CHAPTER 28

SAS Data Files

Definition of a SAS Data File

SAS data file
is a type of SAS data set that contains both the data values and the descriptor information. SAS data files are of the type DATA.

Note: In the SAS System, the term “data set” is used to refer to both SAS data files, which contain data and data set descriptor information, and to SAS data views, which consist entirely of descriptor information. △

native SAS data file
stores the data values and descriptor information in a file formatted by SAS.

interface SAS data file
stores the data in a file that was formatted by software other than SAS. Beginning with Release 6.06, there are engines for reading and writing data from files that were formatted by software such as ORACLE, DB2, SYBASE, ODBC, BMDP, SPSS, and OSIRIS. These files are interface SAS data files, and when their data values are accessed through an engine, SAS recognizes them as SAS data sets.

Note: The availability of engines that can access different types of interface data files is determined by your site licensing agreement. See your system administrator to determine which engines are available. For more information about SAS multi-engine architecture, see Chapter 36, "SAS I/O Engines," on page 511. △

Differences between Data Files and Data Views

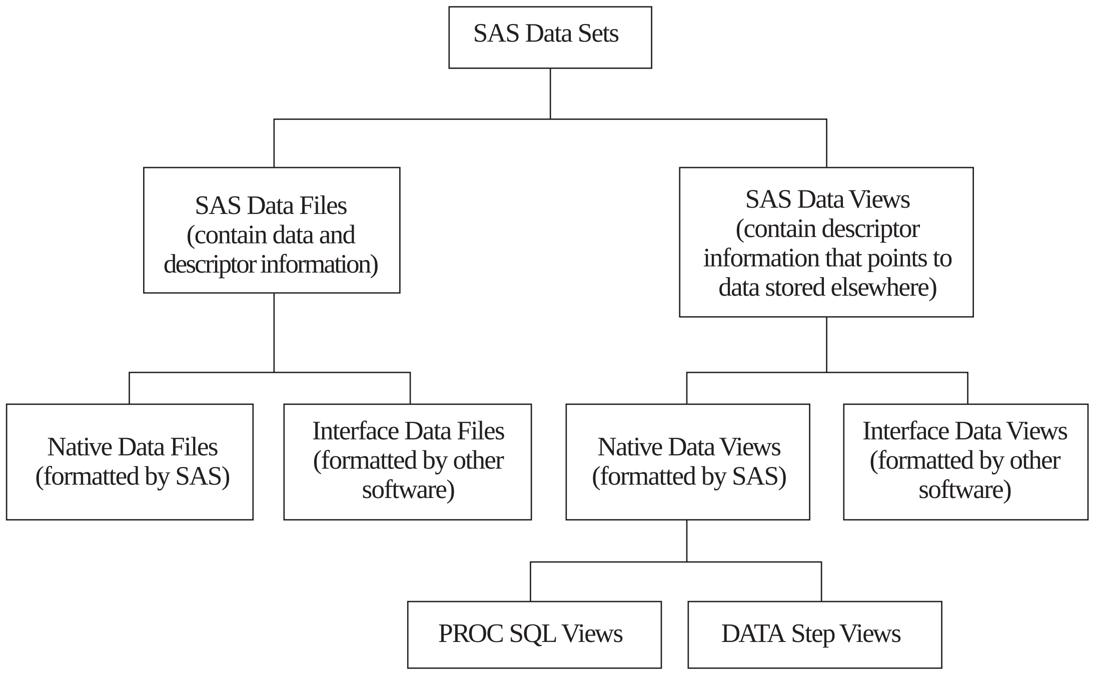
While SAS data files and SAS data views can, for the most part, be used interchangeably in a SAS DATA step, here are a few differences to keep in mind:

- *The main difference is where the values are stored.* A SAS data file is a type of SAS data set that contains both descriptor information about the data and the data values themselves. SAS data views contain only descriptor information that points to data values that are stored elsewhere.
- *A data file is a static picture; a data view is a dynamic picture.* When you reference a data file in a later PROC step, you see the data values as they were when the data file was created or last updated. When you reference a data view in a PROC step, the view executes and provides you with an image of the data values as they currently exist, not as they existed when the view was defined.
- *SAS data files can be created on tape, or on any other storage medium.*

  SAS data views cannot be created or stored on tape, or generated from data files stored on tape. Because of their dynamic nature, SAS data views must derive their information from data files on random-access storage devices, such as disk drives. SAS data views cannot derive their information from files stored on sequentially accessed storage devices, such as tape drives.
- *SAS data views are read-only.* You cannot write to a data view.
- *SAS data files can have integrity constraints.* When you update a SAS data file, you can ensure that the data conforms to certain standards by using integrity constraints. With data views, this may only be done indirectly, by assigning integrity constraints to the data files that the data views reference.
- *SAS data files can be indexed.* Indexing may allow SAS to find data in a SAS data file more quickly. SAS data views cannot be indexed.
- *SAS data files can be encrypted.* Encryption provides an extra layer of security to physical files. SAS data views cannot be encrypted.
- *SAS data files can be compressed.* Compression makes it possible to store physical files in less space. SAS data views cannot be compressed.

The following table illustrates native and interface SAS data files and their relationship to SAS data views.

Figure 28.1 Types of SAS Data Sets


SAS Data Sets
SAS Data Files (contain data and descriptor information)
SAS Data Views (contain descriptor information that points to data stored elsewhere)
Native Data Files (formatted by SAS)
Interface Data Files (formatted by other software)
Native Data Views (formatted by SAS)
Interface Data Views (formatted by other software)
PROC SQL Views
DATA Step Views

Audit Trail

Definition of an Audit Trail

The audit trail is an optional SAS file that you can create to log modifications to a SAS data file. Each time an observation is added, deleted, or updated, information is written to the audit trail about who made the modification, what was modified, and when.

Benefits of an Audit Trail

Many businesses and organizations require an audit trail for security reasons. The audit trail maintains a historical record of the data that enables you to trace a piece of data from the moment it enters the data file to the time it leaves.

An audit trail provides useful information from which to develop usage statistics. For example, for master data files that are updated by multiple applications and users, the audit trail can show which applications and users made updates and what updates were made.

The audit trail is also the only place in the SAS System that stores observations from failed appends and observations that were rejected by integrity constraints. The integrity constraints feature is described in "Integrity Constraints" on page 423. You can write a DATA step to extract the failed or rejected observations from the audit trail, use information describing why they failed to correct them, and then reapply the observations to the data file.

Audit Trail Description

The audit trail is a SAS file created by the SAS base engine with the same libref and member name as the data file, and a data set type of AUDIT. The audit trail replicates the variables in the data file and additionally stores two types of audit variables:

- _AT*_ variables, which automatically store modification data
- "user" variables, which are special variables you can optionally define when you initiate the audit trail.

The _AT*_ variables are described in the following table.

Table 28.1 _AT* Variables

_AT*_ Variable	Description
ATDATETIME	Stores the date and time of a modification
ATUSERID	Stores the logon userid associated with a modification
ATOBSNO	Stores the observation number affected by the modification, except when REUSE=YES (because the observation number is always 0)
ATRETURNCODE	Stores the event return code
ATMESSAGE	Stores the SAS log message at the time of the modification
ATOPCODE	Stores a code describing the type of modification

The _ATOPCODE_ values are listed in the following table.

Table 28.2 _ATOPCODE_ Values

Code	Modification
DA	Added data record image
DD	Deleted data record image
DR	Before-update record image
DW	After-update record image
EA	Observation add failed
ED	Observation delete failed
EW	Observation update failed

The log settings at audit trail initiation determine which _ATOPCODE_ values are logged:

- the "DR" operation code is controlled with the LOG statement BEFORE_IMAGE option
- other operations codes that begin with a "D" are controlled with the DATA_IMAGE option
- operation codes that begin with an "E" are controlled with the ERROR_IMAGE option.

For instructions on specifying log settings, refer to “Initiating an Audit Trail” on page 417. The default behavior is to log all images.

The user variables are unique in the SAS System because they are stored in one file (the audit file) and opened for update in another file, the data file. This enables you to associate data values with the data file without making them part of the data file. For example, you could define a user variable that enables users to enter a “reason for the modification.”

The user variables are processed as follows:

1 You define the variables as part of the audit trail specification.
2 The base engine retrieves the variables from the audit trail and displays them when the data file is opened for update.
3 The users can enter data values for the user variables as they would for any data variable.
4 The data values are written to the audit trail as each observation is saved. In applications like FSEDIT, which save observations as you scroll through them, it may appear that the data values have disappeared.
5 The user variables are not available when the data file is opened for browsing or printing.
6 You modify user variables in the data file. That is, to rename a user variable or modify its attributes, you modify the data file, not the audit file.

For information about defining user variables, see “Defining User Variables” on page 418. If you define user variables, you must store values in them for the variables to be meaningful.

The audit trail must reside in the same SAS library as its associated data file, and a data file can have only one audit file.

Operation

The audit trail operates similarly in local and remote environments. The only difference for applications and users networked with SAS/CONNECT and SAS/SHARE is that the audit trail logs events when the observation is written to permanent storage; that is, when the data is written to the remote SAS session or server. Therefore, the time the transaction is logged may be different than the user’s SAS session.

Performance

Because each update to the data file is also written to the audit file, the audit trail can negatively impact system performance. You may want to consider suspending the audit trail for large, regularly scheduled batch updates. Note that the audit variables are unavailable when the audit trail is suspended.

Reading and Determining the Status of the Audit Trail

The audit trail is read-only. You can read the audit trail with any component of SAS that reads a data set. To refer to the audit trail, use the data set TYPE= option. For example, to print the audit trail, you would issue the statement:

```
proc print data=libref.member-name (type=audit);
  title "Data in the Audit File";
run;
```

If an audit trail exists, PROC CONTENTS reports the audit status and records image settings when it is invoked on its associated data file. You can also use your favorite reporting tool — PROC REPORT or PROC TABULATE, for example — on the audit trail.

Limitations

The audit trail is not recommended for SAS data files that are copied, moved, sorted in place, replaced, or transferred to another operating system because those operations do not preserve the audit trail. In a copy operation on the same host, you can preserve the data file and audit trail by renaming them using the Generation Data Sets feature; however, logging will stop because neither the auditing process nor the Generation Data Sets feature saves the source program that caused the replacement. For more information, see "Generation Data Sets" on page 404.

For data files whose audit file contains user variables, the variable list is different when browsing and updating the data file. The user variables are selected for update but not for browsing. You should be aware of this difference when you are developing your own full-screen applications.

Data values entered for user variables are not stored in the audit trail for delete operations.

If the audit file becomes damaged, you will not be able to process the data file until you terminate the audit trail. Then you can initiate a new audit trail or process the data file without one.

The Audit Trail and Fast-Append

In indexed data sets, the fast-append feature may cause some observations to be written to the audit trail twice, first with a DA operation code and then with an EA operation code. The observations with EA represent those rejected by index restrictions. For more information, see "Appending to an Indexed Data Set" in the PROC DATASETS APPEND statement documentation in the *SAS Procedures Guide*.

Initiating an Audit Trail

You initiate the audit trail in PROC DATASETS with the AUDIT statement. The syntax for initiating the audit trail is:

PROC DATASETS LIB=*libref*;

  AUDIT *SAS-file* <*SAS-password*>;

    INITIATE;

    <**LOG** <BEFORE_IMAGE=YES | NO><DATA_IMAGE=YES | NO>
        <ERROR_IMAGE=YES | NO>>;

    USER_VAR *specification-1* <*...specification-n*>;

where:

SAS-file specifies the SAS data file in the procedure input library that you want to audit.

SAS-password is the SAS password of the data file, if one exists.

The INITIATE statement creates the audit trail.

The LOG statement specifies the data images, or events, to be logged on the audit trail.

BEFORE_IMAGE=YES | NO controls storage of before-update record images.

DATA_IMAGE=YES | NO controls storage of after-update record images.

ERROR_IMAGE=YES | NO controls storage of unsuccessful update record images.

If the LOG statement is omitted, the default setting for all images is YES.

The USER_VAR statement optionally defines user variables to be logged to the audit trail with each update to an observation. Syntax details are provided in “Defining User Variables” on page 418.

The audit file will use the SAS password assigned to the associated data file, and therefore it is recommended that the data file have an ALTER password. An ALTER-level password restricts read and edit access to SAS files. If a password other than ALTER is used, or no password is used, the software will generate a warning message that the files are not protected from accidental update or deletion.

Defining User Variables

You define user variables at audit trail initiation with the USER_VAR statement. The syntax for the USER_VAR statement is:

USER_VAR= *variable-name* <$><*length*><LABEL= *"variable-label"*>
<*...variable-name-n* <$><*length*><LABEL= *"variable-label"*>>;

where:

variable-name is a name for the user variable.

$ indicates the variable is a character value. If $ is not specified, the default is numeric.

length specifies the length of the variable. If a length is not specified, the default is 8 characters.

LABEL="variable-label" specifies a label for the variable.

You can define attributes such as format and informat in the data file with PROC DATASETS.

Controlling the Audit Trail

Once the audit trail is established, you can change which record images are logged, suspend and resume logging, and terminate (delete) the audit file. The syntax for controlling the audit trail is:

PROC DATASETS LIB= *libref*;
AUDIT *SAS-file* <*SAS-password*>;
LOG | **SUSPEND** | **RESUME** | **TERMINATE**;

Replacing the associated data file will also delete the audit trail.

Example of Initiating an Audit Trail

The following example creates and initiates an audit trail for data file MYLIB.SALES, which stores fictional invoice and renewal figures for SAS products. The audit trail will record all events and store one user variable, REASON_CODE, for users to enter a reason for the update.

Subsequent examples will illustrate the affect of a data file update on the audit trail and how to use audit variables to capture observations that are rejected by integrity constraints. The system option LINESIZE is set in advance for the integrity constraints example. A large LINESIZE value is recommended to display the content of the _ATMESSAGE_ variable. The output examples have been modified to fit on the page.

```
options linesize=250;
   /*------------------------------------*/
   /* Create SALES data set.             */
   /*------------------------------------*/

data mylib.sales;
  length product  $9;
  input product invoice renewal;
cards;
FSP        1270.00        570
SAS        1650.00        850
STAT       570.00         0
STAT       970.82         600
OR         239.36         0
SAS        7478.71        1100
SAS        800.00         800
;


   /*----------------------------------*/
   /* Create an audit trail with a     */
   /* user variable.                   */
   /*----------------------------------*/

proc datasets lib=mylib;
  audit sales;
    initiate;
    user_var reason_code $ 20;
run;

   /*-------------------------------------*/
   /* Issue proc contents to view the     */
   /* audit file.                         */
   /* -----------------------------------*/
proc contents data=mylib.sales (type=audit); run;
```

Output 28.1 PROC CONTENTS of MYLIB.SALES

```
                             The CONTENTS Procedure

Data Set Name: MYLIB.SALES                                 Observations:         0
Member Type:   AUDIT                                       Variables:           10
Engine:        V8                                          Indexes:              0
Created:       10:51 Thursday, September 30, 1999         Observation Length: 111
Last Modified: 10:51 Thursday, September 30, 1999         Deleted Observations: 0
Protection:                                                Compressed:          NO
Data Set Type: AUDIT                                       Sorted:              NO
Label:

...                                The CONTENTS Procedure
                  -----Alphabetic List of Variables and Attributes-----
                 #    Variable          Type    Len    Pos    Format
                 ------------------------------------------------------
                5    _ATDATETIME_      Num       8     45    DATETIME.
               10    _ATMESSAGE_       Char      8    103
                6    _ATOBSNO_         Num       8     53
                9    _ATOPCODE_        Char      2    101
                7    _ATRETURNCODE_    Num       8     61
                8    _ATUSERID_        Char     32     69
                2    invoice           Num       8      0
                1    product           Char      9     16
                4    reason_code       Char     20     25
                3    renewal           Num       8      8
```

Example of a Data File Update

The following example inserts an observation into MYLIB.SALES.DATA and prints the update data in the MYLIB.SALES.AUDIT.

```
    /*------------------------------------*/
    /* Do an update.                      */
    /*------------------------------------*/
  proc sql;
    insert into mylib.sales
        set product = 'AUDIT',
            invoice = 2000,
            renewal = 970,
        reason_code = "Add new product";
 quit;

    /*------------------------------------------*/
    /* Print the audit trail. */
    /*------------------------------------------*/
 proc sql;
   select product,
          reason_code,
          _atopcode_,
          _atuserid_ format=$6.,
          _atdatetime_
          from mylib.sales(type=audit);
 quit;
```

Output 28.2 Updated Data in MYLIB.SALES.AUDIT

```
product    reason_code          _ATOPCODE_  _ATUSERID_    _ATDATETIME_
--------------------------------------------------------------------------
AUDIT      Add new product      DA          xxxxxx        30SEP99:10:30:18
```

Example of Using the Audit Trail to Capture Rejected Observations

The following example adds integrity constraints to MYLIB.SALES.DATA and records observations that are rejected as a result of the integrity constraints in MYLIB.SALES.AUDIT.

```
   /*----------------------------------*/
   /* Create integrity constraints.    */
   /*----------------------------------*/
proc datasets lib=mylib;
   modify sales;
   ic create null_renewal = not null (invoice)
             message = "Invoice must have a value.";
   ic create invoice_amt = check (where=((invoice > 0) and
               (renewal <= invoice)))
             message = "Invoice and/or renewal are invalid.";
run;

   /*----------------------------------*/
   /* Do some updates.                 */
   /*----------------------------------*/
 proc sql; /* this update works */
    update mylib.sales
      set invoice = invoice * .9,
      reason_code = "10% price cut"
      where renewal > 800;

 proc sql;  /* this update fails */
    insert into mylib.sales
       set product = 'AUDIT',
           renewal = 970,
       reason_code = "Add new product";

 proc sql;  /* this update works */
    insert into mylib.sales
       set product = 'AUDIT',
           invoice = 10000,
           renewal = 970,
       reason_code = "Add new product";

proc sql;  /* this update fails */
    insert into mylib.sales
       set product = 'AUDIT',
           invoice = 100,
           renewal = 970,
       reason_code = "Add new product";
 quit;

   /*----------------------------------------*/
   /* Print the audit trail. */
   /*----------------------------------------*/
proc print data=mylib.sales(type=audit);
  format _atuserid_ $6.;
  var product reason_code _atopcode_ _atuserid_ _atdatetime_;
title  'Contents of the Audit Trail';
run;

   /*----------------------------------------*/
   /* Print the rejected records.            */
   /*----------------------------------------*/
proc print data=mylib.sales(type=audit);
  where _atopcode_ eq "EA";
  format _atmessage_ $250.;
  var product invoice renewal _atmessage_ ;
title  'Rejected Records';
run;
```

Output 28.3 on page 422 shows the contents of MYLIB.SALES.AUDIT after several updates of MYLIB.SALES.DATA were attempted. Integrity constraints were added to the file, then updates were attempted. Output 28.4 on page 422 prints information about the rejected observations on the audit trail.

Output 28.3 Contents of MYLIB.SALES.AUDIT after an Update with Integrity Constraints

```
                       Contents of the Audit Trail

 Obs    product      reason_code      _ATOPCODE_    _ATUSERID_      _ATDATETIME_
  1     AUDIT     Add new product        DA         xxxxxx      30SEP99:10:30:18
  2     AUDIT     Add new product        DA         xxxxxx      30SEP99:10:32:00
  3     SAS                              DR         xxxxxx      30SEP99:10:46:26
  4     SAS       10% price cut          DW         xxxxxx      30SEP99:10:46:26
  5     SAS                              DR         xxxxxx      30SEP99:10:46:26
  6     SAS       10% price cut          DW         xxxxxx      30SEP99:10:46:26
  7     AUDIT                            DR         xxxxxx      30SEP99:10:46:26
  8     AUDIT     10% price cut          DW         xxxxxx      30SEP99:10:46:26
  9     AUDIT                            DR         xxxxxx      30SEP99:10:46:26
 10     AUDIT     10% price cut          DW         xxxxxx      30SEP99:10:46:26
 11     AUDIT     Add new product        EA         xxxxxx      30SEP99:10:46:32
 12     AUDIT     Add new product        EA         xxxxxx      30SEP99:10:46:38
 13     AUDIT     Add new product        DA         xxxxxx      30SEP99:10:46:44
```

Output 28.4 Rejected Records on the Audit Trail

```
                      Rejected Records
Obs    product    invoice    renewal    _ATMESSAGE_
1      AUDIT         .        970      ERROR: Invoice must have a value. Add/Update
                                       failed for data set MYLIB.SALES because data
                                       value(s) do not comply with integrity constraint
                                       null_renewal.
2      AUDIT      100         970      ERROR: Invoice and/or renewal are invalid.
                                       Add/update failed for data set MYLIB.SALES
                                       because data value(s) do not comply with
                                       integrity constraint invoice_amt.
```

Integrity Constraints

Definition of Integrity Constraints

Integrity constraints are a set of data validation rules that you can specify to restrict the data values accepted into a SAS data file. Using integrity constraints can preserve the correctness and consistency of stored data. SAS enforces the integrity constraints each time data is changed or deleted in a variable that has integrity constraints assigned to it.

There are two categories of integrity constraints:

- General constraints, which allow you to restrict the data values that are accepted for the variables in a single data file, such as requiring that the data values for a variable be unique and/or nonmissing, or making the data values in one variable contingent on the data values in another variable.
- Referential constraints, which allow you to link the data values of the variables in one data file to specific variables in another data file. An example of a referential constraint would be linking the values for an employee name variable in a Personnel data file to a similar variable in a Payroll data file and to an Employee Bonuses data file. Only the names of employees that exist in the Personnel data file would be allowed in the Payroll and Employee Bonuses data files.

Note: In SAS, the term “data set” is used to refer to both SAS files, which contain data and data set descriptor information, and to SAS data views, which consist entirely of descriptor information. Because they are associated with stored data, integrity constraints can only be defined in SAS data files. △

General Integrity Constraints

There are four types of general integrity constraints:

Check	limits the data values in a variable to a specific set, range, or list. This constraint can also be used to make the data values in one variable contingent on the data values in another variable.
Not Null	requires that a variable contain a data value. Missing values for character and numeric data are not allowed.
Unique	requires that the specified variables contain unique data values.
Primary Key	requires that the specified variables contain unique data values and that missing or null data values are not allowed. A data file can have only one primary key.

Referential Integrity Constraints

A referential integrity constraint is created when a primary key integrity constraint in one data file is referenced by a foreign key integrity constraint in another data file.

A foreign key integrity constraint links the data values of one or more variables in its data file to those of the variables specified in a primary key, and controls the action that can be taken when an attempt is made to update or delete the data values in the primary key. The following referential actions can be specified:

RESTRICT	prevents the data values in the primary key from being updated or deleted unless there is no matching value in any referencing foreign key variables. This is the default if no referential action is specified.

NULL	allows primary variables to be updated or deleted, but changes any affected foreign key values to a missing value.

For example:

```
proc sql;
  create table one
    (
     name   char(14),
     CONSTRAINT prim_key  primary key(name)
     );

proc sql;
   create table two
   (
    lname   char(14),
    CONSTRAINT for_key  foreign key(lname) references one
      on delete restrict on update set null
    );
```

The preceding example creates a referential integrity constraint between variable Name in table ONE and variable Lname in table TWO. As the primary key, variable Name will define the acceptable data values for variable Lname. In addition, the foreign key specifies that data values will not be deleted from variable Name unless no matching values exist in variable Lname, and updates will cause affected data values in Lname to be changed to a missing value. The primary key integrity constraint also cannot be deleted until this and any other foreign key integrity constraint that references it has been deleted. There are no restrictions on deleting foreign key constraints.

The following rules must be met for a referential relationship to be established:

- The primary key and foreign key specifications must reference the variables in the same order.
- The variables must be of the same type (character or numeric) and length.
- If the referential integrity constraint is being added to existing variables, the data values in the foreign key must match the values in the primary key or be null. For example, using the example above, if primary key variable Name contained the data values shown below, then foreign key variable Lname could have any of the data values shown below except those in column 4.

Table 28.3 Potential Foreign Key Data Values for Variable "lname"

Data Values in Primary Key name	1	2	3	4
Davis, Jan	Smith, Mike	Davis, Jan	.	Davis, Jan
Smith, Mike	Davis, Jan	Smith, Mike	.	Smith, Mike
		Smith, Mike	.	Johnson, Ed

. = missing value

Note that the variable names in the primary key and foreign key specification can match.

A referential integrity constraint can exist between data files in the same or different SAS libraries with these restrictions:

- If the library of a data set containing a foreign key integrity constraint is temporary, then the library containing the primary key data set must be temporary as well.
- Referential integrity constraints cannot be assigned to data sets in concatenated libraries.

Preservation of Integrity Constraints

These procedures preserve integrity constraints when their operation results in a copy of the original data file:

- in base SAS software, the COPY, CPORT, CIMPORT and SORT procedures
- in SAS/CONNECT software, the UPLOAD and DOWNLOAD procedures
- PROC APPEND, when a DATA= data file does not exist
- PROC SORT and PROCs UPLOAD and DOWNLOAD, when an OUT= data file is not specified.

You can use the CONSTRAINT option to control when integrity constraints are preserved for the COPY, CPORT, and CIMPORT procedures, which always result in a copy, and additionally for the UPLOAD and DOWNLOAD procedures.

Several factors affect which integrity constraints are preserved:

- the nature of the procedure
- whether the procedure is performed on a data file or a library
- for referential integrity constraints, whether the integrity constraint exists between data files in the same or different libraries (intra-libref versus inter-libref integrity constraints).

  Inter-libref referential integrity constraints are preserved in an inactive state. That is, the primary key portion of the integrity constraint is enforced as a general integrity constraint but the foreign key portion is inactive. You must use the DATASETS procedure statement INTEGRITY CONSTRAINT REACTIVATE to reactivate the inactive foreign key constraint.

The following table summarizes the circumstances under which integrity constraints are preserved.

Table 28.4 Circumstances under Which Integrity Constraints are Preserved

Procedure	**Condition**	**Integrity Constraints Preserved in Data Sets**	**Integrity Constraints Preserved in Libraries**
APPEND	DATA= data set does not exist	General	Not applicable
COPY	CONSTRAINT= yes	General	General Intra-libref is referential Inter-libref is referential in an inactive state

Procedure	Condition	Integrity Constraints Preserved in Data Sets	Integrity Constraints Preserved in Libraries
CPORT/ CIMPORT	CONSTRAINT= yes	General	General Intra-libref is referential Inter-libref is referential in an inactive state
SORT	OUT= data set is not specified	General Intra-libref is referential Inter-libref is referential in active state	Not applicable
UPLOAD/ DOWNLOAD	CONSTRAINT= yes and OUT= data set is not specified	General Intra-libref is referential Inter-libref is referential in an inactive state	General Intra-libref is referential Inter-libref is referential in an inactive state

Indexes and Integrity Constraints

The unique, primary key, and foreign key integrity constraints store data values in an index file. If an index file already exists, it is used; otherwise, one is created. Consider the following points when you create or delete an integrity constraint:

- When a user-defined index exists, the index's attributes must be compatible with the integrity constraint in order for the integrity constraint to be created. For example, when adding a primary key constraint, the existing index must have the UNIQUE attribute. When adding a foreign key constraint, the index must *not* have the UNIQUE attribute.
- The unique integrity constraint has the same effect as the UNIQUE index attribute; therefore, when one is used, the other is not necessary.
- Although they might appear to be the same, the NOMISS index attribute and not null integrity constraint have different effects. The integrity constraint prevents missing values in a SAS data file and cannot be added to an existing data file with missing values. The index attribute allows missing data values in the data file but excludes them from the index.
- When any index is created, it is marked as being "owned" by the user and/or by the integrity constraint. A user cannot remove an index owned by an integrity constraint and an integrity constraint cannot remove an index owned by a user. If an index is owned by both, then the index will be removed only after both the integrity constraint and the user have requested the index's removal. A note in the log indicates when an index could not be removed.

Locking

Integrity constraints support both member-level and record-level locking. You can override the default locking level with the CNTLLEV= data set option. Refer to the *SAS Language Reference: Dictionary* for more information on CNTLLEV=.

Specifying Integrity Constraints

You create integrity constraints in the SQL procedure, the DATASETS procedure, or in SCL (SAS Component Language). The constraints can be provided when the data file is created or added to an existing SAS data file. When integrity constraints are added to an existing data file, SAS verifies that the data in the variables to which integrity constraints have been assigned conform to the constraints before the integrity constraints are added.

When specifying integrity constraints, note that you must specify a separate statement for each variable that you want to have the not null integrity constraint. When multiple variables are included in the specification for a primary key, foreign key, or unique integrity constraint, a composite index is created and the integrity constraint will enforce the combination of variable values. The relationship between SAS indexes and integrity constraints is described in “Indexes and Integrity Constraints” on page 426. For more information, see “SAS Indexes” on page 433.

When adding an integrity constraint with SCL, open the data set in utility mode. See “Example 3: Creating Integrity Constraints with SCL” on page 429 for an example. Integrity constraints must be deleted in utility open mode. For detailed syntax information, see *SAS Screen Control Language: Reference*.

When generation data sets are used, you must create the integrity constraints in each data set generation that includes protected variables.

Listing Integrity Constraints

The CONTENTS and DATASETS procedures report integrity constraint information as part of normal processing. For PROC SQL, the commands DESCRIBE TABLE and DESCRIBE TABLE CONSTRAINTS report integrity constraint specifications as part of the data file definition or alone, respectively. SCL provides the ICTYPE and ICVALUE functions for getting information about integrity constraints. Refer to the appropriate documentation for syntax information.

Rejected Observations

You can customize the error message for an integrity constraint by using the MESSAGE= option of the PROC DATASETS ICCREATE statement. For more information, see the full description of the DATASETS procedure in the *SAS Procedures Guide*.

Rejected observations can be collected in a special file using the audit trail functionality.

Examples

Example 1: Creating Integrity Constraints with the DATASETS Procedure

The following sample code creates integrity constraints using the DATASETS procedure. The data file, TV_SURVEY, checks the percentage of viewing time spent on networks, PBS, and other channels, with the following integrity constraints:

- the viewership percentage cannot exceed 100 percent
- only adults can participate in the survey
- “sex” can be male or female.

```
data tv_survey(label='Validity checking');
  length idnum age 4 sex $1;
  input idnum sex age network pbs other;
  datalines;
1 M 55 80 . 20
2 F 36 50 40 10
3 M 42 20 5 75
4 F 18 30 0 70
5 F 84 0 100 0
;

proc datasets nolist;
  modify tv_survey;
    ic create val_sex = check(where=(sex in ('M','F')))
       message = "Valid values for variable SEX are
       either 'M' or 'F'.";
    ic create val_age = check(where=(age >= 18 and age <= 120))
       message = "An invalid AGE has been provided.";
    ic create val_new = check(where=(network <= 100));
    ic create val_pbs = check(where=(pbs <= 100));
    ic create val_ot = check(where=(other <= 100));
    ic create val_max = check(where=((network+pbs+other)<= 100));
quit;
```

Example 2: Creating Integrity Constraints with the SQL Procedure

The following sample program creates integrity constraints using the SQL procedure. The data file PEOPLE lists employees and contains employment information. The data file, SALARY, contains salary and bonus information. The integrity constraints are as follows:

- The names of employees receiving bonuses must be found in the PEOPLE data file.
- The names identified in the primary key must be unique.
- Gender can be male or female.
- Job status can be permanent, temporary, or terminated.

```
proc sql;
   create table people
    (
      name      char(14),
      gender    char(1),
      hired     num,
      jobtype   char(1) not null,
      status    char(10),

     constraint prim_key primary key(name),
     constraint gender check(gender in ('male' 'female')),
     constraint status check(status in ('permanent'
                             'temporary' 'terminated'))
      );

      create table salary
      (
       name     char(14),
       salary   num not null,
       bonus    num,

       constraint for_key foreign key(name) references people
          on delete restrict on update set null
      );
   quit;
```

Example 3: Creating Integrity Constraints with SCL

To add integrity constraints to a data file with SCL, you must create and build an SCL catalog entry. The following sample program creates and compiles catalog entry EXAMPLE.IC_CAT.ALLICS.SCL.

```
INIT:
  put "Test SCL integrity constraint functions start.";
return;

MAIN:
   put "Opening WORK.ONE in utility mode.";
   dsid = open('work.one', 'V');/* Utility mode.*/
   if (dsid = 0) then
     do;
      _msg_=sysmsg();
      put _msg_=;
     end;
     else do;
      if (dsid > 0) then
          put "Successfully opened WORK.ONE in"
              "UTILITY mode.";
     end;

   put "Create a check integrity constraint named teen.";
   rc = iccreate(dsid, 'teen', 'check',
   '(age > 12) && (age < 20)');

   if (rc > 0) then
      do;
       put rc=;
       _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a check"
        "integrity constraint.";
   end;

 put "Create a not-null integrity constraint named nn.";
 rc = iccreate(dsid, 'nn', 'not-null', 'age');

 if (rc > 0) then
    do;
     put rc=;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a not-null"
        "integrity constraint.";
   end;

 put "Create a unique integrity constraint named uq.";
 rc = iccreate(dsid, 'uq', 'unique', 'age');

 if (rc > 0) then
    do;
     put rc=;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a unique"
        "integrity constraint.";
   end;

put "Create a primary key integrity constraint named pk.";
rc = iccreate(dsid, 'pk', 'Primary', 'name');

 if (rc > 0) then
    do;
     put rc=;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a primary key"
        "integrity constraint.";
   end;

 put "Closing WORK.ONE.";
 rc = close(dsid);
 if (rc > 0) then
   do;
    put rc=;
```

```
         _msg_=sysmsg();
         put _msg_=;
        end;

      put "Opening WORK.TWO in utility mode.";
      dsid2 = open('work.two', 'V');
         /*Utility mode */
      if (dsid2 = 0) then
        do;
       _msg_=sysmsg();
         put _msg_=;
        end;
        else do;
         if (dsid2 > 0) then
           put "Successfully opened WORK.TWO in"
               "UTILITY mode.";
        end;

      put "Create a foreign key integrity constraint named fk.";
      rc = iccreate(dsid2, 'fk', 'foreign', 'name',
      'work.one','null', 'restrict');

      if (rc > 0) then
         do;
          put rc=;
          _msg_=sysmsg();
          put _msg_=;
        end;
        else do;
         put "Successfully created a foreign key"
             "integrity constraint.";
        end;

    put "Closing WORK.TWO.";
      rc = close(dsid2);
      if (rc > 0) then
        do;
         put rc=;
         _msg_=sysmsg();
         put _msg_=;
        end;
   return;

   TERM:
     put "End of test SCL integrity constraint"
         "functions.";
   return;
```

After creating the SCL catalog entry, the following code can be submitted to create two data files, ONE and TWO, and execute SCL entry EXAMPLE.IC_CAT.ALLICS.SCL.

```
      /* Submit to create data files. */

   data one two;
      input name $ age;
```

```
cards;
Morris 13
Elaine 14
Tina 15
run;

   /* after compiling, run the SCL program */

proc display catalog= example.ic_cat.allics.scl;
run;
```

Example 4: Removing Integrity Constraints

The following sample program segments remove integrity constraints. In those that delete a primary key integrity constraint, note that the foreign key integrity constraint is deleted first.

This program segment deletes integrity constraints using PROC SQL.

```
proc sql;
alter table salary
  DROP CONSTRAINT for_key;
alter table people
    DROP CONSTRAINT gender
    DROP CONSTRAINT _nm0001_
    DROP CONSTRAINT status
    DROP CONSTRAINT prim_key
    ;
quit;
```

This program segment removes integrity constraints using PROC DATASETS.

```
proc datasets nolist;
   modify tv_survey;
      ic delete val_max;
      ic delete val_sex;
      ic delete val_age;
run;
quit;
```

This program segment removes integrity constraints using SCL.

```
TERM:
   put "Opening WORK.TWO in utility mode.";
   dsid2 = open( 'work.two' , 'V' );  /* Utility mode.     */
   if (dsid2 = 0) then
     do;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
     if (dsid2 > 0) then
        put "Successfully opened WORK.TWO in Utility mode.";
    end;

rc = icdelete(dsid2, 'fk');
if (rc > 0) then
  do;
```

```
      put rc=;
      _msg_=sysmsg();
    end;
  else
      do;
      put "Successfully deleted a foreign key integrity constraint.";
      end;
  rc = close(dsid2);
  return;
```

Example 5: Reactivating an Inactive Integrity Constraint

The following program segment reactivates a foreign key integrity constraint that has been inactivated as a result of a COPY, CPORT, CIMPORT, UPLOAD, or DOWNLOAD procedure.

```
  proc datasets;
     modify data-set;
        ic reactivate fkname references libref;
     run;
  quit;
```

SAS Indexes

Definition of SAS Indexes

An *index* is an optional file that you can create for a SAS data file to provide direct access to specific 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 file. In other words, an index allows you to locate an observation by 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 in which they are stored in the data file. SAS reads each observation, looking for SSN=465-33-8613 until the value is found or all observations are read.
- With an index on 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.

You can either create an index when you create a data file, or create an index for an existing data file. The data file can be either compressed or uncompressed. For each data file, you can create one or multiple indexes. Once an index exists, SAS treats it as part of the data file. That is, if you add or delete observations or modify values, the index is automatically updated.

Benefits of an Index

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 an index or to read the data file sequentially.

- For BY processing, an index returns observations in the index order, which is in ascending value order, without using the SORT procedure even when the data file is not stored in that order.

  Note: If the SORT procedure is used, the index is not used. △

- 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 file.

In addition, an index can benefit other areas of the SAS System. In SCL (SAS Component Language), an index improves the performance of table lookup operations. For the SQL procedure, an index enables the software to process certain classes of queries more efficiently, for example, join queries. For the SAS/IML software, you can explicitly specify that an index be used for read, delete, list, or append operations.

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

Note: An index is never used for the subsetting IF statement in a DATA step or for the FIND and SEARCH commands in the FSEDIT procedure. △

Index File

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

The index file may show up as a separate file or appear to be part of the data file, depending on the operating environment. In any case, the index file is stored in the same SAS data library as its data file.

The index file consists of entries that are organized hierarchically and connected by pointers, all of which are maintained by SAS. The lowest level in the index file hierarchy consists of entries that represent each distinct value for an indexed variable, in ascending value order. Each entry consists of

- a distinct value
- one or more unique record identifiers (referred to as a *RID*) that identifies each observation containing the value. (Think of the RID as an internal observation number.)

That is, in an index file, each value is followed by one or more RIDs, which identifies the observation(s) in the data file containing the value. (Multiple RIDs result from multiple occurrences of the same value.) For example, the following represents index file entries for the variable LASTNAME:

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

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(s) to read the observation(s) that contain the value. Subsequent entries' higher (greater) than the requested value are found by reading the remaining entries and then following the pointers to entries that contain higher values. The result is that SAS can quickly locate the observations that are associated with a value or range of values. For example, using an index to process the WHERE expression,

```
where age > 20 and age < 35;
```

SAS positions the index to the index entry for the first value greater than 20 and uses the value's RID(s) to read the observation(s). SAS then moves sequentially through the index entries reading observations until it reaches the index entry for the value that is equal to or greater than 35.

SAS automatically keeps the index file balanced as updates are made, which means that it ensures a uniform cost to access any index entry, and all space that is occupied by deleted values is recovered and reused.

Types of Indexes

When you create an index, you designate which variable(s) to index. An indexed variable is called a *key variable*. You can create two types of indexes:

- A *simple index*, which consists of the values of one variable.
- A *composite index*, which consists of the values of more than one variable, with the values concatenated to form a single value.

In addition to deciding whether you want a simple index or a composite index, you can also limit an index (and its data file) to *unique values* and exclude from the index *missing values*.

Simple Index

The most common index is a simple index, which is an index of values for one key variable. The variable can be numeric or character. When you create a simple index, SAS assigns to the index the name of the key variable.

The following example shows the DATASETS procedure statements that are used to create two simple indexes for variables CLASS and MAJOR in data file COLLEGE.SURVEY:

```
proc datasets library=college;
   modify survey;
      index create class;
      index create major;
run;
```

To process a WHERE expression using an index, SAS uses only one index. When the WHERE expression has multiple conditions using multiple key variables, SAS determines which condition qualifies the smallest subset. For example, suppose that COLLEGE.SURVEY contains the following data:

- 42,000 observations contain CLASS=97.
- 6,000 observations contain MAJOR='Biology'.
- 350 observations contain both CLASS=97 and MAJOR='Biology'.

With simple indexes on CLASS and MAJOR, SAS would select MAJOR to process the following WHERE expression:

```
where class=97 and major='Biology';
```

Composite Index

A composite index is an index of two or more key variables with their values concatenated to form a single value. The variables can be numeric, character, or a combination. An example is a composite index for the variables LASTNAME and FRSTNAME. A value for this index is composed of the value for LASTNAME immediately followed by the value for FRSTNAME from the same observation. When you create a composite index, you must specify a unique index name.

The following example shows the DATASETS procedure statements that are used to create a composite index for the data file COLLEGE.MAILLIST, specifying two key variables: ZIPCODE and SCHOOLID.

```
proc datasets library=college;
   modify maillist;
      index create zipid=(zipcode schoolid);
run;
```

Often, only the first variable of a composite index is used. For example, for a composite index on ZIPCODE and SCHOOLID, the following WHERE expression can use the composite index for the variable ZIPCODE because it is the first key variable in the composite index:

```
where zipcode = 78753;
```

However, you can take advantage of all key variables in a composite index by the way you construct the WHERE expression, which is referred to as *compound optimization*. Compound optimization is the process of optimizing multiple conditions on multiple variables, which are joined with a logical operator such as AND, using a composite index. If you issue the following WHERE expression, the composite index is used to find all occurrences of ZIPCODE='78753' and SCHOOLID='55'. In this way, all of the conditions are satisfied with a single search of the index:

```
where zipcode = 78753 and schoolid = 55;
```

When you are deciding whether to create a simple index or a composite index, consider how you will access the data. If you often access data for a single variable, a simple index will do. But if you frequently access data for multiple variables, a composite index could be beneficial.

Unique Values

Often it is important to require that values for a variable be unique, like social security number and employee number. You can declare unique values for a variable by creating an index for the variable and including the UNIQUE option. A unique index guarantees that values for one variable or the combination of a composite group of variables remain unique for every observation in the data file. If an update tries to add a duplicate value to that variable, the update is rejected.

The following example creates a simple index for the variable IDNUM and requires that all values for IDNUM be unique:

```
proc datasets library=college;
   modify student;
      index create idnum / unique;
run;
```

Missing Values

If a variable has a large number of missing values, it may be desirable to keep them from using space in the index. Therefore, when you create an index, you can include the NOMISS option to specify that missing values are not maintained by the index.

The following example creates a simple index for the variable RELIGION and specifies that the index does not maintain missing values for the variable:

```
proc datasets library=college;
   modify student;
```

```
      index create religion / nomiss;
run;
```

In contrast to the UNIQUE option, observations with missing values for the key variable can be added to the data file, even though the missing values are not added to the index.

SAS will not use an index that was created with the NOMISS option to process a BY statement or to process a WHERE expression that qualifies observations containing missing values. For example, suppose the index AGE was created with the NOMISS option and observations exist that contain missing values for the variable AGE. SAS will not use the index for the following:

```
proc print data=mydata.employee;
   where age < 35;
run;
```

Deciding Whether to Create an Index

Costs of an Index

An index exists to improve performance. However, an index conserves some resources at the expense of others. Therefore, you must consider costs associated with creating, using, and maintaining an index. The following topics provide information on resource usage and give you some guidelines for creating indexes.

When you are deciding whether to create an index, you must consider CPU cost, I/O cost, buffer requirements, and disk space requirements.

CPU Cost

Additional CPU time is necessary to create an index as well as to maintain the index when the data file is modified. That is, for an indexed data file, when a value is added, deleted, or modified, it must also be added, deleted, or modified in the appropriate index(es).

When SAS uses an index to read an observation from a data file, there is also increased CPU usage. The increased usage results from SAS using a more complicated process than is used when SAS retrieves data sequentially. Although CPU usage is greater, you benefit from SAS reading only those observations that meet the conditions. Note that this is why using an index is more expensive when there is a larger number of observations that meet the conditions.

Note: To compare CPU usage with and without an index, for some operating environments, you can issue the STIMER or FULLSTIMER system options to write performance statistics to the SAS log. △

I/O Cost

Using an index to read observations from a data file may increase the number of I/O (input/output) requests compared to reading the data file sequentially. For example, processing a BY statement with an index may increase I/O count, but you save in not having to issue the SORT procedure. For WHERE processing, SAS considers I/O count when deciding whether to use an index.

To process a request using an index, the following occurs:

1 SAS does a binary search on the index file and positions the index to the first entry that contains a qualified value.

2. SAS uses the value's RID (identifier) to directly access the observation containing the value. SAS transfers the observation between external storage to a *buffer*, which is the memory into which data is read or from which data is written. The data is transferred in *pages*, which is the amount of data (the number of observations) that can be transferred for one I/O request; each data file has a specified page size.
3. SAS then continues the process until the WHERE expression is satisfied. Each time SAS accesses an observation, the data file page containing the observation must be read into memory if it is not already there. Therefore, if the observations are on multiple data file pages, an I/O operation is performed for each observation.

The result is that the more random the data, the more I/Os are required to use the index. If the data is ordered more like the index, which is in ascending value order, fewer I/Os are required to access the data.

The number of buffers determines how many pages of data can simultaneously be in memory. Frequently, the larger the number of buffers, the fewer number of I/Os will be required. For example, if the page size is 4096 bytes and one buffer is allocated, then one I/O transfers 4096 bytes of data (or one page). To reduce I/Os, you can increase the page size but you will need a larger buffer. To reduce the buffer size, you can decrease the page size but you will use more I/Os.

For information on data file characteristics like the data file page size and the number of data file pages, issue the CONTENTS procedure (or use the CONTENTS statement in the DATASETS procedure). With this information, you can determine the data file page size and experiment with different sizes. Note that the information that is available from PROC CONTENTS depends on the operating environment.

The BUFSIZE= data set option (or system option) sets the page size for a data file when it is created. The BUFNO= data set option (or system option) specifies how many buffers to allocate for a data file and for the overall system for a given execution of SAS; that is, BUFNO= is not stored as a data set attribute.

Buffer Requirements

In addition to the resources that are used to create and maintain an index, SAS also requires additional memory for buffers when an index is actually used. Opening the data file opens the index file but none of the indexes. The buffers are not required unless SAS uses the index but they must be allocated in preparation for the index that is being used. The number of buffers that are allocated depends on the number of levels in the index tree and in the data file open mode. If the data file is open for input, the maximum number of buffers is three; for update, the maximum number is four. (Note that these buffers are available for other uses; they are not dedicated to indexes.)

Disk Space Requirements

Additional disk space is required to store the index file, which may show up as a separate file or may appear to be part of the data file, depending on the operating environment.

For information on the index file size, issue the CONTENTS procedure (or the CONTENTS statement in the DATASETS procedure). Note that the available information from PROC CONTENTS depends on the operating environment.

Guidelines for Creating Indexes

Data File Considerations

- For a small data file, sequential processing is often just as efficient as index processing. *Do not create an index if the data file page count is less than three pages.* It would be faster to access the data sequentially. To see how many pages are in a data file, use the CONTENTS procedure (or use the CONTENTS statement in the DATASETS procedure). Note that the information that is available from PROC CONTENTS depends on the operating environment.
- Consider the cost of an index for a data file that is frequently changed. If you have a data file that changes often, the overhead associated with updating the index after each change can outweigh the processing advantages you gain from accessing the data with in index.
- Create an index when you intend to retrieve a small subset of observations from a large data file (for example, less than 25% of all observations). When this occurs, the cost of processing data file pages is lower than the overhead of sequentially reading the entire data file. The smaller the subset, the larger the performance gains.
- To reduce the number of I/Os performed when you create an index, first sort the data by the key variable. Then to improve performance, maintain the data file in sorted order by the key variable. This technique will reduce the I/Os by grouping like values together. That is, the more ordered the data file is with respect to the key variable, the more efficient the use of the index. If the data file has more than one index, sort the data by the most frequently used key variable.

Index Use Considerations

- Keep the number of indexes per data file to a minimum to reduce disk storage and to reduce update costs.
- Consider how often your applications will use an index. An index must be used often in order to make up for the resources that are used in creating and maintaining it. That is, do not rely solely on resource savings from processing a WHERE expression. Take into consideration the resources it takes to actually create the index and to maintain it every time the data file is changed.
- When you create an index to process a WHERE expression, do not try to create one index that is used to satisfy all queries. If there are several variables that appear in queries, then those queries may be best satisfied with simple indexes on the most discriminating of those variables.

Key Variable Candidates

In most cases, multiple variables are used to query a data file. However, it probably would be a mistake to index all variables in a data file, as certain variables are better candidates than others:

- The variables to be indexed should be those that are used in queries. That is, your application should require selecting small subsets from a large file, and the most common selection variables should be considered as candidate key variables.
- A variable is a good candidate for indexing when the variable can be used to precisely identify the observations that satisfy a WHERE expression. That is, the

variable should be *discriminating*, which means that the index should select the fewest possible observations. For example, variables such as AGE, FRSTNAME, and GENDER are not discriminating because it is very possible for a large representation of the data to have the same age, first name, and gender. However, a variable such as LASTNAME is a good choice because it is less likely that many employees share the same last name.

For example, consider a data file with variables LASTNAME and GENDER.

- If many queries against the data file include LASTNAME, then indexing LASTNAME could prove to be beneficial because the values are usually discriminating. However, the same reasoning would not apply if you issued a large number of queries that included GENDER. The GENDER variable is not discriminating (because perhaps half the population are male and half are female).
- However, if queries against the data file most often include both LASTNAME and GENDER as shown in the following WHERE expression, then creating a composite index on LASTNAME and GENDER could improve performance.

  ```
  where lastname='LeVoux' and gender='F';
  ```

  Note that when you create a composite index, the first key variable should be the most discriminating.

Methods of Creating an Index

You can create one index for a data file, which can be either a simple index or a composite index, or you can create multiple indexes, which can be multiple simple indexes, multiple composite indexes, or a combination of both simple and composite. In general, the process of creating an index is as follows:

1. You request to create an index for one or multiple variables using a method such as the INDEX CREATE statement in the DATASETS procedure.
2. SAS reads the data file one observation at a time, extracts values and RID(s) for each key variable, and places them in the index file.
3. SAS then examines the data file to determine if the data is already sorted by the key variable(s) in ascending order. SAS looks in the data file for its *sort assertion*, which is determined from a previous SORT procedure or from a SORTEDBY= data set option:
   - If the values are in ascending order, SAS does not have to sort the values for the index file and avoids the resource cost.
   - If the values are not in ascending order, SAS sorts the data going into the index file in ascending value order.

Note: If a data file's sort assertion is set from a SORTEDBY= data set option, SAS validates that the data is sorted as specified by the data set option. If the data is not sorted appropriately, the index will not be created, and a message displays telling you that the index was not created because values are not sorted in ascending order. △

Methods to create an index are briefly described in this section; for details, refer to the INDEX= data set option in the *SAS Language Reference: Dictionary*.

Using the DATASETS Procedure

The DATASETS procedure provides statements that allow you to create and delete indexes. In the following example, the MODIFY statement identifies the data file, the

INDEX DELETE statement deletes two indexes, and the two INDEX CREATE statements specify the variables to index, with the first INDEX CREATE statement specifying the options UNIQUE and NOMISS:

```
proc datasets library=mylib;
modify employee;
   index delete salary age;
   index create empnum / unique nomiss;
   index create names=(lastname frstname);
```

Note: If you delete and create indexes in the same step, place the INDEX DELETE statement before the INDEX CREATE statement so that space occupied by deleted indexes can be reused during index creation. △

Using the INDEX= Data Set Option

To create indexes in a DATA step when you create the data file, use the INDEX= data set option. The INDEX= data set option also allows you to include the NOMISS and UNIQUE options. The following example creates a simple index on the variable STOCK and specifies UNIQUE:

```
data finances(index=(stock) /unique);
```

The next example uses the variables SSN, CITY, and STATE to create a simple index named SSN and a composite index named CITYST:

```
data employee(index=(ssn cityst=(city state)));
```

Using the SQL Procedure

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 (which is an SQL convention) instead of blanks (which is a SAS convention).

The DROP INDEX statement deletes indexes. The CREATE INDEX statement specifies the UNIQUE option, the name of the index, the target data file, and the variable(s) to be indexed. For example:

```
drop index salary from employee;
create unique index empnum on employee (empnum);
create index names on employee (lastname, frstname);
```

Using Other SAS Products

You can also create and delete indexes using other SAS utilities and products, such as the SAS Explorer, SAS/IML software, SAS Component Language, and SAS/Warehouse Administrator software.

Using an Index for WHERE Processing

WHERE processing conditionally selects observations for processing when you issue 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, by default SAS decides whether to use an index or read all the observations in the data file sequentially. To make this decision, SAS does the following:

1 Identifies an available index or indexes.

2 Estimates the number of observations that would be qualified. If multiple indexes are available, SAS selects the index that returns the smallest subset of observations.

3 Compares resource usage to decide whether it is more efficient to satisfy the WHERE expression by using the index or by reading all the observations sequentially.

Identifying Available Index or Indexes

The first step for SAS in deciding whether to use an index to process a WHERE expression is to identify if the variable or variables included in the WHERE expression are key variables (that is, have an index). Even though a WHERE expression can consist of multiple conditions specifying different variables, SAS uses only one index to process the WHERE expression. SAS tries to select the index that satisfies the most conditions and selects the smallest subset:

- For the most part, SAS selects one condition. The variable specified in the condition will have either a simple index or be the first key variable in a composite index.
- However, you can take advantage of multiple key variables in a composite index by constructing an appropriate WHERE expression, referred to as *compound optimization*.

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

Table 28.5 WHERE Conditions That Can Be Optimized

Condition	Examples

comparison operators, which include the EQ operator; directional comparisons like less than or greater than; and the IN operator	where empnum eq 3374;
	where empnum < 2000;
	where state in ('NC','TX');

comparison operators with NOT	where empnum ^= 3374; where x not in (5,10);
comparison operators with the colon modifier	where lastname gt: 'Sm';
CONTAINS operator	where lastname contains 'Sm';

fully-bounded range conditions specifying both an upper and 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 or IS MISSING operator	where name is null; where idnum is missing;

Condition	Examples
TRIM function	where trim(state)='Texas';
SUBSTR function in the form of: WHERE SUBSTR (*variable, position, length*)='*string*'; when the following conditions are met: *position* is equal to 1, *length* is less than or equal to the length of *variable*, and *length* is equal to the length of *string*	where substr (name,1,3)='Mac' and (city='Charleston' or city='Atlanta');

The following examples illustrate optimizing a single condition:

□ The following WHERE expressions could use a simple index on the variable MAJOR:

```
where major in ('Biology', 'Chemistry', 'Agriculture');
where class=90 and major in ('Biology', 'Agriculture');
```

□ With a composite index on variables ZIPCODE and SCHOOLID, SAS could use the composite index to satisfy the following conditions because ZIPCODE is the first key variable in the composite index:

```
where zipcode = 78753;
```

However, the following condition cannot use the composite index because the variable SCHOOLID is not the first key variable in the composite index:

```
where schoolid gt 1000;
```

Note: An index is not supported for arithmetic operators, a variable-to-variable condition, and the sounds-like operator. △

Compound Optimization

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. Using a single index to optimize the conditions can greatly improve performance.

For example, suppose you have a composite index for LASTNAME and FRSTNAME. If you issue the following WHERE expression, SAS uses the concatenated values for the first two variables, then SAS further evaluates each qualified observation for the EMPID value:

```
where lastname eq 'Smith' and frstname eq 'John' and empid=3374;
```

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

□ At least the first two key variables in the composite index must be used in the WHERE conditions.

□ The conditions are connected using the AND logical operator:

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

Any conditions connected using the OR logical operator must specify the same variable:

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

- At least one condition must be the EQ or IN operator; you cannot have, for example, all fully-bounded range conditions.

Note: The same conditions that are acceptable for optimizing a single condition are acceptable for compound 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 the following examples, assume there is a composite index named IJK for variables I, J, and K:

1 The following conditions are compound optimized because every condition specifies a variable that is in the composite index, and each condition uses one of the supported operators. SAS will position the composite index to the first entry that meets all three conditions and will retrieve only observations that satisfy all three conditions:

```
where i = 1 and j not in (3,4) and 10 < k < 12;
```

2 This WHERE expression cannot be compound optimized because the range condition for variable I is not fully bounded. In a fully-bounded condition, both an upper and lower bound must be specified. The condition I < 5 only specifies an upper bound. In this case, the composite index can still be used to optimize the single condition I < 5:

```
where i < 5 and j in (3,4) and k =3;
```

3 For the following WHERE expression, only the first two conditions are optimized with index IJK. After retrieving a subset of observations that satisfy the first two conditions, SAS examines the subset and eliminates any observations that fail to match the third condition.

```
where i in (1,4) and j = 5 and k like '%c'l
```

4 The following WHERE expression cannot be optimized with index IJK because J and K are not the first two key variables in the composite index:

```
where j = 1 and k = 2;
```

5 This WHERE expression can be optimized for variables I and J. After retrieving observations that satisfy the second and third conditions, SAS examines the subset and eliminates those observations that do not satisfy the first condition.

```
where x < 5 and i = 1 and j = 2;
```

Estimating the Number of Qualified Observations

Once SAS identifies the index or indexes that can satisfy the WHERE expression, the software estimates the number of observations that will be qualified by an available index. When multiple indexes exist, SAS selects the one that appears to produce the fewest qualified observations.

Starting with Version 7, the software's ability to estimate the number of observations that will be qualified is improved because the software stores additional statistics called cumulative percentiles (or *centiles* for short). Centiles information represents the distribution of values in an index so that SAS does not have to assume a uniform distribution as in prior releases. To print centiles information for an indexed data file, include the CENTILES option in PROC CONTENTS (or in the CONTENTS statement in the DATASETS procedure).

Note that, by default, SAS does not update centiles information after every data file change. When you create an index, you can include the UPDATECENTILES option to specify when centiles information is updated. That is, you can specify that centiles information be updated every time the data file is closed, when a certain percent of values for the key variable have been changed, or never. In addition, you can also request that centiles information is updated immediately, regardless of the value of UPDATECENTILES, by issuing the INDEX CENTILES statement in PROC DATASETS.

As a general rule, SAS uses an index if it estimates that the WHERE expression will select approximately one-third or fewer of the total number of observations in the data file.

Note: If SAS estimates that the number of qualified observations is less than 3% of the data file (or if no observations are qualified), SAS automatically uses the index. In other words, in this case, SAS does not bother comparing resource usage. △

Comparing Resource Usage

Once SAS estimates the number of qualified observations and selects the index that qualifies the fewest observations, SAS must then decide if it is faster (cheaper) to satisfy the WHERE expression by using the index or by reading all of the observations sequentially. SAS makes this determination as follows:

- If only a few observations are qualified, it is more efficient to use the index than to do a sequential search of the entire data file.
- If most or all of the observations qualify, then it is more efficient to simply sequentially search the data file than to use the index.

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

To compare resource usage, SAS does the following:

1. First, SAS predicts the number of I/Os it will take to satisfy the WHERE expression using the index. To do so, SAS positions the index to the first entry that contains a qualified value. In a buffer management simulation that takes into account the current number of available buffers, the RIDs (identifiers) on that index page are processed, indicating how many I/Os it will take to read the observations in the data file.

   If the observations are randomly distributed throughout the data file, the observations will be located on multiple data file pages. This means an I/O will be needed for each page. Therefore, the more random the data in the data file, the more I/Os it takes to use the index. If the data in the data file is ordered more like the index, which is in ascending value order, fewer I/Os are needed to use the index.
2. Then SAS calculates the I/O cost of a sequential pass of the entire data file and compares the two resource costs.

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

Note: If comparing resource costs results in a tie, SAS chooses the index. △

Controlling WHERE Processing Index Usage with Data Set Options

In Version 7 or later releases, you can control index usage for WHERE processing with the IDXWHERE= and IDXNAME= data set options.

The IDXWHERE= data set option overrides the software's decision regarding whether to use an index to satisfy the conditions of a WHERE expression as follows:

- □ IDXWHERE=YES tells SAS to decide which index is the best for optimizing a WHERE expression, disregarding the possibility that a sequential search of the data file might be more resource efficient.
- □ IDXWHERE=NO tells SAS to ignore all indexes and satisfy the conditions of a WHERE expression by sequentially searching the data file.
- □ Using an index to process a BY statement cannot be overridden with IDXWHERE=.

The following example tells SAS to decide which index is the best for optimizing the WHERE expression. SAS will disregard the possibility that a sequential search of the data file might be more resource efficient.

```
data mydata.empnew;
   set mydata.employee (idxwhere=yes);
   where empnum < 2000;
```

For details, see the IDXWHERE data set option in *SAS Language Reference: Dictionary*.

The IDXNAME= data set option directs SAS to use a specific index in order to satisfy the conditions of a WHERE expression.

By specifying IDXNAME=*index-name*, you are specifying the name of a simple or composite index for the data file.

The following example uses the IDXNAME= data set option to direct SAS to use a specific index to optimize the WHERE expression. SAS will disregard the possibility that a sequential search of the data file might be more resource efficient and does not attempt to determine if the specified index is the best one. (Note that the EMPNUM index was not created with the NOMISS option.)

```
data mydata.empnew;
   set mydata.employee (idxname=empnum);
   where empnum < 2000;
```

For details, see the IDXNAME data set option in *SAS Language Reference: Dictionary*.

Displaying Index Usage Information in the SAS Log

To display information in the SAS log regarding index usage, change the value of the MSGLEVEL= system option from its default value of N to I. When you issue **`options msglevel=i;`**, the following occurs:

- □ If an index is used, a message displays specifying the name of the index.
- □ If an index is not used but one exists that could optimize at least one condition in the WHERE expression, messages provide suggestions as to what you can do to influence SAS to use the index; for example, a message could suggest sorting the data file into index order or specifying more buffers.
- □ A message displays the IDXWHERE= or IDXNAME= data set option value if the setting can affect index processing.

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. That is, if the view definition includes a WHERE expression using a key variable, then SAS will attempt to use the index. Additionally, there are other ways to take advantage of a key variable when using a view.

In this example, you create an SQL view named STAT from data file CRIME, which has the key variable STATE. In addition, the view definition includes a WHERE expression:

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

If you issue the following PRINT procedure, which refers to the SQL view, along with a WHERE statement that specifies the key variable STATE, SAS cannot optimize the WHERE statement with the index. SQL views cannot join a WHERE expression that was defined in the view to a WHERE expression that was specified in another procedure, DATA step, or SCL:

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

However, if you issue PROC SQL with an SQL WHERE clause that specifies the key variable STATE, then the SQL view can join the two conditions, which allows SAS to use the index STATE:

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

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 that are specified in a BY statement. Indexing a data file enables you to use a BY statement without sorting the data file. By creating an index based on one or more variables, you can ensure that observations are processed in *ascending numeric or character order*. Simply specify in the BY statement the variable or list of variables that are indexed.

For example, if an index exists for LASTNAME, the following BY statement would use the index to order the values by last names:

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

When you specify a BY statement, SAS looks for an appropriate index. If one exists, the software automatically retrieves the observations from the data file in indexed order.

A BY statement will use an index in the following situations:

- The BY statement consists of one variable that is the key variable for a simple index or the first key variable in a composite index.
- The BY statement consists of two or more variables and the first variable is the key variable for a simple index or the first key variable in a composite index.

For example, if the variable MAJOR has a simple index, the following BY statements use the index to order the values by MAJOR:

```
by major;
by major state;
```

If a composite index named ZIPID exists consisting of the variables ZIPCODE and SCHOOLID, the following BY statements use the index:

```
by zipcode;
by zipcode schoolid;
by zipcode schoolid name;
```

However, the composite index ZIPID is not used for these BY statements:

```
by schoolid;
by schoolid zipcode;
```

In addition, a BY statement will not use an index in these situations:

- The BY statement includes the DESCENDING or NOTSORTED option.
- The index was created with the NOMISS option.
- The data file is physically stored in sorted order based on the variables specified in the BY statement.

Note: Using an index to process a BY statement may not always be more efficient than simply sorting the data file, particularly if the data file has a high blocking factor of observations per page. Therefore, using an index for a BY statement is generally for convenience, not performance. △

Using an Index for Both WHERE and BY Processing

If both a WHERE expression and a BY statement are specified, SAS looks for one index that satisfies requirements for both. If such an index is not found, the BY statement takes precedence.

With a BY statement, SAS cannot use an index to optimize a WHERE expression if the optimization would invalidate the BY order. For example, the following statements could use an index on the variable LASTNAME to optimize the WHERE expression because the order of the observations returned by the index does not conflict with the order required by the BY statement:

```
proc print;
  by lastname;
  where lastname >= 'Smith';
run;
```

However, the following statements cannot use an index on LASTNAME to optimize the WHERE expression because the BY statement requires that the observations be returned in EMPID order:

```
proc print;
   by empid;
   where lastname = 'Smith';
run;
```

Specifying an Index with the KEY= Option for SET and MODIFY Statements

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 file.

The following MODIFY statement shows how to use the KEY= option to take advantage of the fact that the data file INVTY.STOCK has an index on the variable

PARTNO. Using the KEY= option tells SAS to use the index to directly access the correct observations to modify.

```
modify invty.stock key=partno;
```

Note: A BY statement is not allowed in the same DATA step with the KEY= option, and WHERE processing is not allowed for a data file with the KEY= option. △

Taking Advantage of an Index

Applications that typically do not use indexes can be rewritten to take advantage of an index. For example:

- Consider replacing a subsetting IF statement (which never uses an index) with a WHERE statement. However, be careful because the statements are processed differently and may produce different results in DATA steps that use the SET, MERGE, or UPDATE statements. This is because the WHERE statement selects observations before they are brought into the Program Data Vector (PDV), whereas the subsetting IF statement selects observations after they are read into the PDV.
- Consider using the WHERE command in the FSEDIT procedure in place of the SEARCH and FIND commands.

Maintaining Indexes

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

Displaying Data File Information

The CONTENTS procedure (or the CONTENTS statement in PROC DATASETS) reports the following types of information.

- number and names of indexes for a data file
- the names of key variables
- the options in effect for each key variable
- data file page size
- number of data file pages
- centiles information (using the CENTILES option)
- amount of disk space used by the index file.

Note: The available information depends on the operating environment. △

Output 28.5 Output of PROC CONTENTS

```
                                    The SAS System

                                The CONTENTS Procedure

            Data Set Name: SASUSER.STAFF                        Observations:         148
            Member Type:   DATA                                 Variables:            6
            Engine:        V8                                   Indexes:              2
            Created:       9:59 Tuesday, May 11, 1999           Observation Length:   63
            Last Modified: 10:03 Tuesday, May 11, 1999          Deleted Observations: 0
            Protection:                                         Compressed:           NO
            Data Set Type:                                      Sorted:               NO
            Label:


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

      Data Set Page Size:         8192
      Number of Data Set Pages:   3
      First Data Page:            1
      Max Obs per Page:           129
      Obs in First Data Page:     104
      Index File Page Size:       8192

                                    The SAS System

                                The CONTENTS Procedure

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

      Number of Index File Pages: 3
      Number of Data Set Repairs: 0
      File Name:                  /remote/obi01/wan0.2/u/sasXXX/sasuser.devn/staff.sas7bdat
      Release Created:            8.00.00B
      Host Created:               HP-UX
      Inode Number:               237883
      Access Permission:          rw-r--r--
      Owner Name:                 XXXXXX
      File Size (bytes):          32768


                                    The SAS System

                                The CONTENTS Procedure

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

                               #    Variable    Type    Len    Pos
                               -----------------------------------
                               4    city        Char     15     34
                               3    fname       Char     15     19
                               6    hphone      Char     12     51
                               1    idnum       Char      4      0
                               2    lname       Char     15      4
                               5    state       Char      2     49
```

```
                              The SAS System

                                 The CONTENTS Procedure

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

                                     Current      # of
              Unique      Update     Update    Unique
#    Index    Option    Centiles    Percent    Values    Variables
-----------------------------------------------------------------------------------------
1    idnum    YES              5          0       148
     ---                                                 1009
     ---                                                 1065
     ---                                                 1105
     ---                                                 1115
     ---                                                 1123
     ---                                                 1130
     ---                                                 1221
     ---                                                 1352
     ---                                                 1385
     ---                                                 1405
     ---                                                 1412

                                     The SAS System

                                 The CONTENTS Procedure

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

                                     Current      # of
              Unique      Update     Update    Unique
#    Index    Option    Centiles    Percent    Values    Variables
-----------------------------------------------------------------------------------------
     ---                                                 1421
     ---                                                 1429
     ---                                                 1436
     ---                                                 1475
     ---                                                 1521
     ---                                                 1616
     ---                                                 1739
     ---                                                 1845
     ---                                                 1919
     ---                                                 1995
2    names                     5          0       148    fname lname
     ---                                                 ABDULLAH       ,ALHERTANI

                                     The SAS System

                                 The CONTENTS Procedure

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

                                     Current      # of
              Unique      Update     Update    Unique
#    Index    Option    Centiles    Percent    Values    Variables
-----------------------------------------------------------------------------------------
     ---                                                 ALICE          ,MURPHY
     ---                                                 ANTHONY        ,COOPER
     ---                                                 CAROL          ,PEARCE
     ---                                                 CLYDE          ,HERRERO
     ---                                                 DIANE          ,NORRIS
     ---                                                 ELIZABETH      ,VARNER
     ---                                                 GRETCHEN       ,HOWARD
     ---                                                 JAKOB          ,BREWCZAK
     ---                                                 JEFF           ,LI
     ---                                                 JOHN           ,MARKS
     ---                                                 JULIA          ,RODRIGUEZ
     ---                                                 LARRY          ,UPCHURCH
```

```
                                   The SAS System

                               The CONTENTS Procedure

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

                                  Current      # of
              Unique      Update    Update    Unique
#    Index    Option    Centiles   Percent    Values    Variables
---------------------------------------------------------------------------------------
     ---                                                LEVI          ,GOLDSTEIN
     ---                                                MARY          ,PARKER
     ---                                                NADINE        ,WELLS
     ---                                                RANDY         ,SANYERS
     ---                                                ROGER         ,DENNIS
     ---                                                SANDRA        ,NEWKIRK
     ---                                                THOMAS        ,BURNETTE
     ---                                                WILLIAM       ,PHELPS
```

Copying an Indexed Data File

When you copy an indexed data file with the COPY procedure (or the COPY statement of the DATASETS procedure), you can specify whether the procedure also recreates the index file for the new data file with the INDEX=YES | NO option; the default is YES, which recreates the index. However, recreating the index does increase the processing time for the PROC COPY step.

If you copy from disk to disk, the index is recreated. If you copy from disk to tape, the index is not recreated on tape. However, after copying from disk to tape, if you then copy back from tape to disk, the index can be recreated. Note that if you move a data file with the MOVE option in PROC COPY, the index file is deleted from IN= library and recreated in OUT= library.

The CPORT procedure also has INDEX=YES | NO to specify whether to export indexes with indexed data files. By default, PROC CPORT exports indexes with indexed data files. The CIMPORT procedure, however, does not handle the index file at all, and the index(es) must be recreated.

Updating an Indexed Data File

Each time that values in an indexed data file are added, modified, or deleted, SAS automatically updates the index. The following activities affect an index as indicated:

Table 28.6 Maintenance Tasks and Index Results

Task	Result
delete a data set	index file is deleted
rename a data set	index file is renamed
rename key variable	simple index is renamed
delete key variable	simple index is deleted
add observation	index entries are added

Task	Result
delete observations	index entries are deleted and space is recovered for resuse
update observations	index entries are deleted and new ones are inserted

Note: Use the SAS System to perform additions, modifications and deletions to your data sets. Using operating system commands to perform these operations will make your files unusable. △

Sorting an Indexed Data File

You can sort an indexed data file only if you direct the output of the SORT procedure to a new data file so that the original data file remains unchanged. However, the new data file is not automatically indexed.

Note: If you sort an indexed data file with the FORCE option, the index file is deleted. △

Adding Observations to an Indexed Data File

Adding observations to an indexed data file requires additional processing. SAS automatically keeps the values in the index consistent with the values in the data file.

Multiple Occurrences

An index that is created without the UNIQUE option can result in multiple occurrences of the same value, which results in multiple RIDs for one value. For large data files with many multiple occurrences, the list of RIDs for a given value may require several pages in the index file. Because the RIDs are stored in physical order, any new observation added to the data file with the given value is stored at the end of the list of RIDs. Navigating through the index to find the end of the RID list can cause many I/O operations.

In Version 7 and later releases, SAS remembers the previous position in the index so that when inserting more occurrences of the same value, the end of the RID list is found quickly.

Appending to an Indexed Data File

Version 7 and later releases provide performance improvements when appending a data file to an indexed data file. SAS suspends index updates until all observations are added, then updates the index with data from the newly added observations. See the APPEND statement in the DATASETS procedure in *SAS Language Reference: Dictionary*.

Recovering a Damaged Index

An index can become damaged for many of the same reasons that a data file or catalog can become damaged. If a data file becomes damaged, use the REPAIR statement in PROC DATASETS to repair the data file or recreate any missing indexes. For example,

```
proc datasets library=mylib;
   repair mydata;
run;
```

Compressed Data Files

You can compress data files to save space. When you create a compressed data file, SAS writes a note to the log indicating the percentage reduction that is obtained by compressing the data file. The compression percentage is obtained by comparing the size of the compressed data set with the size of a noncompressed data file of the same page size and record count. Note that compression may not result in a smaller data file.

To compress SAS data files, use the COMPRESS= data set option or the COMPRESS= system option. When you specify COMPRESS=YES, SAS uses the default compression algorithm. You can also specify your own compression algorithm or use another compression algorithm supplied by SAS by specifying COMPRESS=*algorithm-name*. See the COMPRESS= data set option and the COMPRESS= system option in *SAS Language Reference: Dictionary* for more information.

The following table shows additional options that you can use with COMPRESS= when you create a compressed data file.

Table 28.7 Options that You Can Use with COMPRESS=

To do this ...	Use ...	Example	Restrictions
Control whether a compressed data set may be processed with random access (by observation number)	POINTOBS= YES data set option	`data test (compress=yes pointobs=yes);`	POINTOBS=YES increases CPU usage when you create or update a compressed data set.
Specify whether new observations are written to free space in a compressed SAS data set to save storage space	REUSE=YES data set option or system option	`data test (compress=yes reuse=no);`	If you set REUSE=YES, SAS automatically sets POINTOBS=NO.

Note: POINTOBS=yes and REUSE=yes are mutually exclusive, that is, they cannot be used together. △

You can access observations in a compressed data file by specifying the observation number in:

- FSEDIT
- SET statement, POINT= option
- MODIFY statement, POINT= option.

Note: You cannot access observations by number if you set REUSE=YES. △

See the REUSE= data set option in *SAS Language Reference: Dictionary* for more information on access by observation number.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

Copyright © 1999 SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–441–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999