**The Ascending Column Problem in Fact Tables –Part two: Stat Job!**

July 7, 2011

FROM: <http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/>

In my [last post](http://kejserbi.wordpress.com/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/) I described a common problem with statistics in data warehouses. I also directed you to a trace flag that partially solves the problem. There is an issue with this trace flag: at the time of writing this blog, it does not work on a partitioned tables. Considering that large tables tend to be partitioned, this could be an issue.

Recall from my last post that old statistics will not contain information about the newly inserted rows until stats are updated. Queries asking for data that is outside the bounds of the histogram will estimate 1 row and typically end up with a bad, serial plan. The trace flag works, or doesn’t, by trying to intelligently guess how many rows are such an “out of bound” range instead of assuming the value of 1.

There is another approach that does not rely on the optimizer making guesses: You can intentionally deceive the optimizer, lie about the distribution. Recall that we want the “desired DW plan”. Alas, as with all telling of lies, this exposes you to a lot of risk.

**Disclaimer: What I am about to show you is a: “Don’t do this at home” trick. We are going to see some features in SQL Server used in a ways they are not intended for. And we are going to take control of the optimizer in ways it is not designed to handle. If you go down this path, you are on your own! Don’t call support with your issues. Just because I showed it to you, doesn’t mean you can put this in production.**

Consider this example an exploration and education of the internals of SQL Server. And think of in the context of what you might consider asking for, next time you file an [MSConnect](http://connect.microsoft.com/) item.

On the issue of [MSConnect](http://connect.microsoft.com/): If you care about big (any TB sized) DW workloads: please, please file and vote on items that make your working day hard. It helps if you describe which company you work for and what the impact is on your business of not having the feature or “fix”, I know this is not always possible. I know it takes times, and I appreciate you are busy, but Microsoft does listen, and they listen more closely when you can make it clear why this feature is needed and how it hurts you (I call this the “blood on the table” approach). If your favourite Connect item is about DW functionality, shoot me a mail (include the ConnectID please) to let me know. I may well put up a big fight for items that align with my mission in MS. Please keep in mind: I don’t care about client tools at all, I fight for server side features. We have a lot of people pushing on client side features and they are very important too, I have just chosen to specialize.

And with that said, let us dig in:

Recall that our issue was that the histogram lacked rows/buckets containing information about a range of data. The histogram in SQL server has up to 200 rows, but even if we could add a single row to the histogram on every insert, we would soon run out on a large table.

But how about this idea: Fake a histogram were every year we ever want to store (say 100 of them) contains a lot of data. In fact, make sure that every date range or value we select contain a tremendous amount of rows. One way to create such a histogram is to grow the fact table very large and then capture the stats and make it smaller again. If your fact table is not born with the right proportions, growing fat to get the right histogram proportions and then slimming down is not a very nice approach is it?

Our **Fact** table histogram is not pretty as it is, but how about doing some precision surgery instead? For that, we need an implant. First, we create a table with the same index structure as **Fact:**

**CREATE TABLE** HackStats   
(   
  date\_sk INT NOT NULL   
  , payload CHAR(4000) NOT NULL DEFAULT ‘X’   
)

**CREATE CLUSTERED INDEX** scan\_me **ON** HackStats(date\_sk)

Now, let us prepare the implant. We need something that has these characteristics:

1. Enough histogram divider rows to cover all possible data values we could ask for.
2. A very large value for RANGE\_ROWS and AVG\_RANGE\_ROWS between in each histogram row.
3. A value of 1 for the DISTINCT\_RANGE\_ROWS in each histogram row.

**Ad 1)** Make sure that we do the implant once, and that I lasts for the lifetime **Fact**.

**Ad 2)** Make sure that there is sufficient size and elasticity for any range of dates we might touch.

More about **3)** later…

With a little reverse engineering (no, I did not look at the source), we can create a histogram that match our requirement. It does not take a lot of rows in **HackStats.** Here we go:

**SET** @y = 1950   
**WHILE** @y < 2050 **BEGIN**   
  /\* make the stats dense (low distinct) within the year \*/   
 **SET** @r = 0   
 **WHILE** @r < 100 **BEGIN**   
/\* Make sure the equal rows in the histogram are never hit using invalid   
    dates  \*/   
    **INSERT** HackStats (date\_sk) **VALUES** (@y \* 10000)   
    **INSERT** HackStats (date\_sk) **VALUES** (@y \* 10000 + 5000)   
    **SET** @r = @r + 1   
  **END**   
  **SET** @y = @y + 1   
**END**

You may have been wondering why I added a **payload** column to **HackStats**. For small tables (by page size), SQL Server will do a full sample, even if we ask for a small percent. And the engine is smart enough to know that is has seen all pages. A fully sampled table cannot be hacked and made artificially pretty – in ways you will soon observe. The **HackStats** table has to be small, but large enough (in our case 20K pages) that all rows will not be sampled. And that is why we had to to pad the rows a bit.

We can now hack like this:

 /\* prepare the implant \*/

**UPDATE STATISTICS** HackStats **WITH ROWCOUNT** = 199000000   
**UPDATE STATISTICS** HackStats **WITH SAMPLE** 99 **PERCENT**

The first statement above “fakes” a large table by telling SQL Server that there are a LOT of rows in the table. The second statement samples the distribution of those rows (which we have cunningly crafted to have just the right dimensions, so we need to sample almost all of them).

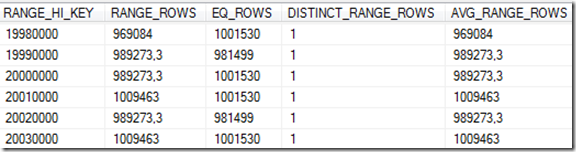
Let us check the quality of the implant:

**DBCC SHOW\_STATISTICS** (HackStats, scan\_me)   
**WITH STAT\_HEADER, HISTOGRAM**

Output (your results may vary a little, depending on which rows got sampled, rerun the sample if you don’t like the output)

[image](http://kejserbi.files.wordpress.com/2011/07/image3.png)

And:

[](http://kejserbi.files.wordpress.com/2011/07/image4.png)

Notice the neat distribution. Any query asking for a year range will get a high estimate of the outcome of scanning the table. SQL Server is smart enough to boost the row estimates based on how many total rows it believes are in the table.

And now, for the dirty little trick ([Brent Ozar](http://www.brentozar.com/), if you are reading this, just don’t say it Smile):

Let us transplant the faked, big stats to the ugly **Fact** table. First, take the implant:

**DBCC SHOW\_STATISTICS** (HackStats, scan\_me)   
**WITH STATS\_STREAM**

This gives an output somewhat like this (your result will vary):

**0x01000000010000000000000000000000B1AB0052000000….**

We don’t really have to worry what is in there, it is some binary representation of the stats. The feature to read the binary stats was put into SQL to allow database clones – and we are abusing it now. But hey, we are in the business of precision surgery: Copy the bit values to the clipboard, past and execute:

**UPDATE STATISTICS** Fact scan\_me WITH   
**STATS\_STREAM** = **0x01000000010000000000000000000000B1AB0052000000….**

And a finishing flourish, to make it fact table look really big:

/\* Make the table expensive to access in the future \*/

**UPDATE STATISTICS** Fact scan\_me **WITH PAGECOUNT** = 200000000

Validate the result:

/\* Check the patient \*/

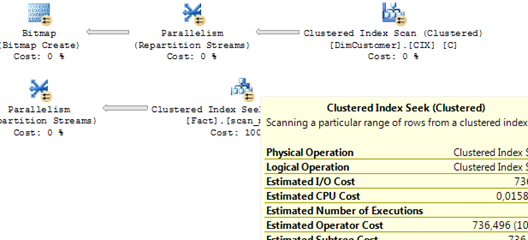
**DBCC SHOW\_STATISTICS** (Fact, scan\_me)   
**WITH HISTOGRAM**

(output similar to **HackStats**)

And let us check our problem query again:

**DBCC FREEPROCCACHE**   
**SELECT** C.shoe\_size, SUM(Measure)   
**FROM** Fact F   
**JOIN** DimCustomer C **ON** F.customer\_sk = C.customer\_sk   
**WHERE** F.date\_sk = 20010111   
  AND C.shoe\_size = 43   
**GROUP BY** C.shoe\_size

Eureka: the “Desired DW Plan”:

[](http://kejserbi.files.wordpress.com/2011/07/image5.png)

Notice something interesting: the **Estimated Operator Cost** of access is about 736. That is a REALLY expensive access to the table. Just the way we like it actually, wouldn’t want the optimizer to get any ideas that this table is cheap to access. Our little trick with hacking the page size gave our **Fact** patient quite a bit of confidence.

I deferred talking about why we need a low number of distinct rows in the histogram. For now, I leave this as an exercise to the reader. Hint: it is related to the bitmap filters and the way **Fact** is accessed.

And remember folks, you do this at your own risk, you are on your own and in unsupported land here… Don’t come crying to support if you break something. Ask for a connect item instead.

[About these ads](http://en.wordpress.com/about-these-ads/)

**Share this:**

* [More](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/)

[Comments (2)](javascript:void(0);) [Trackbacks (0)](javascript:void(0);) [Leave a comment](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/#respond) [Trackback](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/trackback/)

1. http://1.gravatar.com/avatar/77f776c2eaf0cc691e8a0880bb8a191f?s=32&d=retro&r=G

[Brent Ozar](http://www.brentozar.com)

July 31, 2011 at 01:35 | [#1](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/#comment-337)

[Reply](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/?replytocom=337#respond) | [Quote](javascript:void(0);)

HAHAHA, I love this trick! I was sitting here reading it (thanks to a link from James Rowland-Jones) and busted out laughing when I saw my name. This is awesome. Nice work as always, sir.

* + http://0.gravatar.com/avatar/93e3150ebaf2321aef3dc4faee057861?s=32&d=retro&r=G

Thomas Kejser

July 31, 2011 at 04:08 | [#2](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/#comment-338)

[Reply](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/?replytocom=338#respond) | [Quote](javascript:void(0);)

Thank you sir. Have fun experimenting, it probably wouldn’t take much to create a little tool that “fixes” the distribution to your preference.