**The Ascending Key Problem in Fact Tables– Part one: Pain!**

July 1, 2011

FROM: <http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/>

Time for another deep dive into SQL Server. In the last couple of days, I have been having coffee with [Joe Chang](http://sqlblog.com/blogs/joe_chang/default.aspx) in the afternoon, who some of you may know. He is visiting Sweden a lot by the way, so shoot him an email if you are interested in having him present. Joe and I ended up talking about an interesting problem in data warehousing: the ascending key problem. I think it is best illustrated by an example.

Let us say you have a pure star schema (sorry, Inmon people), with a large fact table. If you are following the guidance in the [Top 10 Best Practices](http://sqlcat.com/top10lists/archive/2008/02/06/top-10-best-practices-for-building-a-large-scale-relational-data-warehouse.aspx) from [SQLCAT](http://sqlcat.com/) (bullets 1,2 and 9), you typically partition the fact table by date, and you  cluster on date too. On of the reason you do this, is that it helps you do fast range scans.

Let us try an example star schema that follows this guidance:

**CREATE PARTITION FUNCTION** pf\_daily(INT) **AS RANGE RIGHT FOR VALUES**   
(20000101, 20000102   
, 20000103, 20000104   
, 20000105, 20000106   
, 20000107, 20000108   
, 20000109, 20000110   
, 20000111, 20000112   
, 20000113, 20000114   
)   
**CREATE PARTITION SCHEME** ps\_daily **AS PARTITION** pf\_daily **ALL TO** ([PRIMARY])

**CREATE TABLE** Fact   
(   
  date\_sk INT NOT NULL   
  , customer\_sk INT NOT NULL   
  , payload CHAR(80) NOT NULL DEFAULT REPLICATE(‘X’, 80)   
  , measure MONEY NULL   
)   
/\* Turn off stats (as per typical DW) \*/   
**EXEC** sp\_autostats ‘Fact’, ‘OFF’   
**CREATE CLUSTERED INDEX** scan\_me **ON** Fact(date\_sk) **ON** ps\_daily(date\_sk)

**CREATE TABLE** DimCustomer   
(   
  customer\_sk INT NOT NULL   
  , shoe\_size INT NOT NULL   
)   
**CREATE UNIQUE CLUSTERED INDEX** CIX **ON** DimCustomer(customer\_sk)

If you have a very large fact table, you often don’t want to run stats in the middle of a query, so we have turned auto stats off. In this context, recall that stats in SQL Server are per table, not per partition – which can be a pretty expensive operation. We really want to control when stats are run for large tables.

To generate some test data, let us fill up 10 partitions with 50MB each. For the customer dimension, we will use 100K rows:

**/\* Generate test data \*/**

**DECLARE** @i INT   
**DECLARE** @customer\_size INT   
**SET** @customer\_size = 100000   
**DECLARE** @day\_size\_pages INT    
**DECLARE** @day\_size\_rows INT   
**SET** @day\_size\_pages = 50001 / 8   
/\* last sum are row width plus uniquefier \*/

**SET** @day\_size\_rows = @day\_size\_pages \* 8060 / (4 + 4 + 80 + 8)

/\* Create one day of data \*/   
**CREATE TABLE** OneDay   
(   
  date\_sk INT NOT NULL   
  , customer\_sk INT NOT NULL   
  , payload CHAR(80) NOT NULL DEFAULT REPLICATE(‘X’, 80)   
  , measure MONEY NULL   
)   
**SET** @i = 0   
**WHILE** @i < @day\_size\_rows **BEGIN**   
  **INSERT** OneDay (date\_sk, customer\_sk, measure)   
  **VALUES** (’20000101′, @customer\_size \* RAND(), RAND() \* 1000)   
  **SET** @i = @i + 1   
**END**

/\* create the customer dimension \*/   
**SET** @i = 0   
**WHILE** @i < @customer\_size **BEGIN**   
  **INSERT** DimCustomer (customer\_sk, shoe\_size)   
  **VALUES** (@i, 36 + @i % 15)   
  **SET** @i = @i + 1   
**END**

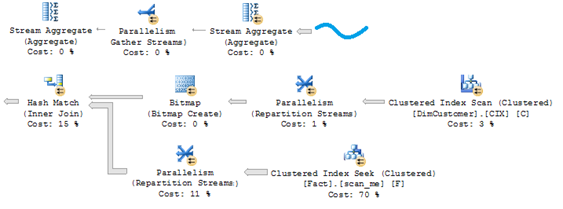
/\* Insert ten days of data in the fact \*/   
**SET** @i = 0   
**WHILE** @i < 10 **BEGIN**   
  **INSERT** Fact (date\_sk, customer\_sk, measure)   
  **SELECT** 20010101 + @i, customer\_sk, measure   
  **FROM** OneDay   
  **SET** @i = @i + 1   
**END**

Good, now we can get started on some testing. Let us first execute a typical, star schema query. But before that, we will update the statistics. To be nice to the query, we will even do a full scan.

**UPDATE STATISTICS** Fact **WITH FULLSCAN**   
**DBCC FREEPROCCACHE**

/\* An ordinary star-schema query arrives \*/   
**SELECT** C.shoe\_size, SUM(Measure)   
**FROM** Fact F   
**JOIN** DimCustomer C **ON** F.customer\_sk = C.customer\_sk   
**WHERE** F.date\_sk = 20010101   
  **AND** C.shoe\_size = 43   
**GROUP BY** C.shoe\_size

On my 2-core laptop and a hot buffer pool, this query runs in a little over 100ms, touching over 500K rows. Life is good. The query plan is what I like to refer to as “the desired DW plan”:

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

All the neat stuff in the SQL Server optimizer that we like for DW workloads:

* Hash Join to dimension tables
* Bitmap index pushdown to the fact table ([BOL link](http://msdn.microsoft.com/en-us/library/bb522541.aspx))
* The right join order (build hash on dimension, use fact table to probe)
* Parallelism
* Cluster index seek into nice, fat range of facts

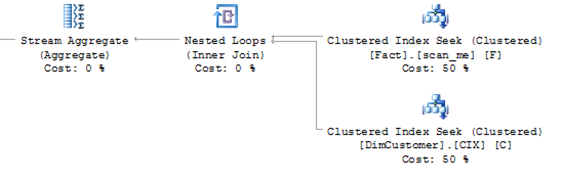
But warehouses are not always nice to us. During the night, some evil ETL developer comes along and loads a new day of data:

/\* Nightly ETL run adds another day of data \*/   
**INSERT** Fact (date\_sk, customer\_sk)   
**SELECT** 20010111, customer\_sk **FROM** OneDay

Alas, the ETL guy did not have time to run statistics – maybe he forgot. Now, let us run the same DW query again, but this time with the newly loaded data:

**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

over 2 seconds on my 2 core laptop, more than 20 times slower! This is the time when people typically complain that SQL server is broken and doesn’t scale. Instead, we adopt a less moaning stance and start a little bit of troubleshooting. As always, we begin with a look at the query plan:

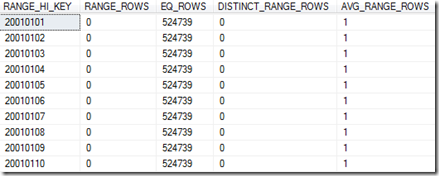
[](http://kejserbi.files.wordpress.com/2011/07/image1.png)

Well, apart from the plan fitting the width of my blog, there is nothing good to say about it. What just happened?

This is the part where we up the geek bar just a little: SQL Server gives you the ability to dig into the statistics like this:

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

Output:

[](http://kejserbi.files.wordpress.com/2011/07/image2.png)

Oh no, the value 20010111 is not even in the histogram. The optimizer estimates zero rows coming out of the fact table. Zero rows are very cheap to access, so the overhead cost of all those nice, parallel and hash join optimizations are not deemed worth it. Of course, the problem here is that the histogram is lying, there are 500K rows in that date range! Loop joins are expensive in that case.

That hurt, what now?… Well, I guess you just have to remember to update your statistics won’t you? And for those of you that can see a problems with the that (and some will) – start the commenting, go vote on Connect, and of course: stay tuned…

… And by the way, one last last thing, until we meet again in the next post: Have a look at this: [Ascending Keys and Auto Quick Corrected Statistics](http://blogs.msdn.com/b/ianjo/archive/2006/04/24/582227.aspx).

Have a great weekend everyone, and consider updating your stats if the warehouse loads data on Sunday…

**Related Posts:** [Part 2 – Stat Job](http://blog.kejser.org/2011/07/07/the-ascending-column-problem-in-fact-tables-part-two-stat-job/)

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

**Share this:**

* [More](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/)

[Comments (2)](javascript:void(0);) [Trackbacks (2)](javascript:void(0);) [Leave a comment](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/#respond) [Trackback](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/trackback/)

1. http://2.gravatar.com/avatar/e0b63be776ce8b2e674ad5fd7ce45536?s=32&d=retro&r=G

[Ayyappan Thangaraj](http://SQLServerRider.wordpress.com)

July 6, 2011 at 17:39 | [#1](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/#comment-326)

[Reply](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/?replytocom=326#respond) | [Quote](javascript:void(0);)

Excellent post. Does statistics solves majority of the performance problems?

Thomas Kejser

July 7, 2011 at 17:48 | [#2](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/#comment-328)

[Reply](http://blog.kejser.org/2011/07/01/the-ascending-key-problem-in-fact-tables-part-one-pain/?replytocom=328#respond) | [Quote](javascript:void(0);)

Ayyappan: Statitistics don’t solve every performance problem. But having good statistics will help the optimizer make the best choices on the data model you have.

In my experience, the vast majority of problems in DW systems are caused by poor data model, and by trying to SAN for Data Warehousing storage. In this blog, I assume all the other homework has been done and you are looking for some extra tips :-)