Paper 1410-2017

Quickish Performance Techniques for Biggish Data

Ryan Kumpfmiller, Ben Murphy, Jaime Thompson and Nick Welke; Zencos Consulting

ABSTRACT

Getting speedy results from your SAS programs when working with bulky datasets is more than elegant coding techniques. There are several different approaches to improving performance when working with biggish data. While you can upgrade your hardware, this only helps run inefficient code and bloated tables quicker. So, you should also consider what results tuning your database and adjusting your SAS platform can bring. In this paper, we will review the various options to give you some ideas on things you can do better.

INTRODUCTION

Shortly after the internet kicked into gear, so did the term Big Data. This term applies to data that can barely be handled using traditional methods. However, there are still organizations grappling with data that is large or “biggish” as one customer stated. This paper provides some *quickish* techniques that users and administrators can use when working with *biggish* data.

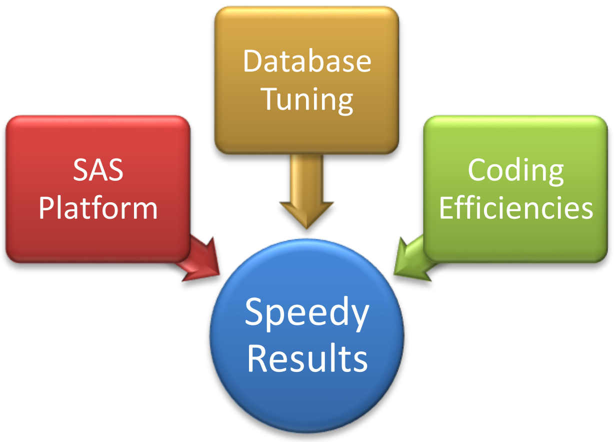
OVERVIEW

If you are working with large data, the end goal is to gain some value from it. It may be as simple as a query for sales last quarter or as complex as a creating sales forecast. The task still requires that you write a program to retrieve the data, shape and aggregate it, and then create the desired output. The entire focus for efficiency is about how well we can manipulate and move the data around.

When seeking programming efficiencies, Lafler suggested in his *Top Ten SAS® Performance Tuning Techniques* paper there were five areas that where we can influence performance:

- Total CPU time
- Total memory usage
- Total data storage needs
- Number of input/output (I/O) operations
- Total program time

Many new programmers and administrators might think that performance is related only to the hardware. As you will see in the remainder of this paper, it is a combination of actions.



MEASURING AND TROUBLESHOOTING PERFORMANCE ISSUES

As part of being able to improve data processing to be more efficient, it is important to be able to recognize when things are running smoothly and when they could be improved. By understanding ways to troubleshoot the technical performance of SAS programs, programmers will be more capable of identifying the opportunities for improvement and how those efficiencies can be achieved. Inspired by the helpful paper from Global Forum 2009, “Troubleshoot Your Performance Issues” [4], a few of these techniques will help programmers understand where there are opportunities for improvement.

USING FULLSTIMER

The SAS system option FULLSTIMER is a valuable resource for measuring performance issues. The FULLSTIMER output is helpful when trying to identify issues with CPU activity, memory activity, and disk input/output activity. The first metrics that are helpful for identifying issues with these types of activity are the timers for Real Time and CPU Time. When the CPU time is close to the real time (within 15%), the process is efficient, and the biggest source of improvement is to use a faster CPU. When there is a gap, especially a significant gap, between the CPU time and real time, the CPU is not getting used efficiently and the other metrics can help identify opportunities for improvement. For more details, and a more extensive explanation of FULLSTIMER, refer to the SAS Online Documentation for FULLSTIMER.

TUNING THE SAS PLATFORM

The SAS platform is the combination of hardware and software used to run programs. This topic discusses some ideas for maintaining servers and tuning your SAS software.

UPGRADING SERVERS

When upgrading or installing new SAS software, the SAS hardware recommendations should be followed. As many experienced administrators know, servers more than three years old should be evaluated. The newer servers offer not only speedier performance due to advancements in computer processing, but other benefits such as efficient power consumption and better management.

In the past, commodity hardware has become more popular to use. Commodity hardware generally refers to inexpensive standardized servers that are easy to purchase from a variety of vendors. These servers do not have a specific purpose and can be used as Windows or Linux based machines.

However, powerful servers are only a small part of the equation. If you install a powerful server without making other changes, then you only enable bloated databases and inefficient programs to run lighting quick!

CONTROLLING SYSTEM OPTIONS

In some situations, not having the performance options set can lead to problems if you’re working with large datasets or building a big OLAP cube. You can also improve the general performance of the SAS server by configuring and tuning options to meet your needs. You can update the options in files like the [SAS CONFIG]/Lev1/SASApp/sasv9_usermods.cfg. SAS Support has documentation on the order of precedence for SAS configuration files.

Note: Every operating system environment is different and may have different options. For example, Windows uses a different file system path than UNIX/Linux. This paper uses Linux x64 operating environment.

Setting the SAS WORK Library Location and Default Permissions

Having the ability to set the SAS WORK library to a different location is also beneficial if you have separate storage options. If you have the option of mounting additional space or hard disks for your SAS WORK library, you can set the library location using this parameter. The SAS WORK location is one of the most intense areas of activity in terms of input/output (I/O) throughput because practically every SAS process uses the WORK library for temporary data needs. Segmenting this activity on a dedicated mount point is ideal for maximum WORK library performance.

```
-work '/saswork'
```

In some cases, it may be necessary to set the default file system permissions for the SAS WORK library when information is written. Use the 'workperms' option to configure this.

```
-workperms 777
```

Check out SAS Support for more documentation on setting the WORK library location and setting the default WORK library file system permissions.

Setting Utility File Location

As it is beneficial to segment hard disk input/output (I/O) for the SAS WORK library location, the same goes for utility files generated by using SQL operations. This area of the file system can become very intensive during heavy data processing just like SAS WORK.

```
-utilloc '/sasutil'
```

Setting the Number of CPU Cores and Enabling Threading

Another good practice is to let SAS know how many CPU processors exist in your environment and if these processors support multi-threading. In our opinion, for large production environments it is better to limit how many CPU cores SAS uses, especially if threading is enabled, to prevent over-loading the system. It's a conservative approach but it's better to be safe and scale up from there. For example, we generally limit the number of cores to 25-50% of the actual number of cores. Otherwise as the load increases, SAS can potentially consume all available CPU cores and impact overall performance for other users.

```
-cpucount 4
-threads
```

Only certain SAS procedures take advantage of threading and parallel CPU activity. These changes typically improve the PROC SORT and PROC SQL steps.

Setting Memory Options for SAS

These are a few memory options which should be configured together. Use these options to limit the amount of memory SAS has available per process. A good practice we follow, which was passed on through SAS Tech Support troubleshooting, is to set the REALMEMSIZE, SUMSIZE, and SORTSIZE to approximately 75% of the total MEMSIZE.

```
-memsize 4G
-realmemsize 3G
-sumsize 3G
-sortsize 3G
```

Check out SAS Support for more documentation on configuring memory options. SAS Support also has good context on MEMSIZE and UTILLOC in this usage note.

Setting Buffer Size for Writing SAS Datasets

Another option is setting the buffer size for writing SAS datasets. You want this to be greater than or equal to the file system block size for where SAS datasets are written. The larger the buffer size for SAS datasets, the less SAS must read or write to the physical storage device. This comes at a cost of consuming more physical memory. Larger buffer sizes should be in multiples of the file system block size to minimize how many times SAS will access the file system. We typically go with values between 4K to 256K depending on the environment and how hardware is configured. Collaborating with the IT systems administrator is important.

```
-bufsize 64K
```

As always, check out SAS Support for more documentation on setting the buffer size. This technical support paper for increasing IO throughput – has a lot of good low-level detail.

A key to success for these performance options is to work directly with the IT systems administrator of the environment to ensure proper setting for each option.

CONSIDER OTHER SAS PRODUCTS

For larger organizations with huge volumes of data and data sources, other SAS products such as SAS GRID could be considered.

A SAS Grid environment is ideal when there are multiple users who want to access multiple servers. It intelligently uses the available computing resources -- specifically it routes jobs to the least "busy" server or node in the Grid on behalf of a user. This saves the user time, since they no longer need to concern themselves with where a program would execute the fastest. In additional, the overall computing power in a Grid can be managed via queues, with some queues being given a higher priority over others in terms of resource consumption.

Grid computing environments can also be scaled easily. Unlike HADOOP, a Grid environment allows for heterogeneous computing resources. In other words, each server or node in the Grid does not have to be identical. Servers can have different amounts of RAM and CPUs, which allows an organization to potentially leverage and combine both their legacy and new hardware as Grid nodes.

CODING TECHNIQUES

Many of us have experienced code execution taking forever to run. In most cases, we set those jobs to run at night so as not to waste a day of work. But, what if we could optimize performance and run the said jobs in a time period that’s more conducive to a day’s work? This topic explains some general coding efficiencies, explains how SAS stores data, and a technique for working with the database.

GENERAL CODING EFFICIENCIES

There are generally accepted methods of programming efficiency which have been expanded on by other papers. The following programming techniques are mentioned in Lafler’s paper, but most often cited by others. These techniques are about reducing your overall programming time.

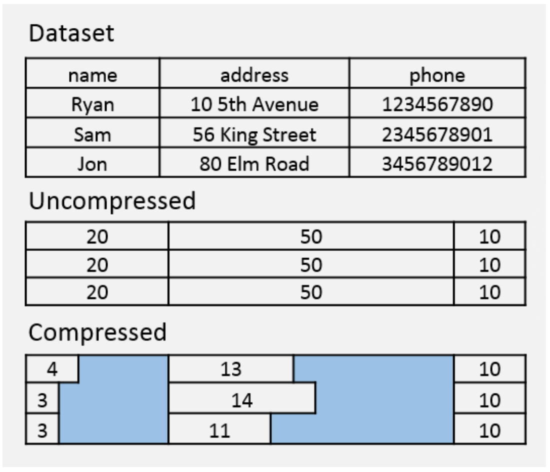
- Reduce the overall data set size by only keeping the needed variables and rows needed. If your query only requires the current year and a few variables, then limit the data pulled for the query.
- Use SAS procedures to analyze data. The SAS procedures are already tuned to be as efficient as possible. It’s easier to use a technique like PROC SQL or PROC SUMMARY to get answers.
- Sorting is an expensive operation. Avoid sorting data when it’s not required.
- Expand or update your knowledge of the SAS language. Consider adding techniques such as HASH programming and DataStep2 to your skillset.

This is not an exhaustive list, but merely a few ideas of easy ways to be more efficient.

USING COMPRESSION

With SAS datasets, sometimes you have data items that need to have a long length to incorporate large values but most of the values might only need a fraction of that length. When it comes to storing that dataset, SAS makes room for the total length of each value, regardless of how much space the actual data requires. This can be very inefficient since you are using up storage for blank space in your data file. Using the compress option on the dataset or program can alleviate this problem.

An example of how compression works in SAS is shown in the figure to the right. The dataset shown has character lengths of 20, 50, and 10 for the data columns “name”, “address”, and “phone”, respectively. The Uncompressed section shows how the data file would look if each of the values were to take up their entire allocated length, no matter what the values were. In the Compressed section, you can see that each of the values only take up the space of their value (for example Ryan = 4-character length). The blue space in the diagram represents the storage space that is saved by using compression and can be used to store other values.



name	address	phone
Ryan	10 5th Avenue	1234567890
Sam	56 King Street	2345678901
Jon	80 Elm Road	3456789012

As an example to see how much space can be saved, we are going to use the compress option on the SASHELP.BIRD dataset.

```
data birdNoCompress;
  set SASHELP.BIRD;
run;

data birdCompress(compress=YES);
  set SASHELP.BIRD;
run;
```

This dataset only has four data items, but one column, text, has a length of 1200 so that it can include long values even though most values only use around 20-50 characters. The code shown to the left is the code that we executed. After execution, we ran a PROC COMPARE on each dataset. The birdNoCompress dataset had a file size of 768KB and the birdCompress dataset was only 192KB, the compressed dataset was about 75% smaller than the original!

Now there are a few things to note about using compression before using the (compress=YES) option of the data step for every program:

- It can be set as on a per-dataset basis (see BIRD example above) or as a system option. The system option compresses every dataset that is created in your program.
- You can only compress numeric or character values in a dataset, not both. “compress=YES” and “compress=CHAR” compresses character values only; “compress=BINARY” does numeric values.
- Using compression decreases I/O and storage space when processing, which is good, but it also increases the program runtime, since the dataset must be uncompressed before being processed. For this reason, compression should be selectively used when working with larger datasets.

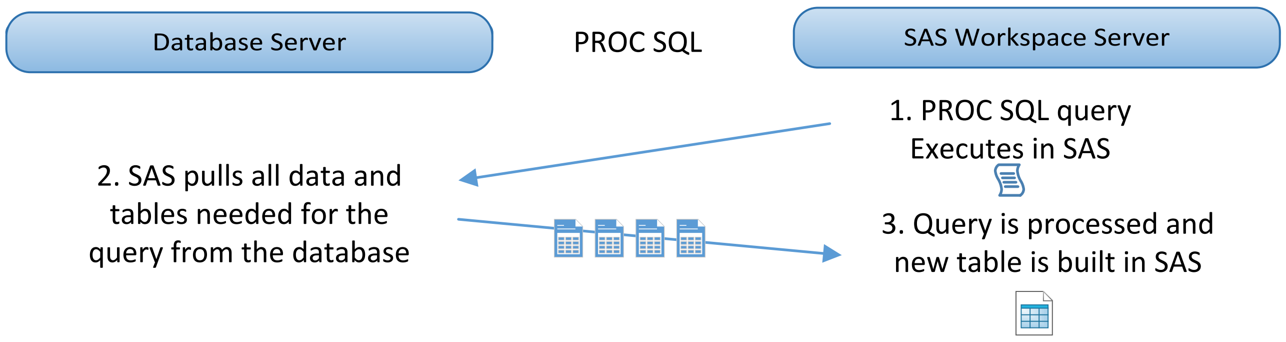
The REUSE= dataset or system option need to be set to YES in order to use the freed up space for new observations. If the option is not set (Default is NO), then new observations are simply appended to the data file.

QUERYING A DATABASE

Pulling data from relation databases is an integral part of SAS programming for many. Fortunately, the SAS/ACCESS software makes it easy for programmers to access their source databases and bring those tables into SAS. However, database environments can tend to be much larger and have more processing power than a SAS environment. When bringing in large or multiple tables from a database, the SAS workspace server may get overwhelmed and PROC SQL queries can take very long to execute. Using the SQL Procedure Pass-Through Facility can help with this problem. In one case, using the pass-through technique with PROC SQL reduced processing time from an average of 30 minutes to less than a minute.

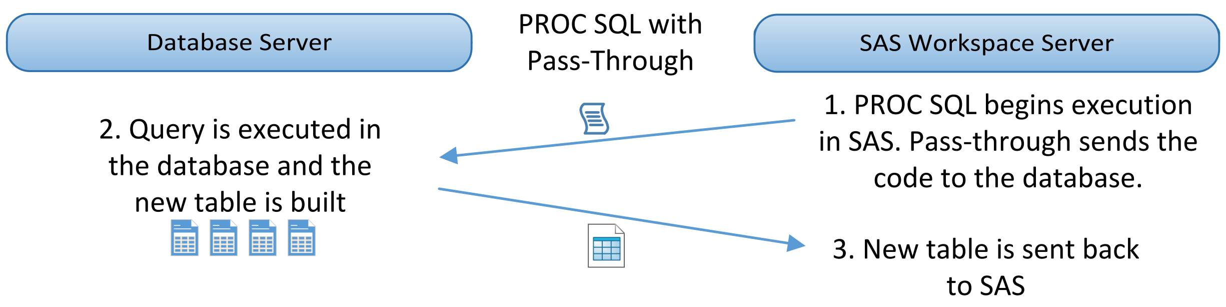
How SAS works with a Database

When using a LIBNAME statement to connect to a database and then executing a PROC SQL step to create a new table, SAS pulls all the necessary tables and data from the database. This process is shown in the following diagram. The space between the two servers would be the network. When you query in SAS and pull data from a database, you are bringing everything across the network first, before executing the query on the SAS Workspace Server:


Database Server
PROC SQL
SAS Workspace Server
1. PROC SQL query Executes in SAS
2. SAS pulls all data and tables needed for the query from the database
3. Query is processed and new table is built in SAS

Push the Work to the Database

For small tables, moving all the data across the network is not much of a problem, but with larger datasets this can become a problem if the network or SAS Workspace Server are not equipped to handle the amount of data and processing. The Pass-Through Facility works by executing everything on the database server, as shown in the following diagram:


Database Server
PROC SQL with Pass-Through
SAS Workspace Server
1. PROC SQL begins execution in SAS. Pass-through sends the code to the database.
2. Query is executed in the database and the new table is built
3. New table is sent back to SAS

Using the Pass-Through facility, the query is sent to the database server, executed on the database server, and then only the resulting single table is sent back across the network to SAS. This can save significant execution time if the database server is better equipped to handle the query compared to sending everything across the network and then executing it on the SAS Workspace Server.

Adding SQL Pass-Through Code

This example shows that the code used to pass queries to the database is simple and straightforward:

```
proc sql;
 connect to oracle (user=user_id password=pword);
 create table SALES_TOTALS as
  select * from connection to oracle
      (select tran_id,
             sales_amount
      from sales
      ) by oracle;
 disconnect from oracle;
quit;
```

In the PROC SQL statement above, the connect and disconnect statements are used to establish and terminate the connection with the database. We use the "connect" statement to establish a connection to our Oracle database and then query the SALES table in the database to create a SALE_TOTALS table, which we then use in our SAS program.

Note: The SQL query that executes in the database must follow that databases' syntax. Since we executed in an Oracle database, the code must follow Oracle's PL/SQL standards.

For more information about using the pass-through facility for your database, refer to the SAS documentation.

ADDING INDEXES

If your data is not stored in a database, then tuning the SAS datasets can be helpful. One method to improve the query time from SAS datasets is to add an index. Indexing gives a SAS dataset direct access to specified observations. Instead of reading every record, SAS can quickly find the requested observations. In some cases, the technique reduced the query time by 95%.

SAS Indexes come in two forms, simple and composite. A simple index is created on one key variable while a composite index is created on multiple variables. Indexes are typically used if you are filtering data frequently or joining columns.

Indexes can be created in PROC DATASETS, PROC SQL and the DATA statement. Here's an example using the DATASETS procedure. An index is added to the SALES_REP data set.

```
/* PROC DATASET EXAMPLE */
proc datasets library=performance:
modify SALES_REP;
index create sales_rep_id / unique;
run;
```

SAS indexes return observations in sorted order. This eliminates the need to sort the data set in subsequent data processing. Indexes may not be appropriate for all situations, so refer to the SAS documentation for more detailed information.

TUNING THE DATABASE

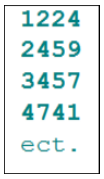
When extracting or querying a database, you want the data to be returned as quickly as possible. Manufacturer's know how their software performs best, it is important to start with their recommendations. This topic discusses a few techniques for working with databases.

GENERAL DATABASE RECOMMENDATIONS

There are several ways to ensure datasets are kept in good form. Here are some general recommendations that can be applied to databases.

- Use characters to save space

  Generally, it is better to store variables as character types to save storage. If you have a short numeric variable, such as Physician ID that might be a 4-digit number, it can still be stored as character value. Class variables are good candidates for being stored as character values. If we changed the variable type to character, we could save 4 bytes per observation, as compared to the standard SAS numeric type which normally uses 8 bytes. Saving just a small number of byes per observation adds up when dealing with big data. Remember to pad the character values with 0s if you want to sort them.

  ```
  1224
  2459
  3457
  4741
  ect.
  ```

- Manage character field sizes

  When adding a new table to the database ensure that the character field sizes are managed. Databases use VARCHAR which means the length of the field can expand as needed for the variable.

  From the earlier discussion about compression, a key takeaway was how SAS viewed the variable size. Even if the data did not require 50-character length, that amount was assigned. When SAS extracts data from the database, it does not understand a setting such as VARCHAR(50) and uses a full 50 character length.

- Add indexes to improve query times

  Tables that are frequently queried can benefit from having an index. Index the variables that are frequently used for joins or for filtering to improve speed.

While these techniques might seem obvious to an experienced database administrator (DBA), it may be less obvious to those new to database administration.

PARTITIONING TABLES

Another tip for improving database performance is to partition the tables. Not only does partitioning improve performance, it streamlines maintenance and reduces the cost of storing "biggish data". This probably sounds intriguing but you may be wondering, 'What is partitioning?' – to keep it simple, partitioning is dividing a data table into independent parts. A classic and useful example of partitioning would be to divide a dataset by time. That way, you have historical data in one table and recent data in a separate table. To run a query on recent data would occur at much greater speeds. You can define how the data is partitioned using the CREATE TABLE or ALTER TABLE commands.

Advantages of Partitioning Tables

During the scan operation, the optimizer accesses those partitions that satisfies a particular query. For instance, if we had a year's worth of accounting records broken up into quarters Jan-Mar in partition 1, Apr-Jun in partition 2, Jul-Sep in partition 3 and Oct-Dec in partition 4. When a query is issued that contains sales data for Jan-Mar, it scans the partition 1 only instead of all records, which would allow the query to complete much sooner.

```
   /* compute the date of the oldest data based on a 45 day save policy.  */
%let oldest_date=%sysfunc(putn(%sysfunc(intnx(day,%sysevalf("&end_loop"d),-
   45)),date9.));

/* compute the current date in the mysql date format */
%let current_date=%sysfunc(putn(%sysevalf("&end_loop"d),yymmddd10.));
%let current_date_param=%bquote('&current_date.');
%put &current_date_param;

/* delete the records and drop the partition for a date in the format of
   yyyy-mm-dd */
%macro drop_partition(date=);
  proc sql;
    connect to mysql;
    execute (alter table staging_test.main_staging truncate partition
        p&date.) by mysql;
    execute (alter table staging_test.main_staging drop partition p&date.)
 by mysql;
    disconnect from mysql;
  quit;
%mend;

/* compute the dates that need to be dropped */
proc sql;
  create table drop_dates as
  select distinct put(activity_dt, yymmddd10.) as date
  from dw_stg.main_staging
  where activity_dt < "&oldest_date"d
  order by date
  ;
quit;
/* truncate and drop the partitions associated with those dates */
data _null_;
     set drop_dates;
     call execute('%drop_partition(date=' || compress(date) || ')');
run;
/* create a partition for the newest day */
proc sql;
     connect to mysql;
     execute (
         alter table staging_test.main_staging
         add partition (
             partition p&current_date. values less
   than(to_days(&current_date_param.))
           )
     ) by mysql;
     disconnect from mysql;
quit;
```

ALTERNATE DATABASE LOAD TECHNIQUES

Some jobs may require a daily load of the data table. Many SAS programmers use the APPEND procedure or a similar technique. Sometimes you need to think outside of the box. For one batch job, we noticed it was taking nearly 30-45 minutes to load some of the data tables. We were loading several tables each morning and the job would expand into the working day.

A solution was to export the SAS dataset to a TXT file and then import the TXT file into the data table. This reduced each table load to less than 5 minutes. The speed increase comes from using the native database bulk load functions so it doesn't have to move the SAS data to MySQL. Here's a simplified example of the code we used:

```
options missing=0 NOQUOTELENMAX;

/*Extract the data to a text file*/

  data _null_;
   set work.Load2Database;
   file load2database.txt dlm="|" lrecl=32767 termstr=lf;
   put var1 var2 var3 var4;
  run;

/*Load the text file into the data table*/
 proc sql;
    %connect_mysql(database=staging_test);
    reset noprint;
      execute
      ("load data local infile %bquote(load2database.txt) into table
  main_stating FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n'
     (var1, var2, var3, var4)"
      ) by MYSQL;
    quit;
```

CONCLUSION

Part of what makes SAS a powerful tool is the flexibility. By enabling programmers to build an endless variety of solutions to tackle complicated problems, the power of SAS is a double-edged sword. With an endless set of options, SAS programmers can get into trouble with inefficient and overly complicated techniques. Each individual situation is unique, so finding the best approach to an efficient solution is up to the SAS programmer. This paper will hopefully help every SAS programmer consider the ways that accessing hardware, rethinking coding techniques, and applying database tuning strategies can help improve efficiency when working with biggish data.

REFERENCES

[1] Lafler, K. Top Ten SAS® Performance Tuning Techniques SAS Global Forum 2012. Available at: http://support.sas.com/resources/papers/proceedings12/357-2012.pdf

[2] Overton, S. "Change the Options to Get More Performance"; Zencos Blog. Available at: https://www.zencos.com/blog/sas-administration-change-the-options-to-get-more-performance/

[3] SAS 9.4 SQL Procedure User's Guide - https://support.sas.com/documentation/cdl/en/sqlproc/63043/PDF/default/sqlproc.pdf

[4] Williams, M., Easter, G., Bradsher, S.; Troubleshoot Your Performance Issues: SAS Technical Support Shows You How; Paper 333-2009 from SAS Global Forum 2009. Available at: http://support.sas.com/resources/papers/proceedings09/333-2009.pdf

ACKNOWLEDGMENTS

Thanks to the Zencos team members who provided content and examples for this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Zencos Consulting
www.zencos.com

Ryan Kumpfmiller
rkumpfmiller@zencos.com

Ben Murphy
bmurphy@zencos.com

Jaime Thompson
jthompson@zencos.com

Nick Welke
nwelke@zencos.com