PharmaSUG 2016 - Paper QT23

SAS Techniques for Managing Large Datasets

Rucha Landge, Inventiv International Pharma Services Pvt. Ltd., Pune, India

ABSTRACT

As SAS programmers we often have to work with large data, having millions of rows, hundreds of columns. Usually it takes enormous time to process these datasets which can have an impact on delivery timelines. This problem triggers us to think that how can we reduce the time required for the execution and compress the size of the output data without losing any valuable information from the data. In this paper we will focus on techniques and concepts to compress the size of huge datasets and work with them efficiently to reduce the processing time. We will have a look at a few features available in the SAS® System like COMPRESS, INDEX, and BUFSIZE which provide you with ways of decreasing the amount of room needed to store these datasets and decrease the time in which observations are retrieved for processing. A few other very common dataset option that we regularly use and improve effective time are the LENGTH statement, DROP and KEEP statements and sub-setting options like WHERE and IF.

INTRODUCTION

Normally a SAS dataset is made up of observations and variables. A 'large' dataset implies numerous observations and variables resulting in an increase in overall size, but is a subjective term that primarily depends on user perception and on the available resources and storage space. Large datasets often occur during integration or pooling of data from various studies. It can take hours to read/retrieve data and use sub-setting conditions to generate a report successfully. In this paper we will discuss the challenges that we face while working on large data and focus on few techniques that can be used to work efficiently.

This paper will elaborate on options like COMPRESS, INDEX, and BUFSIZE. We will also discuss our very routinely used statements like drop, keep, if, where etc.

COMPRESS

A large SAS dataset can be made smaller by using dataset compression tools like the COMPRESS option to save the storage space. Reducing the size of the dataset will reduce the time SAS takes to read or access the data.

Compression is a process that reduces the number of bytes required to represent each observation in a dataset. Below mentioned are the options that can be used while working on compress.

The example discussed below of COMPRESS option is tested in SAS Drug Development® (SDD). We will explore ways of using the COMPRESS option to reduce the setup time of Input data sets, and to reduce the space required for storing these datasets in SDD, thus expediting the overall process of program development.

With favorable conditions and suitable data sets one can achieve up to 30-45% reduction in the run time, which directly affects a project's overall turnaround time in SDD.

Compressing a dataset effectively reduces the size of the dataset, and also significantly reduces the time needed to process very large data sets, as fewer input/output (I/O) operations are needed during processing.

SAS achieves this through two primary sub-options viz. COMPRESS=CHAR/YES and COMPRESS=BINARY.

Let us first discuss what COMPRESS=CHAR/YES does. Both CHAR and YES have the same effect when specified.

This particular SAS option compresses the size of the data set with run-length encoding. Run-length encoding compresses the data set by reducing repeated consecutive characters to two- or three-byte representations. What this means in programming terms is that it best compresses character variable intensive data sets.

We then move to the COMPRESS=BINARY option.

This particular option requests SAS to use Ross Data Compression, which combines run-length encoding and sliding-window compression to compress the dataset. It is most effective when used with numeric variable intensive data sets and data sets with large number of variables per record/observation.

To use the COMPRESS option is fairly simple. We can write it as a system option right at the beginning of the program or macro call.

As an example-

```
Options compress = binary;
```

RECOMMENDATIONS

WHEN TO AND WHEN NOT TO USE IT

Since there are great benefits to using this option, one might be tempted to suggest using this as a standard option in all code. But all good things come at a cost, and in this case the cost is CPU capacity and performance.

Using this option increases the CPU time for reading a dataset because of the overhead of uncompressing the record. The reason for this is that SAS cannot work with compressed data sets. It therefore takes longer to read and process compressed data sets.

So when is it recommended to compress data sets in code? When the size of the dataset is relatively large-approximately greater than 10 Mb. Data sets such as clinical laboratory values, questionnaires, and adverse events, which are inherently very large, are the type of data sets which are most positively affected. In such data sets, the increase in the overhead time decompressing the records is substantially offset by the gains achieved with the reduced size representation and fewer I/O operations required to read or to write to the data set during processing. The reduction in processing time thus achieved compensates for the increase in time spent on decompressing records.

ESTIMATING IF IT WILL BE HELPFUL IN YOUR CODE

Evaluating whether using the COMPRESS option is going to be useful and relevant in a particular project or program needs to be determined on a case-to-case basis. The key parameters to be evaluated are data set size, the nature of operations being performed on the data set and the computational power of the CPU. If the data set size is too small, compression might end up increasing the size of the data set in comparison to the original data set. In the scenario when I was working, data sets larger than 10 Mb gave positive results on compression. In the case of data sets which are large and also very densely populated, which is very rare with clinical data, compression might increase the data set size rather than decreasing it. If the operations performed in the program are very CPU-intensive, such as more than 50% of the code having only statistical procedures instead of sort, set, merge or update operations, then the time to run a program with compressed data sets might be more than the time required to run with normal data sets.

SOME EMPIRICAL FIGURES AND STATS (PERFORMED IN SAS DRUG DEVELOPMENT®)

Size of Project	Time taken with Compress(in hours)	Time taken without Compress(in hours)	Difference	Percentage Reduction(approx.)
8.5Gb	**1.43**	**2.55**	**1.12**	**44%**
44Gb	**6.6**	**10.14**	**3.54**	**35%**
11.5Gb	**3.3**	**4.97**	**1.67**	**34%**

Table 1. Indicative execution times of Projects

An important thing to keep in mind here is that the figures mentioned in Table 1 are for illustrative purposes only. The actual benefit or reduction in execution time may vary with the nature and properties of the data set(s) being tested.

BUFSIZE OPTION

Specifies the size of a permanent buffer page for an output SAS data set.

Syntax mentioned in the SAS documentation as follows:

BUFSIZE= n | nK | nM | nG | hexX | MAX

Syntax Description

n | nK | nM | nG

specifies the page size in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes).

The default is 0, which causes SAS to use the minimum optimal page size for the operating environment.

hexX

specifies the page size as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X.

MAX

sets the page size to the maximum possible number in your operating environment, up to the largest four-byte, signed integer, which is 231-1, or approximately 2 billion bytes.

The BUFSIZE= option is valid only for output data sets, that is, data sets named in the DATA statement of a DATA step or in the OUT= option of a SAS procedure.

The buffer size, or page size, determines the size of the input/output buffer SAS uses when transferring data during processing. A page is the number of bytes of data that SAS moves between external storage and memory in one logical I/O operation. Once specified, the buffer size is a permanent attribute of the data set, and the specified buffer size is used whenever the data set is processed. To change the buffer size, you must use a DATA step to copy the data set and specify a new buffer size or use the SAS default.

Using the BLKSIZE=, BUFNO=, and BUFSIZE= options can speed up execution time by reducing the number of times SAS has to read from or write to the storage medium. However, the improvement in execution time comes at the cost of increased memory consumption.

INDEX OPTION

An INDEX is a physical file structure that serves as an adjunct to a SAS data set. Like the index pages at the rear of a book, an INDEX can be used to locate observations based on the values of the variables upon which an INDEX has been created. These variables are often called KEY or INDEX variables. When an INDEX has been created the SAS System can use it to extract those observations for which the values of the index variable meet the conditions of interest without having to read each observation sequentially. Index allows the process to go directly to the observation of interest.

INDEX should be generally considered for variables which:
a) Have values which are frequently used in WHERE clauses, WHERE data set options, or sub-setting IF statements;
b) Less than about 25% of the observations have the same value of the index variable;
c) The values of the variables are approximately uniformly distributed. Small datasets will not give you desired efficient results.

Once the INDEX has been created, the SAS System will determine whether or not is should be used to extract observations from a data set. Programmers have no control over when an INDEX will be utilized. Adding the MSGLEVEL= (I) option to your program will result in a message in the SASLOG notifying you if the index was utilized.

HOW TO CREATE AN INDEX?

We can create index through several ways. We can use DATA step or the DATASETS or SQL procedures. There are other ways also. We will discuss these four methods.

CREATING INDEX BY DATA STATEMENT

We can declare the variable COUNTRY as index in DATA step either while giving input, i.e. creating dataset, or after the dataset got created. If I want to declare COUNTRY as index while giving inputs, our syntax will be:

```
data PATINFO (index=  COUNTRY));
   input SUBJID  age  ….;
   datalines;
1200  …
…
;
run;
```

CREATING INDEX BY DATASETS PROCEDURE

We can create index through DATASETS procedures also. The syntax is:

```
proc datasets nolist;
   modify PATINFO;
   index create COUNTRY;
run;
```

SOME FACTS ABOUT INDEX:

Here are some tips on using SAS index.

- We can feel the benefit of using SAS index only when the dataset is very large in size. If the page count of the dataset (can be obtained from CONTENTS procedure) is less than 5, use of index degrades the performance.
- As a rule of thumb, it can be said that indexing is most useful if the subset is less than 10% of the bigger dataset. Its performance becomes poor if the size is more than 50% of the original dataset.
- We can define a number of indexes on a SAS dataset
- Multiple index and composite index are not same.
- The indexing variable need not necessarily have unique values. Ideally it should be the most discriminating variable in commonly used queries
- Index cannot be created on compressed dataset.
- Before using index in a SAS dataset for merging datasets, we should be sure that there is enough space left in the location we are creating the merged dataset
- After we make any operation on a SAS dataset that is indexed, the resulting dataset no more remains indexed. So, it advisable to create index only at the final step, but not at the intermediate steps of creating the dataset.
- Index can also be used for match merging.

DROP AND KEEP OPTIONS OR DROP AND KEEP STATEMENTS

A SAS dataset may contain many variables that are either completely blank or not required for report generation. Dropping such variables does not make a huge difference when the size of the dataset is small. But, for large datasets, eliminating such unnecessary variables will definitely improve the efficiency of the program. In addition, it is not always necessary to have all variables present in the input dataset of a SAS Procedure. A common example is when a SAS dataset is used in a procedure to calculate frequency counts or basic summary statistics. It is recommended to exclude any variables that are not directly used in the procedure.

The DROP= and KEEP= options or DATA step statements DROP and KEEP can be used to select variables from a SAS dataset. Variables with all missing values can be determined during DATA Step processing.

Below tables will clearly show the difference and efficiency of drop/keep options or drop/keep statements.

KEEP and DROP DATA options	KEEP and DROP DATA step Statements
Used on input as well as output data	Always applied on output data
More efficient as they can be applied to a variable before being read from source data.	Less efficient as they apply only to the output dataset being written.
SAS will give an error in log if the mentioned variables in the drop or keep options are not present.	SAS will issue a warning if the mentioned variables are not present in the data.

LENGTH STATEMENT

A SAS dataset can have 2 types of attributes to a variable; either variable can be either Numeric or Character. The number of bytes required to store variables in a SAS dataset can be set or controlled by the LENGTH Statement.

Numeric variables in SAS are stored in default lengths of 8 bytes, whereas for Character variables one byte corresponds to one character. It is not necessary that the complete length of the variable is utilized by its values. This is commonly observed during the creation of numeric or flag variables whose only purpose is to hold either 0 or 1. In addition, answers to questionnaires, categorical numeric variables and some character variables holding comments/long values may not fit within the allocated length. It is important to understand the way SAS stores values in its variables and to identify which variables can store the same information in reduced or compressed length. Reducing the variable length can help in reducing both the amount of storage space and the number of I/O operations required to read and write a dataset.

1. **Numeric Variables:** SAS stores numeric data using floating point representation. An 8 byte numeric variable; however, can hold a very large integer value. It is important to remember that the fractional number should be assigned length of 8, if we specify lees length it can lead to truncation of data. If the value is a date value than 4 bytes of storage is sufficient for date value.

NOTE: SAS log will not have any error or warning in case truncation of data occurs.

2. **Character variables:** For character variables, it becomes essential to know the longest value or the longest potential value in a particular variable. The length of each character variable can be set to the number of characters in the longest expected value of the variable by using the LENGTH Statement.

Till now we have discussed how large datasets can be made smaller to improve program efficiency by saving processing time and storage space. The next section of the paper we will discuss a few programming efficiency tips by sub-setting, sorting and reducing disk space while working with these datasets.

PROGRAMMING EFFICIENCY: A FEW MORE CONSIDERATIONS

1. Most of the SAS datasets have flag variables present. As these will have a value of 0/1, we tend to create numeric variables. It is best advised to create character variable instead of numeric variable with a length of 1 byte.

2. It is suggested to test the SAS code first on smaller data sets by splitting the large SAS datasets into smaller manageable units. Logically breaking up the large datasets can assist in smoother and convenient data-handling. For example, a typical large laboratory dataset can be split into different categories such as Hematology, Biochemistry, and Urinalysis etc.

3. One of the most important tips that can help to efficiently utilize the available storage space is to delete the SAS datasets in the WORK library, other temporary space, or permanent space when they are no longer needed. This can be done using the procedure as shown in the example below:

```
/*To Clear work data*/

proc delete data = work._all_ kill;
run;
```

4. _NULL_ can be used as a dataset name in the DATA statement when a DATA step needs to be executed but without creating a SAS dataset. This is particularly useful when creating macro variables based on values obtained from an input SAS data set. This _NULL_ can also be done using PROC SQL as mentioned in the example. For example:

```
/*To count number of observations in a dataset*/

proc sql noprint;
            select count(*)
            into :OBSCOUNT
            from data;
quit;
```

5. When we have to merge 2 datasets, we get error in the log stating “By variables are not sorted in ascending sequence.” This increases our data steps and the datasets in work reducing storage space. It is advisable to use PROC SQL instead of data step merge which deals with the sort while merging.

6. If any 2 data sets contain the same variables and the variables possess same length and type, then use PROCAPPEND for appending instead of the SET statement. The APPEND procedure concatenates much faster than the SET statement, particularly when the BASE= data set is large, because the APPEND procedure does not process the observations from the BASE= data set.

7. Sub-setting or filtering any unwanted records from a dataset, helps us to reduce the size of the data thus saving storage space and improving the efficiency of the program. This purpose is solved by using either a where statement or if statement. If a large dataset requires sub setting based on a specific criteria or condition, then in most cases a WHERE statement proves to be more efficient and faster in performance than an IF statement.

CONCLUSION

The main objective of the paper is to make programmers aware of the hardships faced while working on large datasets. As mentioned in the paper there are techniques which can be used on large datasets to save time but the important fact is one has to consider the advantages and disadvantages to it. It is very important that we test these techniques on our data and decide the best approach to work. A piece of SAS code can only be termed "Efficient" if it can appropriately balance the code development time, processing time.

REFERENCES

- Andrew H. Karp and David Shamlin, SUGI Paper 3-28, Indexing and Compressing SAS® Data Sets: How, Why and Why Not
- Sunil Gupta, SAS Global Forum, Paper 213-2007, WHERE vs. IF Statements: Knowing the Difference in How and When to Apply
- SAS® Documentation.
- Working Efficiently with Large Datasets, Quanticate Whitepaper article.

ACKNOWLEDGMENTS

I would like to thank my colleagues and friends for their valuable feedback in improvising the paper. Last but not the least my family who has been my support system throughout.

CONTACT INFORMATION

Name: Rucha Sanjay Landge
Enterprise: Inventiv Health International Pharma Services Pvt. Ltd.
Address: Commerzone building no – 6 floor no – 6 Yerwada Pune.
City, State ZIP: Pune - 411006
Work Phone: 91- 020- 30569353
E-mail: rucha.landge@inventivhealth.com