BB144

Table Lookups: Getting Started With Proc Format

John Cohen, Advanced Data Concepts LLC, Newark, DE

ABSTRACT

Table lookups are among the coolest tricks you can add to your SAS® toolkit. Unfortunately, these techniques can be intimidating both conceptually and in terms of the programming. We will introduce one of the simplest of these techniques, employing Proc Format and the CNTLIN option as part of our construct. With any luck, this will prove both easy-enough to program and more efficient to run.

INTRODUCTION

Table Lookups is a fancy way of saying that we intend to somehow combine data from two (or more!) different SAS data sets (or *tables*). The basics are that we have a data set *one* and a data set *two* and intend to match up part or all of the observations (or rows) from data set *one* with those of data set *two*. We may choose to only keep the rows which match up in both data sets, all the rows included in data set *one* AND only those which match in data set *two*, or all of the rows from both data sets regardless of whether or not they match up. (This latter case is actually relatively rare and uninteresting unless we want to talk about Cartesian Products. Which we do not in the context of this paper.)

Most of us learned to perform this critical and basic task using the Data Step/By Merge (or Match/Merge as it is also referred to). Sort both data sets *one* and *two* by a *key* variable common to both. Then combine them in a Data Step using the key word *Merge*. The result will be a new data set with the matching rows lined up (and the non-matched rows – if we keep them – interspersed in the appropriate order). The basics are suggested in Figure 1 below.

Figure 1
Data Step/By Merge

```
Proc sort data = one; by key_variable; run;       /** sort One on key **/
Proc sort data = two; by key_variable; run; /** sort Two on same key **/

Data three;                              /** merge One & Two on same key **/
   merge one(in=a) two(in=b);
   by key;
   if a & b;
run;
```

When our individual data sets are fairly small there is little reason to use any technique other than this. It is easy to code and debug, easy to document, and universally recognizable by any SAS programmer who might inherit your programs.

IF IT WORKS, WHY CHANGE A THING?

With any luck, we can use this tried-and-true technique for all of our programs. But what it we are not so lucky? What if some of our data sets start to get really large, so large that it takes 20 minutes or even as much as an hour to finish sorting your data? And even worse, what if to answer a particular business question we have to combine a series of data sets, each in turn sorted by different variables? In Figure 2 we sketch out just such a scenario.

Figure 2
When Data Step/By Merges Go "Bad"

```
Proc sort data = really_big_data_set; by key_variable_one; run;

Proc sort data = even_bigger_data_set; by key_variable_one; run;

Data three;                                        /** same task as in Figure 1 **/
   merge really_big_data_set(in=a)  /** but these are really large tables **/
         even_bigger_data_set(in=b);
   by key_variable_one;
   if a & b;
run;

                                /** sort these and merge by a different key **/
Proc sort data = three; by key_variable_two; run;

Proc sort data = extraordinarily_humongous_data_set; by key_variable_two;
run;

Data four;
   merge three(in=c)
         extraordinarily_humongous_data_set(in=d);
   by key_variable_two;
   if c & d;
run;

                               /** sort these and merge by even another key **/
Proc sort data = four; by key_variable_three; run;

Proc sort data = the_mother_of_all_data_sets; by key_variable_three; run;

Data five;
   merge four(in=e)
         the_mother_of_all_data_sets(in=f);
   by key_variable_three;
   if e & f;
run;
```

So if each of these very large steps is taking say, half-an-hour, an hour, or overnight to complete (assuming it doesn't fail entirely!), we are beginning to have a problem. And how do you easily debug a process which takes so long? SAS being SAS, there are a number of potential solutions, the easiest

being to convince your manager to buy you a faster computer. If that won't be in the budget for this year, though, we will need to come up with a programming solution.

TABLE LOOKUPS

In the Data Step/By Merge examples above we are reading in each individual observation or row from the incoming data sets, organizing them, and then writing these same observations or rows back out into the appropriate (sort) order. Once the respective sorts are completed (on each data set), we then compare the rows on the key variable and then write out the appropriate newly-combined observations into a new data set. If each of these rows is large (many variables) and there are lots of observations, our computing task thus involves reading in and writing back out lots and lots of data. Which pretty much sums up the nature of much of our work. So even though we think about our analytics and statistics and compute-intensive activities, at least in preparing our data for our analyses, much of our task is reading IN and writing back OUT our data. (In computer science jargon this is known as being I/O-bound.) And reading from disk (or even worse, tape or CDs) and writing back out is a very slow, expensive process.

OUR SOLUTION: READING IN LESS DATA

The programming solution we will be exploring here is one in a category called "Table Lookups." If reading in and writing back out data is slowing us down, we will try to minimize the amount of data being processed. There are many ways to achieve that task. (Any paper on SAS programming efficiency will contain examples.) We are interested in a particular set of techniques where we both minimize the amount of data being handled and also use the speediest possible way of reading in and writing back out intermediate data sets, namely using computer memory. While this location still entails moving or copying parts of physical observations, the nature of most computers is that internal memory (RAM) is significantly faster than regular/permanent disk/storage space.

We will employ an option in Proc Format, the **CNTLIN** option, to create a lookup table. This lookup table will include a small subset of the original data set – the key variable and an associated description -- and will be stored in computer memory. We will then use this table to very rapidly search the second, larger data set and write back out the selected observations.

The Proc format task is a multi-step process. First we create a data set with certain special variables. We make sure that this data set is ordered (sorted) by the key variable and has no duplicates. Finally, we run the Proc Format with the **CNTLIN** option. Figure 3 indicates these steps.

Figure 3
Proc format with CNTLIN Option

```
                                          /** Note special variables **/
Data Special_data_set;
   set large_data_set;
   start = key_variable;                          /** usually a code **/
   end = key_variable;
   label = description;        /** usually an accompanying description **/
   type = 'C';                      /** Character, Numeric, or Picture **/
   length fmtname $8;
   fmtname = 'myfmt';           /** standard format naming conventions **/
   drop key_variable description;
run;

                                    /** data set sorted and de-dupped **/
Proc sort data = Special_data_set nodupkey; by start; run;

                                   /** convert SAS data set into a format **/
Proc format cntlin = Special_data_set; run;
```

To use this lookup table we will employ a PUT function in a Data Step in which we read in the larger data set as shown in Figure 4. The exciting elements are that we sorted the smaller data set (relatively cheap) and created a little lookup table stored in memory. We will (in a very efficient, memory-intensive but NOT I/O-intensive process) compare the observations in the large data set – which we never sorted – to the lookup table values in our format and write out only the observations we need.

Figure 4
Using the Format (Stored in Memory) to Perform a Table Lookup

```
Data The_Observations_We_Want;
   Set one_of_our_very_large_but_unsorted_data_sets;

         /** examples of using the stored format to select observations **/
   if put(key_variable,$myfmt.) = "description";

         -or-

   if put(key_variable,$myfmt.)
      in ("description1","description2","description3");
run;
```

In an actual data example the key_variable might be a therapeutic area code, the format be named $th_area, and the description being the names (spelled out) of those therapeutic areas, and our selection criteria might be:

```
if put(therapeutic_are_code,$th_area.) = "congestive heart failure";
```

Or our example instead might map zip code to MSA (Census Bureau's Metropolitan Statistical Area) and be selecting a subset of interest as in:

```
if put(key_variable,$MSA.)
   in ("BUFFALO-NIAGRA FALLS, NY","NASSAU-SUFFOLK, NY","NEW YORK, NY");
```

DISCUSSION

As cool as this is, we would be remiss to not note that there is quite a bit more in the way of programming, debugging, and documentation in making this approach work. Further, there is some processing cost to creating the format. Or format**s**, if several will be needed in the course of completing one analysis.

Standard good programming, testing, and documentation SOPs are the essentials to successfully incorporating any new tool set into one's repertoire. And if one's tasks include combining a file of, say, 3.3 million rows (and 3.42 GB in size) with another file of, say, 26.1 million rows (and 4.38 GB in size), with this result then being matched up with additional files of 90,000-odd observations, 17,654, observations, 8,902 observations, and finally, one containing a bare 1,404 rows -- one therapeutic area at a time, with monthly updates – dreaming of a faster laptop may be less productive than adding a few programming tricks.

Here certain *productionizing* approaches are employed – note that we are using the values of the incoming data to create our format rather than coding these manually. And if we store these formats – as the incoming data are part of a weekly/monthly standard update process – in a common location then everyone in our department can realize a return on the once-a-cycle overhead of creating these formats.

CONCLUSIONS

We have suggested here that certain circumstances may require more sophisticated approaches to combining data sets than the standard Data Step/By Merge. Due to the I/O-bound nature of much of our work certain tasks – as larger and larger data sets are involved – may become tedious or even problematic. One approach is to tackle the I/O-bound nature of many of our tasks by simplifying some of the data being compared and storing the lookup tables in computer memory. Less data need be read and written to slower media in the comparison process and more of the work is performed in much faster RAM. The resulting output data sets nevertheless are identical to those created using standard approach.

At the accompanying presentation we will compare the results of actual data prepared via Data Step/By Merge contrasted with the Proc Format Table Lookup approach along with the attendant benchmarks.

REFERENCES

Jenine Eason, “a Speedy Alternative to Sort/Merge,” SUGI 30, April 2005, Paper 054-30, Philadelphia, PA.

Russell Lavery, “An Animated Guide: Power Merges: The format table lookup,” NESUG 16, September, 2003, Washington, DC.

Base SAS 9.1.3 Procedures Guide, Second Edition, Volumes 1-4, SAS Institute, Inc., March, 2006,Cary, NC

Sample 24702: Performing a table lookup with large nonindexed data sets
http://support.sas.com/kb/24/702.html

Sample 25054: Adding an OTHER category to a format using CNTLIN in PROC FORMAT
http://support.sas.com/kb/25/054.html

ACKNOWLEDGMENTS

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the author at:

John Cohen
Advanced Data Concepts LLC
17 Kensington Lane
Newark, DE 19713
Work Phone: (302) 559-2060
Email: jcohen1265@aol.com

****