From Multi-Minute to Sub-Second Response Time With Indexes

by Tim Berryhill, Pacific Gas & Electric Company

At Pacific Gas & Electric Co., many datasets include more than 1 million observations, and our largest disk databases are thousands of tracks—they may exceed 1 gigabyte. We are very interested in ways to reduce processing time. Many of our applications involve selecting a few records to display from a large dataset. The old way to do this was to create a temporary dataset which was a subset of the large dataset. The new way to do it is to use indexes.

Once we pay the up-front cost to create an index, we can select a few records in a fraction of a second, compared to more than a minute the old way. These dramatic performance gains show up in elapsed time (the waiting time after your client hits the enter key and before the screen fills with new data), in CPU time (the seconds you pay dollars for) and in I/O (input/output operations). In the applications where I get improved performance with indexes, I am only selecting a few observations or even a single observation. This is the key to useful indexing in SAS.

WHY AN INDEX IS NEEDED

To understand why selecting only a few observations is important, you need to know a little about how data is stored and retrieved, first by the operating system and then by SAS. Any operating system stores data on disk in blocks. The blocks are arranged around the disk in tracks. Each time SAS asks for data, the operating system is going to give it the block it requested, and at that time all of the other blocks on the same track are available essentially for free. Once SAS has the block, it breaks it up into pages and then into observations, and when a PROC or a datastep asks for an observation, SAS copies one from the operating system buffer into the program data vector, or PDV. Depending on the size of an observation, a track may hold tens or hundreds of observations.

Using an index does not make it faster to read each track, or to copy each observation from the buffer into the PDV. What the index can do is locate the desired observation so that SAS jumps directly to it, instead of reading sequentially through the entire dataset. In our most extreme example, we keep a 4 million observation customer dataset in about 7000 tracks (350 megabytes) on a 3380 disk drive. When we display a single observation using the index, we read just the single track which has the observation—we do not have to read the other 6,999 tracks. Not reading the other tracks is where we save elapsed time, CPU time and I/O. If we have to read every track anyway, the index will not save us anything—it will slow us down. That is why indexing is not efficient if you are reading most of the observations in a dataset.

WAYS TO USE AN INDEX

I frequently use indexes two different ways. Each of these methods assumes the index already exists—it would not be worthwhile to create a new index to use either method once. The easiest way to use an index is in a WHERE clause or a WHERE statement. For instance, I can use a WHERE statement in PROC FSEDIT. If the variable in the WHERE is indexed, records can be selected much more quickly than without the index. This also works for a WHERE or WHERE ALSO entered on the FSEDIT command line.

The other way I often use indexes is with the KEY= option of the SET statement. This is more complicated. If I have a large file of customer data and a smaller file of daily transactions which I want to display, I can merge the two files and keep the matches or I can index the larger file and select individual records using the index. Here is an example using SET ... KEY=:

```
DATA REPORT;
  DO UNTIL (DONE);
    SET DAILY(KEEP=ACCOUNT transaction de-
        tails) END=DONE;
    SET MASTER(KEEP=ACCOUNT customer details)
        KEY=ACCOUNT;
    OUTPUT;
  END;
  STOP;
RUN;
```

For a small DAILY file and a large MASTER file, the SET ... KEY= can use 1% of the resources the merge would require. There are a lot of tricky details to SET ... KEY=, and I am not going to go into them today. If you are interested, get a copy of Ben Conner's paper "Keys To The Data: Using The Key= Option On The SET Statement", which opened my eyes to the problems of duplicate and missing keys in the MASTER file. You will also need to know about I/O return codes and the related macros to all the benefits of the KEY= option.

There is a third way to use an index which can be useful but is dangerous. If you need to use a dataset in two different sort orders and you do not have enough space for two copies, you can sort it one way and index it the other way. Then you can name the

index in a BY statement to read the entire file in indexed order rather than physical order. This requires enough disk space for the data and the index, rather than enough space for two copies of the data. The problem with this approach only appears when your observations are small compared to your block size, and the dataset fills more than a couple of tracks. Using the index SAS can read each track when an observation in it is next in the indexed order, but it only reads the one observation it needs right then even if each track contains hundreds of observations. Later it will read the same track again to get another observation, and again and again. For small observations, this can require more CPU time, more wall time and thousands more I/O operations than sorting the file. I discovered this with an application running on IBM's CMS operating system where a sort and merge took a few minutes and I cancelled the indexed merge after more than an hour.

CREATING AN INDEX

As I mentioned, there are up-front costs to using indexes. Creating an index will use roughly the resources required to sort the dataset, or to sort a dataset with only the variables you name in the index. If you append or delete observations or edit any of the key fields, you will have to re-create the index. There are at least three ways to create indexes. I use:

```
PROC DATASETS LIBRARY=MYDISK;
   MODIFY MYDATA;
      INDEX CREATE ACCOUNT;
   RUN;
QUIT;
```

This defines a simple index (a single variable index) on the variable ACCOUNT. You can also define compound indexes involving multiple variables. You cannot use two simple indexes in place of a compound index, nor a compound index in place of either simple index. For some applications, it might make sense to create two simple indexes on ACCOUNT and DATE for a dataset and also create the compound index on ACCOUNT and DATE.

Other programmers prefer to define indexes using PROC SQL, and still others use the INDEX= option of the DATA statement to define an index as they create and populate a dataset.

THREE CAVEATS ABOUT INDEXING

Before I wrap up, I will recap three important details about indexes.

First, using an index can slow you down. SAS reads an index from disk into memory to process it, and if using the index does not avoid any other processing, the index will slow you down. In fact, if you sort a file on one key and index it on another, using the index to read the entire file can be many times slower than resorting the file to read it. This happens when you have many small observations in a large dataset. Using the index to jump around in the dataset, you might read a particular track from disk into memory many times to read different observations.

Second, be aware that the index takes up space in a library. You may have enough disk space for a dataset, but not enough additional space for an index to that dataset. Once you have an index, if you modify the indexed dataset and need to re-index it, you may need to explicitly delete the old index first to free the space. This is similar to the problem of trying to make a new version of a dataset which fills more than half of the space available for the library. SAS does not delete the old version until the new version is complete, so you need enough space for two copies at once. The index even takes up space after the indexed dataset is deleted. I was surprised to realize that deleting a dataset did not delete the index.

Third, you can lead a horse to water, but you cannot make it drink. Once you provide an index, SAS has algorithms for deciding whether or not to use it in a particular step. It will put a note in the log if it uses an index, but it will not tell you why it did not use an index which was available.

FOUR RULES OF THUMB

Finally, I have a few rules of thumb for indexing. These are based on my experience using SAS 6.08 on MVS systems—you will have to experiment to find similar guidelines for your environment.

1) If a dataset has small observations and you use most of them most of the time, sort it.
2) If a dataset changes rarely and you often report subsets of 10% or less, create an index.
3) If a dataset has large observations and you often report less than a quarter of the observations, create an index.
4) If you want fast access to individual observations, create an index.