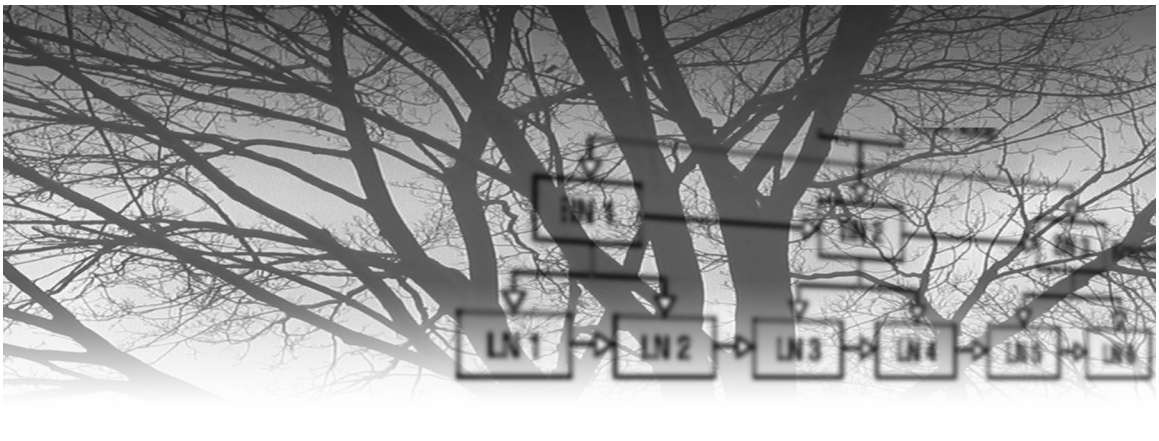


Chapter 1

Introduction to Indexes

The Index Concept

The concept of an index is hardly new to us. We use indexes in everyday life without giving them a second thought. For example, if I were to ask you to find every page in this book that contains the word "centiles," what would you do? You would not read through every page of this book, searching for the word "centiles." Instead, you would go directly to the index in the back of the book, search the index pages for the word "centiles," determine on which non-index pages it could be found from the index entry, and then go directly to those pages. Using the index would have saved you a lot of time and effort.

A similar example would be if I were to ask you to find the pages in this book that contain the name of the first president of the United States. You would go to the index, search through it, and find that no such index entry exists. You would tell me that there is no entry for the name of the first president of the United States, and you would not bother searching through all of the non-index pages of the book. Using the index would have saved you the time and effort of searching through every page in the entire book for an entry that does not exist.

Both examples illustrate how an index improves the efficiency of a search for data. If we find an entry in the index of a book, we can streamline our search effort and go directly to the pages that contain information about that entry. If we do not find an entry for a particular topic, we can conclude that it is not in the book and move on to looking for other entries, or to searching the indexes of other books. Thus, indexes save us time and effort when we are searching for information on a particular topic in a particular venue.

The Index as a SAS Performance Tool

A SAS index is functionally similar to an index in a book. It is used to look up whether a particular value of a key variable exists in the data pages of a SAS data set. If so, then only those pages are accessed; if not, then no data set pages are accessed. In this way, an index is a SAS data set performance tool, because it limits the amount of processing that is done to a given SAS data set. But, it is a performance tool that you must specifically build and overtly use.

When SAS reads a SAS data set without using an index, it reads the entire data set sequentially. SAS data sets are actually segmented (behind-the-scenes) into *pages* on

which observations are stored. SAS moves each data set page from disk to computer memory, starting with the first data set page and ending with the very last data set page. Once a page is in memory, SAS can read the observations stored on that particular page. This process happens with every SAS program you execute that does not use an index.

The movement of SAS data set pages between disk and computer memory is done via Input/Output (I/O) events. I/Os take time to execute and are the slowest events in the life of your SAS program. The more I/Os your SAS program consumes, the longer it takes for your program to run. Conversely, the fewer I/Os your SAS program consumes, the quicker it runs. So you can see that it is advantageous to limit the number of I/Os your SAS program uses, whenever possible.

The main goal of using a SAS index is to read only a small portion of a large SAS data set, instead of reading the entire SAS data set. As with the book index example, above, you want to use the SAS data set index to reduce the time and effort consumed reading observations with a specific value. With SAS, it is a specific index key variable value that you are looking for. When using an index, SAS first consumes I/Os by reading the index pages, searching for the specified value of the key variable. Then, if the value is found in the index, SAS consumes additional I/Os by directly reading *only* those pages that contain the specified value of the index key variable. If a large SAS data set is being accessed and only a few pages contain the specified key variable value, then you have saved many I/Os by having avoided reading the entire SAS data set.

Using a SAS index to access observations in a SAS data set with a specific key variable value can drastically reduce the I/Os and wall clock time of your SAS program. It can also lower CPU time, because less processing is necessary on the fewer pages that are returned to your SAS program. A decline in wall clock time can be good for SAS programmers in all environments. Cutting I/Os and CPU time can be especially beneficial for SAS programmers who work in organizations that have instituted computer resource chargeback programs. Such organizations often charge for CPU time and for I/Os. Using SAS indexes to decrease both of these resources helps you by lowering the amount that you are charged for running your SAS application programs.

Besides reducing computer processing resources, using a SAS index returns the observations in sorted order. They are sorted into ascending key variable(s) value order in your output SAS data set. This eliminates the need to execute subsequent SORT procedures and enhances BY statement processing.

Types of SAS Applications That May Benefit from Indexes

Just about any type of SAS application can benefit from the use of SAS indexes because of the decreased run time that they facilitate. SAS batch applications generally run faster when indexes are used within them to extract small subsets of observations from large SAS data sets. Using SAS indexes can be advantageous when you have a series of long-running batch applications that must be run sequentially. Shrinking a batch window—the time it takes for your SAS batch programs to run each day or night—would definitely be a visible benefit of using SAS indexes.

SAS/IntrNet applications that access small subsets of large SAS data sets certainly profit from the use of SAS indexes. Users of Web applications are sensitive to response time issues. They do not expect to have to wait very long after pressing ENTER to receive their results back in their Internet browsers. Using an index behind-the-scenes to subset a SAS data set that is being queried by a SAS/IntrNet program results in better response time for your users. This gives them greater confidence in the reliability of the SAS/IntrNet Web applications and greater productivity in their use of those applications.

SAS stored procedures used by groups of programmers and non-programmers via SAS Enterprise Guide benefit from the use of indexes. Like the SAS/IntrNet application users, Enterprise Guide users expect good response times from the stored procedures that have been written for them. When the stored procedures that they are invoking access small subsets of observations stored in large SAS data sets, users get their result sets far faster when SAS indexes are judiciously employed behind-the-scenes.

How SAS Indexes Are Structured

Indexes are separate SAS files with a member type of INDEX. Internally, they are divided into pages the same way that SAS data sets are. Indexes are stored in the same SAS data library that contains the data set they are associated with. SAS maintains the relationship between the index and its data set. When observations are added, updated or deleted from the data set, the index file is updated to reflect the changes. All indexes for a given SAS data set are stored in the same index file.

The logical organization of an index is based on the data storage structure known as a B-tree. This means that index entries are grouped into one of three node types: the root node, branch nodes, and leaf nodes. Each node contains a number of individual index entries and is stored on an index page. A particular index page may contain only entries of a single node type. The various nodes are logically connected through a series of node

pointers and through pointers within the entries. The function and structure of an entry varies according to node type.

The following sections explain how the entries in each node are organized.

Root Node

The root node is the highest level node in an index. All accesses of the index begin with the root node and then follow the pointers down to other nodes. There is one root node entry for each child (or subordinate) branch node. Each root node entry contains the highest key variable value stored in a child branch node and a pointer to the beginning of that branch node. The root node is stored on a single index page.

Root node entries contain only two fields: a value field, and a node identifier (NID) field. The value field is equal in length to the key variable (for a *simple* index), or key variables (for a *composite* index), of the indexed SAS data set. The value field contains the highest key variable value stored in the branch node the entry points to. The NID contains a pointer to the subordinate branch node.

Branch Nodes

Branch nodes are the intermediate level nodes in an index. Accesses of the index proceed from the root node to the branch nodes—via a binary search—and then follow pointers down to the leaf nodes. Each branch node is stored on an index page that is filled with only branch node entries. There is one branch node entry per leaf node. Branch node entries contain the highest key variable value stored in a subordinate branch node or leaf node and a pointer to the beginning of that subordinate branch node or leaf node.

The structure of branch node entries is identical to that of root node entries. The value field entry in a branch node contains the highest key variable value stored in the leaf node pointed to by the entry. The NID contains a pointer to the subordinate leaf node.

Leaf Nodes

Leaf nodes are the lowest level nodes in an index. An index search culminates when the entries in a leaf node are examined for the requested key variable value. If the key variable value is found, SAS follows leaf node pointers to specific observations in the SAS data set. Like branch nodes, leaf nodes are stored on index pages that are populated exclusively by leaf node entries. There is one leaf node entry per unique key variable value in the SAS data set that the index is associated with.

Leaf node entries contain a value field and one or more record identifier (RID) fields. The value field is equal in length to the index key variable (for a *simple* index), or to the combined length of the index key variables (for a *composite* index), of the indexed SAS data set. The value field contains a unique key variable value that can be found in one or more observations within the SAS data set. The RID contains a pointer to an observation in the SAS data set that has the value field value in it. SAS uses the RID to directly

access the SAS data set and return the observation with the requested key variable value. If key variable values are unique in a SAS data set and the UNIQUE option is specified, then there is only one pair of value field and RID per leaf node entry. See Chapter 5, "Index-Related Options," for a complete explanation of the UNIQUE option. If the key variable values are not unique, a value field can have any number of RIDs associated with it. Thus, the size of leaf node entries can vary in indexes where the key variable values are not unique.

When an index search finally arrives at a leaf node, the entries are examined in a binary search. The value fields in leaf node entries are compared against the key variable value the program is looking for. If SAS reaches the end of the leaf node binary search without finding the specific key variable value, the value does not exist in the SAS data set.

Figure 1.1 depicts the composition of root node, branch node, and leaf node entries. For any index, the size of the root and branch node entries is always the same. However, indexes with non-unique key variable values can have leaf node entries of varying sizes. Each entry contains one RID for every observation with a specific key value. For example, if three observations have the same key variable value, the leaf node entry will have three RIDs associated with the value field. Node identifiers are 4 bytes on a 32-bit host and 8 bytes on a 64-bit host. Record identifiers are 8 bytes on a 32-bit host and 12 bytes on a 64-bit host.

Figure 1.1 The Structure of Root, Branch, and Leaf Nodes

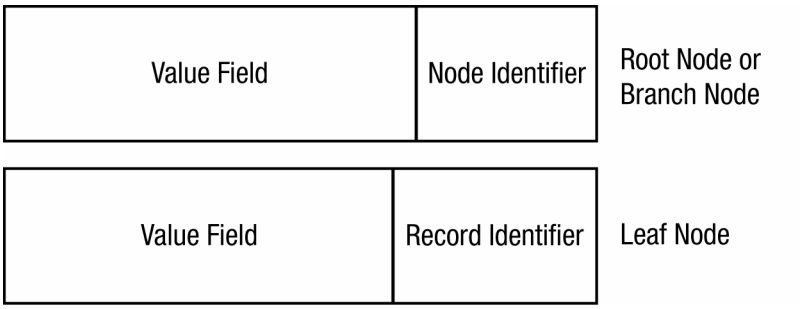
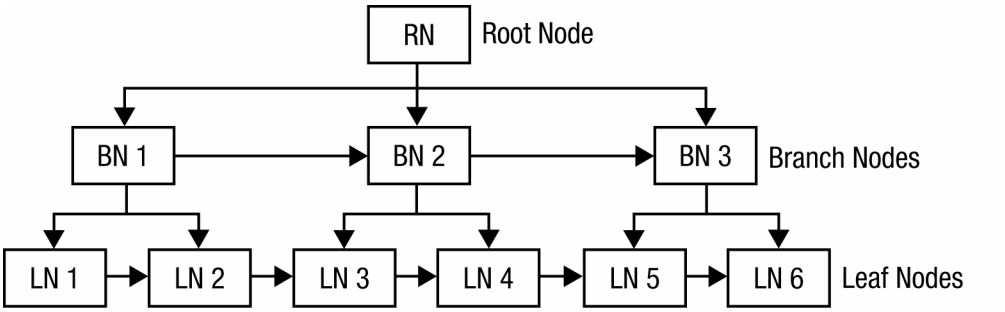


Figure 1.2 illustrates the tree structure of a SAS data set index. In the figure, the root node (RN) has pointers down to the branch nodes (BN). Each branch node has a pointer to the next branch node and pointers down to the leaf nodes (LN). Index searches begin with the root node and follow NIDs down to the lower levels of the index.

Figure 1.2 The Index Tree Structure

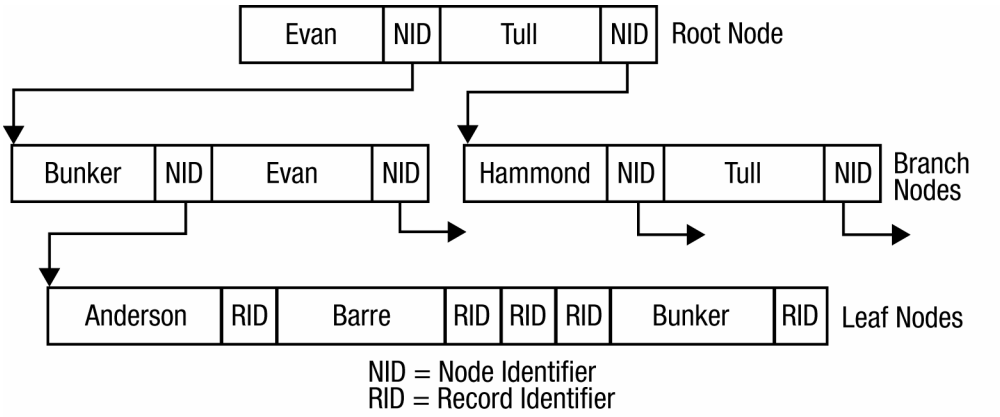


SAS keeps the structure of an index symmetric by balancing the index. It balances the index by keeping each leaf node exactly the same number of levels in distance from the root node. This means that accessing any particular leaf node consumes exactly the same amount of computer resources as accessing any other. If observations are added or deleted from the data set, index node entries are created or deleted at all appropriate levels of the index, depending on the key variable values. If a preponderance of new key variable values falls into a specific range, index nodes are added to expand the index "horizontally," to avoid adding new levels to the index. If a large number of observations are deleted, the index may contract "horizontally." This ensures that changes in the population of a SAS data set do not have a negative impact on the performance of its indexes. SAS performs index balancing tasks at the end of the DATA step in which the index was updated.

Large SAS indexes, especially those with small index page sizes, tend to have more index levels. The greater the number of levels an index has, the more I/Os are consumed during an index search and the longer it takes to complete the search. Conversely, indexes with fewer levels require fewer I/Os to traverse the index during an index search. So it is advantageous to increase the index page size to try to keep the number of levels that an index occupies as low as possible. This may be done with the IBUFSIZE option, discussed in Chapter 5, "Index-Related Options." Because SAS does not report the number of levels an index occupies, you must specify a large index page size value on the IBUFSIZE option and hope that it minimizes the number of index levels, thereby promoting good index performance.

Figure 1.3 presents an example of an index search. In this example, the program is using the index to return all observations with the key variable value of *Barre*.

Figure 1.3 Example of an Index Search



Here is the sequence of events that transpire during the index search:

1. The index search begins with a binary search of the entries in the root node. Each root node entry value field contains the highest key variable value stored in the branch node it points to. The first root node entry, *Evan*, is of a higher key variable value than *Barre*. If *Barre* does exist in the index, it is in one of the subordinate nodes pointed to by this root node entry. SAS follows the NID pointer down to the branch node.

2. SAS starts a binary search of the branch node. The first branch node entry, *Bunker*, is of a higher key variable value than *Barre*. So the index search continues by following the NID pointer from the branch node entry to the beginning of its associated leaf node.

3. When the index search arrives at the leaf node, another binary search is initiated. The first entry in the binary search, *Barre,* is a direct match to the key variable value being sought. There are three RIDs associated with the value field containing *Barre*. Thus, there are three observations in the SAS data set containing the key variable value of *Barre*. SAS follows each RID, one by one, to the SAS data set and returns each of the three observations to the program. When the last observation has been obtained, the SAS program is finished with the index search for *Barre*.

Types of SAS Indexes

SAS gives you the ability to construct two different types of indexes. The difference between the two index types is simply a matter of whether the index is built from a single variable or from multiple variables. Because there are different considerations to keep in mind when constructing either type, both are described separately.

Simple Indexes

A SAS index created from a single variable is known as a *simple index*. The variable that is used to create the index is known as the *index key variable*. You can create a simple index for any variable that exists in a SAS data set. Index key variables may be numeric or they may be character. When you create a simple index, SAS gives the index the same name as the index key variable. Consequently, you can find an index with the same name as the index key variable in the "Alphabetic List of Index and Attributes**"** section of a CONTENTS procedure listing for the indexed SAS data set.

Here is an example of a DATA step that creates a simple index:

```
data indexlib.prodindx(index=(state));
set  indexlib.prodsale;
run;
```

In the example, above, a new SAS data set named INDEXLIB.PRODINDX contains a simple index named STATE after the DATA step executes. The STATE simple index contains one entry for every value of the index key variable STATE found in the INDEXLIB.PRODINDX SAS data set, along with pointers (RIDs) to each observation that contains that value.

If you know that you are going to use a particular variable to obtain small subsets of a large SAS data set on a frequent basis, then you should consider creating a simple index from that variable. If there are other variables that are also often used to subset the SAS data set, then you can make simple indexes for them, too. A SAS data set may have multiple simple indexes associated with it. Chapter 3, "Index Variable Selection Considerations," provides a discussion on how you may determine which variables make good index variable candidates.

Composite Indexes

A SAS index created from two or more variables is known as a *composite index*. Composite index key variables may be numeric, character, or any combination of the two. You may choose to construct a composite index key from variables that occur in any order within an observation—composite index key variables do not need to be

adjacent fields. (SAS actually concatenates the variable values together in the value fields of the index entries that are created for the index.)

Because a composite index is created from two or more variables, SAS cannot pick a name for a composite index. You are responsible for providing a name. You may choose any valid SAS variable name for the name of a composite index. After a composite index is created, you can find the composite index name in the "Alphabetic List of Index and Attributes" section of a CONTENTS procedure listing for the indexed SAS data set. (To see other places that you may get index information, refer to Chapter 6, "Identifying Index Characteristics.")

This is an example of a DATA step that creates a composite index:

```
data indexlib.prodcomp(index=(country_state=(country state)));
set  indexlib.prodsale;
run;
```

In this example, the newly created SAS data set INDEXLIB.PRODCOMP contains a composite index named COUNTRY_STATE after execution of the DATA step. That composite index contains every distinct combination of the values of COUNTRY and STATE found in the INDEXLIB.PRODCOMP SAS data set and pointers to each observation containing that distinct value.

SAS often uses composite indexes to surface observations when only the first variable in a composite index is used in a WHERE expression or BY statement. You should keep this in mind when determining the order of variables to specify in a composite index.

SAS compares the WHERE or BY variables, one by one, from left to right, with the variables in an existing composite index. SAS stops when it reaches the end of the shortest list of matching variables. If one or more of the WHERE or BY variables match one or more of the variables in the composite index, then that composite index may be used.

For example, if you are creating a composite index based on variables COUNTRY and STATE, your first instinct might be to list COUNTRY first in the composite index so that it is COUNTRY/STATE. However, if many of your SAS programs subset the SAS data set with WHERE expressions based on STATE, you would consider creating a STATE/COUNTRY composite index. This increases the likelihood that the composite index will be used in the aforementioned types of queries and can save you the trouble of building a simple index based on STATE.

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 specific constructs that allow SAS to use an existing index:

- a WHERE expression in a DATA or PROC step (see Chapter 10, "Using Indexes with a WHERE Expression")
- a BY statement in a DATA or PROC step (see Chapter 11, "Using Indexes with a BY Statement")
- the KEY option on a MODIFY statement (see Chapter 12, "Using Indexes with the KEY Option on a MODIFY Statement")
- the KEY option on a SET statement (see Chapter 13, "Using Indexes with the KEY Option on a SET Statement")

SAS does *not necessarily* use an existing index even when you do use a WHERE expression or a BY statement. SAS first calculates if using an index would 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 Chapter 4, "Index Centiles.") Here is the three-step algorithm that SAS uses (Clifford 2005):

1. **Compute estimated number of observations qualified by the index.** SAS uses the index's centiles to estimate the total number of observations that would be qualified to be returned by the index. This estimate is accurate to within 5% as long as the centiles are up-to-date.

2. **Calculate the I/O cost per RID.** SAS examines the RIDs (record identifiers) on the first qualifying leaf node index page and calculates the number of different data pages that those RIDs point to. SAS computes an I/O cost per RID by dividing this number into the number of RIDs on an index page. This results in a decimal number that is less than or equal to one.

3. **Calculate the number of data pages that would be read by the index.** SAS multiplies the estimated number of qualified observations (#1 above) by the I/O cost per RID (#2 above) to get the number of SAS data set pages that would be read if the index was used. This number should be much smaller than the total number of pages in the entire SAS data set.

If SAS predicts that it would be more efficient to use a specific index to return observations than to read the entire data set, then it uses that index. If not, then it reads the entire data set sequentially to return the observations. However, SAS does not consider using an index if you do not use a WHERE expression or a BY statement.

SAS automatically uses an index when you specify the KEY option on either a MODIFY statement or a SET statement. It does so because the KEY option specifies exactly which index should be used. You do not have to be concerned with whether or not an existing index is used with the KEY option in a MODIFY or SET statement.

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 override SAS default index usage. Both of these options are discussed in Chapter 5, "Index-Related Options."

Estimating the Size of an Index

SAS stores index entries in a separate index file. These index entries take up space, so it is natural to ask just how much space a prospective index will occupy. SAS Technical Support has created a program that enables you to get a fair estimate of the size of your SAS index. You can find a copy of that program in Appendix D, "Estimating the Number of Pages for a SAS 9 Index." It is also included in the example code for this book, found on its companion Web site at support.sas.com/companionsites.

The index estimation program requires that you provide five values for the computation:

- **PSIZE** This refers to the page size of the index file. Set PSIZE equal to the current value of IBUFSIZE. See the section titled "The IBUFSIZE System Option" in Chapter 5, "Index-Related Options," for a thorough discussion of this index option.
- **VSIZE** This is the total length, in bytes of the variable that you intend to use to create a simple index. If you are going to create a composite index, add the lengths of all variables that will make up the composite key. You can find variable lengths in a CONTENTS procedure listing of the data set you are going to index.
- **UVAL** This parameter is the number of unique values that you expect in your SAS data set for the particular index key. If all values are unique for a simple index, then UVAL should be equal the total number of observations in the SAS data set. If not, or if you are going to create a composite index, you need to run the FREQ procedure to get an idea of the number of unique values. Because this program is computing an estimate, do not worry if you are in the position of estimating the number of unique values.
- **NREC** This value is the total number of observations in the SAS data set. If you are building an index for an existing SAS data set, find this value from a PROC CONTENTS listing. Otherwise, you can estimate this value from how many observations you expect to have in a SAS data set that you are creating.
- **Host** This identifies the operating system hosting the SAS data set and where the index is built. There are ten possible host values:

MVS	OS/390 and z/OS
WIN	Windows NT, 2000, and XP
LNX	RedHat Linux on Intel servers
ALP	OpenVMS Alpha
ALX	Compaq Digital UNIX
HP64	HP 64 UNIX
S64	Solaris 64 UNIX
R64	AIX 64
H6I	HP/UX for Itanium Platform Family, 64-bit
W64	Windows for IPF, 64-bit

Once you supply the five main values and execute the program, it computes the index size and creates a formatted report in the SAS log.

Here is an example of the output from the index size estimation program. In this example, the size of an index created from variable SEQNUM for INDEXLIB.PRODINDX was computed.

```
Index characteristics:
   Host Platform            = WIN
   Page Size (bytes)        = 32256
   Index Value Size (bytes) = 8
   Unique Values            = 2304000
   Total Number of Values   = 2304000
   Number of Index Levels   = 2

Estimated storage requirements for a V9 index:
   Number of Upper Level Pages =        1
   Number of Leaf Pages        =     1145
   Total Number of Index Pages =     1146 or     36,965,376 bytes

Note: the above estimate does not include storage for the index
      directory (usually one page) or the host header page.

Estimation of index size complete.
```

The program first reiterates the five values that were supplied in a section labeled "Index characteristics." Then it displays the number of "Upper Level Pages" (which are used to store the root node and branch nodes), the number of "Leaf Pages," and the "Total Number of Index Pages." In this example, you can see that one index page would be enough to contain the root node and the branch nodes. It would take 1,145 pages to store all of the leaf nodes for the SEQNUM simple index. The total number of index pages would be 1,146. SAS multiplies this by the page size (you entered this value in as PSIZE=) to get the total number of bytes, which is 36,965,376—or about 35 megabytes.

If you are going to create multiple indexes for a SAS data set, then you need to calculate each index separately. When you're done, add the number of pages for each index together to get the total index pages used by *all* indexes for the SAS data set. You can stop there, or multiply *total index pages* by the index page size to get the total number of bytes for the entire index file.

The index size estimate program is a great tool for getting a reasonable estimate of the amount of space needed for your SAS indexes. It is probably most useful for people in organizations where disk space is at a premium, or where people are charged for the disk space that their data sets occupy.

Summary

This chapter introduced the concept of SAS indexes, discussed how SAS indexes are actually performance tools, and described how indexes can benefit various types of SAS applications. Next, the structure of an index was described, including the root, branch, and leaf nodes and the entries that reside within them. An example was provided to illustrate how SAS traverses an index during an index search.

The chapter presented the two types of SAS indexes: a *simple* index made from a single variable and a *composite* index made from two or more variables. It discussed the four SAS programming structures that use SAS indexes: the WHERE expression, the BY statement, the KEY option in a MODIFY statement, and the KEY option in a DATA statement. Then, the three-step algorithm that SAS uses to determine whether or not to use an index for WHERE or BY statement processing was discussed. The chapter concluded by presenting how to estimate the size of an index.