

sqlbits

Session: Saturday, October 3, 2020

Troubleshooting Query Plans: Part 2

Statistics Problems

Kimberly L. Tripp

President / Founder, SQLskills.com

Kimberly@SQLskills.com

@KimberlyLTripp

sqlbits





Consultant / Trainer / Speaker / Writer


- Author / instructor for SQL Server Immersion Events: IEPT01, IEPT02, 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

17 years: 2002-2019



When I'm not working with SQL

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



Kimberly L. Tripp
Founder / President, SQLskills



Kimberly@SQLskills.com



[KimberlyLTripp](https://twitter.com/KimberlyLTripp)



www.sqlskills.com/blogs/Kimberly

Overview

You determined it's NOT a cached plan... (Part 1)

- Selectivity and estimates
- Compatibility mode and cardinality estimate
- Specific examples of limitations with statistics
 - Sampling
 - Inaccuracies in the histogram
 - Uneven distribution

It Starts with Selectivity

- Not just based on the number of rows returned
- Always relative to the number of rows possible (based on input)
- Determined by your predicates:
 - A single predicate is relatively easy (but, still has potential problems)
 - Sampling
 - Existence
 - Accuracy (in larger and / or skewed sets)
 - Multiple predicates – quickly complicates the problem
 - Formulas exists in each of the models with optional trace flags to use different estimates
- Determined by your joins:
 - What is the likelihood of a match?
- **PROBLEM:** Lots of things to consider, lots of choices for SQL to make and NONE of those decisions work ALL the time

Compatibility Mode

- Originally designed to allow “time” for syntax changes to be made after update (time for you to fix your code but still leverage new features)
- Includes some optimizer fixes as of RTM
 - Prior to SQL Server 2014 you can get optimizer fixes with trace flag 4199
 - In SQL Server 2014 and higher – all optimizer hotfixes as of RTM are included with that database compatibility mode
 - In SQL Server 2016 and higher, additional (post-RTM) optimizer hotfixes can be enabled with the scoped configuration option QUERY_OPTIMIZER_HOTFIXES
- In SQL Server 2014, setting the compatibility mode also changed the cardinality estimation mode to the “new CE”
- SQL Server 2016 adds the “legacy CE” database scoped configuration option to separate these

What Does the Cardinality Estimation Model Do?

- Imagine you sell “products”
 - Your products have a “shape”
 - 33% (1/3) of your products are round
 - Your products have a “color”
 - 25% (1/4) of your products are red
- How many of your products are round AND red?

Legacy CE

$$1/4 * 1/3 = 1/12$$

New CE

$$1/4 * 1/9 = 1/36$$

(exponential backoff)

What if?

All red products are round?
“minimum value estimation”

Legacy CE: TF 4137
New CE: TF 9471
2016 SP1+ query hint



- This is what the CE effects – the calculations and estimations that are made when relationships are unknown...
 - Multicolumn indexes/statistics can help but are not always possible (across tables, for example)

Cardinality Estimation Models (1 of 2)

- Legacy CE model: refers to the model introduced with SQL Server 7.0 and used through SQL Server 2012
 - Will be used in SQL Server 2014 for databases whose compatibility mode is not 120 (120 = SQL Server 2014)
 - 80 = SQL Server 2000
 - 90 = SQL Server 2005
 - 100 = SQL Server 2008 and SQL Server 2008 R2
 - 110 = SQL Server 2012
 - 120 = SQL Server 2014
 - 130 = SQL Server 2016
 - 140 = SQL Server 2017
 - 150 = SQL Server 2019
 - If the database compatibility mode is 120 then you can access the Legacy CE when trace flag 9481 has been supplied
 - Add this to a single query to revert to the Legacy CE for only a specific query
 - `OPTION (QUERYTRACEON 9481)`
 - Run this in a script or a session to test multiple queries more easily
 - `DBCC TRACEON (9481);`

Cardinality Estimation Models (2 of 2)

- “New CE” model: refers to the model introduced in SQL Server 2014
 - Used when the compatibility mode of the database is 120 or higher (SQL Server 2014)
 - Used when the compatibility mode of the database is NOT 120 but trace flag 2312 has been supplied
 - Add this to a single query to test the impact for only a specific query
 - **OPTION (QUERYTRACEON 2312)**
 - Run this in a script or a session to test multiple queries more easily
 - **DBCC TRACEON (2312);**
 - In SQL Server 2016, used when your database compatibility mode is 120 or higher AND you have not enabled the “legacy CE” through the scoped configuration option
- Resource: Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator by Joe Sack <http://bit.ly/1mFDB2t>
- My general recommendation for SQL Server 2014 and higher...

Migration / Upgrade \Rightarrow Testing

- **Backup / restore to “upgrade”**
 - This restores an EXACT page-level copy of the database backed up
 - The database does get “upgraded” but the structures are not changed in any way
 - Your backed up compatibility mode is what’s restored (or, the minimum supported)
 - Consider rebuilding indexes (especially if restoring from a version PRIOR to SQL Server 2012)
 - Consider updating all statistics
- **Restoring to SQL Server 2014**
 - Consider leaving compatibility mode as restored (“legacy CE”)
 - During performance testing, try “new CE” in troubleshooting with QUERYOPTION
- **Restoring to SQL Server 2016 and higher**
 - Restore to desired version of SQL Server
 - Change to THAT version’s compatibility mode (130 for 2016 | 140 for 2017 | 150 for 2019)
 - Set the “legacy CE” using the database scoped configuration

Limitation of Statistics

- Sampling
- Inaccuracies in the histogram
 - Concerns with table size
 - Concerns with data distribution
- Uneven distribution and column correlations

Sampling: Always Good?

- Using *actual* showplan tooltip – estimate v. actual rows
- When it's not a cached plan... if query performance is poor AND the actual is significantly OFF from the estimate then you might want to verify the statistics creation (rows v. rows scanned)
- If statistics were based on a sampling and performance is improved after statistics have been updated with FULLSCAN, then you might want to turn off auto updating for this index (using STATISTICS_NORECOMPUTE at the index-level or NORECOMPUTE at the statistics-level) and schedule an UPDATE STATISTICS WITH FULLSCAN instead

UPDATE STATISTICS ...
WITH FULLSCAN

Concerns Around the Histogram

- Stores ACTUAL values from the FIRST (and only first) column of the key
 - Sometimes referred to as the leading column of the index
 - Sometimes referred to as the high-order element of the index
- Never stores more than 201 steps (up to 200 distinct/actual values plus 1 NULL values row – if the column allows NULLs)
- Even when your table has more than 200 distinct values, you may have fewer than 200 steps
- Values chosen aren't evenly distributed
 - Internally, during statistics creation, SQL compresses steps to make room for more. They try to track "interesting" values/anomalies but the more compression that occurs, the more they lose outliers
- Has the best information – but how good is it?
 - This is really the issue. And, unfortunately, it depends – mostly on how skewed the data distribution is...

Important: Describes the entire table... even when partitioned

Data Distribution Matters

- Even distribution is easy
 - Think about “line items per sales order”
 - That’s probably *fairly* consistent at 2 or 3
- Un-even distribution is HARDER
 - Think about sales per product
 - This is all over the place – and varies over time as well...
 - Think about sales per customers
 - Again, all over the place – and, again, varies over time...
- The histogram does a MUCH better job having steps and average distribution per step but what if there are well over 200 distinct values (tens of thousands) and millions of rows with heavy skew between steps?
 - Simply put, the averages just aren’t going to cut it anymore...

Resources

- If you're interested in more details on filtered stats...
 - <https://www.pass.org/Learn/Recordings/Listing.aspx>
 - Search for "Tripp"
 - Scroll and then select:
 - Skewed Data, Poor Cardinality Estimates, and Plans Gone Bad
 - Level: 300, Spotlight Session (90 minutes), Language: English
 - PASS Summit 2013



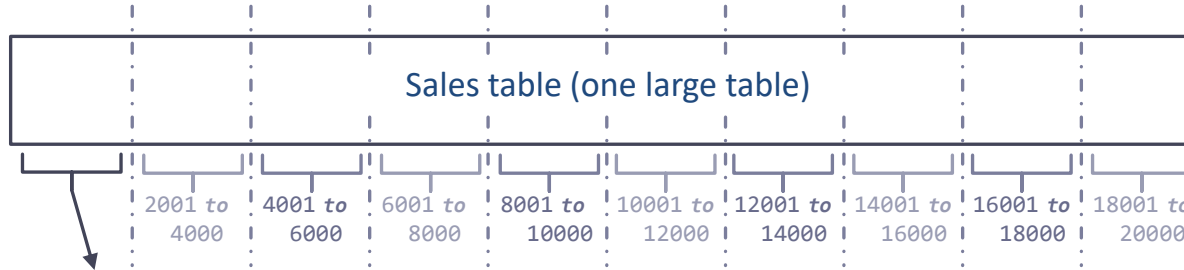


Demo (in the PASS Session): Key Points

- Very useful to help the optimizer assess the usefulness of an existing index (whose histogram is not as accurate due to skew [and probably table size])
- Take a *slightly* skewed example: sales by customer
 - Sales: 30,923,776
 - Customers: 18,484
 - Average = $30,923,776 / 18,484 = 1,673$
 - All density (from density_vector of statistics) = $5.410084E-05$ multiplied by rows = 1,673
 - Skew \Rightarrow high: 30,000+ (only ~6), Low: ~500 (thousands)
 - Not every value can possibly be represented in 200 steps (for histogram)
 - Occasionally, they'll be wrong – average might not be good enough... enter \Rightarrow filtered statistics
- Not an index, just a statistics blob... (smaller, easier to maintain, more accurate over that set)

Filtered Statistics for the Entire Table? (1)

- We've determined there's skew – now what?
 - Manually create a whole bunch of filtered stats?
- Here's the idea – imagine 20,000 customers (1 to 20,000)



```
CREATE STATISTICS FilteredStat
ON [schemaname].[tablename] ([columnname])
WHERE [columnname] >= 1
AND [columnname] < 2000
```


Filtered Statistics to the Rescue

- Filtered statistics can help describe the data – allowing an index to be used even when the statistics on the index are “lossy”
- Great for helping “point” queries
- Can “cover” the entire table
- You can use my code...
 - Blog post: SQLskills procs to analyze data skew and create filtered statistics
 - <https://bit.ly/3c4n4s2>
- What about maintenance?
 - AFTER the index has been maintained (a REBUILD not a reorg), the statistics will be updated
 - Then, just RE-CREATE the additional / associated filtered stats using the procedure:
`[sp_SQLskills_CreateFilteredStats]`

Evenly Distributed Data?

- Scenario
 - You have 90 rows in a table
 - 10 rows have a value of x for column 6
 - Where (physically) in this table are these 10 rows?
- They could be evenly distributed throughout the table...



- But, what if they're not?



Scenario: Unevenly Distributed Data

- Scenario

- AdventureWorksDW2012: FactInternetSales has 60,398
- 2,541 rows have a null for ShipDateKey
- $60,398 / 2,541 = 23.7$ (1 in 23.7 rows is NULL)
- Nonclustered index on ShipDateKey
- Nonclustered index on OrderDateKey

- What does SQL Server do?

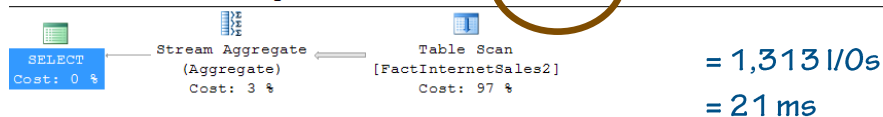
```
SELECT MIN(fis.OrderDateKey)
FROM dbo.FactInternetSales2 AS fis
WHERE fis.ShipDateKey IS NULL
```

Even Distribution: Always Even??

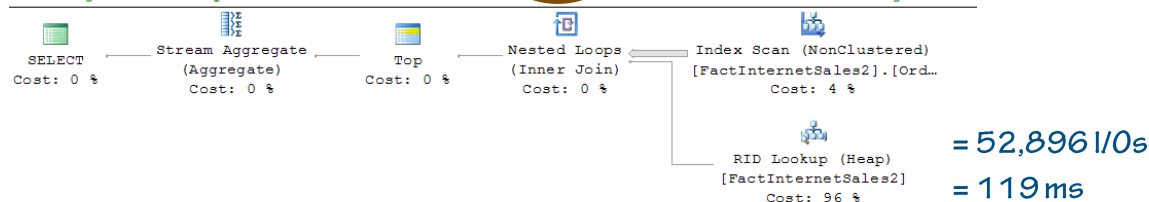
- Table scan is always an option
- Use an index on ShipDateKey to look up the actual date for all orders where ShipDateKey is NULL
 - This means a bookmark lookup must be run for EVERY NULL so that we can get the OrderDateKey
 - The worktable then needs to be sorted to find the lowest order date
- Use an index on OrderDateKey as only 1 on 23.7 sales have a NULL for ShipDateKey
 - SQL Server estimates that they'll find a NULL within 23.7 rows and they won't need a worktable
 - This sounds better...
- But the rows are not evenly distributed!

Evenly Distributed Data??

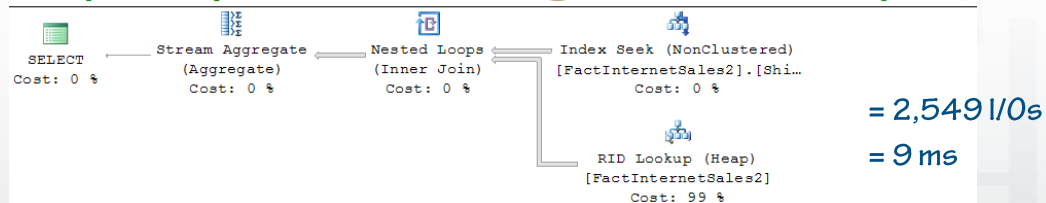
Query 1: Query cost (relative to the batch): 21%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX (0)) WHERE ShipDateKey



Query 2: Query cost (relative to the batch): 2%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WHERE ShipDateKey IS NULL -- 528
Missing Index (Impact 95.4001): CREATE NONCLUSTERED INDEX [<Name of Missing Index



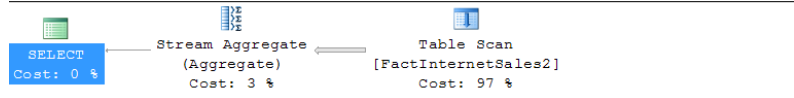
Query 3: Query cost (relative to the batch): 77%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX(ShipDateInd)) WHERE :
Missing Index (Impact 99.863): CREATE NONCLUSTERED INDEX [<Name of Missing Index,



Evenly Distributed Data??


Query 1: Query cost (relative to the batch): 21%

```
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX (0))
```



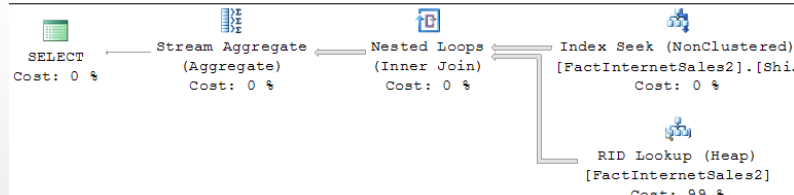
Query 2: Query cost (relative to the batch): 2%

```
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WHERE ShipDateKey
Missing Index (Impact 95.4001): CREATE NONCLUSTERED INDEX [<Name of
```



Query 3: Query cost (relative to the batch): 77%

```
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX(ShipDateKey))
Missing Index (Impact 99.863): CREATE NONCLUSTERED INDEX [<Name of
```



Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	52773
Actual Number of Batches	0
Estimated I/O Cost	0.104606
Estimated Operator Cost	0.0033464 (4%)
Estimated Subtree Cost	0.0033464
Estimated CPU Cost	0.0665948
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	23.7694
Estimated Row Size	19 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4

Object

[AdventureWorksDW2012].[dbo].[FactInternetSales2].
[OrderDateInd]

Output List

Bmk1000, [AdventureWorksDW2012].[dbo].
[FactInternetSales2].OrderDateKey

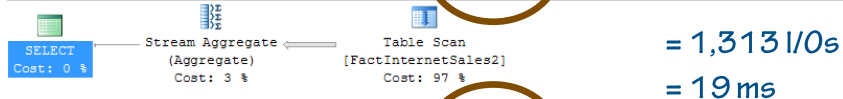
Always Better: Indexes

- Statistics cannot be used to directly access data but:
 - They can only HELP the optimizer determine the best plan
 - They can help determine best join order
- Statistics are just estimates: they help MOST of the time but:
 - There are limitations
 - They take time to create, update, store...
- Indexing can often be A LOT better...

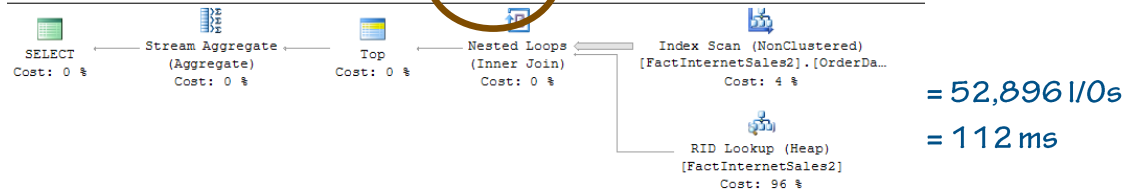
```
CREATE INDEX ShipDateOrderDateInd_SeekableForMin  
ON FactInternetSales2 (ShipDateKey, OrderDateKey)
```

Always Better: Indexes

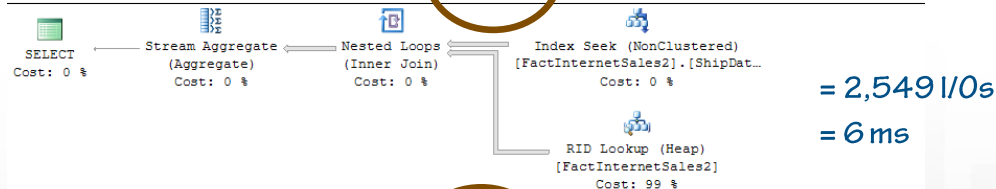
Query 1: Query cost (relative to the batch): 21%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX(0)) WHERE ShipDateKey IS NU



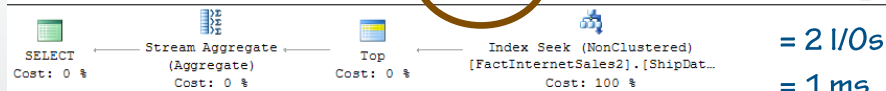
Query 2: Query cost (relative to the batch): 2%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX (OrderDateInd)) WHERE ShipDa



Query 3: Query cost (relative to the batch): 77%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX(ShipDateInd)) WHERE ShipDate



Query 4: Query cost (relative to the batch): 0%
SELECT MIN(OrderDateKey) FROM FactInternetSales2 WITH (INDEX (ShipDateOrderDateInd_Seekal



Always Better: The RIGHT Indexes

- Every SINGLE QUERY can be indexed to make THAT SINGLE query fast
- You can't index EVERY query unless you're read-only/decision support, and even then you're limited by disk space and memory (but, you'll definitely have more indexes in read-only environments)
 - Not all indexes are useful
 - Be sure to regularly re-check the usage statistics as these often change over time
- The better choice is prioritizing and determining which queries really need indexes!
 - Sometimes the problem IS statistics...
 - Are they up-to-date?
 - Sampling?
 - Filtered stats?
 - Sometimes statistics are enough to HELP determine the usefulness of different indexes

Summary: Estimates from Statistics

- Impossible to estimate correctly – ALL of the time
- Assumptions may work for some scenarios and are AWFUL for others
- Don't worry as much about the precise calculation
- Things to think about when you have "statistics" problems:
 - Are they current?
 - Are they being updated automatically with AUTO_UPDATE_STATS or manually with an automated script?
 - Should we increase the frequency for updating these statistics?
 - Were they created using sampling?
 - Does it improve the estimation if I use FULLSCAN instead of sampling?
 - Which CE model are you using?
 - Does using the other CE model fix the problem?
 - Can you rewrite the query and get a better estimate / plan?
 - Can you create a new index (or better [maybe filtered] statistics) to help improve the estimate / plan?

Review

You determined it's NOT a cached plan... (Part 1)

- Selectivity and estimates
- Compatibility mode and cardinality estimate
- Specific examples of limitations with statistics
 - Sampling
 - Inaccuracies in the histogram
 - Uneven distribution



- Demo code/samples: SQLskills, Resources, Demo Scripts and Sample Databases
- Courses on Pluralsight: www.pluralsight.com
 - **SQL Server: Indexing for Performance (7 hr., 11 min)**
 - SQL Server: Why Physical Database Design Matters
 - SQL Server: Optimizing Ad Hoc Statement Performance
 - SQL Server: Optimizing Stored Procedure Performance (Parts 1 and 2)

Questions?



Don't forget to complete an online evaluation!

Troubleshooting Query Plans: Part 2

Statistics Problems

Kimberly L. Tripp

Your evaluation helps organizers build better conferences
and helps speakers improve their sessions.

The logo for sqlbits, featuring the text "sqlbits" in a bold, orange, sans-serif font. The "q" and "l" are lowercase, while "bits" is lowercase. The logo is set against a background of light blue squares of varying sizes.

Thank you!