BB-105

No FREQ-in Way

Renee Canfield, Experian, Atlanta, GA

ABSTRACT

In the consumer credit industry, privacy is key and the scrutiny increases every day. When returning files to a client, they must be depersonalized so the client cannot match back to any personally identifiable identification (PII). This means we must locate any values for a variable that occur on a limited number of records and null them out (i.e. replace them with missing values). Working with large files which have more than one million observations and thousands of variables, locating variables with few unique values is a difficult task. While PROC FREQ and DATA step merging can accomplish the task, using first./last. by variable processing to locate the suspect values and hash objects to merge the data set back together may offer increased efficiency.

INTRODUCTION

This paper will demonstrate an easy application of the hash object for the audience of a beginner or experienced SAS® programmer. It may offer time savings compared to “traditional” SAS procedures. We will compare two methods of solving the business problem, tested on the “rule of 5” (i.e. the values must occur on at least 5 records): one using PROC FREQ and DATA step match-merge and the other using first./last. by variable processing and hash object lookups. We will also utilize SAS metadata, called dictionary tables, in combination with SAS macro assignments in PROC SQL to avoid hard coding our programs.

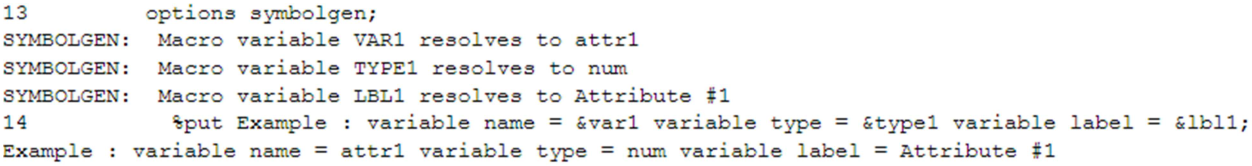
USING DICTIONARY TABLES (METADATA) WITH PROC SQL MACRO ASSIGNMENT

To solve this business problem we must perform the same task on thousands of variables and macros easily lend themselves to such operations. But how will we list the thousands of variable names in SAS? Are we going to upload a file to read in or type out each name in the program? Fortunately neither step is necessary, because SAS automatically stores metadata in dictionary tables during each session. We can query them using PROC SQL and then push the information into macro variable lists to use in loops. No hardcoding is necessary.

```
/* read in raw data file */

data rawreturn;
  retain matchkey 0;
  infile dat('reged1_test.data') missover lrecl=4000 DELIMITER='20'x termstr=lf ;
  %macro inp;
    input
      %do i = 1 %to 999;
        attr&i
      %end;
     ;
    label
      %do j = 1 %to 999;
        attr&j = "Attribute #&j"
      %end;
     ;
    %mend;
    %inp
    matchkey = matchkey + 1; /* create a match key */
run;

data master;
  set rawreturn(keep=matchkey);
run;

/* queries the name field from dictionary.columns table to count the number of
variables (i.e. columns) in the data set rawreturn, minus the matchkey field which
will not be considered because it is an anonymous key that does not need to be
nulled out */

proc sql;
  select count(distinct name) into : ttlcol  /* creates a macro var called ttlcol*/
```

```
      from dictionary.columns
     where libname = 'WORK'
       and memname = 'RAWRETURN' /* data set name */
       and upcase(name) not in ('MATCHKEY');
  %let ttlcol = &ttlcol; /* removes blanks */

  /* dynamically assign macro variables from name, type and label metadata
  (i.e. This simple PROC SQL statement will assign 3 macro variables var, type and lbl
  for each record variable in the rawreturn data set.  We did not have to tell it how
  many records to look for because we stored that number in ttlcol.) */

  select name, type, label
    into : var1 - : var&ttlcol, : type1 - : type&ttlcol, : lbl1 - : lbl&ttlcol
    from dictionary.columns
   where libname='WORK'
     and memname='RAWRETURN'
        and upcase(name) not in ('MATCHKEY');

  quit;

  %put Example : variable name = &var1 variable type = &type1 variable label = &lbl1;
```

Display 1 is an example of the macro variables created from the PROC SQL code above for variable #1.



```
13          options symbolgen;
SYMBOLGEN:  Macro variable VAR1 resolves to attr1
SYMBOLGEN:  Macro variable TYPE1 resolves to num
SYMBOLGEN:  Macro variable LBL1 resolves to Attribute #1
14           %put Example : variable name = &var1 variable type = &type1 variable label = &lbl1;
Example : variable name = attr1 variable type = num variable label = Attribute #1
```

Display 1. SAS Log Window to show macro variable #1 resolved

Table 1 is an example of the data set rawreturn:

matchkey	attr1	attr2	attr3	…	attr999
1	241	4900	20		5
2	183	4900	20		4
3	183	4900	10		5
4	184	5880	10		5
5	241	5880	10		5
6	241	4900	10		5
7	241	2630	20		6
8	241	5880	20		4
9	172	5880	20		7
10	241	4900	20		6

Table 1. rawreturn

PROC FREQ APPROACH

Now that we have a list of variables stored in macro variables, we insert them in a loop and null out those values that occur on less than 5 observations to depersonalize the data. Our first attempt at solving our problem involves running PROC FREQ on every one of the variables individually and looking for any values which appear on less than 5 observations. Then the output data set with counts by variable values is subset to only those where count is less than 5.

```
%macro nullitf;
  %do i=1 %to &ttlcol; /* loop through all of the variables in rawreturn data set */

  /* sort out one variable at a time */
  proc sort data=rawreturn(keep=matchkey &&var&i) out=sorted_&i;
    by &&var&i;
  run;

  /* run a freq on that variable and only keep the values that occur on < 5 obs */
  proc freq data=sorted_&i(keep=&&var&i) noprint;
    tables &&var&i/out=freq_&i(where=(count lt 5) drop=percent);
  run;

  /* merge those “problem” values back onto sorted data set and null them */
  /* we have one data set per variable with values nulled out where necessary */

  data nulled_&i;
    merge sorted_&i(in=o) freq_&i (in=l keep=&&var&i);
    by &&var&i; /* we merge by the variable &&var&i here and not the matchkey */
    if o;

    /* the call missing() routine sets the variable to missing if it exists in the
    freq_&i data set which has values occurring on <5 obs */
    if l then call missing(&&var&i);

  run;

  proc sort data=nulled_&i;
    by matchkey;
  run;

  %end;

  proc datasets lib=work;
    delete sorted: freq: ;
  run; quit;
%mend;

%nullitf
```

We will show an example of the macro processing for the first variable attr1. Table 2 shows the data set sorted_1 which is the rawreturn data set sorted by attr1. Notice we show the first four rows in bold red because the values 172, 183 and 184 occur on less than 5 observations. The frequency procedure counts these values and outputs them in Table 3. In the final Table 4 we see that the values 172, 183 and 184 have been set to missing but the values 241 remain intact.

matchkey	**attr1**
9	**172**
2	**183**
3	**183**
4	**184**
1	241
5	241
6	241
7	241

matchkey	**attr1**
8	241
10	241

Table 2. sorted_1

attr1	**count**
172	1
183	2
184	1

Table 3. freq_1

matchkey	**attr1**
1	241
2	.
3	.
4	.
5	241
6	241
7	241
8	241
9	.
10	241

Table 4. nulled_1

To create the final return data set, the variables are merged together using the DATA step match-merge.

```
%macro createf;

  /* one DATA step and a loop merges all the nulled_&i data sets back together on
  matchkey */

  data hsh.final_return_frq;
    length matchkey 8.;
    merge master(in=m)
    %do i=1 %to &ttlcol;
      nulled_&i
    %end; ;
    by matchkey;
    if m; /* left join on matchkey from master data set */
  run;

  proc datasets lib=work;
    delete nulled: ;
  run; quit;

%mend;

%createf
```

The macro %createf merges all the nulled_&i data sets together on matchkey using a DATA step match-merge like in Figure 1. The final data set is found in table 5.

matchkey
1
2
3
4
5
6
7
8
9
10

matchkey	attr1
1	241
2	.
3	.
4	.
5	241
6	241
7	241
8	241
9	.
10	241

matchkey	attr2
1	4900
2	4900
3	4900
4	.
5	.
6	4900
7	.
8	.
9	.
10	4900

…

matchkey	attr999
1	5
2	.
3	5
4	5
5	5
6	5
7	.
8	.
9	.
10	.

Figure 1. DATA step match-merge

matchkey	attr1	attr2	attr3	…	attr999
1	241	4900	20	…	5
2	.	4900	20	…	.
3	.	4900	.	…	5
4	.	.	.	…	5
5	241	.	.	…	5
6	241	4900	.	…	5
7	241	.	20	…	.
8	241	.	20	…	.
9	.	.	20	…	.
10	241	4900	20	…	.

Table 5. hsh.final_return_frq

FIRST./LAST. BY VARIABLES AND HASH OBJECT APPROACH

In the alternative solution we create a counter variable on the sorted data set to count individual values and only keep those which appear on less than 5 observations. Then a DATA step hash object nulls out the appropriate values and another hash object merges the nulled data sets together.

```
%macro nullith;
  %do i=1 %to &ttlcol; /* loop through all the variables in rawreturn data set */

  /* sort out one variable at a time */
  proc sort data=rawreturn(keep=matchkey &&var&i) out=sorted_&i;
    by &&var&i;
  run;

  /* use first. and last. variables to count the values of a by-group variable */
  /* keep any values that occur on less than 5 records */
```

```
    data firstlast_&i;
      set sorted_&i(keep=&&var&i);
      by &&var&i;
      if first.&&var&i then count = 1; /* start counter at 1 for each &&var&i value */
      else count + 1;

      /* only keep last instance of the variable &&var&i value if count is less than 5
      (ie. value occurs on < 5 observations) */
      if last.&&var&i and count lt 5;

    run;

    /* null out the necessary values by looking them up in the hash object created
    from the firstlast_&i data set */

    data nulled_&i(drop=rc);
      if _n_ = 1 then do;
        if 0 then set firstlast_&i(drop=count); /* set up PDV */

          /* declare and instantiate the hash object n&i created from the data set
          firstlast_&i */
          declare hash n&i(dataset: "work.firstlast_&i");
          n&i..definekey("&&var&i");  /* define the key as &&var&i */
          n&i..definedata("&&var&i"); /* define the data as &&var&i */
          n&i..definedone(); /* done defining the hash object */
      end;

      set sorted_&i;

      /* rc = return code that says whether or not the key variable &&var&i has a
      match in the hash object n&i (=0 if there is a match, <>0 otherwise)*/
      rc = n&i..find(key:&&var&i);

      if rc eq 0 then call missing(&&var&i); /* nulls values out if match found */
    run;

    %end;

    proc datasets lib=work;
      delete sorted: firstlast: ;
    run; quit;
  %mend;

  %nullith
```

The %nullith macro begins the same way the %nullitf did, with a PROC SORT, so you can see Table 2 for details on sorted_1. The nulled_1 data set is also the same so see Table 4. The difference in this code is where we replace the PROC FREQ with a DATA step and use the first./last. by variables to count the distinct values for attr1. Table 6 is the data set firstlast_1 and contains values which occurred on less than 5 records for the variables attr1. We can see that it matches Table 3 so we get the same result with either %nullith or %nullitf.

attr1	count
172	1
183	2
184	1

Table 6. firstlast_1

The %createh macro declares a hash object for each of the nulled_&i data sets. Then instead of a DATA step match-merge, it uses those hash objects to lookup the match key and appends all the variables that have been nulled where appropriate. Instead of reading all the nulled_&i data sets like the previous solution, it loads them all into hash objects in memory to access using FIND().

```
%macro createh;
  data hsh.final_return_hsh(drop=rc);
    length matchkey 8.;

    /** Set up HASH objects for each of the &&ttlcol columns **/
    if _n_ = 1 then do;
      %do i=1 %to &ttlcol;

       /** Set up metadata for hash object (i.e. adds variable info to PDV) **/
       if 0 then set nulled_&i;

        /* declare and instantiate the hash object h&i created from the data set
        nulled_&i */
       declare hash h&i(dataset: "work.nulled_&i");

         h&i..definekey("matchkey"); /* define the key as matchkey */
         h&i..definedata("&&var&i"); /* define the data as &&var&i */
         h&i..definedone(); /* done defining the hash object */
      %end;
    end;

    set master;


    /* use the &ttlcol hash objects to locate the values for the variables that are
    now nulled where necessary */

    %do i=1 %to &ttlcol;
      rc = h&i..find(key:matchkey); /* return code for hash method */

      if rc ne 0 then do; /* if lookup failed, assign a special missing value */
      %if &&type&i eq num %then %do;
        &&var&i = .A;
      %end;
      %else %do;
        &&var&i = 'Z';
      %end;
      end;
      else do;
        /* updates the variable &&var&i from hash object h&i using matchkey */
        h&i..find(key:matchkey);
      end;
      label &&var&i = "&&lbl&i";
    %end;
  run;

  proc datasets lib=work;
    delete nulled: ;
  run; quit;
%mend;

%createh
```

COMPARISON OF THE TWO METHODS

Either method gives the desired output, but the processing is quite different. Depending on your system settings, data set and/or server load, you may find one more advantageous than the other. Using PROC FREQ, significant disk space is taken up by utility files that are written to the work directory. If space is limited and there are many unique values for the variables, you may easily run out of space. In the PROC FREQ solution, we need to read multiple data sets while merging the nulled fields together by the match key to create the final data set and this takes a significant amount of I/O operations. This solution is cleaner to read and uses basic procedures, but may take longer to process than the hash solution.

The hash solution is different because most of the processing is done in memory. In the DATA step, the first./last. by variables count the variable values. Then we lookup these values by loading the count data sets into memory once using a hash object and the FIND() method to lookup the values that must be nulled out. This is faster than the time associated with a DATA step match-merge which reads through the data sets sequentially. In the last step, we create the final data set by match key using a hash object for each nulled data set, which again happens in memory and uses less disk space.

CONCLUSION

There are times when system constraints or large data sets force us to learn new programming techniques. The hash object is extremely useful for lookups, eliminating the need to rely on the DATA step match-merge or PROC SQL joins and reducing run time due to its use of memory instead of disk space. This technique provided the optimal solution to depersonalize the client file in a timely manner for this particular project as PROC FREQ quickly ran out of space. The memory settings were managed carefully to accommodate the multiple large hash objects and the columns were partitioned because we only had so much memory allocated.

A few sample data sets were run through both methods and disk space and real time measurements were compared (see Table 7). Surprisingly both methods used a similar amount of disk space but, depending on the structure of the data set, one method was faster than the other. Please keep in mind that system load had a significant impact on the run times so this was just for demonstration purposes only and multiple runs would have been tested, if time allowed, for accurate benchmarking.

Data Set	Data Set Structure	#Vars	#Obs	FREQ work space (kb)	FREQ utility space (kb)	FREQ real time (hh:mm:ss)	HASH work space (kb)	HASH utility space (kb)	HASH real time (hh:mm:ss)
rawreturn	all numeric fields, many unique values	999	20,000	820136	0	00:05:18	823104	0	00:03:07
sample2	mixed numeric and character, PII data with many unique values	38	5,000,000	9661816	240123904	00:07:19	9661888	240123904	00:19:10
sample3	mixed numeric and character, attribute data mixed with PII	515	3,094,742	63589388	173408256	19:47:28	63591492	173408256	18:54:04

Table 7. Method tests for disk space and real run time

REFERENCES

Genomics REGED1 Data Set: http://www.causality.inf.ethz.ch/challenge.php?page=datasets#cont

Lafler, Kirk Paul. 2011. “An Introduction to SAS Hash Programming Techniques”, Proceedings of the 2011 South East SAS Users Group (SESUG) Conference. Available at http://analytics.ncsu.edu/sesug/2011/BB08.Lafler.pdf

Farris, Jason. 2010. “Working with SAS Hash Objects: Automation, Tips and Pitfalls”, Proceedings of the 1010 Western Users of SAS Software Users Group (WUSS) Conference. Available at http://www.wuss.org/proceedings10/coders/3006_3_COD-Farris.PDF

SAS Institute. “Hash Object Tip Sheet,” Available at http://support.sas.com/rnd/base/datastep/dot/hash-tip-sheet.pdf

SAS Institute. 2007. “SAS Certification Prep Guide: Advanced Programming for SAS 9”. 577-589. Cary, NC: SAS Publishing

ACKNOWLEDGMENTS

I would like to acknowledge Experian for allowing me the time and resources to share this paper with the attendees of SESUG.

CONTACT INFORMATION

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

Name: Renee Canfield
Enterprise: Experian
E-mail: renee.canfield@experian.com