PharmaSUG 2014 - Paper CC16

Need for Speed in Large Datasets – The Trio of SAS® INDICES, PROC SQL and WHERE CLAUSE is the Answer

Kunal Agnihotri, PPD LLC, Morrisville, NC

ABSTRACT

Programming on/with large datasets can often become a time consuming ordeal. One way to handle this type of situation is by using the powerful SAS® INDEXES in conjunction with the WHERE clause in the PROC SQL step. This paper highlights how effective indexes can be created using SQL (more flexible when compared to indexes created using DATA step index option or the DATASETS procedure) and further subset data using the WHERE clause which drastically reduces dataset processing and run time. This combination of techniques gives better results when accessing big datasets than using the said techniques singularly. This paper also throws light on the dual functionality of the SQL created indexes, namely, the ability to create indexes on both new and existing SAS data sets.

INTRODUCTION

A SAS *index* is file which has a direct link to the values of a particular variable or a set of variables in a data set. It is a snapshot of a part of the larger scheme of things. It is a copy of the original data but to a greater degree a copy of a specific part of the data set. The 'specificity' of the copy is what is so useful in accessing large data sets. - this is the **WHAT**. The created index resides in the same library as the data set on which it was created. SAS stores index entries in separate index files.

I'm sure that whoever has had to deal with programming on large/massive/humongous data sets has at some point or the other wondered and/or wished of a way to expedite the painstakingly slow run times of the programs. Well, please stop wondering and join the SAS indexes bandwagon! - this is the **WHY**.

An ideal and frequent candidate for using SAS indexes would be on large lab or ECG data or other such heavy-knit, long and wide data sets. - this is the **WHO**. Please be advised that SAS indexes do not need to be created for marginally big data sets. It is useful only when the data set at hand is genuinely big. It is of course at the discretion of the user whether to create a SAS index. Just because an index is created, that does not mean SAS will use it everytime the user invokes it. SAS judges if the use of the said index is a viable option or not and chooses the right to not put it into practice - which is a good thing. SAS does this to avoid using unnecessary/additional CPU resources. Remember that the index actually uses some memory and resources when it is created. So it should be created and used ONLY when the situation demands. The larger the data set, the larger is the index created. A point to be noted here is that the word 'large' is a relative term. Some studies may consider a 100,000 KB data set to be a 'large dataset'. That size is not big enough to justify the use of indexes. The user should bear in mind the index created itself will take up considerable space. This might hamper performance due to disk space/resources constraints. An ideal 'large' data set should be around or over the 2GB threshold to make indexes work without compromising system performance.

VARIETY IS THE SPICE OF INDEXES

A SAS index can be created either on one variable - simple index or on multiple variables - composite index. Each is explained with examples below. The variables used to create indexes are called index keys. Indexes can indeed be created using the INDEX DATA set option or the DATASETS procedure. The DATA set option only creates an index for a new data set. The DATASETS procedure on the other hand creates an index only for an existing data set. Both these functionalities are available with PROC SQL. Other than accessing the specific part of the large data set, SAS index also returns the observations in sorted order. This eliminates the need to sort the data set in subsequent data processing. This paper uses a LAB analysis data set as an example to create and use SAS indexes. The size of the data set is 2.11 GB and has over a million records in it.

```
proc sql;
     create index paramcd
     on adlb
     ;
quit;
```

Figure 1: Create a simple INDEX using PROC SQL on an existing data set

```
199
200   proc sql;
201       create index paramcd
202       on adlb
203       ;
NOTE: Simple index paramcd has been defined.
204   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           47.20 seconds
      cpu time            4.07 seconds
```

Figure 1 demonstrates how to create a simple INDEX on an already existing SAS data set. The DATA step INDEX option does not have this functionality built in it. SAS picks the name of the index - it is the same name as the key variable on which is it created. As shown in the log for the above program, a simple index was defined on the variable PARAMCD.

Let's move onto creating a simple index on a new data set.

```
proc sql;
     create table adlb (index = (paramcd)) as
     select *
     from adb.adlb
     ;
quit;
```

Figure 2: Create a simple INDEX using PROC SQL on a new data set

Figure 2 shows the functionality of the SQL procedure to create an index in real time - i.e; when the data set is being created in the same step as the index. The DATA step index option is used here. The DATASETS procedure does not have this functionality built in it.

```
33          proc sql;
34              create table adlb (index = (paramcd)) as
35              select *
36              from adb.adlb
37              ;
NOTE: Simple index paramcd has been defined.
NOTE: Table WORK.ADLB created, with 462281 rows and 93 columns.

38          quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           47.11 seconds
      cpu time            17.12 seconds
```

Don't be misled by the times shown in the log. The real and CPU time shown is the cumulative of both the data set and the index creation.

```
proc sql;
     create index sub_vis_par
     on adlb (usubjid,avisitn,paramcd)
     ;
quit;
```

Figure 3: Create a composite INDEX using PROC SQL

Figure 3 shows how a composite index can be created on an existing data set. The user is responsible in naming the index. USUBJID, AVISITN and PARAMCD are the key variables used to create the SUB_VIS_PAR composite index.

```
223    proc sql;
224        create index sub_vis_par
225        on adlb(usubjid,avisitn,paramcd)
226        ;
NOTE: Composite index sub_vis_par has been defined.
227    quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           15.65 seconds
      cpu time            6.78 seconds
```

As seen in the NOTE in log for the program, a composite index has been defined on the ADLB data set.

SPEED THRILLS

One of the biggest time consuming parts of working with a large data set is the sorting of those data sets. Indexes eliminate the need to sort data, provided the index created is based on the variables on which the sort was intended to be. The user should be mindful of the fact that creating indexes also eats up system time and resources. Hence it is imperative that the user creates sensible indexes which the program can/needs to use often. Creating an index

which will only be used once is not a programming performance booster. The user will need to assess the creation of a particular index before actually creating it.

The following examples show the comparison of efficiency in using indexes versus not using them.

```
proc sql;
     create table creat as
     select * from adlb
     where paramcd in ("CREAT")
     ;
quit;
```

Figure 4: Processing analysis lab data using an INDEX

In example 4, a new data set is being created using ADLB as source to subset on creatinine parameter test code "CREAT". The source is the same data set used in example 3. The log has a message in the form of INFO which clearly states that an index was used to optimize the WHERE clause. Please note the time taken to create the subset. The INFO in the log can be activated by using the option MSGLEVEL = I.

```
232   proc sql;
233        create table creat as
234        select * from adlb
235        where paramcd in ("CREAT")
236        ;
INFO: Index PARAMCD selected for WHERE clause optimization.
NOTE: Table WORK.CREAT created, with 25785 rows and 93 columns.

237   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.89 seconds
      cpu time            0.59 seconds
```

Now, a PROC step is used to create the same subset of data. This time, the source will be a data set which does not have an index defined to it but will be identical otherwise to the previous source data set in terms of data. The same variables are used in the BY statement which are used as key variables while the index was being created above. The where clause is identical in both cases. However, the time taken to create the subset is considerably more. While it does take *some* time to create an index, the time it took to create the subset with the good'ol PROC SORT is far more than the combined time it took to create and index and create the subset.

```
proc sort data = lab out = param;
    by usubjid avisitn paramcd;
    where paramcd = "CREAT";
run;
```

Figure 5: Processing analysis lab data using PROC SORT

```
241    proc sort data = lab out = param;
242        by usubjid avisitn paramcd;
243        where paramcd = "CREAT";
244    run;

NOTE: There were 25785 observations read from the data set WORK.LAB.
      WHERE paramcd='CREAT';
NOTE: SAS sort was used.
NOTE: The data set WORK.PARAM has 25785 observations and 93 variables
NOTE: PROCEDURE SORT used (Total process time):
      real time           22.62 seconds
      cpu time            4.93 seconds
```

A whopping difference of 21.73 seconds (deduct about 4 seconds for index creation) real time seconds is saved with the use of the index. Although the time taken varies and depends on a case by case basis, the user should bear in mind the considerable saving he/she may have in the entire course of the program.
USUBJID, AVISITN and PARAMCD variables were used in the above example as these variables are used in most of the processing of the LAB data set during its development/validation process. The user can create multiple simple or composite indexes, but should justify the creation by using them more than once in the program at hand.
Hence the user will not need to separate the output data set created while using the index. The resulting data set will be sorted on the key variables used to define the index though there is no ORDER BY statement specified in the PROC SQL step above.

Some users may wonder if using a TAGSORT would have competitive times if used in the above scenario. The following example shows the difference.

```
proc sort data = lab tagsort out = param;
    by usubjid avisitn paramcd;
    where paramcd = "CREAT";
run;
```

Figure 6: Processing analysis lab data using the TAGSORT option in PROC SORT

```
257   proc sort data = lab tagsort out = param;
258       by usubjid avisitn paramcd;
259       where paramcd = "CREAT";
260   run;

NOTE: SAS sort was used.
NOTE: Tagsort reads each observation of the input data set twice.
NOTE: The data set WORK.PARAM has 25785 observations and 93 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           1:32.65
      cpu time            13.46 seconds
```

The situation seems to have taken a turn for the worse. Except for the addition of the TAGSORT option, the remaining conditions remain identical. The TAGSORT did not seem to add any value to the sort. Instead it added time to the process, which is exactly the opposite to what is intended here. The TAGSORT option may reduce the temporary disk space used during processing but is not of great help in sorting times in this case. Depending on the size of the data and the BY variables used, the time TAGSORT takes may be less than a regular sort, but definitely can not beat the processing time set by the INDEXES.

The following table re-iterates the exercise in the paper.

Method	Real Time (seconds)	CPU time (seconds)
PROC SORT	22.62	4.93
TAGSORT option	1:32.65*	13.36
INDEX	0.89	0.59

* Minutes.

PASS THROUGH COMING THROUGH

Another use of the indexes can be leveraged while accessing the Oracle database using the PROC SQL pass through facility. Study level data stored in the Oracle database can be massive. To retrieve and process this data can be time and system resource consuming. Once the data is retrieved, it would need further processing depending on what needs to be done with the data. It is then when indexes can be put into practice and be useful in cutting down valuable time and system resources.

CONCLUSION

SAS indexes offer impressive time savings when used on large data set processing. They eliminate the need to sort the data repeatedly thereby increasing program efficiency. Care should be taken to not create indexes on trivial variables in the data set as this can slow down processing. The programmers are encouraged to analyze the use of a index creation as they can benefit from faster processing. Using an index becomes beneficial if the related subsets of data will be used repetitively in the course of the program.

REFERENCES

Micheal A. Raithel, The Complete Guide to SAS Indexes, SAS Press.

Kenneth W. Borowiak, Effectively Using the Indices in an Oracle® Database with SAS®
http://www.nesug.org/proceedings/nesug04/po/po12.pdf

ACKNOWLEDGEMENT

The author would like to thank his mentor Ken Borowiak for his encouragement and Latonya Murphy, Thomas Souers and Hunter Everton for their insightful comments on this paper.

DISCLAIMER

The content of this paper are the works of the author and do not necessarily represent the opinions, recommendations, or practices of PPD, LLC.

CONTACT INFORMATION

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

Kunal Agnihotri
3900 Paramount Parkway
Morrisville NC 27560

kunal.agnihotri@ppdi.com
agnihotrikunal@gmail.com