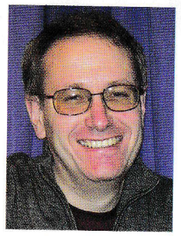
Massaging Statistics in Heterogeneous ASE Environments

By Raymond Mardle

A new stored procedure to support better statistics management



Raymond Mardle is an independent consultant specializing in ASE, Replication Server, and Sybase IQ. He can be reached at raymondpm1@yahoo.co.uk.

Recently, a client experienced a system problem in which the company found they had insufficient statistics in their large tables to allow the optimizer to make the best decisions for joining multi-tens of millions rows tables. Despite advances in statistics storage since they were first moved from table distribution pages (pre-ASE 11.9) to systabstats and sysstatistics in ASE 11.9 onwards (including sampling and the "histogram tuning factor" introduced in ASE 12.5.0.3), using ASE 12.0 meant that some of the newer abilities were not available.

Problems were being encountered mostly with the number of histogram steps that could be used, but they were also experiencing issues with some distribution values. The latter problem was resolved for two columns by using sp_modifystats after statistics were updated to change the selectivity of each of the columns. Each of the two columns had to have a sequence of SQL hand-coded for them, to cater for their different requirements. However, there did not seem to be a simple solution to the problem of histogram steps, taken in the context of a constant update cycle that could see an unknown number of tables recreated each week due to schema changes. This article discusses how we developed a new stored procedure to manage statistics in this environment.

Statistics Updating

In examining the situation which had developed, we found that tables which had remained unchanged in databases in servers updated from pre-ASE 11.9 to ASE 12.0 tended to have more than the now-standard default of 20 histogram steps for some columns. However, any tables created after the upgrade to ASE 12.0 or higher had a default of 20 histogram steps, as did non-leading columns in indexes of the unchanged tables. In a table with
20-plus million rows and a column with many millions (or just thousands) of distinct values, 20 histogram steps is decidedly not enough.

A suite of generic housekeeping scripts did not allow statistics updating to be tailored for individual tables in individual databases, nor use of a "scatter gun" approach to set a high number of histogram cells for every table in every database. A different approach was required to allow the number of requested steps to be set without running individual **update statistics** commands against specific columns.

One method we tried consisted of increasing the number of histogram steps for a table was to run a one-off **update index statistics**, specifying the number of steps relevant for the indexed column with the highest number of distinct values (which would also be applied to all columns in all indexes for the table; though this is not a problem, I did not find it aesthetically pleasing). This required that the number of exception requests had to be increased to run customized commands outside of the normal business week and housekeeping window. Another method, as part of the tasks for a schema change (usually done as table renames, table creation, data insertion, index creation and, if required, trigger creation) was to load a dummy row into the table after its creation, run **update statistics** against each indexed column, specifying a number of steps relevant to that column, and then delete the dummy row before loading in the real data. Both of these methods required customization beyond that usually performed for a table.

What was needed was a way of changing a column's requested step value without running one or more specially tailored **update [index] statistics** command(s).

How Steps are Held in sysstatistics

The sysstatistics table has an impenetrable column naming system. It is defined in ASE 15.0 with a total of 92 columns:

```
create table sysstatistics
(statid         smallint        not null,
 id             int             not null,
 sequence       int             not null,
 moddate        datetime        not null,
 formatid       tinyint         not null,
 usedcount      tinyint         not null,
 colidarray     varbinary(100)      null,
 c0             varbinary(255)      null,
 c1             varbinary(255)      null,
 c2             varbinary(255)      null,
 c3             varbinary(255)      null,
 c4             varbinary(255)      null,
 c5             varbinary(255)      null,
 :
 c78            varbinary(255)      null,
 c79            varbinary(255)      null,
 indid          smallint        not null,
 ststatus       smallint            null,
 partitionid    int             not null,
 spare2         smallint            null,
 spare3         int                 null)

create unique clustered index csysstatistics on sysstatistics
    (id, indid, partitionid, statid, colidarray, formatid, sequence)
```

(ASE 12.0 and 12.5 do not have the last five columns after c79, and the index for those ASE versions does not have indid and partitionid in it. ASE 15.0 versions of the system procedures discussed in this article have not been written yet.)

After some investigation, it was found that the requested steps values are stored in the c5 column of a specific type of row (formatid = 100) within sysstatistics. The c4 column in the same row contains the actual number of histogram steps. Converting the varbinary (255) column to int allowed the requested and actual steps to be determined.

Taking into account the differences in the colid column in syscolumns between ASE 12.0 and ASE 12.5 (tinyint, up to 255 columns, in ASE 12.0; and smallint, up to 32,767 columns in ASE 12.5), it was a relatively simple exercise to write a stored procedure to update the number of requested steps for
a specific column in a table. However, this only worked where a column already had a value in c5 (i.e., it had previously explicitly had its statistics updated or by being the leading column in an index created on the table, when the table had data for both tasks). If a column did not have any statistics, it proved impossible to set the number of requested steps for it… initially.

What Was Tried for Columns with No Statistics

I came to know the row that stores the requested and actual steps as a "format 100." A single dummy "format 100" row was inserted into the sysstatistics table for the required column with the required c5 value and a value of zero for the c4 column. This caused the optimizer to throw an error, because it was not designed for an indication that statistics exist but there are no actual histogram steps.

Instead of zero, a value of one was then tried for the c4 column. This caused connections to stack trace and disconnect when the table column with the bogus c4 value was accessed for queries. As ASE was not written to cater for a row in sysstatistics solely for requested and actual steps without a corresponding histogram cell, it complained in the only way it could—and who can blame it. The designers of statistics in ASE 11.9 surely did not envisage the sort of brute force method I was trying for getting the requested steps set for a column without statistics.

The missing part was a "format 104" row, to hold a dummy value of NULL with a weight of one for a single histogram step. Inserting the "dummy format 104" row with a c4 value of one in the "format 100" row allowed the relevant columns to be accessed with no problems. An **update [index] statistics**, when the table contains data, replaces the "dummy format 104" entry with valid histogram data.

Before the "dummy format 104" row method was uncovered, the changing of the c5 column value for columns with statistics had proven to work with no problems. Updating the index statistics for a table after the value of c5 was changed allowed histogram steps to be stored up to the new number of requested steps. The extension of adding the "dummy format 104" row for columns without statistics did not cause any problems during testing on Solaris, but its first use in a production ASE server was approached with a little trepidation. It was one thing to change the value in an existing column but a completely different level of chutzpah was required to blithely go around adding extra rows to a system table. However, the solution did work.