PharmaSUG 2014 - Paper BB01

Indexing: A powerful technique for improving efficiency

Arun Raj Vidhyadharan, inVentiv Health, Somerset, NJ
Sunil Mohan Jairath, inVentiv Health, Somerset, NJ

ABSTRACT

The primary goal of any programmer is “to get the desired output”. Once we have a plan to achieve this primary goal, there should ideally be a secondary goal, which is “to get the desired output, efficiently”. Most programmers are successful in achieving the primary goal while at least some don’t pay much attention or don’t feel the necessity of the secondary goal. This paper focuses on a technique called indexing in SAS that could drastically improve the performance of SAS programs that access small subsets of observations from large SAS data sets. This paper covers creation of simple and composite indexes, determining ideal candidates for index key variables, understanding when to use index and how to generate index usage messages.

INTRODUCTION

To understand how index helps in accessing a portion of huge dataset efficiently, let us consider a word document with 2000 pages and no table of contents. In order to go to a particular topic in this word document, you will have to scroll through each and every page, starting from page 1, until you get to the page you are looking for because you simply don’t know which page has it. Now consider the same word file with a table of contents and links to their corresponding pages. You can directly select the topic of your preference from the table of contents and click on the page link which will directly take you to the page you are looking for. This drastically reduces your time in getting to the desired page as compared to browsing through every page.

A SAS index functions just as the above example. However, various factors need to be considered in order to have the index work efficiently as you would expect. A badly implemented index is no better than reading the entire data set sequentially. We would look in to these factors in detail in this paper.

THE INDEX FILE

The **index file** is a SAS file that has the same name as its associated data file, and that has a member type of INDEX. There is only one index file per data file; that is, all indexes for a data file are stored in a single file. The index file might be a separate file, or be part of the data file, depending on the operating environment. An index file is stored as a separate file in Windows environment and is a part of the data file in UNIX environment. When stored as separate file in Windows environment, an index file has the extension “.sas7bndx”. In any case, the index file is stored in the same SAS library as its data file. The index file consists of entries that are organized hierarchically and connected by pointers, all of which are maintained by SAS. The lowest level in the index file hierarchy consists of entries that represent each distinct value for an indexed variable, in ascending value order. Each entry contains this information:

- a distinct value
- one or more unique record identifiers (referred to as a RID) that identifies each observation containing the value. (Think of the RID as an internal observation number.)

That is, in an index file, each value is followed by one or more RIDs, which identify the observations in the data file that contains the value. (Multiple RIDs result from multiple occurrences of the same value.) For example, the following represents index file entries for the variable LASTNAME:

Value	RID
Avery	10
Brown	6, 22, 43
Craig	5, 50
Dunn	1

Figure 1.

When an index is used to process a request, such as a WHERE expression, SAS performs a binary search on the index file and positions the index to the first entry that contains a qualified value. SAS then uses the value's RID to read the observation that contains the value. If a value has more than one RID (such as in the value for Brown in the previous example), SAS reads the observation that is pointed to by the next RID in the list. The result is that SAS can quickly locate the observations that are associated with a value or range of values.

UPDATING AN INDEXED DATA FILE

Each time that values in an indexed data file are added, modified, or deleted, SAS automatically updates the index. The following activities affect an index as indicated:

Maintenance Tasks and Index Results	
Task	**Result**
delete a data set	index file is deleted
rename a data set	index file is renamed
rename key variable	simple index is renamed
delete key variable	simple index is deleted
add observation	index entries are added
delete observations	index entries are deleted and space is recovered for reuse
update observations	index entries are deleted and new ones are inserted

Table 1.

WHEN TO INDEX?

The basic goal of having a SAS index is to be able to efficiently extract a small subset of observations from a large SAS data set. In doing so, the amount of computer resources (CPU time, I/O's, Elapsed time, etc.) expended should be less than if SAS read the entire data set sequentially. Therefore, if you will be extracting small subsets from a large SAS dataset, it is appropriate to create an index to help you. Here are the two main considerations for when to create an index.

1. The Size of the Subset

For an index to be effective, it should extract a small subset from a large SAS data set. It is easier to define "small subset" than it is to define "large SAS data set". A "small subset" is from 1% to 15% of the total number of observations found in a SAS data set. Table 2 provides the rules of thumb for the amount of observations that you may efficiently extract from a SAS data set using an index.

Subset Size	Indexing Action
1 % - 15%	An index will definitely improve program performance
16% - 20%	An index will probably improve program performance
21% - 33%	An index might improve or it might worsen program performance
34% - 100%	An index will not improve program performance

Table 2. Index subset guidelines

2. Frequency of Use

The more often an index is used, the more cost effective it becomes in terms of the computer resources necessary for its creation and its upkeep. Many programmers forget that it takes computer resources to initially build an index. The CPU time and I/O's expended during the creation of an index are pure overhead, since no data was processed; no report was produced. It also takes CPU time and I/O's to keep an index updated each time its attendant SAS dataset is updated. Consequently, it does not make sense to build an index and use it only one or two times. Infrequent use of an index would not recoup the computer resources that were expended in creating it. Therefore, if you predict that you would access a SAS data set very often via an index, then that data set is a good index candidate.

TYPES OF INDEXES

When you create an index, you designate which variable or variables to index. An indexed variable is called a key variable. You can create two types of indexes:

- A simple index, which consists of the values of one variable.
- A composite index, which consists of the values of more than one variable, with the values concatenated to form a single value.

In addition to deciding whether you want a simple index or a composite index, you can also limit an index (and its data file) to unique values and exclude from the index missing values.

KEY VARIABLE CANDIDATES

In most cases, multiple variables are used to query a data file. However, it probably would be a mistake to index all variables in a data file, as certain variables are better candidates than others:

- The variables to be indexed should be variables that are used in queries. That is, your application should require selecting small subsets from a large file, and the most common selection variables should be considered as candidate key variables.
- A variable is a good candidate for indexing when the variable can be used to precisely identify the observations that satisfy a WHERE expression. That is, the variable should be **discriminating**, which means that the index should select the fewest possible observations. For example, variables such as AGE, FRSTNAME, and GENDER are not discriminating because it is very possible for a large representation of the data to have the same age, first name, and gender. However, a variable such as LASTNAME is a good choice because it is less likely that many employees share the same last name.

CREATING INDEXES

There are three SAS tools that you can use to create indexes: PROC DATASETS, PROC SQL, and the DATA statement. Each one will accomplish the same end result, so your use of any particular one will be more a matter of preference. This section describes all three methods of creating SAS indexes.

PROC DATASETS

The DATASETS procedure may be used to generate an index on a SAS data set that already exists. The INDEX CREATE statement is used in PROC DATASETS to specify that an index is to be created. Here is the general form of the DATASETS procedure statements needed to build an index:

```
PROC DATASETS LIBRARY=libref;
MODIFY SAS-data-set;
INDEX CREATE varlist / UNIQUE NOMISS
UPDATECENTILES = ALWAYS | NEVER | integer;
```

In the DATASETS procedure, *libref* and *SAS-data-set* are the SAS data library and SAS dataset that is to be modified, respectively. In the INDEX CREATE statement, *varlist* is the list of SAS variables that will become index key values. (See the examples below for the difference between the *varlist* for a Simple index and the *varlist* for a Composite index). The UNIQUE option specifies that key variable values must be unique within the SAS data set.

The NOMISS option specifies that no index entries are to be built for observations with missing key variable values. The UPDATECENTILES option allows you to override when SAS updates the index's centiles. The UNIQUE, NOMISS, and UPDATECENTILES options are optional and do not need to be specified unless you have a particular need for them.

SIMPLE INDEX USING PROC DATASETS

Here is an example of using the DATASETS procedure to create a Simple Index:

```
proc datasets library=cdsales;
modify bighits;
index create cdnumber / unique;
run;
```

In the example, above, the *bighits* SAS dataset in the *cdsales* SAS data library is having a Simple index created for the *cdnumber* variable. Values of *cdnumber* must be unique for the index to be built. And, they must be unique when attempts are made to add additional observations to the *bighits* SAS dataset.

COMPOSITE INDEX USING PROC DATASETS

Here is an example of using the DATASETS procedure to create a Composite Index:

```
proc datasets library=cdsales;
modify bighits;
index create numname=(cdnumber artistname) / nomiss;
run;
```

In the example, above, the *bighits* SAS dataset in the *cdsales* SAS data library is having a Composite index created for the *cdnumber* and *artistname* variables. The name of the Composite index is *numname*. Observations which have values of *cdnumber* or *artistname* missing will not be added to the index.

PROC SQL

The SQL procedure can also be used to add indexes for existing SAS data sets. This can be done by using the CREATE INDEX statement. Here is the general format:

CREATE <UNIQUE> INDEX *index-name* ON *data-set-name(varlist)*;

As with its use in the DATASETS procedure, the UNIQUE option specifies that the values of the index variables must be unique within the SAS data set. *Index-name* is the name of the single index variable for Simple indexes and a programmer-chosen name for Composite indexes. *Data-set-name* is the name of the SAS data set that the index will be created for. If a Composite index is being created, then *varlist* contains the list of variables. Note that neither the NOMISS nor the UPDATECENTILES options are available for indexes created by the SQL procedure.

SIMPLE INDEX USING PROC SQL

Here is an example of creating a Simple index with the SQL procedure:

```
proc sql;
create unique index cdnumber on cdsales.bighits;
quit;
```

In the example, above, a Simple index is created for the *cdsales.bighits* SAS data set, based on the value of the *cdnumber* variable. Values of *cdnumber* must be unique for the index to be created.

COMPOSITE INDEX USING PROC SQL

Here is an example of creating a Composite index with the SQL procedure:

```
proc sql;
create index numname on cdsales.bighits(cdnumber artistname);
quit;
```

In the example, above, a Composite index named *numname* is created for the *cdsales.bighits* SAS data set. The *numname* index is based on the values of the *cdnumber* and *artistname* variables. Since the UNIQUE option was not used, there may be duplicate values of the *cdnumber/artistname* paired variables.

DATA STEP

You can build an index for a new data set by using the INDEX= data set option in the DATA statement. Here is the general format:

DATA *data-set-name*(INDEX=(*varlist* / <UNIQUE><NOMISS>

<UPDATECENTILES= ALWAYS | NEVER | *integer*>));

In the form above, *data-set-name* is the name of the new SAS data set. *Varlist* is the name of the key variable—if this is a Simple index—or a list of variables if this is a Composite index. See the previous sections for the meaning of the UNIQUE, NOMISS, and UPDATECENTILES options.

SIMPLE INDEX USING THE DATA STEP

This is an example of how you can build a Simple index with a DATA step:

```
data cdsales.bighits(index=(cdnumber / unique));
set olddata.oldhits;
... more SAS Statements...
run;
```

In the example, a Simple index is being constructed for the *cdsales.bighits* data set as it is created during execution of the DATA step. The index variable is *cdnumber*. Values of *cdnumber* must be unique for the index to be created.

COMPOSITE INDEX THE DATA STEP

Here is an example of how a DATA step is used to create a Composite index:

```
data cdsales.bighits(index=(numname=(cdnumber artistname) / nomiss));
set olddata.oldhits;
... more SAS Statements...
run;
```

In this example, a Composite index is created for the *cdsales.bighits* data set. The Composite index is named *numname*, and it is built from the values of the *cdnumber* and *artistname* variables. The NOMISS option specifies that observations with missing values for either *cdnumber* or *artistname* are not to have index entries created for them. Also, note that since the UNIQUE option was not used, there can be duplicate values of *cdnumber /artistname* in the *cdsales.bighits* data set.

EXPLOITING INDEXES

Once you have created indexes on SAS data sets, you will want to exploit them. There are four places that you may specify that an index be used to help reduce the processing overhead of your SAS programs. The following sections provide an overview of each.

USING AN INDEX IN A WHERE STATEMENT

The WHERE statement can be used in DATA and PROC steps to exploit a SAS index. The WHERE statement has the following form:

WHERE *where-expression;*

Here is an example of a WHERE statement that exploits a Simple index built from the variable *cdnumber*:

```
data cdsales;
set olddata.oldhits;
where cdnumber eq 123456”;
… more SAS Statements…
run;
```

This example uses a WHERE statement to exploit the *numname* Composite index built from the *cdnumber* and *artistname* variables in the *olddata.oldhits* SAS data set:

```
data cdsales;
set olddata.oldhits;
where cdnumber eq "123456" and artistname eq "Led Zeppelin";
… more SAS Statements…
run;
```

USING AN INDEX IN A BY STATEMENT

You can use the BY statement to return observations that are sorted into ascending order of the index variables' values. Here is the format of the BY statement:

BY <DESCENDING> *varlist* <NOTSORTED>;

In a BY statement, the DESCENDING option specifies that options be returned in descending value order of the *varlist* variable(s). The NOTSORTED option specifies that observations with the same *varlist* BY values are grouped together in the data set. If either the DESCENDING or the NOTSORTED options are specified SAS will not exploit an index to optimize the BY statement. If they are not specified, SAS will use an index, if any one of the following conditions is true:

- SAS will use a Simple index when:
  - *varlist* is the key variable in a Simple index
  - *varlist* contains two or more variables and the first variable in *varlist* is the key variable in a Simple index
- SAS will use a Composite index when:
  - *varlist* is a single variable that is the first key variable in a Composite index
  - *varlist* is made up of two or more variables that match the first two or more key variables in a Composite index

Here is an example of a BY statement that exploits the *numname* Composite index that is made up of the *cdnumber* and *artistname* variables:

```
data cdsales;
set olddata.oldhits;
by cdnumber artistname;
…more SAS statements…
run;
```

In the example, above, observations will be returned in ascending order of the values of *cdnumber/artistname*.

USING AN INDEX IN A KEY OPTION ON A SET STATEMENT

You can use the KEY option on a SET statement to exploit indexes associated with the SAS data set that is being read. The KEY option allows you to retrieve only selected observations from a data set. Here is the format of the SET statement with the KEY option:

SET *data-set-name* KEY=*index-name*;

This is an example of exploiting the KEY= option:

```
data cdsales;
set trans.hits(keep=cdnumber trnartist trnsales);
set olddata.oldhits key=cdnumber;
…more SAS statements…
run;
```

In the example, variables *cdnumber*, *trnartist*, and *trnsales* are read from the *trans.hits* SAS dataset. When each observation in *trans.hits* is read, the value of *cdnumber* is used in an index search of the *olddata.oldhits* SAS dataset, where *cdnumber* is the index key variable in a Simple index. If the index search is successful, then an observation is returned from the *cdnumber* SAS dataset into the new *cdsales* SAS data set.

USING AN INDEX IN A KEY OPTION ON A MODIFY STATEMENT

You use the KEY option in a MODIFY statement to direct SAS to use an index to access the observation in a SAS data set that is to be modified. Here is an example of how this could be coded:

```
data current.cdsales;
set update.sales(keep=cdnumber trnsales);
modify current.cdsales key=cdnumber;
sales = trnsales;
…more SAS statements…
run;
```

In the example above, the values of index key variable *cdnumber*, that resides in the *update.sales* SAS dataset are used to directly access observations in the *current.sales* SAS dataset. The KEY option specifies that SAS use the *cdnumber* Simple index to obtain observations from *current.cdsales* that are to be modified. Successful index searches result in the value of sales in the *current.cdsales* SAS data set being set to the value of *trnsales* that was input from the *update.sales* dataset.

A REAL TIME EXAMPLE:

The following example uses a lab dataset with 139775 observations and tries to extract a subset of 421 observations. An index named *labrec* is created for demonstration. As you can clearly see, the data step without indexing takes 0.22 real time seconds while the dataset with indexing takes only 0.05 real time seconds, an improvement over 77.3 percentage.

```
data adlb;
set ads.adlb;
run;

NOTE: There were 139775 observations read from the data set ADS.ADLB.
NOTE: The data set WORK.ADLB has 139775 observations and 92 variables.
NOTE: DATA statement used (Total process time):
      real time           1.45 seconds
      cpu time            1.38 seconds

/*Without index*/

data test1;
set adlb;
where pramcatn=2 and paramn=1 and avisitn=15;
run;

NOTE: There were 421 observations read from the data set WORK.ADLB.
      WHERE (pramcatn=2) and (paramn=1) and (avisitn=15);
NOTE: The data set WORK.TEST1 has 421 observations and 92 variables.
NOTE: DATA statement used (Total process time):
      real time           0.22 seconds
      cpu time            0.19 seconds

/*With index*/

proc sql;
create index labrec on adlb(pramcatn, paramn, avisitn);
```

```
NOTE: Composite index labrec has been defined.
quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.32 seconds
      cpu time            0.31 seconds

data test2;
set adlb;
where pramcatn=2 and paramn=1 and avisitn=15;
INFO: Index labrec selected for WHERE clause optimization.
run;

NOTE: There were 421 observations read from the data set WORK.ADLB.
      WHERE (pramcatn=2) and (paramn=1) and (avisitn=15);
NOTE: The data set WORK.TEST2 has 421 observations and 92 variables.
NOTE: DATA statement used (Total process time):
      real time           0.05 seconds
      cpu time            0.05 seconds
```

CONCLUSION

SAS Indexes can be used to drastically reduce the computer resources needed to extract a small subset of observations from a large SAS data set. But, before creating an index, you must decide if one is appropriate in accordance to the criteria presented above. After deciding that an index is appropriate, you have three tools to choose from to create one: the DATASETS procedure, the SQL procedure, and the DATA step. You can exploit indexes with the WHERE statement, the BY statement, or the KEY statement used in either a SET or MODIFY statement. In doing so, you will be increasing the efficiency of your SAS programs that use the index.

REFERENCES

- Michael A. Raithel: Creating and Exploiting SAS® Indexes
- Alex Vinokurov & Lawrence Helbers: Using SAS Indexes with Large Databases
- SAS Online Documentation, Version 9.2

CONTACT INFORMATION

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

Name: Arun Raj Vidhyadharan
Enterprise: inVentiv Health Clinical
Address: 500 Atrium Drive
City, State ZIP: Somerset, NJ 08873
Work Phone: 732.652.3490
E-mail: arunraj.vidhyadharan@inventivhealth.com

Name: Sunil Mohan Jairath
Enterprise: inVentiv Health Clinical
Address: 500 Atrium Drive
City, State ZIP: Somerset, NJ 08873
Work Phone: 732.652.3482
E-mail: sunil.jairath@inventivhealth.com