



Leveraging HASH Technologies for Enhanced Clinical Trial Programming Efficiency

Raviteja Nannapaneni

19 March, 2025



Why *HASHM*?

Because I:

- *Prefer simplicity*: Avoid writing excessive syntax for sorting and merging data just to transfer a few variables between datasets.
- *Dislike confusion*: Not fond of deciphering inherited code cluttered with sort-and-merge steps, leading to unclear datasets and uncertainty about their necessity for my study.

What is hash table

- Hash tables are memory resident objects that can be thought of as “lookup tables”.
- Because these tables are memory resident, they offer a very efficient (i.e. SUPER QUICK!) means of retrieving data.
- SAS has an implementation of hash tables that can be used to retrieve data for merging or ordering SAS datasets.
- Data can be merged in from these memory-resident objects very quickly, WITHOUT THE NEED TO SORT EITHER DATASETS.

How fast are hash table merges?

- A merge taking many minutes using traditional SAS sort-and-merge or SAS SQL can be completed in seconds using hash tables.
- While the time saving may be incidental for small-dataset, phase 1 studies but it can be significant for phase 3 studies with larger datasets.
- There is some benchmarking of efficiency in:
<http://www.lexjansen.com/phuse/2007/cs/CS06.pdf>

How do you use 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.

Basic 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;
```

HASHM macro:

%Hashm(**INPUT=**, **VARLIST=**, **IDKEY=**, **WHERCL=**, **RENAME=**);

INPUT, VARLIST, IDKEY, WHERCL – required parameters

RENAME – optional parameter

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>i.e. PROC SORT by-vars or PROC SQL on-vars</i>)
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, WHERECL=%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 l=0 will always be false so no records are read from the source SET table, only the attributes.

MPRINT(HASHM):  if l=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 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:

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

  *****Second call to merge in data from SDTM.DS for EOT Date*****;
%HASHM(INPUT=SDTM.DS, IDKEY=USUBJID SUBJID,VARLIST= DSSTDTC, WHERECL=%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, WHERECL=%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;
```

Hash Objects 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.

Example 3: Performance comparison

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

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

data test3;
  merge test1 test2;
  by studyid usubjid paramcd basetype;
run;
Real time: 33.25
```

VS

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

Real time: 18.80
```

NOTE: Compressed is 35 pages; un-compressed would require 38 pages.
NOTE: DATA statement used (Total process time):
real time 5.79 seconds
cpu time 3.98 seconds

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

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

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

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

Example 3: Performance comparison

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

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

data test3;
  merge test1 test2;
  by studyid usubjid paramcd basetype;
run;
Real time: 33.25
```

VS

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

Real time: 18.80
```

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

Compressed is 35 pages; un-compressed would require 38 pages.
NOTE: DATA statement used (Total process time):
real time 5.79 seconds
cpu time 3.98 seconds

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

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

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

Example 3: Performance comparison

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

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

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

Real time: 33.25
```

VS

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

Real time: 18.80
```

33.25 vs 4.18
More than 8 times faster!

NOTE: There were 221837 observations read from the data set TEST.ADLB.
WHERE ablfl='Y';
NOTE: There were 1683381 observations read from the data set TEST.ADLB.
WORK.TEST2 has 1683381 observations and 144 variables.
The data set WORK.TEST2 decreased size by 72.86 percent.
Compressed is 2481 pages; un-compressed would require 9199 pages.
NOTE: PROCEDURE SORT used (Total process time):
real time 18.80 seconds
cpu time 17.06 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



Thank you
Any questions?

