scripts
 - For more info: Erin Stellato's links about auto update stats/plan invalidation:
 - Statistics and Recompilations: <http://erinstellato.com/2012/01/statistics-recompilations/>
 - Statistics and Recompilations, Part II: <http://erinstellato.com/2012/02/statistics-recompilations-part-ii/>
- **SQL Server 2012+**
 - Plan invalidation is NOT affected by the setting of auto update stats
 - Plan invalidation does NOT occur if data has not changed

Review

- **Cost-based optimization**
- **Data access patterns**
- **Statistics**
 - What do they look like?
 - What are they telling us?
 - How do you see them
 - When/how do they get created
 - When/how do they get updated
- **Additional resources**

Questions!



Additional Resources

- **Stats trace flags of interest**
 - Query-level trace flags
 - Generating additional statistics output from DBCC SHOW_STATISTICS
- **Automating the analysis skew (from histograms)**
 - PASStv: <http://www.youtube.com/user/SQLPASSTV?feature=watch> for the session: **Skewed Data, Poor Cardinality Estimates, and Plans Gone Wrong**



30

© SQLskills. All rights reserved.
<http://www.SQLskills.com>



Stats Trace Flags of Interest

- 9204: statistics used by query
- 9292: statistics header loaded
- 2371: dynamic auto updating of statistics
- 2388: Additional statistics details (see next slide)
- 2389: don't rely on "max" value – query the table to compute the highest value of the column. Useful for ascending columns that are queried long before their statistics are updated. Statistics must have been updated at least 3 times to be "branded" as ascending (Column: Leading Column Type from 2388 output)
- 2390: Read highest value from column disregard branding

Query-level Trace Flags

```
SELECT m.*  
FROM dbo.Member AS m  
WHERE m.firstname LIKE 'Kim%'  
OPTION  
    (QUERYTRACEON 3604  
    , QUERYTRACEON 9204  
    , QUERYTRACEON 9292  
    , RECOMPILE);
```

- Only set for that execution
- Affect only that query

Additional Statistics Output

DBCC TRACEON(2388)

```
DBCC SHOW_STATISTICS  
(  
    'member',  
    'member_ident'  
)
```

DBCC TRACEOFF(2388)

Returns additional details from statistics blob. Columns:

- ❑ Updated
- ❑ Table Cardinality
- ❑ Snapshot Ctr
- ❑ Steps
- ❑ Density
- ❑ Rows Above
- ❑ Rows Below
- ❑ Squared Variance Error
- ❑ Inserts Since Last Update
- ❑ Deletes Since Last Update
- ❑ Leading column Type