Paper 107-2008

%HASHMERGE – A Macro to Hash When It Can, Merge When It Can't

Gregg P. Snell, Data Savant Consulting, Shawnee, KS

ABSTRACT

As documented in several previous papers[1], within certain limitations there is nothing faster than using the Data Step Component Objects, or "hashing", to merge two SAS tables. However, many programmers would like the speed improvement hashing offers but haven't taken the time to learn the new dot-notation syntax. Or, perhaps there is concern the program will run out of memory after spending extra time writing more complicated code. The old sort-n-merge may be slow, but it always works. What if there was a macro that could programmatically decide if hashing would work and generate the appropriate code either way?

%HASHMERGE was written with the SAS® 9 Macro Language Facility and DATA step Component Object Interface, or "hashing". This macro programmatically ascertains if the data to merge could be completed with hashing in the current SAS session. If so, it generates the appropriate hashing code and if not, it generates the old-fashioned sort-n-merge code.

The intended audience for this paper is any SAS programmer familiar with macros that has written more than a few DATA merges and is interested in learning and exploiting the new hashing methods.

%HASHMERGE is (almost) operating system independent and only requires the Base SAS® 9 System.

INTRODUCTION

Merging tables is an essential part of nearly all SAS programs. Unfortunately, it is often the most time consuming (execution) part as well due primarily to I/O. Basic match-merging of two or more tables requires the tables be sorted or indexed prior to merging. Hashing, on the other hand, does not have either requirement. Additionally, the internals of the hash object were designed with speedy execution in mind, so it simply runs faster. On the down side, hashing code is a bit more complex and verbose when compared to a simple match-merge. Also, there are certain limitations which preclude use of the hash object.

Wouldn't it be nice to write one piece of code and take advantage of either method? How about something as simple as:

```
%hashmerge(data=new,
           data_a=small,
           data_b=large,
           vars_b=largevar1 largevar2,
           by=keyvar,
           if=a);
```

MATCH-MERGE VS. HASH-MERGE

Consider the following comparison of fairly typical match-merge and hash-merge code:

Table 1

```
data new;
   merge small(in=a)
         large(in=b
               keep=keyvar
                    largevar1
                    largevar2);
   by keyvar;
   if a;
run;
```

```
data new;
   if 0 then set small                                      <1
                 large(keep=keyvar
                            largevar1
                            largevar2);
   declare hash h_merge(dataset:"large");
   rc = h_merge.DefineKey("keyvar");                        <2
   rc = h_merge.DefineData("largevar1",
                           "largevar2");
   rc = h_merge.DefineDone();                               <3
   drop rc;

   do while (not eof);
      set work.small end=eof;
      call missing(largevar1,
                   largevar2);                              <4
      rc = h_merge.find();
      output;                                               <5
   end;

   stop;                                                    <6
run;
```

Given the simple example above it should be clear that hash-merge code is more verbose than match-merge. Please note some of the additional complexity:

1. The hash object requires the length of variables passed to it be defined prior to creation (the declare statement). Using the "if 0 then set" is a short-cut way to specify the length of all variables rather than listing them individually in LENGTH or ATTRIB statements.
2. The hash object requires unique keys. Consequently, data is automatically de-duped when loaded into a hash object (no warnings or errors).
3. The unique key and data variables must all be explicitly defined.
4. The essence of the "merge" is inside an explicit loop.
5. Unlike match-merging, variables from the contributing table (or hash object in this case) are not set to missing when a match is not found. This must be done explicitly.
6. The "if a" condition isn't necessarily more complicated in hashing, but achieving the same result is different given how each table is handled.

%HASHMERGE IS BORN

My objective was to create a macro that would accomplish the following:

1. Determine if the table being merged could be loaded into a hash object without exceeding the current memory limit.
2. If memory is inadequate for hashing, or if the merge requires all observations from both tables ("if a or b;"), then produce match-merge code.
3. Otherwise, produce hash-merge code that hashes the table not specified by the "if a;" statement.
4. Do all of the above given only what would otherwise be coded as a match-merge.

1. IS THERE ENOUGH MEMORY TO HASH?

To answer this question, one must know: how much memory will the hashed table require and is that less than the remaining available memory in the SAS session?

From the example match-merge code in table 1, one can ascertain the following:

1. The table being created is WORK.NEW
2. Two tables are being joined
    a. WORK.SMALL using all columns
    b. WORK.LARGE using only 3 columns
3. The key variable for joining is KEYVAR
4. Keep all rows from WORK.SMALL and any rows in WORK.LARGE that match.

Since the merge requires all observations in WORK.SMALL (essentially a left-join for those preferring SQL), hash WORK.LARGE. Actually, only three variables (KEYVAR, LARGEVAR1, LARGEVAR2) from this table are required. A simple calculation of rows*variable-lengths should be sufficient to estimate the memory requirement and that information is available in the metadata tables SASHELP.VTABLE and SASHELP.VCOLUMN.

To learn more about SAS metadata, please explore the many excellent SGF/SUGI papers that address this subject. My favorite is "You Could Look It Up: An Introduction to SASHELP Dictionary Views" by Michael Davis. [2]

The view SASHELP.VTABLE will tell many rows there are in WORK.LARGE and SASHELP.VCOLUMN will tell the lengths of the three variables. Here is a sample of the SQL code to calculate how many megabytes of memory will be required for hashing, storing that value in macro variable &mem_b:

```
   select floor(y.nobs*sum(z.length)/1000000) into :mem_b
      from sashelp.vtable as y,
           sashelp.vcolumn as z
      where y.libname="&data_b_lib" and y.memname="&data_b_data" and
            upcase(z.name) in("&by_vars","&data_vars_b") and
            y.libname=z.libname and y.memname=z.memname;
```

How much memory is available to the current SAS session?. For most multi-user systems (Unix, MVS) SAS is always invoked with a value other than 0 (indicating unlimited) for memsize. On Windows, however, the default for memsize is often 0 so it is necessary to poll the operating system to detect available memory. Finally, if detecting available memory fails, set an arbitrary limit to reduce the risk of memory failure. Here is the code snippet to accomplish this task:

```
%let maxmem = %sysfunc(getoption(memsize));
%if &maxmem = 0 %then %do;
    %if &sysscp = WIN %then %do;
       filename mem pipe 'systeminfo';
       data _null_;
          infile mem lrecl=256 pad;
          input;
          if substr(_infile_,1,26)='Available Physical Memory:' then do;
             maxmem = input(compress(trim(scan(_infile_,4,' ')),','),best.);
             gb_mb_kb=scan(_infile_,-1,' ');
             /* convert to megabytes */
             if gb_mb_kb='GB' then maxmem=maxmem*1000;
             else if gb_mb_kb='KB' then maxmem=floor(maxmem/1000);
             call symput('maxmem',put(maxmem,best.));
          end;
       run;
    %end;
    %if &maxmem = 0 %then %let maxmem=500;
%end;
%put maxmem=&maxmem;
```

Of course, the currently running SAS session has already consumed *some* of the memsize allocated at invocation. Include a downward adjustment if programs begin to fail from memory constraints. Otherwise, a simple comparison of &maxmem to &mem_b is all that is needed to answer the question "*Is there enough memory to HASH?*"

2. PRODUCE MATCH-MERGE CODE

Obviously, if &mem_b is greater than &maxmem then match-merge code is the only option. However, even if memory is adequate, there are other conditions that force the match-merge option. Primarily, if the merge is coded with "if a or b;" indicating that all observations from both tables are to be kept regardless of matching. Again, this is a *good* rather than *perfect* solution as coding a hash-merge to accomplish an "or" condition is possible but much more complicated.

The macro compares &mem_b with &maxmem and also parses the passed "if" parameter for the "or" condition and, if necessary, produces the following match-merge code:

```
proc sort data=&data_a (keep=&keep_a) out=a_sorted; by &by; run;
proc sort data=&data_b (keep=&keep_b) out=b_sorted; by &by; run;
data &data;
   merge a_sorted (in=a)
         b_sorted (in=b);
   by &by;
   %if %str(&if) ne %str() %then %do;
      if &if;
   %end;
run;
```

3. PRODUCE HASH-MERGE CODE

As described earlier, hash-merge code is a bit more complicated than match-merge. Consequently, additional metadata on the table to be hashed is required. The variables representing the key for merging are already identified in the passed "by=" parameter. All other variables in this table (might be limited with "vars_b=") must be explicitly passed in the DefineData method and initialized as missing prior to each search of the hash object. Once again, SQL code utilizing SASHELP.VCOLUMN can accomplish this task.

Create a quote-comma separated list of DefineData variables as &data_vars_b (be sure to exclude the key variables):

```
select upcase(trim(name)) into :data_vars_b separated by '","'
   from sashelp.vcolumn
   where libname="&data_b_lib" and
         memname="&data_b_data" and
         upcase(name) in("&data_vars_b") and
         upcase(name) not in("&by_vars");
```

Create a comma-separated list of DefineData variables to use with the MISSING function:

```
select upcase(trim(name)) into :init_vars_b separated by ','
   from sashelp.vcolumn
   where libname="&data_b_lib" and
         memname="&data_b_data" and
         upcase(name) in("&data_vars_b") and
         upcase(name) not in("&by_vars");
```

Then put it all together as hash-merge code:

```
data &data;
   if 0 then set &data_a (keep=&keep_a)
                 &data_b (keep=&keep_b);
   declare hash h_merge(dataset:"&&data_&hashdsn");
   rc=h_merge.DefineKey("&by_vars");
   rc=h_merge.DefineData("&&data_vars_&hashdsn");
   rc=h_merge.DefineDone();
   do while(not eof);
      set &&data_&nohashdsn (keep=&&keep_&nohashdsn) end=eof;
      %if %str(&&init_vars_&hashdsn) ne %str() %then %do;
         call missing(&&init_vars_&hashdsn);
      %end;
      rc=h_merge.find();
      %if %str(&if) = &hashdsn %then if rc=0 then;
      output;
   end;
   drop rc;
   stop;
run;
```

4. CREATE THE MACRO TO LOOK LIKE STANDARD MATCH-MERGE CODE

An important objective for this macro was to write it such that using it in code would look very similar to (what I consider) standard match-merge code. For quick reference, here is the example of match-merge code from table 1:

```
data new;
   merge small(in=a)
         large(in=b
               keep=keyvar
                    largevar1
                    largevar2);
   by keyvar;
   if a;
run;
```

Here is how I coded the macro invocation:

```
%macro hashmerge(data=,
                 data_a=,
                 vars_a=,
                 data_b=,
                 vars_b=,
                 by=,
                 if=);
```

Table 2

Macro parameter	**Description**
data=	table name (with no dataset options) being created by the merge
data_a=	name of first table to merge and typically specified with the option (in=a)
vars_a=	space-separated list of variables to keep from data_a where blank indicates all
data_b=	name of second table to merge and typically specified with the option (in=b)
vars_b=	space-separated list of variables to keep from data_b where blank indicates all
by=	space-separated list of variables to match-merge the two tables
if=	valid options are: blank, a, b, a or b, a and b

LIMITATIONS/ASSUMPTIONS

1. **Only one table can be created as a result of the merge.**
2. **Only two tables can be merged at one time.**
3. **The merge must be one-one or many-one.** (recall that hash objects auto-dedupe)
4. **Table names are passed without any dataset options** (i.e. rename or keep). SAS® 9.1.3 has 38 different dataset options that could be specified in any order and in virtually any combination.
5. **The passed variables must be single variable names space-separated.** In other words, no colon modifiers or range specifications (i.e. var1: or var1-var10)
6. **All tables are SAS datasets** (i.e. no RDBMS tables)

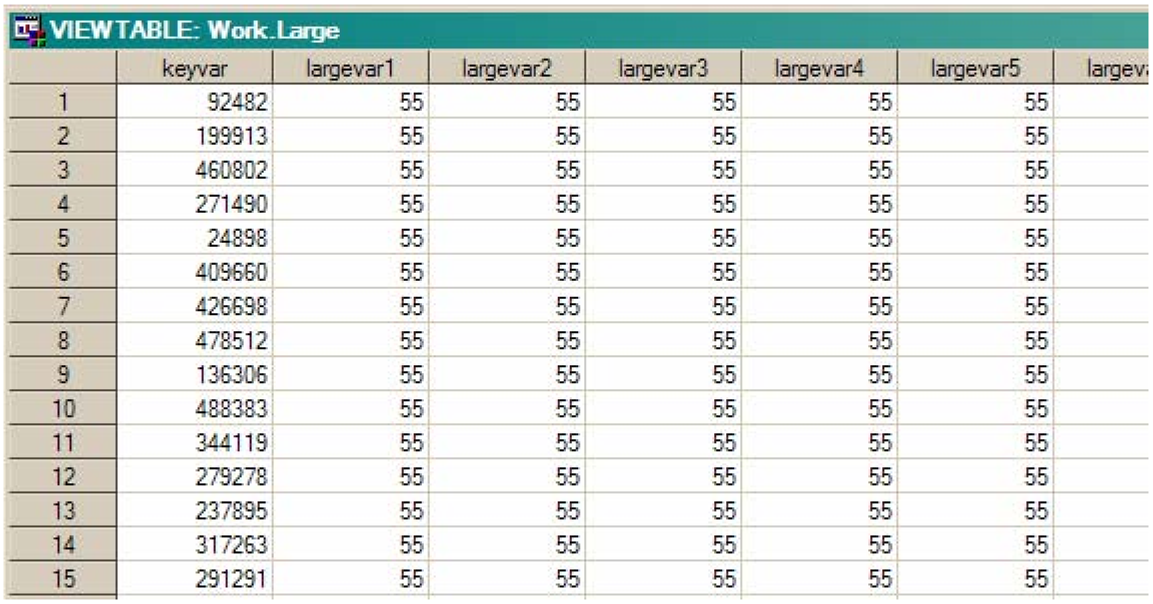
Imposing these limitations significantly reduced macro complexity while still providing basic functionality. Also, the dataset options can be implemented by creating views of the tables to be joined prior to calling this macro.

EXAMPLES

First, create sample data for testing. This code is copied from my SUGI 31 paper [3], originally adapted from a posting on SAS-L [4]:

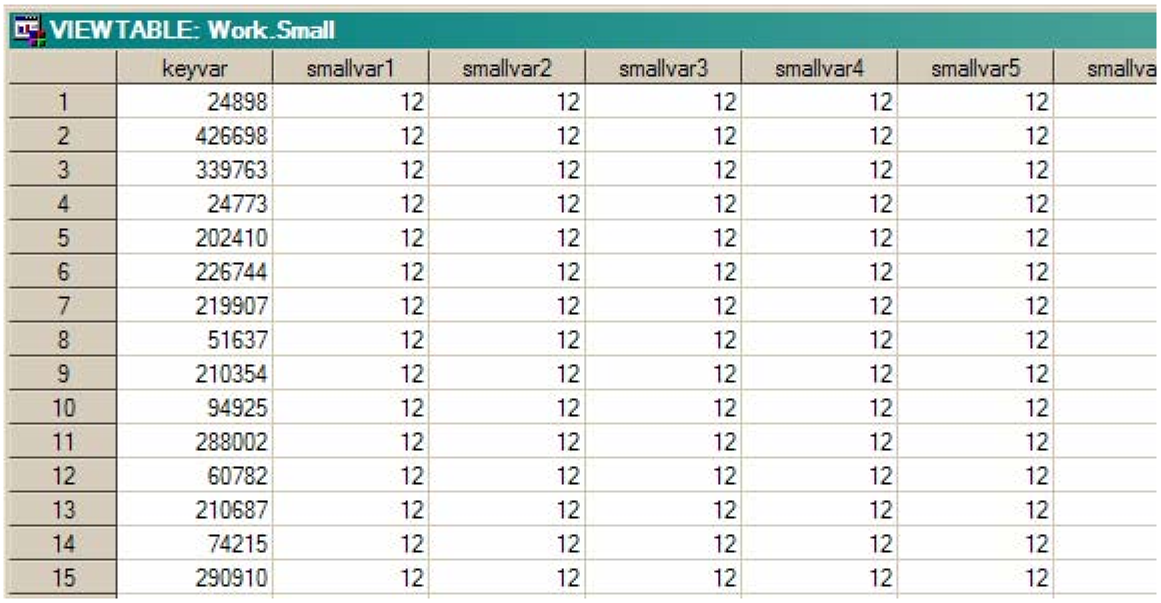
```
%let large_obs = 500000;

data work.small ( keep = keyvar small: )
     work.large ( keep = keyvar large: )
     ;
   array keys(1:500000) $1 _temporary_;
   length keyvar 8;
   array smallvar [20]; retain smallvar 12;
   array largevar [682]; retain largevar  55;
   do _i_ = 1 to &large_obs ;
      keyvar = ceil (ranuni(1) * &large_obs);
      if keys(keyvar) = ' ' then do;
         output large;
         if ranuni(1) < 1/5 then output small;
             keys(keyvar) = 'X';
        end;
   end;
run;
```

```
NOTE: The data set WORK.SMALL has 63406 observations and 21 variables.
NOTE: The data set WORK.LARGE has 315975 observations and 683 variables.
NOTE: DATA statement used (Total process time):
      real time           2:10.71
      cpu time            9.78 seconds
```



VIEWTABLE: Work.Large

	keyvar	largevar1	largevar2	largevar3	largevar4	largevar5	largev
1	92482	55	55	55	55	55	
2	199913	55	55	55	55	55	
3	460802	55	55	55	55	55	
4	271490	55	55	55	55	55	
5	24898	55	55	55	55	55	
6	409660	55	55	55	55	55	
7	426698	55	55	55	55	55	
8	478512	55	55	55	55	55	
9	136306	55	55	55	55	55	
10	488383	55	55	55	55	55	
11	344119	55	55	55	55	55	
12	279278	55	55	55	55	55	
13	237895	55	55	55	55	55	
14	317263	55	55	55	55	55	
15	291291	55	55	55	55	55	



VIEWTABLE: Work.Small

	keyvar	smallvar1	smallvar2	smallvar3	smallvar4	smallvar5	smallva
1	24898	12	12	12	12	12	
2	426698	12	12	12	12	12	
3	339763	12	12	12	12	12	
4	24773	12	12	12	12	12	
5	202410	12	12	12	12	12	
6	226744	12	12	12	12	12	
7	219907	12	12	12	12	12	
8	51637	12	12	12	12	12	
9	210354	12	12	12	12	12	
10	94925	12	12	12	12	12	
11	288002	12	12	12	12	12	
12	60782	12	12	12	12	12	
13	210687	12	12	12	12	12	
14	74215	12	12	12	12	12	
15	290910	12	12	12	12	12	

Here is an example of two fairly small tables (only 3 variables from WORK.LARGE) being merged, once with traditional match-merge and once with %HASHMERGE.

MATCH-MERGE: SMALL EXAMPLE

```
proc sort data=small
           out=a_sorted;
   by keyvar;
run;
proc sort data=large(keep=keyvar largevar1 largevar22 largevar300)
           out=b_sorted;
   by keyvar;
run;
data test1;
   merge a_sorted (in=a)
         b_sorted (in=b);
   by keyvar;
   if a;
run;

NOTE: There were 63406 observations read from the data set WORK.SMALL.
NOTE: The data set WORK.A_SORTED has 63406 observations and 21 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           2.28 seconds
      cpu time            0.21 seconds

NOTE: There were 315975 observations read from the data set WORK.LARGE.
NOTE: The data set WORK.B_SORTED has 315975 observations and 4 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           1:05.84
      cpu time            4.21 seconds

NOTE: There were 63406 observations read from the data set WORK.A_SORTED.
NOTE: There were 315975 observations read from the data set WORK.B_SORTED.
NOTE: The data set WORK.TEST1 has 63406 observations and 24 variables.
NOTE: DATA statement used (Total process time):
      real time           0.28 seconds
      cpu time            0.26 seconds
```

The traditional match-merge took about 68 seconds. Now, with %HASHMERGE:

%HASHMERGE: SMALL EXAMPLE

```
%hashmerge(data=test2,
           data_a=small,
           data_b=large,
           vars_b=largevar1 largevar22 largevar300,
           by=keyvar,
           if=a);

NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.00 seconds

NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.00 seconds

NOTE: The query requires remerging summary statistics back with the original
data.
NOTE: The query requires remerging summary statistics back with the original
data.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.81 seconds
      cpu time            0.07 seconds

mem_a=      10
mem_b=      10

NOTE: The infile MEM is:
      Unnamed Pipe Access Device,
      PROCESS=systeminfo,RECFM=V,LRECL=256

maxmem=1153
NOTE: 276 records were read from the infile MEM.
      The minimum record length was 0.
      The maximum record length was 87.
NOTE: DATA statement used (Total process time):
      real time           8.57 seconds
      cpu time            0.04 seconds

NOTE: There were 315975 observations read from the data set WORK.LARGE.
NOTE: There were 63406 observations read from the data set WORK.SMALL.
NOTE: The data set WORK.TEST2 has 63406 observations and 24 variables.
NOTE: DATA statement used (Total process time):
      real time           58.07 seconds
      cpu time            3.51 seconds
```

%HASHMERGE took about the same amount of time, 67 seconds (9 seconds just to obtain available memory). Notice that the memory required to hash the table/variables passed as 'b' only took 10mb while the computer showed 1153mb available.

The next example includes many more variables with WORK.LARGE so as to push up the memory requirement. Traditional match-merge may include the colon modifier with the variable names in a KEEP dataset option—something not permitted with %HASHMERGE:

MATCH-MERGE: LARGE EXAMPLE

```
proc sort data=small
          out=a_sorted;
   by keyvar;
run;
proc sort data=large(keep=keyvar largevar1: largevar2: largevar3:)
          out=b_sorted;
   by keyvar;
run;
data test;
   merge a_sorted (in=a)
         b_sorted (in=b);
   by keyvar;
   if a;
run;

NOTE: There were 63406 observations read from the data set WORK.SMALL.
NOTE: The data set WORK.A_SORTED has 63406 observations and 21 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           1.26 seconds
      cpu time            0.25 seconds

NOTE: There were 315975 observations read from the data set WORK.LARGE.
NOTE: The data set WORK.B_SORTED has 315975 observations and 334 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           3:34.42
      cpu time            18.11 seconds

NOTE: There were 63406 observations read from the data set WORK.A_SORTED.
NOTE: There were 315975 observations read from the data set WORK.B_SORTED.
NOTE: The data set WORK.TEST has 63406 observations and 354 variables.
NOTE: DATA statement used (Total process time):
      real time           22.87 seconds
      cpu time            3.43 seconds
```

The traditional match-merge took just under 4 minutes. The limitations of %HASHMERGE prevent using the colon modifier when passing the variables to keep. However, this feature may still be used by creating and using a view:

%HASHMERGE: LARGE EXAMPLE

```
data lesslarge/view=lesslarge;
   set large(keep=keyvar largevar1: largevar2: largevar3:);
```

```
run;
%hashmerge(data=test,
           data_a=small,
           data_b=lesslarge,
           by=keyvar,
           if=a);

NOTE: DATA STEP view saved on file WORK.LESSLARGE.
NOTE: A stored DATA STEP view cannot run under a different operating system.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds

NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

NOTE: The query requires remerging summary statistics back with the original
data.
NOTE: The query requires remerging summary statistics back with the original
data.
NOTE: Invalid (or missing) arguments to the FLOOR function have caused the
function to return a missing value.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.23 seconds
      cpu time            0.15 seconds


mem_a=      10
mem_b=       .

NOTE: The infile MEM is:
      Unnamed Pipe Access Device,
      PROCESS=systeminfo,RECFM=V,LRECL=256

maxmem=1105
NOTE: 276 records were read from the infile MEM.
      The minimum record length was 0.
      The maximum record length was 88.
NOTE: DATA statement used (Total process time):
      real time           3.85 seconds
      cpu time            0.03 seconds

2 notes generated by the view were omitted here as view real/cpu times are a
subset of, not an addition to, the subsequent data statement statistics

NOTE: There were 63406 observations read from the data set WORK.SMALL.
NOTE: The data set WORK.TEST has 63406 observations and 354 variables.
NOTE: DATA statement used (Total process time):
      real time           1:51.68
      cpu time            11.56 seconds
```

%HASHMERGE took just under 2 minutes, or **about half the real time of the traditional match-merge**. It should be noted that the memory requirement of the view returned a missing value as the view does not know the number of rows until it is rendered as data. It worked in this example as the memory requirement (around 844mb) was less than available memory. *I believe this limitation with views can be overcome, just not as of the writing of this paper.*

Finally, here is an example where the merge table is too large for memory and the macro generates the traditional match-merge code.

%HASHMERGE: PRODUCES MATCH-MERGE CODE

```
%hashmerge(data=test,
           data_a=small,
           data_b=large,
           by=keyvar,
           if=a);

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

NOTE: The query requires remerging summary statistics back with the original
data.
NOTE: The query requires remerging summary statistics back with the original
data.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.42 seconds
      cpu time            0.26 seconds

mem_a=      10
mem_b=    1726

NOTE: The infile MEM is:
      Unnamed Pipe Access Device,
      PROCESS=systeminfo,RECFM=V,LRECL=256

maxmem=1154
NOTE: 276 records were read from the infile MEM.
      The minimum record length was 0.
      The maximum record length was 88.
NOTE: DATA statement used (Total process time):
      real time           3.71 seconds
      cpu time            0.00 seconds

NOTE: There were 63406 observations read from the data set WORK.SMALL.
NOTE: The data set WORK.A_SORTED has 63406 observations and 21 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           2.06 seconds
      cpu time            0.18 seconds

NOTE: There were 315975 observations read from the data set WORK.LARGE.
NOTE: The data set WORK.B_SORTED has 315975 observations and 683 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           9:07.37
      cpu time            32.53 seconds

NOTE: There were 63406 observations read from the data set WORK.A_SORTED.
NOTE: There were 315975 observations read from the data set WORK.B_SORTED.
NOTE: The data set WORK.TEST has 63406 observations and 703 variables.
NOTE: DATA statement used (Total process time):
      real time           1:24.98
      cpu time            8.23 seconds
```

CONCLUSION

The %HASHMERGE macro will work for most but not every situation where one would normally code a match-merge. Using this macro could save valuable processing time whenever merging two tables to create a third.

My hope is that this is only the beginning of what could become a widely used macro by encouraging everyone to improve upon it and then share it with the greater SAS community. The full code of %HASHMERGE can be found at www.sascommunity.org/wiki/HASHMERGE. Please add any improvements to the code as a new discussion.

REFERENCES

[1] You can find several papers by searching for “hashing” at www.lexjansen.com

[2] Davis, Michael, “You Could Look It Up: An Introduction to SASHELP Dictionary Views” *Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference, Paper 17-26*

[3] Snell, Gregg, “Think FAST! Use Memory Tables (Hashing) for Faster Merging” *Proceedings of the Thirty First Annual SAS Users Group International Conference, Paper 244-31*

[4] SAS-L (an electronic listserv of global SAS professionals) “Our Most Excellent Archives” www.listserv.uga.edu/archives/sas-l.html

ACKNOWLEDGMENTS

I must acknowledge *the* Hash Man, Paul Dorfman, for introducing the concept of hashing to the SAS world. I would also like to acknowledge Art Carpenter and Ron Fehd as the standard-bearers of Macro.

RECOMMENDED READING

Learn how much you do not know about SAS by subscribing to SAS-L:
www.listserv.uga.edu/archives/sas-l.html
SAS OnlineDoc® 9.1.3
support.sas.com/onlinedoc/913

CONTACT INFORMATION

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

Gregg P. Snell
Data Savant Consulting
5632 Noland Rd
Shawnee, KS 66216
(913) 638-4640
gsnell@datasavantconsulting.com
www.DataSavantConsulting.com
www.linkedin.com/in/datasavant

sas Alliance Affiliate Member

sas Certified Advanced Programmer for SAS 9