

Using Hash Object to Merge the Data Efficiently

Long Fang, PPD Inc., Beijing

ABSTRACT

In Dataset programming, it is very common to merge by some variables from other datasets. If we use match-merge, we have to sort the datasets firstly. But for datasets with large amount of records, it will cost lots of I/O time. In a situation like that, a HASH table can be used to provide a lookup table using the key variables without PROC SORT. For example, in SDTM programming, we can build a hash table from SV domain using USUBJID and VISIT as a key variable. Then, we can get the variable SVSTDTC/SVENDTC into other domains without PROC SORT and merge statement. In this paper, we will provide examples to show how to use hash table to merge one or several tables with one table. And there is also an example to use macro to create and name the hash object, specify key variables, and look up the data in hash table. We will also provide some debug tips for the hash object.

INTRODUCTION

In clinical trial, we do a lot of merge operations during SDTM mapping and ADaM creation. Usually, we need to sort the dataset by the merge key firstly and use data step merge statement. We named this as "Match-Merge" approach in this paper. If different datasets merge together by different key variable, we have to do these separately.

But we can do it in another way, Hash object. The hash object provides an efficient, convenient mechanism for quick data storage and retrieval. We named this as "Hash-Merge" in this paper.

WHAT IS A HASH OBJECT?

The hash object and the hash iterator object (It work base on hash object) are two predefined component objects for use in a DATA step provided by SAS. These objects enable you to quickly and efficiently store, search, and retrieve data based on lookup keys.

The hash object is an in-memory lookup table accessible from the DATA step. A hash object is loaded with records and is only available from the DATA step that creates it. A hash table consists of two parts: a key part and a data part. We can assign one or more character and numeric variable as the key part. The data part can be character and numeric variables or empty.

Once a hash object is defined, it can add records by object's ADD method. It can look up data by passing a key to the hash object's FIND method. If a record with the particular key is found, the data part of the record is copied into DATA step variables. In addition to being able to add and find records, there are methods to replace records, remove records, and output records to a data set.

HOW TO USE A HASH OBJECT?

Because the hash is an object, so we have to define it before we can use. There are 3 steps to define a hash object.

1. Declare the hash object.
2. Create an instance of (instantiate) the hash object.
3. Initialize lookup keys and data

After these steps, we can perform many tasks. The one we use most here is FIND Method. It is determines whether the specified key is stored in the hash object and what is the data with it.

Syntax:

```

object.DefineKey('keyvarname-1'<... 'keyvarname-n'>);
object.DefineData('datavarname-1'<...'datavarname-n'>);
object.DefineDone( );
rc=object.Find(<KEY: keyvalue-1,..., KEY: keyvalue-n>);

```

SAS 9 identifies 26 known methods. You can find more information on internet or SAS help.

When we use "Hash-Merge", all other variables in this table must be explicitly passed in object's DefineData method. If can't find the key from the hash object, the variable in hash object will not change. So we should initialize the variable as missing prior to each search of the hash object if necessary. And SQL code utilizing SASHELP.VCOLUMN can accomplish this task.

EXAMPLE FOR HASH-MERGE

When we do dataset programming for SDTM or ADaM, we need a lot of PROC SORT and data step. If you look into the NOTE in log, you will find the real time is significantly longer than the CPU time. The most time was used to load the data from disk and then write back to the disk after the procedure is finished. So try to finish in less steps can make the program efficient.

1) ONE TO ONE MERGE

For example, we merge the variable VISITDY, SVSTDTC and SVSTDY from SV domain to EG domain in SDTM mapping. To do this, we could use match-merge (see Code1).

```

*code 1;
*we assume that eg only collect the visitnum no date information;
data eg5;
    set eg4;
    .....;
run;
*note: cost 4.32 seconds, 73350 observations in eg5;

proc sort data=eg5;
    by usubjid visitnum;
run;
*note: cost 5.59 seconds;

proc sort data=trans.sv(keep=usubjid visitnum visit visitdy svstdtc svstdy) out=sv0;
    by usubjid visitnum;
run;
*note: cost 0.54 seconds, because it is already sorted;

data eg6;
    merge eg5(in=in1) sv0;
    by usubjid visitnum;
    if in1;
run;
*note: cost 5.25 seconds, 73350 observations in eg6;
*total time cost equal to 15.7 second;

proc sort data=eg6 out=eg(label='ecg test results');
    by studyid usubjid egtestcd visitnum egtptref egtptnum;
run;

proc sort data=eg6 out=eg(label='ecg test results');
    by studyid usubjid egtestcd visitnum egtptref egtptnum;
run;

*code 2;

```

```

data eg5;
  if 0 then set trans.sv;
  if _n_=1 then do;
    declare hash sv(dataset:'trans.sv');
    sv.definekey('usubjid','visitnum');
    sv.definedata('usubjid','visitnum','visit','visitdy','svstdtc','svstdy');
    sv.definedone();
  end;
  set eg4;
  rc=sv.find()
  ....;
run;
*note: cost 7.28 seconds, 73350 observations. save more than half time.;

```

If we use the hash object to do a Hash-merge, the code can be like the Code #2. Compare with Code #1, it only use 1 data step and can be done with other data steps and no PROC SORT is needed. Then it will save a lot of I/O time for large dataset.

Note: We only define the object once. So we add "if _N_=1 then ..." to avoid repeat declare of hash. If we forgot to add this, dataset will be load n times which n is equal to the observations in dataset EG4. This will make the code inefficient.

In addition, we suggest to place the smaller dataset into the hash object. It will save memory because the hash object is in-memory table.

2) SEVERAL-TO-ONE MERGE.

Now, this example merges 3 different datasets with one by different key variables.

```

*code 3;
data adxx1;
  length usubjid $ 20 rfstdtc svstdtc $ 19 visitnum visitdy svstdy 8
         visit $ 20 aespil $ 3 aeterm aeecod $ 200;
  if _n_=1 then do;
    declare hash dm(dataset:'dm'); *for getting the reference date;
    dm.definekey('usubjid');
    dm.definedata('usubjid','rfstdtc');
    dm.definedone();

    declare hash sv(dataset:'trans.sv'); *for getting the visit date;
    sv.definekey('usubjid','visitnum');
    sv.definedata('usubjid','visitnum','visit','visitdy','svstdtc',
                  'svstdy');
    sv.definedone();

    declare hash ae(dataset:'trans.ae'); *for getting the ae;
    ae.definekey('usubjid','aespid');
    ae.definedata('usubjid','aespid','aeterm','aeecod');
    ae.definedone();
    call missing(usubjid, rfstdtc, visitnum, visit, visitdy, svstdtc, aespil,
                 aeterm, aeecod); *to avoid the uninitialized note in log.;
  end;
  set adxx0;
  rc=dm.find(); *get the rfstdtc;
  rc=sv.find(); *get the visit information;
  rc=ae.find(); *get the ae information;
run;

```

If we use match-merge instead, we have to use 3 different data steps and PROC SORT steps for 3 different merge operation by keys. We can do it in just one step, using Hash-merge. Of course, we can use the PROC SQL to get

same result. But using data step by hash object is more efficient and we can do other derivation at the same time. Trying to use less data steps will save a lot of I/O time when we manipulate the large dataset.

MACRO

Note, below is just an example for programming. For actual data, we can design the key and data as needed.

```
*code 4;
%macro setsv(part=,vnum=visitnum);
*part=1, only execute the declare object.;
*part=2, only execute the find() method;
*else excute all;

%if &part ne 2 %then %do;
length svstdtc $ 19. visit $ 20 visitdy 8;
if _n_=1 then do;
declare hash hsv(dataset:"trans.sv");
hsv.definekey('usubjid','visitnum');
hsv.definedata('visit','visitdy','svstdtc');
*do not include the visitnum here. why?;
hsv.definedone();
end;
%end;
%if &part ne 1 %then %do;
rc=hsv.find(key:usubjid, key:&vnum );
*we can assign what variable or value pass to the key.;
if rc ne 0 then do;
call missing(visit,visitdy,svstdtc);
*if can not find in hash, then make it as missing value;
end;
%end;
drop rc;
%mend;

data adeg5;
%setsv(part=1);      *define the hash object;
set adeg4;
%setsv(part=2,vnum=10);
*1st: find the date for visitnum=10;
sv10dtc=svstdtc;
%setsv(part=2);
*2nd: find the data for visitnum.
the visit, visitdy, svstdtc will be replace.;
dur=input(svstdtc,yymmdd10.)-input(sv10dtc,yymmdd10.); *calculate the dur;
.....;
run;
```

In this example, we assign the key value for the object.Find() method.

Syntax: rc=object.FIND(<KEY: keyvalue-1,..., KEY: keyvalue-n>);

The specified key value can be variable or constant. But the type of key must match the corresponding key variable that is specified in the object.DefineKey() method call. And the number of "KEY: keyvalue" pairs depends on the number of key variables that you define by using the object.DefineKey() method.

In this example, we use object.Find() several time with different key value. But why not include the variable VISITNUM in the object.DefineData() method? If variable VISITNUM is in the hash object data part, the variable VISITNUM will be replaced by 10 after 1st SV.Find(). And the result will be totally wrong.

In macro, we remember to initialize the variables in hash key or data list. And take care of what will do when the key is not find in hash table. Because the hash will not replace the old variable value if the object.Find() is not zero.

This is just an example. In your study, you can build macro for the frequency merging, which will make the programming work efficient.

MORE THAN MERGE

The data store in hash table is indexed by key variables. When we declare the hash object, we can specify whether or how the data is returned in key-value order. And we can use the hash object OUTPUT method to output the data after sorted. And PROC SORT is not needed. It will save a lot of time.

For this purpose, we pass all variables we need to the object.DefineData() method. PROC SQL code utilizing SASHELP.VCOLUMN can do this.

```
*code 5;
proc sql noprint;    *get the variable list we need to keep.;
    select name into :varlist separated by ',' from sashelp.vcolumn
        where libname='work' and memname='lb0';
run;

data _null_;
    set lb0 end=eof;
    if _n_=1 then do;
        declare hash lb(ordered:'a'); *specify the order of the data;
        lb.definekey('usubjid', 'lbcats', 'lbsecat', 'lbtestcd', 'visitnum');
        lb.definedata('usubjid', 'lbcats', 'lbsecat', 'lbtestcd', 'visitnum', 'lbtest',
            'lbortres', ...);
        *lb.definedata("&varlist"); *we can also do like this if necessary;
        lb.definedone();
    end;

    ...;    *deriving other new variable here.;

    lb.add();
    if eof then lb.output(dataset:'lb1');
run;
```

But we have to make sure there are no duplicate key records. Or the LB.Add() will be error and records may be lost.

DEBUG

The samples are just to illustrate the basics of hash object. But hash-merge code is a bit more complicated than match-merge. But when we learn a new programming technique, the unexpected NOTES, WARNINGS and ERRORS may appear in log usually. In this section, we will show you a few more common examples in SAS log as you begin to use your own hash code.

1) ERROR: Undeclared data symbol description for hash object...

ERROR: DATA STEP Component Object failure. Aborted during the EXECUTION phase.

It is important to note that the DEFINEDONE and DEFINEKEY methods do not affect the PDV. So if you were to execute the above code without the length statement, the log would show the error message as above. To avoid this error, we can predefine the length for these variables or get data attributes by reading a record from the source data.

Eg.

```
data xxxx;
    ...;
    set trans.sv point = _n_ ; * get key/data attributes for parameter type
        matching ;
    * set trans.sv (obs = 1) ; * this will work, too :-)! ;
    * if 0 then set trans.sv ; * and so will this :-)! ;
    * set trans.sv (obs = 0) ; * you will get result with 0 records. :-( ;
    declare hash sv (dataset: 'trans.sv') ;
```

```
...;  
run;
```

2) ERROR: Method defineDone must be called to complete initialization of hash object before line x column y. ERROR: DATA STEP Component Object failure. Aborted during the EXECUTION phase.

The object.DefineDone() is needed to complete the hash object before we use the object. It should not be omitted.

3) NOTE: Variable XXXX is uninitialized.

This note comes up in the SAS log usually. Because we will not do anything for some of the variable in the object.DefineData() method. So The SAS compiler does not see variable assignments. We can add a row for call missing() after the object.DefineDone(). Then these variables are initialized.

4) Invalid object attribute reference object.find.

The FIND is a method. So parentheses '()' is required just like all other method. Eg. object.Add(), object.DefineDone().

5) ERROR: Key not found.

If we use object.Find() directly, when the key is not found in hash table, it will create such log. We can use rc=object.Find() instead. Then it will return a non-zero value.

6) ERROR: Duplicate key.

As code 5, if there are duplicate key records, this ERROR will show in the log and only the first records will be added to the hash table successfully. So we should keep enough key variables to identify each record. If we only need to keep the later one, we can use object.Replace() method instead of object.Add() method.

NOTE: If we use rc=object.Add(), there will be no ERROR in log, but it doesn't mean correct. We should add an if-then statement to make sure all return rc is equal to 0.

7) ERROR: Hash object added XXXX items when memory failure occurred.

FATAL: Insufficient memory to execute data step program. Aborted during the EXECUTION phase.

NOTE: The SAS System stopped processing this step because of insufficient memory.

Because the hash object is an in-memory lookup table. So all of the data is stored in memory which is why it can look up so efficiently. But for same reason, the size of hash object is limited by the available physical memory. When the hash object uses up all the memory, the data step fails. To avoid this:

- 1) Place as little data in the hash object as possible.
- 2) Use codes for the key and data items so they are as small as possible.
- 3) If the hash object only needs to contain a subset of the observations of a data set, create a view that references only the subset.

It is the limitation to sort the large dataset. And in merge operations, we should load the smaller dataset to the hash, and try to load as little data as necessary. Function GETOPTION('XMRLMEM') could be used to estimate remaining memory.

CONCLUSION

This paper introduced the functionality of the hash object and gave examples that use the hash object to perform hash-merge operations and order the records. Comparing to the usual way PROC SORT and match-merge, hash-merge uses less steps and saves time. Particularly, if one of the input data sets fits in memory, a hash object is the fastest choice for merge. If the result dataset fits in memory too, a hash object is the fastest choice for sorting. But if the memory is not enough, then it is not a good choice.

REFERENCES

Secosky, J., Bloom, J. 2006. Getting Started with the DATA Step Hash Object. SAS Institute, Inc.
<http://support.sas.com/rnd/base/topics/datastep/dot/hash-getting-started.pdf> (accessed June 15, 2012).

Alden, Kay and Stroupe, Jane, "The DATA HASH - Not the Monster you Think It Is", PNWSUG, 2007. Available at
[http://www.pnwsug.org/sites/test.pnwsug.org/files/Kay Alden - Data Hash Handout.pdf](http://www.pnwsug.org/sites/test.pnwsug.org/files/Kay%20Alden%20-%20Data%20Hash%20Handout.pdf)

Dorfman, Paul and Wyverman, Koen, "Data Step Hash Objects as Programming Tools". Proceedings of the Thirtieth Annual SAS Users Group International Conference, 2005. Available at
<http://www2.sas.com/proceedings/sugi30/236-30.pdf>

CONTACT INFORMATION

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

Name: Long Fang
Enterprise: PPD Inc.
Address: 8th.Floor, Tower B, Central Point No. 11, Dongzhimen South Ave Dongcheng District
City, State ZIP: Beijing, 100007
Work phone: +8610-57636250
Fax: +8610-57636251
E-mail: Long.fang@ppdi.com
Web : www.ppdi.com

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.

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