Paper 123-29

Creating and Exploiting SAS® Indexes

Michael A. Raithel, Westat, Rockville, MD

Abstract

SAS indexes can drastically improve the performance of programs that access small subsets of observations from large SAS data sets. Yet, many SAS programmers never use them. In this workshop, you will learn how to create simple and composite SAS indexes, determine which variables make good index key variables, discover when creating and using indexes is appropriate, learn about centiles, and find out how to generate index usage messages. You will see—first hand—how indexes can improve processing when subsetting and/or updating a large SAS data set. To be successful in this workshop, you should have base SAS knowledge, including creating and updating permanent SAS data sets. You will leave this fun-filled 75-minute workshop with a practical knowledge of how to create and exploit SAS indexes.

Introduction

It is not too hard to understand how a SAS index can help you to directly access the observations that you need in a particular SAS data set. As an exercise, do the following: Open SAS Online Documentation, Click on the Search tab, enter the word “rtrace” into the Search tab’s window, and click on <Search>. The SAS Online Documentation search function will return about a dozen links. When you click on any of those links, you go directly to a page in the documentation that discusses the SAS RTRACE facility. This saves you the tedious effort of going through the entire SAS Online documentation, page-by-page, looking for occurrences of the word “rtrace”.

A SAS index is analogous to the search function, above. A good index will allow your programs to quickly access the subset of SAS observations that you need from a large SAS data set. This will dramatically improve the speed and efficiency of your SAS programs. Conversely, a badly conceived SAS index will return far too many observations and be no better than reading the entire data set sequentially. (In keeping with the analogy, above, consider how many pages would be returned and how much longer it would take if you searched the SAS Online Documentation for the word “SAS”). That is why it is important to know more about the selection criteria for index variables, as well as the actual creation and use of SAS indexes.

This paper is an abbreviated version of the Hands On Workshop that will be presented at SUGI 29. The following sections present a balanced overview of SAS index creation and usage. To learn more about SAS indexes, refer to the References section at the end of this paper.

To Index or Not to Index?

Perhaps the biggest question associated with SAS indexes is: When is it appropriate to create one? 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 data set, 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 1** 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 1. Index subset guidelines.

People normally consider a SAS data set to be “large” when it contains tens of thousands, hundreds of thousands, or millions of observations. Since SAS normally reads SAS data sets sequentially, an appreciable amount of computer resources are consumed as a large SAS data set is read. Using an index causes SAS to read only a small portion of a larger data set. So, the larger the SAS data set—and the smaller the subset—the more likely that the data set will be a good candidate for indexing.

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 data set 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

You can build an index from a single SAS variable or from multiple SAS variables. An index created from a single SAS variable is called a *Simple index*. An index created from two or more variables is called a *Composite index*. If you will be subsetting a SAS data set via the value of a single variable, then you will want to consider building a Simple index based on that variable. For instance, if you always extract a subset of observations based on *cdnumber*, then *cdnumber* would be a likely candidate for a Simple index. On the other hand, if you are always using several variables to subset your data, then they might be good candidates for the creation of a composite index. For example, if you always subset on the values of *cdnumber* and *artistname*, then building a composite index on the values of *cdnumber* and *artistname* would be in order.

When Indexes are Used

SAS does not automatically use an index to access data in a SAS data set just because you have created one. There are four constructs that you must specify in a SAS program to get SAS to use an existing index. They are listed below, and further explained in the section of this paper titled *Exploiting Indexes*.

- A WHERE statement in a DATA or PROC step
- A BY statement in a DATA or PROC step
- The KEY option on a SET statement
- The KEY option on a MODIFY statement

SAS will not necessarily us an existing index even when you do use a WHERE or BY statement. SAS first calculates if using an index will be more efficient than reading the entire data set sequentially. The internal algorithms take a lot of factors into consideration, including data set size, the index or indexes that are available, and *centile* information. (For more information on "centiles", see the section of this paper titled *Centiles*). If SAS predicts that it will be more efficient to use a specific index to return observations than to read the entire data set, then it will use that index. If not, then it will read the entire data set, sequentially, to return the observations. However, SAS will not even consider using an index if you do not use a WHERE or BY statement (and, of course, if you do not use the KEY option).

Most of the time, SAS makes good decisions regarding whether or not to use an index. But, its internal calculations are not infallible, and sometimes the resources consumed when reading a large subset of data via an index are greater than reading the entire SAS data set. You can use the IDXNAME= and IDXWHERE= options to overeride SAS' default index usage. A discussion of these options is outside the scope of this paper. Refer to the references at the end of this paper for more information on these two options.

SAS will automatically use an index when you specify the KEY option on either a SET statement or a MODIFY statement. So, you do not have to be concerned as to whether or not an existing index is used.

You can have SAS index usage information posted to your SAS Log by specifying MSGLEVEL=I in an OPTIONS statement. Here are examples of messages posted to the SAS Log when an index is used with a BY statement and with a WHERE statement, respectively:

```
INFO:  Index cdnumber selected for BY clause processing.

INFO:  Index cdnumber selected for WHERE clause optimization.
```

If you fail to have MSGLEVEL=I specified in an OPTIONS statement, you will not have any feedback that the indexes you have created are being used. So, you should always make sure that this option is enabled when working with indexes.

Index Variable Selection

Since SAS indexes consume computer resources when being built and maintained, and since they take up disk space, you will want to be selective about which variables you select as index key variables. The major selection criteria are listed below. The ideal Index key variable will satisfy all three criteria.

- **Which variable or variables will you most often use to subset the SAS data set?** If you will most likely always use a particular variable to subset a large SAS data set, then that variable is a good candidate for becoming an index key variable. If it is a single variable, then it is a candidate for becoming a Simple index. For instance, if you are always subsetting a large SAS data set by *cdnumber* to obtain a small subset of observations, then you would make a Simple index from *cdnumber*. Alternately, if you are always subsetting a large SAS data set by two variables, say *cdnumber* and *artistname* then you would create a Composite index from those two variables.

- **Is your proposed index key variable discriminant?** A variable is *discriminant* if its values are very specific to a small set of observations within a SAS data set. In practical terms, this means that choosing particular values of a discriminant variable will yield small subsets from within the large SAS data set. Variables such as Gender, Age, Ethnicity, First Name, and Country are not normally discriminant. Variables such as Last Name, Zip Code, Social Security Number, Patient ID, and Part Number are usually discriminant. The more discriminant a variable is, the better it will be at returning a small subset of observations from a large SAS data set if it becomes an index key variable.

  You can determine how discriminant a variable is before using it to build an index by running the FREQ procedure. Run PROC FREQ against your index candidate variables. Once you have the frequencies, check them against **Table 1**, above. Variables with frequencies that fall in the 1 – 15% range are good candidates for indexing. Those in the 16 – 20% range are not very good candidates, but still may yield some performance improvements if indexed and used. You should not consider any variables with frequency values beyond the 16 – 20% range.

- **Is the SAS data set sorted into ascending order by the proposed index variable?** An index created from a variable that was used to sort a SAS data set into ascending order is a more efficient index. This is so because observations with the same index key variable values will be stored on the same or on adjacent pages in the SAS data set. When the index is used, fewer SAS pages have to be read since observations with the same key variable values are concentrated together. This will result in fewer input/output operations. Therefore, variables used to sort the large SAS data set are prime candidates for index key variables.

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 data set 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. (There will be more information on this in the Centiles section, below). 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 data set 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 data set.

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 data set 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 neigher 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* and *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* /a*rtistname* 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*;

The *where-expression* may be any valid SAS language arithmetic or logical expression. However, only these eleven forms of the WHERE expression will allow SAS to use an existing Simple index:

1. Normal comparison operators such as: `where x > 500;`
2. Normal comparison operators with NOT: `where x ^> 500;`
3. The CONTAINS operator: `where state contains "East";`
4. A comparison operator with the colon modifier: `where state =: "East";`
5. The TRIM function: `where trim(state) = "North Dakota";`
6. Range conditions with upper and lower bounds or range conditions that use the BETWEEN-AND operator:

   ```
   where 42 < X < 112;

   where X between 42 and 112;
   ```

7. The pattern matching operators LIKE and NOT LIKE: `where state like "East %";`
8. The IS MISSING and IS NULL operators:

   ```
   where name is missing;
   ```

```
where iq is null;
```

9. The SUBSTR function when it is of the form:

   where substr(*argument*,*position*,<*n*>) = "*value*";

   In this substring form, the following must be true for an index to be used:

   a. The value of *position* must be 1.

   b. The value of <*n*> must be less than or equal to the length of *argument*.

   c. The value of <*n*> must be equal to the length of *value*

   Here is an example: `where substr(state,1,4) = "East";`

10. The IN operator

```
where region in("East", "West");
```

11. Any WHERE clause composed of two or more of the above ten forms connected via AND:

```
where (region in("East", "West")) and (name is missing);
```

To utilize a Composite index in a WHERE expression, SAS uses "compound optimization". Compound optimization takes place when several variables of a composite index are used in a WHERE expression and are joined together with logical operators such as AND and OR. One of the following conditions must be true for compound optimization to occur:

1. At least the first two key variables in the composite index must be used in the WHERE condition:

```
where state eq 15 and county eq 30 and population lt 20000;
```

2. The conditions are connected using the AND logical operator:

```
where state eq 15 and county eq 30;
```

3. Any conditions using the OR logical operator must specify the same variable:

```
where state eq 20 and (county eq 30 or county eq 40);
```

4. At least one condition must be the EQ or IN operator:

```
where state eq 10 and county in(1,3,5);
```

For the examples, above, consider that there is a composite index named STATCNTY that is made from the STATE and COUNTY variables. SAS recognizes that the conditions are right for using compound optimization and utilizes the REGDIST composite index to optimize WHERE expression processing.

Here is an example of a WHERE statement that exploits the 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 are 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 data set. When each observation in *trans.hits* is read, the value of *cdnumber* is used in an index search of the *olddata.oldhits* SAS data set, 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 data set 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 *cdnumbe* , that resides in the *update.sales* SAS data set are used to directly access observations in the *current.sales* SAS data set. 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* data set.

Centiles

The word "centile" is a contraction of the words "cumulative percentiles". Centiles are a built-in feature of SAS indexes and are essentially twenty-one separate values saved in the index descriptor. These values represent the 0, 5,10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, and 100 percentile values of key variables in an index. The first centile, 0, contains the minimum index key value; the last centile, 100, has the maximum index key value stored in it. The other centiles are such that 5% of the data has index values less than the 5th centile, 20% of the data has index values less than the 20th centile, etc.

SAS uses centiles to determine if it is more efficient to use an index to read a SAS data set, or whether it is more efficient to just read the entire SAS data set. SAS initializes each index's centiles when the index is created and keeps them updated when a certain percentage of the index key variable's values have changed within the data set. The default value for updating centiles is 5%. This means that when 5% of the variables in the SAS data set that are index key variables have their values changed, SAS will recompute the centiles for the index.

You can examine a SAS index's centiles to determine just how balanced the index is. By executing the CONTENTS procedure with the CENTILES option, you can discover if the index key variable values are really spread out throughout the entire SAS data set. If so, then the index will yield efficient processing of the data set. If not, then you might want to reexamine why that particular variable—or variables—were used to create an index. Here is an example of the CONTENTS procedure with the CENTILES option:

```
Proc contents data=trans.hits centiles;
Run;
```

Appendix A contains an example of centiles listed in the *Alphabetic List of Indexes and Attributes* section of a CONTENTS procedure output. In the example, you can see that it lists a Simple index built from the variable STATE. The NOMISS option was specified when the index was created, the centiles will be updated after 5% of the values of STATE have been changed in the SAS data set, and there are 16 unique values of STATE in the SAS data set. The minimum value of STATE is "Baja California Norte", and the maximum value of STATE is "Washington". Most of the values of STATE are discriminant, because most represent 5% or less of the values within the observations in the SAS data set. However, you can see that one particular state, "Illinois", occupies five percentiles. This means that an index search on STATE for the value "Illinois" would return 25% of the observations in the SAS data set. That is not an acceptable observation percentage and could result in SAS consuming more computer resources through using the index than it would by doing a sequential read of the entire data set. This is an insight that you can only get through reviewing the centiles.

There is a lot more to the topic of centiles than can be covered in this paper. For instance, you may reset the default centile refresh rate from 5% to any percentage that you like. Additionally, you can refresh an index's centiles by executing the DATASETS procedure. For more information on centiles, refer to the references at the end of this paper.

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.

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

References

Raithel, Michael A. 2003. *Tuning SAS® Applications in the OS/390 and z/OS Environments, Second Edition.* Cary, NC: SAS Institute Inc.

SAS OnlineDoc®, Version 8, Copyright 2000, SAS Institute, Inc.

Contact Information

Please feel free to contact me if you have any questions or comments about this paper. You may reach me at:

Michael A. Raithel
Westat
1650 Research Boulevard
Room RW4521
Rockville, Maryland 20850

michaelraithel@westat.com

Appendix A. Centiles from a CONTENTS Procedure Listing

-----Alphabetic List of Indexes and Attributes-----

#	Index	Nomiss Option	Update Centiles	Current Update Percent	# of Unique Values	Variables
1	STATE	YES	5	0	16	
	---					Baja California Norte
	---					Baja California Norte
	---					British Columbia
	---					California
	---					Campeche
	---					Colorado
	---					Florida
	---					Illinois
	---					Illinois
	---					Illinois
	---					Illinois
	---					Illinois
	---					Michoacan
	---					New York
	---					North Carolina
	---					Nuevo Leon
	---					Ontario
	---					Quebec
	---					Saskatchewan
	---					Texas
	---					Washington