PharmaSUG 2015 - Paper QT21

Sorting big datasets. Do we really need it?

Daniil Shliakhov, Experis Clinical, Kharkiv, Ukraine

ABSTRACT

Very often working with big data causes difficulties for SAS programmers. As we all know, large datasets can be manipulated and analyzed using SAS®, but these SAS programs may take many hours to run. Often, timelines are short and delivery dates cannot be moved, so decreased running time is critical. One of the most time-consuming tasks for the programmer is sorting large datasets. This paper describes different techniques of quick sorting like using indexes or even how to avoid sorting at all by using hash-objects. The main goal of this paper is to find the best methods for sorting large datasets within SAS depending on the different types of data structure (for example, vertical or horizontal) and different number of variables. To identify the efficiency of different techniques of quick sorting, this paper also provides comparison between the sorting process with and without those techniques. The comparison will show how much resource time-resource is consumed sorting different types of large SAS datasets.

INTRODUCTION

You can ask why the sorting of datasets is an issue? This is not an issue when we are talking about small datasets. But when you need to sort the large dataset (more than 1 million records with a lot of key variables) it may cause some problems. And the main problem is running time. When you work with large datasets, it may take minutes or even hours to run. Let's image the situation when you need to create ADLB dataset for pooling study. During creating this dataset you may need to sort the work datasets a few times: to merge the dataset with subject-level dataset (ADSL), to create baseline flags, to create any derived records or create the analysis flags. In cases where you have few million records in your dataset, it really may take a lot of time to run the program. This paper describes different techniques of sorting SAS datasets.

The techniques described in this paper are:

- Using SAS SORT procedure;
- Using Indexes;
- Using Hash-objects;
- At the end of the paper, I'll compare the time results of different techniques and we will try to find the best one. All provided examples in this paper are based on ADLB dataset with 1 million records that has been created for a pooling study.

To run the SAS codes provided in this paper, SAS 9.2 for Microsoft® Windows® has been used.

PROC SORT

The sort procedure is probably one of the most commonly used SAS procedures. I don't remember any SAS programs without at least one call of PROC SORT.

Let's see how the PROC SORT works:

```
proc sort data=<name_of_your_dataset> <some_options>;
    by <list_of_variables>
run;
```

There is nothing new if you have used SAS for a long time. The SORT procedure is the easiest procedure in SAS and it's more than fine to use this PROC SORT to sort the small datasets. But when you need to sort a very big dataset a few times within one program or even in a few programs, you will probably need to use indexes or even hash objects. Let's see how indexes and hash-objects work.

INDEXES

What is an index? The SAS documentation says that an index is an optional file that you can create for a SAS data file in order to provide direct access to specific observations. There are three main ways to create an index. Indexes can be created using PROC DATASETS, PROC SQL or in the DATA step.

CREATING INDEXES USING PROC DATASETS

One of the ways to create an index is to use PROC DATASETS. Within this SAS procedure you can modify a dataset in a specific library. Please see the simple code that creates a few indexes for ADLB dataset:

```
proc datasets library=analdata;
    modify adlb;
        index create studyid;
        index create usubjid;
        index create lbcat;
        index create paramcd;
        index create avisitn;
        index create avisit;
run;
quit;
```

Provided code above creates 6 simple indexes for study number (studyid), patient number (usubjid), name of category of laboratory test (lbcat), short name of laboratory test (paramcd), numeric patient analysis visit (avisitn), and patient analysis visit (avisitn).

Let's see how much time it took to create 6 indexes using PROC DATASETS. **Error! Reference source not found.** shows the part of log file with time:

```
modify adlb;
         index create studyid;
NOTE: Simple index studyid has been defined.
         index create usubjid;
NOTE: Simple index USUBJID has been defined.
         index create lbcat;
NOTE: Simple index lbcat has been defined.
         index create paramcd;
NOTE: Simple index paramcd has been defined.
         index create avisitn;
NOTE: Simple index avisitn has been defined.
         index create avisit;
NOTE: Simple index avisit has been defined.
run;

NOTE: MODIFY was successful for WORK.ADLB.DATA.
185  quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time           5.45 seconds
      cpu time            8.17 seconds
```

Output 1. Result of using PROC DATASETS to create the INDEXES

CREATING INDEXES USING PROC SQL

The SQL procedure can also be used to create indexes for your SAS dataset:

```
proc sql noprint;
   create index studyid on analdata.adlb;
   create index usubjid on analdata.adlb;
   create index lbcat on analdata.adlb;
   create index paramcd on analdata.adlb;
   create index avisitn on analdata.adlb;
   create index avisit on analdata.adlb;
quit;
```

Let's run this PROC step to see the time difference between PROC DATASETS and PROC SQL.

```
proc sql noprint;
      create index studyid on work.adlb;
```

```
NOTE: Simple index studyid has been defined.
      create index usubjid on work.adlb;
NOTE: Simple index usubjid has been defined.
      create index lbcat on work.adlb;
NOTE: Simple index lbcat has been defined.
      create index paramcd on work.adlb;
NOTE: Simple index paramcd has been defined.
      create index avisitn on work.adlb;
NOTE: Simple index avisitn has been defined.
      create index avisit on work.adlb;
NOTE: Simple index avisit has been defined.
quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           6.53 seconds
      cpu time            7.95 seconds
```

Output 2. Result of using PROC SQL to create the INDEXES

CREATING INDEXES USING DATA STEP

And the last way of creating indexes is using the INDEX dataset option in the DATA statement. Let's see how it works:

```
data analdata.adlb
          (index=(studyid
                  usubjid
                  lbcat
                  paramcd
                  avisitn
                  avisit));
   set analdata.adlb;
quit;
```

Let's run this data step to see how much time it will take to create the indexes using DATA step:

```
data analdata.adlb
            (index=(studyid
                    usubjid
                    lbcat
                    paramcd
                    avisitn
                    avisit));
      set analdata.adlb;
run;

NOTE: There were 1149600 observations read from the data set
      ANALDATA.ADLB.
NOTE: The data set ANALDATA.ADLB has 1149600 observations and 30
      variables.
NOTE: DATA statement used (Total process time):
      real time           54.71 seconds
      cpu time            12.96 seconds
```

Output 3. Result of using DATA step to create the INDEXES

In the examples above simple indexes were created. Simple means that new index created from a single SAS variables. If you need to create an index from two or more variables, this index is called a composite index. In our example it would be better to create 3 composite indexes instead of 6: the first two variables describe a patient (studyid and usubjid), the next two variables describe a laboratory test (lbcat and paramcd), and the last two variables identify an analysis visit (avisitn and avisit). To create composite indexes you can use any of the ways described above. Based on time comparison I would suggest to create indexes within the data step, because it took a lot of time compared to others and we can see the difference. Below you can see the example of creating composite indexes using data step:

```
data analdata.adlb
            (index=(patient=(studyid usubjid)
                    lb_test=(lbcat paramcd)
                    anal_visit=(avisitn avisit)));
      set analdata.adlb;
run;
```

After running this PROC step you can see that it took less time than creating simple indexes.

```
data analdata.adlb
            (index=(patient=(studyid usubjid)
                    lb_test=(lbcat paramcd)
                    anal_visit=(avisitn avisit)));
      set analdata.adlb;
run;

NOTE: There were 1149600 observations read from the data set
      WORK.ADLB.
NOTE: The data set WORK.ADLB has 1149600 observations and 30
      variables.
NOTE: DATA statement used (Total process time):
      real time           23.52 seconds
      cpu time            8.14 seconds
```

Output 4. Result of using PROC SQL to create the composite INDEXES

Now let's look at the difference between PROC SORT and INDEXES. The comparisons provided below:

	PROC SORT	Simple INDEXES			Composite INDEXES DATA STEP
		PROC DATASETS	PROC SQL	DATA STEP	
Actual time	52:20	5:45	6:53	54:71	23:52
CPU time	8:97	8:17	7:95	12:96	8:14

Table 1. Comparison of time usage between PROC SORT and INDEXES

HASH-OBJECTS

Compared to indexes that can be used in any data or procedure steps and even in different programs, hash-objects can be used only in the datastep, and at the end of the datastep the hash object and all its contents disappear. So, what is a hash-object? Hash-objects is a data structure that contains an array of items that are used to map key values (e.g., patient numbers) to their associated values (e.g., patient laboratory result).

The hash-object has quite a complicated syntax with a lot of options. What you need to do is to create, define, fill and then access the hash-object.

To create a hash-object you need to use a declare statement in your data step:

```
declare HASH Hash_Name;
```

Next step is to define the key and other variables:

```
Hash_Name.DefineKey('Key var');
Hash_Name.DefineData('Var1', 'Var2', 'Var3', … , 'Varn');
Hash_Name.DefineDone();
```

When the hash-object is created and defined, it needs to be filled with data. The following syntax should be used:

```
Hash_Name.add();
```

Let's see the whole datastep that will sort our dataset ADLB using hash-objects by STUDYID, USUBJID, LBCAT, PARAMCD, AVISITN, AVISIT:

```
data _null_;
   if 0 then set analdata.adlb;
   if _n_ = 1 then do;
      declare Hash HashSort(ordered:'a');
         HashSort.DefineKey('STUDYID', 'USUBJID', 'LBCAT', 'PARAMCD', 'AVISITN',
                            'AVISIT');
         HashSort.DefineData('STUDYID', 'USUBJID', 'LBCAT', 'PARAMCD', 'AVISITN',
                             'AVISIT', 'AVAL', 'CHG', 'TRT01A');
         HashSort.DefineDone();
   end;
   HashSort.add();
   if eof then HashSort.output(dataset:'sorted_adlb');
run;
```

Let's run this data step to see how much time it will take to sort the dataset ADLB:

```
data _null_;
   if 0 then set work.adlb;
   if _n_ = 1 then do;
      declare Hash HashSort(ordered:'a');
         HashSort.DefineKey('STUDYID', 'USUBJID', 'LBCAT', 'PARAMCD', 'AVISITN',
'AVISIT');
         HashSort.DefineData('STUDYID', 'USUBJID', 'LBCAT','PARAMCD', 'AVISITN',
'AVISIT');
         HashSort.DefineDone();
   end;
   set work.adlb end=eof;
   HashSort.add();
   if eof then HashSort.output(dataset:'sorted_adlb');
run;

NOTE: The data set WORK.SORTED_ADLB has 1149600 observations
      and 6 variables.
NOTE: There were 1149600 observations read from the data set
      WORK.ADLB.
NOTE: DATA statement used (Total process time):
      real time           3.92 seconds
      cpu time            3.16 seconds
```

Output 5. Result of using hash-objects to sort the ADLB dataset

CONCLUSION

SAS has a lot of ways to sort the SAS datasets. One of the commonly used is SAS's SORT procedure but it's fine to use it only when you are working with datasets with small number of observations and key variables. In cases where you need to sort large datasets with a lot of key variables, it is definitely better to use indexes of hash-objects. Let's see the comparison of the different methods one more time:

	PROC SORT	Simple INDEXES			Composite INDEXES DATA STEP	Hash Objects
		PROC DATASETS	PROC SQL	DATA STEP		
Actual time	52:20	5:45	6:53	54:71	23:52	3:92
CPU time	8:97	8:17	7:95	12:96	8:14	3:16

Table 2. Comparison of time usage between PROC SORT, INDEXES and Hash-Objects

As you can see from the table 2, the quickest way of sorting large datasets is a hash-object.

REFERENCES

Michael A. Raithel. “The Basics of Using SAS Indexes”. Proceedings of the 30th Annual SAS Users Group International Conference. Rockville, MD

Gregg P. Snell. “Think FAST! Use Memory Tables (Hashing) for Faster Merging”. Proceedings of the 31th Annual SAS Users Group International Conference. Shawnee, KS

SAS OnlineDoc 9.2. Available at http://support.sas.com/documentation/92

ACKNOWLEDGMENTS

I would like to thank Donnelle M LaDouceur, Peter Lord, Chad Melson and Irina Kotenko for the review of the paper and all support during preparing this paper.

CONTACT INFORMATION

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

Name: Daniil Shliakhov
Enterprise: Experis Clinical
Address: 43/2 Gagarin av.
City, State ZIP: Ukraine, Kharkiv, 61001
Work Phone: +38 057 755 7020 ext. 2402
Fax: +38 057 728 30 35
E-mail: daniil.shlyakhov@intego-group.com