Techniques for Managing Large Data Sets: Compression, Indexing and Summarization  

Abstract

Storage space and accessing time are always serious considerations when working with large SAS® data sets. Two features available in the SAS System, compression and indexing, provide you with ways of decreasing the amount of room needed to store these data sets and decreasing the time in which observations are retrieved for processing. A third technique, summarization, can also be used to manage large data sets.

This tutorial focuses on the “hows, whens, and wherefores”: it offers several methods for decreasing data set size and processing time, considers when such techniques are particularly useful, and looks at the tradeoffs for the different strategies.

Know Your Data

Two techniques for managing large data sets, compression and indexing, were introduced in the 6.06 version of the SAS System. A third technique, summarization, has always been a mainstay of the SAS System. More and more, because of the need to make rapid business decisions, it pays to consider how best to improve the access to these large data sets, and thus to consider whether compression, indexing, and/or summarization might play a part.

It is extremely important to thoroughly know your data before using any of these three techniques. For example, consider these questions:

- How “dense” is the data? If a data set of substantial size has many fields with long strings of repeated values or blanks, and storage limitations are of particular concern, it may be an ideal candidate for compression.

- Upon which variables are users or applications likely to process or classify the data? Some forethought along these lines will optimize the use of indexes.

- How evenly distributed are values of variables? Are there approximately the same number of occurrences for each value of the variable? Do the variables have very few, or very many, distinct values? The proper combination of these factors can also optimize the use of indexes.

- How is the data set being used? For once-a-day or once-a-week reporting? For EIS applications? And how often is the data refreshed? Is it necessary to be able to get to the detail level of the data, or would summarized, much smaller data sets improve access?

Knowing the answers to these questions, and others that may occur to you, will go a long way toward assuring that you make use of the best techniques for the particular situation.

Benchmark Your Results

In addition to knowing your data, it is very important that you carefully test your hypotheses by benchmarking your results. The process of benchmarking involves monitoring how long it takes to run a real job on real data under normal conditions. How much space does an uncompressed data set take up? Now compress it; how much storage have you saved? Use the compressed data set in a typical production application. Is processing time adversely affected because of compression? How long does a particular job take now, using a data set that is not indexed? How long does the job take if the data set has an index? How long does it take for, say, a SAS/EIS® 3-D business graph to display? Is it feasible to display the graph using a summarized form of the data?

Benchmarking may also mean running many different typical queries and comparing the results, or comparing the reduction in size with compression vs. access speed on not just one but many data sets. This is because every situation will yield different results, and the trick is to utilize the technique or techniques that produce the most satisfactory results most of the time.

Furthermore, it is important for this testing to occur under actual working conditions. This means that if the data sets used in the job are of a certain size, the tests should be performed on data sets of the same size. If the data normally resides on a file server, then testing should be done when network traffic is at its usual pitch. If the jobs are typically run on a PC
of a certain speed or memory size, then testing should be done on a similar PC. Replicating the typical conditions for testing purposes will most closely identify how compressing, indexing, or summarizing data sets will work in actual practice.

This tutorial will provide you with the information to use the techniques of compression, indexing, and summarization, and will suggest pros and cons. But it is very much up to you to verify that the use of these techniques for your specific data sets and for your specific uses of those data sets is appropriate.

Tutorial Data Sets and Hardware / Software Environment

For the purposes of this tutorial, benchmarking took place on a wide variety of data sets, with varying numbers of observations and variables. The largest data set was nearly a million observations in size; the ones used most often in testing were nearly a half million observations. The order of the observations was sorted or randomized, indexes were added or removed, and data sets were summarized in different ways for the various tests. For a presentation on this topic given nearly two years ago, the tests were performed on the MVS operating system, using SAS Version 6.08. For this paper, the testing took place on a PC running Windows NT, using SAS Version 6.12. In some cases, results varied because of the differences in operating systems and how they optimize certain procedures; these differences will be cited below.

For testing purposes, the SAS System and the data were both loaded locally on the PC, so neither network traffic nor any issues of contention were involved in the mix.

Data Set Compression

Data set compression is a technique for “squeezing out” excess blanks and abbreviating repeated strings of values for the purpose of decreasing the data set’s size and thus lowering the amount of storage space it requires.

In uncompressed data sets,

- each observation occupies the same number of bytes.
- A variable occupies the same number of bytes in each observation.
- Character values are padded with blanks.
- Numerical values may be padded with binary zeroes.

Thus, two observations containing values for last name and first name might look like this:

<table>
<thead>
<tr>
<th>ADAMS</th>
<th>MIKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARNET</td>
<td>MARYBETH</td>
</tr>
</tbody>
</table>

Each observation takes 35 bytes of storage.

In compressed data sets, however,

- an observation is treated as a single string of bytes.
- Variable types and boundaries are ignored.
- Consecutive repeating characters are collapsed into fewer bytes.
- There is overhead per observation used for compressed length, flags, etc.

The same two observations, if compressed, might look like this:

<table>
<thead>
<tr>
<th>@ADAMS#MIKE#</th>
</tr>
</thead>
<tbody>
<tr>
<td>@BARNET#@MARYBETH#</td>
</tr>
</tbody>
</table>

The @ represents how many uncompressed letters follow. The # represents the number of blanks repeated at this point. Here, the first observation takes 13 bytes of storage, and the second, 18 bytes.

How to Compress a Data Set

One way to compress a data set is to use the COMPRESS=YES system option:

```sas
options compress=yes;
```

All subsequently created data sets will be stored in compressed form.

A second way to compress a data set is to use the COMPRESS=YES data set option. This can be utilized anywhere a new data set is being created, for example:

```sas
data sasuser.new(compress=yes);
set sasuser.master;
… other statements …
run;
```

or

```sas
proc sort data=sasuser.finance
       out=sasuser.ytd (compress=yes);
   by account;
run;
```
The data set’s descriptor portion records that the data set is compressed. Here is a portion of the output from the CONTENTS procedure for a compressed data set. Note the highlighted line.

<table>
<thead>
<tr>
<th>Observations:</th>
<th>466560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables:</td>
<td>12</td>
</tr>
<tr>
<td>Indexes:</td>
<td>0</td>
</tr>
<tr>
<td>Observation Length:</td>
<td>149</td>
</tr>
<tr>
<td>Deleted Observations:</td>
<td>0</td>
</tr>
<tr>
<td>Compressed:</td>
<td>YES</td>
</tr>
<tr>
<td>Reuse Space:</td>
<td>NO</td>
</tr>
<tr>
<td>Sorted:</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Compression: Considerations**

In general, the advantages of utilizing compression are that the amount of storage needed for the data set is reduced, and the number of input/output operations required to read from or write to the data set is reduced, too.

Consider this section of PROC CONTENTS output for a data set with nearly a half million observations.

```
......Engine/Host Dependent Information......

Data Set Page Size: 8192
Number of Data Set Pages: 8641
```

A data set page is defined as the unit of data transfer between the SAS engine and the operating system. It occupies one or more physical blocks. An algorithm determines a page’s optimum size, and is fixed in size when the data set is created.

The amount of storage required for this data set can be calculated by multiplying the data set page size by the number of pages, or 8192 X 8641, for a total of 70,787,072 bytes.

After compressing the data set, the same portion of PROC CONTENTS output looks like this:

```
......Engine/Host Dependent Information......

Data Set Page Size: 8192
Number of Data Set Pages: 6033
```

This time, the calculation of the data set page size is 8192 bytes, multiplied by the number of data set pages, 6033, computes to 49,422,336 bytes.

```
data halfmil(compress=yes);
   set nesug97.halfmil;
   run;
```

NOTE: The data set WORK.HALFMIL has 466560 observations and 12 variables.

NOTE: Compressing data set WORK.HALFMIL decreased size by 30.18 percent.

Compressing other data sets can show a much greater saving in storage size, especially if there are, for example, many numeric variables whose values are stored by default in eight bytes of storage and where those values are mostly small integers, or many character variables whose values include many blanks. If the data set is very small, or if the variables’ values are stored in coded form and thus do not have any padded blanks, then much less saving of space may be evident. In some cases, a compressed data set can even be larger than the original.

Sometimes, giving thought as to how data are stored in a data set in the first place can yield impressive savings in size, and not require compression at all. For example, consider a data set of 50,000 observations which contains responses to a 20-question survey. The possible responses are 1, 2, 3, or 4. First, store the responses as numeric values, thus occupying, by default, eight bytes each. A small portion of the output from PROC CONTENTS displays this:

```
......Engine/Host Dependent Information......

Data Set Page Size: 8192
Number of Data Set Pages: 6033
```

Compressing the data set decreases the storage size by 54.58%. Now, store the responses as character values, thus occupying only one byte each:
This time, compression only causes a decrease in size of 6.85%. However, this second data set, in its uncompressed form, occupies only 59% as much storage space as the first data set in compressed form! This example suggests that, given the opportunity to determine what the attributes of a data set’s variables are and how the data are stored, much thought can be given to matters of efficiency. However, often such decisions are not in our hands, and a technique like compression may prove useful when storage is a consideration.

There are some tradeoffs to consider when using compression.

- Compressed data sets cannot be accessed by observation number.
- Compressed data sets cannot be randomly accessed with the POINT= option.
- While I/O is generally less when using a compressed data set, CPU time is increased, because a data set must be uncompressed to be processed.
- An updated observation may not fit into its original location, sometimes resulting in increased I/O.

For these reasons, it is necessary to take into account not only the space saving afforded by compressing a data set, but whether this technique impacts the speed at which the data set can be processed.

**Compression: Guidelines**

Keeping in mind variations in saving of storage space experienced by compressing data sets, changes in processing time due to compression, and other such considerations, compression makes good sense under the following conditions:

- Compress data sets when storage availability is a matter of particular concern.
- Compress data sets of significant size and, preferably, with much repeating data or padded fields.

**Data Set Indexing**

Consider how you would utilize the index at the back of this NESUG Proceedings. If you were interested in a particular topic, such as compression, you would flip to the keyword index, find “compression” in the “C” section, note the pages that are listed following the topic, and turn directly to those pages in the volume to read the information. For five, ten, or even twenty or thirty page references, this technique proves to be far faster than leafing through every page of the book in hopes of spotting something about the topic.

Similarly, a data set index is a data structure that specifies the location of observations based on the values of one or more key variables. If we could look at the index directly, we might see a table of the unique values of the indexed variable followed by the location of the observations that contain those values. An index on the variable MONTH, for example, might look like the following, where the numbers 1, 2, and 4 represent data set pages, and the numbers in parenthesis are the relative observation numbers within that page matching each distinct key value:

<table>
<thead>
<tr>
<th>Month</th>
<th>Pages</th>
<th>Observation Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>1(1,2,3,...) 2(…)</td>
<td></td>
</tr>
<tr>
<td>February</td>
<td>1(22,23,...) 4(…)</td>
<td></td>
</tr>
</tbody>
</table>

The reason for creating an index for a data set is to speed the access to a subset of the data. Thus, it makes good sense to give thought to which variables are most frequently accessed and what sort of subsets are most often required. Furthermore, simply having one or more indexes associated with a data set does not ensure that they will be used, or that their use ensures the most efficient means of accessing the observations in the data set. For these reasons, it is important to benchmark the use of your data set’s indexes and determine whether, in fact, they are improving access.

The data set’s descriptor portion records that the data set has indexes associated with it. Here is a portion of the output from the CONTENTS procedure for an indexed data set. Note the highlighted line.
Further down in the PROC CONTENTS output, the index is listed:

<table>
<thead>
<tr>
<th>#</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MONTH</td>
</tr>
</tbody>
</table>

You can tell whether an index is being used for a particular operation by putting the option MSGLEVEL=I into effect. Submit the following OPTIONS statement at the beginning of your SAS program:

```sas
OPTIONS msglevel=i;
```

If the index is used, you will see a note like this in your log:

```sas
INFO: Index MONTH selected for WHERE clause optimization.
```

### How to Index a Data Set

There are two types of indexes: simple and composite. A simple index is structured upon one variable in the data set, while a composite index is structured upon two or more variables. A data set can have any number of indexes associated with it, of either type.

If you are using a DATA step to create the data set, you can create the index at the same time. A simple index is named the same as the variable on which it is based. For example, the following DATA step creates an index named CITY on a variable of the same name found in SASUSER.MASTER:

```sas
data sasuser.sales (index=(city));
set sasuser.master;
... other programming statements ...
run;
```

If a SAS data set already exists, you can use the DATASETS procedure to add an index. To add a simple index on the variable STATE to the data set SASUSER.SALES, the syntax is:

```sas
proc datasets lib=sasuser;
modify sales;
index create state;
quit;
```

The syntax for adding a composite index named BOTH based on the variables STATE and CITY, use the following form of the INDEX statement of the procedure DATASETS:

```sas
proc datasets lib=sasuser;
modify sales;
index create both=(state city);
quit;
```

Similarly, you can use the DATASETS procedure to delete an index:

```sas
proc datasets lib=sasuser;
modify sales;
index delete state;
index delete both;
quit;
```

A third method for creating an index is by opening the ACCESS window from within Display Manager, typing a ‘C’ beside the data set you would like to index to open the CONTENTS window, and from the CONTENTS window command line, typing the INDEX CREATE command. The Index Create window opens, and you can specify the variable or variables to be indexed.

### Indexing: Considerations

An index can provide fast access to a subset of observations, and return observations in sorted order, even when the data file is not stored in that order. An index is

- updated automatically when observations are added to or deleted from the data set.
• is deleted automatically when a data set is rebuilt with the DATA step, sorted with the FORCE option in PROC SORT, or physically deleted.
• resynched when a data set or variables in a data set are renamed.

The SAS System determines when to use the index. An index will always be used when a SET statement or MODIFY statement contains the KEY= option. An index is considered for use when

• a BY statement variable list begins with an index key variable or a composite index primary key variable.
• a WHERE expression references a simple index key variable or a composite index primary key variable.
• an SQL join involving indexed variables is requested.

An index is not used

• with the subsetting IF statement in a DATA step or a FIND command in PROC FSEDIT.
• for BY-group processing based on index variables, if the index has the NOMISS attribute, or if the data set is sorted by the BY variables.
• if the NOMISS attribute is set and a WHERE expression includes missing values (whether or not there are missing values in the data).
• if the SAS System determines that it is more efficient to read the data sequentially.

On the mainframe, using SAS version 6.08, benchmarking revealed that any processing that involved retrieving up to about 25-30% of the observations in the data set was always faster if an index was used. In the Windows 6.12 version of SAS, the results were not as consistent. If the index was based on a variable with low cardinality (in other words, few distinct values), and/or if the values demonstrated high skewness (were scattered randomly throughout the data set), an index did not appreciably improve access time, and in fact sometimes resulted in slower access time than not having an index at all.

The difference in performance is a direct result of how the two operating systems manage the process of sequential access. When the unique values are scattered randomly throughout the data set, and thus each data set page must be read to retrieve the selected observations even in the presence of an index, the reading becomes more sequential than random. On MVS, a “channel program” strings together multiple I/O requests, thus optimizing these sequential reads. With no such internal optimization on Windows, the time to perform the sequential reads is compounded by the time it takes to access the index itself.

A way to consistently improve accessing time when using an index, however, is to also sort the data set on the indexed variable. In the tests performed for this paper, indexing and sorting the data set consistently speeded access to up to 66% of the observations requested for WHERE processing in a data set.

There are some tradeoffs to consider when using indexing.

• Extra CPU cycles and I/O operations are needed to create an index.
• Extra disk space is needed to store an index.
• Extra memory is needed to load index pages and code for use.
• Extra CPU cycles and I/O operations are necessary to maintain an index.

Indexing: Guidelines

Keeping in mind the necessity of carefully testing your results, it is valuable to consider indexing a data set when:

• there are a fairly large number of distinct values for the variable to be indexed
• the data is not randomly scattered throughout the data set, or, better yet, the data set can be sorted by the same variable as is used by the index
• the data set is frequently queried and subset on the indexed variables.

Summarization

Many processes in SAS require that data be “collapsed” into subgroups or classes or categories for analysis and reporting. Some procedures that utilize the CLASS statement, like MEANS and TABULATE, do this clustering automatically. Others which use the BY statement may require that the data set be sorted first in order to do their clustering. Still others, like SQL and GCHART, have statements or clauses that enable you to specify how the data should be grouped.

To know whether storing your data in summarized form prior to processing is a technique that might work for you, consider the following questions:
To what level of detail must the data be accessible? In the case of “drill-down” applications, such as in SAS/EIS or SAS/AF® objects, it may be possible to summarize data to several levels, and maintain the detail data set separately for special requests.

How often is the data refreshed? Since summarizing a very large data set may take a significant amount of time, this process may be best performed overnight or over a weekend. This means that a report or graph based on a summarized version of the data may not reflect the most current additions to the detail file.

Is it possible to anticipate how best to summarize the data? In other words, are standard reports generated or consistent requests made that allow for a fixed form of summarization to be used, or is any unexpected combination of clustering and analysis required at any given time?

Summarization seems most indicated as a viable technique when the pattern for clustering the data is defined, as for SAS/EIS and SAS/AF applications and predefined reports.

A new data structure, the MDDB (multi-dimensional data base), also stores data in summarized form, and is uniquely suited to SAS/EIS applications. However, a discussion of MDDB is beyond the scope of this paper.

How to Summarize a Data Set

The SAS System provides many procedures for summarizing your data. One of these is PROC SUMMARY. To summarize the nearly one-half million-observation SALES data set referred to in other portions of this paper by year, quarter, and month, use the following code:

```sas
proc summary data=nesug.sales;
class year qtr month;
var dolls;
output out=summary(where=(_type_=7))
   sum=sumdol;
run;
```

PROC SUMMARY, by default, produces the N-WAY results of combining every combination of the variables year, quarter, and month. The first observation in the output data set has a _type_ value of 0 and contains the sum of sales (variable DOLLS), stored in a variable named SUMDOL, for the entire data set. Observations where _type_ is equal to 1 contain the summaries for month; _type_ 2 are observations for month and quarter, and so on.

Subsetting on a _type_ value of 7 leaves only the observations representing the unique occurrences of all three variables. The first three observations in the output data set SUMMARY look like this:

<table>
<thead>
<tr>
<th>OBS</th>
<th>YEAR</th>
<th>QTR</th>
<th>MONTH</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>SUMDOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1993</td>
<td>1</td>
<td>February</td>
<td>7</td>
<td></td>
<td>8977440109</td>
</tr>
<tr>
<td>2</td>
<td>1993</td>
<td>1</td>
<td>January</td>
<td>7</td>
<td></td>
<td>19252033607</td>
</tr>
<tr>
<td>3</td>
<td>1993</td>
<td>1</td>
<td>March</td>
<td>7</td>
<td></td>
<td>19414047404</td>
</tr>
</tbody>
</table>

This summarization reduced the size of the data set from over 466,000 observations to 51.

Summarization: Considerations

Compare these two invocations of PROC GCHART:

```sas
proc gchart data=nesug.halfmil;
  vbar year / discrete sumvar=dolls type=sum;
quit;

proc gchart data=nesug.summary;
  vbar year / discrete sumvar=sumdol type=sum;
quit;
```

The first step produces a vertical bar chart with a separate bar for each year, and uses as its input source the detail data set. The second produces the identical bar chart, but utilizes the summary data set. The first step took 18 times longer to produce the results.

Other tests involving using the detail data set as compared to a summarized data set resulted in large differences in processing time, and the reasons are clear - the summarized data set has vastly fewer observations, already has the data in the necessary “collapsed” form, and thus requires much less I/O and CPU time for processing.

Summarization: Guidelines

The technique of summarization requires that it is acceptable to pre-process the data, and thus assumes that a periodic refreshing of the data, whether it be every night or once a week or once a month, is sufficient for keeping the data current. Also, it must be known in advance how the data needs to be summarized, and that some level of roll-up above detail level is acceptable for the type of reports or applications the data is used for. It is possible that anticipating every possible summary combination could lead to having a large number of summary data sets, but since they tend not to be very large, the tremendously increased speed of processing may offset the greater maintenance required to manage the data library.
Conclusion

The tools exist within the SAS System to write extremely efficient programs and to store SAS data sets in more efficiently used storage, and to improve the accessing of those data sets. This paper has suggested three techniques to implement this efficiency: compression, indexing, and summarization. The most important element to efficiency, however, is the thoughtful and careful consideration of how these techniques can best be used, and the thorough testing of these techniques before launching them into production.

For Additional Information:

- SAS Language, Reference, Version 6, First Edition
- Advanced SAS Programming Techniques and Efficiencies course notes
- SAS Companions to your particular operating system
- What’s New, on-line help (part of Display Manager, Windows 6.12 Release of SAS Software)

Special Thanks:

Jim Craig, Principal Systems Developer, SAS Institute Inc., Austin, TX

Jane Stroupe, Technical Training Specialist, SAS Institute Inc., Chicago, IL

Mary Rose Cascio, Technical Training Specialist, SAS Institute Inc., New York, NY

Mark Leventhal, Associate Applications Developer, SAS Institute Inc., New York, NY

SAS, SAS/AF, and SAS/EIS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.