Paper SM12

Leveraging HASH Technologies for Enhanced Clinical Trial Programming Efficiency

Raviteja Nannapaneni, Cytel, United States
Dmitrii Fedorychev, Cytel, Germany

ABSTRACT

In the ever-evolving landscape of clinical trial programming, the integration of HASH technologies emerges as a pivotal solution. This presentation explores the potential of integrating HASH technologies into clinical trial programming workflows, to expedite processes and alleviate server load. Particularly beneficial for studies with extensive databases and multifaceted integration projects spanning a multitude of trials, HASH technologies offer a promising avenue for optimizing performance. Utilizing the capabilities of HASH opens up avenues for streamlining tasks and optimizing efficiency in clinical trial programming. With this presentation we will delve into the diverse array of tools available to capitalize on HASH technologies, offering insights into how its adoption can revolutionize efficiency and scalability within the clinical trial ecosystem.

Why do we need Hash Objects:

Within the SAS system, the hash object provides an efficient, convenient mechanism for quick data storage and retrieval using lookup keys. The major benefit of hash objects (hash tables) is shortened processing time as they are stored in-memory, which gives fast access to data. They can be invaluable for large dataset manipulation as they would allow for dynamic insertion, deletion, and updating key-values in a Data Step. The specific coding for hash objects can be a little tricky to navigate for those who are not familiar with the syntax. We present a simple, easy-to-use, SAS macro that uses the hash object technology, reaping the benefits of the hash technologies.

Basics requirement for Hash Tables:

A hash object is created within a Data Step using the declare statement. In SAS, the minimum required components to use a hash object are,

DECLARE - This step involves creating the hash object using the declare statement.
DEFINEKEY – Specifies the variable that will act as the key.
DEFINEDATA – Specifies the variable that will be value.
DEFINEDONE – Finalizes the hash object definition. SAS will not know that the hash object definition is complete, and trying to add or find data will result in an error.

Minimum Required Syntax for Hash Objects

```
DATA _NULL_;
    DECLARE HASH N();
    N.DEFINEKEY('KEYVAR');
    N.DEFINEDATA('VALUEVAR');
    N.DEFINEDONE();

   input ID $ Name $;      /*we are adding to hash object (N) using add() method*/
    N.add();
    datalines;
    101 John
    102 Jane
    103 Joe;

   Set dataset_name      /*Read data from the dataset and add it to the hash object*/
     N.add();
 RUN;
```

Note: Input statement is used to read data from an external source as datalines. The length statement is essential when dealing with character variables in hash objects, as it explicitly defines how much memory will be allocated for the key and value variables.

Hash Object Limitations:

1. Cannot be used in PROC STEP, only possible in DATA STEP.
2. Must be part of _N_=1 observation.
3. Complexity of Implementation, For users who are not familiar with hash objects, the implementation can be complex. Incorrectly managing keys and data types, or misunderstanding how hash objects are used, can lead to inefficiencies, errors, or difficulty in debugging.
4. When dealing with Many-to-many merging, Hash Merge table *only allows first qualifying match.* So, it is very important to define keys correctly. If in case hash object is used for merging, it will not preserve the order of dataset. It is important to manually sort the data after the hash object operation.

Introduction to Macro:

Because HASH syntax can be complex and difficult to digest / ingest and for everyday work. We did develop a macro that leverages HASH technology for day-to-day activities. Below are the details of the macro call.

%hashm (INPUT=, VARLIST=, IDKEY=, WHERCL=, RENAME=);

Parameter	Description
Input	Name of source dataset e.g. SDTM.DM
VARLIST	List of space-separated variables to be merged on, listed in any order but named as they appear on the source dataset
IDKEY	List of space-separated join key variables ***(i.e. PROC SORT by-vars or PROC SQL on-vars)***
WHERCL	Any valid SAS where-clause used to sub select source dataset
RENAME	Used to rename requested variables so that they do not get conflict with existing variables

Example 1: Merging dataset using hash object:

```
data xx;
   set TEST.EX;
   %HASHM(INPUT=TEST.DM, WHERCL=%str(not missing(ARM)), IDKEY=USUBJID SUBJID,
          VARLIST=RFSTDTC, RENAME=%str(RFSTDTC=START_DATE));
run;
```

```
****************************************
*********** HASHM starts ************
****************************************

MPRINT(HASHM):   options mprint nosgen nomlogic;

When you use hash tables you have to define the length and type of the added variables.
The macro does this by using a SET statement on the source table to pick up the attributes.
Here 1=0 will always be false so no records are read from the source SET table, only the attributes.

MPRINT(HASHM):   if 1=0 then do;
MPRINT(HASHM):   set TEST.DM(keep=RFSTDTC rename=(RFSTDTC=START_DATE));
MPRINT(HASHM):   end;
MPRINT(HASHM):   if _n_=1 then do;

Hash merging requires objects on the target dataset (i.e. numeric reference variables) to be able to
use the source.
The macro uses a _TEMPORARY_ numeric array for these variables, dropped on completion by default.

MPRINT(HASHM):   ARRAY obj4rc (2) _TEMPORARY_;
MPRINT(HASHM):   DECLARE hash obj4(DATASET:"TEST.DM(where=(not missing(ARM))
rename=(RFSTDTC=START_DATE))");

These object variables objN and objNrc are unique for each macro call.
The macro uses a global macro variable to assign these N values each time the macro is called.

MPRINT(HASHM):   obj4rc(1)=obj4.defineKey('USUBJID' , 'SUBJID');
MPRINT(HASHM):   obj4rc(1)=obj4.defineData("START_DATE");
MPRINT(HASHM):   obj4rc(1)=obj4.defineDone();
MPRINT(HASHM):   call missing(START_DATE);
MPRINT(HASHM):   end;
MPRINT(HASHM):   obj4rc (2)=obj4.find();
MPRINT(HASHM):   if (obj4rc (2) > 0) then call missing(START_DATE);

****************************************
*********** HASHM ends ************
****************************************
```

From the above macro call we can see,

- No extra code is written.
- No limit for number of variables.
- Macros can be embedded
- Statisticians **love** it!

Example 2: Merging multiple datasets using hash object:

There is no limit to calling macro in a single datastep. In the below example we are merging source dataset with other domains multiple times using *HASHM*.

```
Data ADEG;
  Set SDTM.EG;
        **************First call to merge in data from SDTM.DM*****************;
%HASHM(INPUT=SDTM.DM, IDKEY=USUBJID SUBJID, VARLIST=RFXSTDTC);

        *********Second call to merge in data from SDTM.DS for EOT Date********;
%HASHM(INPUT=SDTM.DS, IDKEY=USUBJID SUBJID,VARLIST= DSSTDTC, WHERCL=%STR(DSSCAT='END OF
STUDY'),RENAME=%STR(DSSTDTC=EOSDTC);

        *********Third call to merge in data from SDTM.DS for EOS Date*********;
%HASHM(INPUT=SDTM.DS, IDKEY=USUBJID SUBJID,VARLIST= DSSTDTC,
WHERCL=%STR(DSDECOD='RANDOMIZED'),RENAME=%STR(DSSTDTC=RANDDTC);

        ********Forth call to merge in data from SDTM.SV to dates***********;
%HASHM(INPUT=SDTM.SV, IDKEY=USUBJID SUBJID VISITNUM, VARLIST=SVSTDTC, SVENDTC);

EGDY=(EGDTC-DM_RFSTDTC)+(EGDTC>=DM.RFSTDTC);
run;
```

Example 3: Performance comparison

As mentioned earlier, using hash objects will have quick turnaround and improve performance when we need to perform lookup or merging on large data. Below example proves that merging by Hash objects is significantly faster than sort-and-merge or PROC SQL techniques. In the following example we are trying to derive baseline values for ADLB dataset which has large number of records.

Normal sort-and-merge method:

```
data test1(keep=studyid usubjid paramcd basetype aval rename=(aval=base));
   set test.adlb(where=(ablfl='Y'));
run;
```

```
NOTE: DATA statement used (Total process time):
      real time           5.79 seconds
      cpu time            3.98 seconds
```

```
proc sort;
   by studyid usubjid paramcd basetype;
run;
```

```
NOTE: There were 221837 observations read from the data set WORK.TEST1.
NOTE: The data set WORK.TEST1 has 221837 observations and 5 variables.
NOTE: Compressing data set WORK.TEST1 decreased size by 7.89 percent.
      Compressed is 35 pages; un-compressed would require 38 pages.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.25 seconds
      cpu time            0.31 seconds
```

```
proc sort data=test.adlb out=test2;
   by studyid usubjid paramcd basetype;
run;
```

```
NOTE: There were 1683381 observations read from the data set TEST.ADLB.
NOTE: The data set WORK.TEST2 has 1683381 observations and 143 variables.
NOTE: Compressing data set WORK.TEST2 decreased size by 72.88 percent.
      Compressed is 2481 pages; un-compressed would require 9149 pages.
NOTE: PROCEDURE SORT used (Total process time):
      real time           14.22 seconds
      cpu time            17.20 seconds
```

```
data test3;
   merge test1 test2;
   by studyid usubjid paramcd basetype;
run;
```

```
NOTE: There were 221837 observations read from the data set WORK.TEST1.
NOTE: There were 1683381 observations read from the data set WORK.TEST2.
NOTE: The data set WORK.TEST3 has 1683381 observations and 143 variables.
NOTE: Compressing data set WORK.TEST3 decreased size by 72.90 percent.
      Compressed is 2479 pages; un-compressed would require 9149 pages.
NOTE: DATA statement used (Total process time):
      real time           12.99 seconds
      cpu time            12.98 seconds
```

***Using Hash Merge*:**

```
data test2;
   set test.adlb;
 %HASHM(INPUT=TEST.ADLB, WHERCL=%str(ABLFL='Y'),
 IDKEY=%%STR(USUBJID PARAMCD BASETYPE), VARLIST=AVAL, RENAME=%str(AVAL=AVAL_BASE));
run;
```

```
There were 221837 observations read from the data set TEST.ADLB.
WHERE ablfl='Y';
There were 1683381 observations read from the data set TEST.ADLB.
The data set WORK.TEST2 has 1683381 observations and 144 variables.
Compressing data set WORK.TEST2 decreased size by 72.86 percent.
Compressed is 2497 pages; un-compressed would require 9199 pages.
DATA statement used (Total process time):
real time           18.80 seconds
cpu time            17.06 seconds
```

Real time: 18.80 Seconds

```
data test2;
   set test.adlb;
run;
```

```
NOTE: There were 1683381 observations read from the data set TEST.ADLB.
NOTE: The data set WORK.TEST2 has 1683381 observations and 143 variables.
NOTE: Compressing data set WORK.TEST2 decreased size by 72.88 percent.
      Compressed is 2481 pages; un-compressed would require 9149 pages.
NOTE: DATA statement used (Total process time):
      real time           14.62 seconds
      cpu time            12.81 seconds
```

Method	*Total Time taken*
Data step Merge	**33.25 seconds**
Hash Table Merge	**4.18* Seconds**

Note: After excluding time taken for initiation of SAS dataset (which took 14.62 seconds).

CONCLUSION

In conclusion, hash objects offer an efficient method for managing and processing large datasets. By leveraging the capabilities of the SAS Hash Object, users can significantly improve performance in data lookup, retrieval, and manipulation tasks. These hash objects provide a flexible solution for in-memory data storage, compared to traditional methods like sorting or merging datasets. Furthermore, their ability to handle complex data structures optimally makes them invaluable.

REFERENCES

1. Introduction to SAS® Hash Objects, Chris Schacherer, https://support.sas.com/resources/papers/proceedings15/3024-2015.pdf
2. Comparison of different ways using table lookups on huge tables, Ralf Minkenberg https://www.lexjansen.com/phuse/2007/cs/CS06.pdf
3. Hash Table Look-up: Easier than You'd Think, Virginia Blum, https://www.lexjansen.com/nesug/nesug13/99_Final_Paper.pdf
4. Getting Started with DATA Step Hash Objects, Joshua M. Horstman https://www.lexjansen.com/sesug/2021/SESUG2021_Paper_98_Final_PDF.pdf
5. A Hands-on Introduction to SAS® DATA Step Hash Programming Techniques, Kirk Paul Lafler https://support.sas.com/resources/papers/proceedings20/4415-2020.pdf

ACKNOWLEDGMENTS

Thank you to our colleagues at Cytel for their collaboration in the development of this important tool.

CONTACT INFORMATION

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

Raviteja Nannapaneni
Manager, Statistical Programming
Cytel, Inc.
675 Massachusetts Ave
Cambridge, MA 02139
Raviteja.Nannapaneni@cytel.com
www.cytel.com