Introduction to Statistics in SQL Server

Andy Warren
www.sqlandy.com
@sqlandy
www.linkedin.com/in/sqlandy
Why Do We Need Statistics?

We can’t build a good plan to get the rows we need without having an idea of how many rows we’re going to get!
A Tale of Two Queries

**Question:** Will the two queries below use a similar query plan?

A) select * from person.Contact where LastName like 'S%'
B) select * from person.Contact where LastName like 'SM%'

Not enough info you say? What if I told you that:

Query A returns 2694 rows
Query B return 669 rows
LastName Like S% = Scan

For Query A, we see that SQL has decided to do a table scan – brute force!
LastName Like Sn% = Lookup

For Query B, we see that SQL has decided to take a lighter weight approach – a bookmark lookup.
Why Are the Plans Different?

SQL has a method that we can use to get an approximation of how many rows will be returned – that is our “statistics”. That in turn allows us to make smarter decisions about the plan we choose for the query.
High Level Overview of Stats

Created in various ways:
- Auto creation
- Based on indexes
- Manually

Maintained in various ways:
- Auto update based on thresholds
- Index rebuilds
- Maintenance plans
- Manual updates

Are a point in time view of the data distribution
Creating Stats – Via Indexes

When an index is created a matching stat is created. This will handle 95% of your stats needs.
Creating Stats – Via Indexes

Example of matching stat
Creating Stats - Automatic

The default setting for each database is to have automatic creation of stats enabled, allowing SQL to create a new stat if a query uses a column in a where clause or join that doesn’t have a stat. Think of this as a safety net for stats.
Creating Stats - Automatic

Example: System created statistic

System created stat on column #9 of table id 145C0A3F (in hex!)
See the resource slide for the link to the blog post by Paul Randall that explains it
Creating Stats - Manually

It’s not common, but you might need to create a statistic manually. Here is an example:

create statistics EmailAddress on person.contact (EmailAddress)
How are Statistics Updated?

Unlike indexes, statistics are a batch operation. That decreases the load on the system, but it means that over time the accuracy of the stats can decrease as the distribution of the data changes from what it was at the time we built our statistic.

The fix is to periodically update our statistics:

• By association when we rebuild indexes
• Directly, either manually or via a job
• Based on thresholds if auto update enabled
Updating Stats Via Rebuild

• This only works for a true rebuild, not a defrag/reorg!
• This only works if they created the index with the default behavior to create/maintain stats (STATISTICS_NORECOMPUTE = OFF)
• This only works for index related stats. Stats created manually or auto created are not changed as part of an index rebuild even if one of the columns is part of an index
Updating Stats Directly

The most surgical approach to updating stats is to use UPDATE STATISTICS which allows us to:

• Update a single statistic, or all stats on a table
• Specify the sampling rate or reuse the previous sample rate
• Update index based stats, other stats, or both
• Disable automatic statistics update on a stat

If you need to update all the stats in a database, look at sp_updatestats or maintenance plans
Updating Stats Directly

Examples:

• update statistics Person.contact(ndxemail) with fullscan
• update statistics Person.contact(ndxemail) with sample 50 PERCENT
• update statistics Person.contact with columns
• update statistics Person.contact with index

Note: If the table is less than 8 meg then you will get a 100% sample even if you request less.
Updating Stats Directly

Define Update Statistics Task
Configure the maintenance task.

- **Databases:** All databases
- **Object:**
- **Selection:**

**Update:**
- All existing statistics
- Column statistics only
- Index statistics only

**Scan type:**
- Full scan
- Sample by

Use this if you know you’ll be rebuilding your indexes in the same maintenance window.

Specifying less than 100% sample can decrease the time it takes, but you have to figure out how low you can go without compromising results.
Updating Stats Directly

For routine maintenance you can also use sp_updatestats:

• Only updates stats that need updating (based on update thresholds we’ll cover in a bit)

• Does rebuild stats for disabled non-clustered indexes

• By default will select a “default” sample rate, if you want to use the one you set, use ‘resample’

Sp_updatestats

Sp_updatestats ‘resample’
Updating Stats Automatically

Default is True, and in most cases should be True!
Update Thresholds

The auto update stats event will fire based on these rules:

• When table row count goes from zero to not zero
• Table had less than 500 rows and there have been more than 500 changes to the leading column of the stat since the last stat update
• Table had more than 500 rows and there have been at least 500 + 20% changes to the leading column in the stat since the last update
• For temp tables, the first update fires at six changes
Viewing Stats

As you might expect, there are a few different ways to view the statistics so you can examine the details:

• Management Studio (handy, no syntax to remember!)
• DBCC Show_Statistics

You can also get info about stats name and status by querying
• Sys.Stats
• Sys.Stats_Columns

We’re going to focus on DBCC Show_Statistics
DBCC Show_Statistics

dbcc show_statistics ('person.contact', 'ndxlastname')
Understanding the Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Updated</th>
<th>Rows</th>
<th>Rows Sampled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndxLastName</td>
<td>Sep 6 2010 8:37AM</td>
<td>19972</td>
<td>19972</td>
</tr>
</tbody>
</table>

Recent update -

100% sample - good
Understanding The Header

<table>
<thead>
<tr>
<th>Steps</th>
<th>Density</th>
<th>Average key length</th>
<th>String Index</th>
<th>Filter Expression</th>
<th>Unfiltered Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.574212</td>
<td>15.09443</td>
<td>YES</td>
<td>NULL</td>
<td>19972</td>
</tr>
</tbody>
</table>

Interesting, but not too useful to humans

No filter (ok) (SQL 2008 +)
Column Densities

Not all that interesting, but sometimes can help you realize that you might gain from re-ordering columns.
The Good Stuff

For example, "Smithson" would fall into the buck on line 160.

> Smith and <= Srini

Lucky here, we know EXACTLY how many rows match Smith
More Good Stuff

<table>
<thead>
<tr>
<th>RANGE_HI_KEY</th>
<th>RANGE_ROWS</th>
<th>EQ_ROWS</th>
<th>DISTINCT_RANGE_ROWS</th>
<th>AVG_RANGE_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>155 Sharma</td>
<td>0</td>
<td>119</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>156 She</td>
<td>0</td>
<td>123</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>157 Shen</td>
<td>1</td>
<td>88</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>158 Simmons</td>
<td>15</td>
<td>109</td>
<td>11</td>
<td>1.363636</td>
</tr>
<tr>
<td>159 Smith</td>
<td>21</td>
<td>667</td>
<td>10</td>
<td>2.1</td>
</tr>
<tr>
<td>160 Srin</td>
<td>23</td>
<td>62</td>
<td>14</td>
<td>1.642857</td>
</tr>
<tr>
<td>161 Stewart</td>
<td>21</td>
<td>93</td>
<td>16</td>
<td>1.3125</td>
</tr>
<tr>
<td>162 Suarez</td>
<td>30</td>
<td>105</td>
<td>9</td>
<td>3.333333</td>
</tr>
<tr>
<td>163 Subram</td>
<td>0</td>
<td>85</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>164 Sun</td>
<td>15</td>
<td>72</td>
<td>4</td>
<td>3.75</td>
</tr>
<tr>
<td>165 Suri</td>
<td>2</td>
<td>91</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Only 14 distinct values

For "Smithson" we don't know exactly how many rows, only that there are 23 rows other than Srin in this range.

On average, we expect to get 1-2 rows for any value OTHER than Srin.
Using Multiple Ranges

<table>
<thead>
<tr>
<th>RANGE_HI_KEY</th>
<th>RANGE_ROWS</th>
<th>EQ_ROWS</th>
<th>DISTINCT_RANGE_ROWS</th>
<th>AVG_RANGE_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>155 Shama</td>
<td>0</td>
<td>115</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>156 She</td>
<td>0</td>
<td>123</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>157 Shen</td>
<td>1</td>
<td>88</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>158 Simmons</td>
<td>15</td>
<td>109</td>
<td>11</td>
<td>1.363636</td>
</tr>
<tr>
<td>159 Smith</td>
<td>21</td>
<td>667</td>
<td>10</td>
<td>2.1</td>
</tr>
<tr>
<td>160 Srini</td>
<td>23</td>
<td>62</td>
<td>14</td>
<td>1.642857</td>
</tr>
<tr>
<td>161 Stewart</td>
<td>21</td>
<td>93</td>
<td>16</td>
<td>1.3125</td>
</tr>
<tr>
<td>162 Suarez</td>
<td>30</td>
<td>105</td>
<td>9</td>
<td>3.333333</td>
</tr>
<tr>
<td>163 Subram</td>
<td>0</td>
<td>85</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>164 Sun</td>
<td>15</td>
<td>72</td>
<td>4</td>
<td>3.75</td>
</tr>
<tr>
<td>165 Suri</td>
<td>2</td>
<td>91</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

For a wildcard like our 'S%' example, we can get a quick and close approximation of the total rows by summing range rows plus eq rows that start with S.
If All Goes Well

With the necessary stats in place and appropriate updates, then we’ve got the information we need for SQL to make a pretty good guess on how many rows will match, and from there build a query plan that matches the expected load.

This happens most of the time.

But I bet you want to hear about how things can go awry!
And When Things Go Wrong

Typically stats related problems fall into a couple of categories:

• No stats
• Out of date stats (let’s say “not updated lately”)

And one problem that can happen even with current stats:

• Uneven data distribution
No Stats = Guess = Bad!

If we have no stats for a column, we force the query optimizer to guess – not good

Normally we'll figure out missing indexes first, and that fixes the stats problem at the same time

But it is possible to have an index with no stats, or just a case where the optimizer wants stats it doesn't have - watch for the "!" on the plan operator and investigate when you see it.
No Stats - Should Be Rare

If you keep the default behaviors enabled you’ll always have stats. Well, almost always. There are a few edge cases where things don’t behave quite as expected:

• No stats on table variables
• No stats on table valued functions
• No stats on CLR columns unless binary ordering

Otherwise, if you find you’re missing stats, get that fixed and then keep it fixed!
Good Stats Gone Bad

Having a significant mismatch in actual vs estimated often indicates stale stats

<table>
<thead>
<tr>
<th>Physical Operation</th>
<th>Index Seek</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Operation</td>
<td>Index Seek</td>
</tr>
<tr>
<td>Actual Number of Rows</td>
<td>1</td>
</tr>
<tr>
<td>Estimated I/O Cost</td>
<td>0.003125</td>
</tr>
<tr>
<td>Estimated CPU Cost</td>
<td>0.0001581</td>
</tr>
<tr>
<td>Estimated Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Operator Cost</td>
<td>0.0032831 (50%)</td>
</tr>
<tr>
<td>Estimated Subtree Cost</td>
<td>0.0032831</td>
</tr>
<tr>
<td>Estimated Number of Rows</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Row Size</td>
<td>70 B</td>
</tr>
<tr>
<td>Actual Rebinds</td>
<td>0</td>
</tr>
<tr>
<td>Actual Rewinds</td>
<td>0</td>
</tr>
</tbody>
</table>

Watch for cases where the actual and estimate number of rows varies significantly.

Significant based on size, expected 10 returned 20 is fine, expected 10 returned 1000 - not good!
Advanced Techniques

• DB Setting: Update Statistics Async prevents delays when a stats update is triggered by allowing the query to use the existing plan until the new stats are ready

• Query Hint: OPTION (KEEP PLAN) changes the threshold for recompile on temp tables to match that of permanent tables (rarely used)

• Query Hint: OPTION (KEEPFIXED PLAN) will prevent recompiles based on changes to stats (rarely used)
Best Practices

• Enable auto create, auto update
• Update stats as often as you rebuild indexes, or more so
• Update only column statistics if you’ve already rebuilt your indexes in the same session
• Watch for stats related issues by checking estimated vs. actual rows in the query plan
Resources

- 2005 Stats Whitepaper
- 2008 Stats Whitepaper
- Paul Randall on Auto Created Stats
- Kim Tripp on Filtered Stats
- Glenn Berry on Out of Date Stats
- Recompilation Whitepaper
- Kendal Van Dyke on Identifying Overlapping Stats
Thanks for Attending!

Please connect with me

www.sqlandy.com
@sqlandy
www.linkedin.com/in/in/sqlandy
SP_HelpStats - Deprecated

Sp_helpstats is a quick way to return stats information about a table, but it has been deprecated. Instead, use the sys.stats and sys.stats_columns tables to get the same info.

```sql
sp_helpstats 'Person.Contact', 'ALL';
```

![Results Table]

<table>
<thead>
<tr>
<th>statistics_name</th>
<th>statistics_keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>_WA_Sys_00000009_145C0A3F</td>
<td>EmailPromotion</td>
</tr>
<tr>
<td>AK_Contact_rowguid</td>
<td>rowguid</td>
</tr>
<tr>
<td>ndxEmail</td>
<td>EmailAddress</td>
</tr>
<tr>
<td>ndxFirstName</td>
<td>FirstName</td>
</tr>
<tr>
<td>ndxLastName</td>
<td>LastName</td>
</tr>
<tr>
<td>PK_Contact_ContactID</td>
<td>ContactID</td>
</tr>
<tr>
<td>Test</td>
<td>EmailAddress</td>
</tr>
</tbody>
</table>
SP_CreateStats

Creates single column stats for any column that isn’t the leading column in an existing statistic.

EXEC sp_createstats 'indexonly';

Messages
Table 'AdventureWorks.Sales.Store': No columns without statistics found.
Table 'AdventureWorks.Production.ProductPhoto': No columns without statistics found.
Table 'AdventureWorks.dbo.Users': No columns without statistics found.
Table 'AdventureWorks.Production.ProductProductPhoto': No columns without statistics found.
Table 'AdventureWorks.Sales.StoreContact': No columns without statistics found.
Table 'AdventureWorks.Person.Address': No columns without statistics found.
Table 'AdventureWorks.Production.ProductReview': Creating statistics for the ReviewerName
Table 'AdventureWorks.Production.TransactionHistory': Creating statistics...
SP_AutoStats

Used to change the NO_RECOMPUTE setting for all statistics on a table or index. The NO_RECOMPUTE flag is stored at the stat level in sys.stats.

```sql
EXEC sp_autostats 'Person.Contact', 'OFF';
select * from sys.stats where no_recompute=1;
```
Sys.Stats

Sys.Stats and Sys.Stats_Columns let you see all the available statistics. For example, we can use this to see which stats have NO_RECOMPUTE enabled.