BB-121

Move over MERGE, SQL and SORT. There is a faster game in town!

#Hash Table

Karen J. Price, Rock Hill Schools

ABSTRACT

The purpose of this paper and presentation is to introduce the basics of what a hash table is and to illustrate practical applications of this powerful Base-SAS DATA Step construct. We will highlight features of the hash object and show examples of how these features can improve programmer productivity and system performance of table lookup and sort operations. We will show relatively simply code to perform typical “look-up” match-merge usage as well as a method of using the hash object to add or remove records.

INTRODUCTION

Looking for a value in one data set when given a value from another is a core operation for many SAS programs. This is of critical interest when we are faced with a dataset that has missing data. For instance, we have a file with test scores for students, but for various reasons, data for the students’ teacher and/or course name/number were not populated in the file. Since we have access to the students’ course registration in a separate database file, we would like to fill in the missing information.

With many lookup mechanisms, the value being looked for is called a "key". The "key" provides the means to access the data. 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 where it is created. A hash record consists of two parts: they key and the data. Once a hash object is loaded with records, a lookup occurs 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 populated 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.

There are numerous reasons to utilize the hash object including:

- Key lookup occurs in memory which is far more efficient in terms of processing
- When a key lookup occurs, only a small subset of the records are searched because it is only searching a listing of the key values.
- The key and data parts of a record can consist of more than one value so there is no need to format or concatenate variable values to create the key or the data.
- The hash object allocates memory as records are added which means the hash object only assigns as much memory as it needs and the number of records that can be stored is only limited by the amount of RAM available to SAS.
- When loading a hash object from a data set, the data set does not need to be sorted or indexed (this was a major advantage for me over multiple proc sorts).

The first section of this paper introduces the functionality of the hash object with an example that loads a hash object with records and performs lookup operations. Second, we see how using hash tables we can add or replace records.

USING THE HASH OBJECT

In using the hash object as an in-memory look-up table, there are several components that must be understood. First, in the example shown here, we have a base table containing our test data. In this table we have missing values for a number of students in two of the fields (mathteachnbr and mathsectionid).

Work.testdata

Student_number	Name	MathScore	Mathteachnmbr	MathSectionid

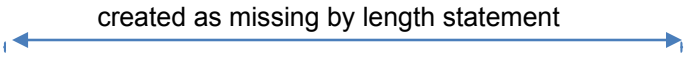
We have a second dataset containing information for these variables with missing information for the math teachers and courses.

Work.math

student_number	teachernumber	course_number	expression

In the compile phase of the DATA Step, we define the lengths for our program data vector (PDV) for the base table and empty variables are created.

PDV



Permid	Name	MathScore	Mathteachnmbr	MathSectionid	student_number	teachernumber	course_number	expression

```
data missing;
length student_number 8 teachernumber $12 course_number $12 expression $12;
```

Next, we populate the hash object in the execute phase of the DATA Step. Here, we declared the name m for the hash. In the execute phase, the hash is populated when it hits the first observation for the key.

```
if _n_ eq 1 then do;

declare hash m (dataset: "math");
```

Hash tables are objects and we use 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 we see a set of parentheses. This is where you put the specifications for the method. Here, we are defining the key as student_number. DefineKey is the method and 'student_number' is a specification expected by the method DefineKey.

```
rc=m.definekey("student_number");
```

Here, using the method DefineData we then identify the fields to use as data in the hash that we will then later populate in the PDV with the find call. Then, we complete the hash table definition with the DefineDone method. Using call missing, we initialize the lookup variables in the PDV. This will prevent us from receiving a warning message about the variables being uninitialized. We use the rc=m.find(key:student_number) to populate the PDV with the data from the hash. Of note, The RC field is an abbreviation for ReturnCode. In reality, you can use another variable name here as this is our means to test the success or failure of the step. As a novice, I have always stuck with rc and I must admit that was, in part, because I did not learn until later what rc represented!

```
rc=m.definedata ("teachernumber", "course_number","expression");
rc=m.definedone();
call missing (student_number,teachernumber,course_number,expression);

set work.testdata;
student_number=input(permid,12.);
rc=m.find(key:student_number);
```

In this example, we are then using the data from the populated PDV to populate the mathteachnbr and mathsectionid variables. We only use the teachernumber data from the hash where the return call was a success (if rc eq 0) and there was no data in the current mathteachnbr field. Additionally, in cases where we had no mathsectionid in the original base table, we use the data from course_number and expression to create the matchsectionid using a compress function.

```
if rc eq 0 and mathteachnbr eq "" then do;
        mathteachnbr=teachernumber;
        mathsectionid=compress(course_number||expression);
        end;
run;
```

CONCLUSION

In this paper, we demonstrated the basic merge function of hash tables. Admittedly, I am a novice to this technique and recognize there are far more sophisticated uses of hash table processing. In such, this paper provides a basic introduction to the hash. In our work, we used multiple hash tables to populate data for a variety fields with missing data. The ability to utilize hash tables eliminated the need for multiple sorts or indexing. In many instances, users may not view the gain in performance as a sufficient reason to use hash tables, but when dealing with very large data sets or situations where multiple sorts would be required, the hash provides a brilliant solution.

REFERENCES

Lafler, Kirk Paul. “An Introduction to SAS Hash Programming Techniques”. Proceedings from the 2011 Southeast SAS User Group Meeting. Available at

http://analytics.ncsu.edu/sesug/2011/BB08.Lafler.pdf

Loren, Judy. “How Do I Love HASH Tables? Let me Count the Ways!” Proceedings from the 2008 SAS Global Forum. Available at

http://www2.sas.com/proceedings/forum2008/029-2008.pdf

ACKNOWLEDGEMENTS

CONTACT INFORMATION

Your comments and questions are valued. Please feel free to contact the author at:

Karen J. Price, Research Specialist
Rock Hill School District 3
660 N. Anderson Road
Rock Hill, SC 29731
(803)981-1082
kprice@rhmail.org