USING NEW SAS® DATABASE FEATURES AND OPTIONS

William D. Clifford, SAS Institute Inc., Austin, TX
Stephen M. Beatrous, SAS Institute Inc., Cary, NC
John T. Stokes, SAS Institute Inc., Austin, TX
Kevin L. Mosman, SAS Institute Inc., Austin, TX

ABSTRACT

This paper describes some of the new SAS® database performance tuning features and provides guidelines for their use. As is true with most performance options, many of these features improve the utilization of one resource at the expense of another. The benefits and costs of each new feature are examined. All of these features are optional since the system provides defaults. Actual examples of performance improvements using the new features are given. These examples are based upon the SAS Usage Notes data set, with which you may be familiar.

INTRODUCTION

The portable base SAS I/O engine contains some optional features that can be tuned to match an application's resource requirements. By default, the SAS System supplies values for these features based upon somewhat limited data. The SAS programmer (someone who creates applications by writing SAS code) is often able to determine values for these features that will increase the performance of an application.

There is usually a price to be paid for performance gains. The objective of this paper is to describe some of the parameters that control the resource utilization and to provide guidelines so that the SAS programmer can make informed choices about how to set these optional parameters to improve an application's performance.

Three new features are presented and guidelines are given for their use.

1. options to set the size and quantity of I/O buffers.
2. two kinds of data set records: compressed and un-compressed
3. indexes for data set variables.

The SAS I/O architecture is presented as background information in preparation for discussing each parameter that can affect performance. Their default value, and the syntax necessary to change them is included in the discussion. General, rather than specific, guidelines for setting these parameters are given because an application's data requirements and the computing environment in which it runs vary greatly. It is due to this variation that the guidelines presented here are not guaranteed to produce the described benefits in all cases. The best performance will be achieved by experimenting (tuning) with the parameter values.

The SAS Usage Notes application is used to demonstrate the performance improvements and additional resource costs. Performance and cost are measured for individual resource utilization features.

Explanation of Examples

The examples chosen for demonstrating the new database features reflect a variety of applications on a production file. The intent is to show the effects of the new database features on a "real" application. By using a so called "real" application we expect to legitimize our conclusions.

We believe the SAS Usage Notes data set represents a "typical" SAS data set with respect to size and content.

Number of Observations:	6176
Number of Variables:	61
Record Length:	2395

And the Usage Notes data set is distributed to SAS sites. Its content, therefore, may already be familiar to our readers.

The performance impact of each feature is reported in terms of:

- disk usage
- elapsed time
- CPU time
- I/O count (EXCP count on MVS)

The measurements were all done on IBM® MVS, but we expect that similar results could be

demonstrated on all of the other hosts.

The performance of nine "typical" applications is measured using the SAS Usage Notes data set. The applications include interactive queries using the FSEDIT procedure, report generation, and batch updates. Each application is given a name and described below.

It is important to understand the format used to report the performance effects described later in the paper. Each application was run using the default option values to generate "base" performance numbers. Then each application was run to demonstrate a single option, generating "new" performance numbers. The performance effect reported is new/base. Thus values close to 1 indicate little or no change in performance. Large numbers indicate an increase in resource utilization, that is, worse performance. Numbers less than 1 indicate a decrease in resource utilization, or improved performance. For example, 5 means 5 times more resources were used, and .02 means 50 times fewer resources were used.

FSED1 - In the FSEDIT procedure, scroll down 100 records and up 100 records. The purpose of FSED1 is to measure the performance of a casual browse of the data set.

FSED2 - Using FSEDIT, browse a subset of the SAS/SHARE® Usage Notes with the where clause:

```
WHERE PROD = 'SHARE'
```

Scrolling down 100 records and up 100 records measures the performance of a simple query.

FSED3 - Using FSEDIT, browse a subset of the SAS/SHARE Usage Notes that concern system abends.

```
WHERE PROD = 'ETS' AND KEYS ? 'ABEND'
```

Scrolling down 100 records and up 100 records measures the performance of a more complex query.

FSED4 - In FSEDIT, ask to see a record that does not exist on the file. This is done by using the where clause:

```
WHERE PROD='NOPROD'
```

The purpose of FSED4 is to force the entire file to be searched for a particular variable's value. Every record must be examined in order to determine that the value does not exist.

FSED5 - Using FSEDIT, update a keyed (indexed) variable in 25 records.

FSED6 - Same as FSED5, but the variable updated is not keyed. The purpose of FSED5 and FSED6 is to measure the cost of updating keyed variables.

ADD - Use the APPEND procedure to obtain the cost of adding 1000 new records to the data set.

REPORT- Use the TABULATE procedure to produce a report of a subset of the records in the data set. The report tabulates the number of SAS/GRAPH® and SAS/SHARE usage notes by release.

BATCH - This is an application that consists of a series of data steps and SORT procedures to update a master file from a transaction file. Running the BATCH example reads the original data set and a transaction file and recreates the data set after merging the appropriate data from the transaction file. The purpose of BATCH is to measure the performance of the kind of job that might be run as a nightly update to a data set.

An appendix at the end of the paper includes the actual results from running these programs.

SAS I/O ARCHITECTURE

SAS software is composed of three main layers: the applications, or PROC layer, on top, the supervisor/engine layer in the middle, and the host layer at the bottom. The applications layer and the supervisor/engine layer are both portable. The host layer is machine and operating system dependent. This paper's primary focus is on the portable layers.

Data is organized into multiple levels of varying-sized units. At the conceptual level, the smallest unit of data is the variable. A record, also known as an observation, is composed of one or more variables. The term "record" will be used in this paper, but it is understood that "record" and "observation" are synonyms. Moving closer to physical storage considerations, a page contains one or more records, and is stored on a mass storage device in one or more physical blocks.

The applications layer manipulates data in terms of variables and records. The supervisor/engine

layer sees data as variables, records, and pages. And the host layer sees data as pages and blocks of files. The page size is a permanent attribute of the file. A data set is a file containing observations (records).

The host layer is responsible for understanding the physical characteristics of its mass storage devices. Thus it is responsible for determining optimal page sizes and making that information available to the supervisor/engine layer. The supervisor/engine layer selects a page size based upon one or more of the following:

1. the host's block size
2. the host's optimal page sizes
3. the application's record size (the sum of the variables' sizes)
4. the record type
5. application-specified tuning options.

Each open file requires one or more page-size memory buffers to hold a file's data while it is being processed. The supervisor/engine selects the number of page buffers based upon one or more of the following:

1. file open mode
2. file size
3. application-specified tuning options.

APPLICATION-SPECIFIED OPTIONS

Three tuning options are described in this section: BUFNO, BUFSIZE, and COMPRESS. Each option can be specified on a global OPTIONS statement, or may be specified as a data set option to override the global option value. For example:

```
OPTIONS BUFSIZE = 4000;

DATA A ( BUFSIZE = 4000 );
```

BUFSIZE

BUFSIZE is used to set the data set page buffer size and thus the mass storage page size for the data set. The page size is a permanent attribute of the data set and applies only when a data set is created. The syntax is:

BUFSIZE = <*bsize*> ;

This option specifies that the data set page size is to be set to a value at least as large as <*bsize*>.

Four mutually exclusive cases are used to determine the actual page size when the data set is created. The host supplies the supervisor/engine with:

```
min_ps      - the physical block size (i.e minimum
              page size)
max_ps      - maximum page size
min_opt_ps  - the minimum optimal page size
max_opt_ps  - the maximum optimal page size
max_waste   - the maximum percent of wasted space
              allowed on a page

Let bufsize  = 0 if no BUFSIZE option specified
             = <bsize>
    onerecpg = the smallest page, rounded up to a
               multiple of min_ps, that will hold
               one record (i.e. record size + page
               overhead + record overhead)
    tenrecpg = the smallest page, rounded up to a
               multiple of min_ps, that will hold
               ten records
    pagesize = data set page size
    waste    = percent of wasted space on a page
```

Regardless of the case, the page size computed by the supervisor/engine must be a multiple of min_ps, and between min_ps and max_ps.

```
Case 1: bufsize is non-zero
        pagesize = MAX( onerecpg, (((bufsize - 1)
        / min_ps) + 1) * min_ps)

Case 2: onerecpg is larger than max_opt_ps
        pagesize = onerecpg

Case 3: file type is un-compressed (described later
        in the COMPRESS section)
        pagesize is set to a multiple of min_ps such
        that

           a. min_opt_ps <= pagesize <= max_opt_ps
              and
           b. waste <= max_waste or
           c. waste is minimized for all page sizes
              satsifying (a).

Case 4: file type is compressed (described later
        in the COMPRESS section)

        pagesize = MIN( tenrecpg, max_opt_ps )
        pagesize = MAX( pagesize, min_opt_ps )
```

BUFNO

BUFNO is used to specify how many page buffers to allocate for an open data set. Unlike BUFSIZE, this specification is not a permanent attribute of the data set. The syntax is:

BUFNO = <*nbufs*> ;

Use of this option specifies that <*nbufs*> page

buffers be allocated for the relevant data set(s). By default, one buffer is allocated.

COMPRESS

A data set page contains an integral number of records. Thus records do not span page boundaries, and a record plus the page overhead bytes (approximately 32) may not exceed the size of a page.

SAS Version 5 of the SAS System supports a single internal record type: un-compressed, or fixed-length records. SAS Version 6 of the SAS System supports un-compressed records as well as a new optional record format called compressed, or variable-length records. A given data set contains only one type of record, either compressed or un-compressed records.

Un-compressed records are stored in an unmodified format: numerics are stored in floating point and characters are stored in fixed-length fields padded with blanks according to their length specification. Each un-compressed record of a given data set occupies exactly the same number of bytes of storage.

Compressed records start with the un-compressed format but are compressed to remove redundancy. Thus a compressed record generally requires fewer bytes of storage than does its un-compressed counterpart. Each compressed record in a given data set may be a different size.

Each page of a data set with un-compressed records contains the same number of records. Pages of a data set with compressed records contain a variable number of records. The significance of this will be discussed shortly.

In Version 5, the only data compression feature available was the use of short numeric variables. Version 6 adds the capability to compress the entire record regardless of its constituent variables. A feature for supporting a user-written compression function is under consideration. Such a function could be designed to compress specific classes of data better than the SAS general purpose compression algorithm (described below). In addition, such a user-written function could provide data encryption for sensitive data.

The COMPRESS option is used to set the data set record type, either un-compressed or compressed. In the absence of a COMPRESS option, the default record type is un-compressed. This option is a permanent attribute of the data set and has meaning only when a data set is created. The syntax is:

COMPRESS = <*flag*> ;

If <*flag*> is "NO", data sets are created with un-compressed records. A value of "YES" for <*flag*> specifies that compressed records are to be used.

SAS Compression Algorithm

The compression function supplied by the SAS System treats the entire record as a single string of bytes. Variable types and variable boundaries are ignored. The algorithm compresses identical consecutive bytes into a maximum of three bytes. From 3 to 129 blanks are compressed to 2 bytes. From 3 to 66 binary zeros are compressed to 2 bytes. And from 3 to 63 occurrences of any other byte are compressed to 3 bytes.

Costs and Benefits of These Options

Use of these options changes the way resources are allocated and what algorithms are used to process the data. This section identifies the impact of specifying each of the options. In many cases a combination of more than one option will be necessary to achieve the optimal performance for a given application. And since applications tend to vary greatly in their processing requirements, the option values for one application will not necessarily provide the same benefits for other applications.

BUFSIZE Tradeoffs

1. The use of large pages reduces the number of I/Os needed to read/write the data set. That is, more data is transferred per I/O with large pages. The disadvantage is that more memory is required for large pages.
2. The use of smaller pages requires less memory but increases the number of I/Os.
3. A page size may be selected that does a better job of packing records into the page than the default algorithm. This can reduce the data set size and thus reduce the number of I/Os to read/write the data set. A potential hazard is that the page size that best fits the record size may not be optimum for the storage device.
4. Selecting a page size that best matches the characteristics of the mass storage device

may waste space in each page because the records are not optimally packed into the page.

Effects of the BUFSIZE Option

Base BUFSIZE = 12288 New BUFSIZE = 24567
Effects are New/Base

PROGRAM	Change in CPU Time	Change in ELAPSED Time	Change in I/O (EXCP count)
ADD	.90	.84	.76
REPORT	.72	.62	.51
BATCH	.77	.68	.52
FSED1	.99	.69	.54
FSED2	.94	.69	.53
FSED3	.92	.66	.52
FSED4	.79	.71	.51
FSED5	.99	.74	.58
FSED6	1.0	.81	.58

In this example, the new page size was doubled. We expected and received a reduction of almost one half of the base I/Os. The ADD program did not show such a large improvement because the page size of the data set containing the records to be added was not increased. The ADD I/O count includes I/Os for the new data set (double page size) plus I/Os for the added records (regular page size).

BUFNO Tradeoffs

Benefits derived from using the BUFNO option to increase the number of buffers all require additional memory as the cost.

Additional buffers can mean fewer I/Os when accessing pages more than once, such as scrolling backwards in FSEDIT, or doing processing on very small data sets. The ideal case for a very small data set is when the number of page buffers is equal to the number of pages on the data set.

More buffers can reduce I/O when doing random processing since they increase the probability that the desired page is already in memory. Examples of random processing include the use of FSEDIT, the POINT= option, and use of a where clause.

Effects of the BUFNO Option

Base BUFNO = 1 New BUFNO = 17
Effects are New/Base

PROGRAM	Change in CPU Time	Change in ELAPSED Time	Change in I/O (EXCP count)
ADD	1.0	1.0	1.0
REPORT	1.0	1.0	1.0
BATCH	1.0	1.0	1.0
FSED1	1.0	.86	.52
FSED2	1.0	1.0	1.0
FSED3	1.0	1.0	1.0
FSED4	1.0	1.0	1.0
FSED5	1.0	1.2	1.0
FSED6	1.0	1.0	1.0

The Usage Notes data set record size is somewhat large, allowing only six records per page. The choice of BUFNO = 17 was made to demonstrate the I/O savings when there are enough buffers to hold all of the needed pages, in this case 17. The performance results show no change for any of the programs except FSED1, the program that reads 100 records and then scrolls back through them. Thus demonstrating that no I/O was necessary for the backwards reading.

COMPRESS Tradeoffs

1. In general, a data set with compressed records requires less mass storage than the same data set with un-compressed records. And thus fewer I/Os are required to process a compressed record data set.
2. The space on a data set occupied by an un-compressed record that has been deleted is never reused. All records added to the data set are stored at the end of the data set. Deleted records may be "removed" only by making a new copy of the old data set.

   Deleted records in a compressed record data set are reused. The SAS System determines where a new record is to be added. It will reuse a deleted record's space if one of suitable size is found, or the new record will be stored at the end. Reusing deleted space prevents the data set size from increasing as long as previously deleted records can furnish sufficient space. Thus new records are not guaranteed to be stored at the end of the file.
3. As with Version 5, a Version 6 data set with

un-compressed records is addressable via "observation number". Since a data set with compressed records has a variable number of records per page, it is not possible to directly calculate the "address" of a given record. Thus such a data set is not addressable via "observation number".

4. Record compression and the additional bookkeeping needed to manage the reuse of deleted record space requires more CPU time to prepare these records for I/O.
5. An updated record in a compressed record file may not fit in its original storage location and there may not be enough extra space on the page. In this case, the larger record is stored on another page with a "pointer" to it in the original location. Accessing this record in the future may cause an extra I/O, but no more than one since such a record is never more than one pointer away from its original location.

Effects of the COMPRESS Option

Base COMPRESS = NO New COMPRESS = YES
Effects are New/Base

PROGRAM	Change in CPU Time	Change in ELAPSED Time	Change in I/O (EXCP count)
ADD	3.0	.94	.71
REPORT	2.8	.29	.28
BATCH	3.4	.52	.29
FSED1	1.0	1.0	.38
FSED2	1.5	.50	.31
FSED3	1.6	.41	.30
FSED4	2.7	.29	.29
FSED5	1.1	.91	.39
FSED6	1.1	.81	.39

Using the compressed record option reduces the data set size from 1256 pages to 349 pages, a 3.6-fold decrease in size. As can be seen from the performance data, the number of I/Os decreased in proportion to the decrease in data set size. And, as expected, CPU time increased.

Guidelines for Using These Options to Improve Performance

Use the CONTENTS procedure to obtain the page size, number of pages, number of records, and the number of records per page for the data set(s) in your application. There are more details later in the section on TOOLS FOR EXPERIMENTATION.

A. If you can afford the extra memory:

   1. Increase the page size. The change in page size will be approximately inversely proportional to the change in the number of I/Os. For example, if you double the page size, you half the number of I/Os.
   2. If you re-read records from a small data set and can allocate a page buffer for each page to be processed, you can save the I/Os for all but the first read of the page.

B. If your system is memory-constrained, reduce the page size.

C. In general:

   1. Change the page size if a different size will reduce the amount of wasted space in each page because records would be packed more efficiently. For one-page data sets, use a page size that minimizes the wasted space.
   2. Consult with your systems analyst to see if a different page size would be more suitable for the mass storage device you are using, or if a different device would be more suitable for your data.

D. Applications may access records by observation number. Some examples are:

   1. the POINT= DATA step option
   2. direct access by observation number to a record in FSEDIT
   3. the FIRSTOBS and OBS options. Note that these options will work with a compressed record file, but they may be slower because the positioning to the specified record will be done by a sequential read of the file.

   If access by observation number is not required, using "COMPRESS = YES" will generally produce a smaller data set requiring fewer I/Os to process. Depending upon the host, the CPU time may not be significantly different than the time required for an un-compressed record data set.

APPLICATION-SPECIFIED INDEX OPTIONS

Index Overview

An index is an auxiliary data structure used to assist in the location (selection) of records specified by the value of a variable. For example "all the records with AGE > 65".

An index is called a regular index if it contains the values of only one variable. A composite index contains the values for more than one variable concatenated to form a single value. A given data set may have multiple regular and/or composite indexes.

In addition, a given index may be specified to contain only unique values. The creation of such an index also prohibits duplicate values for its variable(s) from being stored in the data set. This option is very useful for preventing duplicate values from incorrectly getting into a data set for variables such as social security number. By default, duplicate values are permitted in an index and thus in its data set.

Another permanent index attribute is an option to prevent missing values from being stored in an index. Unlike the unique index option, the missing option does not prevent missing values from being stored in the data set. This feature is useful if the variable(s) contain many missing values that would make the index large and thus slower to access than if they were excluded. By default, missing values are stored in an index.

For more details on indexes, see the paper titled "Version 6 SAS Data Base System Architecture: Current and Future Features" by Steve Beatrous and Billy Clifford presented at SUGI 13.

Syntax and Semantics

There are several SAS software products that allow you to create and use indexes. Each will be briefly described.

1. The DATASETS procedure supports statements that allow you to create and delete indexes, and to specify the UNIQUE and NOMISS options.

```
Identify the target data set by using:

   MODIFY <data_set_name>;

Create an index named <index_name> from the
variable(s) in <variable_list> by using:

INDEX CREATE <index_name>
[= (<variable_list>)][/[NOMISS][UNIQUE]];

Indexes can be deleted by:

   INDEX DELETE <index_name_list>;

An example:  MODIFY EMPLOYEE;
             INDEX CREATE EMPNUM / UNIQUE;
             INDEX CREATE NAMES = ( LASTNAME
               FRSTNAME );
             INDEX DELETE SALARY AGE;
```

2. The SQL procedure supports index creation and deletion, and the UNIQUE option. Note that the variable list requires that variable names be separated by commas (the SQL convention) instead of blanks (the SAS convention).

```
Create a regular or composite index on a table
by using:

CREATE [UNIQUE] INDEX <index_name> ON
  <table_name> (<variable_list>);

Delete an index by using:

DROP INDEX <index_name> FROM <table_name>;

An example:  CREATE UNIQUE INDEX EMPNUM ON
               EMPLOYEE (EMPNUM);
             CREATE INDEX NAMES ON EMPLOYEE
               (LASTNAME, FRSTNAME);
             DROP INDEX SALARY FROM EMPLOYEE;
```

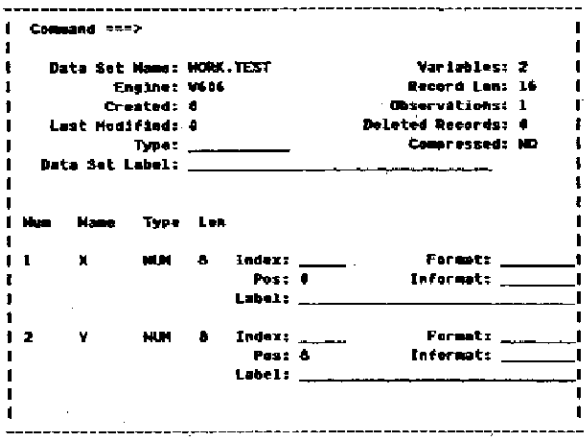
3. The IML procedure allows you to create a regular index with the following statement:

```
INDEX <variable_name>;
```

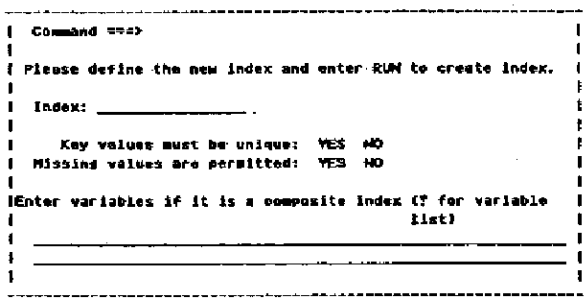
4. The SAS Display Manager System supports a SERVICES command that presents a primary menu for creating and reviewing indexes. Typing "SERVICES" on the command line creates a screen that contains a list of libnames and members. An example is:

```
Command ===>

   Libname   Name      Memtype   Index

_  SASUSER   PROFILE   CATALOG
_  WORK      TEST      DATA
```

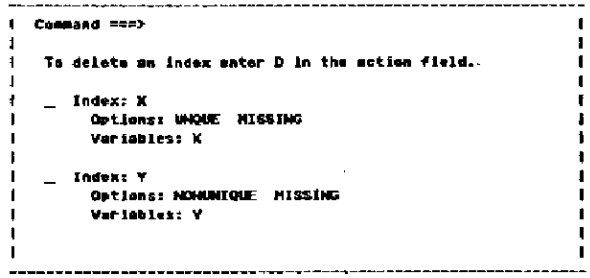
Typing "C" (for contents) in the action field for WORK will create a new screen that lists attributes about the data set TEST.



To create an index, type "INDEX CREATE" on the command line. A smaller screen appears that provides fields for specifying the index attributes.



You can review the indexes that exist and delete any that are no longer needed by typing "INDEX REVIEW" on the command line.



When Indexes are Used

Use of an index provides two major benefits. The first is fast access to a "small subset" of the records represented in the index. And second, all values retrieved via the index are returned in sorted order.

1. The base engine automatically tries to use existing indexes to optimize WHERE and BY statement processing. Variables specified in the where clause are checked for being indexed, including being the first variable of a composite index. The where clause optimizer selects a single index to provide a coarse filter for records returned to the supervisor. The supervisor may do a final filtering of records that get returned to the application if the engine returns records that only partially satisfy the where clause.

   An index is selected for optimization only if both of the following conditions are true. First, the selected index must be the cheapest, that is, select the fewest records among all indexes that could be used. And second, reading the estimated number of records from the candidate index must be cheaper than a sequential pass of the entire data set. The algorithm that estimates the cost to use an index includes the number of data set buffers in its computation.

   When BY processing is specified, the base engine looks for an index, including a composite index, that matchs the variables in the BY statement. If such an index is found, it is used to retrieve the desired records and thus eliminates the need to sort the data set. When a BY statement and a WHERE statement are both used, the same index will be used for optimization if one can be found that satisfies both the WHERE and the BY requirements. However, use of a BY statement may prohibit the use of an index for where clause optimization if no index exists that can be used for both purposes.
2. Where clauses in PROC SQL are passed down to the base engine and thus use indexes as described above. In addition, when performing a join operation on two tables, PROC SQL will look for an index that matches the join variables to optimize the processing.
3. If you use PROC IML, you must explicitly specifiy that an index be used for read, delete, list, or append operations via the INDEX command. If the named index exists, it is used to retrieve the desired records. If the index does not exist, it is created, and then used for record retrieval.

Index Tradeoffs

As with many performance improvements, there is a cost associated with the use of an index.

1. Extra CPU cycles and I/Os are necessary to create and maintain an index. When a new value for an indexed variable is added to the data set, it must also be added to the appropriate index(es). When a value is deleted from the data set, the value must be deleted from the appropriate index(es). And when a value changes, the old value must be deleted and the new value added to the affected index(es).
2. The use of an index requires extra memory for page buffers. Opening the data set opens the index file, but none of the indexes. Unless an index is actually used, then no page buffers are allocated. The number of buffers allocated for a given index depends upon the number of levels in the index tree and the data set open mode. The maximum number of buffers needed per accessed index is three for an open mode of input, and four for an open mode of update.
3. Extra disk space on a separate file is required to store the index's data structures. A rough approximation to the amount of disk space required can be calculated by plugging values for PSIZE, VSIZE, UVAL, NOBS, and WASTE into the following equations:

```
Let PSIZE   = the index page size
    VSIZE   = indexed variable's size
    UVAL    = number of unique values for the
              variable
    NOBS    = total number of occurrences of
              the variable
              (i.e. number of records in the
              data set)
              For a UNIQUE index, NOBS = UVAL.
    LPAGES  = number of leaf
              (i.e. bottom-level) index
              pages
    UPAGES  = number of upper-level index
              pages
    MAXENT  = maximun number of entries on an
              uppper-level page
    HSIZE   = 20 bytes for page header (on
              most 32 bit machines)
    RSIZE   = 8 bytes for record identifier
              (on most 32 bit machines)
    OFF     = 2 bytes for each offset if the
              index is not unique (on most
              32 bit machines)
            = 0 if the index is unique
    PGNO    = 4 bytes for storage of page
              number (on most 32 bit
              machines)
    WASTE   = 0.10 if the variable's values
              in the data set are mostly
              in ascending order
            = 0.30 if the variable's values
              in the data set are randomly
              ordered
            = 0.50 if the variable's values
              in the data set are mostly
              in descending order

    LPAGES = ( ( (NOBS * (RSIZE + OFFSET)) +
             (UVAL * (VSIZE + OFFSET)) +
             (PSIZE - HSIZE - 1) ) * (1.0 +
             WASTE) ) / (PSIZE - HSIZE);

    MAXENT = (PSIZE - HSIZE) /  (OFF +
             VSIZE + PGNO);
    LOLEV  = LPAGES;

UPAGES = 0;
       LEVELS = 0;
       WHILE ( LOLEV > 1 )
          {
          CURLEV = (LOLEV + MAXENT - 1 ) /
            MAXENT;
          UPAGES = UPAGES + CURLEV;
          LOLEV  = CURLEV;
          LEVELS = LEVELS + 1;
          }
```

The total number of pages for the index is UPAGES + LPAGES. Note that this number does not include the page(s) for the index directory. For most applications there will only be one directory page for the index file.

This formula was used to estimate the size of the index file for an index on the Usage Notes variable PROD. Using PSIZE=6144, VSIZE=8, UVAL=23, NOBS=6176, and WASTE=.30 produces UPAGES=1 and LPAGES=14. Adding UPAGES, LPAGES, and 1 for the directory page gives a total of 16 pages, the actual index file size.

4. It is not always possible for the base engine's where clause processor to determine with great accuracy when to use an index for optimization or when to use a sequential pass of the data set. This is the "small subset" problem mentioned above.

   The minimum and maximum values, the number of records, and the assumption of a uniform value distribution over the minimum and maximum range is used to estimate how many records a particular where clause condition (for example, AGE > 35) will select. If the value distribution is not uniform, or the minimum and/or maximum values are skewed significantly from the rest of the values, then the cost estimate will not be very accurate and a less efficient access method may be chosen.
5. Any procedure or data step that makes a copy or a new version of a data set with indexes must rebuild the indexes for the new data set. This will consume extra resources.

   For example, given data set A with indexes, "DATA A; SET A;" will create a new version of A without indexes. And when the old version of A is deleted, the indexes are also deleted. The indexes, if desired, must be recreated (for example, via PROC DATASETS) for the new version of A.
6. Since the index(es) for a data set are stored in a separate file, any manipulation of files without the use of SAS Software must

consider the optional index file.

7. An index defined with the NOMISS option may not be able to be used for optimization if the operation expects to include missing values.

Effects of the INDEX Option

Base: no indexes New: one index
Effects are New/Base

PROGRAM	Change in CPU Time	Change in ELAPSED Time	Change in I/O (EXCP count)
ADD	6.8	11.	15.
REPORT	.68	.58	.67
BATCH	2.7	3.7	4.3
FSED1	1.0	.71	1.2
FSED2	.79	.22	.11
FSED3	.69	.35	.19
FSED4	.0065	.0014	.0032
FSED5	1.1	1.8	2.3
FSED6	1.0	1.3	1.1

All programs retrieving data using a where clause (REPORT, FSED2, FSED3, and FSED4) showed improved performance due to optimization via the index. FSED4 searches for a value not in the data set. Without an index, the entire data set (1256 pages) must be searched, but the index can determine that the value does not exist very quickly. Thus FSED4 with an index is 300 times cheaper than FSED4 without an index. Updates to the indexed data set, however, are more costly. In addition, extra mass storage is required for the 16-page index file.

Guidelines for Using Indexes to Improve Performance

1. Keep the number of indexes created to a minimum to reduce disk storage and to reduce index update costs. If the data set is to be updated frequently, the cost to update many indexes may be significant.
2. The where clause cost estimation routine is most accurate if the indexed variable's values are uniformly distributed and the minimum and maximum values are consistent with the rest of the values.
3. Don't create an index unless the data set is at least three pages long. It's faster to access the data set sequentially.
4. Don't use the NOMISS attribute if you expect the index to be used to optimize where clauses that will select missing values.
5. An index performs best when it is requested to retrieve a small number of records. "Small" is difficult to quantify, but an upper limit is surely no more than half of the records in the data set.

TOOLS FOR EXPERIMENTATION

As mentioned in the beginning, optimal performance for an application requires some experimentation with the various tuning options. This section identifies some facilities available for the collection of data used to assist in the evaluation of options.

A. PROC CONTENTS provides information about the physical characteristics of a data set that are useful when trying to select an optimal page size. Important values are:

"Observations"	-number of non-deleted records in the data set
"Observation length"	-record size in bytes
"Compressed"	-"NO" records are not compressed -"YES" records are compressed
"Data Set Page Size"	-size of the data set pages
"Number of Data Set Pages"	-total number of pages in the data set
"First Data Page"	-page number of the page containing the first data record. Header records are stored in front of the data records
"Max Obs per Page"	-maximum number of data records that a page can hold
"Obs in First Data Page"	-number of data records in the first data page
"Index File Page Size"	-size of the index file pages, if there is an index
"Number of Index File Pages"	-total number of pages in the index file

With this information, you can determine how close to optimal the default page size is and experiment with various BUFSIZE values to improve it. Note that "First Data Page," "Max Obs per Page," and "Obs in First Page" are only provided by PROC CONTENTS for an un-compressed record data set. These values have little meaning for a data set with compressed records since each record may be a different size.

Below is sample output from PROC CONTENTS for an un-compressed data set.

```
                    CONTENTS PROCEDURE

Data Set Name: TEST.TEST2                Observations:          9
Member Type:   DATA                      Variables:             5
Engine:        V606                      Indexes:               3
Created:       24FEB89:17:16:54          Observation Length:    37
Last Modified: 24FEB89:17:17:02          Deleted Observations:  0
Data Set Type:                           Compressed:            NO
Label:

     -----Alphabetic List of Variables and Attributes-----

                #    Variable    Type    Len    Pos
                ----------------------------------
                3    AGE         Num       8     16
                4    DOB         Char      8     21
                2    FNAME       Char      5      8
                1    LNAME       Char      8      0
                5    SALARY      Num       8     29

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

                          Num of
                Unique    Unique
#    Index      Option    Values    Var1     Var2     Var3
-------------------------------------------------------------
1    COMPOI     YES            9    FNAME    LNAME    DOB
2    LNAME      YES            9
3    SALARY                    7

     -----Engine/Host Dependent Information-----

          Data Set Page Size:          4096
          Number of Data Set Pages:    1
          First Data Page:             1
          Max Obs per Page:            83
          Obs in First Data Page:      9
          Index File Page Size:        4096
          Number of Index File Pages:  4
```

B. The measurement of computer resources utilized is more difficult and very host-specific. CPU time and elapsed time can be measured by using the following statement:

```
OPTIONS STIMER;
```

This option will cause CPU time and elapsed time used to be written to the log after the execution of each procedure. The format and content of the messages is host-dependent and is similar to the following IBM MVS example:

```
NOTE:  The PROCEDURE CONTENTS printed pages 1-2.
NOTE:  PROCEDURE CONTENTS used the following resources:
       CPU time      - 00:00:00.10
       Elapsed time  - 00:00:01:15
       EXCP count    - 51
       Task memory   - 266K (24K data, 242K program)
```

C. The systems analyst at your site may have better tools for measuring performance so that you can determine the most effective use of the options described in this paper.

CONCLUSION

New Version 6 SAS features and options provide a means for the SAS programmer to tune applications for improved performance. The guidelines presented must be viewed as general statements of use and not guarantees of better performance. Since many factors affect performance, experimenting with these options is the best way to determine how your application will benefit.

APPENDIX: Actual Performance Data

---------- PROGRAM=ADD ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:00.73	0:09.28	413
BUFNO	0:00.76	0:08.90	413
BUFSIZE	0:00.66	0:07.76	315
COMPRESS	0:02.18	0:08.68	294
INDEX	0:04.98	1:43.67	6344

---------- PROGRAM=REPORT ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:01.45	0:27.56	1280
BUFNO	0:01.51	0:27.53	1281
BUFSIZE	0:01.04	0:17.15	658
COMPRESS	0:04.06	0:07.97	364
INDEX	0:00.98	0:16.07	858

---------- PROGRAM=BATCH ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:06.02	1:14.19	3850
BUFNO	0:06.12	1:10.99	3849
BUFSIZE	0:04.65	0:50.30	1983
COMPRESS	0:20.28	0:38.31	1101
INDEX	0:16.06	4:37.21	16670

---------- PROGRAM=FSED1 ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:02.75	0:10.97	21
BUFNO	0:02.76	0:09.45	11
BUFSIZE	0:02.74	0:07.56	11
COMPRESS	0:02.75	0:11.28	8
INDEX	0:02.75	0:07.75	24

---------- PROGRAM=FSED2 ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:03.87	0:23.96	937
BUFNO	0:03.93	0:23.93	933
BUFSIZE	0:03.64	0:16.62	498
COMPRESS	0:05.81	0:11.93	287
INDEX	0:03.04	0:05.24	101

---------- PROGRAM=FSED3 ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:04.41	0:30.47	1309
BUFNO	0:04.47	0:30.72	1309
BUFSIZE	0:04.06	0:20.25	685
COMPRESS	0:07.06	0:12.59	390
INDEX	0:03.04	0:10.80	249

---------- PROGRAM=FSED4 ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:03.10	0:54.40	2537
BUFNO	0:03.25	0:55.04	2537
BUFSIZE	0:02.44	0:38.72	1294
COMPRESS	0:08.51	0:15.72	706
INDEX	0:00.02	0:00.08	8

---------- PROGRAM=FSED5 ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:01.61	0:02.68	59
BUFNO	0:01.62	0:03.18	57
BUFSIZE	0:01.59	0:01.98	34
COMPRESS	0:01.73	0:02.44	23
INDEX	0:01.70	0:04.77	133

---------- PROGRAM=FSED6 ----------

NAME	CPU Time	Elapsed Time	I/O (EXCP) Count
BASE	0:01.38	0:02.17	59
BUFNO	0:01.39	0:02.30	58
BUFSIZE	0:01.37	0:01.75	34
COMPRESS	0:01.50	0:01.75	23
INDEX	0:01.39	0:02.89	63
