PharmaSUG China 2015 - Paper 14

It's Not Just Fast Merging, It's Powerful Hashing

Mijun Hu, Novartis, Shanghai, China

ABSTRACT

Ever since hash object was introduced in SAS® 9, a bunch of papers have commended its high merging efficiency but however overlooked its other powerful functions. In fact, besides the widely used find method of performing the merge, we can also utilize check, find_next, add and output method, etc. to achieve certain purposes with higher efficiency and readability but less coding. The advantage is utterly obvious in large database such as ADLB because it eliminates several times of entire database copying in terms of manipulating multiple hashing within one data step. This paper is intended for any intermediate level hash users who mainly only use find method but would like to explore other functionalities of hash object in real examples.

INTRODUCTION

ADLB dataset is always tough especially in compound pooling due to enormous scale of data. A highly efficient programming code can play a significant role in this process because it not only saves you from infinite waiting run time but also makes the code more compact and readable. Normal find method can only merge common variables but other hash methods can also facilitate deriving ADLB variables such as ABLFL, ANL01FL, BASE, BASECAT, AVISIT, etc. This paper is based on a real compound pooling ADLB(almost 10 million records with the size of 30 gigabytes) example to demonstrate the power and convenience that hash object brings us.

You need to understand the basic syntax of hash object before reading this paper. Knowing how to apply simple find method is the minimal requirement.

RECAP WHAT HASH OBJECT CAN DO

The hash object provides an efficient, convenient mechanism for quick data storage and retrieval. The hash object stores and retrieves data based on lookup keys. There are two steps to create a hash object:

1. Declare and instantiate the hash object.
2. Initialize lookup keys and data.

Below code put selected variables from ADSL with only records in safety population into a hash object named h1:

```
declare hash h1(dataset:"data_a.adsl(where=(saffl='Y') keep=&common_var saffl)");
       h1.defineKey('usubjid');
       h1.defineData(all:'yes');
       h1.defineDone();
```

After you declare and instantiate a hash object, you can perform many tasks by different methods, including these:

1. Store and retrieve data.
2. Maintain key summaries.
3. Replace and remove data.
4. Compare hash objects.
5. Output a data set that contains the data in the hash object.

Of course you also have to pre-specify the variable type and length, initialize its value etc to avoid unwanted messages in the log. For detailed syntax please refer to introductory HASH papers.

Other than the find method, there are many other useful hash methods:

- Check: Checks whether the specified key is stored in the hash object.
- Find_next: Sets the current list item to the next item in the current key's multiple item list and sets the data for the corresponding data variables.
- Add: Adds the specified data that is associated with the given key to the hash object.
- Output: Creates one or more data sets each of which contain the data in the hash object.

The rest hash methods are not covered in this paper.

DIFFICULTIES OF ADLB DEVELOPMENT

As we know, normally ADLB contains the largest data among all ADaM, every addition set statement will increase the running significantly. On the other hand, the analysis visit window in compound pooling ADLB, different parameters in different studies can all have different mapping rules to derive AVISITN/AVISIT, code them programmatically inside the code makes the code extremely lengthy and is subject to mistakes. Creating formats in a separate file sounds like a better choice but still subject to typo.

To overcome above two difficulties, we can fetch external data including common variables, baseline records, analysis visits by mounting external data set into hash objects because multiple hash data retrievals can be perform in one single data step. It is safest to programmatically generate all the visit window high/low boundary and save them to a permanent dataset(let's name it AVISIT under libref DATA_A).

TABLE: Data_a.Avisit

AWTARGET	AWLO	AWHI	GROUPC	AVISITN	AVISIT
1	-999	1	A2211_LB_0	0	Baseline
8	2	11	A2211_LB_0	1	Week 1
15	12	22	A2211_LB_0	2	Week 2
29	23	43	A2211_LB_0	4	Week 4
57	44	71	A2211_LB_0	8	Week 8
85	72	99	A2211_LB_0	12	Week 12

Display 1. The structure of AVISIT data set, same GROUPC value indicate this is one analysis visit window

Visits within in the same visit window group can be identified by a given variable(say GROUPC), we can merge by GROUPC and on the condition that ADY is between AWLO and AWHI, in order to retrieve AVISITN/AVISIT. (see Display 1)

TECHNICAL PART:

As we know, below I am going to show how to derive ABLFL, ANL01FL, BASE, BASECAT, AVISIT, AVISITN by hash methods.

POPULATE ABLFL, ANL01FL

Normally ABLFL(baseline flag) and ANL01FL(analysis flag) are put on the records fulfilling certain criteria, e.g. flag ABLFL=Y for the record that is last one on or before the treatment date with a valid assessment; flag ANL01FL=Y for the record that is the closest to the target analysis day. Traditional approach usually sort and set the whole data set several times and flag the first record with Y within each by group. This whole data manipulation takes a long time to run, sometimes even exceed your server's capacity to perform a big SORT procedure.

Let's look into the core through the phenomenon. Essentially we only need to pick out the records that fulfilling the criteria and flag those records with Y. What if we first give each record with a unique identifier (such as LBSEQ, or _bdsvar_seq=_N_, you name it), and then only save the identifier number of records to be flagged to a list, then we only need to put Y for the records where you can find its identifier in that list.

Two data sets are created:

1. Data set _BDSVAR_BASE only contains the baseline records.
2. Data set _BDSVAR_ANL01FL only contains the records closest to the analysis target date.

The time of creating these two data sets are much less than sorting/setting the whole lab data set several times because you can to filter out only the variables and records you need to SORT procedure.

Both data sets only need to include one variable _BDSVAR_SEQ at least but recommend to include other related variables for debugging purpose.

```
declare hash h1(dataset:'_bdsvar_base');
  h1.defineKey('_bdsvar_seq');
  h1.defineDone();

declare hash h3(dataset:'_bdsvar_anl01fl');
  h3.defineKey('_bdsvar_seq');
  h3.defineDone();

if h1.check()=0 then ablfl='Y';
if h3.check()=0 then anl01fl='Y';
```

In fact, BASE, BASEC, BASECAT type variables can also be populated by hashing _BDSVAR_BASE data set but with a different key and data components.(key: USUBJID PARAMCD)

POPULATE AVISITN, AVISIT, AWHI, AWLO AND AWTARGET

The difficulty to get AVISIT related variables from AVISIT data set(see **display 1**) is it's not merged by variables values but by condition that ADY is between AWHI and AWLO. Luckily we can overcome this by find_next method.

```
declare hash h4(dataset:'data_a.avisit',multidata:'y');
  h4.defineKey('groupc');
  h4.defineData('avisit','avisitn','awlo','awhi','awtarget');
  h4.defineDone();

if h4.find()=0 then
  do while (^(awlo<=ady<=awhi));
    if h4.find_next()^=0 then
      do;
        call missing(avisit,avisitn,awlo,awhi,awtarget);
        leave;
      end;
  end;
```

We first load the whole AVISIT data set into hash h4 with option multidata:'y', otherwise hash will only keep the first record with the same key value it encounters. Later for a certain key, we keep fetching the record to the data set from h4 sequentially until the condition AWLO<=ADY<=AWHI is met or we cannot find any record that can full fill this condition.(i.e. ADY is beyond the analysis visit window)

ADD RECORDS TO A HASH OBJECT AND EXPORT IT TO A DATA SET

It is very likely GROUPC in AVISIT dataset is not an exhausted list, especially when new studies are added into the compound pool. We'd better to set up some defensive code to remind us to update AVISIT timely. One traditional way is, for every record if GROUPC is not found in h4, we can put a warning message to the log. But, hey, this is compound pooling ADLB, if there is one GROUPC missing from h4, probably hundreds of thousands warnings will output to the log for the same key. It is not only difficult to remove the duplicates from logs but also increase the time of running due to writing redundant messages into log.

Again we can take advantage of hash object. Remember hash object only keep the first key record and automatically remove dup key records.(unless you specify multidata:'y'). Now let's rewrite the code above, add a new hash object called h4_nm and add the keys that cannot be found in h4 to h4_nm.

In the last iteration of DATA STEP, we export the hash h4_nm to an external dataset _BDSVAR_H4_NM

```
declare hash h4(dataset:'data_a.avisit',multidata:'y');
  h4.defineKey('groupc');
  h4.defineData('avisit','avisitn','awlo','awhi','awtarget');
  h4.defineDone();
```

```
declare hash h4_nm();
  h4_nm.definekey('groupc');
  h4_nm.defineDone();

if h4.find()^=0 then rc=h4_nm.add();
else
  do while (^(awlo<=ady<=awhi));
    if h4.find_next()^=0 then
      do;
        call missing(avisit,avisitn,awlo,awhi,awtarget);
        leave;
      end;
end;

……

if last then h4_nm.output(dataset:'_bdsvar_h4_nm');
```

we can write a separate step to inform us these mismatches

```
data _null_;
  set _bdsvar_h4_nm;
  put 'WAR' 'NING: ' GROUPC= 'cannot be found in data_a.avisit';
  if _n_=10 then
    do;
      put 'At most 10 warn' 'ing records show, for a full list, please check the
             dataset _bdsvar_H4_NM';
      stop;
    end;
run;
```

Now you do not have to find GROUPC in the log but instead, in the data set, with duplicate records removed. But the log can still warn you AVISIT needs update.

CONCLUSION

This paper only shows applications of find, find_next, check, add, output methods in ADLB development. However the hash dictionary contains more methods than these and it is always worthy of exploring any potential advantages of them over the traditional approach such as PROC SQL or MERGE statement. The examples here only scratches the surface of their potential utility. I believe they can be used in a variety of circumstances after more investigation. We should not reject to acquire new technology because the convenience you get overweighs the time invested on the studying it.

REFERENCES

SAS® 9.3 Component Objects: Reference

CONTACT INFORMATION

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

Name: Mijun Hu
Enterprise: Novartis
Address: 402 Room, 72 Liangxiu Rd
City, State ZIP: Shanghai, 201203
Work Phone: 021-61606319
E-mail: Mijun.hu@novartis.com