

Perish the Sort: Using Indexes and Hash Objects for Efficient Programming

Jason A Smith, Argo Analytics Ltd, Cambridge, UK

ABSTRACT

Every SAS® programmer will be well versed in using PROC SORT. Are we sorting data when we do not need to? Unnecessary sorting can be time consuming, particularly when using large datasets. Is there any way that SAS will allow us to classify or merge data without needing to sort it first?

Using indexes and hash objects are techniques that the SAS programmer can use to eliminate (or at least reduce) reliance on the SORT procedure. This paper will present some examples of how to easily include these techniques in your programming and allow you to “perish the sort”.

INTRODUCTION

This paper will look at indexes, hash objects and other programming techniques that you can use to combine or classify data without the need to sort or order the datasets first.

In line with general SAS usage, this paper will refer to indices as “indexes”.

INDEXES

Creating an index allows you to set, merge or summarise a dataset using a BY statement without the need to sort the data first. An index can be easily created at dataset creation, be quicker to code than a sort procedure and can improve the program’s execution time. This section will give examples of creating simple and composite indexes.

When using indexes for the first time it can be useful to set MSGLEVEL=I to provide additional information in the SAS log:

```
options msglevel=i;
```

SIMPLE INDEX

A simple index consists of a single key variable, which can be either character or numeric. The index must be named the same as the variable name.

SIMPLE INDEX EXAMPLE

```
data classfit (index=(name));  
  set sashelp.classfit;  
run;
```

After submitting this in SAS, the log confirms that the index has been created:

```
NOTE: There were 19 observations read from the data set SASHELP.CLASSFIT.  
NOTE: The data set WORK.CLASSFIT has 19 observations and 10 variables.  
NOTE: Simple index name has been defined.  
NOTE: DATA statement used (Total process time):  
      real time           0.01 seconds  
      cpu time            0.01 seconds
```

The data can now be used with a BY statement without the need for the dataset to be sorted:

```
data class;  
  merge sashelp.class classfit;  
  by name;  
run;
```

PhUSE 2016

The SAS log confirms that the index has been used:

```
INFO: Index Name selected for BY clause processing.
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: There were 19 observations read from the data set WORK.CLASSFIT.
NOTE: The data set WORK.CLASS has 19 observations and 10 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

COMPOSITE INDEX

A composite index consists of multiple key variables, which can be any combination of character or numeric. The index must be given a unique name that does not match any existing variable name.

COMPOSITE INDEX EXAMPLE

```
data class (index=(sexage=(sex age)));
  set sashelp.class;
run;
```

The SAS log confirms that a composite index has been created:

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.CLASS has 19 observations and 5 variables.
NOTE: Composite index sexage has been defined.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

The data can now be used with a BY statement without being sorting:

```
proc means data=class;
  by sex age;
  var height weight;
run;
```

The SAS log confirms that the index has been used:

```
56      proc means data=class;
57      by sex age;
INFO: Index sexage selected for BY clause processing.
NOTE: An index was selected to execute the BY statement.
      The observations will be returned in index order rather than in physical order.
      The selected index is for the variable(s):
Sex
Age
58      var height weight;
59      run;

NOTE: There were 19 observations read from the data set WORK.CLASS.
NOTE: PROCEDURE MEANS used (Total process time):
      real time          0.25 seconds
      cpu time           0.19 seconds
```

PhUSE 2016

MULIPLE INDEXES

Any number of indexes can be created in a data step. This is an example of creating a simple index on variable NAME along with a composite index on variables SEX and AGE:

MULTIPLE INDEXES EXAMPLE

```
data classfit (index=(name sexage=(sex age)));
  set sashelp.classfit;
run;
```

The SAS log confirms that both the simple index NAME and composite index SEXAGE have been created:

```
NOTE: There were 19 observations read from the data set SASHELP.CLASSFIT.
NOTE: The data set WORK.CLASSFIT has 19 observations and 10 variables.
NOTE: Composite index sexage has been defined.
NOTE: Simple index name has been defined.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds
```

UNIQUE INDEX

A unique index can be used to ensure that the key variable(s) are unique for each row. SAS will reject the index and give an error if any duplicate keys exist. It is recommended to use this type of index only when you expect the key to be unique and want SAS to provide an error so that you can check any duplicates.

UNIQUE INDEX EXAMPLE

```
data class (index=(sex/unique));
  set sashelp.class;
run;
```

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.CLASS has 19 observations and 5 variables.
ERROR: Duplicate values not allowed on index Sex for file CLASS.
ERROR: Index creation failed for one or more indexes.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

INDEX MANAGEMENT

An index can also be easily added to (or deleted from) an existing dataset using either PROC DATASETS or PROC SQL.

PROC DATASETS EXAMPLE

```
proc datasets nolist;
  modify classfit;
  index delete name;
  index create sex;
  index create namesex=(name sex)/unique;
quit;
```

```
58      index delete name;
NOTE: Index Name deleted.
59      index create sex;
NOTE: Simple index Sex has been defined.
60      index create namesex=(name sex)/unique;
NOTE: Composite index namesex has been defined.
```

PhUSE 2016

PROC SQL EXAMPLE

```
proc sql;
  drop index sex from classfit;
  create index age on classfit;
  create unique index agetname on classfit(age,name);
quit;
```

```
57          drop index sex from classfit;
NOTE: Index sex has been dropped.
58          create index age on classfit;
NOTE: Simple index age has been defined.
59          create unique index agetname on classfit(age,name);
NOTE: Composite index agetname has been defined.
```

HASH OBJECTS

Hash objects are a type of data structure consisting of key items and data items that allows SAS to efficiently search for data. The hash object is stored in memory and only exists during the execution of the data step in a similar way to a temporary array.

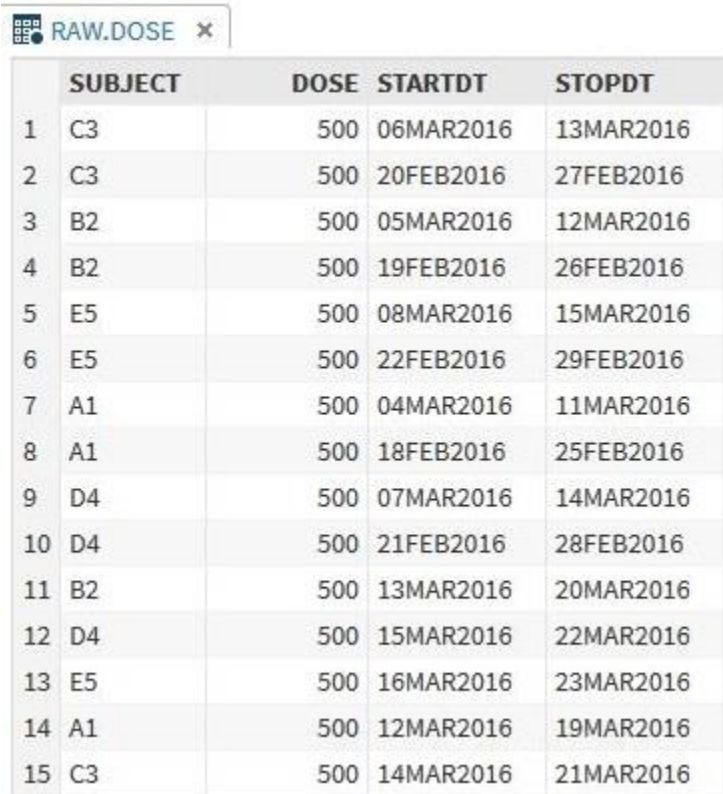
A hash object can be used to combine two existing datasets by loading one dataset into the hash object and by using the key variable(s) to retrieve the data item variables into the other dataset. There is no need for either dataset to be sorted or ordered and the order of the original dataset is unchanged.

This can be very useful in, for example, a raw data-cut program, where the datasets are not sorted and the existing order of the datasets should be retained.

USING A HASH OBJECT TO COMBINE UNSORTED DATASETS

In this example, the requirement is to cut the RAW.DOSE dataset, only keeping subjects in Cohort A. The cohort variable is in the RAW.COHORT dataset. The cohort variable will be added to the CUT.DOSE dataset for confirmation.

The RAW.DOSE dataset contains several records for each subject. The dataset is not sorted:



	SUBJECT	DOSE	STARTDT	STOPDT
1	C3	500	06MAR2016	13MAR2016
2	C3	500	20FEB2016	27FEB2016
3	B2	500	05MAR2016	12MAR2016
4	B2	500	19FEB2016	26FEB2016
5	E5	500	08MAR2016	15MAR2016
6	E5	500	22FEB2016	29FEB2016
7	A1	500	04MAR2016	11MAR2016
8	A1	500	18FEB2016	25FEB2016
9	D4	500	07MAR2016	14MAR2016
10	D4	500	21FEB2016	28FEB2016
11	B2	500	13MAR2016	20MAR2016
12	D4	500	15MAR2016	22MAR2016
13	E5	500	16MAR2016	23MAR2016
14	A1	500	12MAR2016	19MAR2016
15	C3	500	14MAR2016	21MAR2016

PhUSE 2016

The COHORT dataset contains 5 subjects and their assigned cohort. The dataset is not sorted:



	SUBJECT	COHORT
1	E5	A
2	C3	A
3	D4	B
4	B2	B
5	A1	A

HASH OBJECT CODE

The following code will create a new CUT.DOSE dataset containing only subjects in Cohort A. There is no need to sort the datasets and the CUT.DOSE dataset will retain the original order.

```
data cut.dose;
  length SUBJECT $2 COHORT $1;

  if _n_=1 then do;
    declare hash h(dataset:"raw.cohort (where=(cohort='A'))"); ①
    h.defineKey("subject"); ②
    h.defineData("cohort"); ③
    h.defineDone();
    call missing (subject,cohort);
  end;

  set raw.dose; ④
  rc = h.find(); ⑤
  if rc = 0 then output; ⑥
  drop rc;
run;
```

NOTES

- ① read the RAW.COHORT dataset into the hash object, only keeping Cohort A subjects.
- ② define the key variable as SUBJECT. This is the equivalent to the BY variable in a merge. Any number of variables can be included in a comma-separated list.
- ③ define the data item variable(s) that are to be output to the new dataset. Any number of variables can be included in a comma-separated list. This method is optional.
- ④ read in the RAW.DOSE dataset.
- ⑤ h.find() is the method used to retrieve the data from the hash object.
- ⑥ a return code of zero indicates that the find was successful.

SAS LOG AND OUTPUT

The SAS log confirms that only 9 of the original 15 observations have been output:

PhUSE 2016

```
NOTE: There were 3 observations read from the data set RAW.COHORT.  
      WHERE cohort='A';  
NOTE: There were 15 observations read from the data set RAW.DOSE.  
NOTE: The data set CUT.DOSE has 9 observations and 5 variables.  
NOTE: DATA statement used (Total process time):  
      real time          0.02 seconds  
      cpu time           0.02 seconds
```

The cut dataset contains only the Cohort='A' patients and the original order has been retained:

	SUBJECT	COHORT	DOSE	STARTDT	STOPDT
1	C3	A	500	06MAR2016	13MAR2016
2	C3	A	500	20FEB2016	27FEB2016
3	E5	A	500	08MAR2016	15MAR2016
4	E5	A	500	22FEB2016	29FEB2016
5	A1	A	500	04MAR2016	11MAR2016
6	A1	A	500	18FEB2016	25FEB2016
7	E5	A	500	16MAR2016	23MAR2016
8	A1	A	500	12MAR2016	19MAR2016
9	C3	A	500	14MAR2016	21MAR2016

SPD ENGINE

The SAS Scalable Performance Data Engine (SPD Engine) supports the implicit sort when it encounters a BY statement for processing data. This method is particularly beneficial for dealing with large datasets and is only recommended for advanced SAS users.

CLASSFIT has been copied from SASHELP to WORK. This dataset is not sorted as we can see from the first six observations:

	Name	Sex	Age	Height	Weight
1	Joyce	F	11	51.3	50.5
2	Louise	F	12	56.3	77
3	Alice	F	13	56.5	84
4	James	M	12	57.3	83
5	Thomas	M	11	57.5	85
6	John	M	12	59	99.5

If we attempt to use a BY statement on this dataset:

```
data unsorted;  
  set classfit;  
  by name;  
run;
```

PhUSE 2016

SAS will give us an error:

```
ERROR: BY variables are not properly sorted on data set WORK.CLASSFIT.
Name=Louise Sex=F Age=12 Height=56.3 Weight=77 predict=76.488485693
lowermean=67.960050237 uppermean=85.016921149 lower=51.314521735 upper=101.66244965
FIRST.Name=1 LAST.Name=1 _ERROR_=1 _N_=2
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 3 observations read from the data set WORK.CLASSFIT.
WARNING: The data set WORK.UNSORTED may be incomplete. When this step was stopped
         there were 1 observations and 10 variables.
WARNING: Data set WORK.UNSORTED was not replaced because this step was stopped.
```

We can utilise the SPDE implicit sort by re-assigning the WORK library as the USER library using the SPD Engine:

```
libname user spde "%sysfunc(pathname(work))";
```

```
NOTE: Libref USER was successfully assigned as follows:
      Engine:          SPDE
      Physical Name:
      /tmp/SAS_work78580000A4B_localhost.localdomain/SAS_work393E0000A4B_localhost.localdomain/
```

SAS uses the USER library rather than the WORK library as default when it is assigned. As we have assigned the USER library using the SPD Engine, even though it has the same physical location as the WORK library, any data will be implicitly sorted by SAS as required.

If we try the same UNSORTED data step as before:

```
data unsorted;
  set classfit;
  by name;
run;
```

This time the data step has been successful:

```
NOTE: There were 19 observations read from the data set USER.CLASSFIT.
NOTE: The data set USER.UNSORTED has 19 observations and 10 variables.
NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time           0.02 seconds
```

The output dataset is now in NAME order, as we can see from the first six observations:

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83

PhUSE 2016

CONCLUSION

This paper has looked at ways of classifying and merging data without the need to sort the data first.

Using an index can simplify coding by replacing the sort procedure with an index option consisting of as little as two words, as well as potentially reducing execution time.

A hash object can be used to combine two unsorted datasets in a single data step, replacing the need for potentially several uses of the sort procedure.

These techniques can be used to reduce reliance on the SORT procedure resulting in shorter code, quicker and neater programming as well as improved execution time.

RECOMMENDED READING

SAS Institute Inc.: SAS Certification Prep Guide Advanced Programming for SAS9 (SAS Institute Inc., 2011)
Chapter 13: Creating Samples and Indexes

Michele M. Burllew: SAS Hash Object Programming Made Easy (SAS Institute Inc., 2012)

SAS Online Guide to Understanding SAS Indexes:

<http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000440261.htm>

SAS 9.3 Component Objects: Reference

<https://support.sas.com/documentation/cdl/en/lecompobjref/63327/PDF/default/lecompobjref.pdf>

CONTACT INFORMATION

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

Jason A Smith
Argo Analytics Ltd
32 Woodlark Road
Cambridge CB3 0HS, UK
Work Phone: +44 7792 046599
Email: jason@argoanalytics.co.uk
LinkedIn: <http://www.linkedin.com/in/jason-a-smith>
Web: <http://www.argoanalytics.co.uk>

Brand and product names are trademarks of their respective companies.