Paper 1122-2017

Using Hash Tables for Creating Electronic Codebooks

Raghav Adimulam, Westat

ABSTRACT

In projects that span multiple years, (e.g., longitudinal studies) there are usually thousands of new variables introduced at the end of every year or at the end of each phase of the project. These variables usually have the same stem or core as the previous year's variables. However, they differ only in a digit or two that usually signifies the year number of the project. Therefore, every year, there is this extensive task of comparing thousands of new variables to older variables for the sake of carrying forward key database elements corresponding to the previously defined variables. These elements can include the length of the variable, data type, format, discrete or continuous flag, and so on.

In our SAS program, hash objects are efficiently used to cut down not only the time lapsed, but also the number of DATA and PROC steps used to accomplish the task. The resulting clean and lean code is much easier to understand. A macro is used to create the data set containing new and older variables. For a specific new variable, the FIND method in hash objects is used in a loop to find the match to the most recent older variable. Overall, what was originally taking about a dozen PROC SQL steps is now brought down to a single data step using the hash tables' idea.

INTRODUCTION

For the past five years, the author has been involved in a multi-year longitudinal study that collects and publishes tens of thousands of variables yearly. The study uses Electronic Codebook software to allow users to identify and examine variables available in the published data file. Each of these variables has a fixed set of attributes assigned to it in the previous years of the study, like label, length, data type, formats, discrete/continuous flags, number of decimal places, description text, ID, etc. Tens of thousands of variables are collected each year over the duration of the study. In the present scenario, over a period of 6 years, the study has produced around 14000 variables. So, in the new year of study, i.e. in the seventh year, a new batch of around 2300 variables are being introduced.

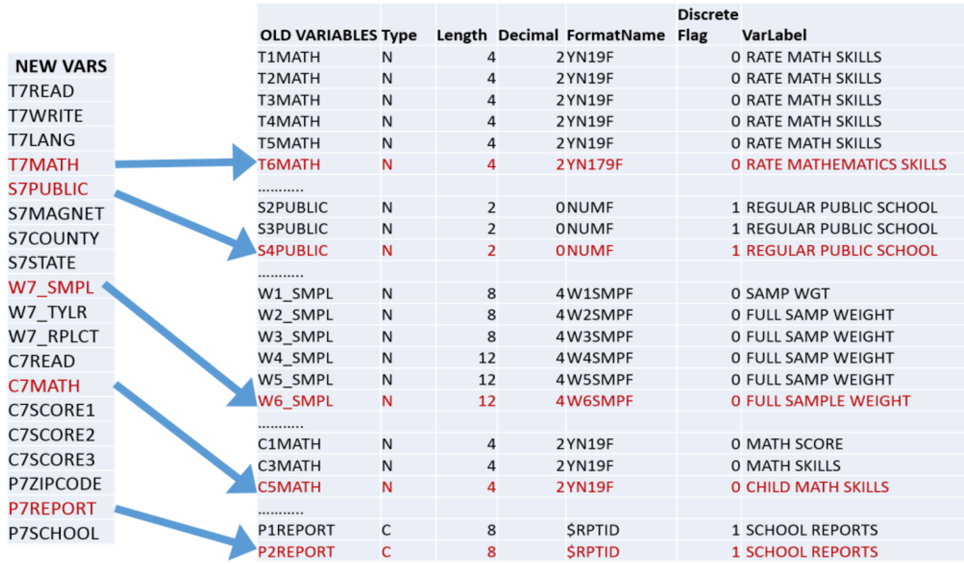
Variables in the study are prefixed in the format **Xn**. The **X** is always a character and can be either a P, C, T, S or W. The **n** is always a number and can be one of 1, 2, 3, 4, 5 or 6 depending on the year of the study that the variable belongs to. The current batch of 2300 variables all have **n** equal to 7. The rest of the variable [substr(var,3)] after stripping the first two characters is herewith referred to as the *core* or *stem* of the variable. An example of the variables used in the study are shown in Figures 1 and 2.

PROBLEM STATEMENT

Our task is now to find as many possible matches of the 2300 new variables to that of the existing set of 14000 old variables. The new variables are present in an Excel file called NewVars.xlsx, whereas the existing 14000 variables are present in an Access database metadata, wherein they are stored along with the other attributes related to the variables. The desired matching for a new variable should be the older variable from the most recent year (as shown in Figure 1).

The traditional approach of matching involves using Proc SQL. The Proc SQL matching method takes at least 12 clauses for the whole process. First we extract all the X6 variables from the Meta Access database and match them with NewVars excel file to get matches and non-matches. Then we take the non-matches from the previous step and match them to the X5 variables. This process of taking the non-matches from the previous step and matching to the next round of variables will continue until all rounds are matched up, which is until the X1 variables.

Finally, the matches from all the steps above are accumulated to get the grand total of matches. It is expected that the matches found are close to 90%. The non-matches(~200) are the ones that are sent to the project team for a decision on the attributes assignment.



OLD VARIABLES	Type	Length	Decimal	FormatName	Discrete Flag	VarLabel
T1MATH	N	4	2	YN19F	0	RATE MATH SKILLS
T2MATH	N	4	2	YN19F	0	RATE MATH SKILLS
T3MATH	N	4	2	YN19F	0	RATE MATH SKILLS
T4MATH	N	4	2	YN19F	0	RATE MATH SKILLS
T5MATH	N	4	2	YN19F	0	RATE MATH SKILLS
T6MATH	N	4	2	YN179F	0	RATE MATHEMATICS SKILLS
....						
S2PUBLIC	N	2	0	NUMF	1	REGULAR PUBLIC SCHOOL
S3PUBLIC	N	2	0	NUMF	1	REGULAR PUBLIC SCHOOL
S4PUBLIC	N	2	0	NUMF	1	REGULAR PUBLIC SCHOOL
....						
W1_SMPL	N	8	4	W1SMPF	0	SAMP WGT
W2_SMPL	N	8	4	W2SMPF	0	FULL SAMP WEIGHT
W3_SMPL	N	8	4	W3SMPF	0	FULL SAMP WEIGHT
W4_SMPL	N	12	4	W4SMPF	0	FULL SAMP WEIGHT
W5_SMPL	N	12	4	W5SMPF	0	FULL SAMP WEIGHT
W6_SMPL	N	12	4	W6SMPF	0	FULL SAMPLE WEIGHT
....						
C1MATH	N	4	2	YN19F	0	MATH SCORE
C3MATH	N	4	2	YN19F	0	MATH SKILLS
C5MATH	N	4	2	YN19F	0	CHILD MATH SKILLS
....						
P1REPORT	C	8		$RPTID	1	SCHOOL REPORTS
P2REPORT	C	8		$RPTID	1	SCHOOL REPORTS

Figure 1. Desired mapping of the new variables to the older variables

THE HASH SOLUTION

The alternative to the Proc SQL solution is the Hash Object solution that accomplishes the task with just two Data steps – One with Hash Objects and another with a Macro. The workflow of the Hash program is given in Figure 2.

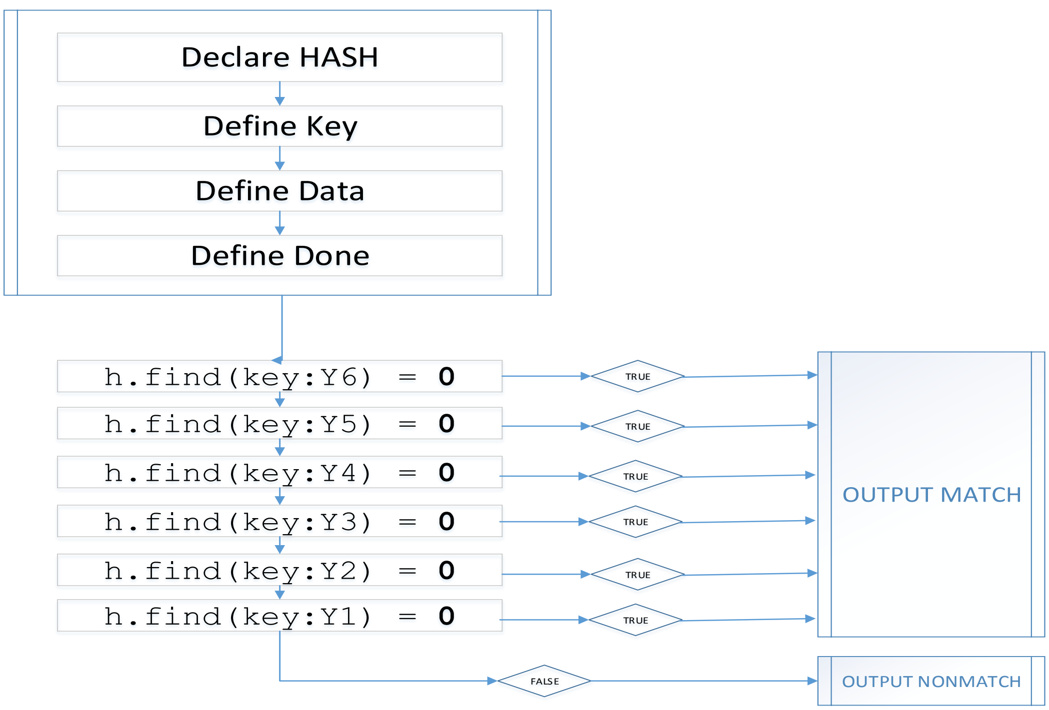


Figure 2. Workflow of the Hash program

```
/**************************************************************************
Hash_Matching_Program.sas by R.Adimulam
**************************************************************************/

option mlogic mprint validvarname=upcase nofmterr compress=binary noxwait
fullstimer ps=55 ls=120 nocenter ;

libname newyr "H:\MY DOCUMENTS\SASGlobalForum\NewVars.xlsx"  access=readonly;
libname oldvar "H:\MY DOCUMENTS\SASGlobalForum\ALLVAR.accdb" access=readonly;

data _null_; x=sleep(2); run;

/* Import ECB metadata from the Access table containing Older variable names*/
data Access;
 set oldvar.variables
      (keep=FieldName DescriptionText Length Type Decimal DiscreteFlag
      FormatName VarLabel);
run;

/* Import New variable names stored in an Excel file to SAS dataset NEWVAR */
%macro varn;
data newvar ;
  length var_name $30 F1 - F6 $3 Core Y1 - Y6 $30 ;
  set newyr."Sheet1$"n ;
    if not missing(var_name) and substr(var_name,2,1)='7' ;
    core = substr(var_name,3) ;
    %do i=1 %to 6;
        F&i. = cats(substr(var_name,1,1),&i.) ;
        if not missing(F&i.) then Y&i. = cats(F&i.,core) ;
    %end;
run ;
%mend;
%varn;

/* Use hash objects to find matches and nonmatches */
data match (keep=var_name FieldName DescriptionText Length
                   Type Decimal DiscreteFlag FormatName VarLabel FlagMatch)
    nonmatch (keep=var_name) ;
   if 0 then set Access;
   if _n_=1 then do;
     declare hash h(DATASET:"Access");
      h.definekey('FieldName');
      h.defineData('FieldName','DescriptionText','Length','Type',
                   'Decimal','DiscreteFlag','FormatName','VarLabel');
      h.defineDone();
  end;
  set Newvar ;
 if    h.find(key:Y6) = 0 then do; FlagMatch=6; output match; end;
 else if h.find(key:Y5) = 0 then do; FlagMatch=5; output match; end;
 else if h.find(key:Y4) = 0 then do; FlagMatch=4; output match; end;
 else if h.find(key:Y3) = 0 then do; FlagMatch=3; output match; end;
 else if h.find(key:Y2) = 0 then do; FlagMatch=2; output match; end;
 else if h.find(key:Y1) = 0 then do; FlagMatch=1; output match; end;
 else output nonmatch;
run;

libname _all_ clear;
```

The VARN Macro prepares the dataset that will be subsequently used by the Hash Objects Data Step. The macro contains a Data Step reading the Excel file NewVars using the Newyr Libname engine. Six new variables F1 – F6 are created containing the two-byte alpha-numeric prefix Xn, where n indicates the Year number as discussed previously. The core of the variables is extracted using the formula `core = substr(var_name,3) ;`

Another set of six new variables are created, called Y1 – Y6, using the CATS function to concatenate the core with F1 – F6 variables respectively.

```
%do i=1 %to 6;
    F&i. = cats(substr(var_name,1,1),&i.) ;
    if not missing(F&i.) then Y&i. = cats(F&i.,core) ;
%end;
```

The %DO - %END clause in the macro eliminates the redundancies in the code by creating the new F1 – F6 and Y1 – Y6 variables with minimal macro code. At the end of the execution of the VARN macro, we create a dataset called NEWVAR that is shown below in *Figure 3.*

var_name	F1	F2	F3	F4	F5	F6	Core	Y1	Y2	Y3	Y4	Y5	Y6	LABEL
T7READ	T1	T2	T3	T4	T5	T6	READ	T1READ	T2READ	T3READ	T4READ	T5READ	T6READ	T7 RATE READING SKILLS
T7WRITE	T1	T2	T3	T4	T5	T6	WRITE	T1WRITE	T2WRITE	T3WRITE	T4WRITE	T5WRITE	T6WRITE	T7 RATE WRITING SKILLS
T7LANG	T1	T2	T3	T4	T5	T6	LANG	T1LANG	T2LANG	T3LANG	T4LANG	T5LANG	T6LANG	T7 RATE ORAL LANGUAGE SKILLS
T7MATH	T1	T2	T3	T4	T5	T6	MATH	T1MATH	T2MATH	T3MATH	T4MATH	T5MATH	T6MATH	T7 RATE MATHEMATICS SKILLS
T7SCIENCE	T1	T2	T3	T4	T5	T6	SCIENCE	T1SCIENCE	T2SCIENCE	T3SCIENCE	T4SCIENCE	T5SCIENCE	T6SCIENCE	T7 RATE SCIENCE SKILLS
T7SOCIAL	T1	T2	T3	T4	T5	T6	SOCIAL	T1SOCIAL	T2SOCIAL	T3SOCIAL	T4SOCIAL	T5SOCIAL	T6SOCIAL	T7 RATE SOC STUDIES SKILLS
S7PUBLIC	S1	S2	S3	S4	S5	S6	PUBLIC	S1PUBLIC	S2PUBLIC	S3PUBLIC	S4PUBLIC	S5PUBLIC	S6PUBLIC	S7 REGULAR PUBLIC SCHOOL
S7MAGNET	S1	S2	S3	S4	S5	S6	MAGNET	S1MAGNET	S2MAGNET	S3MAGNET	S4MAGNET	S5MAGNET	S6MAGNET	S7 PUBLIC MAGNET SCHOOL
S7CHARTR	S1	S2	S3	S4	S5	S6	CHARTR	S1CHARTR	S2CHARTR	S3CHARTR	S4CHARTR	S5CHARTR	S6CHARTR	S7 CHARTER SCHOOL
S7RELIG	S1	S2	S3	S4	S5	S6	RELIG	S1RELIG	S2RELIG	S3RELIG	S4RELIG	S5RELIG	S6RELIG	S7 RELIGIOUS SCHOOL
S7EARLY	S1	S2	S3	S4	S5	S6	EARLY	S1EARLY	S2EARLY	S3EARLY	S4EARLY	S5EARLY	S6EARLY	S7 EARLY CHILDHOOD CENTER
S7SCHOOL	S1	S2	S3	S4	S5	S6	SCHOOL	S1SCHOOL	S2SCHOOL	S3SCHOOL	S4SCHOOL	S5SCHOOL	S6SCHOOL	S7 SCHOOL ID (PUBLIC)
S7COUNTY	S1	S2	S3	S4	S5	S6	COUNTY	S1COUNTY	S2COUNTY	S3COUNTY	S4COUNTY	S5COUNTY	S6COUNTY	S7 SCHOOL COUNTY CODE
S7STATE	S1	S2	S3	S4	S5	S6	STATE	S1STATE	S2STATE	S3STATE	S4STATE	S5STATE	S6STATE	S7 SCHOOL STATE CODE
S7ZIPCODE	S1	S2	S3	S4	S5	S6	ZIPCODE	S1ZIPCODE	S2ZIPCODE	S3ZIPCODE	S4ZIPCODE	S5ZIPCODE	S6ZIPCODE	S7 SCHOOL ZIP CODE
S7CENSUS	S1	S2	S3	S4	S5	S6	CENSUS	S1CENSUS	S2CENSUS	S3CENSUS	S4CENSUS	S5CENSUS	S6CENSUS	S7 SCHOOL CENSUS CODE
W7_SMPL	W1	W2	W3	W4	W5	W6	_SMPL	W1_SMPL	W2_SMPL	W3_SMPL	W4_SMPL	W5_SMPL	W6_SMPL	W7 FULL SAMP WGT
W7_PRIM	W1	W2	W3	W4	W5	W6	_PRIM	W1_PRIM	W2_PRIM	W3_PRIM	W4_PRIM	W5_PRIM	W6_PRIM	W7 PRIMARY SAMPLING
W7_TYLR	W1	W2	W3	W4	W5	W6	_TYLR	W1_TYLR	W2_TYLR	W3_TYLR	W4_TYLR	W5_TYLR	W6_TYLR	W7 TAYLOR SERIES
W7_RPLCT	W1	W2	W3	W4	W5	W6	_RPLCT	W1_RPLCT	W2_RPLCT	W3_RPLCT	W4_RPLCT	W5_RPLCT	W6_RPLCT	W7 REPLICATE WGT 1
C7READ	C1	C2	C3	C4	C5	C6	READ	C1READ	C2READ	C3READ	C4READ	C5READ	C6READ	C7 READ SKILLS
C7MATH	C1	C2	C3	C4	C5	C6	MATH	C1MATH	C2MATH	C3MATH	C4MATH	C5MATH	C6MATH	C7 MATH SKILLS
C7SCORE1	C1	C2	C3	C4	C5	C6	SCORE1	C1SCORE1	C2SCORE1	C3SCORE1	C4SCORE1	C5SCORE1	C6SCORE1	C7 TEST SCORE 1
C7SCORE2	C1	C2	C3	C4	C5	C6	SCORE2	C1SCORE2	C2SCORE2	C3SCORE2	C4SCORE2	C5SCORE2	C6SCORE2	C7 TEST SCORE 2
C7SCORE3	C1	C2	C3	C4	C5	C6	SCORE3	C1SCORE3	C2SCORE3	C3SCORE3	C4SCORE3	C5SCORE3	C6SCORE3	C7 TEST SCORE 3
C7SCORE4	C1	C2	C3	C4	C5	C6	SCORE4	C1SCORE4	C2SCORE4	C3SCORE4	C4SCORE4	C5SCORE4	C6SCORE4	C7 TEST SCORE 4
C7SCORE5	C1	C2	C3	C4	C5	C6	SCORE5	C1SCORE5	C2SCORE5	C3SCORE5	C4SCORE5	C5SCORE5	C6SCORE5	C7 TEST SCORE 5
P7CENSUS	P1	P2	P3	P4	P5	P6	CENSUS	P1CENSUS	P2CENSUS	P3CENSUS	P4CENSUS	P5CENSUS	P6CENSUS	P7 HOME CENSUS TRACT CODE
P7ZIPCODE	P1	P2	P3	P4	P5	P6	ZIPCODE	P1ZIPCODE	P2ZIPCODE	P3ZIPCODE	P4ZIPCODE	P5ZIPCODE	P6ZIPCODE	P7 HOME ZIP CODE
P7REPORT	P1	P2	P3	P4	P5	P6	REPORT	P1REPORT	P2REPORT	P3REPORT	P4REPORT	P5REPORT	P6REPORT	P7 SCHOOL REPORTS SENT
P7SCHOOL	P1	P2	P3	P4	P5	P6	SCHOOL	P1SCHOOL	P2SCHOOL	P3SCHOOL	P4SCHOOL	P5SCHOOL	P6SCHOOL	P7 SCHOOL ASSIGNED
P7CONFNO	P1	P2	P3	P4	P5	P6	CONFNO	P1CONFNO	P2CONFNO	P3CONFNO	P4CONFNO	P5CONFNO	P6CONFNO	P7 PARENT TEACHER CONF

Figure 3. New variables (NEWVAR) table arranged for Hash processing

The last data step in the program uses the Hash Objects to create two datasets called MATCH and NONMATCH by keeping only the relevant fields in them.

In the data step, the first IF statement, which is highlighted below, is a non-executable statement because of the presence of 0 in the IF clause. However, during compile-time, the SAS compiler reads the Access table's metadata of the variables and adds it to the PDV, thereby avoiding having to use the LENGTH statement to define the variables' metadata.

```
if 0 then set Access;
```

The `declare hash` statement creates the Hash object 'H' in memory and loads the hash object from the dataset ACCESS.

```
declare hash h(DATASET:"Access");
```

The `definekey` method defines a set of hash object keys, which is FieldName in this case.

The `defineData` method defines the data elements - DescriptionText, Length, Type, Decimal, DiscreteFlag, FormatName and VarLabel - to be stored in the hash object.

The `defineDone` method indicates that the key and data definitions are complete.

```
 h.definekey('FieldName');
h.defineData('FieldName','DescriptionText','Length','Type',
             'Decimal','DiscreteFlag','FormatName','VarLabel');
h.defineDone();
```

All the above methods are executed only once at _n_=1 which is the first iteration of the data step.

The `find()` method determines whether the given key has been stored in the hash object 'H'. It will not need any argument tags if it is using the current value of the *FieldName* column, since we defined the hash object with *FieldName* column as the key. However, in our case, the name of the variable to lookup (Y1 – Y6 variables) are not the same as the name of the key item in the hash. Therefore, we use the `key:` argument tag followed by one of the variable names.

In the Hash IF..THEN..ELSE clause, we first try to find matches of the New variables to any variables from the immediately preceding year, which is year 6. Then from there onwards we use the ELSE condition to go in descending order of the year number as in 5, then 4, then 3, then 2 and finally year 1.

```
if        h.find(key:Y6) = 0 then do; FlagMatch=6; output match; end;
 else if h.find(key:Y5) = 0 then do; FlagMatch=5; output match; end;
 else if h.find(key:Y4) = 0 then do; FlagMatch=4; output match; end;
 else if h.find(key:Y3) = 0 then do; FlagMatch=3; output match; end;
 else if h.find(key:Y2) = 0 then do; FlagMatch=2; output match; end;
 else if h.find(key:Y1) = 0 then do; FlagMatch=1; output match; end;
 else output nonmatch;
```

We start the search for the older variables by first IF statement using the `key:Y6`. If the key Y6 is found in the hash object, the data variables are updated and the return code is set to zero. In addition, the *FlagMatch* variable is set to the value 6 and the variable values are output to the dataset MATCH. If the key is not found, the return code is non-zero and the ELSE statement looks for the next key match in Y5 and so on in the order (Y4 Y3 Y2 Y1). If none of the Yn keys are found, the remaining new variable names are sent to the dataset NONMATCH, which was found to have 219 variable names out of a total 2300 new variable names that needed matching.

The purpose of the *FlagMatch* variable is to be an indicator of the year the new variable found the match in. The number of matches from each year can be seen in Table 1.

```
proc freq data=match; tables FlagMatch; run;
```

FLAGMATCH	Frequency	Cumulative Frequency
1	1	1
2	28	29
4	40	69
5	142	211
6	1870	2081
Non-matches = 219		

Table 1: Frequency of the FlagMatch variable.

CONCLUSION

Hash tables are powerful tools when performing a lot of match-merging between the SAS® data sets. In the paper, both the HASH solution and the traditional Proc SQL solution are given so that the user may get the comparative sense of how both these solutions look side-by-side in comparison and how many lines of code can be saved by adopting the Hash object techniques.

Too often, the problem with programmers embracing Hash solutions is the absence of this stark comparison of listing the HASH code route alongside the traditional alternative of DATA/PROC steps. The table below illustrates this point. The first column is the last Data step in the Hash solution that replaced all of the code in second column – all of the twelve Proc SQL clauses and one data step.

Equivalent HASH object solution	Equivalent Proc SQL code solution

Data match nonmatch;
if 0 then set Access;
if _n_=1 then do;
declare hash h(DATASET:"Access");
h.definekey('FieldName');
h.defineData('Text');
h.defineDone();
end;
set Newvar;
	/** Match Access DB with newVar and get match_6 and nomatch_6 */
	proc sql noprint ;

if h.find(key:**Y6**) = 0 then do; FlagMatch=**6**; output match; end;	create table match_6 as select distinct a.*,b.* from Access a, newVar b where a.FieldName = b.**Y6**; create table nonmatch_6 as select * from newVar where var_name not in (select var_name from match_6) ;
else if h.find(key:**Y5**) = 0 then do; FlagMatch=**5**; output match; end;	/** Match Access DB with nonmatch_6 and get match_5 and nonmatch_5 */ create table match_5 as select distinct a.*,b.* from Access a, nonmatch_6 b where a.FieldName = b.**Y5**; create table nonmatch_5 as select * from nonmatch_6 where var_name not in (select var_name from match_5) ;
else if h.find(key:**Y4**) = 0 then do; FlagMatch=**4**; output match; end;	/** Match Access DB with nonmatch_5 and get match_4 and nonmatch_4 */ create table match_4 as select distinct a.*,b.* from Access a, nonmatch_5 b where a.FieldName = b.**Y4**; create table nonmatch_4 as select * from nonmatch_5 where var_name not in (select var_name from match_4) ;

`else if h.find(key:Y3) = 0 then do;`
`FlagMatch=3; output match; end;`
	`/** Match Access DB with nonmatch_4 and`
	`get match_3 and nonmatch_3 */`
	`create table match_3 as`
	`    select distinct a.*,b.*`
	`      from Access a, nonmatch_4 b`
	`      where a.FieldName = b.Y3;`
	
	`create table nonmatch_3 as`
	`    select * from nonmatch_4`
	`    where var_name not in`
	`    (select var_name from match_3) ;`

`else if h.find(key:Y2) = 0 then do;`
`FlagMatch=2; output match; end;`
	`/** Match Access DB with nonmatch_3 and`
	`get match_2 and nonmatch_2 ;*/`
	`create table match_2 as`
	`    select distinct a.*,b.*`
	`      from Access a, nonmatch_3 b`
	`      where a.FieldName = b.Y2;`
	
	`create table nonmatch_2 as`
	`    select * from nonmatch_3`
	`    where var_name not in`
	`    (select var_name from match_2) ;`

`else if h.find(key:Y1) = 0 then do;`
`FlagMatch=1; output match; end;`
 	
`else output nonmatch;`
`run ;`
	`/** Match Access DB with nonmatch_2 and`
	`get match_1 and nonmatch_1 */`
	`create table match_1 as`
	`    select distinct a.*,b.*`
	`      from Access a, nonmatch_2 b`
	`      where a.FieldName = b.Y1;`
	
	`create table nonmatch as`
	`    select * from nonmatch_2`
	`    where var_name not in`
	`    (select var_name from match_1) ;`
	`quit ;`

	`/* Merge all the match datasets to get`
	`overall matches */`
	`data MATCH ;`
	`set match_6 match_5 match_4 match_3`
	`match_2 match_1 ;`
	`run ;`

Table 2: HASH approach *vs* PROC SQL approach

REFERENCES

SAS® 9.4 Component Objects: Reference, Third Edition; Dictionary of Hash and Hash Iterator Object Language Elements https://support.sas.com/documentation/cdl/en/lecompobjref/69740/HTML/default/viewer.htm

Dorfman, Paul M.; Vyverman, Koen. 2009. "The SAS® Hash Object in Action." SAS Global Forum 153-2009, Washington DC. http://support.sas.com/resources/papers/proceedings09/153-2009.pdf

Eberhardt, Peter "The SAS® Hash Object: It's Time To .find() Your Way Around" SAS Global Forum 151-2010. http://support.sas.com/resources/papers/proceedings10/151-2010.pdf

ACKNOWLEDGMENTS

The author would like to thank Michael Raithel for the suggestions and critiques which were greatly appreciated.

RECOMMENDED READING

- Dorfman, A. H. and R. Valliant. 1993. "Quantile Variance Estimators in Complex Surveys." *Proceedings of the Survey Research Methods Section*, 866–871. Alexandria, VA: American Statistical Association.

CONTACT INFORMATION

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

Raghav Adimulam
Westat
1600 Research Blvd
Rockville, MD 20850
adimulam@gmail.com