SQLskills Online Immersion Event

IEQuery: Immersion Event on Fixing Slow Queries, Inefficient Code, and Caching/Statistics Problems

Part III: How to Differentiate Caching / Statistics problems and SOLVE THEM!

Kimberly L. Tripp
Kimberly@SQLskills.com
@KimberlyLTripp

SQLskills
immerse yourself in sql server

Kimberly L. Tripp

Founder / President, SQLskills

Kimberly@SQLskills.com @KimberlyLTripp
www.sqlskills.com/blogs/Kimberly

SQLskills
immerse yourself in sql server

Consultant / Trainer / Speaker / Writer

- Author / instructor for SQL Server Immersion Events: IEPTO1, IEPTO2, and IEHADR
- Author / presenter for Pluralsight
- Instructor and exam author for SQL Server and Sharepoint MCMs
- Author / manager of SQL Server 2005 and 2008 Launch Content
- Author / speaker at Microsoft TechEd, SQLPASS, ITForum, TechDays, and SQLintersection
- Author of SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability, SQLCLR for DBAs
- Author / presenter for MSDN and TechNet (25+)
- Instructor / SME for Microsoft University

Data Platform MVP

Recognized as an MVP by Microsoft since 2002

When I'm not working with SQL

- Traveler/adventurer and avid diver/photographer: www.BlueWaterImages.com
- @BlueWaterImages

Blue Water IMAGES

SQL intersection MVP Microsoft Most Valuable Professional PLURALSTIGHT

You Know The Drill

- **Lecture 1: 90 minutes from 10:00am until 11:30am PT**
  - Please use "CHAT" for questions *during* the lecture
- **Open Q&A 1: 30 minutes from 11:30am until 12:00 noon PT**
  - Will address unanswered questions from chat
  - May open up for "open mic" questions
- **Mandatory break: 30 minutes from 12:00 noon until 12:30pm PT**
  - Everyone needs a bit of a break!
- **Lecture 2: 90 minutes from 12:30pm until 2:00pm PT**
  - Please use "CHAT" for questions *during* the lecture
- **Open Q&A 1: up to 60 minutes from 2:00pm until 3:00pm PT**
  - Will address unanswered questions from chat
  - Will open up for "open mic" questions

Time Conversions
10:00am PT
= 1:00pm ET
= 6:00pm UTC

Course Resources & Pluralsight Access

- **Paul (paul@sqlskills.com) will send you an email for a FREE 30-day access to ALL SQLskills online training classes on Pluralsight**
  - No strings attached, no credit-card required
  - If you don't receive it, send mail to Paul
- **Course resources will be sent upon completion (Friday+)**
  - I may want to update a slide and/or add some notes/resources based on questions and discussions
  - The resource zip will be password protected. Please do not distribute; this is for course attendees only. The password will be sent in email.
- **Email Paul (Paul@SQLskills.com) if you have problems finding it or opening it from Monday (Feb 18) forward; I plan to finalize the resources (complete the whiteboard, etc.) on Friday!**

Course Resources

- **Course content (all presenter decks for all three days)**
- **Whiteboard(s)**
- **Demos**
  - All instructor-led demo scripts
  - All reference scripts
- **Reference links (in the "resources zip")**
- **Online recordings for these sessions**

Please do not redistribute: These resources are for IEQuery attendees only.
Please respect our IP and time in creating these resources and do not share/forward.

Schedule – Thursday

- **Lecture 1: Troubleshooting Statement Execution and Caching**
  - Different ways to execute statements
  - Statements caching for reuse
  - Statement auto-parameterization
  - Dynamic string execution
  - sp_executesql
  - Stored procedures
  - Literals, variables, and parameters
  - The life of a plan in cache
  - Plan cache limits
  - Bringing it all together

Schedule – Thursday

- **Lecture 2: Troubleshooting Plan Problems Related to Statistics (not Caching)**
  - Statement selectivity
  - What kinds of statistics exist
  - How does SQL Server use statistics
  - Creating additional statistics
  - Updating statistics

SQLskills

Overview: Root-cause Analysis

Where is the ACTUAL problem?

- Cache?
- Statistics?

SQLskills
immerse yourself in sql server

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

BETTER: Troubleshooting Methodologies (1 of 2)

- **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>` or `sp_executesql` (same way)
  - VS. recompiling: `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 but that's not a problem here…]
- **Do you get a different plan for the second?**
  - Parameter sensitivity ("sniffing") problem
    - Long term solution is to changing the code

BETTER: Troubleshooting Methodologies (2 of 2)

- **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

Lecture 1: Troubleshooting Statement Execution and Caching

Overview: Statement Execution and Caching

- Different ways to execute statements
- Statements caching for reuse
- Statement auto-parameterization
- Dynamic string execution
- sp_executesql
- Stored procedures
- Literals, variables, and parameters
- The life of a plan in cache
- Plan cache limits
- Bringing it all together

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
(*a bit more on this is coming up*)

Some Statements can be Cached for Reuse (1 of 2)

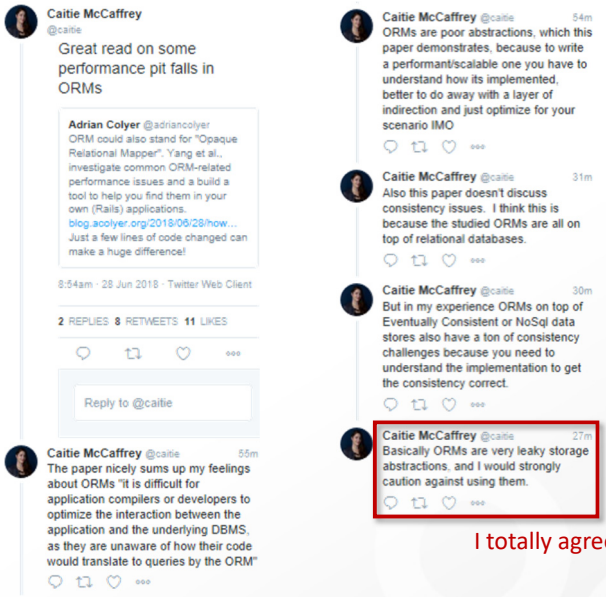
- **Ad hoc statements and dynamic strings are evaluated at runtime**
  - If a statement is very simple ("safe"), then it can be parameterized and cached
    - It's generally a good thing that the plans are cached
      - Saves CPU/time
      - Reduced footprint in the cache
    - This can lead to a small 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
```


Why is this Such an Issue?

- **EVIL ORMS!**
- **ORM = Object Relational Mapper**
- **Pro – Rapid Application Development**
- **Con – Fast to production, SLOW for the life of the application**

- **How NOT to structure your database-backed web applications: a study of performance bugs in the wild**
- **https://bit.ly/2IxpkI4**



I totally agree!

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: Keep It Simple!

- **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 BEFORE changing to forced by using query_hash and query_plan_hash (check out the Pluralsight course on SQL Server: Optimizing Ad Hoc Statement Performance, Section: Plan Cache Pollution)
- **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**

Ad Hoc Statement / Dynamic String Execution (DSE)

- **Statement / 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

Literals, Variables, and Parameters (1 of 2)

hidden slide w/extra details

Check out the Pluralsight courses for in-depth information about these options!

- **Literals**

```
SELECT … WHERE member_no = 12
```

  - member_no is being compared to a specific, defined value
  - Literals are "known" values at the time of optimization (compilation)
  - They provide the best information for optimization
- **Variables**

```
DECLARE @mno int = 12
SELECT … WHERE member_no = @mno
```

  - @mno is being defined and set in the declaration line
  - Variables are "unknown" values at the time of optimization (compilation) as the assignment does not occur until runtime
  - They provide NO useful information for optimization
  - The optimizer has to use "averages" instead of real values
    - This can be both good and bad…

Literals, Variables, and Parameters (2 of 2)

hidden slide w/extra details

Check out the Pluralsight courses for in-depth information about these options!

- **Parameters look like variables but they're not…**
- **Variables CAN be defined within a stored procedure**

```
CREATE PROCEDURE procname
   ( @parameter1    int )
AS
DECLARE @mno int = 12
SELECT … WHERE member_no = @mno
SELECT … WHERE member_no = @parameter1
```

  - @mno is being defined and set inside the stored procedure; this is a variable
  - @parameter1 is being defined as a parameter and the value will be KNOWN at execution
  - If this procedure does not yet have a plan defined in cache, then SQL Server will evaluate PARAMETER values during optimization (not VARIABLES)
  - This leads to the phrase: parameters can be "sniffed," variables are unknown.

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_dbcmptlevel
    - 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 Invalidation

hidden slide w/extra details

Check out the Pluralsight courses for in-depth information about these options!

- **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.

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 and higher
    - 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?

Reducing Plan Cache Bloat & Optimizing Statement Execution

- **Analyze your plan cache to see if you have ad hoc "bloat"**
  - See plan cache blog post slide for query/details to analyze your cache
- **Turn on the server-wide configuration "optimize for ad hoc workloads"**
- **Setup a job to regularly monitor for plan cache bloat and clear the SQL Plans portion of the cache using DBCC FREESYSTEMCACHE**
- **Work to change the way that ad hoc requests are made**
  - Use ad Hoc for statements with unstable plans
  - Use sp_executesql for statements with stable plans
  - If you want centralized logic, code reuse, and compiled / cached plans (when they're stable) and lots of other options (for when the plans are not stable), use stored procedures
    - Written by database developers that should
      - Know the data / workload / requirements
      - Know how SQL Server works
    - Can be written to balance CPU / caching for optimal performance!

Review: Statement Execution and Caching

- **Different ways to execute statements**
- **Statements caching for reuse**
- **Statement auto-parameterization**
- **Dynamic string execution**
- **sp_executesql**
- **Stored procedures**
- **Literals, variables, and parameters**
- **The life of a plan in cache**
- **Plan cache limits**
- **Bringing it all together**

Questions!

SQLskills
immerse yourself in sql server

Lecture 2: Troubleshooting Plan Problems Related to Statistics

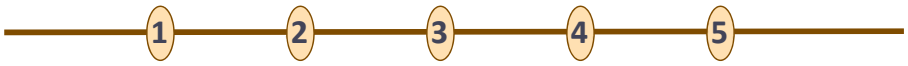
SQLskills
immerse yourself in sql server

Overview: Plan Problems Related to Statistics

- Statement selectivity
- What kinds of statistics exist
- How does SQL Server use statistics
- Creating additional statistics
- Updating statistics

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 / algebrization ⇒ query tree
3) Cost-based optimization
   (*statistics are used to come up with optimization…*)
4) Compilation
5) Execution

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!**

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 there's always a "tipping point" (where one option is better than another – based on the amount of data that's going to be processed)
    - The "tipping point" between using a non-clustered index that does NOT cover a query vs. performing a table scan happens at a very low percentage (usually 1-2%)
    - Non-clustered indexes that "support" a WHERE clause might not be *selective enough* to be used
      - This can cause plan changes / instabilities
    - NOTE: To learn more about the tipping point, check out my Indexing course on Pluralsight
  - 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?

What Kinds of Statistics Exist?

- Statistics on indexes
- Auto-created 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"
    - *See hidden slide, link, and demo script for help on understanding this VERY cool feature!*

DBCC AUTOPILOT

hidden slide w/extra details

Check out my script samples / examples for using 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

	All density	Average Length	Columns
(3)	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

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
(4)	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?!

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**
  (NOTE: *I almost never use this; I use queries against sys.stats but I included here so you'd know it exists*)

Index Name	AUTOSTATS	Last Updated
[member_ident]	ON	2008-08-26 17:18:12.593
[member_corporation_link]	ON	2008-08-26 17:18:12.673
[member_region_link]	ON	2008-08-26 17:18:12.793
[MemberName]	ON	2008-10-29 11:13:29.220
[_WA_Sys_00000003_0CBAE877]	ON	2008-10-29 11:28:32.313

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

- **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)

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

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)

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

Histogram Step Value

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

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

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

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]

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

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;
```

	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

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

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?

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

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

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

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]

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

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

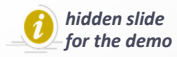
- Table scan (always an option)
- No indexes exist for SEEKING (no index with first name as the high-order element)
- What about scanning the NC index which has first names in it?
- What would be the best algorithm?

hidden slide for the demo

Each of these next few slides corresponds to a portion of the demo

How Do You See Them? (2)

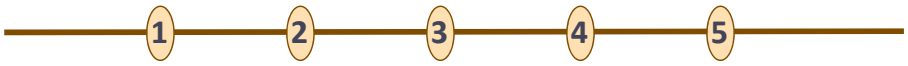
- **Query [sys].[indexes]**
  - Use OBJECT_ID and INDEX_ID as input to STATS_DATE
- **Query [sys].[stats] (preferred)**
  - Shows ALL statistics (index-level, column-level, hypothetical, etc.)
  - Use OBJECT_ID and STATS_ID as input to STATS_DATE
- **Use STATS_DATE ([object_id], [index_id]) in a query**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



Each of these next few slides
corresponds to a portion of the demo

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)

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'
```


RECOMMENDATION: 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/
    - Use @UpdateStatistics parameter / settings
  - Roll your own?
    - Programmatically evaluate the stat_header data
    - Use sys.dm_db_stats_properties: If 2-5% of the data has changed since last stats update OR (stats_date older than 1 week AND last update was sampled (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

Summary: How Does Auto-Updating Work?

- **You don't really want auto-updating to be your method of updating statistics**
  1) It could happen during production
  2) It's likely to be later than desired (causing performance problems along the way)
- **But, LEAVE IT ON (auto_update_statistics) as a safety measure**

- **Auto-updating in all versions prior to 2016**
  - Percentage-based threshold unless trace flag 2371
- **Auto-update in version 2016 and higher**
  - Dynamic auto-updating tied to table size (not percentage); same as having turned on 2371
- **Incremental stats – good, but not great**
  - Positive: less data has to be read for statistics updates (especially useful in ever-increasing tables)
  - Negative: statistic the optimizer uses for optimization is still limited to 200 steps and can become lossy for earlier partitions

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

hidden slide w/extra details

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

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

Asynchronous Statistics Updates

hidden slide w/extra details

- **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 ~20%; otherwise, likely to have been a problem earlier)**

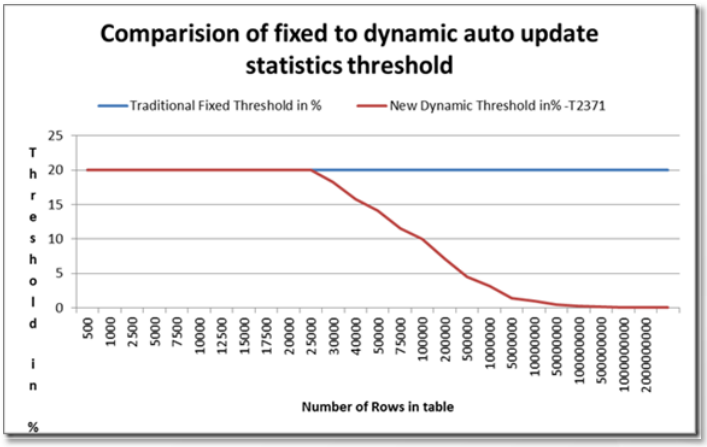
- **ALTER DATABASE databasename SET AUTO_UPDATE_STATISTICS_ASYNC ON**

- **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

hidden slide w/extra details

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



Incremental Stats Updates

hidden slide w/extra details

- 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
  - 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: 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

Review: Plan Problems Related to Statistics

- Statement selectivity
- What kinds of statistics exist
- How does SQL Server use statistics
- Creating additional statistics
- Updating statistics

Questions!

Review: What To Do Next?

What to do next?

- **Lots of content…**
  - Reviewing sooner rather than later will help!
  - Using a mix of mediums will help you see things in different ways and likely for some of you one will be better than another – multiple will be the way to make it stick!
- **A possibly review strategy?**
  - Print the slides + the whiteboard
  - Watch the course more interactively while taking notes and playing with demos
    - Small chunks – a module every day or two for a couple of weeks
  - Go through the recommended videos
  - Offer some "short courses" or "brown bag" lunch sessions where you deliver 30-45 minute topics to team members

    If you want to learn something, read about it. If you want to understand something, write about it. If you want to master something, teach it.

    – Yogi Bhajan
  - Always keep learning!

- **Get that Pluralsight code from Paul!**
- **Then, watch this course 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 course**
  SQL Server: Optimizing Stored Procedure Performance (Part 1)
  - http://pluralsight.com/training/Courses/Description/sqlserver-optimizing-stored-procedure-performance
  - Just over 7 hours on procedure caching and recompilation
- **Then, watch this course**
  SQL Server: Optimizing Stored Procedure Performance – Part 2
  - https://www.pluralsight.com/courses/sqlserver-optimizing-stored-procedure-performance-part2
  - Just under 4 hours on session settings and caching

Plan Caching and Recompilation 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.mspx
- **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

Optimizing Procedural Code on PASStv

- **Start with this blog post (and sample code):**
  - SQLskills SQL101: Stored Procedures
    - http://www.sqlskills.com/blogs/kimberly/sqlskills-sql101-stored-procedures/
  - THEN, Building High Performance Stored Procedures
    - http://www.sqlskills.com/blogs/kimberly/high-performance-procedures/
- **Then, watch my PASS Summit 2014 presentation through PASStv**
  - http://www.sqlpass.org/summit/2014/passtv.aspx
  - See, Day 1: 4:30pm
    - Dealing with Multipurpose Procs and PSP the RIGHT Way! - Kimberly Tripp
- **Then, check out this post: Stored Procedure Execution with Parameters, Variables, and Literals**
  - https://www.sqlskills.com/blogs/kimberly/stored-procedure-execution-with-parameters-variables-and-literals/

Statistics Resources

- **Whitepaper: Statistics Used by the Query Optimizer in Microsoft SQL Server 2008**
- **2013 – PASStv from PASS Summit: VLDBs and Data Skew (i.e. filtered statistics): http://www.sqlpass.org/summit/2013/PASStv.aspx?watch=li5HwaZF8tc**
  (NOTE: skip ahead 4 mins because they didn't tell me that they were *already* recording BEFORE the official session start time… and, well, the first 4 mins are not related to SQL)
- **Joe Sack's whitepaper: *Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator*: https://msdn.microsoft.com/en-us/library/dn673537.aspx**

SQLskills

THANK YOU!

SQLskills
immerse yourself in sql server