Paper 3-28

Indexing and Compressing SAS® Data Sets: *How, Why and Why Not*

Andrew H. Karp
Sierra Information Services, Inc.
Sonoma, California USA
and
David Shamlin
SAS Institute, Inc.
Cary, NC USA

Abstract

This paper discusses the roles variable indexing and data set compression play in the SAS System. We discuss when these tools might be appropriate to use, and the situations where potential drawbacks from using them will probably outweigh their potential benefits. The paper also identifies key enhancements to both variable indexing and data set compression that were implemented in Version 8 of the SAS System, and which will remain available with the release of Version 9.

We are in the process of testing the benefits and drawbacks of applying these tools to various-sized data sets on several operating systems. We will present the results of our research at SUGI 28 and also make them available via the web at **www.SierraInformaiton.com**

Introduction

Many SAS® Software users are confronted with one or both of the following challenges, especially when working with "large" data sets. First, how can the amount of time it takes to select/retrieve subsets of observations from SAS data sets be reduced? Second, how can we reduce the size of SAS data sets without having to eliminate observations or variables? While these are separate issues, addressing different aspects of SAS software capabilities, we address both in this paper.

Data set indexes are used to speed the location of observations in a SAS data set. By creating indexes on variables in your data sets, it is possible to reduce the amount of time it takes the SAS System to find observations when you use a WHERE clause. Indexes also permit BY-group processing of data sets even if the observations in the data set are not sorted by the values of the BY variables you specify in a Data or Procedure step. And, they can be used in a Data Step where a match-merge is implemented using the KEY= option.
We discuss how indexing works in the SAS System, and the situations where you may want to consider adding indexes to key variables in your data sets. We also provide some insights into situations where indexing is *not* an optimal approach to selecting subsets of observations from your SAS data sets.

Data set compression tools are discussed in the second section of this paper. When applied properly, this feature of BASE SAS Software can reduce the size of your data sets. But, as we discuss below, you may save some storage space using SAS data set compression tools only to find that CPU utilization significantly increases for various SAS tasks (i.e., a PROC or Data Step). For this reason, we also explain the use of the LENGTH statement in the Data Step and how it might, depending on the unique characteristics of the data set to which it would be applied, be used in the Data Step as an alternative to SAS data set compression capabilities.

Data set indexing and compression capabilities were added in Release 6.07 of the SAS System, and were significantly enhanced with the release of Version 8. We will discuss what these changes are, and how you can take advantage of them when considering the use of indexes and/or data set compression in your work.

No significant changes to these tools were made in SAS 9, so you should expect similar results, all other factors being equal (i.e., operating system, hardware and data set size), with data set indexing and compression when migrating to SAS 9 at your site. So, this paper will hopefully be of use to SAS users who make the transition to SAS 9 soon, as well as those for whom the transition will be further off in the future.

We'd like to start with a caution before discussing these topics in detail. Indexing and compression tools are, in our opinion, important “weapons” in the SAS programmer's arsenal to, respectively, speed observation retrieval time or reduce data set size. But they are not “one size fits all” approaches to these challenges. Blindly adding indexes to all (or most) of the variables in your data set, or invoking the COMPRESS=YES SAS System option at the top of every program you write are not optimal solutions to reducing observation retrieval time or reducing the size of your data sets. We suggest that you take the time to test, tune and explore different approaches to these problems with your own data on your own operating systems/platforms rather than relying on “boilerplate” solutions that may not be appropriate for your unique data structures.

Depending on a number of factors we discuss below, there are many situations where the amount of CPU time required to create the index might exceed the storage savings obtained from using it. It is possible that even if you request data set compression, the resulting data set will be the same size (or larger) than if you did not request compression. (A note is written to the SAS log when this occurs.) Or, you may find that the CPU utilization requirements to process DATA and/or PROC steps against the compressed data set are higher than if you applied them to an uncompressed version of the same data set.

For these reasons, we suggest SAS programmers take the time to fully investigate how data set indexing and compression tools work in the SAS System. Hopefully, what we discuss here, as well as information available from other resources we reference at the end of this paper, will enable you to decide how, why (and, perhaps, why not) to utilize these features of the SAS System.

Data Set Indexing

An index is a physical file structure that gives the SAS System information on where to find observations with values of variables specified as key or index variables.

Without indexes, the SAS System tests/reads each observation in a data set, in the order they are physically stored in the data set, to find those that, for example, satisfy the conditions in a WHERE clause. This is called *a sequential read* of a data set and is the default method by which the SAS System extracts observations from your SAS data sets. Prior to Version 6, sequential reading was the only method by which the SAS System could access observations in your data sets. Version 8 introduced the ability to create indexes on SAS data sets and optimize this kind of processing. There are two types of indexes the SAS System supports: *simple* and *composite*. A *simple index* is based on the values of one variable, and has the same name as the index, or *key,* variable.. For example, a SAS programmer at a hospital might want to place a simple index on a variable called MRN, or medical record number, which uniquely identifies each patient to receive services at that facility. Conceptually such an index might look like

MRN	Observations
123-45-6788	1,5,6
456-78-9999	8,9
678-90-1234	4,2
890-123-456	3,7,10

for a (hypothetical) 10 observation data set. With this index in place, SAS does not need to evaluate every observation in the data set when a WHERE clause is used to retrieve observations. Suppose that all records for MRN equal to 890-123-456 were desired. Instead of reading all 10 observations in the data set, one at a time, to find the three of interest, SAS would use the index to locate observations 3, 7, and 10 return them without having to examine all 10 observations.

A *composite index* is composed of two or more variables which are concatenated (joined) to form a single value within the index. The name of the composite index must be given by the user when the index is created. (See example below.)

Here is a hypothetical example where a composite index might be useful. Suppose a SAS programmer worked in the marketing analysis department of a large hardware store chain. She routinely needs to extract observations for a SAS data set containing shipment from the company's warehouses to the individual stores. The extractions are based on both store number and the product category of the shipped products, and each record in the inventory shipment data set has multiple observations for both STORE_NUMBER *and* PRODUCT_TYPE. Here is a conceptualization of what a composite index, called COMBINATION (that is, the combination of STORE_NUMBER and PRODUCT_TYPE might look like.

COMBINATION	Observations
100LUMBER	1,5,6
101NAILS	2,9.12
101NAILS	3,4,11
100PAINT	7,8,13
100PAINT	10,14

This composite index could be used to find, say, Store 100's paint shipments without having to search each inventory shipment record sequentially.

There are three ways to create indexes for variables in your SAS data sets. First, you can use the **INDEX=** *SAS data set option* to create indexes when you build the data set in a Data Step. Second, if you are using **PROC SQL** to create a data set, you can use that procedure's **CREATE INDEX** statement to add indexes to your data sets. Third, the **INDEX CREATE** statement in **PROC DATASETS** can be used to create indexes on existing SAS data sets, and is therefore useful if you want to avoid having to execute a Data Step or **PROC SQL** task just to rebuild a data set to include an index. (The **INDEX DELETE** statement in **PROC DATASETS** will delete previously-created indexes.)

For indexes created for SAS data sets in the OS/390 (mainframe) operating system, the index is stored in the same physical location as the data set. (Please note, however, that indexes cannot be created for data sets stored on tape or other sequential media.) In other operating systems, the index is in a separate file, in the same data library, as the data set itself. Both the index and the data set to which it is associated are stored in the same SAS Library. Below is a screen capture of an index icon that a user running SAS in the Windows operating system will see when they explore the files and folder on their computer.

test4.sas7bndx
SAS System Data Set Index
11,841 KB

You may want to consider applying indexes to one or more of the variables in your data set if: a) the values of the variables are used repeatedly in WHERE clauses or BY statements, or used to retrieve observations using a match-merge between two data sets and b) the CPU savings realized from using the index across many queries, data steps and/or procedure steps is greater than the CPU required to build the index. For example, spending 60 seconds of CPU time to build an index that saves only 10 second of CPU time in a WHERE clause query that is used only one time is not an effective use of computing resources.

Indexes are also potentially warranted if the variable(s) upon which the index will be applied discriminate among the observations the in the data set. That is, the values of the indexed variable(s) make it easy to identify subsets of the observations. So, we need to keep in mind both the repetition value of the values of a variable to obtain desired subsets of observations *and* the distribution of the values themselves.

You can use BASE SAS Procedures such as **PROC FREQ** and/or **PROC UNIVARIATE** to assess the distribution of values of variables upon which you are considering applying indexes. **PROC UNIVARIATE's** new (to Version 8) **HISTOGRAM** option is often useful to obtain a graphical depiction of how the values of a variable are distributed.

According to SAS Technical Report TS-580, ***Indexing in the SAS System, Version 6,*** by Denise J. Moorman and Deanna Warner, if the candidate index variable has one value for more than about one-third of the observations, indexing is not appropriate. For subsets between 21% and 33% of the data set, an index *may* improve performance Subsets between 16%and 21% are likely to improve performance and variables which subset of 20% of less of the observation are very likely to improve performance.

There are two options you can specify when creating an index. The **UNIQUE** option tells the SAS system that each observation has a unique value for the index variable (if this is not the case, SAS will not create the index and you'll see an error message in your SASLOG). Using the **UNIQUE** option for index variables such as account numbers, customer identification numbers and other sorts of identifiers is a good idea if you need to extract individual observations from a data set based on the values of these variables.

The **NOMISS** option is applied when the user does not want missing values of the index variable included in the index itself. By default, the indices SAS creates includes information about which observations have missing values for the index variable.

In some analytic situations the presence of a missing value is informative and you might want to have SAS quickly locate all observations where the value of an index variable is missing. In those situations, the default operation of the SAS variable indexing facility is desirable.

For example, at an insurance company, a variable representing the date the claim was settled will have a missing value until settlement actually occurs. If an index is applied to this variable, the default action is to include in the index information for all observations with missing values for the claim settlement date. In some analytic situations we might conceptualize a missing value for claim settlement date as “informing” us that perhaps the file needs additional review, for example. If we wanted to regularly extract observations from this data set where the claims date is missing, then including missing values in the index is appropriate.

In other situations the missing values on an index variable are not interesting. If that’s the case with your data, then using the NOMISS option is probably something to consider.

We recommend you consider using indexes for “high repetition value” projects/data sets where many WHERE clauses or BY-statements will be used against the data set. For ad-hoc or infrequent queries, there is often little to no value from using CPU resources to build the index.

On what variables in your data sets should indexes be created? As we’ve mentioned before, the “repetition value” of the variable’s use to extract observations is a key consideration. What variables in your workplace are used most often to obtain subsets of observations, and how often are those subsets selected relative to how often the data set is updated? You’ll be able to identify which variables are good candidates for indexing by answering those questions with your own files and your own analytic/data processing requirements.

SAS decides whether to use an index to retrieve observations by estimating the number of observations that satisfy for the condition(s) in the query. In Version 6 SAS assumed that the values of the index variable where uniformly distributed between the largest and smallest values. In Version 8, SAS stores information about the distribution of index variable values in what are called cumulative percentiles or *centiles.* You can determine the values of your index’s centiles by using the **`CENTILES`** option in **`PROC DATASETS`**.

Storing information about the distribution of index variables in centiles allows the SAS System to make a more intelligent estimate of whether it is better to use the specified index or retrieve observations via a sequential read.

While we believe that in most cases the SAS System’s judgment about when to use or not use an index is probably the correct one to make, there may be times, after testing and tuning your applications, to override the SAS System’s decision to use or not use the index. Specifying **`OPTIONS MSGLEVEL= I;`** while testing your application will give you information in the SASLOG about whether the SAS System is using an index to retrieve observations.

If you want to force SAS to use an index, specify the new (to Version 8) **`IDXNAME=`***index name* as a data set option. Conversely, if you want to keep SAS from using an index, specify the **`IDXWHERE=NO`** data set option, which was also added to Version 8 of the SAS System.

Before forcing SAS to either use or not use an index, test and see if SAS makes the best choice for your situation. With the new centiles information added in Version 8, it is now even more likely that SAS’ ability to correctly determine when to use, or not use an index, has improved signicantly.

Conclusion: Data Set Indexing

SAS data set indexing capabilities can, in many situations, increase the speed with which you can extract observations from data sets using a **`WHERE`** clause, use **`BY`**-Group processing on unsorted data sets, and perform updates or match-merges using the **`KEY=`** data set option.

But, as we’ve pointed out, these benefits need to be balanced against the costs of applying indexing to variables in your data sets. These include increases in the data set’s size to accommodate the index portion, and the CPU required to build the index.

Several enhancements to indexing capabilities where added to the SAS System in Version 8. These include storing information about the actual distribution of the values of the index variable in CENTILES rather than assuming the values are uniformly distributed, as well as the IDXWHERE and IDXNAME options.

Data Set Compression

Data set compression tools were added in Release 6.07 of the SAS System, and enhanced significantly in Version 8. In some situations, these features of BASE SAS software will reduce the size of your datasets, at a cost of potential increased CPU utilization when executing data or procedure steps on the \compressed data set.

In Version 6, specifying the **COMPRESS = YES** data set option would result in having SAS compress repeating values or repeating blanks in character variables. In Version 8, this is the default data set compression method. The new **COMPRESS = BINARY** option in Version 8 instructs the SAS System to attempt to compress numeric variables. (In Version 8 users can specify either **COMPRESS = YES or COMPRESS=CHAR** to have the SAS System attempt compression on character variables.)

When compression is requested, the SAS System will, depending on the characteristics of each observation in the data set, remove repeating blanks, characters or numbers (depending on which compression approach you specify) and add a "tag" to each observation containing the information SAS needs to uncompress the observation when it is used in a subsequent SAS Procedure or Data Step.

The issue, then, is whether the overhead of adding the compression information to each observation is larger or smaller than the amount of storage space saved by specifying compression.

The SAS System reports the results of its efforts to apply the requested data set compression algorithm in the SASLOG. When using Version 6, many SAS users were surprised that when they requested compression, the SASLOG notes told them the resulting data set was larger than the uncompressed data set they started with! While this is counterintuitive, understanding how the SAS System compression tools works points to why this situation occurred in Version 6, and can still, in some situations, happen in Version 8.

For 32-byte hosts, 12 bytes are added to each observation for compression, and for 64-bit hosts, the 24 bytes of overhead are added per observation. Only when the resulting byte-length savings exceed either 12 or 24 bytes (depending on the host) will specifying data set compression actually reduce the size of the data set.

In Version 6, specifying **COMPRESS = YES** resulted in having the SAS System add the 12 (or 24) byte decompression "tag" to each observation, regardless of whether or not the variables in the data set could actually yield any sort of data set page size savings. The result was, in some cases, having a larger (or "bloated") data set than the one you started with.

In Version 8, the SAS System checks the byte lengths of the variable in the data set upon which compression has been requested. When character compression is requested. SAS will not add the decompression "tag" to the observations (thus increasing the size of the data set). When you request numeric compression, it is still possible to create a "bloated" data set that contains the decompression "tag" even if the SAS System can't find anything for the compression algorithm to "do" with your data set.

Checking the SASLOG will tell you if the SAS System compressed the data set or not, and if it did, the number of pages of data set memory that were saved by doing so.

At this point in reading our paper some readers may be asking why SAS does not, for the sake of making all data sets as small as possible, automatically compress them during data step processing. Or, you may be wondering if there is a global option you can specify to have the SAS System apply data set compression to ALL the data sets you create in a single SAS program.

Well, there is a SAS system option that will instruct SAS to apply compression algorithms to all data sets created after you specify it in a SAS program. Submitting the statement **OPTIONS COMPRESS=YES;** will result in the SAS System attempting to compress all temporary and permanent SAS data sets you create from that point forward in the program,

Using this system option without due consideration to the larger issues of the effects of data set compression is not, in our opinion, an effective approach to managing the size of your data sets. There are several reasons why we don't favor the "blind" use of this SAS System option, or to suggest SAS users automatically assume applying data set compression algorithms is an optimal approach to all data sets in all situations.

First, as we will discuss in further detail at SUGI 28, additional CPU resources are required to both create a compressed data set *and* to apply SAS tasks to the compressed data set. The SAS System must uncompress the data set every time it applies a data step or procedure step to it. This can increase CPU utilization as much as 50%, based on the results of our initial tests of running some BASE SAS Procedures on compressed versus uncompressed versions of the same data set.

Data set compression should, therefore, be used sparingly, and only after exploring whether or not it is appropriate for your data sets, programs and applications.

So, how can we make our data sets smaller without using SAS data set compression tools? Obviously, dropping unnecessary variables and deleting observations that are not required One effective way to eliminate unnecessary variables is to remember to drop index variables created when using a do-loop with an **ARRAY** statement.

Another very effective way to reduce the size of SAS data sets is to use the **LENGTH** Statement. This statement is used to declare the byte lengths of variables in the Program Data Vector, and can be used to adjust the byte-lengths of both numeric and character variables.

The default byte length of numeric variables is 8 bytes. As you can see from the following table, an 8 byte numeric variable can hold a VERY large integer! If you know your data well enough, you can assign an appropriate, but smaller byte length for the values of your numeric variables.

Largest Integers Which Can Be Stored in Various Byte Lengths in the SAS System

Byte Length	Windows And UNIX	OS/390 (Mainframe)
2	N/A	256
3	8,192	65,536
4	2,097,152	16,777,216
5	536,870,912	4,294,967,296
6	137,438,953,472	1,099,511,627,776
7	35,184,372,088,832	281,474,946,710,656
8	9,007,199,254,740,992	72,057,594,037,927,936

By appropriately adjusting the byte lengths of variables more observations can be placed in each page of data set memory which has two potential benefits for SAS Software users concerned about memory utilization and processing requirements.

First, the number of pages required to hold the data set is often reduced, resulting in a smaller data set, without compromising the values of the variables in the data set. Second, the number of input-output operations (called EXCP events in the OS/390 operating system) required to move the data set in to and out of the CPU is often reduced, thus contributing to improved processing time.

Before using the LENGTH statement to assign byte lengths to variables you need to find out the largest value of variable in the data set. Using **PROC MEANS** with the **MAX** statistics keyword, or **PROC UNIVARIATE,** with numeric variables will report the largest value of numeric variables in your data sets. When deciding the appropriate length of a numeric variable, keep these three tips in mind:

1) Don't reduce the lengths of SAS datetime variables, Values of these variables can become quite large, as they represent the number of seconds from January 1, 1960 to the date and time they represent.
2) Do not adjust the byte lengths of numeric variables that contain decimals. SAS needs all eight bytes to accurately store the decimal part of a numeric variable.
3) Make sure you consider potential increases in the values of the variables for which you are going to adjust the byte-lengths. For example a clinical researcher building a data set on OS/390 may be comfortable assigning a byte length of two (2) to a numeric variable representing the number of children a female patient has delivered, as it would be reasonable to assume that no one has 256 children. But, the researcher might want to assign a byte-length of three (3) to a variable representing patient weight, since it is possible in case someone enrolls in the study who weighs more than 256 pounds.

For character variables, you'll first need to find out the longest value (string) of the variable among all the observations in the data set. Let's say you're working with a data set where the character variable ADDRESS was initially assigned a byte length of 100. In looking at your data it seems that not all of these 100 bytes are actually being used, so the next time you create the data set you'd like to shorten the length of the variable, hoping in the process to make the data set smaller and without sacrificing the accuracy of the information contained in the variable

A useful took for this task is the **LENGTH** SAS Programming Function (not to be confused with the **LENGTH** Statement), which returns an integer giving the right-most non-blank character in the argument.

In this example, the value returned by the **LENGTH** Function is the number of bytes, of the 100 assigned, for each observation, that are actually being used to store information in th data set.

For example, the following assignment statement would determine how many bytes of the character variable ADDRESS were used by each observation in the data set.

```
Length_ADDR = LENGTH(ADDRESS);
```

The value of **LENGTH_ADDR** for each observation will have the value we need: of the 100 assigned bytes for this variable, how many of those bytes are actually being used for the address. Now, all we need to do is apply **PROC MEANS** or **PROC UNIVARIATE** to the newly-created variable to obtain the largest actual number of bytes used to store address in the variable.

As a result of this analysis you might find that the maximum number of bytes actually required for this variable, is less than the 100 bytes currently allocated. If so, you can adjust the byte-length of the variable ADDRESS then next time you build the data set.

Conclusion: Data Set Compression

SAS System tools for data set compression, first added in Version 6 of the SAS System can, depending on the characteristics of your data set, reduce the number of data set pages required to hold the data set in memory. But, CPU utilization will usually increase when a Data or Procedure Step is applied to the compressed data set.

In Version 6, specifying COMPRESS = YES as a data step option in a data step results in SAS attempting compression on the character variables in your data set. For SAS Versions 8 and 9, the (new to Version 8) COMPRESS = BINARY data step option will result in having SAS attempt compression of numeric variables in SAS data sets.

When users request data set compression SAS adds a “tag” of 12 or 24 bytes, depending on the host system, to each observation. The “tag” contains the information SAS needs to uncompress the observation when it is used in subsequent data or procedure steps.

In Version 8, SAS will determine if the potential savings (in bytes) for compression of character variables exceeds the size of the decompression “tag.” If it does not appear that savings will exceed the size of the “tag,” then SAS will not honor the user’s request for character variable compression. It is still possible, in both Versions 8 and 9 of the SAS System, if numeric compression is requested, to wind up with a larger data set than the one you started with. The SASLOG should be examined carefully to see if the data set compression request actually results in a smaller data set.

The **LENGTH** Statement is often an appropriate alternative to data set compression. It is used to specify the number of bytes SAS will use to store variables in your data sets. Appropriate use of the LENGTH Statement can result in smaller data set sizes without some of the drawbacks which result in applying data set compression.

CONCLUSION

In many situations, appropriate use of SAS data set indexing capabilities will result in reduced amount of time to locate observations in SAS data sets. Enhancements in Version 8 of SAS System software have improved the ability of indexes locate observations versus their performance in Version 6, when indexing was first added to the SAS System. But, applying indexes to your variables increases the size of your data set and do not always enhance observation retrieval.

Data set compression will, in many instances, reduce the size of your data sets. In Version 8, you can request that the SAS System attempt compression of either numeric or character variables (in Version 6, only character variable compression was available). Requesting compression may not always result in a smaller data set. And, CPU requirements when SAS tasks are applied to a compressed data set are often higher than when those same tasks are applied to the same, but uncompressed, data set.

Using the **LENGTH** Statement to adjust the widths of variables is often an effective alternative to applying data set compression.

SAS Software users may want to consider the benefits, as well as the potential drawbacks, of applying data set indexing and/or compression capabilities to their data sets

References

SAS Companion for the Microsoft Windows Environment, Version 8
SAS Companion for the OS/390 Operating System, Version 8
SAS Companion for the UNIX Operating System, Version 8
SAS Technical Report TS-580: Indexing in the SAS System available for download at: http://www.sas.com/service/techsup/tnote/tnote_index5.html

Contact Information

The authors can be contacted at:

Andrew H. Karp
Sierra Information Services, Inc.
19229 Sonoma Highway PMB 264
Sonoma, CA 95476 USA
707 996 7380 voice
SierraInfo@aol.com
www.SierraInformation.com

David Shamlin
SAS Institute, Inc.
Worldwide Headquarters
SAS Campus Drive
Cary, NC 27513 USA
919/677-8000 voice
David.Shamlin@sas.com
www.SAS.com

Acknowledgements

The authors are grateful for the technical review of this paper by Billy Clifford of SAS Institute in Austin, TX. Greg Dunbar, Anne Albright and Davetta Scales of SAS Institute in Cary, NC were instrumental in providing the run time performance analyses, and we gratefully acknowledge their assistance to us as well.

Copyright Reference