SESUG Paper 34-2021

SAS® Performance Tuning Techniques

Kirk Paul Lafler, @sasNerd, Spring Valley, California

Abstract

The Base-SAS® software provides users with many choices for accessing, manipulating, analyzing, and processing data and results. Partly due to the power offered by the SAS software and the size of data sources, many application developers and end-users are in need of guidelines for more efficient use. This presentation highlights performance tuning techniques for SAS users to apply in their applications. Attendees learn DATA and PROC step language statements and options that can help conserve CPU, I/O, data storage, and memory resources while accomplishing tasks involving processing, sorting, grouping, joining (merging), and summarizing data.

Introduction

When developing SAS program code and/or applications, efficiency is not always given the attention it deserves, particularly in the early phases of development. System performance requirements can greatly affect the behavior an application exhibits. Active user participation is crucial to understanding application and performance requirements.

Attention should be given to each individual program function to assess performance criteria. Understanding user expectations (preferably during the early phases of the application development process) often results in a more efficient application. Consequently, the difficulty associated with improving efficiency as coding nears completion is often minimized. This paper highlights several areas where a program's performance can be improved when using SAS software.

Efficiency Objectives

Efficiency objectives are best achieved when implemented as early as possible, preferably during the design phase. But when this is not possible, for example when customizing or inheriting an application, efficiency and performance techniques can still be "applied" to obtain some degree of improvement. Efficiency and performance strategies can be classified into five areas: CPU Time, Data Storage, Elapsed Time, I/O, and Memory.

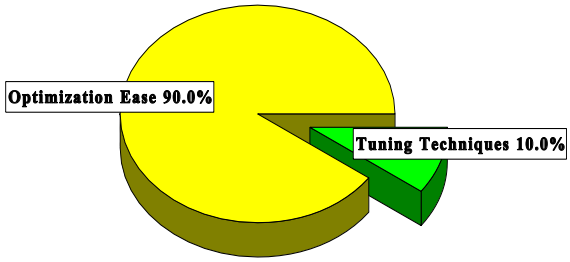
Jeffrey A. Polzin of SAS Institute Inc. shared his thoughts about measuring efficiency, *"CPU time and elapsed time are baseline measurements, since all the other measurements impact these in one way or another."* He continues by saying, *"... as one measurement is reduced or increased, it influences the others in varying degrees."*

The simplest of requests can fall prey to one or more efficiency violations, such as retaining unwanted datasets in work space, not subsetting early to eliminate undesirable observations, or reading wanted as well as unwanted variables. Much of an application's inefficiency can be avoided with better planning and knowing what works and what does not prior to beginning the coding process. Most people do not plan to fail - they just fail to plan. Fortunately, efficiency gains can be realized by following a few guidelines.

Guidelines to Hold Dear

The difference between an optimized software application (or process) versus one that is not can be dramatic. By adhering to practical guidelines, an application can achieve efficiency in direct relationship to economies of scale. Generally, as much as 90% of efficiency improvements can be gained quickly and with relative ease by applying simple strategies. But, the final 10% can often be a challenge. Consequently, you will need to be the judge as to whether your application has reached "relative" optimal efficiency while maintaining a virtual balance between time and cost.

Efficiency Scale



The following suggestions are not meant as an exhaustive list of all known efficiency techniques, but as a sampling of proven methods that can provide some measure of efficiency. Performance tuning techniques are presented for the following resource areas: CPU time, data storage, I/O, memory, and programming time. Selective coding examples are illustrated in ***Table 1***.

CPU Time

1) Use KEEP= or DROP= data set options to retain desired variables.
2) Use WHERE statements, WHERE= data set option, or WHERE clauses to subset SAS datasets.
3) Create and access SAS datasets rather than ASCII or EBCDIC raw data files.
4) Use IF-THEN / ELSE or SELECT-WHEN / OTHERWISE in the DATA step, or a Case expression in PROC SQL to conditionally process data.
5) Use the DATASETS procedure COPY statement to copy datasets opposed to DATA-SET constructs.
6) Use DATA step hash techniques to perform lookups and merges (or joins).
7) Turn off the Macro facility when not needed by specifying the NOMACRO system option.
8) Avoid unnecessary sorting - understand when a sort is needed.
9) Use procedures that support the CLASS statement to take advantage of group processing without sorting.
10) Use the Stored Program Facility for complex DATA steps.
11) CPU time and elapsed time can be reduced with the SASFILE statement.

Data Storage

1) Use KEEP= or DROP= data set options to retain desired variables.
2) Process only the variables you need which removes unwanted variables from the program data vector (PDV).
3) Use LENGTH statements to reduce the size of a variable.
4) Use data compression strategies to reduce the amount of storage used to store datasets.
5) Create character variables for data that won't be used for analytical purposes.
6) Shorten data by using informats and formats.
7) To allow a DATA step to be used without creating a data set, use a DATA _NULL_ statement.
8) More DASD space may be needed to hold a specified amount of data when the default physical BLKSIZE of 6KB is used.
9) When sufficient disk space is unavailable to perform a sort, the SORT procedure's TAGSORT option should be considered.
10) Remove unwanted SAS datasets with PROC DATASETS.

Input/Output (I/O)

1) Read only data that is needed from external data files.
2) Minimize the number of times a large data set is read by subsetting in a single DATA step.
3) Use KEEP= or DROP= data set options to retain only desired variables.
4) Use a WHERE statement, WHERE data set option or PROC SQL WHERE-clause to subset data.
5) Use data compression for large data sets.
6) Use the DATASETS procedure COPY statement to copy datasets with indexes.
7) Use the SQL procedure to consolidate steps.
8) Store data in SAS data sets, not external files to avoid excessive read processing.
9) Perform data subsets as early as possible to reduce the number of reads.
10) Use indexed data sets to improve access to data subsets.
11) Use the OUT= option with PROC SORT to reduce I/O operations.
12) The BUFNO= option can be specified to adjust the number of open page buffers when processing SAS data sets.

Memory

1) Read only data that is needed.
2) Process only the variables you need which removes unwanted variables from the program data vector (PDV).
3) Use WHERE statements, WHERE data set options, or WHERE clauses to subset data sets when possible.
4) Avoid storing SAS catalogs in memory because they consume large quantities of memory.
5) If using arrays, create them as _TEMPORARY_ to reduce memory requirements.
6) Increase the REGION size when the amount of available memory is insufficient.
7) Use the SORTSIZE= system option to limit the amount of memory that is available to sorting.
8) Use the SUMSIZE= system option to limit the amount of memory that is available to summarization procedures.
9) Use the MEMSIZE= system option to control memory usage with the SUMMARY procedure.
10) Use the MVARSIZE= system option to specify the maximum size of in-memory macro variable values.
11) Use memory-resident DATA step constructs like Hash objects to take advantage of available memory and memory speeds.

Programming Time

1) Use the SQL procedure for code simplification.
2) Use procedures whenever possible.
3) Document programs and routines with comments.
4) Utilize macros for redundant code.
5) Code for unknown data values.
6) Assign descriptive and meaningful variable names.
7) Store formats and labels with the SAS data sets that use them.
8) Use procedures such as PROC SQL when appropriate to consolidate the number of process steps.
9) Use the DATASETS procedure COPY statement to copy data sets with indexes.
10) Test program code using "complete" test data.
11) Assign redundant steps to function keys, particularly during debugging and tuning operations.

Survey Results

A survey was conducted to elicit responses from participants on efficiency and performance. The ***Efficiency and Performance Survey*** is illustrated in ***Table 2***. Analyzing the responses from each participant provided a better appreciation for what users and application developers look for as they apply efficiency methods and strategies.

The purpose for constructing the survey in the first place began in order to assess the general level of understanding that people have with various efficiency methods and techniques. What was found was quite interesting. The majority of users and application developers want their applications to be as efficient as possible. Many go to great lengths to implement sound strategies and techniques achieving splendid results. Unfortunately for others, a lack of familiarity with effective techniques often results in a situation where the application works, but may not realize its true potential.

Survey participants often indicated that efficiency and performance tuning is not only important, but essential to their application. Many cite response time as a critical objective and are always looking for ways to improve this benchmark. Charles Edwin Shipp of Shipp Consulting offers these comments on applying efficiency techniques, *"Efficiency shouldn't be considered as a one-time activity. It is best to treat it as a continuing process of reaching an optimal balance between competing resources and activities."*

Other universally accepted findings consist of using WHERE, LENGTH, CLASS and KEEP=/DROP= data set options to retain only those variables necessary to the application; avoiding unnecessary sorting; verify the efficiency of simple and/or composite indexes using the IDXNAME= or IDXWHERE= OPTION; using SAS functions; and constructing DATA _NULL_ steps as effective techniques to improve the efficiency of an application.

Techniques receiving "strong" (between “Sometimes” and “Always”), but not unanimous, support among survey participants include using system options to control resources; deleting unwanted WORK datasets; combining two or more steps into a single step; storing and using formats and informats; creating and using simple and composite indexes consisting of discriminating variables; using the APPEND procedure to concatenate two data sets; constructing IF-THEN/ELSE statements to improve conditional processing; and saving intermediate files, especially for large multi-step jobs.

Sunil Kumar Gupta of Gupta Programming offers these suggestions on assigning informats, formats, and labels, *"Informats, formats, and labels are stored with many of our important SAS datasets to minimize processing time. A reason for using this technique is that many popular procedures use stored formats and labels as they produce output, eliminating the need to assign them in each individual step. This provides added incentives and value for programmers and end-users, especially since reporting requirements are usually time critical."*

A very interesting approach being used more users to achieve greater efficiency is to use the SQL Pass-Through Facility to access data stored in one or more database environments. The advantage for users is that this forces all processing to be performed on the host database (e.g., Oracle, DB2, Access, etc.) which is where it should be. Also, the SAS software and its associated processing costs are automatically transferred to the host database for even greater efficiencies.

The techniques cited by survey participants as "Sometimes" being used to achieve efficiency include using DATA set options, using data compression, conserving memory by turning off unnecessary components and/or options, using the SQL procedure to consolidate and simplify multiple operations, using the Stored Program Facility, creating and using DATA and SQL views to control environments where duplication of data is rampant, and using the DATASETS procedure COPY statement for databases with one or more indexes.

Program Code Examples

The following code examples illustrate the application of a few popular CPU, data storage, I/O, memory, and programming efficiency techniques.

1. Using the KEEP= data set option instructs the SAS System to load only the specified variables into the program data vector (PDV), eliminating all other variables from being loaded.

```
data sql_users ;
  set sands.members
       (keep=name company phone user) ;
  if upcase(user) = 'SAS-SQL' ;
run ;
```

2. The CLASS statement provides the ability to perform by-group processing without the need for data to be sorted first in a separate step. Consequently, CPU time can be saved when data is not already in the desired order. The CLASS statement can be used in the MEANS and SUMMARY procedure.

```
proc means data=mortgage ;
  var prin  interest ;
  class state ;
run ;
```

3. By using IF-THEN/ELSE statements opposed to IF-THEN statements without the ELSE, the SAS System stops processing the conditional logic once a condition holds true for any observation.

```
data capitols ;
 set states ;
 if state='CA' then
       capitol='Sacramento' ;
 else if state='FL' then
       capitol='Tallahassee' ;
 else if state='TX' then
       capitol='Austin' ;
run ;
```

4. To subset data without running a DATA step, use a WHERE statement in a procedure. I/O and memory requirements may be better for it.

```
proc print data=sands.members n noobs ;
  where upcase(user) = 'SAS-SQL' ;
  title1 'SAS-SQL Programmers/Users' ;
run ;
```

5. To avoid using default lengths for variables in a SAS dataset, use the LENGTH statement. Significant space can be saved for numeric variables containing integers since the 8-byte default length is reduced to the specified size. Storage space can be reduced.

```
data _null_ ;
  length pageno rptdate 4 ;
  set  sales ;
  file report  header=h ;
  put @10 item   $20.
      @35 sales  comma6.2 ;
return ;
h:
  rptdate=today() ;  pageno + 1 ;
  put  @20 'Sales Report'
     /  @1 rptdate  mmddyy10.
     / @30 'Page ' pageno 4.  // ;
return ;
run ;
```

6. PROC SQL can be used to simplify and consolidate coding requirements and save programming time.

```
proc sql ;
  title1 'SAS-SQL Programmers/Users' ;
  select *
    from  sands.members
      where upcase(user) = 'SAS-SQL'
        order by name ;
quit ;
```

7. To improve data storage and I/O requirements, consider compressing large datasets.

```
data  sands.members (compress = yes) ;
   <  additional statements  >
run ;
```

8. To remove duplicate observations from a data set without the cost associated with using PROC SORT, PROC SUMMARY and a CLASS statement can be specified.

```
proc summary data=sands_members nway ;
  where upcase(user) = 'SAS-SQL' ;
  class name company phone user ;
  id name company phone user ;
  output out=SQL_Users_without_DupRecs
            (drop=_type_) ;
run ;
```

Table 1. Program Code Examples

Determining the Size of a Data Set

When developing SAS data sets, program code and/or applications, efficiency is not always given the attention it deserves, particularly in the early phases of development. Data sizes and system performance can greatly affect an application's behavior. To address issues related to how much disk space will be needed for the creation of data sets, program libraries and other application needs, SAS users can make a few calculations and/or learn how to access metadata content to obtain the information they want. This tip explores the various ways to determine, or estimate, the size of a data set – a question many users are curious about when discussing SAS performance and tuning techniques.

Using PROC SQL and DICTIONARY.TABLES

The SAS System collects valuable information (known as "metadata") about all known SAS libraries, data sets (tables), catalogs, indexes, macros, system options, views and a collection of other "read-only" tables called Dictionary tables and SASHELP views. One specific Dictionary table, TABLES, and its SASHELP view equivalent, VTABLE, contains details about a SAS session's data sets. In the following PROC SQL code, the specification of a PROC SQL SELECT-clause is illustrated to access the contents of four columns found in the TABLES Dictionary table, specifically LIBNAME, MEMNAME, MEMTYPE and FILESIZE to display the size of the CARS data set.

PROC SQL and Dictionary.TABLES:

```
PROC SQL ;
  TITLE 'Filesize for CARS Data Set' ;
  SELECT LIBNAME,
         MEMNAME,
         FILESIZE FORMAT=SIZEKMG.,
         FILESIZE FORMAT=SIZEK.
    FROM DICTIONARY.TABLES
      WHERE LIBNAME = 'SASHELP'
        AND MEMNAME = 'CARS'
        AND MEMTYPE = 'DATA' ;
QUIT ;
```

Results:

Filesize for CARS Data Set

Library Name	Member Name	Size of File	Size of File
SASHELP	CARS	192KB	192K

Analysis:

As shown in the results, above, the CARS data set filesize is 192KB. **Note:** When the SIZEKMG. format is specified in a format= option, SAS determines whether to apply **KB** for kilobytes, **MB** for megabytes, or **GB** for gigabytes; and divides the numeric filesize value by one of the following values:

KB	**1024**
MB	**1048576**
GB	**1073741824**

Using PROC PRINT and SASHELP.VTABLE

In the next example, the specification of a PROC PRINT is illustrated to access the contents of three columns found in the VTABLE SASHELP view, specifically LIBNAME, MEMNAME and FILESIZE to display the size of the CARS data set.

PROC PRINT and SASHELP.VTABLE:

```
PROC PRINT DATA=SASHELP.VTABLE NOOBS ;
  VAR LIBNAME MEMNAME FILESIZE ;
  WHERE LIBNAME = 'SASHELP'
    AND MEMNAME = 'CARS' ;
  FORMAT FILESIZE SIZEKMG. ;
  TITLE 'Filesize for SASHELP.CARS Data Set' ;
RUN ;
```

Results:

Filesize for SASHELP.CARS Data Set

libname	memname	filesize
SASHELP	CARS	192KB

Using DATA _NULL_, SASHELP.VEXTFL and CALL SYMPUTX

In the final example, a DATA _NULL_ is illustrated to access the contents of the VEXTFL SASHELP view with a FILENAME statement. An assignment statement is specified to calculate the FILESIZE value for the size of the CARS data set. The CALL SYMPUTX left justifies and trims the trailing blanks from the numeric FILESIZE value of 196608.

DATA _NULL_ and SASHELP.VEXTFL:

```
filename myfile 'C:\Program Files\SAS9.4\SASFoundation\9.4\\CORE\SASHELP\Cars.sas7bdat' ;
DATA _NULL_ ;
  SET SASHELP.VEXTFL (WHERE=(FILEREF='MYFILE')) ;
  /* Calculate the Filesize in MB */
  FILESIZE = FILESIZE / (1024 ** 2) ;
  CALL SYMPUTX ('FILESIZE',FILESIZE) ;
RUN ;
```

Results:

	fileref	xpath	xengine	modate	filesize	level	directory	exists	temporary
1	#LN00002	TERMINAL	TERMINAL	01JAN60:00:00:00	0	0	no	no	yes
2	#LN00004	C:\Users\KPL\Documents	DISK	28MAR17:03:51:46	0	0	yes	yes	yes
3	#LN00006	C:\Users\KPL\Desktop	DISK	27MAR17:20:45:09	0	0	yes	yes	yes
4	MYFILE	C:\Program Files\SAS9.4\SASFoundation\9.4\\CORE\SASHELP\Cars.sas7bdat	DISK	19JUN13:22:57:13	196608	0	no	yes	no
5	#LN00001	C:\Program Files\SAS9.4\SASFoundation\9.4\core\sasmsg	DISK	28APR14:03:21:58	0	1	yes	yes	yes
6	#LN00001	C:\Program Files\SAS9.4\SASFoundation\9.4\accelmva\sasmsg	DISK	28APR14:03:21:58	0	2	yes	yes	yes
7	#LN00001	C:\Program Files\SAS9.4\SASFoundation\9.4\access\sasmsg	DISK	28APR14:03:21:58	0	3	yes	yes	yes
8	#LN00001	C:\Program Files\SAS9.4\SASFoundation\9.4\cmp\sasmsg	DISK	28APR14:03:21:58	0	4	yes	yes	yes
9	#LN00001	C:\Program Files\SAS9.4\SASFoundation\9.4\spdsclient\sasmsg	DISK	28APR14:03:21:58	0	5	yes	yes	yes

Learning Necessary Techniques

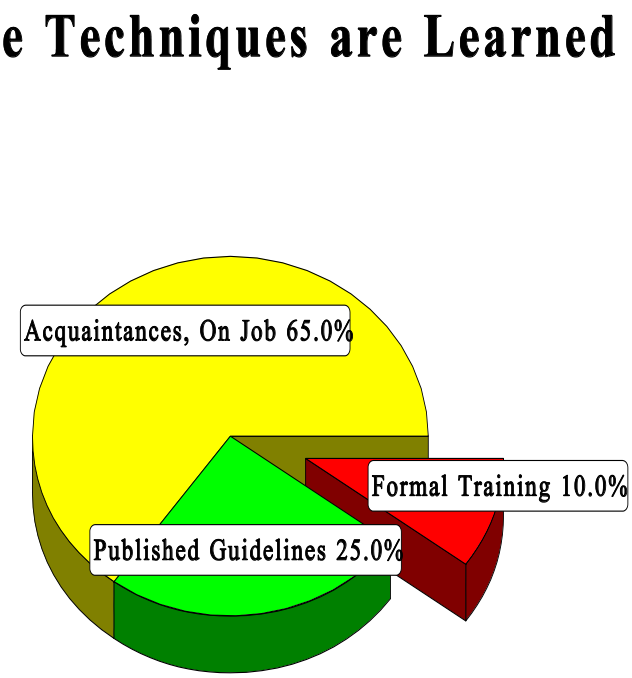
So how do people learn about efficiency techniques? A small number learn through formal training. Others find published guidelines (e.g., book(s), manuals, articles, etc.) on the subject. The majority indicated they learn techniques as a result of a combination of prior experiences, through acquaintances (e.g., User Groups), and/or on the job.

Any improvement is better than no improvement. Consequently, adhering to a practical set of guidelines can benefit significantly for many years to come. Survey responses revealed the following concerns:

1) An insufficient level of formal training exists on efficiency and performance.

2) A failure to plan in advance of the coding phase.

3) Insufficient time and inadequate budgets can often be attributed to ineffective planning and implementation of efficiency strategies.



Conclusion

The value of implementing efficiency and performance strategies into an application cannot be over-emphasized. Careful attention should be given to individual program functions, since one or more efficiency techniques can affect the scalability and/or behavior of your application or program. Efficiency techniques are learned in a variety of ways, from formal classroom instruction to published guidelines in books, manuals, articles, and videotapes. But the greatest value comes from the experience of others, word-of-mouth, and on the job.

References

Fournier, Roger, 1991. Practical Guide to Structured System Development and Maintenance. Yourdon Press Series. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 136-143.

Hardy, Jean E. (1992), "*Efficient SAS Software Programming: A Version 6 Update*," Proceedings of the Seventeenth Annual SAS Users Group International Conference, 207-212.

Lafler, Kirk Paul (2019). *PROC SQL: Beyond the Basics Using SAS, Third Edition*, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2017), *"Top Ten SAS® Performance Techniques",* Table Talk Presentation at the 2017 SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul and Mary Rosenbloom (2016), *"Best-Practice Programming Techniques Using SAS® Software",* Proceedings of the 2016 Pharmaceutical SAS Users Group (PharmaSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2016, 2015, 2014, 2012, 2009, 2007), "*Top Ten SAS Performance Tuning Techniques*," Proceedings of the 2016 Western Users of SAS Software (WUSS) Conference; Proceedings of the 2015 Western Users of SAS Software (WUSS) Conference; Proceedings of the 2014 MidWest SAS Users Group (MWSUG) Conference; Proceedings of the 2014 SouthEast SAS Users Group (SESUG) Conference; Proceedings of the 2012 MidWest SAS Users Group (MWSUG) Conference; Proceedings of the 2012 NorthEast SAS Users Group (NESUG) Conference; 2012 Kansas City SAS Users Group (KCASUG) Conference; 2009 Twin Cities Area SAS Users Group (TCASUG) Meeting; and the Proceedings of the 2007 Western Users of SAS Software (WUSS) Conference.

Lafler, Kirk Paul (2012), "*Essential SAS Coding Techniques for Gaining Efficiency*," Proceedings of the 2012 MidWest SAS Users Group (MWSUG) Conference.

Lafler, Kirk Paul (2000), "*Efficient SAS Programming Techniques*," Proceedings of the 2000 MidWest SAS Users Group (MWSUG) Conference.

Lafler, Kirk Paul (1985), "*Optimization Techniques for SAS Applications*," Proceedings of the Tenth Annual SAS Users Group International Conference, 530-532.

Polzin, Jeffrey A. (1994), "*DATA Step Efficiency and Performance*," Proceedings of the Nineteenth Annual SAS Users Group International Conference, 1574-1580.

Rosenbloom, Mary and Lafler, Kirk Paul (2013), "Best Practices: PUT More Errors and Warnings in My Log, Please!", Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Rosenbloom, Mary and Lafler, Kirk Paul (2012), "Best Practices: Clean House to Avoid Hangovers", Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Rosenbloom, Mary and Carpenter, Art (2015), "Are You a Control Freak? Control Your Programs – Don't Let Them Control You!", Proceedings of the 2015 Western Users of SAS Software (WUSS) Conference.

SAS Institute Inc. (1990), SAS Programming Tips: A Guide to Efficient SAS Processing, Cary, NC, USA.

Sherman, Paul and Russell Lavery (2005), "WHERE The Statement Is Not The Clause," Accessed: September 23, 2020. https://www.beoptimized.be/pdf/dwdb_where_the_statement.pdf.

Sloan, Stephen (2020), "Twenty Ways to Run Your SAS® Program Faster and Use Less Space," Proceedings of the 2020 Annual SAS Global Forum (SGF) Conference.

Valentine-Query, Paige (1991), "*Introduction to Efficient Programming Techniques*," Proceedings of the Sixteenth Annual SAS Users Group International Conference, 266-270.

Wilson, Steven A. (1994), "*Techniques for Efficiently Accessing and Managing Data*," Proceedings of the Nineteenth Annual SAS Users Group International Conference, 207-212.

Acknowledgments

The author thanks the 2021 SESUG Conference Committee, particularly Planning and Administration Chairs Louise Hadden and Richann Watson for accepting my abstract and paper; the 2021 SESUG Executive Committee for organizing and supporting this virtual conference event; SAS Institute Inc. for providing SAS users with wonderful software; and SAS users everywhere for being the nicest people anywhere!

Trademark Citations

Author Information

Kirk Paul Lafler is a lecturer and adjunct professor at San Diego State University; an advisor and adjunct professor at the University of California San Diego Extension; and teaches SAS, SQL, Python, R and Excel courses, seminars, workshops, and webinars to students, professionals and users around the world. Kirk has been a SAS user since 1979 and is the author of several books including, PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019) along with papers and articles on a variety of SAS topics. Kirk has also been selected as an Invited speaker, educator, keynote and section leader at SAS conferences; and is the recipient of 25 “Best” contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler
SAS® / SQL Consultant, Application Developer, Programmer, Data Analyst, Educator and Author
E-mail: KirkLafler@cs.com
LinkedIn: https://www.linkedin.com/in/KirkPaulLafler/
LinkedIn: https://www.linkedin.com/in/Order-of-Magnitude-Analytics/
Twitter: @sasNerd

EFFICIENCY AND PERFORMANCE SURVEY

Contact: _____ Organization: _____
Telephone: _____ Contact Date: _____

"I am conducting a survey for a regional SAS user group paper that I am writing. The topic of the paper is efficiency and how it relates to the SAS Software. Could you spare a few minutes to answer a few questions on this subject?"

1. Are efficiency and performance issues important in your environment? ❑ Yes ❑ No ❑ Sometimes

2. Have you received any training (formal or informal) in efficiency and performance strategies? ❑ Yes ❑ No

3. Do you take the time to resolve efficiency and performance issues in an application? ❑ Yes ❑ No ❑ Sometimes

4. Rate whether the following efficiency measurement categories have importance in your environment.
(Use the following rating scale: 1=Not Important, 2=Somewhat Important, 3=Very Important.)

   a. _____ CPU Time b. _____ Data Storage c. _____ Elapsed Time d. _____ I/O e. _____ Memory

5. In response to question #4, which measurement has the greatest importance in your environment? _____
Why?: _____

6. At what time(s) during the application development process do you consider using efficiency and performance techniques?

____	Requirements Definition Phase	____	Testing Phase
____	Analysis Phase	____	Implementation Phase
____	Design Phase	____	Maintenance/Enhancement Phase
____	Coding Phase		

7. Rate the following techniques and/or strategies that you have used in your environment to improve a program's/application's efficiency and/or performance? *(Use the following rating scale: 1=Never, 2=Sometimes, 3=Always.)*

_____ System Options such as BUFNO=, BUFOBS=, BUFSIZE= COMPRESS=, etc.
_____ DATA Step Options such as NOMISS or NOSTMTID.
_____ LENGTH or ATTRIBUTE Statements to reduce the size of numeric variables and storage space.
_____ Create numeric variables for the purpose of analysis; otherwise create character variables - less CPU intensive.
_____ KEEP / DROP statements or KEEP= / DROP= data set options to select only variables desired.
_____ Delete Unwanted Datasets in the WORK area.
_____ Combine Steps to minimize the number of DATA and/or PROC steps.
_____ Data Compression using the COMPRESS= data set option.
_____ Conserve on Memory (e.g., turning off NOMACRO, array processing)
_____ Formats and Informats to save CPU during complex logic assignments.
_____ Avoid unnecessary sorting using PROC SORT.
_____ Control sorting by combining two or more variables at a time when sorting is necessary.
_____ Subsetting IF statements to subset data sets.
_____ WHERE statements to subset data sets.
_____ Indexes to optimize the retrieval of data.
_____ Construct IF-THEN/ELSE statements to process condition(s) with greatest frequency first.
_____ Save intermediate files in multi-step applications.
_____ DATA Step Hash programming techniques.
_____ PROC APPEND (or PROC DATASETS with APPEND) versus SET statement to concatenate datasets.
_____ PROC SQL to consolidate multiple operations into one step.
_____ PROC SQL Pass-Through Facility to pass logic to target database for processing.
_____ Stored Program Facility to store SAS DATA steps in a compiled format.
_____ DATA Views and SQL Views to create "virtual" tables.
_____ SAS Functions to perform common tasks.
_____ DATASETS Procedure COPY statement to copy datasets with built-in indexes.
_____ DATA _NULL_ step to avoid creating a dataset when one is not needed but processing is.
_____ CLASS statement in procedures that support it to avoid having to sort data.
Other: _____

Thank you for participating in this survey!

Table 2. Efficiency and Performance Survey