Paper RF-12-2014

USING SAS HASH OBJECTS TO CUT DOWN PROCESSING TIME

Girish Narayandas, Optum, Eden Prairie, MN

ABSTRACT

Hash tables are in existence since SAS 9 version and are part of data step programming. The hash application and syntax style is rather different from the regular SAS data step programming and often programmers shy away from using this great feature. The hash object is a great option for lookups when using unsorted data that can fit into memory. Hash object provides an efficient and convenient mechanism for quick data storage and retrieval. SAS HASH use can achieve great I/O efficiencies and enormous time savings when merging tables with the DATA Step.

INTRODUCTION

Hash Table is a lookup table that is designed to efficiently store non-contiguous keys (ids, account numbers, codes, etc.) that may have wide gaps in their alphabetic and numeric sequences. SAS provides two pre-defined component objects for use in a DATA step: the hash object and the hash iterator object. These objects basically enable you to rapidly store, search, and retrieve data based on lookup keys efficiently there by cutting down the run time.

The DATA step component object interface enables you to create and manipulate these component objects by using statements, attributes, and methods. You use the DATA step object dot notation to access the component object's attributes and methods

The component object is established by the DECLARE statement in the SAS DATA step.

```
  DECLARE HASH PROV (DATASET: ‘LIB.PROV’, HASHEXP:20);
```

The name of the object is followed by DATASET: and optional constructor statements such as ORDERED, HASHEXP. ‘Ordered’ determines how the key variables are to be ordered in the hash table. Before I attempt to explain HASHEXP attribute, let’s understand how SAS HASH processing occurs in very simple terms:

- When the object is declared, it creates a key per value
- Hash keys are evenly distributed into hash slots so that each slot contains as few key-value pairs as possible
- When an item is looked up, its key is hashed to find the appropriate slot.
- Then, the slot is searched for the right key-value pair

HASHEXP is an indicator of how to allocate memory and distribute the hash items in memory. Simple rule is that, the larger the hash table, the larger the value HASHEXP should be assigned. The maximum size for HASHEXP is 20 (Eberhardt et al 2010). Finding out an efficient value for HASHEXP: is dependent on multiple factors. This is an exponent so a value of 10 results in

2^{10}=1,024 buckets or slots. The default size is 8 (2^8=256). For a million items HASHEXP = 9 or 10; 512 to 1024 slots) should provide good performance (Carpenter et al 2012 & SAS Documentation).

For a more thorough explanation of HASHEXP and the hash memory model see Dorfman et al 2008, Eberhardt et al 2010 and Carpenter et al 2012. This paper is not intended to be complete SAS HASH reference guide, nor is an attempt to thoroughly explain the deeper mechanism of how SAS HASH tables work. The intention is provide a quick starter guide with some basic examples in an attempt to attract SAS beginners and SAS users who haven't had a chance to use HASH tables to start exploring SAS HASH tables.

SYNTAX AND EXAMPLES

Hash tables are objects, so you call methods to use them. The syntax of calling a method is to put the name of the hash table, then a dot (.), then the method you want to use. Following the method is a set of parentheses in which you put the specifications for the method. You will see this kind of syntax in the statement below:

EXAMPLE 1: Joining two datasets with a simple key

Members Table

Member_ID	Prov_ID	GRP_ID	GRP_CATEGORY
MBR0001	PR001	GRP01	GCAT1
MBR0002	PR002	GRP01	GCAT3
MBR0003	PR003	GRP03	GCAT2
MBR0004	PR001	GRP02	GCAT2
MBR0005	PR002	GRP01	GCAT3
MBR0006	PR004	GRP02	GCAT1
MBR0007	PR002	GRP01	GCAT2

Provider Table

Prov_ID	Prov_Name	Prov_State
PR001	Scarborough Clinic	WA
PR002	Atlantic Health	ME
PR003	Fairway Clinics	NH
PR004	Camden Clinic	VA

```
DATA MBR_PROV;
      IF 1=2 THEN SET LIB.MBR LIB.PROV;
      IF _N_ = 1 THEN DO;
         DECLARE HASH PROV(DATASET: 'LIB.PROV');
         PROV.DEFINEKEY ('PROV_ID');
         PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE');
         PROV.DEFINDONE;
      END;
   SET LIB.MBR;
     IF PROV.FIND(KEY:PROV_ID) = 0 THEN OUTPUT;
RUN;
```

Here is a simple example where we are merging a member and a provider table using a prov_id key and bring in prov_name and prov_state fields into the MBR_PROV dataset that is being created. To join table using hash objects there are FOUR basic steps to follow:

STEP 1: The DATA, SET and RUN statements are part of basic data step programming components in SAS that any SAS programmer is familiar with. We are merging LIB.MBR and LIB.PROV and creating a MBR_PROV dataset. LIB.MBR is being used as the base dataset because we want all the fields from the base dataset (usually the larger dataset of the two or more datasets that are used). Since hash tables use memory, smaller dataset is used to build hash table to avoid running into possible memory issues.

```
DATA MBR_PROV;
      IF 1=2 THEN SET LIB.MBR LIB.PROV;
      IF _N_ = 1 THEN DO;
         DECLARE HASH PROV(DATASET: 'LIB.PROV');
         PROV.DEFINEKEY ('PROV_ID');
         PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE');
         PROV.DEFINDONE;
      END;
   SET LIB.MBR;
     IF PROV.FIND(KEY:PROV_ID) = 0 THEN OUTPUT;
RUN;
```

STEP 2: HASH processing needs to know about the data fields and their properties in a hash table before they can be used. Data field properties could be defined using a length and format statements, however a more efficient way would be use a SET statement as shown below. The SET statement will be read at compilation, data variable properties in the datasets will be extracted, but the statement will never actually execute due to the obviously false statement "if 1=2" condition.

```
DATA MBR_PROV;
      IF 1=2 THEN SET LIB.MBR LIB.PROV;
      IF _N_ = 1 THEN DO;
         DECLARE HASH PROV(DATASET: 'LIB.PROV');
         PROV.DEFINEKEY ('PROV_ID');
         PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE');
         PROV.DEFINDONE;
      END;
   SET LIB.MBR;
     IF PROV.FIND(KEY:PROV_ID) = 0 THEN OUTPUT;
RUN;
```

STEP 3: In this step, the DECLARE statement is used to name and instantiate the object, i.e. we are defining a HASH named PROV and stating what dataset need to be loaded into the memory. `PROV.DEFINEKEY ('PROV_ID')` defines the unique key variable(s) that will be used to join with other tables. `PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE')` denotes the data fields we want to have available for use in memory. `PROV.DEFINDONE` is part of the HASH syntax that closes the hash argument.

```
DATA MBR_PROV;
      IF 1=2 THEN SET LIB.MBR LIB.PROV;
      IF _N_ = 1 THEN DO;
         DECLARE HASH PROV(DATASET: 'LIB.PROV');
         PROV.DEFINEKEY ('PROV_ID');
         PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE');
         PROV.DEFINDONE;
      END;
   SET LIB.MBR;
     IF PROV.FIND(KEY:PROV_ID) = 0 THEN OUTPUT;
RUN;
```

STEP 4: We are almost there, This is the final step where the DATA step is reading observations from the LIB.MBR table and using key fields specified, PROV_ID to join to the has table. Now, the SAS process attempts to find matching pairs between MBR and PROV tables using the data field specified in the DEFINEKEY; PROV_ID. The 'IF' condition in this statement will result in an INNER JOIN. Therefore, only matching records between the two tables will be written to the output dataset.

```
DATA MBR_PROV;
      IF 1=2 THEN SET LIB.MBR LIB.PROV;
      IF _N_ = 1 THEN DO;
         DECLARE HASH PROV(DATASET: 'LIB.PROV');
         PROV.DEFINEKEY ('PROV_ID');
         PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE');
         PROV.DEFINDONE;
      END;
   SET LIB.MBR;
     IF PROV.FIND(KEY:PROV_ID) = 0 THEN OUTPUT;
RUN;
```

Resulting dataset:

Member_ID	Prov_ID	GRP_ID	GRP_CATEGORY	Prov_Name	Prov_State
MBR0001	PR001	GRP01	GCAT1	Scarborough Clinic	WA
MBR0002	PR002	GRP01	GCAT3	Atlantic Health	ME
MBR0003	PR003	GRP03	GCAT2	Fairway Clinics	NH
MBR0004	PR001	GRP02	GCAT2	Scarborough Clinic	WA
MBR0005	PR002	GRP01	GCAT3	Atlantic Health	ME
MBR0006	PR004	GRP02	GCAT1	Camden Clinic	VA
MBR0007	PR002	GRP01	GCAT2	Atlantic Health	ME

EXAMPLE 2: Joining THREE datasets with composite keys:

The second example is really an enhancement to the first example that illustrates joining three datasets and using composite keys (more than one key field). It involves adding a second key field to the GRP table. In the first example we needed to pick up PROV_NAME and PROV_STATE based on the value of PROV_ID. Depending on table size and number of data fields used, the SORT procedure can be quite expensive in terms of I/O efficiencies plus adding to the overall jobs run time. Hash tables allow us to avoid sorting the two or potentially more dataset which can eat up resources with the real time large datasets. In this example, we are adding a Group (GRP) table and using two keys GRP_CATEGORY and GRP_ID and bring in GRP_NAME in to the output table, MBR_PROV_GRP.

Members Table			
Member_ID	Prov_ID	GRP_ID	GRP_CATEGORY
MBR0001	PR001	GRP01	GCAT1
MBR0002	PR002	GRP01	GCAT3
MBR0003	PR003	GRP03	GCAT2
MBR0004	PR001	GRP02	GCAT2
MBR0005	PR002	GRP01	GCAT3
MBR0006	PR004	GRP02	GCAT1
MBR0007	PR002	GRP01	GCAT2

Group Table		
GRP_CATEGORY	GRP_ID	GRP_NAME
GCAT1	GRP01	NORTH WEST AAA
GCAT1	GRP02	NORTH WEST BBB
GCAT2	GRP01	SOUTH WEST AAA
GCAT2	GRP02	SOUTH WEST BBB
GCAT3	GRP01	EASTERN AAA
GCAT3	GRP02	EASTERN BBB
GCAT3	GRP03	EASTERN CCC

Provider Table		
Prov_ID	Prov_Name	Prov_State
PR001	Scarborough Clinic	WA
PR002	Atlantic Health	ME
PR003	Fairway Clinics	NH
PR004	Camden Clinic	VA

```
DATA MBR_PROV_GRP;
      IF 1=2 THEN SET SASUSER.MBR SASUSER.PROV SASUSER.GRP;
      IF _N_ = 1 THEN DO;
         DECLARE HASH PROV(DATASET: 'SASUSER.PROV');
         PROV.DEFINEKEY ('PROV_ID');
         PROV.DEFINEDATA ('PROV_NAME', 'PROV_STATE');
         PROV.DEFINEDONE ();

         DECLARE HASH GRP(DATASET: 'SASUSER.GRP');
         GRP.DEFINEKEY ('GRP_CATEGORY', 'GRP_ID');
         GRP.DEFINEDATA ('GRP_NAME');
         GRP.DEFINEDONE ();
      END;

   SET SASUSER.MBR;
     IF (PROV.FIND(KEY: PROV_ID) = 0 )
       AND (GRP.FIND (KEY:GRP_CATEGORY, KEY:GRP_ID )= 0) THEN OUTPUT;

RUN;
```

Resulting dataset:

Member_ID	Prov_ID	GRP_ID	GRP_CATEGORY	Prov_Name	Prov_State	GRP_NAME
MBR0001	PR001	GRP01	GCAT1	Scarborough Clinic	WA	NORTH WEST AAA
MBR0002	PR002	GRP01	GCAT3	Atlantic Health	ME	EASTERN AAA
MBR0004	PR001	GRP02	GCAT2	Scarborough Clinic	WA	SOUTH WEST BBB
MBR0005	PR002	GRP01	GCAT3	Atlantic Health	ME	EASTERN AAA
MBR0006	PR004	GRP02	GCAT1	Camden Clinic	VA	NORTH WEST BBB
MBR0007	PR002	GRP01	GCAT2	Atlantic Health	ME	SOUTH WEST AAA

FURTHER READING

This is just the beginning of HASH tables; the goal of the paper was to introduce SAS HASH to the beginners as well as experienced SAS programmers who haven't had the chance to explore into SAS HASH. The paper provides couple of basic examples that SAS users can take and start using them in their settings and take advantage of its efficiency and other advantages.

SORT Data using HASH: The hash object is instantiated in the (example 3, marked 1 in the orange box) with the HASH name: GSORT. The hash object is defined to contain the table SASUSER.PROV and will be ordered using the key variable(s) in ascending order. The use of DATASET: constructor method should suffice to load the entire table into the hash object. The '**ORDERED**' constructor is an optional statement that can be used to specify the order of sorting. Values ('a', 'ascending', 'yes', 'y') for ascending order and ('descending', 'd', 'n', 'no') for descending order.

```
DECLARE HASH GSORT(DATASET: 'SASUSER.GRP', ORDERED:'A');
```

OTHER METHODS: ADD, DELETE, REMOVE, REPLACE, OUTPUT
DEFINEKEY, DEINE DATA, DEFINEDONE and FIND methods are the most popular HASH methods, however there are additional methods available in the SAS HASH objects that are worth mentioning such as ADD, DELETE, REMOVE, REPLACE. The '**ADD**' method is useful when additional key and data values need to be added on the fly to the hash object. '**DELETE**' method simply deletes a hash table from the memory. '**REMOVE**' allows to remove a data line associated with a define key mentioned in the hash table. '**REPLACE**' helps to replace the data values associated with a particular define key stated in the hash. '**OUTPUT**' method helps to write the HASH table to a reusable SAS dataset and to the specified library.

Below code example shows the usage of the above discussed HASH methods. They are shown in the example 3 (block marked 2 in the orange box). 'BEFORE' and 'AFTER' data is also shown after the code section to illustrate the additional HASH methods.

EXAMPLE 3: SORTING and ADDITIONAL HASH METHODS

```
DATA _NULL_;
    IF 1=2 THEN SET SASUSER.PROV SASUSER.GRP;
    IF _N_ = 1 THEN DO;
  DECLARE HASH GSORT(DATASET: 'SASUSER.GRP', ORDERED:'A');     1
    GSORT.DEFINEKEY ('GRP_NAME');
    GSORT.DEFINEDATA ('GRP_NAME','GRP_CATEGORY', 'GRP_ID');
    GSORT.DEFINEDONE ();
    GSORT.OUTPUT (DATASET:'GRP_SORTED');

   DECLARE HASH PSORT(DATASET: 'SASUSER.PROV', ORDERED:'D');
     PSORT.DEFINEKEY ('PROV_STATE');
     PSORT.DEFINEDATA ('PROV_STATE','PROV_NAME', 'PROV_ID');
     PSORT.DEFINEDONE ();
     PSORT.ADD (KEY: 'AL',DATA:'AL',DATA:'BUSCH HEALTH', DATA:'PR005');
     PSORT.REMOVE (KEY:'WA');
     PSORT.REPLACE (KEY: 'NH', DATA:'NY',DATA:'FAIRWAY CLINICS',DATA:'PR003');   2
     PSORT.OUTPUT (DATASET:'PROV_SORTED');
     PSORT.DELETE();

    END;
      SET SASUSER.PROV END=EOF SASUSER.GRP;
RUN;
```

SORTING DATA EXAMPLE

BEFORE		
GRP_CATEGORY	GRP_ID	GRP_NAME
GCAT1	GRP01	NORTH WEST AAA
GCAT1	GRP02	NORTH WEST BBB
GCAT2	GRP01	SOUTH WEST AAA
GCAT2	GRP02	SOUTH WEST BBB
GCAT3	GRP01	EASTERN AAA
GCAT3	GRP02	EASTERN BBB
GCAT3	GRP03	EASTERN CCC

AFTER		
GRP_NAME	GRP_CATEGORY	GRP_ID
EASTERN AAA	GCAT3	GRP01
EASTERN BBB	GCAT3	GRP02
EASTERN CCC	GCAT3	GRP03
NORTH WEST AAA	GCAT1	GRP01
NORTH WEST BBB	GCAT1	GRP02
SOUTH WEST AAA	GCAT2	GRP01
SOUTH WEST BBB	GCAT2	GRP02

ADD, DELETE, REMOVE & REPLACE

BEFORE		
Prov_ID	Prov_Name	Prov_State
PR001	Scarborough Clinic	WA
PR002	Atlantic Health	ME
PR003	Fairway Clinics	NH
PR004	Camden Clinic	VA

AFTER		
Prov_State	Prov_Name	Prov_ID
VA	Camden Clinic	PR004
NY	FAIRWAY CLINICS	PR003
ME	Atlantic Health	PR002
AL	BUSCH HEALTH	PR005

One of the 'errors' we may come across is a duplicate key value. By default the SAS hash object keeps the first value it encounters by default. In certain situations, if one wants to keep the last value of a key REPLACE method: could be used.
Prior to SAS 9.2, there was no way to have multiple items with the same key; we will see an example later on how to maintain multiple items with the same key. Logically one would expect a lookup table to have unique key values; however, since the hash object can be used for more than a simple lookup table it needs a mechanism to deal with duplicates. If you prefer to load your hash object by specifying a DATASET: tag, you can also specify if you want the first or last occurrence kept. By default we get the first occurrence. If we want the last occurrence we can use the DUPLICATE: tag:

```
DECLARE HASH GSORT(DATASET: 'SASUSER.GRP', DUPLICATE:'REPLACE');
```

ADVANTAGES AND LIMITATIONS

Advantages:

a. An alternative to any PROC SQL JOIN or DATA step MERGE that runs faster especially when there is very large dataset and a small dataset involved
b. With HASH, there is no need for preparatory sorting or indexing before joining the table
c. Handles joining multiple tables all in one step even though when they use different key fields
d. Can bring in multiple values from just one lookup operation
e. Guess what, lookup values during hash session can be either numeric or character fields

Limitations:

a. The amount of data that can be loaded into a hash object is the amount of memory available to the SAS session. So larger the lookup table, additional memory may be required.
b. SAS HASH syntax and terminology is different from the conventional SAS syntax which is more intuitive, hence can take some getting used to, before one feels comfortable using HASH.

CONCLUSION:

SAS HASH is an underutilized memory based powerful programming tool in the of SAS tool box collection. This paper introduced the basic elements of SAS HASH; what a hash table means, general syntax and functioning, advantages of using HASH, and a few simple examples. Although the processing time for a SAS programming step might depend on several factors, in my experience, I was able to cut down processing time from hours to minutes using SAS HASH. I hope SAS users across various industries take full advantage of this programming technique to cut down on the processing time for various common coding operations that they handle in SAS such as sorting, table lookup operations, merges etc.

TRADEMARK CITATIONS

REFERENCES

Blackwell, John (2010). Find() the power of Hash - How, Why and When to use the SAS® Hash Object

Carpenter, Art (2012). Carpenter's Guide to Innovative SAS Techniques textbook (Chapter 3.3, pages 117 -130)

Dorfman, Paul (2005). "Data Step Hash Objects as Programming Tools," Proceedings of the SAS Global Forum 2005 (SUGI 30) Conference.

Lafler, Kirk Paul (2011). 'An Introduction to SAS Hash Programming Techniques,' Proceedings of the 2011 South East SAS Users Group (SESUG) Conference

Loren, Judy (2006). "How *Do I Love Hash Tables? Let Me Count The Ways!,*" Proceedings of the Nineteenth Northeast SAS Users Group Conference.

Data Step Concepts: Using the HASH Object: SAS Documentation http://support.sas.com/documentation/cdl/en/lrcon/65287/HTML/default/viewer.htm#n1b4cbtmb049xtn1vh9x4waiioz4.htm

Warner-Freeman, Jennifer (2007). I cut my processing time by 90% using hash tables - You can do it too! North East SAS Users Group (NESUG 2007)

AUTHOR INFORMATION

Your comments and questions are valued and encouraged. Contact the author at
Optum Inc
13625 Technology Drive,
Eden Prairie, MN 55344
Phone: 952-974-1928
girish.narayandas@optum.com
www.optum.com