Indexing in the SAS® System, Version 6

Denise J. Moorman and *Deanna Warner*

Denise J. Moorman is a technical support analyst at SAS Institute. Her area of expertise is base SAS® software. Denise has a BS degree in elementary education from Liberty University and a computer programming certificate from North Carolina State University. She has been a SAS software user for five years.

Deanna Warner is a senior technical writer at SAS Institute. She has a Bachelor of Liberal Studies in writing from St. Edward's University. Deanna has been with SAS Institute in Austin, Texas since 1982.

Abstract

The purpose of this article is to explain indexing in the SAS® System. Questions and answers are provided to help you understand what an index is and how one works, when and how to create an index, when SAS uses an index, how an index is maintained, and what you can do to enhance the performance of an index.

Content

The following topics are discussed:

Introduction

Indexing became available in Version 6 of the SAS System to improve performance of SAS applications. An index, like any tool, is more beneficial when understood. This first set of questions provides some general information about SAS indexes.

Question: What is an index?

Answer: An index is an optional file that you can create for a SAS data set to provide quicker and more efficient access to observations. The index stores values in ascending value order for a specific variable or variables and includes information as to the location of those values within observations in the data set. That is, an index provides the ability to locate an observation by value. When creating an index, you designate which variable or variables to index; an indexed variable is called a *key variable*.

Question: When does the SAS System use an index?

Answer: In general, SAS can use an index to improve performance in the following situations:

- For WHERE processing, an index can provide faster and more efficient access to a subset of data. Note that to process a WHERE expression, SAS decides whether to use the index or read the entire data set sequentially.

- For BY processing, an index returns observations in the index order even when the data set is not stored in that order.

- For the SET and MODIFY statements, the KEY= option allows you to specify an index in a DATA step to retrieve particular observations in a data set.

Question: How does the SAS System use an index?

Answer: SAS uses an index to directly access a value. For example, suppose you want the observation with SSN (social security number) equal to 465-33-8613:

- Without an index, SAS accesses observations sequentially in the order they are stored in the data set. SAS reads each observation one after another, looking for SSN=465-33-8613 until all observations are read.

- With an index for the variable SSN, SAS accesses the observation directly. SAS satisfies the condition using the index and goes straight to the observation containing the value without having to read each observation in the data set.

Question: What type of indexes are available?

Answer: There are two types of indexes: a simple index and a composite index. A data set can have multiple simple indexes, composite indexes, or both.

- A *simple index* is an index of values for one key variable. SAS automatically names the index the same name as the variable.

- A *composite index* is an index of two or more key variables. The values from the variables are concatenated to form a single value. When you create a composite index, you specify a name for the index.

In addition to deciding whether you want a simple index or a composite index, you can also:

- limit an index (and its data set) to unique values by specifying the UNIQUE option when you create an index. This prevents duplicate values from being stored in both the data set and the index.

- exclude missing values from the index. Missing values can be prohibited from being stored in the index by specifying the NOMISS option when you create an index. However, the missing values are permitted to exist in the data set.

Understanding the Index File

The index file is a SAS file, which has the same name as its associated data set and a member type of INDEX. There is only one index file per data set...all indexes for a data set are stored in a single file.

Question: Are indexes stored in the same physical file as the data set?

Answer: For the MVS operating system, the index file and data set reside in the same physical location. However, in other operating environments, the index file is a separate file. In any case, the index file is stored in the same SAS library as its data set.

Question: What is the internal structure of an index file?

Answer: The SAS System creates a binary tree (B-Tree) hierarchical structure consisting of entries that represent each distinct value for a key variable. The entries are in ascending value order. Each entry consists of a distinct value and one or more unique record identifiers (referred to as a RID) that identifies each observation containing the value. You can think of the RID as an internal observation number. For example, the following table represents index entries for the variable LASTNAME:

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

SAS automatically keeps the index file balanced as updates are made, resulting in a uniform resource cost to access any index entry. All space occupied by deleted values is recovered and reused.

Question: Are there any other internal structures of an index file besides that of a B-tree?

Answer: Yes. On special hardware, a bitmap index is used. Refer to *Scalable Performance Data Server™ User's Guide, Version 1, First Edition.*

Question: How is the index file read? Is the entire index file loaded to memory when opened?

Answer: When an index is used to process a request, such as a WHERE expression, SAS does a binary search on the index file and positions the index to the first entry that contains a qualified value. SAS then uses the value's RID or RIDs to read the observations containing the value. The entire index file is not loaded to memory; one index page is accessed at a time. The most recent index page is kept in memory.

Question: Extra disk space is required to store the index file. Is there a way to calculate the amount of disk space needed?

Answer: To calculate disk space required for an index file, you can use a formula provided in the 1989 SUGI Proceedings, "Using New SAS Database Features and Options" on page 343. This formula is appropriate for all operating environments and both simple and composite indexes. The formula can be accessed on the SAS Institute World Wide Web at the following URL:

www.sas.com/techsup/download/datastep/indexest.sas

Caution: The formula is untested and should be used cautiously.

Question: If a data set is encrypted using the ENCRYPT= data set option, is the index file also encrypted?

Answer: Yes, the index file is also encrypted. Prior to Release 6.08 TSLEVEL 430, the utility files used to sort the values when creating an index were not encrypted.

Deciding Whether to Create an Index

Even though an index can reduce the time required to locate a set of observations, especially for a large data set,

there are costs associated with creating, storing, and maintaining the index. When deciding whether to create an index, you must consider increased resource usage along with the performance improvement.
Question: Can you provide some guidelines for determining whether to index?

Answer: When considering whether to index a data set, use the following guidelines:

- Create an index when you intend to retrieve a small subset of observations, for example, less than 25% of all observations.

- For a small data set, sequential processing is often just as efficient as index processing. If the number of data set pages is less than three, then it is not beneficial to index any variable in the data set. To determine the number of data set pages, issue the CONTENTS procedure. Note that the available information from PROC CONTENTS depends on the operating environment.

- Consider how often your applications will use an index. An index must be used often in order to make up for the resources used from creating and maintaining it.

- Limit the number of indexes for a data set to reduce disk storage and to reduce update costs.

- Do not create an index for a data set that is frequently changed. If you have a data set that changes often, the overhead associated with updating the index after each change can outweigh the processing advantages you gain from using the index.

Question: Using the previous guidelines, my data set is a good candidate for indexing, but how do I decide which variable or variables to index?

Answer: In most cases, multiple variables are used to query a data set; however, it would be a mistake to index all variables. You should index variables whose data is discriminate and uniformly distributed.

Question: What is a discriminate variable?

Answer: A key variable should be discriminating, which means that the variable can limit the number of observations returned. Select a key variable that tends to have more unique values for observations. For example, EMPID and LASTNAME are discriminating because it is less likely that many employees share the same employee number or last name. However, AGE, FRSTNAME, and GENDER are not discriminating because it is very possible for a large number of observations to have the same age, first name, and gender.

Question: Why should a key variable have data that is uniformly distributed?

Answer: Data that is uniformly distributed means that the number of observations are evenly distributed between the minimum and maximum values. Data distribution is important because of the way SAS decides whether to use an index to process a WHERE expression. When determining whether to use an index, SAS estimates the number of observations that will be qualified, then compares resource costs of using the index to sequentially reading the entire data set. To estimate the number of qualified observations, SAS assumes that the data is uniformly distributed.

To determine data distribution for a variable, you can use PROC CHART or PROC UNIVARIATE. If the plot line is relatively smooth or flat, then the data is uniformly distributed. If the plot line portrays the classic bell curve, the data has a normal distribution and indexing would tend to be less effective. For more information on PROC CHART and PROC UNIVARIATE, refer to *SAS® Procedures Guide, Version 6, Third Edition*.

For example, assume you are considering indexing a variable that represents IQ values ranging from 55 to 145. Below is a PROC CHART on data set WORK.IQ for variable IQ:

```
proc chart data=work.iq;
   vbar iq/discrete midpoints=55 to 145 by 15;
run;
```

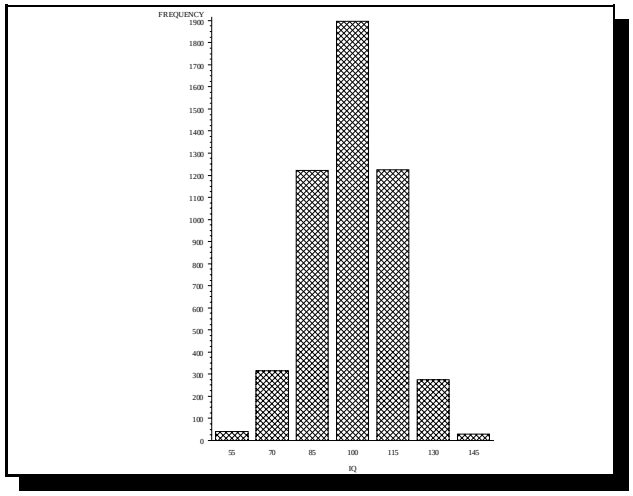

Figure 1 PROC CHART on WORK.IQ

To illustrate the plot line, Figure 2 plots the IQ values on the horizontal axis, showing that the distribution of values approximates a bell-shaped normal probability curve. The shaded area under the curve represents the qualifying observations with an IQ of 130 or above, which is a relatively small subset.

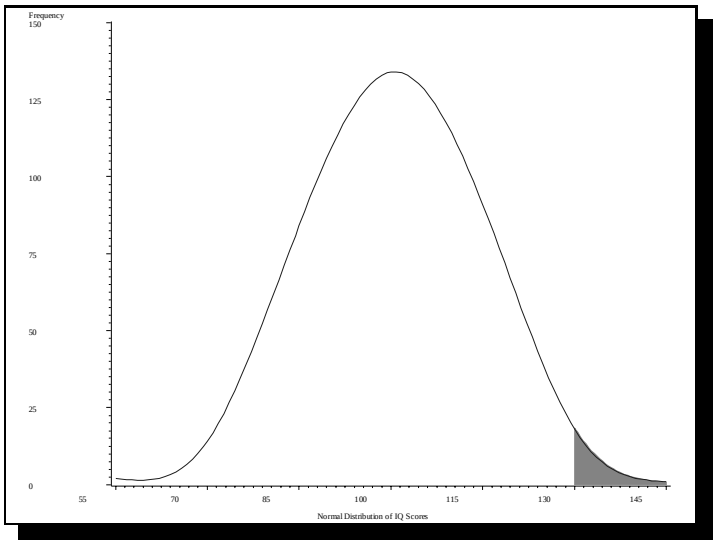

Figure 2 Distribution of IQ Values

However, to decide whether to use an index to process the condition `where IQ > 130;`, SAS would estimate the number of qualifying observations based on a uniform distribution. Figure 3 shows how a uniform distribution of IQ values would look. The shaded area represents the qualifying observations with an IQ of 130 or above, which is a larger subset than actually exists.

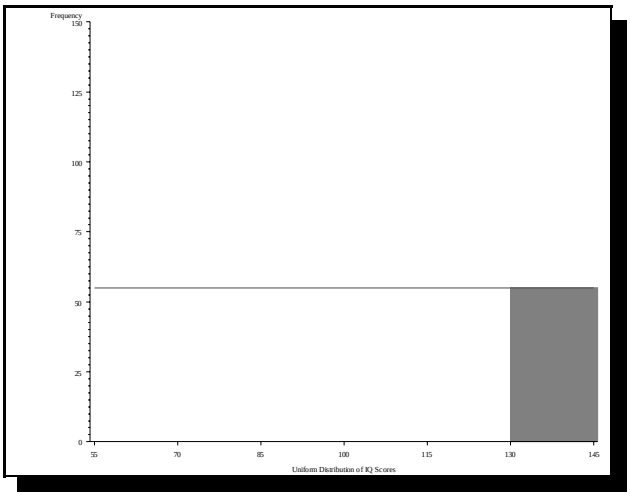

Figure 3 Uniform Distribution of Values

By assuming a uniform distribution of values, SAS would over estimate the number of qualifying observations, which could cause the software to make the wrong decision as to whether to use the index. Therefore, if a key variable's data is not uniformly distributed, SAS may not make the correct decision.

Note that in Version 7, the ability of the SAS System to estimate the number of qualifying observations will be improved by the software storing additional statistics called cumulative percentiles (or *centiles*). Centiles information represents the distribution of values in an index so that SAS will not have to assume a uniform distribution for a key variable.

Question: Are variables with normally distributed data then not acceptable as key variables?

Answer: No. This is not to say that indexes are not useful for normally distributed data. For uniform data, queries that test for equality (where iq = 100;) or for ranges (where iq between 100 and 115;) are typically better than using open-ended queries (where iq > 130;).

Question: Does using an index save CPU time?

Answer: Yes, it can. To understand, compare using a book index to using a SAS index:

- A book index allows a reader to locate a topic along with its specific page number. The reader then goes to the appropriate pages and reads only about the topic. This saves the reader time by not having to read other topics in order to locate the topic of interest.

- Using a SAS index is much the same. It allows SAS to read only the observations of interest. All the unrelated observations are not read. Therefore, an index can save CPU and input/output (I/O) time.

Performance gains with an index increase as the size of the subset decreases in comparison to the size of the data set. The chart below provides basic guidelines:

Subset Size	**Indexing Action**
100%-34%	Do not index. A sequential read of the entire data set is probably more efficient.
33% - 21%	An index may improve performance. Consider comparing CPU usage with and without an index. For some operating environments, you can issue the STIMER and FULLSTIMER system options to write performance statistics to the SAS log.
20% - 16%	An index is likely to improve performance.
15% - 1%	An index is very likely to improve performance.

Caution: The previous percentages are only guidelines. Many factors in an operating environment can affect performance.

Question: If data is rarely accessed, is it worth the resources to create, use, and maintain an index?

Answer: As a general rule, an index should be used often in order to make up for the resources used from creating and maintaining the index. However, you should consider improved response time as well as resources.

For example, assume a data set is used at the front desk of a hotel. A state inspector comes in once a month and requests maintenance dates on the pool. Even though the query is not often, it is still important for the hotel desk clerk to get the information as quickly as possible. On the other hand, if the state inspector sends the request through the mail, an index is not warranted because the request is not often and response time is not important.

Creating an Index

The following questions explain how you can create an index and provide some guidelines to consider when you are creating one.

Question: How do I create an index?

Answer: The base SAS product offers multiple methods to create and delete an index. Each is briefly described below. For details, refer to the appropriate documentation.

1. The DATASETS procedure supports creating and deleting indexes as well as the NOMISS and UNIQUE options.

```
INDEX CREATE indexname <=(variable-list)></ UNIQUE NOMISS>;

INDEX DELETE index-list;
```

For example, using the data set WORK.EMPLOYEE, the following statements perform several functions. First, they create a composite index named NAME on variables LAST and FIRST. They also create a unique simple index on the variable SSN. Finally, they remove the existing index AGE.

```
proc datasets;
modify employee;
index create name=(last first);
index create ssn / unique;
index delete age;
```

2. The SQL procedure also creates and deletes indexes and supports the UNIQUE option.

```
CREATE <UNIQUE> INDEX indexname ON table-name(column-name
   <,column-name>...);

DROP INDEX indexname <,indexname>...FROM table-name;
```

For example, using the WORK.EMPLOYEE data set, the following statements create a composite index named NAME on variables LAST and FIRST, create a unique simple index SSN, and delete the existing index AGE.

```
proc sql;
create index name on work.employee(last, first);
create unique index ssn on work.employee(ssn);
drop index age from work.employee;
```

3. Release 6.07 provided the INDEX= data set option so that you could create an index in a DATA step. The INDEX= data set option also supports the UNIQUE and NOMISS options.

```
INDEX=(index-1 <=(variable-list1)><...index-n<=(variable-list-
  n)>><</UNIQUE><</NOMISS>));
```

For example, to create a unique simple index on variable SSN and a composite index named NAME on variables LAST and FIRST, you could issue the following DATA statement:

```
data work.employee(index=(ssn/unique name=(last first)));
```

4. The SAS Display Manager also allows creating and deleting indexes with the INDEX CREATE and INDEX REVIEW windows. For a complete description, refer to the SAS Technical Report P-222 “Changes and Enhancements to Base SAS Software” on page 119.

5. The SAS/IML® software and SAS/Warehouse Administrator™ software support index creation and deletion. For more information, refer to *SAS/Warehouse Administrator™ Users Guide* and *SAS/IML® Software Usage and Reference, Version 6, First Edition*.

Question: Does the SAS System sort the values for a key variable before creating the index?

Answer: Yes. Values are retrieved from the data set, sorted in ascending value order, then written to the index file.

Question: Will sorting the data set provide better performance?

Answer: Yes, in most cases. To reduce the number of I/Os performed when creating an index, first sort the data set by the key variable. Then to improve performance when using the index, maintain the data set in key variable sorted order. This technique reduces I/Os by grouping like values.

I/Os are reduced because data is transferred between external storage to memory in data set pages. A page is an amount of data (the number of observations) that can be transferred for one I/O request. Each data set has a specified page size. For example, consider how SAS would process the WHERE expression `where 10<x<30;`, which requests a range of values. To access the first qualified value, SAS would move the appropriate data set page to memory. For the next qualified value, if the data set is sorted, chances are that the value resides on the same page, which is already in memory and does not require another I/O. In comparison, if the data set is not sorted and the values are distributed throughout the data set, the next qualified value would be on a different data set page and would require another I/O to transfer it.

Note that it takes additional CPU time to sort a data set. Generally, the index should be used often to recoup the resources used to keep the data set sorted.

Question: Can an index be created on tape?

Answer: Since a tape is a sequential media, an index cannot be created on tape.

Question: To create a composite index to be used for BY processing, does the order of the variables in the index definition matter?

Answer: Yes. For example, the following BY statement specifies SSN first, then DATE. You would need to define the composite index in the same order. If the composite index is defined with DATE first and then SSN, the index cannot be selected for the BY statement.

```
by ssn date;
```

Using an Index for WHERE Processing

WHERE processing conditionally selects observations by your issuing a WHERE expression. Using an index to process a WHERE expression improves performance and is referred to as optimizing the WHERE expression. To process a WHERE expression, SAS decides whether to use an index or sequentially read all the observations in the data set.

Question: How does the SAS System determine whether to use an index to optimize a WHERE expression?

Answer: To make the decision, SAS first identifies an available index or indexes, estimates the number of observations that will be qualified, then compares resource usage to decide whether it is more efficient to satisfy the WHERE expression using the index or by reading all the observations.

Question: Are there certain types of conditions that are acceptable for index processing?

Answer: SAS attempts to use an index for the following types of conditions:

Operators	**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 lower 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 frstname like '%Rob_%;
IS NULL and IS MISSING operators	where name is null; where idnum is missing;

Question: Are there any functions that allow the WHERE expression to be optimized?

Answer: Yes, the TRIM and SUBSTR functions support using an index. However, the SUBSTR function must be in the following form:

```
WHERE SUBSTR (variable,start<,length>)='character_string'
```

with the following conditions:

- *start* is equal to 1
- *length* is less than or equal to the variable length
- *length* is equal to the length of the *character_string*.

Question: My WHERE expression compares a key variable to another variable. Can the index be used?

Answer: No. Variable-to-variable conditions are not supported.

Question: What happens if a WHERE expression has multiple conditions, such as

```
where zipcode='78753' and id='55';
```

Answer: Even though a WHERE expression can consist of multiple conditions specifying different variables, SAS uses only one index and tries to select the one that satisfies the most conditions and selects the smallest subset.

For the most part, SAS selects a condition specifying a variable that has either a simple index or is the first key variable in a composite index. However, you can take advantage of multiple key variables in a composite index by constructing the WHERE expression appropriately, which is referred to as *compound optimization.*

Question: How do I take advantage of compound optimization?

Answer: Compound optimization is the process of optimizing multiple conditions specifying different variables, which are joined with logical operators, such as AND or OR, using a composite index. Note that for compound optimization, SAS supports all the same conditions as single-condition optimization except for the CONTAINS operator, the pattern-matching operators LIKE and NOT LIKE, and the IS NULL and IS MISSING operators. Also, functions are not supported.

For compound optimization to occur, all of the following must be true:

- Starting at the left side of a composite index definition, at least the first two key variables must be used in the WHERE conditions.

- The conditions must be connected using the AND logical operator or using the OR logical operator as long as the OR conditions use the same variable. For example:

```
where lastname eq 'Smith' and frstname eq 'John';

where frstname='John' and (lastname='Smith' or
lastname='Jones';
```

- At least one condition must use the EQ or IN operator, which means that you cannot have, for example, all range conditions.

Question: How does the SAS System estimate the number of observations that will be qualified by an available index?

Answer: To estimate the number of observations, SAS assumes a uniform distribution of data, which means that the number of observations should be evenly distributed between minimum and maximum values. If the data for a key variable is not uniformly distributed, SAS could over estimate the number of qualified observations, which could cause SAS to make an incorrect decision when comparing resource costs.

Question: When multiple indexes exist, how does the SAS System determine which index to use?

Answer: Once SAS identifies the index or indexes that can satisfy the WHERE expression, the software estimates the number of observations that will be qualified. When multiple indexes are available, SAS selects the one that returns the smallest subset of observations. As a general rule, SAS uses an index that qualifies approximately one-third or fewer observations.

Question: Once the SAS System estimates the number of qualified observations and selects the index that qualifies the fewest observations, how does SAS compare resource usage?

Answer: SAS decides if it is faster and, therefore, less expensive to satisfy the WHERE expression using the index or reading the entire data set sequentially by comparing resource usage. That is, if only a few observations are qualified, it is more efficient to use the index. However, if most or all of the observations qualify, then it is more efficient to simply read the entire data set.

This decision is much like a reader deciding whether to use an index at the back of a book. A book index is designed to allow a reader to locate a topic along with the specific page number. Using the index, the reader would go to the specific page number and read only about a specific topic. If the book covers 42 topics and the reader is interested in only one, then the index saves time by preventing the reader from reading other topics. However, if the reader is interested in 39 topics, then searching the index for each topic would take more time than simply reading the entire book.

To make the decision, SAS first predicts the number of I/Os it will take to satisfy the WHERE expression using the index. Then SAS calculates the I/O cost of a sequential pass of the entire data set. Finally, SAS compares the two resource costs. If comparing resource costs results in a tie, SAS chooses the index.

Factors that affect the comparison include the size of the subset relative to the size of the data set, data set value order, data set page size, the number of allocated buffers, and the cost to uncompress a compressed data set for a sequential read.

Question: I have a NOMISS index. SAS does not use the index even though the variable in the WHERE expression is indexed. Can SAS use NOMISS indexes to optimize a WHERE expression?

Answer: A NOMISS index cannot be used for any WHERE condition that qualifies missing values. For example, SAS may use an index on the variable X for the following WHERE expressions since the conditions exclude missing values:

```
where 1<x<4;
where x in (1,2,3);
```

However, for the following WHERE expressions, SAS cannot use an index on the X variable since the conditions qualify missing values:

```
where x<4;
where x=.;
where x not in (1,2,3);
```

Question: When using the logical operator OR in a WHERE expression, the index is not used. Why?

Answer: An index that is selected to optimize a WHERE expression must return all observations that satisfy the WHERE conditions. Often with an OR operator, one index cannot return all possible observations. For example, consider the following WHERE expression:

```
where x=1 or y=3;
```

Assume a simple index on X and a simple index on Y. If X is used, it only selects the observations with a value of 1 for variable X; it does not guarantee selection of all observations with a value of 3 for variable Y. The same occurs if Y is used. It cannot guarantee selection of all observations with a value of 1 for variable X. Therefore, neither index can return all observations that meet the WHERE criteria.

However, consider the following WHERE expression:

```
where (x=1 or y=3) and j<4;
```

Assume a simple index on all three variables. In this situation, the simple index J can select all observations that meet the WHERE criteria.

Question: Assume indexes exist in all combinations for the variables specified in the following WHERE expression. What is the selection process for the best index?

```
where x=1 and y=2 and z=3;
```

Answer: SAS considers the indexes in the following order:

1. First, SAS considers all composite indexes that include all three variables.

   ```
   xyz, yzx, zxy, xzy, yxz, zyx
   ```

2. Next, SAS considers other composite indexes.

   ```
   xy, yx, yz, zx, xz, zy
   ```

3. Then, SAS considers composite indexes that include key variables not listed in the WHERE expression.

   ```
   xya, yxa
   ```

4. Finally, SAS considers simple indexes on the variables.

   ```
   x, y, z
   ```

Question: If a WHERE expression is used with a BY statement, what happens?

Answer: If both a WHERE expression and a BY statement are specified, SAS searches for one index that satisfies the requirements for both. If one is not found, the BY statement takes precedence. If an index is available for the WHERE expression only, it is not used.

Question: What happens if a composite index satisfies some conditions in the WHERE expression but not all?

Answer: The composite index can still be used for optimization, but less selectivity occurs from the index. For example, suppose you have a composite index for LASTNAME and FRSTNAME. If you issue the following WHERE expression, SAS uses the concatenated values from the composite index for the first two conditions, then examines the subset and eliminates observations that do not satisfy the last condition:

```
where lastname eq 'Smith' and frstname eq 'John'
   and empnum eq 3374;
```

Question: A WHERE clause is available for PROC SQL. Can the WHERE clause use an index?

Answer: Yes, the SQL WHERE clause can use an index just like a WHERE statement or a WHERE= data set option. Additionally, the SQL Query Optimizer can use indexes to optimize joining of data sets. If a data set is indexed on the join key used in the WHERE clause, the rows of one data set can be accessed sequentially while the matching rows are accessed through the index.

Question: How can I force the SAS System to use an index or to not use an index to optimize a WHERE expression?

Answer: Generally, SAS determines whether an index is used. However, the internal system option $IWEIGHT= can affect whether the index is more likely or less likely to be used. $IWEIGHT= is available beginning in Release 6.07 and can be specified in an OPTIONS statement. The default value is 100, which tells SAS to decide whether to use an index based on its standard resource comparisons. If you specify a smaller value, SAS is more likely to use the index; if you specify a larger value, SAS is more likely to not use the index.

To cancel a WHERE expression optimization, provide a value greater than or equal to the number of observations in the data set multiplied by 100, such as, *number-of-observations* x 100. $IWEIGHT= has no affect on BY processing.

To reset $IWEIGHT=, issue

```
options $iweight=100;
```

Caution: $IWEIGHT= is an undocumented and untested option. Adjusting its value could adversely affect your performance.

Note that in Version 7, you will be able to control index usage for WHERE processing with the IDXWHERE= and IDXNAME= data set options. The IDXWHERE= data set option will allow you to override the software's decision regarding whether to use an index, and the IDXNAME= data set option will allow you to direct SAS to use a specific index.

Question: Is there a way to verify the usage of an index?

Answer: To display information in the SAS log regarding index usage, specify MSGLEVEL=I in a system option statement.

```
options msglevel=i;
```

The SAS log displays an INFO message regarding index usage. Note that in the SAS/SHARE® environment, the option must be specified to the server.

Question: In what order are observations returned if an index is selected?

Answer: If SAS selects and uses an index, the observations are retrieved in ascending value order. This happens because the values of key variables are stored in the index in ascending value order. If an index is not used, observations are returned in the physical order in which they reside in the data set.

Question: Can the values of a key variable be retrieved in descending order?

Answer: No.

Using an Index with Views

You cannot create an index for a data view; it must be a data file. However, if a data view is created from an indexed data file, index usage is available. For example, if the view definition includes a WHERE expression using a key variable, then the SAS System will attempt to use the index.

Question: I created an SQL view named STAT from the data file CRIME, which has an index on the variable STATE. In addition, I defined a WHERE expression for the SQL view definition.

```
proc sql;
create view stat as
select * from crime
where murder>7;
```

When I issue the following PRINT procedure, which refers to the SQL view, along with a WHERE statement that specifies the key variable STATE, why is the index not used?

```
proc print data=stat;
   where state > 42;
```

Answer: SQL views cannot join a WHERE expression defined in the view to a WHERE expression specified in another procedure, DATA step, or SCL. Therefore, the index for STATE cannot be used to optimize the WHERE statement.

However, if you execute the SQL view STAT in a PROC SQL step with an SQL WHERE clause that specifies the key variable STATE, the SQL view can join the two conditions to create `where murder>7 and state>42;`, which allows SAS to use the index STATE. For example

```
proc sql;
select *
from stat where state>42;
quit;
```

Question: Can a WHERE expression specified, for example, in a WHERE statement be joined with a WHERE expression defined in a DATA step view?

Answer: No. For example, consider the following DATA step that creates a view named STAT from the data file CRIME, which has an index on the variable STATE. In addition, the view definition includes a WHERE expression:

```
data stat/view=stat;
set crime;
where murder>7;
run;
```

If you refer to the view in another DATA step that specifies a WHERE statement using the key variable STATE, the view cannot join the two conditions and the index on STATE cannot be used.

```
data result;
set stat;
where state>42;
run;
```

Question: Can SAS/ACCESS views accept WHERE statements from DATA and PROC steps?

Answer: Yes. WHERE statements and BY statements are pushed down into the DBMS from the SQL procedure, other procedures, and DATA steps.

Question: Are PASSTHROUGH views treated differently than SAS/ACCESS® views?

Answer: SQL PASSTHROUGH views do not recognize WHERE or BY statements specified outside the view. The DBMS code embedded in the SQL PASSTHROUGH view is executed as written to take advantage of special features such as indexing. This processing occurs on the DBMS side; therefore, use of the index is determined by the DBMS.

Using an Index for BY Processing

BY processing allows you to process observations in a specific order according to the values of one or more variables specified in a BY statement. Indexing a data set allows you to use a BY statement without sorting the data set. By creating an index based on one or more variables, you can ensure that observations are processed in ascending numeric or character order.

Question: Is an index used if the variables specified in the BY statement are key variables?

Answer: Unlike WHERE processing, the BY statement always uses a suitable index, unless the DESCENDING or NOTSORTED option is specified in the BY statement or the index was created with the NOMISS option. For example, if an index exists for LASTNAME, the following BY statement uses the index to order the values by last names:

```
proc print;
   by lastname;
```

Question: I have a simple index for variable SSN. I issue the following BY statement, but I get a message that values are not properly sorted. Is the index being used?

```
by ssn date;
```

Answer: The index on SSN is selected for processing. However, the values of DATE must also be arranged in sorted order within SSN.

Question: If a simple index exists on the variable SSN and a composite index exists on SSN and DATE in that order, which index is used and why?

Answer: The composite index is selected to process the BY statement. SAS selects the index that most closely matches the variables in the BY statement.

Specifying an Index with the KEY= Option

The SET and MODIFY statements provide the KEY= option, which allows you to specify an index in a DATA step to retrieve particular observations in a data set.

Question: I know the KEY= option for the MODIFY and SET statement became available in Release 6.07. Can it take advantage of composite indexes as well as simple indexes?

Answer: Yes. The KEY= option accepts the index name, which is the variable name for a simple index. To use a composite index, specify the composite index name.

The syntax of the KEY= option is specified in the form of

```
SET dsname KEY=index-name;
MODIFY dsname KEY=index-name;
```

Maintaining Indexes

The SAS System provides several procedures that you can issue to maintain indexes, and there are several operations within SAS that automatically maintain indexes for you.

Question: How do I display data set information so that I can keep track of my indexes?

Answer: The CONTENTS procedure reports the following types of information. (However, the available information depends on the operating environment.)

- number and names of indexes for a data set
- names of key variables

- options (NOMISS and UNIQUE) for each key variable
- data set page size
- number of data set pages
- amount of disk space used by the index file.

Question: My index is not copied to the output data set specified in a DATA step when specifying an indexed data set in the SET statement. How do I maintain the index on the data set?

```
data a; set a ; /* data set A is an indexed data set */
run;
```

or

```
data b; set a ; /* data set A is an indexed data set */
run;
```

Answer: Maintain an index on data sets by using procedures, functions, and statements that perform an update-in-place. Following is a list of procedures, statements, and functions that perform updates on the original data set while maintaining the original indexes.

PROC FSEDIT
PROC FSVIEW
PROC APPEND
PROC SQL
MODIFY statement available in DATA step
SCL functions such as UPDATE and APPEND.

Question: I have a data set with the variable SSN (social security numbers). I have seven indexes that have SSN as a key variable. I delete between 5 to 10% of observations, but the update is taking a long time. Can the number of indexes be affecting the CPU time it takes for my job to run?

Answer: Yes. Any time values for a key variable are added, deleted, or modified in the data set, all of the indexes for that variable are updated. Computer resources are consumed in maintaining each index. Therefore, it is best to limit the number of indexes that you create for a data set.

Question: I have a simple index on variable X and a composite index on variables X and Y. When observations are removed, are both indexes affected?

Answer: When adding or removing observations, SAS automatically maintains all the indexes associated with the data set. Any updates to values of a key variable updates the data set's indexes as well.

Question: What is a logical deletion?

Answer: A logical deletion means that the observations are removed from the index file but still reside on the physical media. The physical deletion is not performed until the data set is moved or rebuilt. Use the CONTENTS procedure to determine the number of logical deletions performed on the data set.

Question: Is there a way to recover an observation that is logically deleted?

Answer: No.

Question: If I delete the physical data set, does the index file remain?

Answer: If the data set is deleted with a SAS utility such as PROC DATASETS, the associated index file is also

deleted.

Question: What happens if the index file is deleted by a method outside of the SAS System?

Answer: SAS knows an index file is associated with a data set. If the index file cannot be located when accessing the data set, you are prompted to rebuild the index file.

Question: If I rename the data set, what happens to the associated index file?

Answer: When the data set is renamed using a *SAS utility* such as PROC DATASETS, the index file name is automatically renamed to the new data set name. If the data set name is renamed by changing the physical filename outside the SAS System, the physical filename containing the index file must also be changed to correspond with the new data set name. If the index filename is not changed, the index file cannot be located when the data set is accessed. You are then prompted to rebuild the index file.

Question: What happens to an index if the key variable name is changed?

Answer: When a key variable is renamed, the index is also updated automatically. For a simple index, the index name changes. For a composite index, the variable is renamed, and the index name remains the same.

Enhancing Performance

The following information provides some tips on how you can enhance index performance.

Question: How does increasing or decreasing the data set page size affect performance?

Answer: SAS transfers data between external storage to memory in *pages*, which is an amount of data (the number of observations) that can be transferred for one I/O request; each data set has a specified page size. In general, the larger the page size, the fewer the number of I/Os that will be required. For example, if the page size is 512 bytes, then one I/O transfers 512 bytes of data (or one page). To reduce I/Os, you can increase the page size. However, this will increase the amount of memory usage. If you have memory constraints, you can reduce the page size, but this will increase the number of I/Os. For additional information, refer to the *SAS Companion* for your operating environment.

For information on data set characteristics like the data set page size and the number of data set pages, issue the CONTENTS procedure. The BUFSIZE= data set option (or system option) sets the page size for a data set when it is created.

Question: Is there an option similar to BUFSIZE= for controlling the default index page size?

Answer: An undocumented system option called $IBUFSIZE= is available and follows the same constraints and guidelines as the BUFSIZE= option.

Caution: $IBUFSIZE= is an undocumented and untested option. Adjusting its value could adversely affect your performance.

Question: How does increasing or decreasing allocated buffers affect processing?

Answer: The number of allocated buffers determines how many data set pages can simultaneously be in memory. SAS transfers data between external storage to a buffer, which is the memory into which data is read or from which data is written. Frequently, the larger the number of buffers, the fewer the number of I/Os required. However, if memory is a constraint, reduce the number of allocated buffers. For additional information, refer to the *SAS Companion* for your operating environment.

The BUFNO= data set option (or system option) specifies how many buffers to allocate for a data set.
Question: Is there an option similar to BUFNO= to specify buffers for the index file?
Answer: An undocumented system option called $IBUFNO= is available and follows the same constraints and guidelines as the BUFNO= option.

Caution: $IBUFNO= is an undocumented and untested option. Adjusting its value could adversely affect your performance.

Conclusion

Indexing can be a powerful performance tool for your SAS applications. For each data set, you can create multiple simple and/or composite indexes. Indexes can improve performance for WHERE processing and BY processing. In addition, you can specify an index using the KEY= option for the SET and MODIFY statements. Note that for WHERE processing, SAS decides whether to use an index.

Even though an index can reduce the time required to locate a set of observations, there are costs associated with creating, storing, and maintaining an index. Consider indexing data sets only if the majority of your queries create small subsets of a large data set. Key variables should be discriminating, and the values should be uniformly distributed between the minimum and maximum values.

So that the SAS System can automatically maintain an index, use functions, statements, and procedures that update-in-place.

Reprinted with permission from Observations®. This article, number obswww08, is found at the following URL: www.sas.com/obs

References

The following are reference sources used for this article:

Beatrous, Steve and Clifford, William (1988), "Version 6 SAS? Data Base System Architecture: Current and Future Features," *Proceedings of the Thirteenth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., 145-154.

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

Clifford, William, Beatrous, Steve, Stokes, John, and Mosman, Kevin (1989), "Using New SAS® Database Features and Options," *Proceedings of the Fourteenth Annual SAS® Users Group International Conference*, Cary, NC:
SAS Institute Inc., 335-346.

Kent, Paul, Technical Note - TS320, *Inside PROC SQL's Query Optimizer*, Cary, NC: SAS Institute Inc.

Norton, Andrew (1992), "The View Review: PROC SQL views, SAS/ACCESS® Views, and DATA Step Views," *Proceedings of the Seventeenth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc.,

1059-1064.

Raithel, Michael A. (1995), *Tuning SAS® Applications in the MVS Environment*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS® Language: Reference, Version 6, First Edition* , Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS/IML® Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS® Procedures Guide, Version 6, Third Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS/Warehouse Administrator™ User's Guide, Release 1.1, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1991), SAS® Technical Report P-222, *Changes and Enhancements to Base SAS Software, Release 6.07*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1996), *Scalable Performance Data Server™ User's Guide, Version 1, First Edition*, Cary, NC: SAS Institute Inc.