www.sheepsqueezers.com

Using the SAS Hash and Hash Iterator Objects

This presentation as well as other presentations and documents found on the sheepsqueezers.com website may contain quoted material from outside sources such as books, articles and websites. It is our intention to diligently reference all outside sources. Occasionally, though, a reference may be missed. No copyright infringement whatsoever is intended, and all outside source materials are copyright of their respective author(s).

Table of Contents

INTRODUCTION

This document introduces two new SAS features – the Hash Object and the Hash Iterator Object. These features allow you to quickly retrieve *data* based on a lookup *key*. That is, given a lookup key – say, an NDC_CODE – the Hash Object returns any associated data for that NDC_CODE, say the brand name (BN). Your lookup key can be any numeric or character variable, for example, PATIENT_KEY, NDC_CODE or ICD9_CODE. The value can be numeric or character as well, say PAT_GENDER (character), BRAND_NAME (character) or PAT_AGE (numeric).

Why Not a SORT-MERGE or PROC FORMAT CNTLIN?

So, why would you use this method instead of a SORT-MERGE, or a PROC FORMAT CNTLIN? Actually, it depends on the size of your datasets. If one of the datasets is very large, sorting could take quite a long time. If the dataset containing your lookup keys and data – say PATIENT_KEY and PAT_GENDER – using a PROC FORMAT could take up a lot of system memory and processing time to create. Also, at least for me, the rigmarole of creating the CNTLIN dataset used with PROC FORMAT is annoying and the extra code for the SORT-MERGE can be repetitive and tedious. The Hash Object features make use of your lookup dataset without a lot of extra coding and, thus, can reduce programming time.

What's a Hash?

The verb *to hash* means *to chop (as meat and potatoes) into small pieces*. For our purposes, the Hash Object takes a SAS dataset and chops it up into several pieces; that is, the rows of the dataset are broken out into several buckets. The number of buckets, by default, is 256, but you can specify how many buckets you want based on the number of rows in your lookup dataset. (More on that later.)

So, how does this help speed up lookups? Without getting too technical, each lookup key is passed through a hash *function* which maps the key to one and only one bucket. The bucket is then searched to find the data associated with that key. An example of this bucket idea is the telephone book. Say you want to call *Johann Rotnibbler*. Your brain automatically creates a hash function by taking *Rotnibbler* and mapping it to the letter *R*, the first letter of *Rotnibbler*. You then turn to the *R*'s in the phone book and you begin to look for *Rotnibbler*. If you did not map *Rotnibbler* to *R*, you'd start from the first page of the telephone book and search sequentially until you find the phone number for *Rotnibbler, Johann*. Using the hash object allows you to search through smaller amounts of data rather than all of the data. Let's agree never to say the word *Rotnibbler* again, okay?

One important note is that the Hash Object expects there to be *one and only one* observation per lookup key(s). Any duplicates are not put in the Hash Object. Check the SAS Log carefully when using the SAS Hash Object!

What are the Steps to Using a Hash Object?

In order to use a hash object, you have to first create – also called *instantiate* – a hash object. You then populate the hash object with data from your smaller lookup dataset. You then declare one or more variables as lookup keys and additional data. Then, you use the lookup keys from your big dataset to find the data in the hash object. The result is an additional column (or columns) of data

pulled from the hash object and added as an additional column (or columns) to your big dataset. Once you are done with the hash object, you can then destroy it. The act of destroying a hash object removes it from memory and you cannot use it again after that.

What is the Lifetime of a Hash Object?

A hash object exists only during a single data step. That is, you cannot create a hash object in one data step and use it in another data step. Also, hash objects are not used in SAS Procs.

Where Do I Use the Hash Object?

You create, use and destroy the hash object in a data step. Hash objects are not used in Procs. You can create one or more hash objects in a single data step. Hash objects are automatically destroyed after the data step ends, so you cannot create a hash object in one data step and use it in another data step.

Why is it called a Hash *Object*? What's an *Object*?

The word *object* comes from Object Oriented Programming (OOP). OOP programming is different from the *procedural programming* that we do everyday in SAS, Microsoft SQL Server's T-SQL, Microsoft Visual Basic for Applications, etc., and it is different from the *functional programming* we occasionally do in Microsoft Excel when we use `=SUM(A1..A7)`, etc.

In *procedural programming*, we create one or more procedures (just think of SAS macros) each of which performs a specific task, such as pull data from the database, match data against patient master, summarize data and produce reports. Along with procedures, you can also create additional pieces of information stored in global variables (macro variables).

In *functional programming*, instead of using procedures we create functions to perform specific tasks. You've seen examples of functional programming in Microsoft Excel using functions like SUM(), AVERAGE(), etc. You can also use Microsoft Visual Basic for Applications to create your own functions and use them in the cells of a spreadsheet.

In *object oriented programming*, an *object* contains (or *encapsulates*) procedures and additional pieces of information. In OOP terminology, procedures are called *methods* and the pieces of information are called *properties* or *attributes*.

An example of an object is a car. Cars have several methods such as *start engine* (START_ENGINE method), *turn on windshield wipers* (RUN_WIPERS method) and *turn on radio* (RADIO_ON method). Cars have several attributes such as *color of the car* (COLOR attribute), *engine size in liters* (SIZE attribute), *number of sparkplugs* (PLUGS attribute), *number of doors* (DOORS attribute), etc.

As mentioned above, in order to use the Hash Object you have to *instantiate* it. That is, you have made a live copy of the object which you can use. An object itself is just a definition and does not really exist. This idea is similar to creating variables in Visual Basic for Applications. If you need a string variable and an integer variable, you create them using the DIM statement and reference the String or Integer data types:

```
Dim strPATIENTGENDER as String
Dim intPATIENTKEY as Integer
```

You never use the String or Integer data types alone in your VBA programs. You must create a variable with the data type String or Integer. In our case, we're creating a variable of data type Hash Object.

To continue our Car example, let's create a variable called MyJalopy of data type Car Object:

```
Declare Car MyJalopy;
```

Just like String or Integer variables, the statement above declares that the variable MyJalopy exists and is of type Car, but objects need to be initialized. Here is how you initialize the object MyJalopy:

```
MyJalopy = _NEW_ Car();
```

In order to execute methods, you have to use *dot notation*. That is, you specify the name of the instantiated object, MyJalopy in this case, put a dot (period), and then type the name of the method you want to execute. So, let's start MyJalopy's engine:

```
MyJalopy.START_ENGINE();
```

Suppose we want to change the color of MyJalopy to green. We can set the COLOR attribute like this:

```
MyJalopy.COLOR=”green”;
```

These ideas are the same for the Hash Object and Hash Iterator Object, as we will see in the next section.

Annotated Example #1

Suppose we have a SAS dataset called PATKEYS containing a list of PATIENT_KEYs. To this dataset, we want to add each patient's age, gender and three-digit zip code. Let's assume we have this information in a SAS dataset called PATIENT_INFO with the following columns: PATIENT_KEY, PAT_AGE, PAT_GENDER, and PAT_ZIP3. Here is how you would use the SAS Hash Object to add the PAT_AGE, PAT_GENDER, and PAT_ZIP3 to your SAS dataset PATKEYS.

```
/* Create PATKEYS                                                    */
data PATKEYS;
 length PATIENT_KEY 8;
 PATIENT_KEY=1;output;
 PATIENT_KEY=2;output;
 PATIENT_KEY=3;output;
 PATIENT_KEY=4;output;
 PATIENT_KEY=5;output;
run;

/* Create PATIENT_INFO                                               */
data PATIENT_INFO;
 length PATIENT_KEY PAT_AGE 8 PAT_GENDER $ 1 PAT_ZIP3 $ 3;
 PATIENT_KEY=1;PAT_AGE=34;PAT_GENDER='M';PAT_ZIP3='123';output;
 PATIENT_KEY=2;PAT_AGE=44;PAT_GENDER='M';PAT_ZIP3='223';output;
 PATIENT_KEY=3;PAT_AGE=54;PAT_GENDER='F';PAT_ZIP3='323';output;
 PATIENT_KEY=4;PAT_AGE=64;PAT_GENDER='F';PAT_ZIP3='423';output;
run;

/* Use Hash Object to lookup PATIENT_KEY and get PAT_AGE,PAT_GENDER and PAT_ZIP3 */
data PATKEYS_WITH_INFO(drop=rc);

 /* You must define your data variables here even though they are already defined */
 /* in your dataset PATIENT_INFO. The Hash object will complain if you do not     */
 /* do this.                                                                      */
 length PAT_AGE 8 PAT_GENDER $ 1 PAT_ZIP3 $ 3;

 /* SET the list of PATIENT_KEYs here.                               */
 set PATKEYS end=lastcase;

 /* Only necessary to declare a Hash Object once…when _N_=1          */
 if _n_=1 then do;

  /* Declare the variable oHashPAT as a Hash Object                  */
  Declare Hash oHashPAT;

  /* Initialize the Hash Object oHashPAT.  Note that the information */
  /* contained in PATIENT_INFO is being pulled into the Hash here.   */
  /* HASHEXP and DATASET are explained below.                        */
 oHashPAT=_new_ Hash(hashexp:3,dataset:'PATIENT_INFO');

  /* Define the Lookup Key.  Here, PATIENT_KEY is our lookup key.    */
  oHashPAT.DefineKey('PATIENT_KEY');

  /* Define the Data associated with the lookup key.  Here, PAT_AGE, */
  /* PAT_GENDER, and PAT_ZIP3 is the data.                           */
  oHashPAT.DefineData('PAT_AGE','PAT_GENDER','PAT_ZIP3');

  /* Tell SAS you are now done defining the lookup keys and data.    */
  oHashPAT.DefineDone();

 end;

 /* Next, lookup this PATIENT_KEY from the dataset PATKEYS and pull  */
 /* back the PAT_AGE, PAT_GENDER, and PAT_ZIP3.                      */
```

…code continued on next page…

```
 /* Note that variable RC.  This is the return code from the Find    */
 /* Method.  If it is zero, then Find found the PATIENT_KEY in the   */
 /* hash object oHashPAT; otherwise, you can delete the row as we do */
 /* here or you can set PAT_AGE, PAT_GENDER, PAT_ZIP3 to defaults.   */
 rc=oHashPAT.Find(key:PATIENT_KEY);
 if rc=0 then output;
 else delete;

 /* Delete the Hash object when done with it                         */
 if lastcase then oHashPAT.Delete();
run;

proc print data=PATKEYS_WITH_INFO;
 var _all_;
run;
```

Here is the dataset PATKEYS_WITH_INFO:

Obs	PAT_AGE	PAT_GENDER	PAT_ZIP3	PATIENT_KEY
1	34	M	123	1
2	44	M	223	2
3	54	F	323	3
4	64	F	423	4

Note that PATIENT_KEY "5" is not in the dataset PATKEYS_WITH_INFO. This is because that particular patient was not in the PATIENT_INFO dataset and, hence, not in the hash object. This is ultimately controlled by using the return code "rc" after the Find method is executed by using the SAS *output* statement.

What's the Hash Object Attribute HASHEXP?

As mentioned above, the Hash Object breaks up a dataset into buckets. By default, SAS creates 256 buckets, but you can control this by using the Hash Object attribute HASHEXP. This attribute takes a positive integer *N* and SAS will create 2^N buckets. There is no formula you can use to determine the number of buckets which will give you the fastest runtime, so you'll have to try a few *N* values to determine what is best for your situation.

As you can guess, there's a balancing act between the number of buckets and runtime. It's **not** true that the more buckets the faster the runtime.

Also, if you believe you want to use a specific number of buckets (denoted by B), here is how you can calculate the N needed for HASHEXP:

$2^N=B$

→ $LOG_2\ (2^N) = LOG_2(B)$

→ $N = LOG_2(B)$

→ $N= LOG_{10}(B)/LOG_{10}(2)$

In SAS, the function LOG() is equal to $LOG_{10}()$.

Note that the number N in this case will most likely not be an integer, so you'll have to round it to the nearest integer. The largest N you can specify is 16 for 65536 buckets.

What's the Hash Object Attribute DATASET?

The Hash Object attribute DATASET allows you to specify a pre-existing SAS dataset to be used by the Hash Object. In Annotated Example #1 above, we used the dataset PATIENT_INFO to read in patient-level attributes. You can specify any work SAS dataset by entering the name of the dataset, or you can specify a permanent SAS dataset by prepending the libname before the dataset. For example, if the dataset PATIENT_INFO were in a subdirectory referenced by the libname PATS, then the Hash Object DATASET attribute would look like this:

```
oHashPAT=_new_ Hash(dataset:'PATS.PATIENT_INFO');
```

Why Does the SAS Log Say My Variables are Uninitialized?

When you use the SAS Hash Object to bring in additional data based on your key(s), and you've used a SAS Length (as required), SAS will complain that your variables are uninitialized. For example, here is part of the SAS Log from Annotated Example #1:

```
NOTE: Variable PAT_AGE is uninitialized.
NOTE: Variable PAT_GENDER is uninitialized.
NOTE: Variable PAT_ZIP3 is uninitialized.
```

Since you are not programmatically setting these three variables to a value, like PAT_AGE=75; SAS thinks they are missing. You can get rid of these messages by using the SAS CALL MISSING() function when after the declaration section. Here is an example:

```
/* Only necessary to declare a Hash Object once…when _N_=1          */
 if _n_=1 then do;

  /* Declare the variable oHashPAT as a Hash Object                 */
  Declare Hash oHashPAT;

  /* Initialize the Hash Object oHashPAT.  Note that the information */
  /* contained in PATIENT_INFO is being pulled into the Hash here.   */
  /* HASHEXP and DATASET are explained below.                        */
 oHashPAT=_new_ Hash(hashexp:3,dataset:'PATIENT_INFO');

  /* Define the Lookup Key.  Here, PATIENT_KEY is our lookup key.    */
  oHashPAT.DefineKey('PATIENT_KEY');

  /* Define the Data associated with the lookup key.  Here, PAT_AGE, */
  /* PAT_GENDER, and PAT_ZIP3 is the data.                           */
  oHashPAT.DefineData('PAT_AGE','PAT_GENDER','PAT_ZIP3');

  /* Tell SAS you are now done defining the lookup keys and data.    */
  oHashPAT.DefineDone();

  /* Use CALL MISSING() to set your data variables to missing.       */
  /* This remove the SAS Log NOTE about unitialized variables.       */
  call missing(PAT_AGE,PAT_GENDER,PAT_ZIP3);

 end;
```

NOTE: You usually don't want to put your lookup key(s) in the CALL MISSING function. One case I found where you can is when a lookup key appears in the both the DefineKey and the DefineData methods.

Annotated Example #2

In this example, we create a Hash Object with multiple lookup keys. Here we access the CustomerDescriptor table to get the LastName and FirstName of the customer based on the lookup keys CUSTOMER_KEY and CUSTOMER_DESCRIPTOR_ NUMBER. Assume that we have a list of CUSTOMER_KEYs and CUSTOMER_DESCRIPTOR_NUMBERs in a work dataset called ListOfCusts.

```
data Customer_Info(drop=rc);
 length FirstName $ 15 LastName $ 20;
 set ListOfCusts end=lastcase;

 if _n_=1 then do;
  Declare Hash objHashPD;
  objHashPD=_new_ Hash(hashexp:5,dataset:'sasin.CustomerDescriptor');
  rc=objHashPD.DefineKey('CustomerKey',CustomerDescriptorNumber');
  rc=objHashPD.DefineData('FirstName','LastName');
  rc=objHashPD.DefineDone();
  call missing(FirstName,LastName);
 end;

 rc=objHashPD.Find(key:CustomerKey,key:CustomerDescriptorNumber);
 if rc=0 then output;
 else delete;

 /* Delete the Hash object when done with it */
 if lastcase then objHashPD.Delete();
run;
```

As you can see above, in order to use multiple keys in a SAS Hash Object, you just separate each variable by commas in the DEFINEKEY and FIND methods.

Annotated Example #3

We've all read in a text file using the INFILE and INPUT statements and then subsequently sorted the file based on one or more variables. The SAS Hash Object allows you to create a sorted SAS dataset *while reading in an external file at the same time*. The trick is to create an empty Hash Object – we won't be specifying a SAS dataset in the DATASET attribute – and use the Hash Object attribute ORDERED along with the Hash Object method Add(). In order to save the data stored in the Hash Object to a SAS Dataset, we will use the Hash Object method Output().

```
data _null_(drop=rc);
 length PATIENT_KEY PAT_AGE 8;
 infile "MyPatientData.txt" delimiter="|" end=lastcase;

 if _n_=1 then do;
  Declare Hash objHashPD;

  /* Note the use of the Hash Object Attribute ORDERED.  Here want to ensure*/
  /* that the data is sorted by the key value PATIENT_KEY.                  */
  objHashPD=_new_ Hash(hashexp:1,ordered:'ascending');

  rc=objHashPD.DefineKey('PATIENT_KEY');

  /* Note that we are defining the key PATIENT_KEY in the DefineData method.*/
  /* We do this because the Output() method only outputs the data columns   */
  /* and not the key columns!!                                              */
  rc=objHashPD.DefineData('PATIENT_KEY','PAT_AGE');
  rc=objHashPD.DefineDone();
 end;

 /* Read in a row of data from the external text file.                     */
 input @1 PATIENT_KEY PAT_AGE;

 /* Add this row of data to the Hash Object using the Add() method.        */
 rc=objHashPD.Add(key:PATIENT_KEY,data:PATIENT_KEY,data:PAT_AGE);

 /* Since this is the last record, output the data columns to a SAS dataset */
 /* called MyData and then delete the Hash Object.                         */
 if lastcase then do;

  objHashPD.Output(dataset:'work.MyData');
  objHashPD.Delete();

 end;

run;
```

Note that the Hash Object attribute ORDERED only orders by the lookup key values and not by the data values. Also, since the Output() method only outputs the data values and not the lookup key values, you must include any lookup key(s) as part of the Hash Object data (using the DefineData() method). If not, you will not have those columns in the output SAS dataset.

If you want your output dataset to be sorted in descending order, then specify `ordered:'descending'` instead of `ordered:'ascending'` in the Hash Object parameters.

What Other Hash Object Methods Are There?

So far we've used the Hash Object methods Find(), Add(), and Output(). There are several more we'll mention in passing:

`rc=oHash.Remove(key:keyval_1, key:keyval_2…)`
This method removes a value from the Hash Object based on one or more lookup keys. The return code (rc) will be zero if the remove succeeded; otherwise, a non-zero value will be returned.

`rc=oHash.Replace(key:keyval_1,key:keyval_2…,data:dataval_1,data:dataval_2,…)`
This method replaces the data values based on the lookup keys. The return code (rc) will be zero if the replace succeeded; otherwise, a non-zero value will be returned.

`rc=oHash.Check(key:keyval_1,key:keyval_2…)`
This method checks if the specified lookup keys exist in the hash. The return code (rc) will be zero if the key(s) were found; otherwise, a non-zero value will be returned.

What Other Hash Object Attributes Are There?

The Hash Object has one attribute: the NUM_ITEMS attribute. This returns the number of observations in the Hash Object. One way you can use this information is to set a macro variable containing the number of observations for use later on in your program. This is especially useful if you are using the OUTPUT() method to create a SAS dataset from the data in your Hash Object. For example,

```
…
totrecs=oHash.NUM_ITEMS;
call symput('mTotRecs',left(totrecs));
…
```

What is the Hash Iterator Object?

The Hash Iterator Object is used to access the data stored in a Hash Object one observation at a time either in ascending or descending order (depending on the value of the Hash Object attribute ORDERED). Note that you must have a Hash Object already defined before you can create the Hash Iterator Object. You can think of this as a Microsoft SQL Server T-SQL or Oracle PL/SQL Cursor. In the following annotated (admittedly contrived) example, we will create a Hash Object from a pre-existing SAS dataset containing drug name (DRUG_NAME) per patient (PATIENT_KEY). DRUG_NAME is a character variable with values like LIPITOR, WELLBUTRIN XL, etc. We will then use the Hash Iterator Object to help us create a single character string containing all drug names for each patient delimited by a plus-sign.

Annotated Example #4

```
/* Create a list of drug names for each patient.                              */
data PATIENT_DRUG_USAGE(keep=PATIENT_KEY DRUG_NAME);
 length PATIENT_KEY 8 DRUG_NAME $ 50;
 infile cards;
 input @1 PATIENT_KEY       2.
       @5 DRUG_NAME   $char50.;
cards;
01  LIPITOR
01  WELLBUTRIN
02  VIAGRA
02  CIALIS
02  ASPIRIN
02  PRILOSEC
03  ZOMED
03  EXCEDRIN
03  LOTRIMIN
03  NEXIUM
;
run;

data PATIENT_CONCOMITANCY(keep=PREVIOUS_PATIENT_KEY DRUGS_CONCOM
                          rename=(PREVIOUS_PATIENT_KEY=PATIENT_KEY));
 length PATIENT_KEY 8 DRUG_NAME $ 50 DRUGS_CONCOM $ 200;

 if _n_=1 then do;
  Declare Hash objHashPD;
  Declare Hiter objHIterPD;
  objHashPD=_new_ Hash(hashexp:2,ordered:'ascending',dataset:'PATIENT_DRUG_USAGE');
  objHIterPD=_new_ HIter('objHashPD');
  rc=objHashPD.DefineKey('PATIENT_KEY','DRUG_NAME');
  rc=objHashPD.DefineData('PATIENT_KEY','DRUG_NAME');
  rc=objHashPD.DefineDone();
  call missing(PATIENT_KEY,DRUG_NAME);
 end;

 /* Go to the first observation in the Hash Object.                           */
 rc=objHIterPD.First();

 /* Initialize DRUGS_CONCOM to blank.                                         */
 DRUGS_CONCOM="";

 /* Set the variable PREVIOUS_PATIENT_KEY to the current PATIENT_KEY.  This is    */
 /* used during the WHILE Loop to determine when we are at a new PATIENT_KEY. You */
 /* can this of this as a poor man`s FIRST.PATIENT_KEY.                       */
 PREVIOUS_PATIENT_KEY=PATIENT_KEY;
 do while(rc=0);

  /* Build our concomitant drug string DRUGS_CONCOM.                          */
  DRUGS_CONCOM=trim(left(trim(left(DRUGS_CONCOM)) || "+" || trim(left(DRUG_NAME))));
```

…code continued on next page…

```
  /* Move to the next observation in the Hash Object.                          */
  /* If there is no next observation, rc is not zero and the loop will not repeat */
  rc=objHIterPD.Next();

  /* If the current PATIENT_KEY is not the same as the last iteration of the loop */
  /* then get rid of the plus-sign in from of DRUGS_CONCOM, output the record to  */
  /* the SAS dataset, set the variable PREVIOUS_PATIENT_KEY to the PATIENT_KEY and*/
  /* reset DRUGS_CONCOM to blank.                                               */
  if (PREVIOUS_PATIENT_KEY ~= PATIENT_KEY) then do;
   DRUGS_CONCOM=substr(DRUGS_CONCOM,2);
   output;
   PREVIOUS_PATIENT_KEY=PATIENT_KEY;
   DRUGS_CONCOM="";
  end;

 end;

 /* Since we are here, the last observation of the Hash Object was found.      */
 /* Note that we just skipped out of the loop and did not issue an OUTPUT.     */
 /* We do that here after getting rid of the initial plus-sign.                */
 DRUGS_CONCOM=substr(DRUGS_CONCOM,2);
 output;

 /* Delete the Hash and Hash Iterator Objects when done with them.             */
 objHashPD.Delete();
 objHIterPD.Delete();
run;

proc print data=PATIENT_CONCOMITANCY;
run;
```

Here is the output of PATIENT_CONCOMITANCY:

Obs	DRUGS_CONCOM	PATIENT_KEY
1	LIPITOR+WELLBUTRIN	1
2	ASPIRIN+CIALIS+PRILOSEC+VIAGRA	2
3	EXCEDRIN+LOTRIMIN+NEXIUM+ZOMED	3

Annotated Example #5 – Hash of Hashes

```
data testdata;
 retain DAYS_SUPPLY (45);

 patient_key=1;brand="Brand A";brand_id=1;svc_date='01JAN2007'd;output;
 patient_key=1;brand="Brand A";brand_id=1;svc_date='01FEB2007'd;output;
 patient_key=1;brand="Brand A";brand_id=1;svc_date='01MAR2007'd;output;
 patient_key=1;brand="Brand A";brand_id=1;svc_date='01APR2007'd;output;
 patient_key=1;brand="Brand A";brand_id=1;svc_date='01MAY2007'd;output;

 patient_key=1;brand="Brand B";brand_id=2;svc_date='01MAR2007'd;output;
 patient_key=1;brand="Brand B";brand_id=2;svc_date='01APR2007'd;output;
 patient_key=1;brand="Brand B";brand_id=2;svc_date='01MAY2007'd;output;
 patient_key=1;brand="Brand B";brand_id=2;svc_date='01JUN2007'd;output;
 patient_key=1;brand="Brand B";brand_id=2;svc_date='01JUL2007'd;output;
 patient_key=1;brand="Brand B";brand_id=2;svc_date='01AUG2007'd;output;
run;

proc sort data=testdata;
 by patient_key brand_id;
run;

data _null_;
 set testdata;
 by patient_key brand_id;
 if _n_=1 then do;

  Declare Hash OHOHASH; /* Hash of Hashes */
  oHOHash=_new_ Hash(hashexp:3);
  oHOHash.DefineKey('BRAND_ID');
  oHOHash.DefineData('OHASH');
  oHOHash.DefineDone();

  Declare Hiter OHITER('OHOHASH');
  Declare Hash OHASH; /* Hash */

 end;

 if first.brand_id then do;
 put "Here we are:" BRAND_ID;
  OHASH = _new_ hash (ordered:'a');
  OHASH.definekey('BRAND_ID','DATES_ON_THERAPY');
  OHASH.definedata('BRAND_ID','DATES_ON_THERAPY');
  OHASH.definedone();
 end;

 do i = 0 to DAYS_SUPPLY;
  DATES_ON_THERAPY=SVC_DATE + i;
  rc=OHASH.Find(key:BRAND_ID,key:DATES_ON_THERAPY);
  if rc~=0 then do;
   OHASH.Add(key:BRAND_ID,key:DATES_ON_THERAPY,data:BRAND_ID,data:DATES_ON_THERAPY);
  end;
 end;

 if last.brand_id then do;
  OHASH.replace();
  OHOHASH.replace();
  put "Last!";
 end;

 if last.patient_key then do;
  ds=1;
  do rc=OHITER.next() by 0 while(rc=0);
   OHASH.output(dataset: 'out'||put(ds,best.-L));
   ni=OHASH.num_items;
   put ni;
```

```
    ds=ds+1;
    rc=OHITER.next();
   end;
  end;

run;
```

What Other Hash Iterator Object Methods Are There?

In the previous example, we used the Hash Iterator Object methods First() and Next(). There are several more methods we'll mention in passing:

`rc=oIHash.Prev()`
This method returns the previous value from the Hash Object. The return code (rc) will be zero if the method succeeded; otherwise, a non-zero value will be returned.

`rc=oIHash.Last()`
This method returns the last value from the Hash Object. The return code (rc) will be zero if the method succeeded; otherwise, a non-zero value will be returned.

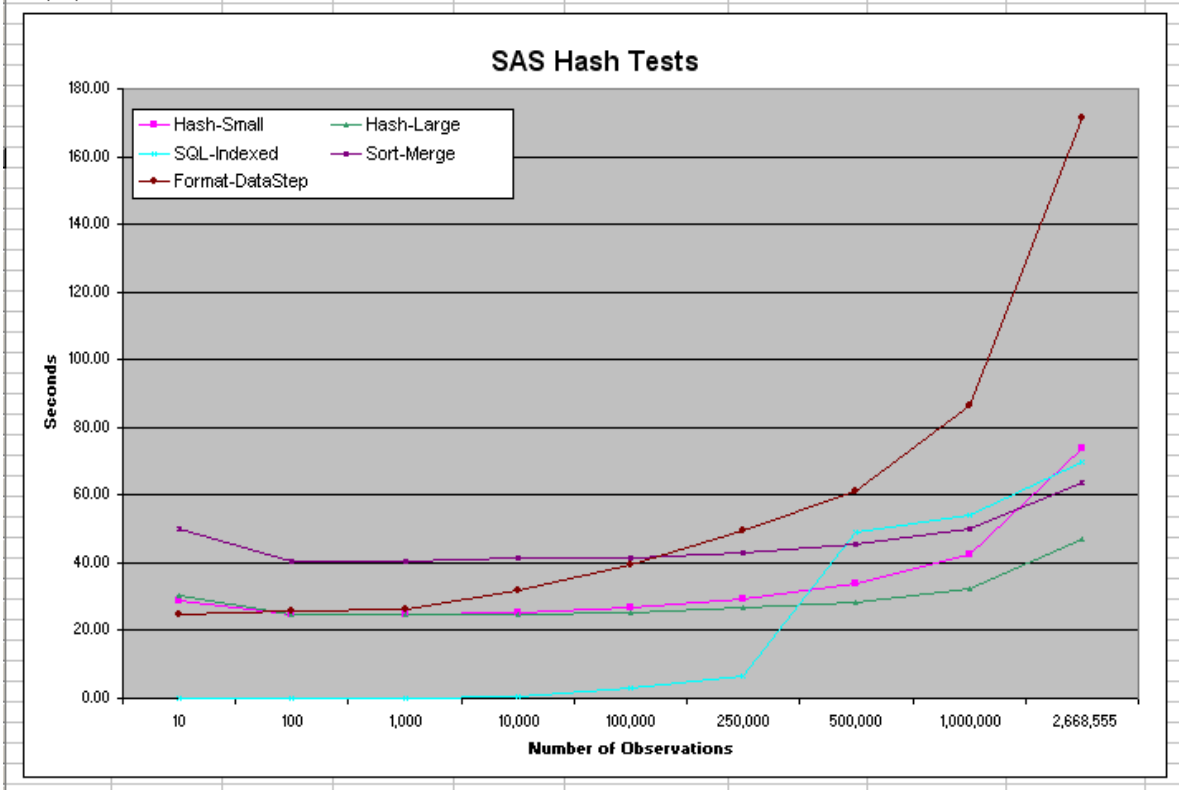
But What of Performance?

All of this Hash Object stuff sounds great, but how's the performance compared to traditional methods? In one test, using a little more than two million observations, the SAS Hash Object outperformed SORT-MERGE, PROC SQL and PROC FORMAT by at least twenty seconds and at most two minutes. Of course, your mileage may vary!

METHOD	**TOTAL TIME (real time seconds)**
Hash Object	45.07
SORT-MERGE	64.32
PROC SQL	70.27
PROC FORMAT	170.92

Here are the results of the same test mentioned above, but using different sized data for the small lookup table. Hash-Small indicates that the smaller dataset was loaded into the Hash Object, while the larger dataset was "set". You'll note that an indexed SQL query performs quite well until about 10 to 15% of the larger dataset. This is consistent with index performance in large databases such as Oracle or SQL Server.

NumOfObs	Hash-Small	Hash-Large	SQL-Indexed	Sort-Merge	Format-DataStep
10	28.77	30.39	0.09	49.98	24.73
100	24.80	24.71	0.08	40.42	25.60
1,000	24.85	24.67	0.12	40.25	26.53
10,000	25.37	24.83	0.53	41.23	31.79
100,000	26.80	25.40	2.83	41.47	39.62
250,000	29.33	26.57	6.58	43.02	49.55
500,000	33.70	28.37	49.07	45.56	61.31
1,000,000	42.55	32.39	53.97	50.21	86.35
2,668,555	73.61	46.86	69.80	63.70	171.30



SAS Hash TipSheet

SAS Institute has prepared a SAS Hash Tip Sheet. It is located at:
support.**sas**.com/rnd/base/datastep/dot/**hash-tip-sheet**.pdf.

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at comments@sheepsqueezers.com.