Paper 3407-2015

You Say 'Day-ta', I Say 'Dat-a' – Measuring Coding Differences, Considering Integrity and Transferring Data Quickly

Mary Federico Katz and Jason R. Beene, Wells Fargo Bank

ABSTRACT

We all know there are multiple ways to use SAS® language components to generate the same values in datasets and output, e.g.: DATA Step vs PROC SQL, If-Then-Else vs Format table conversions, PROC MEANS vs PROC SQL summarizations, just to name a few. However, do you ever compare those different ways to determine which are the most efficient in terms of CPU and elapsed time usage? Do you ever consider the time a programmer takes to develop or maintain code? In addition to determining the most efficient syntax, do you validate your resulting datasets? Do you ever check data values that need to remain the same after being processed by multiple steps and make sure they really don't change? We will share some of our best practices that have proven to save computer and human resources. We will also explain some data validation and comparison techniques that ensure data quality and integrity. In comparing PROC SQL and a DATA step, we show a situation where multiple PROC SQLs can be replaced by a single DATA step. In our distributed computing environment, instead of exporting a SAS dataset from the SAS server to Excel on the client, we show how much quicker a SAS dataset can be transferred by using PROC DOWNLOAD on the server and then PROC EXPORT on the client to convert the SAS dataset to an Excel file.

INTRODUCTION

Jason and Mary met during the conversion of a SAS application from a distributed PC-SAS and UNIX analytic server environment to a distributed PC-SAS and Unix SAS Grid analytic server environment. Mary was assigned to assist Jason in the conversion of over 100 SAS programs to ensure they worked successfully in the new environment. Jason was very new to SAS programming, and these programs were written by a very accomplished SAS professional who had moved onto a new position. Jason is an expert in Microsoft tools, including SQL Server and SQL programming. Mary is a seasoned SAS developer who grew up programming in DATA Steps and PROCedures. The application is pretty typical of SAS client-server applications where the application uses PC-SAS on the 'client' PC and SAS on the UNIX analytic 'server'. The application, which runs daily, is composed of these steps:

1. Cron scheduler to run SAS programs to create the grid server's SAS datasets sourced from SQL Server and Teradata databases.
2. Windows scheduler to execute the macro-based PC-SAS programs that:
    - Create local, or 'client', SAS datasets from Microsoft Access databases and Excel workbooks from multiple network shared drives.
    - Transfer the PC SAS datasets to the UNIX platform.
    - Combine and process the new data with existing SAS datasets on UNIX.
    - Produce report data and export to Excel templates on network shared drives.

Through the conversion process, Mary exposed Jason to some benefits of SAS DATA Step/PROC programming, but at the same time, Jason shared his SQL knowledge with Mary, who had resisted SQL for many years! A multitude of SAS user groups' proceedings, Internet discussion forums and books that compare the advantages and disadvantages of traditional SAS programming techniques with those of SQL were referenced, including a couple of excellent and thorough comparisons listed in this paper's references (Dickstein, Pass, Davis, 2007 and Lafler, 2014). Given the new SAS Grid computing environment, Jason and Mary decided to perform comparisons of typical data processing functionality using different coding techniques. The results realized from the new environment are displayed and

discussed, however, the authors encourage you to test various techniques in your own SAS environments.

In addition to comparing coding techniques, data quality and integrity processes are introduced. When working with data sources, it is a best practice first to examine your variables to ensure they are accurate and second to understand how to process them. If certain data values must remain consistent through various processing steps, then you need to guarantee that they actually do remain consistent. As such, this paper's examples demonstrate simple data quality and integrity coding techniques as well as produce resource consumption comparisons for those techniques. Note that the SAS Dataset terms 'variables', 'observations' and 'datasets' are used in these examples. In SQL terminology, those terms are 'columns', 'rows' and 'tables', respectively.

For the examples in this paper, the following software was used:

On the PC: Windows 7, Base SAS 9.3., SAS/CONNECT, SAS/ACCESS and Microsoft Office 2010.

On the SAS Grid: Linux, Base SAS 9.3, SAS/CONNECT, SAS/ACCESS.

To keep examples simple, two test SAS datasets were created. One was approximately 8 GB in size and the other one was 65 MB. All DATA Step, PROCs and PROC SQL code and corresponding output was generated on the SAS Grid Server. Accessing data from multiple database management systems tables, such as Teradata, SQL Server, Oracle, etc., is not covered. Also, resource consumption in PROC SQL realized by implicit and explicit SQL conventions is not considered.

WHY COMPARE MULTIPLE WAYS TO PROCESS YOUR DATA?

Why is it wise to compare coding techniques and results from them? Comparing the techniques will better allow you to understand tradeoffs and impacts as you move forward; the following are some of the things you should consider. Some processes take a few lines of code in one method and many more in another. Think about future maintenance of your code: write your programs as simply and clearly as possible, and document the logic performed should you (or someone else) need to modify the code. Some methods take more CPU time, some take more elapsed time. Processing at different times of the day, week or month may impact performance due to network traffic and/or load on the SAS server. Always make sure that the same output is produced by different coding techniques that should produce identical results.

To track the processing performance of your coding techniques, explore SAS Options such as:

- FULLSTIMER: prints a list in the SAS log of the host-dependent resources that were used for each step and for the entire SAS session.
- MSGLEVEL=I: prints additional notes in the SAS log pertaining to index usage, merge processing, and sort utilities, along with standard notes, warnings, CEDA message, and error messages.
- Specific to SQL, stimer: records and displays query execution time.

UNDERSTAND YOUR SOURCE DATA

Understand the variables in your source – this should be the first thing you do with a new data source You need to understand variable attributes, validate data values, explore the interactions of the variables, etc. Knowing that kind of information is imperative when you transform values, derive new variables, retain certain variable values, assign formatted values, conditionally delete observations, handle missing values, and so on. Common coding techniques using PROC SQL and DATA/PROC steps are shown, along with the resources used to accomplish a few of these data quality and integrity tasks.

VARIABLE ATTRIBUTES – ARE THEY AS EXPECTED?

Once you read a raw data source into a SAS dataset or are given an existing SAS dataset, PROC CONTENTS or PROC DATASETS (with the CONTENTS statement) can be used to list information about your dataset (e.g., number of observations, number of variables, creation and modification dates, etc.) and the variables' names and their attributes. You should compare these values with what is expected

from your upstream data source. Here is basic code PROC CONTENTS that can produce output for examining variable attributes:

```
proc contents data=bigdata.sgf;
run;
```

Output 1 shows an example of partial PROC CONTENTS output.

```
Data Set Name        BIGDATA.SGF                                  Observations          11700000
Member Type          DATA                                         Variables             260
< additional information deleted for demonstration purposes >

                          Alphabetic List of Variables and Attributes

 # Variable                    Type Len Format       Informat     Label

24 ADDRESS                     Char  30 $30.         $30.         Address
25 CITY                        Char  21 $21.         $21.         City
23 FULL_NAME                   Char 100 $100.        $100.        Full Name
…etc.
```

Output 1. Sample Output from a PROC CONTENTS

There are a couple ways using PROC SQL to get similar variable (i.e., 'column') attribute information by reading SAS 'Dictionary Tables'. These read-only tables contain information about the libraries, tables, options, and much more in your current SAS Session. (Eberhardt and Brill, 2006, provide a thorough explanation of Dictionary Tables.) The same variable attribute information can be reported from the 'Columns' Dictionary table.

This first SQL coding method simply lists the variables and their attributes in the SAS log:

```
proc sql;
   describe table bigdata.sgf;
quit;
```

Output 2 shows partial results from DESCRIBE TABLE.

SAS LOG:

```
create table BIGDATA.SGF( compress=CHAR  bufsize=131072 )
  (
   FULL_NAME char(100) format=$100. informat=$100. label='Full Name',
   ADDRESS char(30) format=$30. informat=$30. label='Address',
   ….etc.
```

Output 2. Example of results from PROC SQL's DESCRIBE TABLE displayed in the SAS Log Window

This second SQL method produces a report with the same information:

```
proc sql;
   select name, type, length, format, informat, label
   from dictionary.columns
   where upcase(libname)='BIGDATA' and upcase(memname)='SGF';
quit;
```

Output 13 shows partial results from a SQL query against Dictionary Tables.

```
                                Column    Column   Column  Column     Column
Column Name                     Type      Length   Format  Informat   Label
-------------------------------------------------------------------------------------------
FULL_NAME                       char         100    $100.    $100.     Full Name
ADDRESS                         char          30     $30.     $30.     Address
 …etc.
```

Output 3. Example of results from a PROC SQL Query displayed in the SAS Output Window

Table 1 displays the resource statistics representative in a few repeated comparisons of ways to get variable attribute information.

FULLSTIMER Statistics	Description of FULLSTIMER Statistics	PROC CONTENTS	PROC SQL/ Describe	PROC SQL/ Dictionary.columns
real time	Clock time spent to process SAS job	0.02 seconds	0.00 seconds	13.25 seconds
user CPU time	CPU time spent in user program	0.00 seconds	0.00 seconds	0.66 seconds
system CPU time	CPU time spent for system overhead tasks that support SAS code	0.00 seconds	0.00 seconds	.88 seconds
memory	Amount of memory to run a step	1213.42k	776.37k	2701.73k
OS Memory	Largest amount of memory available to SAS during the step	12076.00k	11816.00k	12184.00k
….remaining statistics not included for our demonstration ….				

Table 1. Typical resource statistics for ways to report Dataset variables' attributes

Even though PROC SQL with Dictionary.columns used more resources overall, the consumption is minimal. The technique to use could be determined by your personal output preference.

CATEGORICAL DATA – CHECK VALUES FROM FREQUENCY DISTRIBUTIONS TO VALIDATE DATA VALUES

PROC FREQ and PROC SQL Group By are common methods to generate frequency distributions for categorical, or 'group by', data. The distributions show the counts of unique values for categorical variables. It is a best practice to examine those values to ensure quality. Are the results as expected? Are there missing values that will need special handling? This example is a simple one-way distribution of the categorical variable, status_cd. Both procedures produce the same counts for the unique values and produce similarly formatted reports, but consider the code required, the computer resources used in each and the output values produced. Here is the code for both techniques:

```
proc freq data=bigdata.sgf;
   table status_cd / nocum nopercent missing;
run;

proc sql;
   select status_cd, count(*) as Frequency
   from bigdata.sgf
   group by status_cd;
quit;
```

Table 2 displays the resource statistics from the one-way frequency distributions.

FULLSTIMER Statistics	PROC FREQ	PROC SQL
real time	25.86 second	28.01 seconds
user CPU time	19.35 seconds	22.70 seconds
system CPU time	1.74 seconds	2.16 seconds
Memory	1134.15k	266998.00k
OS Memory	12076.00k	276312.00k

Table 2. Example of resource comparison for one-way frequency distributions

In repeated executions, the real time, user CPU time and system CPU time were very similar to the values shown in Table 2. However, note the memory statistic: PROC FREQ always used much less memory because PROC FREQ has been optimized for such computations. Once you decide which coding technique you prefer, make sure your variables' unique values make sense!

A few other best practices for validating categorical data include generating frequency distributions from the interaction of multiple variables, determining how to handle missing values and applying 'integrity constraints' on some or all of your variables' values.

NUMERICAL DATA – CREATE DESCRIPTIVE STATISTICS TO CHECK FOR ACCURACY

PROC MEANS and PROC SQL Group By with statistical functions are common methods to generate descriptive statistics for numerical data that represent measures. This example has only one 'classification', or 'group by', variable (i.e.,state) for which descriptive statistics are computed. Here is the code required for the two methods:

```
proc means data=bigdata.sgf
   (where=(state in ('CA','OR','WA','HI','AK','NV','AZ','CO','ID','UT')))
   sum mean std min max;
   class state;
   var sale_amt;
   format sale_amt comma12.2;
run;

proc sql;
   select              state,
      COUNT(*) as      N,
      SUM(sale_amt) as Sum format=12.2,
      AVG(sale_amt) as Mean format=12.2,
      STD(sale_amt) as Std_dev format=12.2,
      MIN(sale_amt) as Minimum format=12.2,
      MAX(sale_amt) as Maximum format=12.2
   from bigdata.sgf
   where state in ('CA','OR','WA','HI','AK','NV','AZ','CO','ID','UT')
   group by state;
quit;
```

Table 3 displays the resource statistics for generating descriptive statistics for a single numeric variable and one classification variable.

FULLSTIMER Statistics	PROC MEANS	PROC SQL
real time	28.24 seconds	29.31 seconds
user CPU time	19.80 seconds	20.47 seconds
system CPU time	2.52 seconds	2.91 seconds
Memory	9264.85k	105256.29k
OS Memory	19024.00k	114268.00k

Table 3. Example of resource comparison of descriptive statistics on one numeric variable

As seen in our other techniques, the real time, user CPU time and system CPU time were very similar to the values shown in Table 4. However, again the memory used in PROC MEANS is much less than PROC SQL. As with PROC FREQ, PROC MEANS has also been optimized for producing statistical values. As for human resources in this example, the syntax in PROC MEANS is simpler than PROC SQL. If additional analysis variables are required, they can simply be added to the VAR statement of PROC MEANS. PROC SQL requires multiple summary function statements for each analysis variable. In tests of two analysis variables and one classification variable, similar resource consumption measures resulted: real time, user CPU time and system CPU time were very similar, but memory is much less with PROC MEANS.

From a data quality perspective, examine the descriptive statistics generated and make sure they make sense for your data. You will determine your preferred technique to generate those statistics with experimentation and experience.

CREATE NEW AND TRANSFORM EXISTING VARIABLES

Now that you have an understanding of your important variables in your source data, it may be necessary to create new variables, transform existing variables, eliminate observations, create new datasets, etc. SAS has numerous ways to do these operations. For comparison, the following simple coding techniques create new variables and transform existing ones. Although these operations can be performed on the source dataset, new datasets are created to maintain integrity of that source. Overwriting a dataset can be very dangerous! If you do so, examine your SAS Log for any errors or warnings that could result from overwriting.

CREATE NEW VARIABLES

Three ways to create a new variable in a dataset are compared: DATA Step's IF/THEN/ELSE, PROC FORMAT/DATA Step and PROC SQL with CASE logic. In this example, invalid values of the variable STATE were realized from a one-way frequency distribution. A new variable, NEW_STATE, will be created to contain either the valid value of STATE, or an indicator of the type of invalid data: 'Missing', 'Numeric' or 'Unknown'. Here is the SAS code for these three methods:

```
   data states_ifthenelse;              /* (1) DATA Step's IF-THEN-ELSE     */
      length new_state $7;              /*     statements                   */
      set bigdata.sgf;
      new_state = state;
      /* Clean up invalid states */
      if state = '  ' then                     new_state='Missing';
      else if state in ('zz','. ','XX') then  new_state='Unknown';
      else if '00' le state le '99' then      new_state='Numeric';
   run;

   proc format;                          /* (2a) Translation table, $clean, */
      value $clean                       /*      defined in PROC FORMAT     */
      '  '           = 'Missing'
      'zz','. ','XX' = 'Unknown'
      '00'-'99'      = 'Numeric';
   run;

data states_format;                      /* (2b) $clean is referenced and   */
   length new_state $7;                  /*      applied in DATA Step's     */
      set bigdata.sgf;                   /*      PUT function               */
      new_state = put(state,$clean.);
   run;

   proc sql;                             /* (3) PROC SQL with CASE logic   */
      create table states_case as
      select *,
      case
          when state= '  ' then                 'Missing'
          when state in ('zz','. ','XX') then   'Unknown'
          when '00' le state le '99' then       'Numeric'
          else state
      end as new_state
      from bigdata.sgf;
   quit;
```

Table 4 displays the resource statistics for creating a new variable.

FULLSTIMER Statistics	DATA Step IF-THEN-ELSE	PROC FORMAT and DATA Step*	PROC SQL CASE Logic
real time	1:27.58	1:29.83	1:26.64
user CPU time	1:08.7	1:11.66	1:10.83
system CPU time	8.74 seconds	10.80 seconds	9.04 seconds
memory	1636.57k	1631.43k	2426.31k
OS Memory	12588.00k	12332.00k	13360.00k
* Note: computer resources for execution of PROC FORMAT not included since measures were minimal.			

Table 4. Example of resource comparison for one new variable being created

The three ways to create a new variable based on an existing variable in the source dataset seemed to all use similar computer resources. In this case, program maintenance may be easier with the PROC FORMAT/DATA STEP technique.

TRANSFORM EXISTING VARIABLES

At times, data values for existing variables need to be replaced. The same three techniques used to create a new variables can be used to transform or update an existing variable as shown in this code:

```
data status_ifthenelse;              /* (1) DATA Step's IF-THEN-ELSE    */
   set bigdata.sgf;                  /*     statements                  */
   if      status_cd = ' ' then  status_cd='Z';
   else if status_cd = 'A' then  status_cd='X';
   else if status_cd = 'C' then  status_cd='Y';
   else                          status_cd='?';
run;

proc format;                         /* (2a) Translation table, $status, */
   value $status                     /*      defined in PROC FORMAT      */
         'A'  = 'X'
         'C'  = 'Y'
         ' '  = 'Z'
         other='?';
run;
                                     /* (2b) $status is referenced and   */
data status_format;                  /*      applied in DATA Step's PUT  */
   set bigdata.sgf;                  /*      statement.                  */
   status_cd = put(status_cd,$status.);
run;

proc sql;                            /* (3) PROC SQL with CASE logic     */
   create table status_case as
      select *
      from bigdata.sgf;
   update  status_case
      set status_cd =
         case
            when status_cd= 'A' then   'X'
            when status_cd= 'C' then   'Y'
            when status_cd= ' ' then   'Z'
          else '?'
       end;
quit;
```

Table 5 displays the resource statistics for updating an existing variable.

FULLSTIMER Statistics	DATA Step IF-THEN-ELSE	PROC FORMAT and DATA Step*	PROC SQL CASE
real time	1:26.42	1:26.51	4:43.34
user CPU time	1:08.38	1:09.04	4:11.13
system CPU time	10.65 seconds	10.06 seconds	19.19 seconds
memory	1629.48k	1631.53k	1654.68k
OS Memory	12332.00k	12332.00k	12332.00k
Note: computer resources for execution of PROC FORMAT not included since measures were minimal.			

Table 5. Example of resource comparison when existing variable recoded

In the PROC SQL CASE example, the sometimes arduous process of dropping the original variable and adding a new one on the new dataset could have been used. Instead, the UPDATE statement is used for demonstration. The combined time of the creating a new table and using the UPDATE statement with PROC SQL was more than that of the Data Step techniques. Memory usage was about the same in all three techniques.

FIND THE BEST CODING TECHNIQUE TO SOLVE YOUR PROBLEM – USE MINIMAL RESOURCES AND ENSURE INTEGRITY

Here is an example of two ways to create a new dataset based on binary variables in another dataset. Both ways produce the same results but require differing amounts of human and computer resources. Our source data contain unique observations representing customer problems where service representatives research the causes and determine a solution. Each observation has a Y(es)/N(o) indicator for any phase a problem goes through until is it resolved. The problem's resolution phases include: opened, reviewed, completed, closed, reopened, etc. The opened, completed and reopened phases need to be tracked with other input variables for further analysis. The corresponding Y/N binary variables are simply named: 'opened', 'reopened' and 'completed'. There are various ways to capture those variables' values for analysis; the way chosen was to create a new dataset containing a unique observation for each open, reopen or complete phase. If a problem was opened, but never reopened nor completed, then its observation would appear only once in the output dataset. If a problem had been opened, closed and reopened, then that problem's observation would appear three times in the output.

CREATE NEW DATASETS

First, PROC SQL is used to create three datasets: one with all opened problems, one with all closed problems and the last one with all reopened problems. This method requires reading the source three times. The three datasets are concatenated to create the final dataset. Here is the SQL code:

```
proc sql stimer;                          /* Using PROC SQL: requires      */
   create table opened as select          /* processing the source         */
       full_name,                         /* the source data three times.  */
       address,
       city,
       zip,
       sale_amt,
       sale_dt,
       status_cd,
       "opened" as status length 9
   from bigdata.sgf
   where opened = "Y";
quit;

proc sql stimer;
   create table completed as select
```

```
      full_name,
      address,
      city,
      zip,
      sale_amt,
      sale_dt,
      status_cd,
      "completed" as status length 9
   from bigdata.sgf
   where completed = "Y";
quit;

proc sql stimer;
   create table reopened as select
      full_name,
      address,
      city,
      zip,
      sale_amt,
      sale_dt,
      status_cd,
      "reopened" as status length 9
   from bigdata.sgf
   where reopened = "Y";
quit;

data all_sql;                    /* Now concatenate the 3 SQL datasets */
   set opened
       completed
       reopened
       ;
run;
```

This DATA Step solution processes each problem observation at one time, and an observation is output for each opened, reopened and closed variables set to Y(Yes) by this code:

```
data data_step;           /* Data Step: one run through the source data */
   length status  $9;
   set bigdata.sgf;
   keep full_name    address     city        zip
        sale_amt     sale_dt     status_cd   status;

   if opened = 'Y'  then do;
         status='opened';
         output;
   end;
   if completed = 'Y'  then do;
         status='completed';
         output;
   end;
   if reopened = 'Y'  then do;
         status='reopened';
         output;
   end;
run;
```

Table 6 displays the resource statistics for the steps to create a new dataset by using PROC SQL and a DATA Step.

FULLSTIMER Statistics	Dataset Opened	Dataset Reopened	Dataset Completed	Combine all to create ALL_SQL	Dataset DATA_SET
real time	1:02.66	49.47 seconds	49.92 seconds	32.33 seconds	1:02.73
user CPU time	38.14 seconds	38.12 seconds	33.45 seconds	28.40 seconds	49.10 seconds
system CPU time	8.51 seconds	4.30 seconds	4.01 seconds	3.86 seconds	6.65 seconds
memory	1402.71k	1402.71k	1402.71k	2763.84k	1579.90k
OS Memory	11308.00k	11308.00k	11308.00k	13364.00k	11564.00k

Table 6. Example of resources required when using PROC SQL and DATA Step to create new dataset based on three status variables

Table 7 displays the total resource statistics for the PROC SQL method to create a new dataset as compared to the DATA Step method.

FULLSTIMER Statistics	PROC SQL	DATA Step
real time	3:23.97	1:02.73
user CPU time	2:30.18	49.10 seconds
system CPU time	20.86 seconds	6.65 seconds

Table 7. Resources comparison in two techniques: 3 PROC SQLs vs 1 DATA Step

The two final datasets were identical, yet the DATA Step method used less human and computer resources. Sometimes PROC SQL produces a dataset with fewer resources as compared to a DATA Step solution. Experimentation and experience will help you realize which technique is best for you.

COMPARING DATASETS: PROC COMPARE

The last example showed two ways to create output datasets. You may have situations where you need to verify that you have identical datasets, or have datasets with certain variables whose values should be identical. Data integrity of these types can be tested by using the SAS procedure, PROC COMPARE, which compares datasets and reports mismatches of observations, variables' values, variables' attributes and much more. PROC COMPARE code in its simplest form is:

```
proc compare base=all_sql compare=data_step;
run;
```

Output 14 shows partial results from PROC COMPARE comparing the ALL_SQL and DATA_SET datasets.

```
                          The COMPARE Procedure
                Comparison of WORK.ALL_SQL with WORK.DATA_STEP
                              (Method=EXACT)

                            Data Set Summary

Dataset                   Created           Modified  NVar         NObs

WORK.ALL_SQL     19JUL14:02:38:07  19JUL14:02:38:07     8     33735575
WORK.DATA_STEP   19JUL14:02:38:51  19JUL14:02:38:51     8     33735575


                            Variables Summary

                  Number of Variables in Common: 8.
```

```
                          Observation Summary

                  Observation                Base      Compare

                  First Obs                     1            1
                  First Unequal                 1            1
                  Last  Unequal          33735575     33735575
                  Last  Obs              33735575     33735575

        Number of Observations in Common: 33735575.
        Total Number of Observations Read from WORK.ALL_SQL: 33735575.
        Total Number of Observations Read from WORK.DATA_STEP: 33735575.

        Number of Observations with Some Compared Variables Unequal: 0.
        Number of Observations with All Compared Variables Equal: 33735575.
```

Output 4. Example of results from PROC COMPARE

THE POWER OF DATA TRANSFER SERVICES

Data Transfer Services is the best solution for the transfer of SAS data and external files between a SAS/CONNECT client and a server. The term Data Transfer Services is used to describe PROC UPLOAD and PROC DOWNLOAD. When you use the UPLOAD or DOWNLOAD procedures, SAS/CONNECT will utilize network file compression before transferring the data between the client and server.

The goal of network file compression is to reduce the total number of buffers used by packing each buffer to capacity. SAS' network file compression algorithm uses run-length encoding and sliding window compression. Like all compression, the results are data dependent.

A Transfer Status window is also displayed when PROC UPLOAD or PROC DOWNLOAD is used. This can give you an estimate for how long you have for that much needed coffee break.

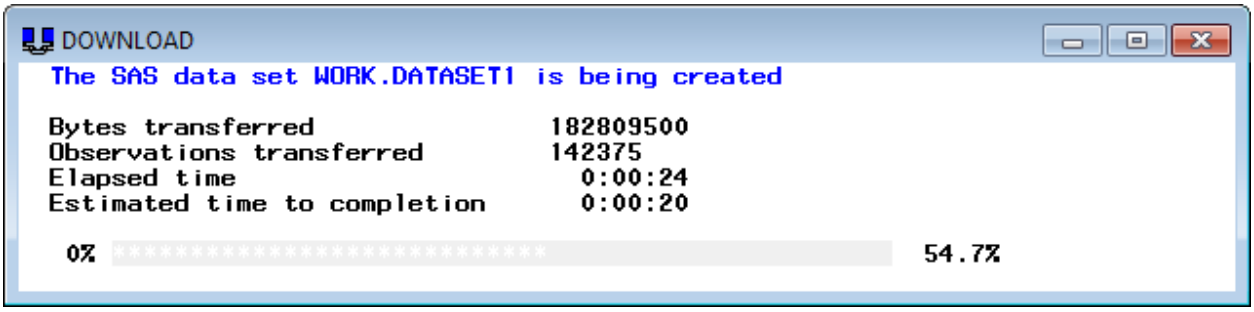

Figure 1. The Transfer Status Window for Downloading a SAS Dataset

Remote Library Services enables you to read, write, and update remote data as if it were stored on the client. It is useful when you need to maintain a single copy of the data on a server and keep the processing on the client. However, it is not ideal for transferring data between a client and server.

Suppose you are tasked with taking over a GRID based SAS Program that imports or exports a large amount of data. If you use Remote Library Services, you will most likely experience lengthy run times and connectivity issues. However, by using PROC DOWNLOAD, you can dramatically mitigate these.

In the following example the objective is to export a SAS dataset to a CSV file. The SAS dataset has 260,303 observations, 34 variables, and a size of 65.1 MB. First, Remote Library Services was used, followed by Data Transfer Services. As you can see below, a significant amount of time was saved using Data Transfer Services.

REMOTE LIBRARY SERVICES

This code exports a SAS dataset directory from the SAS Grid server to a .csv file saved on the client desktop:

```
proc export
   data=rwork.Dataset1
   dbms=csv
   outfile="C:\Users\Jrbeene\Desktop\Dataset1.csv"
   replace;
run;
```

DATA TRANSFER SERVICES

This code first downloads a SAS dataset from the SAS Grid server and then exports the client version of the SAS data to a .csv file saved on the client desktop:

```
rsubmit;
   proc download data = work.Dataset1;
run;
endrsubmit;

proc export
   data=work.Dataset1
   dbms=csv
   outfile="C:\Users\Jrbeene\Desktop\Dataset1.csv"
   replace;
run;
```

Table 9 displays the test results from using RLS and DTS.

	Remote Library Services		Data Transfer Services	
Trial	**Real Time**	**CPU Time**	**Real Time**	**CPU Time**
1	409.30	25.66	48.94	3.42
2	408.32	25.91	45.95	3.19
3	411.80	25.33	45.51	3.22
4	409.89	25.88	45.45	3.24
5	410.31	26.06	45.55	3.26
6	409.59	26.05	46.02	3.24
7	408.72	26.50	45.77	3.29
8	410.56	25.55	46.07	3.28
9	405.29	24.52	46.17	3.33
10	405.31	24.75	45.71	3.23

Table 9. Remote Library Services and Data Transfer Services were each tested 10 times

Table 10 displays the average results across the 10 tests in Table 9.

			Data Transfer Services	
	Remote Library Services	**Data Transfer Services**	**Seconds Faster**	**Percentage Faster**
Real Time	408.91	46.11	362.80	887%
CPU Time	25.62	3.27	22.35	784%

Table 10. Average results across the ten tests in Table 9

Notice that it took significantly less real and CPU time when utilizing Data Transfer Services over Remote Library Services. This example shows that using the appropriate method can have a very real impact on computer and human resources.

CONCLUSION

Even though there are many ways to do the same thing with SAS programming, try experimenting with different approaches. Consider the various ways to also ensure data quality and integrity to guarantee reliable results. Tracking the computer resources required to run different techniques may free up valuable time – yours and your computers'. It takes practice, but you may be able to one day look at a problem and know exactly which coding approach will produce your desired results with the least amount of human and computer resources. Keep trying new ideas. Learn from others and share what you learn!

REFERENCES

SAS Institute Inc. 2011. *SAS/CONNECT® 9.3 User's Guide*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/connref/63066/PDF/default/connref.pdf

SAS Institute Inc. 2012. *SAS® 9.3 SQL Procedure User's Guide*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/sqlproc/63043/PDF/default/sqlproc.pdf

SAS Institute Inc. 2012. *SAS® 9.3 Language Reference: Concepts*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/lrcon/65287/PDF/default/lrcon.pdf

Dickstein, Craig, Pass, Ray, & Davis, Michael. 2007. "DATA Step vs. PROC SQL: What's a neophyte to do?". *Proceedings of the SAS Global Forum 2007 Conference.* Cary, NC. Available at http://www2.sas.com/proceedings/forum2007/237-2007.pdf

Lafler, Kirk Paul. 2014. "Powerful and Hard-to-Find PROC SQL Features". *Proceedings of the SAS Global Forum 2014 Conference.* Cary, NC. Available at http://support.sas.com/resources/papers/proceedings14/1240-2014.pdf

Eberhardt, Peter & Brill, Ilene. 2006. "How Do I Look it Up If I Cannot Spell It: An Introduction to SAS® Dictionary Tables". *Proceedings of the SAS users Group International 31 Conference.* Cary, NC. Available at http://www2.sas.com/proceedings/sugi31/259-31.pdf

ACKNOWLEDGMENTS

We appreciate the feedback, suggestions and encouragement from our mentor, Peter Eberhardt. We also thank our spouses, Peachy Beene and Harvey Katz, for their support and patience.

CONTACT INFORMATION

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

Mary Federico Katz
marymfk77@gmail.com

Jason Beene
jrbeeneSAS@gmail.com

The views and opinions expressed here are those of the authors and do not necessarily reflect the views and opinions of Wells Fargo Bank. Wells Fargo Bank is not, by means of this article, providing technical, business, or other professional advice or services and is not endorsing any of the software, techniques,

approaches, or solutions presented herein. This article is not a substitute for professional advice or services and should not be used as a basis for decisions that could impact your business.