Climb up towards Sql Server Statistics

by Alessandro Mortola

Sponsors


UNIPORDENONE
moltiplica il valore della formazione


Règolo
FARM


DATA
SATURDAYS

About me

alessandro.mortola@gmail.com

www.linkedin.com/in/alessandromortola

github.com/AlexMortola

- I have worked with all versions from Sql Server 7.0 to Sql Server 2025
- I spoke several times at Sql Saturday, Data Saturday and SqlStart! editions
- I am an author for sqlservercentral.com
- I am certified Sql Server 2016 Dev
- I'm currently working at Zucchetti as
  - Sql Server DBA (and PostgreSQL is coming…)
  - Analyst

Agenda

- Why Statistics?
- Structure and guessing! dbcc show_statistics
- Estimation of rows – particular cases
- Settings and syntax clauses
- Fullscan or Sampling rate? The process of statistics creation
- T-SQL and DMVs
- Takeaways

DATA SATURDAYS

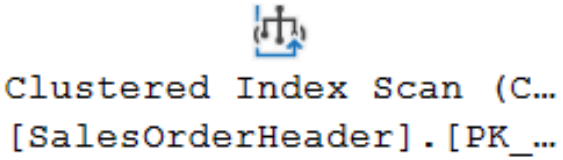
Why does SqlServer need Statistics? What if we were the Query Optimizer?

```
select oh.RevisionNumber, od.ProductID
from Sales.SalesOrderHeader oh
inner join Sales.SalesOrderDetail od on oh.SalesOrderID = od.SalesOrderID
where oh.CustomerID = 29844
order by od.ProductID;
```

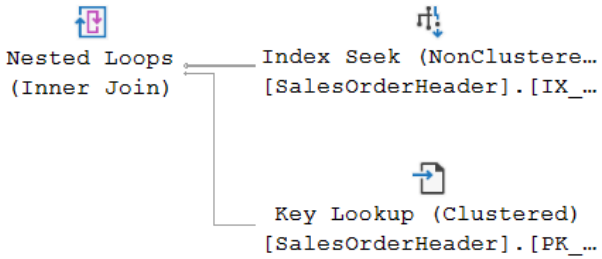
Main Questions - #1

Which is the best data access for each table?

Clustered Index Scan?



Index Seek + Key lookup (if any)?



DATA SATURDAYS

What if we were the Query Optimizer?

```
select oh.RevisionNumber, od.ProductID
from Sales.SalesOrderHeader oh
inner join Sales.SalesOrderDetail od on oh.SalesOrderID = od.SalesOrderID
where oh.CustomerID = 29844
order by od.ProductID;
```

Main Questions - #2

Which is the best way to join Sales.SalesOrderHeader with Sales.SalesOrderDetail?

Nested Loop

Hash Match

Merge Join

DATA SATURDAYS

What if we were the Query Optimizer?

```
select oh.RevisionNumber, od.ProductID
from Sales.SalesOrderHeader oh
inner join Sales.SalesOrderDetail od on oh.SalesOrderID = od.SalesOrderID
where oh.CustomerID = 29844
order by od.ProductID;
```

Main Questions - #3

How much granted memory do I need to sort?

Sort

DATA SATURDAYS

Statistics – A crucial piece of information

Statistics are one of the most fundamental pieces of information needed to build execution plans, as the **Query Optimizer** relies on the **Cardinality Estimator** that relies primarily on **Statistics** to "guess" the number of rows that flow from one operator to another

The option of automatically creating and updating statistics should work out fine in almost all cases *[Holger Schmeling]*

A statistical method reduces the complete data set to a compact structure while still retaining some ability to estimate distribution

It is not exact and if it were, it would not be statistics [Joe Chang]

Of course, they must be UP TO DATE!!!

DATA SATURDAYS

How can statistics be generated?

Index related

When an index is created, a statistic with the same name is created with 100% of data if both `RESUMABLE = ON` and partitioned objects are NOT used

Automatic

Automatic statistics are created by the Query Optimized when needed. They are single-column and their name begins with "_WA_Sys_"

Created by user

You can create statistics not related to indexes using CREATE STATISTICS syntax

Structure and guessing!
dbcc show_statistics

DATA SATURDAYS

DBCC SHOW_STATISTICS

DBCC SHOW_STATISTICS (table_name , target) [WITH { STAT_HEADER | DENSITY_VECTOR | HISTOGRAM } [, ...n]] [;]

Example

```
dbcc show_statistics ('Sales.SalesOrderHeaderEnlarged',
                                    'st_SalesOrderHeaderEnlarged_01') with stat_header;
```

The information related to Statistics is returned as three result sets:

- Header
- Density Vector
- Histogram

Note

Internally, Statistics are stored as Binary Large Objects (BLOBs)

DATA SATURDAYS

DBCC SHOW_STATISTICS Header

```
dbcc show_statistics ('Sales.SalesOrderHeaderEnlarged',
                      'st_SalesOrderHeaderEnlarged_01') with stat_header;
```

Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	st_SalesOrderHeaderEnlarged_01	Jan 18 2025 3:54PM	1290065	76912	199	0.01741718	12	NO	NULL	1290065	0

Total number of rows when the statistics were last updated

Total number of rows sampled for statistics calculations

Number of steps in the Histogram

If 'Yes', stats object contains summary stats separately

Total number of rows in the table before applying the filter expression

Percent used for stats updates that don't specify a sampling percentage

DATA SATURDAYS

DBCC SHOW_STATISTICS
Density Vector

```
dbcc show_statistics ('Sales.SalesOrderHeaderEnlarged',
                      'st_SalesOrderHeaderEnlarged_01') with density_vector;
```

Results | Messages

	All density	Average Length	Columns
1	0.0008598453	8	OrderDate
2	0.0001193887	12	OrderDate, TerritoryID

1 / number of distinct values
Results display density for each “prefix of columns”

DATA SATURDAYS

DBCC SHOW_STATISTICS Histogram

```
dbcc show_statistics ('Sales.SalesOrderHeaderEnlarged',
                      'st_SalesOrderHeaderEnlarged_01') with histogram;
```

Results | Messages

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	2011-06-20 00:00:00.000	0	601.6175	0	1
2	2011-10-31 00:00:00.000	4081.853	735.3102	22	184.8711
3	2012-02-04 00:00:00.000	4266.628	952.561	17	250.0635
4	2012-04-06 00:00:00.000	3645.111	1203.235	23	157.9139
5	2012-06-03 00:00:00.000	2200.505	1069.542	6	365.2788

RANGE_ROWS / DISTINCT_RANGE_ROWS

Upper bound column value for a histogram step. The column value is also called a key value

Estimated number of rows, the column value of which falls within a histogram step

Estimated number of rows, the column value of which equals the upper bound of the histogram step

Estimated number of rows with a distinct column value within a histogram step

Histogram – The creation

Sql Server creates the histogram from the sorted set of column values in three steps.

1. **Histogram initialization**

   A sorted set of distinct values is processed, up to 200 values

   Only RANGE_HIGH_KEY and EQUAL_ROWS values are collected

2. **Scan with bucket merge**

   Each successive value, kept in a sorted order, is either added to the latest range or a new range is created; in this case one pair of existing and neighbouring ranges, selected to minimize information loss, should merge. The number of steps stays at 200 throughout this step

3. **Histogram consolidation**

   More ranges can merge if a significant amount of information is not lost

The lowest value is the first histogram step (except for the artificial NULL step if present) and the highest value is the last histogram step

Particular cases

DATA SATURDAYS

What happens when…
We filter for a value outside the histogram?

```
select SalesOrderId, OrderDate
from Sales.SalesOrderHeaderEnlarged
where OrderDate = '20280315';
```

last_updated	rows	rows_sampled	unfiltered_rows	modification_counter
2026-01-10 18:14:52.1133333	1290065	68600	1290065	30

All density	Average Length	Columns
0.0008810573	8	OrderDate

step_number	range_high_key	range_rows	equal_rows	distinct_range_rows	average_range_rows
1	2022-06-18 00:00:00.000	0	73.83115	0	1
2	2022-10-23 00:00:00.000	4011.613	843.1201	30	133.2331
3	2023-01-17 00:00:00.000	4576.628	524.608	18	253.3107
4	2023-03-04 00:00:00.000	2354.233	1292.784	7	334.9639
5	2023-05-28 00:00:00.000	5593.657	618.288	13	428.6473
…	…	…	…	…	…
195	2027-12-27 00:00:00.000	7062.698	711.968	10	703.5305
196	2028-01-21 00:00:00.000	6554.184	1311.52	9	725.3896
197	2028-02-24 00:00:00.000	7006.197	487.136	9	775.4165
198	2028-03-13 00:00:00.000	1205.367	562.08	4	300.0143
199	2028-03-14 00:00:00.000	4.023425	29.96028	0	1136.618

What the CE takes mainly into account:

- The Statistics (Header, Density Vector, Histogram)
- The modification_counter (more details soon)
- The data type of the column
- If the leading column is unique
- The leading column type (Unknown, Ascending, Stationary - TF 2388)
- Last but not least… the Compatibility Level !!!

Have a look at the XE "query_optimizer_estimate_cardinality" to examine in depth

DATA SATURDAYS

What happens when…
We filter for a value outside the histogram?

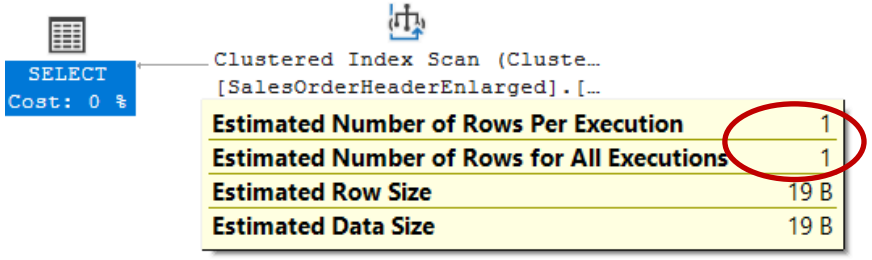
```
select SalesOrderId, OrderDate
from Sales.SalesOrderHeaderEnlarged
where OrderDate = '20280315';
```

last_updated	rows	rows_sampled	unfiltered_rows	modification_counter
2026-01-10 18:14:52.1133333	1290065	68600	1290065	30

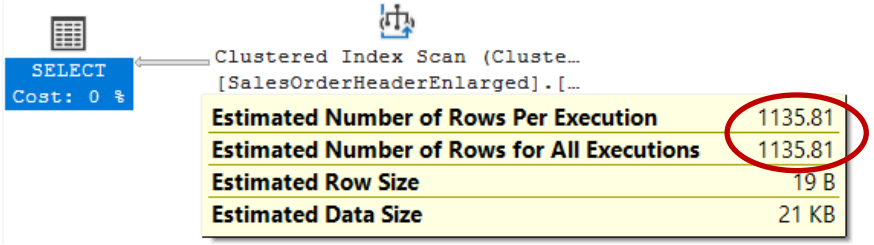
All density	Average Length	Columns
0.0008810573	8	OrderDate

step_number	range_high_key	range_rows	equal_rows	distinct_range_rows	average_range_rows
1	2022-06-18 00:00:00.000	0	73.83115	0	1
2	2022-10-23 00:00:00.000	4011.613	843.1201	30	133.2331
3	2023-01-17 00:00:00.000	4576.628	524.608	18	253.3107
4	2023-03-04 00:00:00.000	2354.233	1292.784	7	334.9639
5	2023-05-28 00:00:00.000	5593.657	618.288	13	428.6473
…	…	…	…	…	…
195	2027-12-27 00:00:00.000	7062.698	711.968	10	703.5305
196	2028-01-21 00:00:00.000	6554.184	1311.52	9	725.3896
197	2028-02-24 00:00:00.000	7006.197	487.136	9	775.4165
198	2028-03-13 00:00:00.000	1205.367	562.08	4	300.0143
199	2028-03-14 00:00:00.000	4.023425	29.96028	0	1136.618

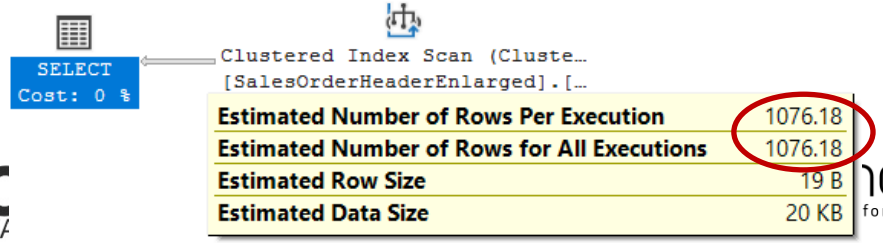
Sql Server 2012



Sql Server 2014



Sql Server 2025



DATA SATURDAYS

What happens when… Statistics are not available?

Operator	Estimate
>, >=, <, <=	30%
BETWEEN/LIKE	9% Exception: in 2014 BETWEEN with variables/parameters with sniffing disabled the estimate is 16.4317%
= with a unique col	1 row
= with a nonunique col prior to 2014	C ¾ (C = table cardinality)
= with a nonunique col in 2014	C ½

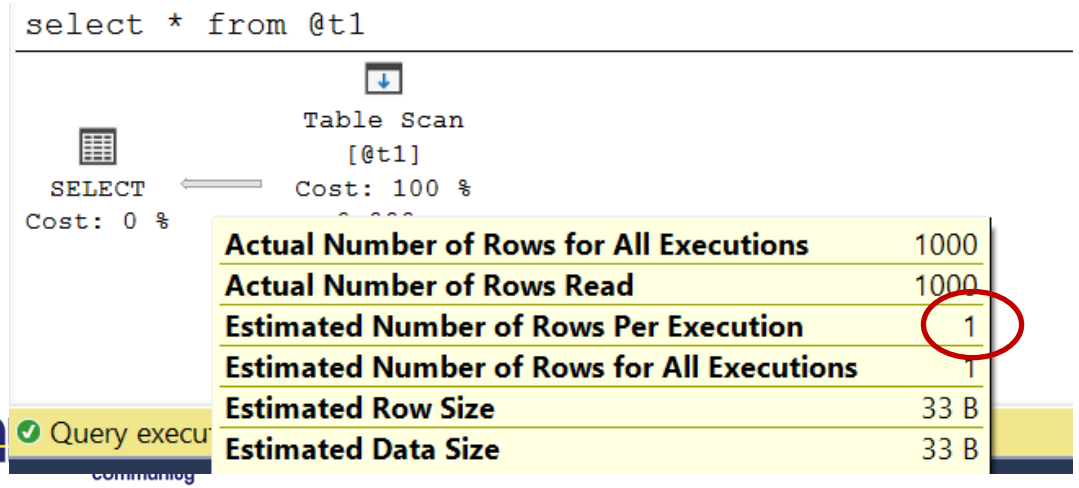
DATA SATURDAYS

What happens when… We query a table variable?

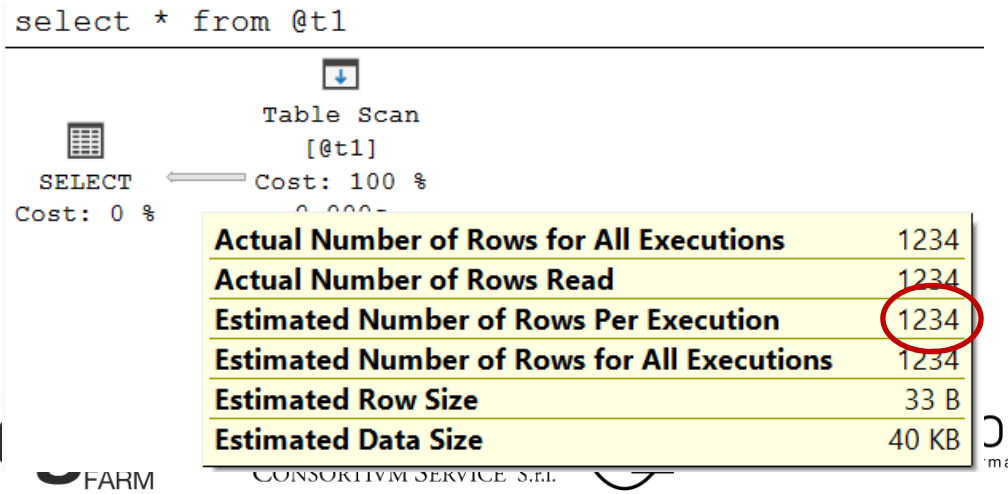
```
declare @t1 table (id int, f1 varchar(36));

--Insert some records into @t1

select * from @t1;
```

Table Variable Deferred Compilation

Before Sql Server 2019



Starting from Sql Server 2019

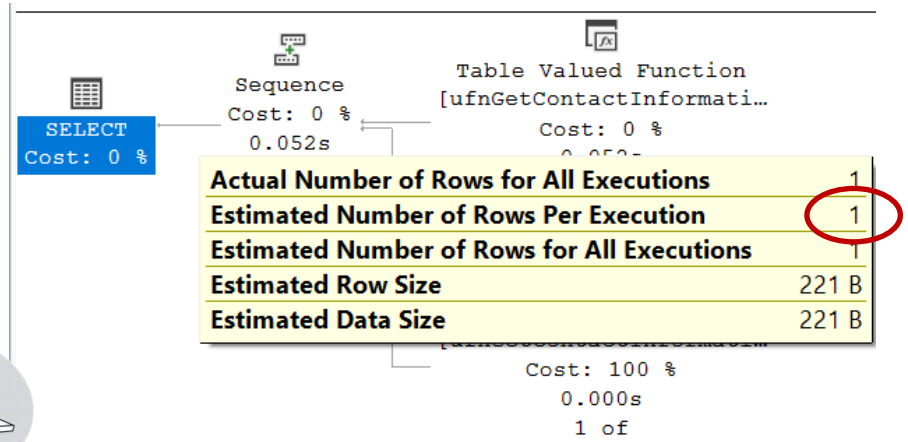


DATA SATURDAYS

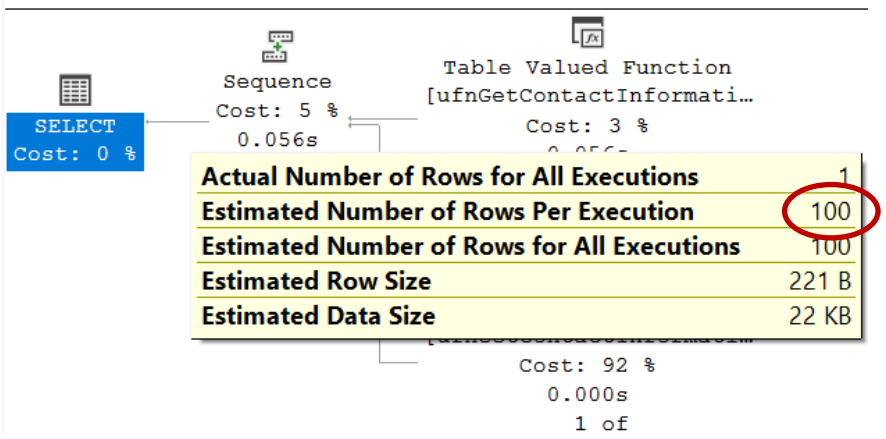
What happens when… We query a Multi-Statement Table-Valued function?

```
select *
from dbo.ufnGetContactInformation(290) ms;
```

Before Sql Server 2014



Starting from Sql Server 2014



Settings

DATA SATURDAYS

Auto Create / Auto Update (database level – The default is ON for both)

AUTO_CREATE_STATISTICS

- If it is set to ON, SQL Server will automatically create statistics on individual columns used in a predicate, not already indexed and without a histogram in an existing statistics object.
- The name of the statistics is formed by a constant part (_WA_Sys_), column_id and the object_id of the table, both of them converted to hexadecimal format.
  E.g.: _WA_Sys_00000002_2DB1C7EE

AUTO_UPDATE_STATISTICS

- Query Optimizer determines when statistics might be **out-of-date (*)** and then updates them when they are used by a query.
- If this setting is OFF, statistics are not updated and the Query Optimizer continues to use the out-of-date statistics.
- Even if it is set to ON, statistics are NOT updated when the index is reorganized
- Starting from Sql Server 2019 you can monitor how long queries wait for statistics to be “refreshed” using WAIT_ON_SYNC_STATISTICS_REFRESH wait

(*) More details later

DATA SATURDAYS

AUTO_UPDATE_STATISTICS_ASYNC (database level)

The default is OFF and in this case the Query Optimizer updates statistics synchronously.

Sync updates
When statistics are out-of-date, the Query Optimizer waits for updated statistics before compiling and executing the query.

Async updates
Queries compile with existing statistics even if they are out-of-date.
Statistics are typically updated soon after.

To set the async statistics update both AUTO_UPDATE_STATISTICS and AUTO_UPDATE_STATISTICS_ASYNC need to be set to True

Starting from Sql Server 2022, as well as in Azure SQL Database and Azure SQL Managed Instance, at database-scoped configuration, the ASYNC_STATS_UPDATE_WAIT_AT_LOW_PRIORITY option is available

DATA SATURDAYS

OUT-OF-DATE STATISTICS

The Query Optimizer determines when statistics might be out-of-date by counting the number of modifications since the last statistics update and comparing the number of row modifications to a threshold, that is based on the table cardinality.

Up to Sql Server 2014

Table type	Table cardinality	Recompilation threshold (#modifications)
Temporary	n < 6	6
Temporary	6 <= n <= 500	500
Permanent	n <= 500	500
Temporary or permanent	n > 500	~~500 + (0.20 * n)~~

Starting from Sql Server 2016 → MIN (500 + (0.20 * n), SQRT(1000. * n))

E.g.: *With a table with 1 million rows we have:* ***MIN (200500, 31622)***

DATA SATURDAYS

How to check “row modifications”?

We can check the current row modifications for each statistics using the

```
sys.dm_db_stats_properties (object_id, stats_id)
```

The “modification_counter” value is automatically incremented by 1 every update that involves the leading statistics column as well as every insert and every delete

Modification_counter doesn’t rollback !!!

DATA SATURDAYS

Incremental statistics

Applies to: Sql Server 2014 and later

AUTO_CREATE_STATISTICS ON (INCREMENTAL = ON)

It is at **database level** and the default is OFF
When ON, the statistics created are per partition statistics
When OFF, stats are combined for all partitions

CREATE STATISTICS … WITH INCREMENTAL = ON

The default is OFF
When ON, the statistics created are per partition

UPDATE STATISTICS … WITH RESAMPLE ON PARTITIONS (1, 2, 3)

It forces the partitions specified to be recomputed and merged to build the global statistics.
WITH RESAMPLE is required

DATA SATURDAYS

Syntax clauses related to statistics

AUTO_DROP

Up to Sql Server 2019, while automatic statistics do not block a schema change, because in this case they are automatically dropped, statistics created by the user, block any schema change.
Starting from Sql Server 2022, if the statistics are created with the auto_drop option, they behave as auto-created statistics

PERSIST_SAMPLE_PERCENT

When ON, the statistics retain the creation sampling percentage for subsequent updates that do not explicitly specify a sampling percentage

```
[ WITH
    [ FULLSCAN
        [ [ , ] PERSIST_SAMPLE_PERCENT = { ON | OFF } ]
      | SAMPLE number { PERCENT | ROWS }
        [ [ , ] PERSIST_SAMPLE_PERCENT = { ON | OFF } ]
```

DATA SATURDAYS

Filtered statistics

WHERE

This clause allows to create filtered statistics.
It specifies an expression for selecting a subset of rows to include when creating the statistics object.
In several situations (e.g. correlated columns), filtered statistics can help the optimizer to get a more precise number of rows estimate.

- modification_counter is only incremented if a statement updates the first field that defines the statistics even if the records involved in the updates do not meet the “where” condition that defines the statistics
- modification_counter is **not** incremented if the update modifies the fields involved in the WHERE clause
- Do not rely only on auto-update statistics for filtered objects !!!

Full scan or sampling rate?

DATA SATURDAYS

FULLSCAN or Sampling Rate? Two questions

Question #1

In which situations are the statistics computed on a Full Scan of the table and when on a Sampling Rate?

Question #2

If Sampling Rate is chosen, how many and which rows are taken to build statistics?

DATA SATURDAYS

FULLSCAN or Sampling Rate?

Index statistics are always created with FULLSCAN (*)

The automatic update decision to use either a FULLSCAN or a Sampling Rate depends on the table size:

- If the table < 8 MB then it updates the statistics with a FULLSCAN
- If the table > 8 MB then it updates the statistics with a Sampling Rate

(*) If both `RESUMABLE = ON` and partitioned objects are NOT used

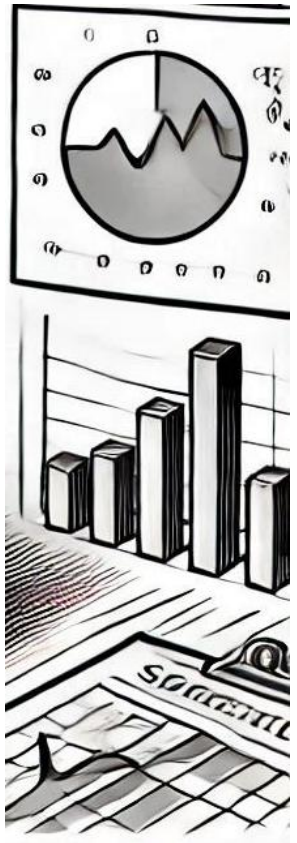
DATA SATURDAYS

Sampling Rate: How many rows? Which rows?

SQL Server uses a random number generator to decide if a page qualifies or not and all rows on selected pages contribute to the statistics

Statistics generation queries often access the base table (rather than a nonclustered index) to avoid the clustering of values that naturally occurs on nonclustered index pages

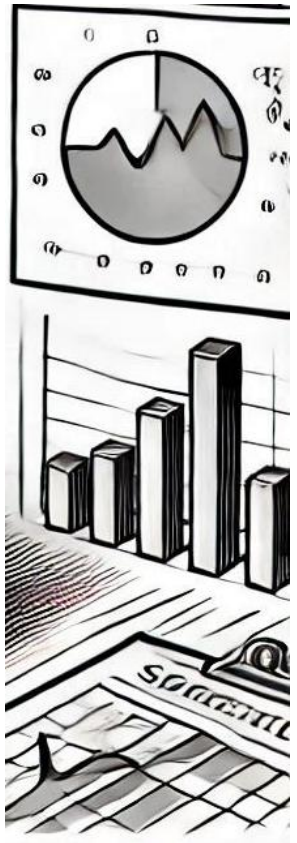
The algorithm should produce the best stats possible with as few resources as possible and as fast as possible



DATA SATURDAYS

Sampling Rate: How many rows? Which rows?

Sql Server reduces the sampling rate as the number of rows in the table is increased to make sure we are not scanning too many data

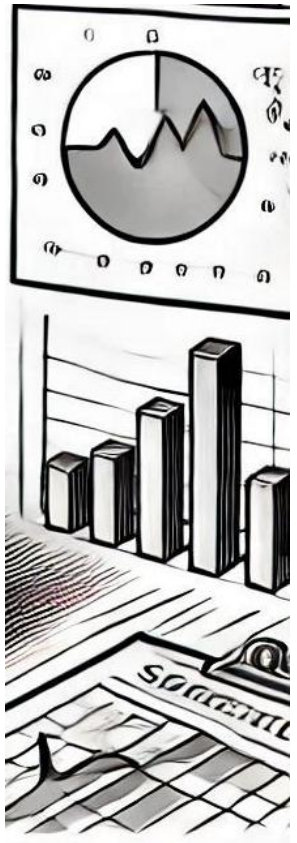


By default, the sampling rate ranges from 100% for tables smaller than 8MB to less than 1% for large tables

DATA SATURDAYS

Sampling Rate: How many rows? Which rows?

If the histogram is built using a Sample, then the values of EQ_ROWS, RANGE_ROWS, DISTINCT_RANGE_ROWS and AVERAGE_RANGE_ROWS are estimated and do not need to be integers



The tool used to select the Sampled Rows is the clause **TABLESAMPLE**

DATA SATURDAYS

TABLESAMPLE

```
SELECT StatMan([SC0], [SB0000])
FROM (
          SELECT TOP 100 PERCENT [SC0],
                    step_direction([SC0]) OVER (ORDER BY NULL) AS [SB0000]
          FROM (
                    SELECT [OrderDate] AS [SC0]
                    FROM [Sales].[SalesOrderHeaderEnlarged]
                              TABLESAMPLE SYSTEM(5.876638e+00 PERCENT) WITH (READUNCOMMITTED)
               ) AS _MS_UPDSTATS_TBL_HELPER
          ORDER BY [SC0], [SB0000]
      ) AS _MS_UPDSTATS_TBL
OPTION (MAXDOP 6);
```

We can get this internal query using:

- *SQL Server Profiler*

  "SP:StmtCompleted" EventClass

- *Extended Events*

  "sp_statement_completed" event

- The **percentage coefficient** used for TABLESAMPLE is the result of an internal computation based on the number of pages and rows of the table
- **step_direction** analyses data distribution across histogram buckets detecting jumps, dips or plateaus between adjacent buckets to determine the "direction" of data from one bucket to the next
- **StatMan** function is an internal component that should produce the best statistics possible

TABLESAMPLE - Syntax

```
TABLESMPLE SYSTEM (sample_number {ROW | PRECENT})
                                  [REPEATABLE (repeat_seed)]
```

TABLESAMPLE *extracts a random number of rows from a table, based on a percentage or number of rows*

SYSTEM

It is an option defined by the ANSI standard. In Sql Server it is the only option and for this reason it is the default

ROW

It does not represent the exact number or returned rows, but an approximation

PERCENT

It represents an approximation of the percentage of the returned rows

REPEATABLE

With this option every execution will return the same set of records

Rows sampled

The mystery of the creation of auto-statistics

```
SELECT StatMan([SC0], [SB0000])
FROM (
            SELECT TOP 100 PERCENT [SC0],
                        step_direction([SC0]) OVER (ORDER BY NULL) AS [SB0000]
            FROM (
                        SELECT [OrderDate] AS [SC0]
                        FROM [Sales].[SalesOrderHeaderEnlarged]
                                                TABLESAMPLE SYSTEM(5.876638e+00 PERCENT)
                                                                    WITH (READUNCOMMITTED)
                  ) AS _MS_UPDSTATS_TBL_HELPER

            ORDER BY [SC0], [SB0000]

      ) AS _MS_UPDSTATS_TBL

OPTION (MAXDOP 6);
```

No REPEATABLE clause

Different set of records each time

Different statistics each time?

NO! The generated statistics are always the same because for sampled UPDATE STATISTICS, the REPEATABLE value is 1

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
2011-05-31 00:00:00.000	0	43	0	1
2011-12-21 00:00:00.000	4084	118	203	20.11823
2012-05-21 00:00:00.000	7931	194	151	52.52318
2012-07-07 00:00:00.000	4626	219	46	100.5652
2012-08-07 00:00:00.000	3520	235	30	117.3333
2012-09-19 00:00:00.000	5508	301	42	131.1429
2012-10-20 00:00:00.000	4546	293	30	151.5333
2012-11-21 00:00:00.000	5426	252	31	175.0323
2013-01-07 00:00:00.000	7847	327	46	170.587

Locks

Locks

During the process of creation and update of the statistics the main lock types used are the following:

Mode	Resource Type	Note
SCH_S - Schema stability	OBJECT (Table) METADATA	It is compatible with all lock modes, except the schema modification (SCH-M) lock mode
X - Exclusive	OBJECT (Table)	It lasts for few micro sec. It does not interfere with any IX lock due to sessions that have open transaction with X lock at key level because it is set on **UPDSTATS resource_subtype**
SCH_M – Schema modification	METADATA	

Locks

During the process of drop of the statistics the main lock types used are the following:

Mode	Resource Type	Note
SCH_M - Schema modification	OBJECT (Table)	If there are open transactions with row modifications the drop is locked
SCH_S	METADATA	

T-SQL and DMVs

DATA SATURDAYS

CREATE STATISTICS and UPDATE STATISTICS

It is possible to create or update statistics using the T-SQL statements CREATE STATISTICS and UPDATE STATISTICS with which we can set all the clauses we have seen so far.

```
CREATE STATISTICS statistics_name
ON { table_or_indexed_view_name } ( column [ , ...n ] )
        [ WHERE <filter_predicate> ]
        [ WITH
        [ FULLSCAN
          [ [ , ] PERSIST_SAMPLE_PERCENT = { ON | OFF } ]
          | SAMPLE number { PERCENT | ROWS }
          [ [ , ] PERSIST_SAMPLE_PERCENT = { ON | OFF } ]
          | <update_stats_stream_option> [ , ...n ]
          [ [ , ] NORECOMPUTE ]
          [ [ , ] INCREMENTAL = { ON | OFF } ]
          [ [ , ] MAXDOP = max_degree_of_parallelism ]
          [ [ , ] AUTO_DROP = { ON | OFF } ] ] ];
```

```
UPDATE STATISTICS table_or_indexed_view_name
    [ { { index_or_statistics__name } | ( { index_or_statistics_name } [ , ...n ] ) } ]
        [ WITH
        [ FULLSCAN     [ [ , ] PERSIST_SAMPLE_PERCENT = { ON | OFF } ]
        | SAMPLE number { PERCENT | ROWS }
                [ [ , ] PERSIST_SAMPLE_PERCENT = { ON | OFF } ]
        | RESAMPLE [ ON PARTITIONS ( { <partition_number> | <range> } [ , ...n ] ) ]
                | <update_stats_stream_option> [ , ...n ] ]
        [ [ , ] [ ALL | COLUMNS | INDEX ]
        [ [ , ] NORECOMPUTE ]
        [ [ , ] INCREMENTAL = { ON | OFF } ]
        [ [ , ] MAXDOP = max_degree_of_parallelism ]
        [ [ , ] AUTO_DROP = { ON | OFF } ] ] ;
```

DATA SATURDAYS

DMVs #1

sys.stats contains a row for each statistics object that exists for the tables, indexes, and indexed views in the database

	name	stats_id	auto_created	user_created	has_filter	filter_definition	is_incremental	has_persisted_sample	auto_drop
1	PK_SalesOrderHeaderEnlarged_SalesOrderID	1	0	0	0	NULL	0	0	0
2	AK_SalesOrderHeaderEnlarged_rowguid	2	0	0	0	NULL	0	0	0
3	AK_SalesOrderHeaderEnlarged_SalesOrderNumber	3	0	0	0	NULL	0	0	0
4	IX_SalesOrderHeaderEnlarged_CustomerID	4	0	0	0	NULL	0	0	0
5	IX_SalesOrderHeaderEnlarged_SalesPersonID	5	0	0	0	NULL	0	0	0
6	_WA_Sys_00000003_4AD81681	6	1	0	0	NULL	0	0	1
7	_WA_Sys_0000000D_4AD81681	7	1	0	0	NULL	0	0	1
8	_WA_Sys_00000017_4AD81681	8	1	0	0	NULL	0	0	1
9	_WA_Sys_00000006_4AD81681	9	1	0	0	NULL	0	0	1
10	_WA_Sys_00000002_4AD81681	10	1	0	0	NULL	0	0	1
11	_WA_Sys_00000007_4AD81681	11	1	0	0	NULL	0	0	1
12	_WA_Sys_00000015_4AD81681	12	1	0	0	NULL	0	0	1

sys.stats_columns provides statistics information for each column and should be joined with the previous one on object_id and stats_id

DATA SATURDAYS

DMVs #2

sys.dm_db_stats_properties returns properties of statistics for the specified database object (table or indexed view) in the current database

	stats_id	last_updated	rows	rows_sampled	steps	unfiltered_rows	modification_counter	persisted_sample_percent
1	6	2024-11-17 15:00:40.3000000	1290065	70054	199	1290065	0	0

STATS_DATE returns the date of the most recent update for statistics on a table or indexed view

sys.dm_db_stats_histogram returns the statistics histogram for the specified database object (table or indexed view) in the current database.
Similar to DBCC SHOW_STATISTICS WITH HISTOGRAM.

	step_number	range_high_key	range_rows	equal_rows	distinct_range_rows	average_range_rows
1	1	2011-06-07 00:00:00.000	0	220.1672	0	1
2	2	2011-12-08 00:00:00.000	2861.277	550.418	18	158.9598
3	3	2012-03-30 00:00:00.000	4301.145	348.5981	14	307.2246
4	4	2012-08-22 00:00:00.000	7143.961	256.8617	23	310.607
5	5	2012-09-11 00:00:00.000	2381.321	1541.17	7	340.1887
6	6	2012-10-25 00:00:00.000	7199.341	1687.948	16	449.9588
7	7	2012-12-20 00:00:00.000	6608.626	2458.533	13	508.3558
8	8	2012-12-22 00:00:00.000	0	917.3633	0	1

DATA SATURDAYS

DMVs #3

sys.dm_db_incremental_stats_properties returns properties of incremental statistics for the specified database object (table) in the current database. It is similar to sys.dm_db_stats_properties which is used for non-incremental statistics.

	stats_id	partition_number	last_updated	rows	rows_sampled	steps	unfiltered_rows	modification_counter
1	1	1	NULL	NULL	NULL	NULL	NULL	NULL
2	1	2	2024-12-13 06:51:00.3966667	1607	1607	200	1607	0
3	1	3	2024-12-13 06:51:00.4000000	3915	3915	200	3915	100
4	1	4	2024-12-13 06:51:00.4000000	14182	14182	200	14182	0
5	1	5	2024-12-13 06:51:00.4033333	11761	11761	181	11761	0

Stored Procedures

sp_autostats displays or changes the automatic statistics update option, AUTO_UPDATE_STATISTICS, for an index, a statistics object, a table, or an indexed view

```
exec sp_autostats @tblname = N'dbo.Posts';
```

	Index Name	AUTOSTATS	Last Updated
1	[PK_Posts__Id]	ON	2018-09-11 11:19:08.023
2	[_WA_Sys_00000006_060DEAE8]	ON	2026-01-09 07:21:17.427

The syntax: `sp_autostats <table_name> [, {ON|OFF} [, <index_name>] ]`

sp_updatestats runs UPDATE STATISTICS against all user-defined and internal tables in the current database.

The syntax: `sp_updatestats [ [ @resample = ] 'resample' ]`

sp_helpstats returns statistics information about columns and indexes on the specified table

The syntax: `sp_helpstats [ @objname = ] N'objname' [ , [ @results = ] N'results' ][ ; ]`

`@results: ALL` lists statistics for all indexes and also columns that have statistics created on them

STATS only lists statistics not associated with an index

What about Memory-optimized tables and Columnstore Indexes?

Memory-optimized tables

- Statistics are always created with FULLSCAN
- For natively compiled stored procedures, execution plans for queries in the procedure are optimized when the procedure is compiled
- Natively compiled stored procedures can be manually recompiled using sp_recompile

Columnstore Indexes

- Columnar data does not work the same as B-Tree
- In CSI, data is stored in segments, which are compressed chunks of data. SQL Server maintains some internal metadata (like min and max values)
- sp_updatestats skips columnstore indexes

```
(indexproperty(stat.object_id, name, 'iscolumnstore') = 0)
```

- SQL Server still creates automatic statistics for columnstore indexes and the QO relies also on them to build the plan; the segment elimination remains fundamental
- Be careful with Clustered Columnstore Indexes. The modification_counter is not properly updated; in this case it is preferred to update the statistics manually.

Takeaways

DATA SATURDAYS

Takeaways

Statistics work out fine in almost all cases

Keep them up-to-date !!!

Updating statistics has a cost (CPU, IOs..) and may invalidate cached plans.

Scheduling a (weekly) update process is a good practice, but not a “one-size-fits-all”

DATA SATURDAYS

Takeaways

When an index is created or rebuilt, by default, statistics are based on the FULLSCAN; as soon as statistics are auto-updated, they are based on a Sampling Rate

The statement REORGANIZE of an index does not touch related statistics

DATA SATURDAYS

Takeaways

You can tune queries with:

- Multi-column statistics
- Statistics on computed columns
- Filtered statistics (but handle with care!!!)
- SAMPLE PERCENT setting

Bonus

The strange case of CXPACKET with MAXDOP 1

Resources

DATA SATURDAYS

Resources

Fabiano Amorim - 13 optimizer statistic problems you didn't know you had
https://www.youtube.com/watch?v=SFIg9JFmDHk

Analyze and tune Sql Server statistics - Redgate
https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/analyze-and-tune-sql-server-statistics/

Paul White - Cardinality Estimation: Combining Density Statistics
https://sqlperformance.com/2017/08/sql-optimizer/combining-density

Itzik Ben-Gan - Seek and You Shall Scan Part II: Ascending Keys
https://www.itprotoday.com/sql-server/seek-and-you-shall-scan-part-ii-ascending-keys

Dave Ballantyne - Extended Events - inaccurate_cardinality_estimate
(It is in the "debug" channel!)
https://www.sqlservercentral.com/blogs/extended-events-inaccurate_cardinality_estimate

Are there any questions?

Thank you!