PharmaSUG 2015 - Paper BB18

Not Just Merge - Complex Derivation Made Easy by Hash Object

Lu Zhang, PPD, Beijing, China

ABSTRACT

Hash object is known as a data look-up technique widely used in data steps for many of its advantages. Before SAS 9.2, hash object was mostly used to accomplish efficient data merging in a DATA step. However, the hash object did not allow storing and retrieving duplicate keys before SAS 9.2. This constraint was eliminated in SAS 9.2 where in this version hash objects in DATA step can perform data look-up even though the keys are not unique. With this improvement, many complex derivations in our daily work become more straightforward and simpler than before. In this paper, the added features to hash object in SAS 9.2 will be discussed. Also examples in analysis database derivations will be given to illustrate how hash object works to improve the implementation efficiency of complicated derivation algorithms.

INTRODUCTION

In analysis database (ADB) programming, we may face the situation that for a bunch of grouped observations, we need to refer to other grouped instances to get the derived information for analysis. To achieve that, we have to do extra data manipulations to combine the two parts of information appropriately if we are using regular routines such as MERGE statement or SQL JOIN. That is because both in MERGE statement and SQL JOIN, multiple merge occurs if the keys are not unique, which may lead to unexpected wrong results. When it comes to hash object before SAS 9.2, we also need to deal with the duplicate keys first as demonstrated by Dorfman and Shajenko [2005-B]. But beginning from SAS 9.2, with the update to hash objects, such derivations could be much simpler.

The new features added to hash object in SAS 9.2 are as following,

1. Ignore duplicate keys when loading a data set into the hash object. With this feature added in, non-unique keys can be defined in hash objects.
2. Specify whether multiple data items are allowed for each key. With this feature added in, multiple records associated with the non-unique keys in hash objects could be loaded as needed.

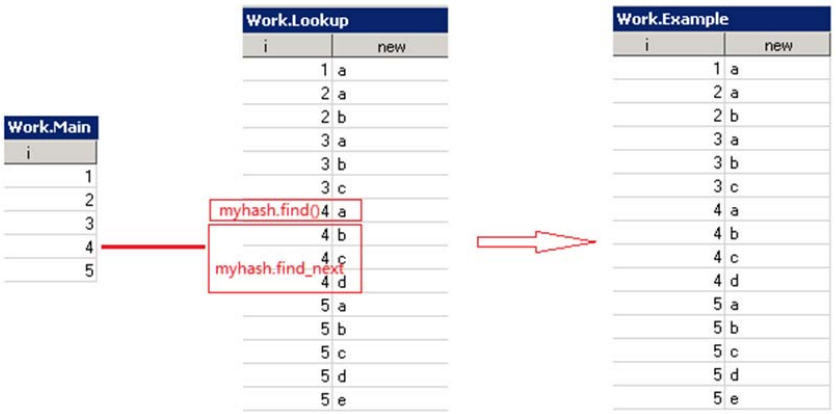
In this paper, the application of these new features will be discussed through two real examples in analysis database derivation.

SYNTAX

```
data example;
    if 0 then set lookup;
    set main;
    if _n_=1 then do;
        declare hash myhash (dataset:'lookup', multidata:'y');
        myhash.definekey('i');
        myhash.definedata('new');
        myhash.definedone();
        call missing(new);
    end;
    rc=myhash.find();
    do while (rc=0);
        output;
        rc=myhash.find_next();
    end;
run;
```

By default, the FIND() method of hash object returns the first value associated with the defined key in the lookup dataset. In above code, the tag option <MULTIDATA:'Y'> allows the method of FIND_NEXT() to be utilized. After FIND() method returns the first value associated with the defined key, if the defined key has multiple values

associated, subsequent values are returned by FIND_NEXT() method. The dataset of 'main', 'lookup' and the output data set named 'example' are shown as below.



In the lookup dataset, the defined key is not unique. Within a DO-WHILE loop, the multiple records associated with each key are traversed. Take i=4 for example, MYHASH.FIND() returns the value 'a' which is the value of the first record associated with the key (i=4). The values of the rest records (values 'b', 'c', and 'd') are returned by MYHASH.FIND_NEXT() within the DO-WHILE loop. After the OUTPUT statement, all values associated with the defined key 'i' are merged on the main dataset.

The following two examples will show how we can benefit from this feature.

EXAMPLE 1: PSA RESPONSE DERIVATION

Prostate-Specific Antigen (PSA) is commonly used as an indicator of prostate cancer. In this example, the PSA response is derived in ADB for efficacy analysis. The PSA response here is defined as a decrease from baseline of at least 50%, which must be confirmed by the first assessment at 6 weeks later, with also a decrease from baseline of at least 50%.

Assuming we have the pre-dataset named PSA_PRE with variables as following.

SUBJID: *Subject identifier*
ASSDY: *Assessment day*
PSAVAL: *PSA value*
BPSA: *Baseline PSA value*

The difficulty of the derivation lies in identifying the confirmative PSA value at 6 weeks later. For each subject, there are multiple assessments performed. To find each assessment a confirmative value, we should look for all assessments performed after current record. If we just use a MERGE statement to combine such two parts of information, we would get a mess. Usually, the following 2 approaches may implement the derivation.

1. RETAIN statement. It can keep the information of current record and pass them to next records. So when we have the PSA_PRE data in the order of SUBJID and descending ASSDY, we can achieve our goal to identify if there is any PSA response. But derivation using RETAIN would be very complex and hard to maintain.
2. TRANSPOSE procedure and ARRAY statement. In this way, the vertical PSA values at different assessment day will be transposed to be horizontal. Thus combined with ARRAY statement, we can decide if a PSA response is qualified. However, after transpose, there may be numerous variables if there are multiple assessments performed. And the maximum number of assessments should be derived to decide the dimension of the array.

In this paper we won't go any further on above approaches. Just see how hash object works in such derivation.

The code is shown as below.

```
proc sort data=psa_pre;                                          1
    by subjid assdy;
run;

data psa_res;
    length _assdy _psaval 8;

    set psa_pre;                                                 2
    if _n_=1 then do;
        declare hash psah (dataset:'psa_pre(rename=(assdy=_assdy psaval=_psaval))', multidata:'y');
        psah.definekey('subjid');
        psah.definedata('subjid','_assdy','_psaval');
        psah.definedone();
        call missing(_assdy, _psaval);
    end;
    rc=psah.find();

    if .<psaval<=bpsa*.5 then do while(rc=0);                    3.1
        if _assdy>=assdy+42 then do;                             3.2
            if .<_psaval<=bpsa*.5 then do;                       3.3
                psaresfl='Y';
                leave;
            end;
            else if _psaval>bpsa*.5 then leave;                  3
        end;

        rc=psah.find_next();
    end;
run;
```

In part 1, as a preparation, we sort the pre-dataset PSA_PRE in ascending order of ASSDY per subject.

Then following with a DATA step, in part 2, we declare a hash object named PSAH with tag option <multidata:'y'>. Note we will need to rename ASSDY and PSAVAL to _ASSDY and _PSAVAL respectively since we declare the PSA_PRE itself as the hash object.

In part 3, the logic is as following.

- 3.1: For records no more than 50% baseline, the DO-WHILE loop will start to find whether a confirmative value exists.
- 3.2: Since PSA_PRE is in ascending order of SUBJID and ASSDY, the records in hash object are traversed by such order. Under the condition of _ASSDY>=ASSDY+42, when the first record in hash object meet this condition, it should be the first assessment at 6 weeks later from current observation in the DATA step.
- 3.3: Once the first assessment after 42 days is identified, the code will compare its PSA value against baseline value to decide if current observation could be confirmed as a PSA response. The LEAVE statement is important here to stop DO-WHILE loop for current record and begin to read the next record in the data step. It assures only the PSA value of the first assessment after 42 days is used.

From above code, we can see using hash object to implement such derivation is very straightforward. By using hash object within a data step, all assessments could be traversed for each observation, which makes it easy to decide if current record could be confirmed as a PSA response. Compared with other approaches, the code is simpler and more efficient.

EXAMPLE 2: AE CYCLE DERIVATION

In this example, cycle number will be calculated for the treatment-emergent adverse events (TEAEs) for future summarization. The derivation rules are in the following.

- TEAEs which occur on Day 1 Cycle 1 belong to Cycle 1.

- After cycle 1, TEAEs will be categorized by the “throw-back rule”, that is, TEAEs that occur on Day 1 of a cycle will be allocated to the previous cycle.
- All TEAEs which occur after Day 1 of last cycle will be included in the last cycle.

Assuming the cycle information is derived in a pre-generated dataset ADCYC, which is in ascending order of cycle number per subject. The variables are as below.

SUBJID: *Subject identifier*
CYCLEN: *Cycle number*
CYCLESDT: Start date of cycle

Here we can’t directly merge the two parts of information together because for each subject, there could be multiple adverse events in AE dataset and multiple cycle records in ADCYC dataset. We have to do extra data manipulations to derive the AE cycles. Or just using hash object in one data step. See code as below.

```
data _ae_cycle;
    set ae;
    if 0 then set adb.adcyc(keep=subjid cyclen cyclesdt);
    if _n_=1 then do;
        declare hash cyc (dataset:'adb.adcyc', multidata:'y');
        cyc.definekey('subjid');
        cyc.definedata('subjid', 'cyclen', 'cyclesdt');
        cyc.definedone();
    end;
    rc=cyc.find();                                          1

    do while(rc=0);
        if aestdt=cyclesdt then do;
            if cyclen=1 then acyclen=cyclen;
            else acyclen=cyclen-1;
            leave;
        end;
        else if aestdt>cyclesdt then do;
            acyclen=cyclen;
        end;
        rc=cyc.find_next();                                 2
    end;
run;
```

In part 1, a hash object named CYC is declared with ADB.ADCYC as the look-up table.

In part 2, same as example 1, for each record in AE, we will find all information associated with the same SUBJID in ADCYC. Then under the IF-THEN statement, if an AE occurs on Day 1, we will set the AE cycle number ACYCLEN to either Cycle 1, or Cycle n-1 depending on the cycle number and leave the DO-WHILE loop. Else, temporary AE cycle number will be attained if AE occurs after Day 1 of that cycle. Note here we don’t leave the DO-WHILE loop since the ACYCLEN will be overwritten until the most recent cycle associated with the key is reached.

CONCLUSION

In this paper, we discussed the improved features of hash object when non-unique keys are defined. We explored the efficiency and simplicity of using hash object in complex derivations and demonstrated the usage with examples.

To summarize, in many derivations which require reference to multiple instances, hash object could provide very straightforward and efficient solution.

More than just MERGE and above examples, there would be more to reveal that hash object can do to facilitate our daily work.

REFERENCE

Dorfman, P., Shajenko, L. (2005-B). “Crafting Your Own Index: Why, When, How”. http://www2.sas.com/proceedings/sugi31/232-31.pdf

Kenneth W. Borowiak. 2006. “A Hash Alternative to the PROC SQL Left Join”. http://www.nesug.org/proceedings/nesug06/dm/da07.pdf

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

ACKNOLEDGEMENTS

The author would like to thank Jianrong Li and Kenneth W. Borowiak for their insightful comments in reviewing an earlier publication of this paper.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of PPD.

CONTACT INFORMATION

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

Name: Lu Zhang
Enterprise: PPD Inc.
Address: 8/F, Tower B, Centra Point Plaza, No. 11 Dongzhimen South Ave., Dongcheng Dist.
City, State ZIP: Beijing, 100007
Work Phone: +86 10-57636329
Fax: +86 10-57636251
E-mail: Lu.zhang18@ppdi.com
Web: www.ppdi.com