SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 9: Statistics – Internals and Updates

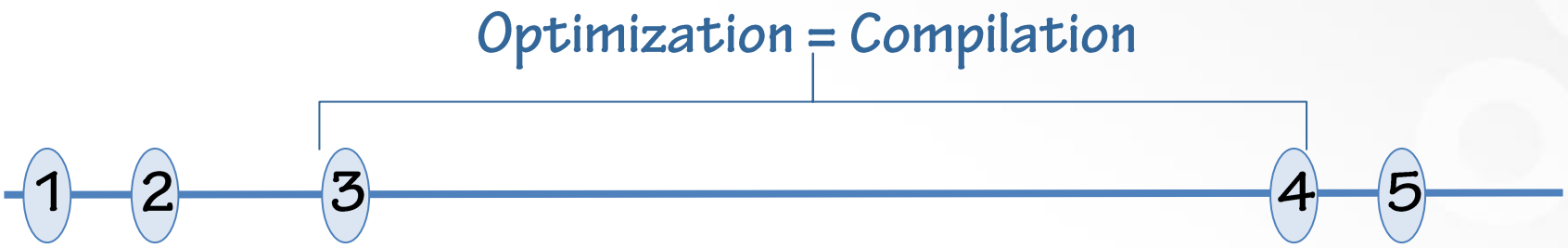
Kimberly L. Tripp

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**

Statement Execution Simplified



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

1. Parse
2. Standardization / normalization / algebrization ⇒ query tree
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)**
- **Understanding optimization / estimation is important for troubleshooting a wide variety of solvable query problems!**

Performance Problems

- **Query performance inconsistencies**
  - Execution time varies
  - Statement's execution plan varies
  - IO / CPU metrics vary
- **Variations due to:**
  - Statement execution method
  - Parameters
  - Time (fragmentation, statistics not current, plan not valid)
- **Sledgehammer approaches**
  - Updating statistics
  - Rebuilding indexes
  - Clearing cache

Troubleshooting Methodologies

- **Is this a CACHED plan method?**
  - Stored procedure
  - sp_executesql
- **Test to see if the optimal plan varies from the cached plan?**
  - `EXECUTE <procedure> <params>`
  - `EXECUTE <procedure> <params> WITH RECOMPILE`
  - NOTE: execute with recompile does NOT allow the parameterization embedding optimization [doesn't optimize as effectively as `OPTION (RECOMPILE)` so there are features that might not be leveraged EXCEPT when using `OPTION (RECOMPILE)` at the statement level]
- **Do you get a different plan for the second?**
  - Parameter sensitivity ("sniffing") problem
    - Long term solution is to changing the code

Troubleshooting Methodologies

- **Is this a poorly performing but NEW (non-cached) plan?**
  - AdHoc statement
  - First execution
  - Plan doesn't change when using WITH RECOMPILE
- **Use ACTUAL EXECUTION**
  - Estimated plan vs. actual plan
    - Be sure to check executions * rows (not just estimated rows vs. actual rows)
  - Problems / solutions that can be exposed by incorrect row estimations
    - Statistics out-of-date -> updating scenarios
    - Estimate is incorrect because of skewed data -> filtered statistics scenarios
    - Estimate is incorrect because of estimation algorithm -> cardinality estimation scenarios

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'
WHERE e.city = 'Glenview'
WHERE e.city = 'Peoria'
```

not selective enough

not an easy answer

selective enough

- When is an index on "City" useful?
  - When the data is selective ENOUGH…

More importantly, how does SQL Server know?

What Kinds of Statistics Exist?

- **Statistics on indexes**
- **Auto-created, column-level statistics**
  - Named _WA_SYS_
    - Paul's blog: *How are auto-created column statistics names generated?* (http://bit.ly/1gjY28K)
  - 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 unless graphical plan on)
GO
SET AUTOPILOT OFF
```

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

Overall, this is weird data?!

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

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 17:18:12.5
[member_corporation_link]	ON	2008-08-26 17:18:12.6
[member_region_link]	ON	2008-08-26 17:18:12.7
[MemberName]	ON	2008-10-29 11:13:29.2
[_WA_Sys_00000003_0CBAE8	ON	2008-10-29 11:28:32.3

Statistics Header

Straightforward information; key things to look at are: updated, rows vs. rows sampled, and whether or not the index is filtered

- Updated: date last updated (or, created)
- Rows: number of rows relating to the statistical set (at the time of creation)
- Rows Sampled: number of rows used to generate the sample set
- Steps: number of rows in the histogram (<= 201)
- Density: not used or useful, from earlier versions
- Average key length: average of entire key incl. columns internally added
- String Index: may be useful for strings under 80 characters. For strings longer than 80 characters, only the first 40 and last 40 characters are used for the string summary. For large strings, accurate frequency estimates for substrings that appear only in the ignored portion of a string are not available
- Filter Expression: if filtered index then this is the expression for set
- Unfiltered Rows: number of rows in the table at the time the statistic was created (same as rows if the index is not filtered)

Density Vector

- **Based on a left-based subset of key columns, details the distribution of data**
- **Rows * All density = average number of rows given that column or combination of columns**
  - Index LastName, FirstName will have average for last names alone as well as the combination of last names and first names
  - Index FirstName, LastName will have average for first names alone as well as the combination of first names and last names
  - 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 rows returned on average]
  - Density for LastName, FirstName combo – what does that tell you?

Histogram

More details about the histogram are coming up in the problems / solutions section (Module 11)!

- **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
- **Histogram has the most detail about the first column (sometimes referred to as the high-order element [LastName])**
  - Anderson 385 rows
  - Barr 385 rows
- **Cannot modify characteristics of statistic structures**
  - Steps chosen / total number of steps
  - How it's built (step compression)
  - Updating can only be done when the entire set is evaluated
    - Update statistics
    - Index rebuild (not reorg)
- **New in SQL Server 2016 (13.x) SP1 CU2: sys.dm_db_stats_histogram**

Are They Really True?

- 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?

Statistics Usage

- **Estimations for known values**
  - Histogram step value: EQ_ROWS
  - Histogram value in step range: AVG_RANGE_ROWS
  - Distinct value estimation (TUPLE_CARDINALITY)
    - 1 / All density
- **Unknown values**
  - Density vector average
    - Density * Rows
- **Selectivity for a set (indirect index usage)**
  - Index statistics are often used from the index that the query uses
  - Column-level or index-level statistics can be analyzed independently to determine other possible algorithms (even when not covered)

Histogram Step Value

```
SELECT [m].*
FROM [dbo].[Member] AS [m]
WHERE [m].[LastName] = 'Chen'
```

```
DBCC SHOW_STATISTICS
('Member', 'MemberName')
WITH HISTOGRAM;
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	385
Actual Number of Batches	0
Estimated I/O Cost	0.107569
Estimated Operator Cost	0.118726 (100%)
Estimated Subtree Cost	0.118726
Estimated CPU Cost	0.011157
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	385
Estimated Row Size	189 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0

Predicate

[Credit].[dbo].[member].[lastname] as [m].[lastname] like 'Chen'

Object

[Credit].[dbo].[member].[member_ident] [m]

Editor | Results | Messages

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_
1	ANDERSON	0	385	0	1
2	BARR	0	385	0	1
3	CHEN	0	385	0	1
4	DORR	0	385	0	1

Histogram Value In Step Range

```
SELECT [m].*
FROM [dbo].[member] AS [m]
WHERE [m].[corp_no] = 404;
```

```
DBCC SHOW_STATISTICS
('Member', 'member_corporation_link')
WITH HISTOGRAM;
```

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	0
Actual Number of Batches	0
Estimated Operator Cost	0.0032897 (14%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001647
Estimated Subtree Cost	0.0032897
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	7
	15 B
	0
	0
	True
	1

Editor | Results | Messages

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
137	402	0	8	0	1
138	403	0	4	0	1
139	407	14	5	2	7
140	408	0	7	0	1

4 rows for value 403 | 5 rows for value 407 | 14 rows between 403 and 407 but not including 403 or 407
14 rows between 403 and 407 over 2 distinct values = 7 rows "on average"
404 / 405 / 406 will estimate 7… is that correct?

Distinct Value Estimation

```
SELECT DISTINCT [m].[corp_no]
FROM [dbo].[member] AS [m];
```

```
DBCC SHOW_STATISTICS
('Member', 'member_corporation_link')
WITH DENSITY_VECTOR;
```

Editor | Results | Messages

	All density	Average Length	Columns
1	0.0025	0.6008	corp_no
2	0.0001	4.6008	corp_no, member_no

```
All density * rows = average
1 / All density = tuple_cardinality
1 / 0.0025 = 400
```

Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	400
Actual Number of Batches	0
Estimated I/O Cost	0
Estimated Operator Cost	0.0052 (16%)
Estimated CPU Cost	0.0052
Estimated Subtree Cost	0.0320746
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	400
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	0

Output List

[Credit].[dbo].[member].corp_no

Group By

[Credit].[dbo].[member].corp_no

Unknown Values (1)

```
DECLARE @Lastname varchar(15) = 'Chen';
SELECT [m].*
FROM [dbo].[Member] AS [m]
WHERE [m].[LastName] = @Lastname;
```

```
DBCC SHOW_STATISTICS
('Member', 'MemberName')
WITH DENSITY_VECTOR;
```

	All density	Average Length	Columns
1	0.03846154	5.6154	lastname
2	0.0001	16.5526	lastname, firstname
3	0.0001		
4	0.0001		

All density * rows = average
0.03846154 * 10000 = 384.615

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	385
Actual Number of Batches	0
Estimated I/O Cost	0.107569
Estimated Operator Cost	0.118726 (100%)
Estimated Subtree Cost	0.118726
Estimated CPU Cost	0.011157
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	384.615
Estimated Row Size	189 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0

Predicate

[Credit].[dbo].[member].[lastname] as [m].[lastname]= [@Lastname]

Object

[Credit].[dbo].[member].[member_ident] [m]

Unknown Values (2)

```
DECLARE @Lastname varchar(15) = 'Fish'
SELECT [m].*
FROM [dbo].[Member] AS [m]
WHERE [m].[LastName] = @Lastname;
```

```
DBCC SHOW_STATISTICS
('Member', 'MemberName')
WITH DENSITY_VECTOR;
```

	All density	Average Length	Columns
1	0.03846154	5.6154	lastname
2	0.0001	16.5526	lastname, firstname
3	0.0001		
4	0.0001		

All density * rows = average
0.03846154 * 10000 = 384.615

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	0
Actual Number of Batches	0
Estimated I/O Cost	0.107569
Estimated Operator Cost	0.118726 (100%)
Estimated Subtree Cost	0.118726
Estimated CPU Cost	0.011157
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	384.615
Estimated Row Size	189 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0

Predicate

[Credit].[dbo].[member].[lastname] as [m].[lastname]= [@Lastname]

Object

[Credit].[dbo].[member].[member_ident] [m]

Indirect Index Usage (Set Selectivity)

```
SELECT [m].[LastName], [m].[FirstName],
   [m].[MiddleInitial], [m].[Phone_no],
   [m].[City]
FROM [dbo].[Member] AS [m]
WHERE [m].[FirstName] LIKE 'Kim%'
OPTION (QUERYTRACEON 3604, QUERYTRACEON 9204, RECOMPILE);
```

Output to the client

Statistics used

Re-evaluate

- Table scan (always an option)
- No indexes exist for SEEKING (no index with first name as the high-order element)
- What about scanning the NC index which has first names in it?
- What would be the best algorithm?

How Do You See Them? (2)

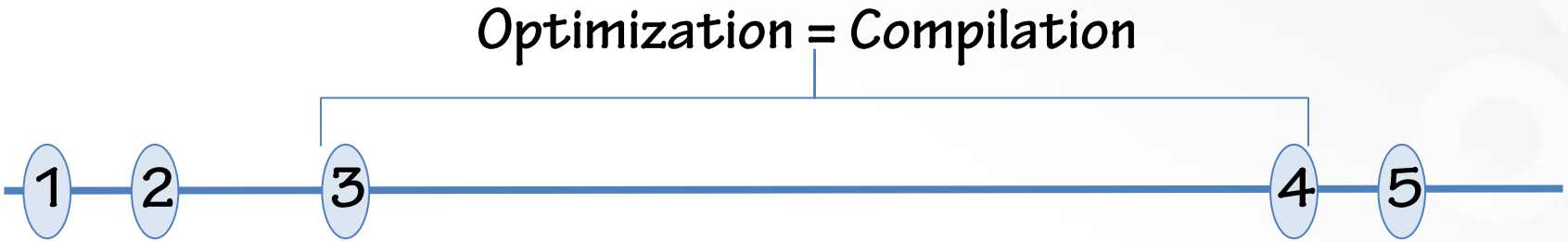
- **Query [sys].[indexes]**
  - Use OBJECT_ID and INDEX_ID as input to STATS_DATE
- **Query [sys].[stats] (preferred)**
  - Shows ALL statistics (index-level, column-level, hypothetical, etc.)
  - Use OBJECT_ID and **STATS_ID** as input to STATS_DATE
- **Use `STATS_DATE ([object_id], [index_id])` in 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+)**
  - Great for automation routines

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 = TABLE_CARDINALITY/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

If you want some examples of how to do this, see the SQLskills procedures for analyzing data skew.

Statement Execution: Statistics Events



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)

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 an 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 existing index:**
  - Gives the optimizer more options for using existing indexes
    - Scanning an index for highly selective secondary values
  - Can make index usage more likely for secondary conditions; helping understand set selectivity where creating an index is not ideal:
    - Don't warrant a permanent index because the queries are neither frequent enough or critical enough
    - Aren't selective enough across many values / only some values can benefit
- **For columns in search arguments or joins where some are selective and others aren't (and again, you've decided not to index)**
  - Can help to determine position of a table in a join

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 statistics**
  - If auto update statistics is ON (for ***both*** the DB and the statistic)
  - Statistics are invalidated when:
    - In versions PRIOR to SQL Server 2016: If a minimum of 500 + 20% of the data changes
    - In SQL Server 2016 OR in prior versions with a TF: the threshold is dynamic and tied to the number of rows in the table (*more details coming up*)
- **Manually update statistics**
  - Executing UPDATE STATISTICS
    - Might want to **decrease** the frequency of updating 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 for large table where distribution is changing significantly but you're not reaching 20%
  - Consider turning off *auto update stats* at the statistic-level instead of the database-level (more granular control); use:
    - STATISTICS_NORECOMPUTE on the index definition
    - NORECOMPUTE on the statistics definition

Auto Update Statistics

If someone challenges you on the overhead of auto update statistics it could be because of the original (poor) design.

- **SQL Server 7.0**
  - Invalidated when sysindexes.rowmodctr reached
  - Updated when invalidated (**yikes!**)
- **SQL Server 2000**
  - Invalidated when sysindexes.**row**modctr reached
  - ✓ Updated **when needed**
- **SQL Server 2005**
  - Invalidated when sysrowsetcolumns.**rc**modified reached (column modification counter *this is both good and bad…*)
  - ✓ Updated when needed
- **SQL Server 2008+**
  - Invalidated when sysrscols.rcmodified reached
  - ✓ Updated when needed

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 (unlikely to be a problem AT the auto-update threshold as it would have likely been a problem leading up to that threshold)

```
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.

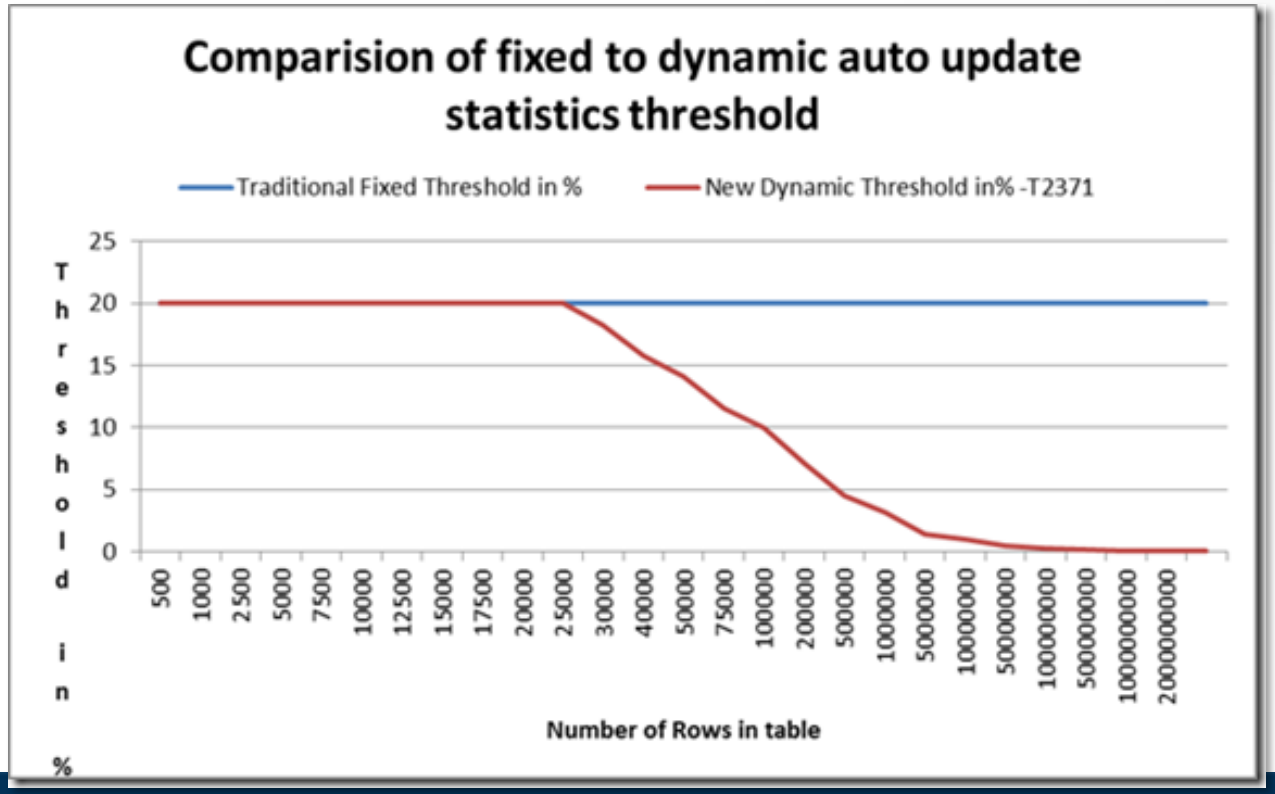
FIXED: Cumulative Update 2 for SQL Server 2012 SP1, Cumulative Update 5 for SQL Server 2012, Cumulative Update 4 for SQL Server 2008 R2 SP2

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

Dynamic Auto-Updating Threshold

- Server-wide trace flag 2371 prior to SQL Server 2016
- NEW behavior in SQL Server 2016+ if in compat mode 130+
- Originally released in SQL Server 2008 R2 SP1
- Blogged by: Juergen Thomas (SQLCat) 7 Sep 2011

Dynamic threshold
=SQRT(rows * 1000)



Incremental Stats Updates

- New feature for SQL Server 2014 partitioned tables
- Statistics update triggered when threshold reached at the partition-level
  - Build partition-level statistics
  - Compress table-level statistics
  - Merge partition-level statistics in with table-level statistics
  - Resulting table-level statistics are used for estimation
- Positive: statistics updates are triggered a lot earlier for partitions that are updates (especially useful in ever-increasing tables)
- Positive: less data has to be read for statistics updates (especially useful in ever-increasing tables)
- Negative: the statistic the optimizer uses for optimization is still limited to 200 steps and can become lossy for earlier partitions
  - 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.
  - Ola Hallengren's code
    - http://ola.hallengren.com/
    - Updating Statistics with Ola Hallengren's Script by Erin Stellato
    - Integrate your own logic into Ola's code
      - @OnlyModifiedStatistics = 'Y' (only one row)
      - Better to use: @StatisticsModificationLevel = '%'
        NOTES: You can specify a *percentage* but you need to do some calculations with the dynamic threshold to see if this is more or less frequent than auto updating
      - OR consider turning off auto-updating (which would use sampling) in favor of your off-hours job that might use 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 have set auto_update_statistics to OFF, add sp_recompile @tname to your stats scripts (remembering SCH_M problems)
  - 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
    - Only for UPDATE STATISTICS
    - An index rebuild will update statistics as well as cause plan invalidation, even if no data has changed thus increasing the importance for "*only when data has changed*" logic in maintenance routines.

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**
  - Quite a few of these are in the demo scripts (especially for module 11)
  - Many are query-level trace flags – must be sys admin to use
    - Unless the code is within a stored procedure (yes, really!)
    - Blog Post: Setting CE TraceFlags on a query-by-query (or session) basis
      - https://bit.ly/2Kf6AyL
    - LOTS of new / semi-new (2016+) query hints to use INSTEAD. I've included most of these for searching against older code, etc. and just as REFERENCE
    - REVIEW
      - Books Online: DBCC TRACEON - Trace Flags (Transact-SQL)
      - Books Online: Hints (Transact-SQL) – Query
  - Generating additional statistics output from DBCC SHOW_STATISTICS can be helpful / interesting

Stats Trace Flags of Interest

- **Statistics used by query (Legacy CE: 9204 / no equivalent in New CE)**
  - Could use trace flag: 2363
- **9292: statistics header loaded**
- **2388: Additional statistics details (see last slide)**
- **In SQL Server 2017, CU3 with compatibility mode 120 and higher AND only with legacy CE OFF – you can see the new plan attributes in showplan XML: StatisticsInfo and OptimizerStatsUsage**

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
- Affects only that query

- If using a NEW query hint – use this structure:

```
SELECT …
FROM …
OPTION (USE HINT ('query_hint'))
```

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