Paper 103-26

50 WAYS TO MERGE YOUR DATA – INSTALLMENT 1…

Kristie Schuster, Lab*One*, Inc., Lenexa, Kansas
Lori Sipe, Lab*One*, Inc., Lenexa, Kansas

ABSTRACT

When you need to join together two datasets, how do you go about it? Do you use your tried and true methods or do you evaluate multiple methods and determine which is going to be the most efficient for your application? The following paper compares four different ways to merge together two different datasets and compares the system resources used for each method.

The SAS tools used in these examples are all in Base SAS. The operating system used is Open VMS but you would most likely see similar efficiencies across all operating systems. The skill level should be considered beginner to intermediate.

INTRODUCTION

LabOne, Inc. operates a centralized laboratory designed to provide high-quality clinical outpatient testing services to the health care industry, risk-assessment testing services to the insurance industry and substance abuse testing services to both regulated and non-regulated industries.

The laboratory business has been growing steadily for the past few years and the Business Information Services Department has been maintaining and reporting on the laboratory data for at least the past ten years. Our datasets are continually growing and we are constantly challenged to find ways to manipulate the data in an efficient manner.

Several years ago, one of our Statistical Engineers had developed what we refer to as GENERIC.SAS, a shell program we use to start every new project. This program was developed to prevent hardcoding, sets up all of the macros necessary to pull in the most common variables and performs the standard manipulations common to all of our programs. This program was developed utilizing v6.06 of SAS and has been used by all Statistical Engineers since that time. When the shell program was written, we did not have the number of options that are available to us today. In the midst of doing our research, we discovered that there are several ways to combine datasets and perhaps it was time to re-write this program to utilize some of the new tools that have been developed.

The four methods we will look at are a straight merge of two datasets, a merge using an index, a merge using formats and a merge using SQL.

METHOD ONE – MERGE STATEMENT

This method utilizes the merge statement to join the two datasets together. With this type of merge, both datasets must be sorted in the same order. If they are not already in the desired sort order, you must use additional processing time to perform the sort. In addition, all records must be read from the datasets before merging is accomplished.

```
/* MERGE TWO DATASETS W/MERGE STATEMENT */

   PROC SORT DATA=TEMP.GAPINFO OUT=GAPINFM;
   BY CLTNUM;

NOTE: The data set WORK.GAPINFM has 18607
observations and 4 variables.

NOTE: PROCEDURE SORT used the following computer
resources:

Elapsed time: 000:00:01.39
CPU time:     0 00:00:00.73

   PROC SORT DATA=TEMP.CALLINFO OUT=CALLINFM;
   BY CLTNUM;


NOTE: The data set WORK.CALLINFM has 3798
observations and 3 variables.

NOTE: PROCEDURE SORT used the following computer
resources:

Elapsed time:   0 00:00:00.35
CPU time:       0 00:00:00.20

   DATA FINAL;
   MERGE GAPINFM(IN=A) CALLINFM(IN=B);
   BY CLTNUM;
   IF A AND B;
   RUN;

NOTE: The data set WORK.FINAL has 3798
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:        0 00:00:00.98
CPU time:            0 00:00:00.37

/* MERGE TWO LARGE DATASETS W/MERGE STATEMENT */

PROC SORT DATA=TEMP.APPINF2 OUT=APPINF2;
   BY CONTNUM;

NOTE: The data set WORK.APPINF2 has 922967
observations and 4 variables.

NOTE: PROCEDURE SORT used the following computer
resources:

Elapsed time:  0 00:02:02.82
CPU time:      0 00:00:39.10

PROC SORT DATA=TEMP.CALLINF2 OUT=CALLINF2;
    BY CONTNUM;

NOTE: The data set WORK.CALLINF2 has 359805
observations and 6 variables.

NOTE: PROCEDURE SORT used the following computer
resources:

Elapsed time: 0 00:01:03.02
```

```
CPU time:     0 00:00:17.23

    DATA FINALL;
    MERGE APPINF2(IN=A) CALLINF2(IN=B);
       BY CONTNUM;
       IF A AND B;
       RUN;

NOTE: The data set WORK.FINALL has 359805
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:02:42.15
CPU time:      0 00:00:18.96
```

Analysis of Method 1: We found that using the Merge statement to join two small datasets (<20,000 records) was fairly efficient in the elapsed time and CPU time. Using the same datasets, only the SQL procedure was ranked higher in efficiency in both elapsed and CPU time. When it came to joining two large datasets (>250,000 records), the Merge statement won the title of 'Overall Most Efficient Method to Merge.'

Some noticeable advantages that we observed included:

1) The ease of combining multiple (more than 2) datasets.
2) Aside from disk space, you are not limited in the size of the tables with which you are working.
3) The merge can be performed on more than variable, allowing greater flexibility in data merging.

Our research showed the major drawback of using the Merge statement to be that your datasets must be sorted in the same order. This may require an additional step (and more coding on your part) to sort the datasets. You should also keep in mind that an exact match must be found on the by variables and the by variables must be found in both datasets.

METHOD TWO – MERGE BY INDEX

This method utilizes an index to join the two datasets together. If the dataset is not already indexed by the variable by which you are sorting, you must create the index. An index can be created in several ways. For example, you can create an index like the one in the example, or you could utilize the PROC SQL CREATE INDEX statement. You can use a simple index as in our example or a composite index utilizing two or more columns in a table. You cannot explicitly tell the SAS System to use an index that has been created in processing. The SAS System will determine the most efficient way to process a query or statement.

```
/* MERGE TWO SMALL DATASETS USING AN INDEX */

  DATA GAPINFO1 (INDEX=(CLTNUM));
    SET TEMP.GAPINFO;
    RUN;

NOTE: The data set WORK.GAPINFO1 has 18607
observations and 4 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time: 0 00:00:05.30
CPU time:     0 00:00:01.15

   DATA CALLINF1 (INDEX=(CLTNUM));
   SET TEMP.CALLINFO;
   RUN;

NOTE: The data set WORK.CALLINF1 has 3798
observations and 3 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:   0 00:00:01.27
CPU time:       0 00:00:00.31

   DATA FINALI;
   SET CALLINF1;
   SET GAPINFO1 KEY=CLTNUM;
   RUN;

NOTE: The data set WORK.FINALI has 3798
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:00:14.56
CPU time:      0 00:00:01.33

/* MERGE TWO LARGE DATASETS USING AN INDEX */

    DATA APPINF2 (INDEX=(CONTNUM));
    SET TEMP.APPINF2;
    RUN;

NOTE: The data set WORK.APPINF2 has 922967
observations and 4 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:04:54.13
CPU time:      0 00:00:53.16

    DATA CALLINF2 (INDEX=(CONTNUM));
    SET TEMP.CALLINF2;
    RUN;

NOTE: The data set WORK.CALLINF2 has 359805
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:01:44.18
CPU time:      0 00:00:21.35

      DATA FINALL;
      SET CALLINF2;
      SET APPINF2 KEY=CONTNUM;
      RUN;

NOTE: The data set WORK.FINALL has 359805
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 01:50:39.81
CPU time:      0 00:09:49.24
```

Analysis of Method Two: Using an index to join the two small datasets proved to be the least efficient method of the four. The same held true when joining the two larger datasets. With this in mind, we determined that the index was not particularly useful for our merge. We agreed, however, that because an index allows SAS to read large datasets very quickly, you would tend to see greater benefits when working with extremely large datasets. While you do gain speed at which the dataset is read, you must remember that more CPU and I/O time is used to create and maintain the index.

There are other points to think about when using an index to merge, such as the need to create an index if one does not already exist. Like the Merge statement, the indexed variable must be an exact match and must also exist in all datasets. If you are joining two datasets utilizing multiple SET statements, you must remember that you must use a true subset of the larger dataset. In other words, you will get unexpected results if you try to join two datasets where you have observations that exist in one or the other, but does not exist in both.

METHOD THREE – MERGE BY FORMAT

This method uses a Format procedure to join the two datasets. Undoubtedly you have used the Format procedure many times to create your own formats. It is an excellent tool for re-defining what value you want to use for a given variable. For example, using a formatted variable called AGE on a dataset having patients of all ages and assigning AGE the value of *X* for patients between the ages of 20 and 25, allows you to represent *X* as '20 yrs - 25 yrs.'

Merging datasets with the Format Procedure adds an interesting twist to this procedures function. Using a data step, read in one of the datasets and set the value, variable type (character/numeric), and format name for your key variable. The CNTLIN option then builds the format using the specified dataset. In another data step, the second dataset is then set where the format is equal to the value you have assigned, giving you just those records that meet your selection criteria.

```
/* MERGE TWO SMALL DATASETS USING A FORMAT */

  DATA GAPINFOF;
  SET TEMP.GAPINFO;
  RUN;

NOTE: The data set WORK.GAPINFOF has 18607
observations and 4 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:   0 00:00:01.26
CPU time:       0 00:00:00.37

 DATA CALLINF ;
 SET TEMP.CALLINFO;
 RUN;

NOTE: The data set WORK.CALLINF has 3798
observations and 3 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:   0 00:00:00.29
CPU time:       0 00:00:00.10

   DATA CALLINF2;
   SET CALLINF (RENAME=(CLTNUM=START));
   LABEL='Y';
   TYPE='C';
   FMTNAME='$CALL';
   RUN;

NOTE: The data set WORK.CALLINF2 has 3798
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:   0 00:00:00.20
CPU time:       0 00:00:00.08

  PROC FORMAT CNTLIN=CALLINF2;

NOTE: Format $CALL has been output.

   RUN;

NOTE: PROCEDURE FORMAT used the following
computer resources:

Elapsed time:   0 00:00:00.93
CPU time:       0 00:00:00.43

  DATA GAPINF2;
  SET GAPINFOF;
  WHERE PUT(CLTNUM,$CALL.) EQ 'Y';
  RUN;

NOTE: The data set WORK.GAPINF2 has 3798
observations and 4 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:00:00.76
CPU time:      0 00:00:00.47

/* MERGE TWO LARGE DATASETS USING A FORMAT */

     DATA APPINF2;
     SET TEMP.APPINF2;
     RUN;

NOTE: The data set WORK.APPINF2 has 922967
observations and 4 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:01:04.55
CPU time:      0 00:00:13.24

    DATA CALLINF2;
    SET TEMP.CALLINF2;
    RUN;

NOTE: The data set WORK.CALLINF2 has 359805
observations and 6 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:00:44.53
CPU time:      0 00:00:06.51

   DATA CALLINFL;
   SET CALLINF2 (RENAME=(CONTNUM=START));
   LABEL='Y';
   TYPE='N';
   FMTNAME='CALLL';
   RUN;

NOTE: The data set WORK.CALLINFL has 359805
observations and 9 variables.

NOTE: DATA statement used the following computer
resources:
Elapsed time:  0 00:00:41.39
CPU time:      0 00:00:07.65

  PROC FORMAT CNTLIN=CALLINFL;

NOTE: Format CALLL has been output.
```

```
  RUN;

NOTE: PROCEDURE FORMAT used the following
computer resources:

Elapsed time:  0 00:02:16.48
CPU time:      0 00:00:41.84

   DATA APPINFL;
   SET APPINF2;
   WHERE PUT(CONTNUM,CALLL.) EQ 'Y';
   RUN;

NOTE: The data set WORK.APPINFL has 359805
observations and 4 variables.

NOTE: DATA statement used the following computer
resources:

Elapsed time:  0 00:02:56.98
CPU time:      0 00:00:39.71
```

Analysis of Method Three: The Format method ranked third in efficiency when working with the two small datasets, but managed to move up to second when working with the larger datasets. We were not actually able to join all of the variables that existed in our formatted dataset, but rather could only capture the formatted variable. So despite the fact that this method increased efficiency as the dataset size grew, it still did not prove to be a very good method for joining our tables. Should you have a situation however, where you only need to pull in one variable to identify your population, using a formatted variable could be a very efficient and beneficial join. We found an ideal example in our GENERIC program where we are able to capture company demographics from a table that stores company information, and join this to a table containing our historical data by using a formatted company code. This method is working quite well and has made this program more efficient.

METHOD FOUR – MERGE BY PROC SQL

This method uses PROC SQL to join the two datasets. The SQL procedure allows the programmer to create a table(s), essentially a SAS dataset. Within a single SQL procedure, you have the ability to select specific rows by using a Where statement and even to sort your data by using an Order by statement. Merging multiple datasets using SQL is done utilizing the Where statement.

```
/* MERGE TWO SMALL DATASETS USING PROC SQL  */

    PROC SQL;
    CREATE TABLE FINALS AS
    SELECT DISTINCT B.CLTNUM, STIME, ETIME,
    COMPID, LNAME
    FROM TEMP.GAPINFO A, TEMP.CALLINFO B
    WHERE A.CLTNUM=B.CLTNUM
    ORDER BY CLTNUM;

NOTE: Table WORK.FINALS created, with 3798 rows
and 5 columns.

    QUIT;

NOTE: PROCEDURE SQL used the following computer
resources:
Elapsed time:      0 00:00:02.25
CPU time:          00:00:00.90

/* MERGE TWO LARGER DATASETS USING PROC SQL */

       PROC SQL;
       CREATE TABLE FINALL AS
       SELECT DISTINCT A.CONTNUM, B.TSTID,
       B.TSTDTE,B.PROCDATE
       FROM TEMP.APPINF2 A, TEMP.CALLINF2 B
       WHERE A.CONTNUM=B.CONTNUM
       ORDER BY CONTNUM;

NOTE: Table WORK.FINALL created, with 359805
rows and 4 columns.

       QUIT;

NOTE: PROCEDURE SQL used the following computer
resources:

Elapsed time:      0 00:13:14.57
CPU time:          0 00:02:14.99
```

Analysis of Method Four: We determined that the SQL procedure can be a very efficient tool when merging two small datasets, as shown by ranking first in efficiency for the first test. However, this method fell all the way to third place, behind the Merge statement and the Format procedure, when we tested the join on larger datasets. The SQL procedure is very I/O intensive and as the size of your dataset grows your efficiency will decrease drastically, particularly if you're working on an I/O bound system. This procedure has another disadvantage in that you are limited in the number of tables that you can join, the maximum being 16.

On the other hand, using the SQL procedure does have some significant advantages. For example, your data does not need to be sorted prior to execution and you may use multiple key variables to merge by and they need not exist in all datasets. As a programmer, a great advantage to using SQL is that it typically requires less coding!

CONCLUSION

	ELAPSED TIME		CPU TIME	
METHOD	SMALL	LARGE	SMALL	LARGE
SQL	2.25	13:14.57	.90	2:14.99
MERGE	2.72	5:47.99	1.30	1:15.29
FORMATS	3.44	7:43.93	1.45	1:48.95
INDEX	21.13	1:57:18.12	2.79	11:03.75

Ok, so we don't have 50 ways to Merge Your Data yet, who knows what the future holds! The four techniques described above are to provide you, the programmer, with a variety of options. Our research indicates that the SQL procedure is most efficient when joining smaller datasets, but the Merge statement is most efficient when dealing with very large datasets. Basic use of these methods tends to be fairly straightforward and is easy to code and understand. The Index and Format methods are more complicated and are a little harder to code, but would work well in the appropriate application. Ultimately you will need to evaluate each method and make a determination which provides the most efficiency depending on your data and what you need to do with it.

CONTACTS

Your comments and questions are valued and encouraged.

Contact the authors at:

Kristie Schuster
Lab*One*,Inc.
10101 Renner Boulevard
Lenexa, Kansas 66219-9752
Phone: (913)577-1318
Fax: (913)888-4160
Email: kristie.schuster@labone.com
Web: www.labone.com

Lori Sipe
Lab*One*,Inc.
10101 Renner Boulevard
Lenexa, Kansas 66219-9752
Phone: (913)577-1957
Fax: (913)888-4160
Email: lori.sipe@labone.com
Web: www.labone.com