Renaming Variables for Source Data Residing Outside of a SAS® Data Set
Barbara Morse, Anthem Blue Cross Blue Shield, South Portland, ME

ABSTRACT
What do you do when the table column names of your non-SAS source data change and you don’t have the time or desire to change your library of existing SAS code? This paper will suggest two possible solutions for handling the renaming without making large changes to existing code.

INTRODUCTION
HEDIS is a set of standardized performance measures that assess the quality and allow for the comparison of health plans. It is used in health plan accreditation and includes measures of quality, access to care, patient satisfaction, utilization, and health plan management. We do HEDIS reporting and analysis covering three health plans. Our data resides in a Teradata data warehouse using a star-schema model. Because of the volume of data and the large number of tables that we would need to access on a regular basis, direct access to the data with SAS has not been an efficient solution. Instead, our IT support area builds separate claims and enrollment extract tables that include the variables and timeframes that we need. These are stored as Teradata tables. We use SAS/ACCESS to Teradata with LIBNAME statements to access the tables. This solution has worked well and we have developed a large library of SAS code for producing the 50+ HEDIS measures and related analyses.

PROBLEM
Due to new standards our IT support informed us that our extracts must use the standard warehouse naming conventions and that they could no longer support the short table column names that we have become familiar with and upon which all of our SAS code is based. Renaming variables in a SAS data set is not a difficult task but we do not have the space to store permanent SAS versions of the Teradata data to hold the renamed variables. Changing the variable names in our SAS code was, of course, an option but we chose to look for another solution in order to avoid the potential for mistakes and confusion during a period where tight deadlines allowed no room for error in making significant code changes. Following are two solutions that we explored.

SOLUTION 1. CREATE A SQL VIEW
A straightforward approach is to include a PROC SQL step to build a view of each Teradata table which replaces each new column name with a column-name alias that matches the existing variable name used in our code.

```sql
proc sql;
   connect to teradata as hedis;
   create view clms_2005 as select *
       from connection to hedis
       (select
          mbr_certid as bcbsid,
          clm_birth_date as clmdob,
          clm_type as clmtype,
          discharge_status as dschstat,
          ...
          place_of_service as place,
          icd9_proc_code as proccode
       from prod.clms_2005);
   disconnect from hedis;
quit;
```

This code can be easily inserted with a %INCLUDE statement. In testing, we found that although this solution worked, it was very resource intensive. A job processing one year of claims which previously took 47 minutes now took over four and a half hours to finish. We have successfully used Teradata views before so it is my expectation that if this renaming view were to reside on the Teradata server, this could be a more efficient solution. However, in our case, the Teradata view was not going to be an option.

SOLUTION 2. BUILD RENAME MACRO VARIABLE LISTS USING INTO:
Because we must access our Teradata tables for each program that is run, we try to minimize resources by limiting the number of columns (variables) to be read from the data tables. The list of variables to be kept changes depending on the job. Our existing code structure uses a %LET statement to create a macro variable list of the required variables. We use a KEEP= option with the macro variable list for each table accessed to specify the variables needed in any particular job. For example:

```sas
libname indata teradata user=&aeuid
   password=&aepwd tdpid=dwprod
   schema=hedis;

* Teradata tables;
%let inclma = indata.hedis_clms_2004;
%let inclmb = indata.hedis_clms_2005;

* variables to keep;
%let clmvars= bcbsid firstdt lastdt
   revcode provid clmdob product
   plancd pos dx1-dx5 admtdt cpt
   region icdproc typebill clmtype;

data ambsurg;
   set &inclma(keep=&clmvars)
      &inclmb(keep=&clmvars);
   where ...
```

Building upon this existing structure, the strategy for this
solution is to replace the macro variable &clmvars in the KEEP = option with two new variable strings: one a list of the new column names &newvars, and the second, a ‘RENAME =’ list of new to old variable names like:

```sas
set &inclma(keep=&newvars rename= ( newvar1=oldvar1 ... ));
```

The &newvars variable will contain the ‘new’ variable names that correspond to the old variables in &clmvars. These are the variables that are passed to Teradata. The resulting names in the SAS data set will be the ‘old’ variable names from the RENAME = list and will match those used in the existing code. The WHERE statement will continue to refer to the old variable names.

The most confusing thing about this solution is keeping track of which set of variable names is being used as well as whether the variable names refer to actual variables in the data set or to observations of a data set. I will refer to the new variable names as ‘TABLE’ variables and the existing SAS code variables as ‘CODE’ variables.

Using SASHELP:VCOLUMNS and Excel, I created a variable name reference SAS data set SAVE.COLNAMES with an observation for each TABLE variable appearing in my tables along with its corresponding CODE variable name. I also included the varnum variable available from VCOLUMNS (see Figure 1. below.) From this data set a permanent SAS format, $CHGFMT, was built to use in translating each CODE variable name to its corresponding TABLE variable name.

I planned to make use of the INTO: feature of PROC SQL to build new macro variable lists but there were two issues that needed to be resolved first. Any existing variable name range in &clmvars needed to be expanded to its individual variables. For example, diag1-diag5 would need to be expanded to diag1 diag2 diag3 diag4 diag5. Also, because the SAS format and &clmvars are created independently of each other, there needed to be some check that the variable names entered in &clmvars are actually valid names existing in the format.

Because the ‘KEEP=&clmvars’ option in the existing code structure already accomplishes these two things, I chose to transpose SAVE.COLNAMES to create a data set where the CODE variable name observations become data set variables. Using &clmvars with this data set results in a data set with any expanded variable name ranges and the set of selected variables. An error condition will result if there is any invalid variable name in &clmvars.

```sas
proc transpose data=save.colnames (where=(name ne ''))
   out=selectvars(keep=&clmvars);
   var varnum;
   id name;
run;
```

The data set SELECTVARS contains those CODE variables required to be kept. There is just one observation in the data set. (See Figure 2.)

Another PROC TRANSPOSE is required to turn the variable names back into data set observations. The following code accomplishes this. See Figure 3. for the results of this transpose.

```sas
proc transpose data=selectvars
   out=transtoobs (keep=_name_ rename = (_name_ = name));
run;
```

Data set TRANSTOOBS now contains one observation for each selected CODE variable. The variable names are stored in the variable ‘name’. This is what is needed to now build the macro variable lists to handle the renaming task in the following code.

```sas
proc sql noprint;
   select trim(put(name,$chgfmt.))||
      "=" || name
   into :renamestr separated by " ",
   :newvars separated by " 
   from work.transtoobs;
quit;
```

%put renames=&renamestr ;
%put keeps=&newvars ;
This code creates two macro variable lists: &renamestr is a list of the form “TABLE variable name = CODE variable name” for each selected variable, separated by spaces. &newvars is a list of the TABLE variable names with each name separated by spaces. The %PUT statements can be used to view the resulting macro variable lists.

As an example, if &clmvars = “admdt bcbsid clmdob firstdt”, then

&renamestr= “admit_date=admdt  mbr_certid=bcbsid  clm_birth_date=clmdob  begin_serv_date = firstdt”
and
&newvars= “admit_date mbr_certid clm_birth_date begin_serv_date”

To use these lists the existing code will change to:

data ambsurg;
  set &inclma(keep=&newvars rename = (&renamestr)) &inclmb((keep=&newvars rename = (&renamestr));
  where ... 

To further shorten the changes to existing code, the two macro variables could be combined into one new variable to make recoding of the existing code simpler.

%let newclmvars= keep= &newvars rename = (&renamestr); 
data ambsurg;
  set &inclma(&newclmvars) &inclmb(&newclmvars);
  where ...

In testing the code for solution 2 on the same one year of data, the processing time dropped back to 45 minutes which is approximately what the original code took and indicates that this a more efficient solution to the problem.

CONCLUSIONS

SAS offers ways around seemingly difficult roadblocks. The most obvious solution may or may not end up being the ‘right’ solution for the situation. In this example, the use of INTO: proved to be an efficient way to create macro variable lists to solve the problem of having to recode variable names for many existing SAS programs.

REFERENCES

[1] SUGI 27, Using the Magical Keyword “INTO:” in Proc SQL, by Thiru Satchi at Blue Cross and Blue Shield of Massachusetts


ACKNOWLEDGMENTS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

HEDIS is a registered trademark of the National Committee for Quality Assurance (NCQA).

Other brand and product names are trademarks of their respective companies.

Special thanks goes to Ian Whitlock for giving me the encouragement and support to write and present a paper and for assistance with segments of the solution code.

DISCLAIMER

The contents of this paper express the work and opinions of the author and do not necessarily represent recommendations or practices of Anthem Inc. The code is supplied to illustrate possible solutions with SAS and no warranty is implied that its use will fit any particular situation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

Barbara Morse
Anthem Blue Cross Blue Shield
2 Gannett Drive
South Portland, ME 04106
Work Phone: 207 822-7192
Fax: 207 822-5193
Email: Barbara.Morse@anthem.com

*****************************************************