Paper 124-25

Power Indexing: A Guide to Using Indexes Effectively in Nashville Releases

Diane Olson, SAS Institute Inc., Cary, NC

ABSTRACT

In Version 6 of the SAS®System software, there were performance problems with some index features. This paper addresses how those problems have been resolved in the Nashville releases (Version 7 and later releases). It also supplies guidelines to help you decide when indexes will benefit your application and increase the performance of your SAS programs.

Questions answered in this paper include:

- ❑ What are indexes?
- ❑ How do I create an index?
- ❑ What are the benefits of an index and when does the SAS System use indexes to access data?
- ❑ What additional resources do indexes use?
- ❑ What are the index improvements in Nashville releases?
- ➢ Centiles
- ➢ More User Control
- ➢ APPEND Procedure
- ➢ Expanded WHERE Optimization
- ➢ MSGLEVEL=I
- ❑ When will an index help me the most?
- ❑ How can I tune my use of indexes?

INTRODUCTION

Indexes can boost your application's performance time under certain circumstances. However, inappropriate use of indexes can actually degrade performance. Armed with the information below, you can decide if your application can take advantage of the SAS index facility in Nashville releases. The new features and performance enhancements of both indexes and the APPEND procedure, as well as potential pitfalls, are discussed.

WHAT IS AN INDEX?

Much like a book's index allows you to locate a particular subject quickly, a data file's index allows observations with specific values to be accessed quickly. An index is an inverted tree structure that stores values of key variables in ascending order. A *key variable* is a variable designated when the index is created. The index includes information as to the location of the key variable's values within observations in the data file. So, instead of reading through a data file to find a particular value of a variable or variables, an index identifies the exact location of those observations.

Modifying, deleting, or adding observations to a data file automatically updates the index or indexes associated with that data file. An index is a performance-tuning tool, and conserves some resources at the expense of others.

CREATING AN INDEX

An index can be created on an existing data file or when creating a data file. Either compressed or uncompressed data files may be indexed. You can create one index or multiple indexes on a single data file.

There are several ways to create an index. You can use the DATA step, the DATASETS, SQL, or IML procedures, the SAS Explorer, Screen Control Language (SCL), or the SAS/Warehouse Administrator™ software. Here are three examples of creating the same index on a data set. The first example, using the DATA step, creates the index as the data file is being created:

```
data frogs (index=(toxicity));
   input species $ color $ location $
      habitat $ toxicity $ endangered $;
   datalines;
bullfrog green East boggy medium yes
river olive South swamps high no
sheep brown Texas pasture none no
…
;
```

The following DATASETS procedure creates the index on an existing data file:

```
proc datasets nolist;
   modify frogs;
      index create toxicity;
run;
```

Finally, the following SQL procedure creates the index on an existing data file:

```
proc sql;
   create index toxicity on frogs(toxicity);
```

For more information on the DATA step, see the *SAS Language Reference: Dictionary*. For more information on PROC DATASETS and PROC SQL, see the *SAS Procedures Guide*.

A *simple index* is defined on one variable's values. The examples above create a simple index. You can also create one index for two or more variables; this is called a *composite index*. While the name of the simple index is that of the variable, you must specify a unique name for the composite index. The name cannot be the same as any of the data file's variables or indexes, and must comply with SAS naming conventions. Here are SAS code snippets to show how composite indexes may be created:

From the DATA step:

```
data frogs (index= (lethal_locales=
   (toxicity location)));
```

From PROC DATASETS:

```
index create lethal_locales = (toxicity
   location);
```

From PROC SQL:

```
create index lethal_locales on
   frogs(toxicity,location);
```

Note that the SQL procedure requires a comma to separate the key variables.

BENEFITS FROM INDEXES

Now you know how to create indexes on your data files, but how can the SAS System use them to benefit you? The SAS System may use an index when processing the following:

- WHERE expression
- KEY= in the SET and MODIFY DATA step statements
- SCL table lookup
- SQL join queries
- BY processing

Those applications that use any of these statements to process small subsets of data from large data files may reduce the subset extraction time using indexes. While the BY statement does not subset data, BY statement processing can use indexes for its sorted order.

THE WHERE EXPRESSION

A WHERE expression restricts processing on a data file to a subset of the observations. Using an index and a WHERE expression together is called "optimizing the WHERE expression". See the conditions below for WHERE conditions that can be optimized:

Table 1.1 WHERE Conditions That Can Be Optimized

Condition	Examples
comparison operators, which include the EQ operator; directional comparisons like less than or greater than; and the IN operator	where empnum eq 3374; where empnum < 2000; where state in ('NC','TX')';
comparison operators with NOT	where empnum ^= 3374; where x not in (5, 10);
comparison operators with the colon modifier	where lastname gt: 'Sm';
CONTAINS operator	where lastname contains 'Sm';
fully-bounded range conditions specifying both an upper and loser limit, which includes the BETWEEN-AND operator	where 1<x<10; where empnum between 500 and 1000;
pattern-matching operators LIKE and NOT LIKE	where firstname like '%Rob_%';
IS NULL or IS MISSING operator	Where name is null; where idnum is missing;
TRIM function	where trim(state) = 'Texas';
WHERE SUBSTR(*variable*, *position*, *length*)='*string*'; when the following conditions are met: *position* is equal to 1, *length* is less than or equal to the length of *variable,* and *length* is equal to the length of *string*	where substr(name,1,3)='Mac' and (city='Charleston' or city='Atlanta');

When processing the WHERE expression, the SAS System decides whether to use the index or to read the data file sequentially. First, the SAS System identifies the available indexes for use with the WHERE expression, if that expression can be optimized. A composite index may be used for WHERE expression optimization only when the first key variable is a variable in the WHERE expression.

For example, with the following WHERE expression, the simple index LOCATION could be used for optimization:

```
where= (location in ('Antarctica', 'Alaska',
   'Siberia'))
```

However, the composite index of variables TOXICITY and LOCATION could not be used because LOCATION is not the first key variable in the composite index.
After the available indexes are identified, the SAS System estimates the number of observations qualified by each of those indexes. The index that selects the smallest subset of observations is chosen, and resources required to use the index are compared against the resources required to process the data file sequentially. Factors in computing whether to use the index include:

- size of the subset of observations identified by the index relative to the data file size
- data file value order (that is, sorted in ascending index value order or not)
- data file page size
- number of allocated buffers
- cost of uncompressing data file for a sequential read

Continuing to use the FROGS data file with the simple indexes LOCATION and TOXICITY and the composite index LETHAL_LOCALES with the WHERE expression

```
where=( toxicity= 'high' and
   location='South')
```

both TOXICITY and LETHAL_LOCALES qualify for use with the WHERE expression. LETHAL_LOCALES is chosen for resource usage comparison with sequential access, as it identifies a smaller subset of observations that can optimize the WHERE expression.

As a general rule, the SAS System uses an index if it estimates the WHERE expression will select one-third or fewer of the data file's observations. However, if the number of qualified observations is less than 3% of the data file's observations, the index is automatically used; no resource usage comparison is done.

Compound optimization of the WHERE expression is achieved by taking advantage of all the variables in the index by carefully constructing your WHERE expression. Often only the first variable of a composite index is used to optimize a WHERE expression. Even if the composite index is defined with more key variables than are used in the WHERE expression, the composite index can still be used for optimization as long as the first key variable of the composite index is in the WHERE expression. In our example, if we have a composite index using TOXICITY and SPECIES, the index would still qualify for use in optimizing our WHERE expression even though the variable LOCATION is not a key variable in that index.

THE KEY= INDEX OPTION

The MODIFY and SET statements provide the KEY=index option, which allows you to specify an index to access particular observations based on the indexed values.

For the MODIFY statement, you use the KEY=option to name an index defined on the data set that is being modified. You can then specify a lookup value from a secondary data source. That data source (typically a SAS data set named in a SET statement or an external file read by an INPUT statement) provides a like-named variable, which is then used as a key to search the master data file to locate the observation. Once the observation is located, you can modify it as needed.

For example, if you had an additional data file, ENDANGERED, which contained the variables SPECIES and

NEW_ENDANGERED, we would index the FROGS data file by the variable SPECIES. To update the FROGS variable ENDANGERED, we could simply use the following code:

```
data frogs;
   set endangered;
   modify frogs key=species;
   if _iorc_=0 then do;
      endangered = new_endangered;
      replace;
   end;
   else _error_=0;
run;
```

The SPECIES index is used to locate an observation in the FROGS data file having the same value as an observation read from the ENDANGERED data file. When the same SPECIES value is found, the value of ENDANGERED is replaced with the value of NEW_ENDANGERED. The automatic variable _IORC_ contains the return code for each I/O operation that the MODIFY statement attempts to perform.

With the SET statement, the KEY= option also provides non-sequential access to the data file's observations based on the key variable(s). This access supports the concept of table lookup from an additional data source.

For example, if we want to produce a data file containing our FROGS information along with POPULATION from a different data file:

```
data book_of_frogs;
   set frogs;
   set pop_frogs key=species;
run;
```

The DATA step reads the primary data file, FROGS, and a lookup data file, POP_FROGS. It uses the index SPECIES to read POP_FROGS non-sequentially, by looking for a match between the SPECIES value in each data file. The result is to create BOOK_OF_FROGS with the variables from FROGS, plus the variable POPULATION when the values of SPECIES for the two data files are equal.

For more information, see the SET and MERGE statements in *SAS Language Reference: Dictionary*.

SETKEY IN SCL

SETKEY in SAS Component Language (SCL) defines an index key for retrieving rows from a SAS data file. It establishes a set of criteria for reading observations by comparing the value of the columns from the SDV (SCL Data Vector) to the key value in the rows.

For more information, see SETKEY in *SAS Component Language: Reference*.

OTHER USES OF INDEXES

If a data file is indexed, BY processing of the key variable(s) is allowed without having to sort the data. When you specify the BY statement, if the file is not sorted on that variable, the SAS System automatically looks for an index to use. If it exists, the observations are retrieved using the index. However, using the index instead of sorting the data file may not be more efficient. In general, the use of an index for BY processing is for convenience and not performance.

Indexes may also be used for some internal SQL optimization, but you cannot directly request it. SQL will automatically use the index in cases where it will speed performance.

LAW OF CONSERVATION

The Law of Conservation of Energy states that energy is neither lost nor gained, but simply changes from one form to another. The same is true with indexes; some resources are consumed at the expense of others. The use of indexes is a balancing act - deciding what additional resources you can give up in order to get the performance benefits. CPU usage, I/O, memory and disk space are all affected when using indexes.

Consider how sequential access and index (direct) access work to find an observation. With sequential access, the SAS System reads a page from disk into memory. All of the observations on that page are processed. This reading and processing continues until the end of the file. With an index, the SAS System determines the location of the next observation using the index, reads a new page *if necessary* finds the observation on the page and uses it. This continues for each value of the index that satisfies the subset criteria (a WHERE expression, for example).

With the index, the SAS System reads only the observations that match the subset criteria. The cost for an individual observation read for an index is higher, because the sequential access reads the page only once and then processes each observation. However, with the index, the SAS System does not read the observations that do not meet the subset criteria. If the index selects a large portion of the observations, the increased cost of using the index eats away at the savings from the decreased number of reads.

When describing the index read above, note that the new page is read only if necessary. If it is already in memory, a new page read is not done. Therefore, if the data file is sorted in ascending order by the key variable(s), performance of statements using the index will be better, as fewer reads will be required.

The creation of an index, as well as maintaining the index when the data file is modified, require additional CPU usage.

The number of I/O operations required to read a subset of data may also increase with an index; the more random the data, the more I/O operations required to read the subset via an index. Maintaining the ascending value order of key variables will result in fewer I/O operations. In a worst case scenario, if the ascending order of the value's observations were located on multiple pages on disk, an I/O operation would be necessary for each observation.

To create and use an index requires more memory and disk space than sequential access of the observations. The index is stored on disk, either as a separate file or as a part of the data file, depending on your operating environment.

INDEX IMPROVEMENTS

There were quite a few changes and enhancements made in the Nashville releases to improve performance and give you more control when using indexes.

CENTILES

Version 6 index optimization assumed a uniform distribution of data between the minimum and maximum key variable values, causing the subset estimation for WHERE expressions to be erroneous if your data were distributed differently. For example, if your data included social security number as a key variable, estimation would be correct, since the data would be uniformly distributed. On the other hand, if your key variable was salary information having a bell-shaped distribution, the subset

estimation would overestimate the subset of observations at the lowest and highest salaries, but underestimate the middle salary range subset.

In Nashville releases, accuracy of estimating the subset size has improved significantly with the use of additional data statistics called cumulative percentiles, or *centiles*. The information provided by centiles represents the distribution of values in an index. Previously, only the maximum and minimum values were kept in the index. In the Nashville releases, twenty-one centiles are kept: 0, 5, 10,…, 95, 100 percentiles, where 0 percentile is the minimum value of the data, 20% of the data is less than the value held in the 20 percentile, 50 percentile is the median value, and 100 percentile is the maximum value. By default, centile information is not updated after every data file change, although you can specify that the update be done. You can also specify that the centile data be updated when the file is closed or when a certain percent of the values for the key variable(s) have been changed. The default is 5%. You can also specify that the centile values never be updated. See the INDEX CENTILES statement and the UPDATECENTILES option in the PROC DATASETS documentation in the *SAS Procedures Guide*.

MORE USER CONTROL

In Version 6, indexes were used to optimize a WHERE expression when the SAS System determined it would be more efficient than sequential access. In some cases, it was wrong, but you could not change the behavior. In other cases, perhaps you would want the SAS System to use the index whether or not the index was determined to be the most efficient. The IDXWHERE= and IDXNAME= data set options have been added in the Nashville releases to allow you control over these situations.

IDXWHERE=YES tells the SAS System to decide which index is the best for optimizing a WHERE expression, disregarding the possibility that a sequential search might be more resource efficient. IDXWHERE=NO tells the SAS System to ignore all indexes and satisfy the conditions of a WHERE expression by sequentially searching the file.

The SAS System automatically chooses which index to use after determining the size of the data subset of each index. Previously there was no method to specify which index should be used. IDXNAME= gives you the option to tell the SAS System which index to use. This option provides another performance gain in that you can first allow the SAS System to determine which index is the most efficient, then specify that index name with IDXNAME=. From that point on, the SAS System no longer uses resources to decide whether to use an index. It also no longer needs to determine which index to use, freeing those resources. (See MSGLEVEL=I later in this paper for tips on determining which index the SAS System is using.)

THE APPEND PROCEDURE

In Version 6, the APPEND procedure updated the index of a data file as new observations were added, creating slow performance, most especially for large data files. As a stopgap measure, people resorted to removing the index, appending data, and then adding the index to the file once again. In Nashville releases, part of PROC APPEND was rewritten so that the index is not updated until all the new observations have been added. Then the key variable(s) are internally sorted, and the data is inserted into the index in ascending order. The data file itself is not sorted.

You will see the performance boost with no change to your SAS application code. The change is transparent, unless there are errors. The most common error will be violations of the UNIQUE option. If you are using the UNIQUE option on your index to assure unique values, and a non-unique value is detected, the APPEND procedure does not detect this error until after the data has been appended. In that case, the observation is deleted. However, you cannot know which of the duplicated value's observations will be deleted. If this is a concern, you may use the APPENDVER=V6 option, which tells the SAS System to use the Version 6 PROC APPEND method of updating the index; this also results in Version 6 performance.

Note that sorting your data before appending will reduce the index update overhead. (See MSGLEVEL=I later in this paper for more APPEND information.)

EXPANDED WHERE OPTIMIZATION

As noted earlier in Table 1.1, cases where WHERE expressions can be optimized have expanded in the Nashville releases. Please see that table for detailed information.

DETERMINING IF INDEXES WILL HELP YOU

There are no hard and fast rules for deciding if indexes will improve your application's performance. If you follow these guidelines, however, you should be able to make an informed decision:

- If your data file is small, sequential processing is usually just as fast or faster. If your page count (available from the CONTENTS procedure) is less than three pages, do not use an index; it will degrade your performance.
- Consider the cost of the index if the data file is frequently changed. An index is automatically updated when the data file is updated, requiring additional resources.
- If the subset of data for the index is not small, it may require more resources to access the data than sequential access. Conversely, when you intend to retrieve a small subset of observations from a large data file, an index will most likely be more efficient. The smaller the subset, the greater the performance gains.
- Consider your data access needs. An index must be used often in order to make up for the resources consumed when creating and maintaining it.
- Do not use more indexes than you actually need. Find the most discriminating variables in commonly used queries and use them as your key variables.

TUNING INDEX USE

Once you have determined that using an index is beneficial to your application, you can tune your index usage. Be sure to select key variables according to which variables you use in queries. Remember that as your subset gets smaller, your index performance gains get larger.

The following table is to be used as a rule of thumb, and not as an absolute. Performance gains are very data dependent.

Table 1.2 Estimating Performance from Subset Size

Percentage of Observations	Performance Gains
1-10	excellent
11-25	good
26-50	marginal
51-100	poor

When creating an index to process a WHERE expression, do not create one index to try to satisfy all queries. If there are several

variables that appear in the queries, the queries may get better performance with simple indexes on the most discriminating of the variables. For better index performance, sort your data file into ascending order on the key variable before you index the data file. If appending data to the indexed file, sort the data you are appending before executing the APPEND procedure. The more sorted your key variable data is, the better your index performance. Note that you cannot sort an indexed data file without losing the index; optimally, you should sort the data before creating an index.

Consider replacing an IF statement that subsets, but does not use indexes, with a WHERE expression. Be careful with the change, as the two statements are processed differently.

Consider using WHERE expressions in the FSEDIT procedure instead of the SEARCH and FIND expressions, which do not use indexes.

THE MSGLEVEL=I OPTION

The MSGLEVEL= system option can also help tune your index use, as well as giving you information about your APPEND process. Simply set the option value to I, instead of the default N. If an index is being used, the name of the index will be printed to the log. If an index is not used, but one exists that could optimize the WHERE expression, a message will be printed to the log suggesting what to change in order to use the index. For example, the log message may suggest that you sort your data or increase the number of your memory buffers:

INFO: Index TOXICITY not used. Sorting into index order may help.
INFO: Index LOCATION not used. Increasing bufno to 2 may help.

MSGLEVEL=I also records information from the APPEND procedure. If the fast append process is used, you will get a note to the SAS log stating so. If, however, the SAS System could not use the fast append code, that fact will be noted in the log, as well as the reason. Possible reasons include no member-level locking available, the existence of referential integrity constraints, the use of Cross Environment Data Access (CEDA), or a WHERE expression present on the BASE data file.

CONCLUSION

The indexing facility has changed to make it more efficient to use in the Nashville releases. Performance improvements were also included in these changes. There are many factors to consider when determining whether indexes can speed your data access. If you are querying large data files for small subsets of observations, indexes will increase your performance. Other situations require that you weigh the resources used to create and maintain the index against the increased access time. When used properly, an index is a performance-tuning tool that helps you get the best possible performance for your SAS applications.

REFERENCES

SAS Institute Inc. (1999), *SAS Language Reference: Concepts*, Cary, NC.

Beatrous, Steve and Armstrong, Karen (1991), "Effective Use of Indexes in the SAS® System", *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, 605-614.

ACKNOWLEDGMENTS

Thanks to very diligent reviewers: Deanna Warner, Jim Craig, Lisa Brown, Steve Beatrous, James Holman, Jane Stroupe and Meg Pounds.

CONTACT INFORMATION

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

Diane Olson
SAS Institute Inc.
SAS Campus Drive
Cary NC 27513
Work Phone: 919-677-8000 press 1-4924
Fax: 919-677-4444
Email: Diane.Olson@sas.com