SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 11: Cardinality Estimation Issues

Kimberly L. Tripp

Kimberly@SQLskills.com

Overview

- **Selectivity and estimates**
- **Query complexity**
- **Estimates from statistics**
  - Sampling
  - The histogram
  - Filtered statistics
  - Uneven distribution
- **Migrations / Upgrades / Regressions**
- **Appendix: Changes to Cardinality Estimation (CE) in SQL 2014**

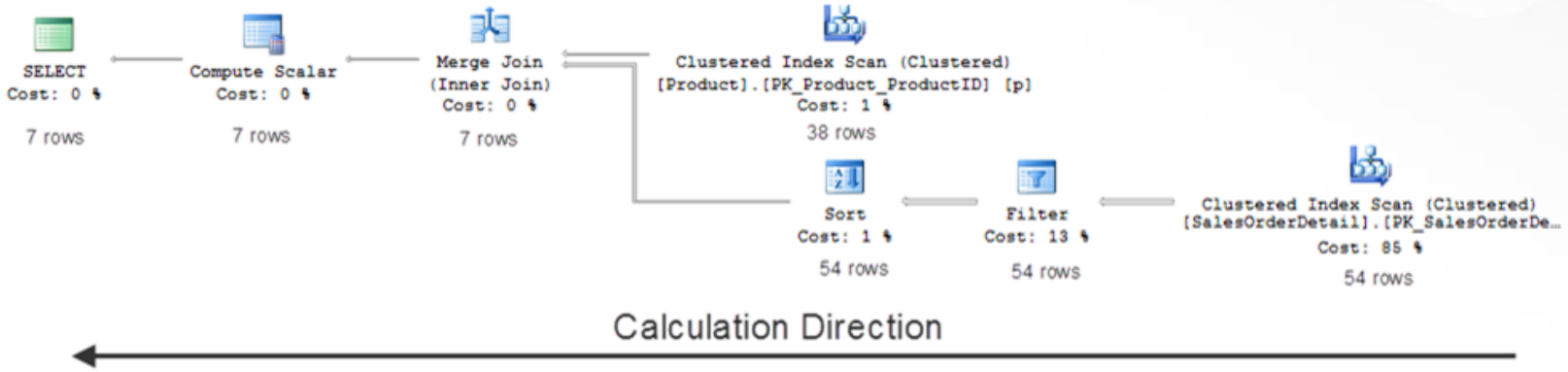
It Starts with Selectivity

- **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**

hidden slide
w/extra details

Selectivity Makes or Breaks Your Plan

- The more complicated the predicates, the joins, etc. the harder it is to calculate how those translate up the chain of operations



- In complicated plans, each individual operator estimates its cost based on the estimate passed as input
- Each estimate is a major factor in deciding:
  - Physical operator algorithms
  - Plan shapes (such as join orders)
- If the estimates are wrong then the entire plan can be a mess…

Legacy CE Model Assumptions (1 of 2)

- **A value queried is expected to exist (inclusion)**
  - If there's a step = yes, it exists and the step's value will be used
  - If the value falls within a step then the average (**uniformity**) will be used
    - For values that don't exist – this average might be WAY off
      - Imagine that a step covers the values from 101 to 200
        - There are 250K rows in that range
        - There are 5 actual values = 115, 135, 167, 172, 195
        - The average will be 250K / 5 = 50K
        - The "estimated" number of rows for EVERY value possible (102, 103, 104, etc.) will be 50K even though there are only 5 values (they expect that you're querying a EXISTING value)

Legacy CE Model Assumptions (2 of 2)

- **Statistics from separate columns are not correlated (independence)**
  - Column city and column state are NOT related
    - 1/3 of your data is for IL and 1/4 of your data is for Chicago
    - How much of your data is for Chicago, IL?
      - We think ¼
        - Minimum (1/4) must be correct because we know ALL of the Chicago rows are in IL
      - The CE thinks 1/12
        - .33 x .25 = .0825
    - 1/5 of your data is for KS and 1/8 of your data is for Kansas City
    - How much of your data is for Kansas City, KS?
      - This is more interesting though, right? Harder to estimate…

NOTE: Multi-column statistics or a composite index can solve these problems. And, there are trace flags to handle the first situation: TF 4137 in legacy CE / TF 9471 in new CE; this tells the QO to use the minimum and not the calculation / independence.

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?

Sampling: Always Good?

- Using *actual* showplan tooltip – estimate v. actual rows
- 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
```

Changes to Sampling Over the Versions

- **Prior to SQL Server 2016, sampling was not parallelized**
  - Often, prior to 2016, it was better (even faster) to FULLSCAN with parallelism (when possible / off hours) than it was to serialize with sampling
  - In 2016 and higher, parallelism is used when the database compatibility model is 130 or higher
- **Default sample size is tied to the size of the data**
- **If you find it's not yielding good numbers, you can TRY increasing sample size until you get a good percentage (note, this may still have problems for other values / steps)**
  - Once you find an adequate sample size, use PERSIST_SAMPLE_PERCENT = ON to retain that sample size on subsequent rebuilds, including auto updates
- **Using RESAMPLE, on the whole table uses the persisted sample sizes (per statistic) to update statistics**
- **To sample or not to sample…**

Very Large Tables (VLTs) Present Problems

- **There are numerous problems that large tables present, some of the more frustrating are related to statistics**
  - Incremental builds in partitioned tables help but don't solve the accuracy problem with the histogram
  - The accuracy problems created because of the limited number of steps in the histogram (*more on this coming up*)
- **What if you had MORE but smaller tables…**
  - Current month (highly volatile / relatively small)
    - Easier to update more frequently
    - Fewer problems with estimates and min / max values
  - Prior months (less volatile / relatively small when isolated)
    - Easier to update with updates less and less frequent as time moves forward
  - Eventually, data moves into read-only state
    - Final 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…

Demo

Understanding the histogram

- What information is stored in the histogram?
- How does SQL Server use it?

Demo: 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 ⇒ 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 ⇒ filtered statistics
- **Not an index, just a statistics blob… (smaller, easier to maintain, more accurate over that set)**

Assessing Level Of Data Skew

- **Can we access the histogram – programmatically?**

```
INSERT TemporaryWorktable
    EXEC ("DBCC SHOW_STATISTICS … WITH HISTOGRAM")
```

  - Should be easy enough? (er, famous last words)
    - RANGE_HI_KEY (if you want to analyze it against the base table, needs to be the same data type as the base table)
    - LOTS of dynamic string execution and different lookups to reconstruct data types (including collations)
      - Which, by the way – has a slightly different syntax when in a CAST clause vs. in a column definition
- **End result:** `[sp_SQLskills_AnalyzeColumnSkew]`
  - Plus, `[sp_SQLskills_AnalyzeAllLeadingIndexColumnSkew]`
- **Both need to be added to master and marked as system objects:**

```
EXEC [sys].[sp_MS_marksystemobject]
                    'sp_SQLskills_AnalyzeAllLeadingIndexColumnSkew'
```

Assessing Level of Data Skew

- **Parameters and uses for this:**

```
[sp_SQLskills_AnalyzeColumnSkew]
    @schemaname       sysname = NULL
  , @objectname       sysname = NULL
  , @columnname       sysname = NULL
  , @difference       int = 1000
    -- Min diff between average and largest difference in that step
  , @factor           decimal(5, 2) = 2.5
    -- Min factor of the difference against the average
  , @numofsteps       tinyint = 10
    -- Min number of steps that have to meet this/these
  , @percentofsteps   tinyint = 10
    -- Minimum PERCENT of steps that have to meet this/these
  , @keeptable        char(5) = 'FALSE'
    -- If TRUE keeps ALL of the worktables
  , @tablename        nvarchar(520) = NULL OUTPUT
    -- Fully delimited name of the worktable in tempdb
```

- **Lots of options and relatively low impact (requires an index that has the column specified [@columnname] as the leading column)**

Scripts: Key Points

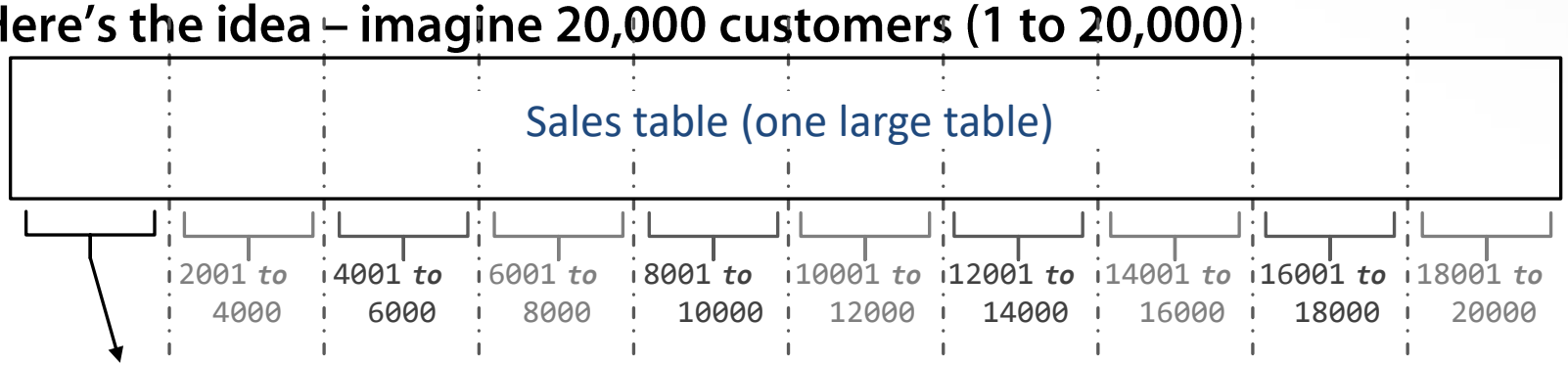
- **Not everyone will have skewed data**
- **Might already suspect you have a problem**
  - Poor estimates where you've tried:
    - Updating statistics more frequently
    - Updating statistics with FULLSCAN more frequently
    - Adding OPTION (RECOMPILE) to your code
    - But… it still didn't work – the estimates were still *off*

  NOTE: There are still other things it could be. But, here's where this proc can really help you to understand what your data really looks like!
- **Target only specific (probably large [over 50-60GB]) tables**
  - NOTE: It's not the size of the table that's important here – even a small table that's 100-200MB can show signs of skew. But, the performance problems that result from a small table just aren't as noticeable.
- **Don't forget to clean up the worktables in tempdb:**

```
EXEC [sp_SQLskills_HistogramTempTables] @management = 'DROP'
```

Filtered Statistics For The Entire Table? (1)

- **We've determined there's skew – now what?**
  - Manually create a whole bunch of filtered stats?
    - How do we break down the data?
    - How do we account for new rows?
    - How do we maintain this?
- **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 For The Entire Table? (2)

- **So, that seemed simple**
  - Not all data is as simple as an ever-increasing integer
  - Most data actually changes… (what happens with new rows over 20,000)
  - How would we automate this process?
- **End result:**

```
[sp_SQLskills_CreateFilteredStats]
```

- **Needs to be added to master and marked a system object:**

```
EXEC [sys].[sp_MS_marksystemobject] 'sp_SQLskills_CreateFilteredStats'
```

- **Requires a couple of other procedures:**
  - `[sp_SQLskills_CreateFilteredStatsString]`
  - `[sp_SQLskills_DropAllColumnStats]`

Automating Filtered Statistics Creation

- **Parameters and uses for this:**

```
[sp_SQLskills_CreateFilteredStats]
    @schemaname      sysname = NULL
  , @objectname      sysname = NULL
  , @columnname      sysname = NULL
  , @filteredstats   tinyint = 10
    -- Number of filter chunks to define
  , @everincreasing  bit = 0
    -- depending on whether or not your table is ever increasing
  , @maxforfs        sql_variant = NULL
    -- Projected value to extend out until the next redistribution
  , @fullscan        varchar(8) = NULL
    -- Generate the statistic using a FULLSCAN or SAMPLE
  , @samplepercent   tinyint = NULL
    -- If @fullscan = SAMPLE, do you want to override SQL's sample
```

- **Lots of options and relatively low impact to create (will leverage an existing index; fullscan might require that index to end up in cache)**
- **Might not give you any benefits and it will need to be maintained!**

Scripts: Key Points

- **Must supply values, cannot use only the defaults**
- **You'll need to estimate a new max value based on:**
  - How long you want these filtered statistics to live
  - How much data churn/change you see in these values
- **Each time you run this, ALL SQLskills-created, column-level, filtered statistics previously created are deleted.**
  - Other, user-created, filtered statistics are NOT deleted – you will want to remove those as they might overlap with these and cause problems
- **If you want to clean up the SQLskills-created, filtered stats:**

```
EXEC [sp_SQLskills_DropAllColumnStats]
  @schemaname = N'dbo'
, @objectname = N'factinternetsales'
, @columnname = N'customerkey'
, @dropall = N'TRUE'
```

How Many Filtered Indexes/Stats?

Per Table

- **SQL Server 2000**
  - Columns: 1,024
  - Nonclustered indexes: 249
  - Statistics: 249 (shared with nonclustered)
- **SQL Server 2005**
  - Columns: 1,024
  - Nonclustered indexes: 249
  - Statistics: 2,000 (referenced in sys.stats)
- **SQL Server 2008/R2**
  - Columns: 30,000
  - Nonclustered indexes: 999
  - Statistics: 10,000 (30,000 in R2)

Valid Index IDs

- **Table always has at least one index id**
  - For a heap the index id is always 0
  - For a clustered table the index id is always 1
  - Nonclustered indexes *can* start at 2
- **999 nonclustered indexes per table**
  - Index ids start with 2 but statistics use the same counter range
  - Index ids 251 through 255 are reserved (from prior use cases)
    - NOTE: 255 was used in earlier releases but 251-254 have never been used (at least not that I know of ☺)
- **30,000 statistics per table (SQL Server 2008 R2+)**
- **Index and statistics ids run from 2 to 250 and from 256 to 31005**
  - NOTE: The datatype used is int (sys.indexes.index_id and sys.stats.stats_id). Technically they *could* support more...

This Is So Cool…

But, It Won't Always Work… Sigh

- **Session setting requirements for filtered indexes do NOT apply to the creation OR usage of filtered statistics**
- **Interval subsumption problems**
  - Might be able to do a query rewrite but this is kind of a nightmare
- **Might be able to add recompile**
- **Might be able to add a plan guide**

- **To be honest, there are better – architectural ways – to handle very large tables with skew problems**
  - Don't have very large tables
    - Create multiple tables and union them together as a view
      - Partitioned Views (UNION ALL and constraints)

hidden slide w/extra details

Session Settings Requirements

APPLY Only to Filtered Indexes (NOT Filtered Statistics)

- See BOL topic: Set Options that Affect Results
- Session settings control behavior – and the result of some computations
- Data in these persisted structures must be consistent (which is why these apply to filtered indexes)
- Session settings that must be on:
  - ANSI_NULLS
  - ANSI_WARNINGS
  - QUOTED_IDENTIFIER
  - CONCAT_NULL_YIELDS_NULL
  - ANSI_PADDING
  - ARITHABORT
- Session setting that must be off:
  - NUMERIC_ROUNDABORT

```
Msg 1934, Level 16,
  State 1, Line 1
CREATE INDEX failed because the
following SET options have
incorrect settings:
'QUOTED_IDENTIFIER'. Verify that
SET options are correct for use
with indexed views and/or indexes
on computed columns and/or
filtered indexes and/or query
notifications and/or XML data
type methods and/or spatial index
operations.
```

Client Consistency Requirements

APPLY Only to Filtered Indexes (NOT Filtered Statistics)

- **Consistency with table(s), view and the clustered index (on the view) creation OR table and the filtered index**
  - All tables on which the view is based, the view itself and the index must be created with the correct session settings set or the index cannot be created on the view
- **Consistency with base table access**
  - All INSERT, UPDATE and DELETE statements must be executed with correct session settings or the insert, update or delete will fail
- **Consistency with query access**
  - All queries that SELECT against views with indexes (or tables with filtered indexes) must access them with the correct session settings set otherwise the data will need to be recalculated, rejoined or recomputed

Interval Subsumption (1 of 2)

Filter use (per query)

- **Filters (indexes/statistics) cannot be used when the query's predicate is not a subset of a SINGLE filter interval:**
  - Filtered index: January data
    - `WHERE [date] >= '20110101' AND [date] < '20110201'`
  - Filtered index: February data
    - `WHERE [date] >= '20110201' AND [date] < '20110301'`
  - Predicates and their usage:
    - `WHERE [date] = '20110115'` **YES**
    - `WHERE [date] BETWEEN '20110105' AND '20110115'` **YES**
    - `WHERE [date] BETWEEN '20110115' AND '20110215'`

      NO, cannot use multiple filtered objects
- **BAD NEWS: This is why there's no such thing as partition-level statistics. SQL Server cannot combine the filtered indexes and use them individually. So, you need a table-level index (that's partition-aligned) but the statistics are less accurate.**

Interval Subsumption (2 of 2)

Filter use (per query)

- **Filtered statistics examples:**
  - Filtered statistic over a range:
    - `WHERE ([CustomerKey]>= 11000 AND [CustomerKey] < 11566 )`
  - Filtered statistic over a range:
    - `WHERE ([CustomerKey]>= 11566 AND [CustomerKey] < 12363)`
  - Predicates and their usage:
    - `WHERE [c].[CustomerKey] IN (11509, 11503, 11123)` **YES**

      YES, all values are in only ONE filtered statistic
    - `WHERE [c].[CustomerKey] IN (11509, 12345)`

      NO, cannot use multiple filtered objects

      WORKAROUND: Use dynamic string execution to UNION ALL the individual values
  - Predicates and their usage:
    - `WHERE [c].[CustomerKey] BETWEEN 11500 AND 11600`

      NO, cannot use multiple filtered objects

      GOOD NEWS: The larger the range, the less you need to use the more accurate values (averages are fine when they're over larger ranges)

Demo

Interval subsumption

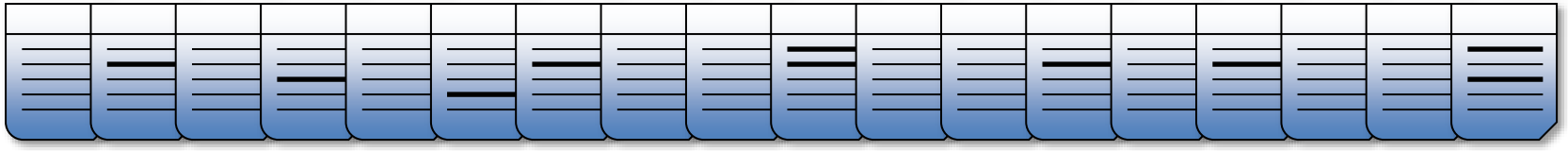
- When won't SQL Server use a filtered object?
- Is there a workaround?

The Bad News…

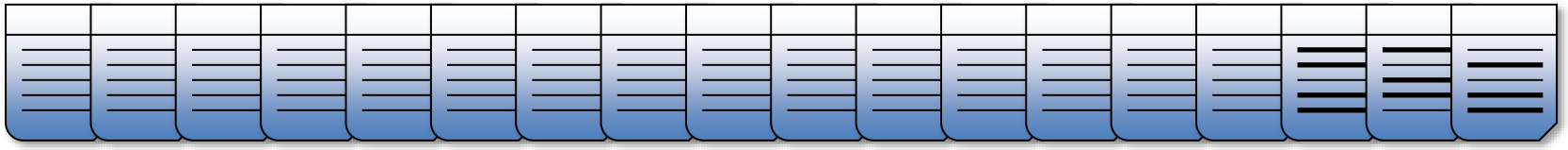
- **Filtered indexes/filtered stats: stats may get HORRIBLY out of date:**
  - SQL Server ONLY tracks a colmodctr NOT related to the filtered set or size. As a result, the stats (even filtered stats) are only invalidated when the threshold has been reached
  - For better performance, you need to automate and control their updates (do NOT rely on "auto update statistics")
  - Good news is that they're relatively small and creating a job to automated them is relatively easy!
  - See this blog post: https://www.SQLskills.com/BLOGS/KIMBERLY/post/Filtered-indexes-and-filtered-stats-might-become-seriously-out-of-date.aspx (http://bit.ly/1knEE2)
- **Stored procedures and sp_executesql will not use filtered objects unless you recompile the statement [add OPTION (RECOMPILE)]**
- **Forced parameterization (the database option) will generalize statements and many filters won't be eligible during optimization**
- ***Similar summary, don't go wild with this feature; it has powerful – but specific – uses!***

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

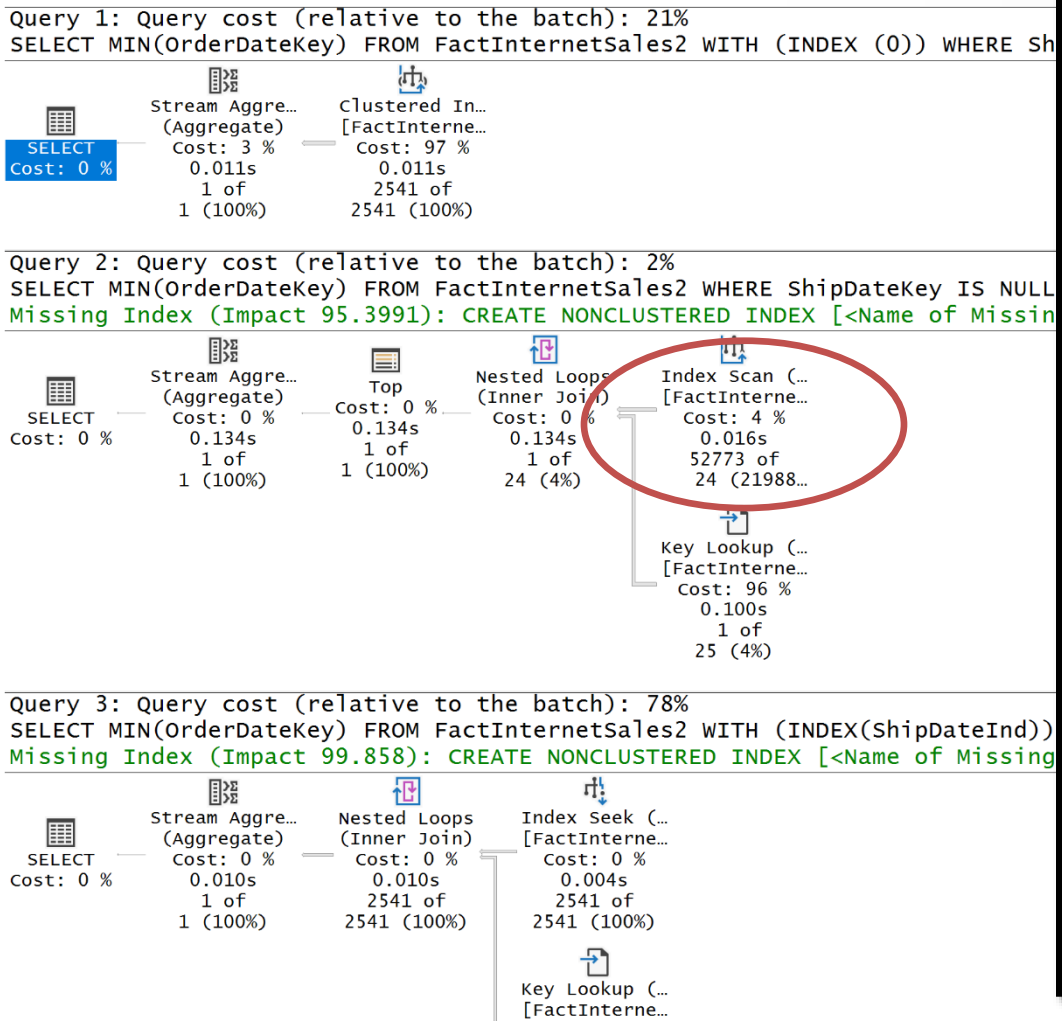
- **Scenario**
  - AdventureWorksDW201x: FactInternetSales has 60,398
  - 2,541 rows have a null for ShipDateKey
  - 60,398/2,541 = 23.7 (1 in 23.7 rows is NULL)
  - Nonclustered index on ShipDateKey
  - Nonclustered index on OrderDateKey
- **What does SQL Server do?**

```
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??



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
Number of Rows Read	52773
Actual Number of Rows for All Executions	52773
Actual Number of Batches	0
Estimated I/O Cost	0.166829
Estimated Operator Cost	0.0033699 (4%)
Estimated CPU Cost	0.0665948
Estimated Subtree Cost	0.0033699
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	23.7694
Estimated Number of Rows to be Read	60398
Estimated Row Size	30 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4

Object

[AdventureWorksDW2016].[dbo].[FactInternetSales2].[OrderDateInd]

Output List

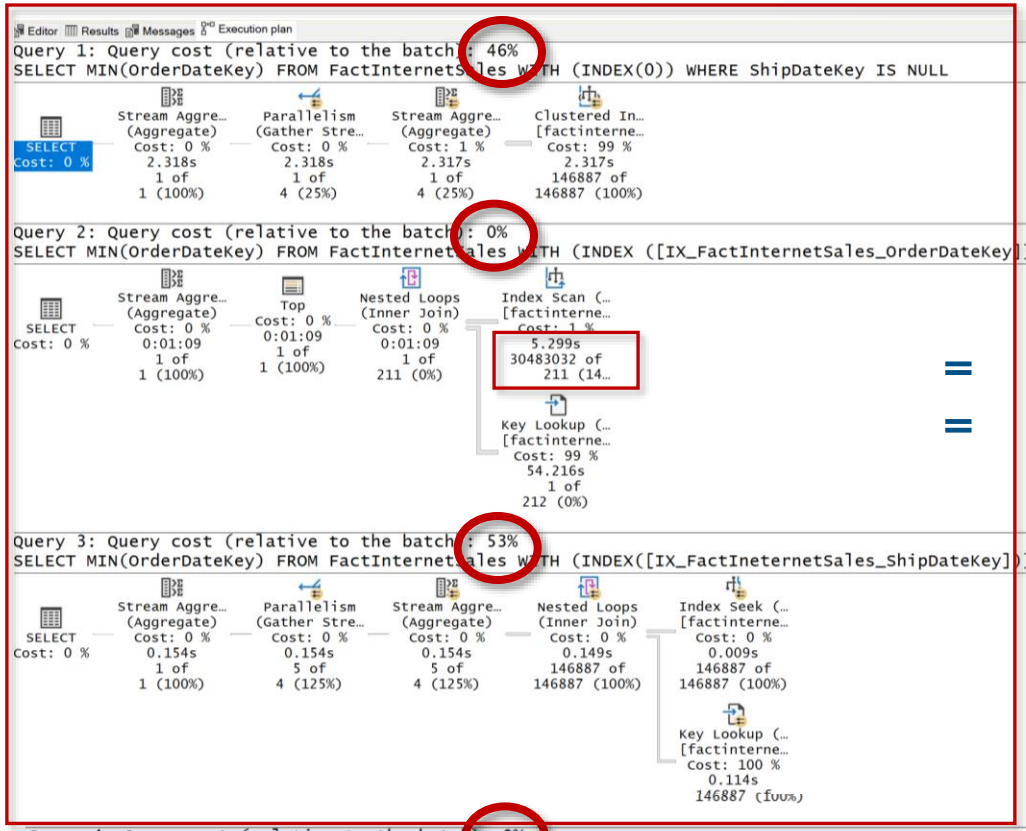
[AdventureWorksDW2016].[dbo].[FactInternetSales2].OrderDateKey, [AdventureWorksDW2016].[dbo].[FactInternetSales2].SalesOrderNumber, [AdventureWorksDW2016].[dbo].[FactInternetSales2].SalesOrderLineNumber

Always Better: Indexes

- **Statistics cannot be used to directly access data but:**
  - They can only HELP the optimizer determine a better 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 [dbo].[FactInternetSales2]
          ([ShipDateKey], [OrderDateKey])
```

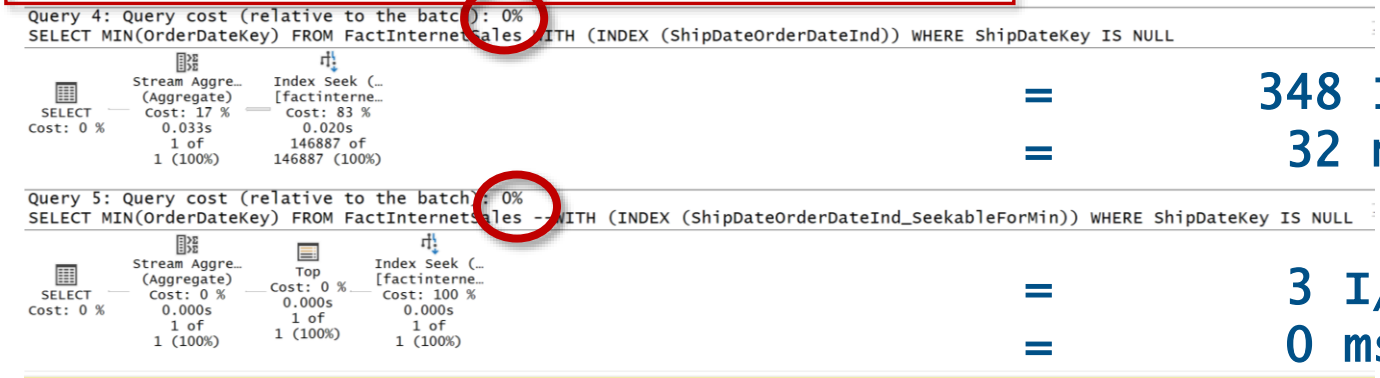
Always Better: Indexes



= 478,678 I/Os
= 2,221 ms

= 123,895,432 I/Os
= 67,285 ms

= 606,948 I/Os
= 250 ms



= 348 I/Os
= 32 ms

GREEN HINT

= 3 I/Os
= 0 ms

= 1 ms

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 (but that's about it…lots of indexes – yeah!)**
- **But a better choice is prioritizing and determining which queries really need indexes!**

- **But, think about the *least expensive* fixes first**
  - Is it a cached plan? Maybe just a caching problem!
  - Is it out-of-date stats? Might just need to increase the frequency of updates!
  - Is it sampling? Change to fullscan!
  - Or, it is more difficult → add statistics, change code, add indexes

Compatibility Model and 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 (shows as 70 in showplan)**
- **New CE model: refers to the model introduced in 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
- **Not every version supports every compatibility model**
  - When restored, a database KEEPs the compatibility mode that's set in the database when it was backed up (it is NOT upgraded)
    - Exception: if the compatibility mode you were using is no longer supported then it's updated to the minimum compatibility model supported by the version of SQL Server where you restored
- **Resource: Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator by Joe Sack http://bit.ly/1mFDB2t**

Compatibility Model and Cardinality Estimation Models (2 of 2)

- From SQL Server BOL:
  ALTER DATABASE (Transact-SQL) Compatibility Level

Product	Database Engine Version	Default Compatibility Level Designation	Supported Compatibility Levels
SQL Server 2019 (15.x)	15	150	150, 140, 130, 120, 110, 100
SQL Server 2017 (14.x)	14	140	140, 130, 120, 110, 100
Azure SQL Database	12	150	150, 140, 130, 120, 110, 100
Azure SQL Database Managed Instance	12	150	150, 140, 130, 120, 110, 100
SQL Server 2016 (13.x)	13	130	130, 120, 110, 100
SQL Server 2014 (12.x)	12	120	120, 110, 100
SQL Server 2012 (11.x)	11	110	110, 100, 90
SQL Server 2008 R2	10.5	100	100, 90, 80
SQL Server 2008	10	100	100, 90, 80
SQL Server 2005 (9.x)	9	90	90, 80
SQL Server 2000 (8.x)	8	80	80

Database Scoped Configurations

- **SQL Server 2016 introduced "scoped configurations"**

```
ALTER DATABASE SCOPED CONFIGURATION
LEGACY_CARDINALITY_ESTIMATION = { ON | OFF | PRIMARY }
```

- **Creates confusion because now there's more than one place to look**
  - Compatibility mode can be set to the NEW cardinality estimation model
  - Database can still run the legacy CE using the scoped configuration
- **Compatibility mode confusion:**
  - Meant to allow time for syntax changes after upgrade (time for you to fix your code but still upgrade)
  - Optimizer fixes as of **RTM**
    - **Post-RTM fixes can be enabled with the scoped configuration option QUERY_OPTIMIZER_HOTFIXES**
  - Prior to 2014 that was it… in 2014 the compatibility model also changes to the new CE if set to 120 or higher and to get the optimizer hotfixes you'd turn on trace flag 4199
  - SQL Server 2016 adds the scoped configuration to separate these:
    - Your database compatibility mode can be in 130, 140, or 150 – so you get optimizer fixes/enhancements (not related to cardinality)
    - You can enable the "legacy CE"
    - **IMPORTANT:** If you need to downgrade your **compatibility mode** to get something to "work" (outside of CE) then the SQL team considers that to very likely be a bug!

Trace Flags vs. Query Hints

- **Trace flags are meant more for administrative use**
  - Some are "global" only and do nothing at the session level
    - See BOL under: scope "global only"
    - Using DBCC TRACEON (#, -1) sets a trace flag globally but only until next restart
    - Set as a startup option (-T #) if you want this set for each service restart
  - Check FIRST to see if there's a better way to set these AND re-check on each SP / upgrade
- **Query hints are a MUCH better way of enabling these behaviors:**
  - OPTION clause for your query
  - SELECT …
    OPTION (USE HINT ('query_hint', 'query_hint')
  - Example – instead of using trace flag 9481

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

Legacy Cardinality Estimation Model:

Cardinality Estimation, Compatibility Mode, and Query Hints

- YUCK: Service-wide trace flag on start up (set in service manager) → -T 9481
- Testing?: Temporary, but server-wide trace flag → DBCC TRACEON (9481, -1)
- Better: Session-level Testing! → DBCC TRACEON (9481)
- SQL Server 2014: any compatibility mode less than 120
- SQL Server 2016+:
  - Any compatibility mode less than 120
  - Database compatibility mode >= 120 but scoped database configuration option is on: `ALTER DATABASE SCOPED CONFIGURATION LEGACY_CARDINALITY_ESTIMATION = { ON | OFF | PRIMARY }`
- Seeing which CE you're using: (Properties Window with Showplan)
  - CardinalityEstimationModelVersion: 70 (legacy)

New Cardinality Estimation Model (1 of 2)

Cardinality Estimation, Compatibility Mode, and Query Hints

- **Yuck: Service-wide trace flag on start up → -T 2312**
- **Testing: Temporary / server-wide trace flag → DBCC TRACEON (2312, -1)**
- **Better: Session-level Testing! → DBCC TRACEON (2312)**
- **SQL Server 2014: compatibility mode 120**
- **SQL Server 2016+:**
  - Any compatibility mode greater than or equal to 120
  - Each compatibility mode has optimizer fixes AND sometimes subtle changes / fixes to the cardinality estimation model
  - If you want to use the cardinality estimation model for that database compatibility model
    - `OPTION (USE HINT ('FORCE_DEFAULT_CARDINALITY_ESTIMATION'))`
    - Overrides using the legacy cardinality estimation model when set through the database scoped configuration

- ***Continued on next slide***

New Cardinality Estimation Model (2 of 2)

Cardinality Estimation, Compatibility Mode, and Query Hints

- If you want to get ONLY the RTM optimizer fixes for that version
  - `OPTION (USE HINT ('QUERY_OPTIMIZER_COMPATIBILITY_LEVEL_n'))`
  - **Does NOT affect the cardinality estimation model** when set through the database scoped configuration
- If you want to get the optimizer hotfixes post-RTM
  - `OPTION (USE HINT ('ENABLE_QUERY_OPTIMIZER_HOTFIXES'))`
  - **Does NOT affect the cardinality estimation model** when set through the database scoped configuration
  - Equivalent to trace flag 4199 (for more information see: **DBCC TRACEON - Trace Flags (Transact-SQL)**

- **Seeing which CE you're using: (Properties Window with Showplan)**
  - CardinalityEstimationModelVersion: #
    - 120 = SQL Server 2014
    - 130 = SQL Server 2016
    - 140 = SQL Server 2017
    - 150 = SQL Server 2019

Migrations / Upgrades / Regressions

- **Option 1: Least surprise**
  - Upgrade existing databases (through backup / restore)
  - Leave their existing compatibility level intact
    - Restoring / attaching does not "upgrade" the compatibility level
  - When troubleshooting, test trace flag 2312 against queries whose estimates are inaccurate (see if they benefit from the new CE model)
    - If so, add the OPTION hint in the specific query that benefits
- **Option 2: Better / safer process + with ideal testing (2016 and higher)**
  - Change to that version's compatibility level 130+ (for optimizer fixes)
  - Set the legacy CE using the scoped configuration option
  - TEST (using the query hints to see where beneficial)
  - Change to the new CE by turning the database scoped configuration off `LEGACY_CARDINALITY_ESTIMATION = OFF`
  - TEST (possibly using the legacy CE if you find a query that needs it)
  - Add the optimizer fixes post RTM with the database scoped configuration option
  - TEST

Review

- **Selectivity and estimates**
- **Query complexity**
- **Estimates from statistics**
  - Sampling
  - The histogram
  - Filtered statistics
  - Uneven distribution
- **Migrations / Upgrades / Regressions**
- **Appendix: Changes to Cardinality Estimation (CE) in SQL 2014**

Resources

- **Whitepaper: Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator by Joe Sack http://bit.ly/1mFDB2t**
- **Books Online: ALTER DATABASE (Transact-SQL) Compatibility Level**
- **Books Online: DBCC TRACEON - Trace Flags (Transact-SQL)**
- **Books Online: Hints (Transact-SQL) – Query**
- **Automating the analysis skew (from histograms)**
  - Online PASS Recording: https://www.pass.org/
    - Choose "**All Recordings**" from the "**Learn**" drop-down
    - Then, enter a filter (left side) of Tripp (or, "skewed data")
    - Watch my PASS Summit 2013 presentation:
      Skewed Data, Poor Cardinality Estimates, and Plans Gone Wrong

Questions!

Thank you!

We hope to see you at another Immersion Event!

SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Appendix: Changes to Cardinality Estimation (CE) in SQL 2014

Kimberly L. Tripp

Kimberly@SQLskills.com

Changes to Cardinality Estimation (CE) in SQL 2014

- CE model assumptions
- Correlation for multiple predicates – legacy CE model
- Correlation for multiple predicates – new CE model
- Minimum Estimate (Eliminate CE Calculation)
- Modified out-of-range value estimation (both models)
- Join estimate algorithm changes
- Simple containment
- Base containment
- Estimates from Join-Containment Queries (both models)
- Cardinality Estimation Models

CE Model Assumptions

- **Independence**
  - Filters are uncorrelated in absence of statistics indicating otherwise
- **Uniformity**
  - Values in a histogram step are evenly distributed (spread) and have the same frequency
- **Inclusion**
  - When using a column-equal-constant predicate, it is assumed the value actually exists
- **Containment**
  - When estimating an equality join, it is assumed that there is a maximum overlap of distinct values (think “PK-to-FK” relationship”)

Correlation for Multiple Predicates
(Legacy CE model)

- "Independence" assumption at work…
- Selectivity of conjunctive predicates are computed as the multiplication of individual selectivities
- Result: calculation expects even distribution across the predicates

$$p0 * p1 * p2 * p4$$

Correlation for Multiple Predicates
(New CE model)

- Predicates are sorted by selectivity, keeping only the four most selective predicates for use in the calculation
- Successive predicate is then "softened" = The calculator name references "ExponentialBackoff" – which is defined as "*an algorithm that uses feedback to multiplicatively decrease the rate of some process, in order to gradually find an acceptable rate.*"

$$\text{Rows} * p0 * \mathbf{POWER}(p1, 0.5) * \mathbf{POWER}(p2, 0.25) * \mathbf{POWER}(p3, 0.125)$$

- $p0$ = most-selective predicate
- $p1$ = second most-selective predicate (back off by $^1/_2$)
- $p2$ = third most-selective predicate (back off by $^1/_4$)
- $p3$ = forth most-selective predicate (back off by $^1/_8$)

Side-by-side CE Example w/Parameters & Variables

```
SELECT …
WHERE c1 = @v1 AND c2 = @p1
```

- Selectivity of c1 based on the density_vector (average number of rows for a given c1 value) is 25% or 1/4
- Selectivity of c2 based on the histogram (parameter sniffing) is 1/10

- Legacy CE
  - 1/4 * 1/10 = 1/40
- New CE
  - Most selective (1/10) multiplied by the next most selective with the fraction squared
  - 1/10 * 1/16 = 1/160

Minimum Estimate (Eliminate CE Calculation)

- **Trace Flag 4137 introduced in SQL Server 2008 and available through SQL Server 2012**
  - See KB: 2658214
    FIX: Poor performance when you run a query that contains correlated AND predicates in SQL Server 2008 or in SQL Server 2008 R2 or in SQL Server 2012
  - Not really a "fix" but an alternative to how estimates are handled with multiple **conjunctive** predicates (ANDs not ORs)
  - Instead of letting the CE do the estimation, this TF causes SQL Server to use the minimum number (from the two tables) as the estimate (instead of the CE model's calculation)
- **In SQL Server 2014, trace flag 4137 cannot be used unless you're in a compatibility mode less than 120**
- **In SQL Server 2014, if you want to use the minimum estimate (for conjunctive predicates) you can use trace flag 9471**
- **Better to use the hint (trace flags more for FYI / searching code!): 'ASSUME_MIN_SELECTIVITY_FOR_FILTER_ESTIMATES'**

Table-valued Parameters and Poor Estimates

- **When a TVP's rowset is estimated (prior to execution), the data set is empty**
  - There is absolutely no way to know how many rows will match (there's no statistics, there's no data, there's just nothing that can be used)
  - SQL Server uses heuristics to estimate rows
    - Legacy CE estimates 1 row
    - New CE estimates 100 rows
- **Inside of your code, some people (traditionally) have used OPTION (RECOMPILE) to have the statements that use TVPs to get their estimates updated at runtime**
  - Pro: better estimates -> better plans
  - Con: additional CPU to recompile
- **New option (2012SP2+ / 2014CU3+): trace flag 2453 (eliminates recompilation if current value similar to last value)**
  - **New Trace Flag to Fix Table Variable Performance**

Modified Out-of-Range Value Estimation

- **Legacy CE Model**
  - "Ascending key problem" - query predicates reference newly inserted data that falls out of the range of a statistic object histogram
  - Can result in under-estimates
  - Trace flags 2389 and 2390 to enable automatic generation of statistics for ascending keys
- **New CE Model**
  - Assumption that histogram out-of-range rows DO exist
  - Uses the density vector to get the average frequency (calculated by multiplying the number of rows by the all density)
  - Trace flag 4139 is an option in the New CE
  - **Better to use the hint (trace flags more for FYI / searching code!): 'ENABLE_HIST_AMENDMENT_FOR_ASC_KEYS'**

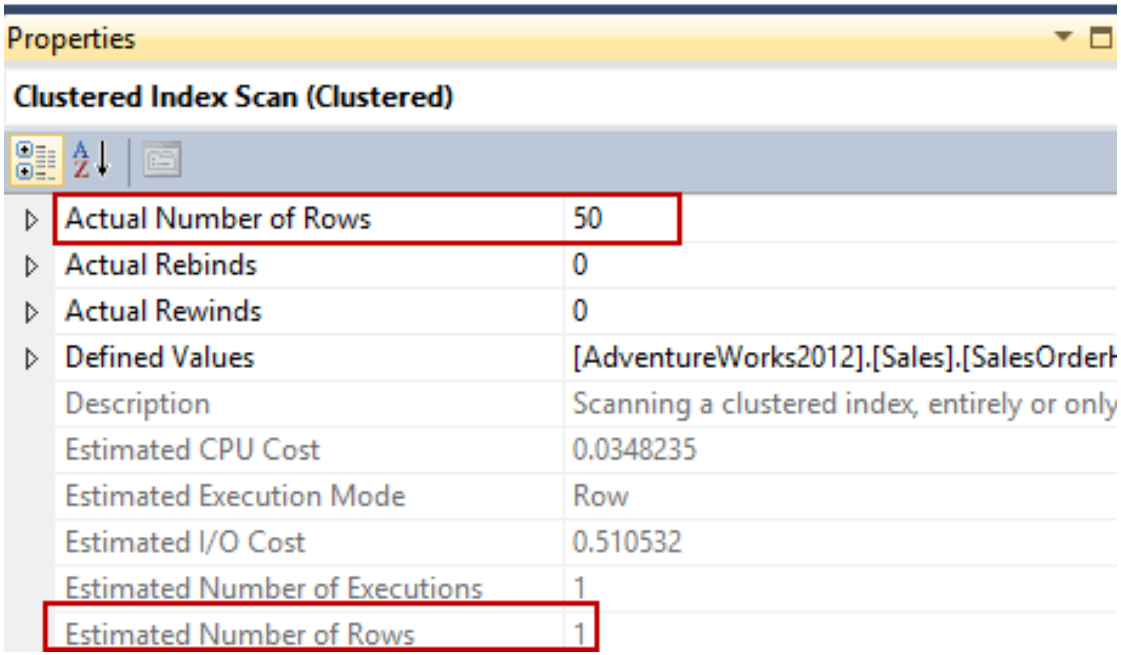
Modified Out-of-Range Value Estimation
(Legacy CE model)

```
DBCC SHOW_STATISTICS('Sales.SalesOrderHeader',
                     _WA_Sys_00000003_4B7734FF);
```

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
182	2008-06-07 00:00:00.000	87	82	1	87
183	2008-06-09 00:00:00.000	74	68	1	74
184	2008-06-11 00:00:00.000	59	86	1	59
185	2008-06-14 00:00:00.000	123	102	2	61.5
186	2008-06-16 00:00:00.000	81	65	1	81
187	2008-06-18 00:00:00.000	86	72	1	86
188	2008-06-20 00:00:00.000	82	75	1	82
189	2008-06-22 00:00:00.000	91	76	1	91
190	2008-06-24 00:00:00.000	62	74	1	62
191	2008-06-26 00:00:00.000	72	79	1	72
192	2008-06-28 00:00:00.000	73	89	1	73
193	2008-07-01 00:00:00.000	119	37	2	59.5
194	2008-07-08 00:00:00.000	180	51	6	30
195	2008-07-13 00:00:00.000	117	19	4	29.25
196	2008-07-17 00:00:	99	39	3	33
197	2008-07-24 00:	183	40	6	30.5
198	2008-07-30 00:00:	148	23	5	29.6
199	2008-07-31 00:00:00.000	0	40	0	1

Modified Out-of-Range Value Estimation
(Legacy CE model)

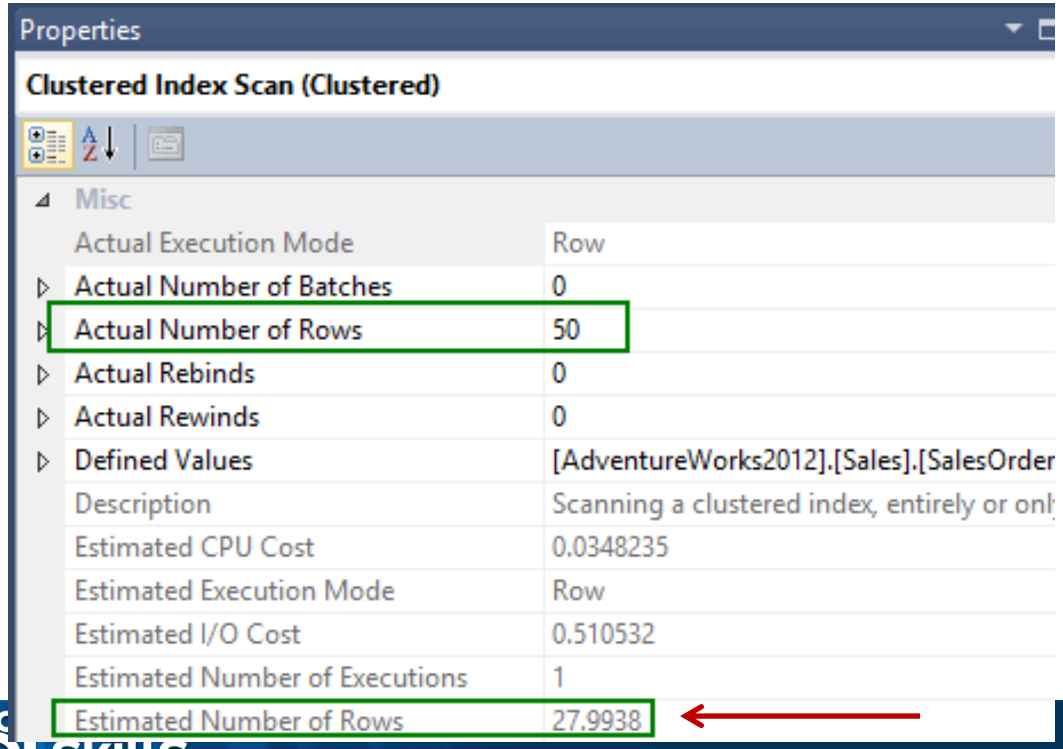
```
SELECT [SalesOrderID], [OrderDate]
FROM [Sales].[SalesOrderHeader]
WHERE [OrderDate] = '2014-02-02 00:00:00.000'
OPTION (QUERYTRACEON 9481);  -- CardinalityEstimationModelVersion 70
```



Modified Out-of-Range Value Estimation
(New CE model)

Name	Updated	Rows	Rows Sampled	Steps
_WA_Sys_00000003_4B7734FF	Mar 15 2014 10:53AM	31465	31465	199

All density	Average Length	Columns
0.0008896797	8	OrderDate



Rows * All density
= 31465 * .0008896797
= 27.9938

Join Estimates

- **Simplified join estimation algorithms**
  - Various changes to how joins are estimated – for example –aligning histograms on minimum and maximum boundaries instead of step-by-step alignment
- **Legacy CE model: "simple containment" assumption**
  - In the presence of a non-join filter predicate against a join table, some correlation is assumed
- **New CE model: "base containment"**
  - Filter predicates on separate tables are NOT assumed to be correlated with each other
  - **If you're in the New CE and want to use simple containment: 'ASSUME_JOIN_PREDICATE_DEPENDS_ON_FILTERS'**

Understanding Join-Containment

```
SELECT   [od].[SalesOrderID], [od].[SalesOrderDetailID]
FROM     [Sales].[SalesOrderDetail] AS [od]
INNER JOIN [Production].[Product] AS [p]
     ON [od].[ProductID] = [p].[ProductID]
WHERE    [p].[Color] = 'Red'
         AND [od].[ModifiedDate] = '2008-06-29 00:00:00.000';
```

Are red products correlated with orders placed on 6/29/2008?

"Simple Containment" Assumptions
(Legacy CE model)

Histogram of the Product table's [ProductID] column is loaded

Histogram is scaled down by applying the selectivity of the [p].[Color] = 'Red' filter predicate

Histogram of the SalesOrderDetail's [ProductID] column is loaded

Histogram is scaled down by applying the selectivity of the [od].[ModifiedDate] = '2008-06-29 00:00:00.000' filter predicate

The two histograms are merged together, assuming containment

"Base Containment" Assumptions
(New CE model)

Histogram of the Product table's [ProductID] column is loaded without scaling down via filter predicates

Histogram of the SalesOrderDetail's [ProductID] column is loaded without scaling down via filter predicates

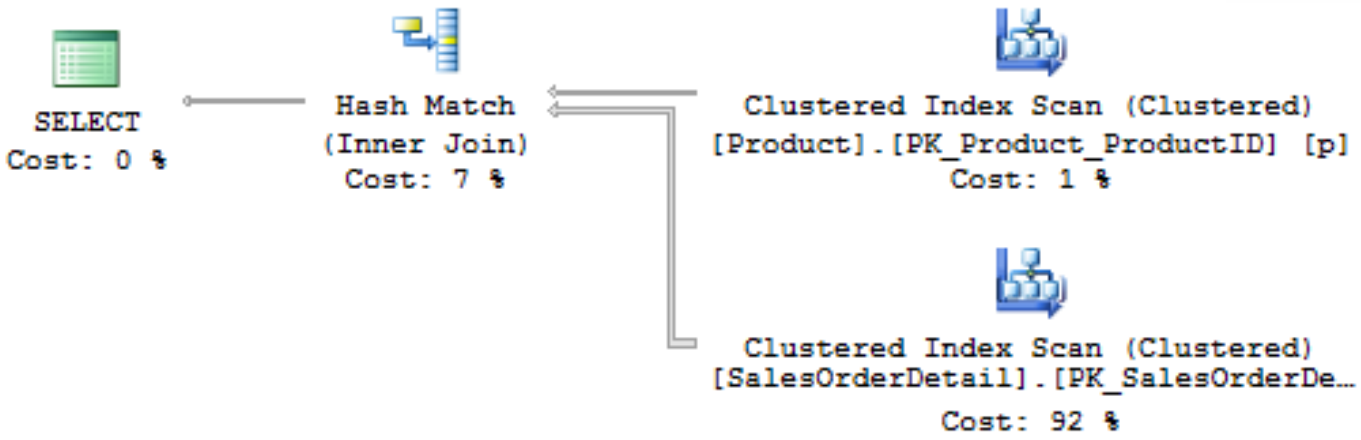
Join selectivity is computed with the two histograms, assuming containment

CE computes selectivity of the filter predicates on [p].[Color] and [od].[ModifiedDate] respectively

Join selectivity, [p].[Color] selectivity and [od].[ModifiedDate] selectivities are multiplied in order to calculate the final result

Estimates from Join-Containment Queries

```
SELECT [od].[SalesOrderID], [od].[SalesOrderDetailID]
FROM    [Sales].[SalesOrderDetail] AS [od]
INNER JOIN [Production].[Product] AS [p]
ON [od].[ProductID] = [p].[ProductID]
WHERE   [p].[Color] = 'Red' AND
[od].[ModifiedDate] = '2008-06-29 00:00:00.000';
```



Legacy CE	New CE
51.5437 rows	24.5913