CHAPTER 24

Reading, Combining, and Modifying SAS Data Sets

Definitions

In the context of DATA step processing, the terms reading, combining and modifying have these meanings:

Reading a SAS data set
refers to opening a SAS data set and bringing an observation into the program data vector for processing.

Combining SAS data sets
refers to reading data from two or more SAS data sets and processing them by

- concatenating
- interleaving
- one-to-one reading
- one-to-one merging
- match-merging
- updating a master data set with a transaction data set.

The methods for combining SAS data sets are defined in “Combining SAS Data Sets: Methods” on page 330.

Modifying SAS data sets
refers to using the MODIFY statement to update information in a SAS data set in place. The MODIFY statement can save disk space because it modifies data in place, without creating a copy of the data set. You can modify a SAS data set with programming statements or with information that is stored in another data set.

Overview of Tools

The primary tools that are used for reading, combining, and modifying SAS data sets are four statements: SET, MERGE, MODIFY, and UPDATE. This section describes these tools and shows examples. For complete information about these statements see the Statements section of *SAS Language Reference: Dictionary*.

Reading SAS Data Sets

Reading a Single SAS Data Set

To read data from an existing SAS data set, use a SET statement. In this example, the DATA step creates data set PERM.TOUR155_PEAKCOST by reading data from data set PERM.TOUR155_BASIC_COST and by calculating values for the three new variables Total_Cost, Peak_Cost, and Average_Night_Cost.

```
data perm.tour155_peakcost;
   set perm.tour155_basic_cost;
   Total_Cost=AirCost+LandCost;
   Peak_Cost=(AirCost*1.15);
   Average_Night_Cost=LandCost/Nights;
run;
```

Reading from Multiple SAS Data Sets

You can read from multiple SAS data sets and combine and modify data in different ways. You can, for example, combine two or more input data sets to create one output data set, merge data from two or more input data sets that share a common variable, and update a master file based on transaction records.

For details about reading from multiple SAS data sets, see "Combining SAS Data Sets: Methods" on page 330.

Controlling the Reading and Writing of Variables and Observations

If you do not instruct it to do otherwise, SAS writes all variables and all observations from input data sets to output data sets. You can, however, control which variables and observations you want to read and write by using SAS statements, data set options, and functions. The statements and data set options that you can use are listed in the following table.

Table 24.1 Statements and Options That Control Reading and Writing

Task	Statements	Data set options	System options
Control variables	DROP	DROP=	
	KEEP	KEEP=	
	RENAME	RENAME=	
Control observations	WHERE	WHERE=	FIRSTOBS=
	subsetting IF	FIRSTOBS=	OBS=
	DELETE	OBS=	

Task	Statements	Data set options	System options
	REMOVE		
	OUTPUT		

Use statements or data set options (such as KEEP= and DROP=) to control the variables and observations you want to write to the output data set. The WHERE statement is an exception: it controls which observations are read into the program data vector based on the value of a variable. You can use data set options (including WHERE=) on input or output data sets, depending on their function and what you want to control. You can also use SAS system options to control your data.

Combining SAS Data Sets: The Essentials

What You Need to Know before Combining Information Stored In Multiple SAS Data Sets

Many applications require input data to be in a specific format before the data can be processed to produce meaningful results. The data typically comes from multiple sources and may be in different formats. Therefore, you often, if not always, have to take intermediate steps to logically relate and process data before you can analyze it or create reports from it.

Application requirements vary, but there are common factors for all applications that access, combine, and process data. Once you have determined what you want the output to look like, you must

- determine how the input data is related
- ensure that the data is properly sorted or indexed, if necessary
- select the appropriate access method to process the input data
- select the appropriate SAS tools to complete the task.

The Four Ways That Data Can Be Related

Relationships among multiple sources of input data exist when each of the sources contains common data, either at the physical or logical level. For example, employee data and department data could be related through an employee ID variable that shares common values. Another data set could contain numeric sequence numbers whose partial values logically relate it to a separate data set by observation number.

You must be able to identify the existing relationships in your data. This knowledge is crucial for understanding how to process input data in order to produce desired results. All related data fall into one of these four categories, characterized by how observations relate among the data sets:

- one-to-one
- one-to-many
- many-to-one
- many-to-many.

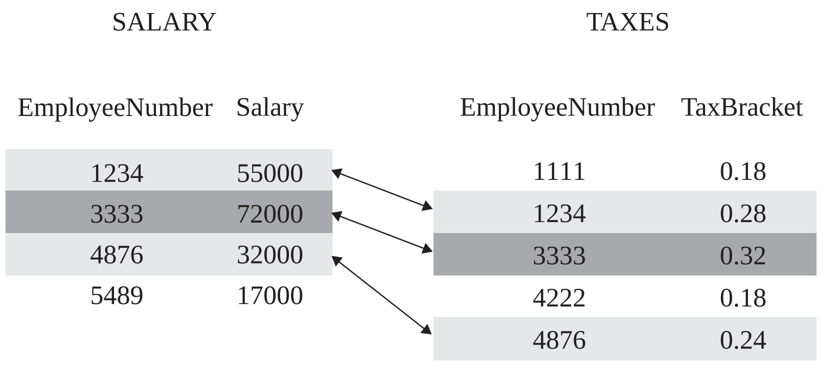
To obtain the results you want, you should understand how each of these methods combines observations, how each method treats duplicate values of common variables, and how each method treats missing values or nonmatched values of common variables. Some of the methods also require that you preprocess your data sets by sorting them or by creating indexes. See the description of each method in “Combining SAS Data Sets: Methods” on page 330.

One-to-One

In a one-to-one relationship, typically a single observation in one data set is related to a single observation from another based on the values of one or more selected variables. A one-to-one relationship implies that each value of the selected variable occurs no more than once in each data set. When you work with multiple selected variables, this relationship implies that each combination of values occurs no more than once in each data set.

In the following example, observations in data sets SALARY and TAXES are related by common values for EmployeeNumber.

Figure 24.1 One-to-One Relationship



SALARY

EmployeeNumber	Salary
1234	55000
3333	72000
4876	32000
5489	17000

TAXES

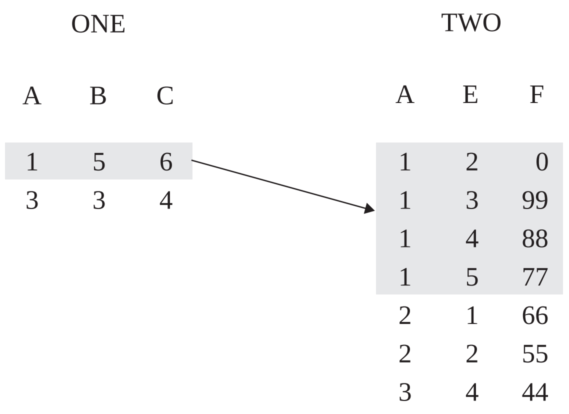
EmployeeNumber	TaxBracket
1111	0.18
1234	0.28
3333	0.32
4222	0.18
4876	0.24

One-to-Many and Many-to-One

A one-to-many or many-to-one relationship between input data sets implies that one data set has at most one observation with a specific value of the selected variable, but the other input data set may have more than one occurrence of each value. When you work with multiple selected variables, this relationship implies that each combination of values occurs no more than once in one data set, but may occur more than once in the other data set. The order in which the input data sets are processed determines whether the relationship is one-to-many or many-to-one.

In the following example, observations in data sets ONE and TWO are related by common values for variable A. Values of A are unique in data set ONE but not in data set TWO.

Figure 24.2 One-to-Many Relationship



ONE

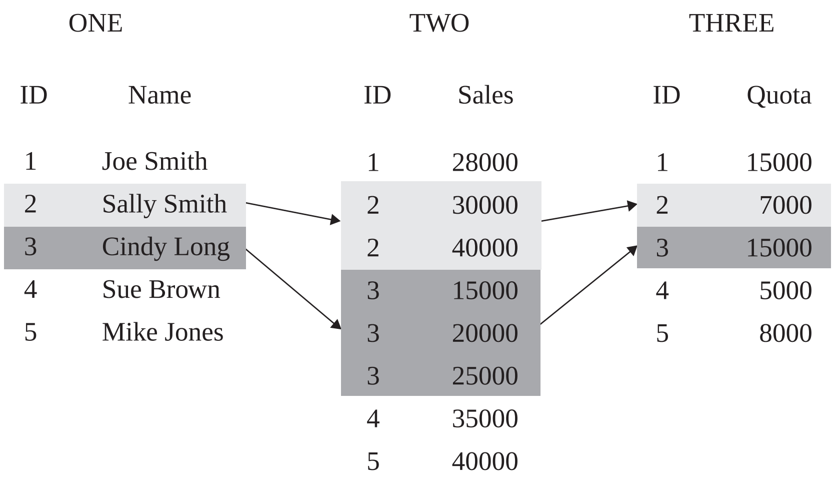
A	B	C
1	5	6
3	3	4

TWO

A	E	F
1	2	0
1	3	99
1	4	88
1	5	77
2	1	66
2	2	55
3	4	44

In the following example, observations in data sets ONE, TWO, and THREE are related by common values for variable ID. Values of ID are unique in data sets ONE and THREE but not in TWO. For values 2 and 3 of ID, a one-to-many relationship exists between observations in data sets ONE and TWO, and a many-to-one relationship exists between observations in data sets TWO and THREE.

Figure 24.3 One-to-Many and Many-to-One Relationships



ONE

ID	Name
1	Joe Smith
2	Sally Smith
3	Cindy Long
4	Sue Brown
5	Mike Jones

TWO

ID	Sales
1	28000
2	30000
2	40000
3	15000
3	20000
3	25000
4	35000
5	40000

THREE

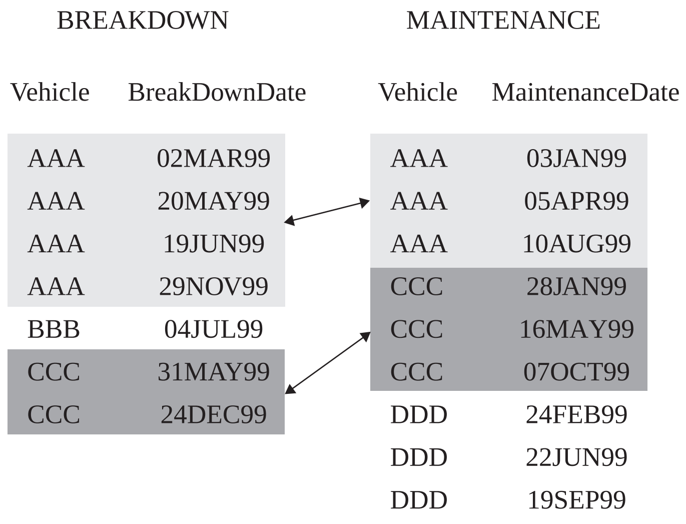
ID	Quota
1	15000
2	7000
3	15000
4	5000
5	8000

Many-to-Many

The many-to-many category implies that multiple observations from each input data set may be related based on values of one or more common variables.

In the following example, observations in data sets BREAKDOWN and MAINTENANCE are related by common values for variable Vehicle. Values of Vehicle are not unique in either data set. A many-to-many relationship exists between observations in these data sets for values AAA and CCC of Vehicle.

Figure 24.4 Many-to-Many Relationship



BREAKDOWN

Vehicle	BreakDownDate
AAA	02MAR99
AAA	20MAY99
AAA	19JUN99
AAA	29NOV99
BBB	04JUL99
CCC	31MAY99
CCC	24DEC99

MAINTENANCE

Vehicle	MaintenanceDate
AAA	03JAN99
AAA	05APR99
AAA	10AUG99
CCC	28JAN99
CCC	16MAY99
CCC	07OCT99
DDD	24FEB99
DDD	22JUN99
DDD	19SEP99

Access Methods: Sequential versus Direct

Overview

Once you have established data relationships, the next step is to determine the best mode of data access to relate the data. You can access observations sequentially in the order in which they appear in the physical file. Or you can access them directly, that is, you can go straight to an observation in a SAS data set without having to process each observation that precedes it.

Sequential Access

The simplest and perhaps most common way to process data with a DATA step is to read observations in a data set sequentially. You can read observations sequentially using the SET, MERGE, UPDATE, or MODIFY statements. You can also use the SAS File I/O functions, such as OPEN, FETCH, and FETCHOBS.

Direct Access

Direct access allows a program to access specific observations based on one of two methods:

- □ by an observation number
- □ by the value of one or more variables through a simple or composite index.

To access observations directly by their observation number, use the POINT= option with the SET or MODIFY statement. The POINT= option names a variable whose current value determines which observation a SET or MODIFY statement reads.

To access observations directly based on the values of one or more specified variables, you must first create an index for the variables and then read the data set using the KEY= statement option with the SET or MODIFY statement. An *index* is a separate structure that contains the data values of the key variable or variables, paired with a location identifier for the observations containing the value.

Note: You can also use the SAS File I/O functions such as CUROBS, NOTE, POINT and FETCHOBS to access observations by observation number. △

Overview of Methods for Combining SAS Data Sets

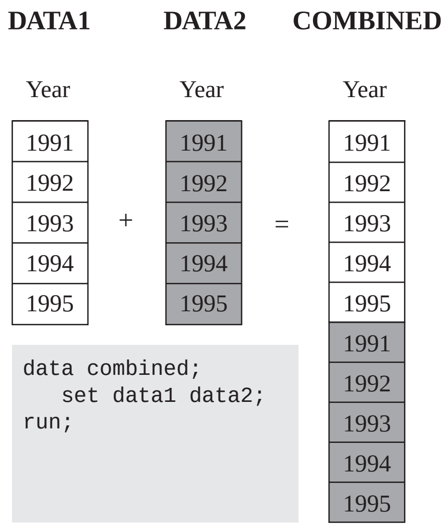
You can use these methods to combine SAS data sets:

- concatenating
- interleaving
- one-to-one reading
- one-to-one merging
- match merging
- updating.

Concatenating

The following figure shows the results of concatenating two SAS data sets. Concatenating the data sets appends the observations from one data set to another data set. The DATA step reads DATA1 sequentially until all observations have been processed, and then reads DATA2. Data set COMBINED contains the results of the concatenation. Note that the data sets are processed in the order in which they are listed in the SET statement.

Figure 24.5 Concatenating Two Data Sets

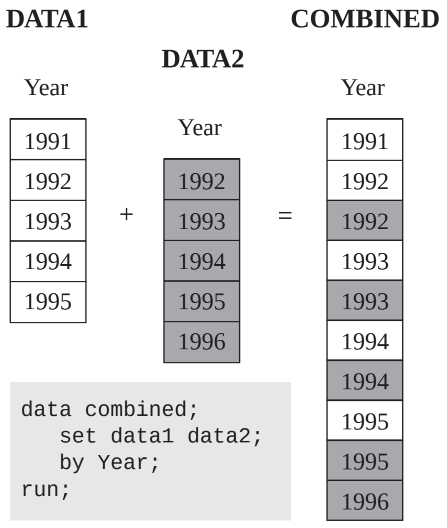


```
data combined;
   set data1 data2;
run;
```

Interleaving

The following figure shows the results of interleaving two SAS data sets. Interleaving intersperses observations from two or more data sets, based on one or more common variables. Data set COMBINED shows the result.

Figure 24.6 Interleaving Two Data Sets

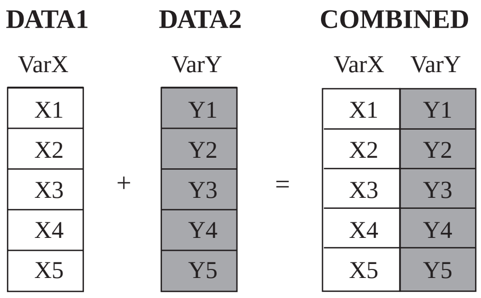


```
data combined;
   set data1 data2;
   by Year;
run;
```

One-to-One Reading and One-to-One Merging

The following figure shows the results of one-to-one reading and one-to-one merging. One-to-one reading combines observations from two or more SAS data sets by creating observations that contain all of the variables from each contributing data set. Observations are combined based on their relative position in each data set, that is, the first observation in one data set with the first in the other, and so on. The DATA step stops after it has read the last observation from the smallest data set. One-to-one merging is similar to a one-to-one reading, with two exceptions: you use the MERGE statement instead of multiple SET statements, and the DATA step reads all observations from all data sets. Data set COMBINED shows the result.

Figure 24.7 One-to-One Reading and One-to-One Merging



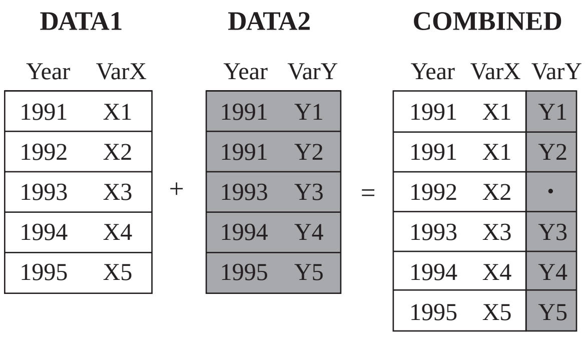
```
data combined;
   set data1;
   set data2;
run;

data combined;
   merge data1 data2;
run;
```

Match-Merging

The following figure shows the results of match-merging. Match-merging combines observations from two or more SAS data sets into a single observation in a new data set based on the values of one or more common variables. Data set COMBINED shows the results.

Figure 24.8 Match-Merging Two Data Sets



DATA1

Year	VarX
1991	X1
1992	X2
1993	X3
1994	X4
1995	X5

+

DATA2

Year	VarY
1991	Y1
1991	Y2
1993	Y3
1994	Y4
1995	Y5

=

COMBINED

Year	VarX	VarY
1991	X1	Y1
1991	X1	Y2
1992	X2	.
1993	X3	Y3
1994	X4	Y4
1995	X5	Y5

```
data combined;
   merge data1 data2;
   by Year;
run;
```

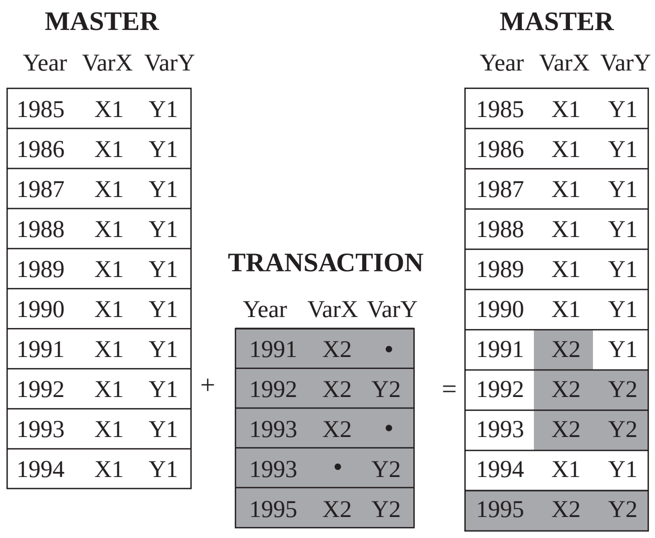
Updating

The following figure shows the results of updating a master data set. Updating uses information from observations in a transaction data set to delete, add, or alter information in observations in a master data set. You can update a master data set by using the UPDATE statement or the MODIFY statement. If you use the UPDATE statement, your input data sets must be sorted by the values of the variables listed in the BY statement. (In this example, MASTER and TRANSACTION are both sorted by Year.) If you use the MODIFY statement, your input data does not need to be sorted.

UPDATE replaces an existing file with a new file, allowing you to add, delete, or rename columns. MODIFY performs an update in place by rewriting only those records that have changed, or by appending new records to the end of the file.

Note that by default, UPDATE and MODIFY do not replace nonmissing values in a master data set with missing values from a transaction data set.

Figure 24.9 Updating a Master Data Set



MASTER

Year	VarX	VarY
1985	X1	Y1
1986	X1	Y1
1987	X1	Y1
1988	X1	Y1
1989	X1	Y1
1990	X1	Y1
1991	X1	Y1
1992	X1	Y1
1993	X1	Y1
1994	X1	Y1

+

TRANSACTION

Year	VarX	VarY
1991	X2	•
1992	X2	Y2
1993	X2	•
1993	•	Y2
1995	X2	Y2

=

MASTER

Year	VarX	VarY
1985	X1	Y1
1986	X1	Y1
1987	X1	Y1
1988	X1	Y1
1989	X1	Y1
1990	X1	Y1
1991	X2	Y1
1992	X2	Y2
1993	X2	Y2
1994	X1	Y1
1995	X2	Y2

```
data master;
   update master transaction;
   by Year;
run;
```

Overview of Tools for Combining SAS Data Sets

Using Statements and Procedures

Once you understand the basics of establishing relationships among data, the ways to access data, and the ways that you can combine SAS data sets, you can choose from a variety of SAS tools for accessing, combining, and processing your data. The following table lists and briefly describes the DATA step statements and the procedures that you can use for combining SAS data sets.

Table 24.2 Statements or Procedures for Combining SAS Data Sets

		Access Method			
Statement or Procedure	Action Performed	Sequential	Direct	Can Use with BY statement	Comments
BY	controls the operation of a SET, MERGE, UPDATE, or MODIFY statement in the DATA step and sets up special grouping variables.	NA	NA	NA	BY-group processing is a means of processing observations that have the same values of one or more variables.
MERGE	reads observations from two or more SAS data sets and joins them into a single observation.	X		X	When using MERGE with BY, the data must be sorted or indexed on the BY variable.
MODIFY	processes observations in a SAS data set in place. (Contrast with UPDATE.)	X	X	X	Sorted or indexed data are not required for use with BY, but are recommended for performance.
SET	reads an observation from one or more SAS data sets.	X	X	X	Use KEY= or POINT= statement options for directly accessing data.
UPDATE	applies transactions to observations in a master SAS data set. UPDATE does not update observations in place; it produces an updated copy of the current data set.	X		X	Both the master and transaction data sets must be sorted by or indexed on the BY variable.
PROC APPEND	adds the observations from one SAS data set to the end of another SAS data set.	X			
PROC SQL[1]	reads an observation from one or more SAS data sets; reads observations from up to 32 SAS data sets and joins them into single observations; manipulates observations in a SAS data set in place; easily produces a Cartesian product.	X	X	X	All three access methods are available in PROC SQL, but the access method is chosen by the internal optimizer.

1 PROC SQL is the SAS implementation of Structured Query Language. In addition to expected SQL capabilities, PROC SQL includes additional capabilities specific to SAS, such as the use of formats and SAS macro language.

Using Error Checking

You can use the _IORC_ automatic variable and the SYSRC autocall macro to perform error checking in a DATA step. Use these tools with the MODIFY statement or with the SET statement and the KEY= option. For more information about these tools, see "Error Checking When Using Indexes to Randomly Access or Update Data" on page 357.

How to Prepare Your Data Sets

Before combining SAS data sets, follow these guidelines to produce the results you want:

- Know the structure and the contents of the data sets.
- Look at sources of common problems.
- Ensure that observations are in the correct order, or that they can be retreived in the correct order (for example, by using an index).
- Test your program.

Knowing the Structure and Contents of the Data Sets

To help determine how your data are related, look at the structure of the data sets. To see the data set structure, execute the DATASETS procedure, the CONTENTS procedure, or access the SAS Explorer window in your windowing environment to display the descriptor information. Descriptor information includes the number of observations in each data set, the name and attributes of each variable, and which variables are included in indexes. To print a sample of the observations, use the PRINT procedure or the REPORT procedure.

You can also use functions such as VTYPE, VLENGTH, and VLENGTHX to show specific descriptor information. For a short description of these functions, see the Variable Information functions in Chapter 6, "Functions and CALL Routines," on page 43. For complete information about these functions, see "Functions and CALL Routines" in *SAS Language Reference: Dictionary*.

Looking at Sources of Common Problems

If your program does not execute correctly, review your input data for the following errors:

- *variables that have the same name but that represent different data*

  SAS includes only one variable of a given name in the new data set. If you are merging two data sets that have variables with the same names but different data, the values from the last data set that was read are written over the values from other data sets.

  To correct the error, you can rename variables before you combine the data sets by using the RENAME= data set option in the SET, UPDATE, or MERGE statement, or you can use the DATASETS procedure.

- *common variables with the same data but different attributes*

  The way SAS handles these differences depends on which attributes are different:

  - *type attribute*

    If the type attribute is different, SAS stops processing the DATA step and issues an error message stating that the variables are incompatible.

    To correct this error, you must use a DATA step to re-create the variables. The SAS statements you use depend on the nature of the variable.

- *length attribute*

  If the length attribute is different, SAS takes the length from the first data set that contains the variable. In the following example, all data sets that are listed in the MERGE statement contain the variable Mileage. In QUARTER1, the length of the variable Mileage is four bytes; in QUARTER2, it is eight bytes and in QUARTER3 and QUARTER4, it is six bytes. In the output data set YEARLY, the length of the variable Mileage is four bytes, which is the length derived from QUARTER1.

  ```
  data yearly;
     merge quarter1 quarter2 quarter3 quarter4;
     by Account;
  run;
  ```

  To override the default and set the length yourself, specify the appropriate length in a LENGTH statement that *precedes* the SET, MERGE, MODIFY, or UPDATE statement.

- *label, format, and informat attributes*

  If any of these attributes are different, SAS takes the attribute from the first data set that contains the variable with that attribute. However, any label, format, or informat that you explicitly specify overrides a default. If all data sets contain explicitly specified attributes, the one specified in the first data set overrides the others. To ensure that the new output data set has the attributes you prefer, use an ATTRIB statement.

  You can also use the SAS File I/O functions such as VLABEL, VLABELX, and other Variable Information functions to access this information. For a short description of these functions, see the Variable Information functions in "Functions and CALL Routines by Category" on page 51. For complete information about these functions, see "Functions and CALL Routines" in *SAS Language Reference: Dictionary*.

Ensuring Correct Order

If you use BY-group processing with the UPDATE, SET, and MERGE statements to combine data sets, ensure that the observations in the data sets are sorted in the order of the variables that are listed in the BY statement, or that the data sets have an appropriate index. If you use BY-group processing in a MODIFY statement, your data does not need to be sorted, but sorting the data improves efficiency. The BY variable or variables must be common to both data sets, and they must have the same attributes. For more information, see Chapter 23, "BY-Group Processing in the DATA Step," on page 303.

Testing Your Program

As a final step in preparing your data sets, you should test your program. Create small temporary SAS data sets that contain a sample of observations that test all of your program's logic. If your logic is faulty and you get unexpected output, you can use the DATA step debugger to debug your program. For complete information about the DATA Step Debugger, see *SAS Language Reference: Dictionary*.

Combining SAS Data Sets: Methods

Concatenating

Definition

Concatenating data sets is the combining of two or more data sets, one after the other, into a single data set. The number of observations in the new data set is the sum of the number of observations in the original data sets. The order of observations is sequential. All observations from the first data set are followed by all observations from the second data set, and so on.

In the simplest case, all input data sets contain the same variables. If the input data sets contain different variables, observations from one data set have missing values for variables defined only in other data sets. In either case, the variables in the new data set are the same as the variables in the old data sets.

Syntax

Use this form of the SET statement to concatenate data sets:

SET *data-set(s)*;

where

data-set
: specifies any valid SAS data set name.

For a complete description of the SET statement, see *SAS Language Reference: Dictionary*.

DATA Step Processing During Concatenation

Compilation phase
: SAS reads the descriptor information of each data set that is named in the SET statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step.

Execution — Step 1
: SAS reads the first observation from the first data set into the program data vector. It processes the first observation and executes other statements in the DATA step. It then writes the contents of the program data vector to the new data set. The SET statement does not reset the values in the program data vector to missing, except for variables whose value is calculated or assigned during the DATA step.

Execution — Step 2
: SAS continues to read one observation at a time from the first data set until it finds an end-of-file indicator. The values of the variables in the program data vector are then set to missing, and SAS begins reading observations from the second data set and so forth until it reads all observations from all data sets.

Example 1: Concatenation Using the DATA Step

In this example, each data set contains the variables Common and Number, and the observations are arranged in the order of the values of Common. Generally, you concatenate SAS data sets that have the same variables. In this case, each data set also contains a unique variable to show the effects of combining data sets more clearly. The following shows the ANIMAL and the PLANT input data sets in the library that is referenced by the libref EXAMPLE:

```
         ANIMAL                              PLANT

OBS  Common  Animal  Number       OBS  Common  Plant     Number

 1     a     Ant       5           1     g     Grape       69
 2     b     Bird                  2     h     Hazelnut    55
 3     c     Cat      17           3     i     Indigo
 4     d     Dog       9           4     j     Jicama      14
 5     e     Eagle                 5     k     Kale         5
 6     f     Frog     76           6     l     Lentil      77
```

The following program uses a SET statement to concatenate the data sets and then prints the results:

```
libname example 'SAS-data-library';

data example.concatenation;
   set example.animal example.plant;
run;

proc print data=example.concatenation;
   var Common Animal Plant Number;
   title 'Data Set CONCATENATION';
run;
```

Output 24.1 Concatenated Data Sets (DATA Step)

```
                     Data Set CONCATENATION                          1

             Obs    Common    Animal    Plant       Number

               1      a       Ant                      5
               2      b       Bird                     .
               3      c       Cat                     17
               4      d       Dog                      9
               5      e       Eagle                    .
               6      f       Frog                    76
               7      g                 Grape         69
               8      h                 Hazelnut      55
               9      i                 Indigo         .
              10      j                 Jicama        14
              11      k                 Kale           5
              12      l                 Lentil        77
```

The resulting data set CONCATENATION has 12 observations, which is the sum of the observations from the combined data sets. The program data vector contains all variables from all data sets. The values of variables found in one data set but not in another are set to missing.

Example 2: Concatenation Using SQL

You can also use the SQL language to concatenate tables. In this example, SQL reads each row in both tables and creates a new table named COMBINED. The following shows the YEAR1 and YEAR2 input tables:

```
YEAR1                 YEAR2

Date1                 Date2

1996
1997                  1997
1998                  1998
1999                  1999
                      2000
                      2001
```

The following SQL code creates and prints the table COMBINED.

```
proc sql;
   title 'SQL Table COMBINED';
   create table combined as
      select * from year1
      outer union corr
      select * from year2;
      select * from combined;
quit;
```

Output 24.2 Concatenated Tables (SQL)

```
                              SQL Table COMBINED                         1

                                     Year
                                 --------
                                     1996
                                     1997
                                     1998
                                     1999
                                     1997
                                     1998
                                     1999
                                     2000
                                     2001
```

Appending Files

Instead of concatenating data sets or tables, you can append them and produce the same results as concatenation. SAS concatenates data sets (DATA step) and tables (SQL) by reading each row of data to create a new file. To avoid reading all the records, you can append the second file to the first file by using the APPEND procedure:

```
proc append base=year1 data=year2;
run;
```

The YEAR1 file will contain all rows from both tables.

Note: You cannot use PROC APPEND to add observations to a SAS data set in a sequential library. △

Efficiency

If no additional processing is necessary, using PROC APPEND or the APPEND statement in PROC DATASETS is more efficient than using a DATA step to concatenate data sets.

Interleaving

Definition

Interleaving uses a SET statement and a BY statement to combine multiple data sets into one new data set. The number of observations in the new data set is the sum of the number of observations from the original data sets. However, the observations in the new data set are arranged by the values of the BY variable or variables and, within each BY group, by the order of the data sets in which they occur. You can interleave data sets either by using a BY variable or by using an index.

Syntax

Use this form of the SET statement to interleave data sets when you use a BY variable:

SET *data-set(s)*;

BY *variable(s)*;

where

data-set
: specifies a one-level name, a two-level name, or one of the special SAS data set names.

variable
: specifies each variable by which the data set is sorted. These variables are referred to as BY variables for the current DATA or PROC step.

Use this form of the SET statement to interleave data sets when you use an index:

SET *data-set-1 . . . data-set-n* KEY= *index*;

where

data-set
: specifies a one-level name, a two-level name, or one of the special SAS data set names.

index
: provides nonsequential access to observations in a SAS data set, which are based on the value of an index variable or key.

For a complete description of the SET statement, including SET with the KEY= option, see *SAS Language Reference: Dictionary*.

Sort Requirements

Before you can interleave data sets, the observations must be sorted or grouped by the same variable or variables that you use in the BY statement, or you must have an appropriate index for the data sets.

DATA Step Processing During Interleaving

Compilation phase

- □ SAS reads the descriptor information of each data set that is named in the SET statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step.
- □ SAS creates the FIRST.*variable* and LAST.*variable* for each variable listed in the BY statement.

Execution — Step 1

SAS compares the first observation from each data set that is named in the SET statement to determine which BY group should appear first in the new data set. It reads all observations from the first BY group from the selected data set. If this BY group appears in more than one data set, it reads from the data sets in the order in which they appear in the SET statement. The values of the variables in the program data vector are set to missing each time SAS starts to read a new data set and when the BY group changes.

Execution — Step 2

SAS compares the next observations from each data set to determine the next BY group and then starts reading observations from the selected data set in the SET statement that contains observations for this BY group. SAS continues until it has read all observations from all data sets.

Example 1: Interleaving in the Simplest Case

In this example, each data set contains the BY variable Common, and the observations are arranged in order of the values of the BY variable. The following shows the ANIMAL and the PLANT input data sets in the library that is referenced by the libref EXAMPLE:

```
        ANIMAL                         PLANT

 OBS  Common  Animal         OBS  Common  Plant
  1     a     Ant             1     a     Apple
  2     b     Bird            2     b     Banana
  3     c     Cat             3     c     Coconut
  4     d     Dog             4     d     Dewberry
  5     e     Eagle           5     e     Eggplant
  6     f     Frog            6     f     Fig
```

The following program uses SET and BY statements to interleave the data sets, and prints the results:

```
data example.interleaving;
   set example.animal example.plant;
   by Common;
run;
```

```
proc print data=example.interleaving;
   title 'Data Set INTERLEAVING';
run;
```

Output 24.3 Interleaved Data Sets

```
                          Data Set INTERLEAVING                          1

                  Obs    common    animal    plant

                    1      a       Ant
                    2      a                 Apple
                    3      b       Bird
                    4      b                 Banana
                    5      c       Cat
                    6      c                 Coconut
                    7      d       Dog
                    8      d                 Dewberry
                    9      e       Eagle
                   10      e                 Eggplant
                   11      f       Frog
                   12      f                 Fig
```

The resulting data set INTERLEAVING has 12 observations, which is the sum of the observations from the combined data sets. The new data set contains all variables from both data sets. The value of variables found in one data set but not in the other are set to missing, and the observations are arranged by the values of the BY variable.

Example 2: Interleaving with Duplicate Values of the BY variable

If the data sets contain duplicate values of the BY variables, the observations are written to the new data set in the order in which they occur in the original data sets. This example contains duplicate values of the BY variable Common. The following shows the ANIMAL1 and PLANT1 input data sets:

```
        ANIMAL1                        PLANT1

OBS   Common   Animal1         OBS   Common   Plant1

 1      a       Ant             1      a      Apple
 2      a       Ape             2      b      Banana
 3      b       Bird            3      c      Coconut
 4      c       Cat             4      c      Celery
 5      d       Dog             5      d      Dewberry
 6      e       Eagle           6      e      Eggplant
```

The following program uses SET and BY statements to interleave the data sets, and prints the results:

```
data example.interleaving2;
   set example.animal1 example.plant1;
   by Common;
run;
```

```
proc print data=example.interleaving2;
   title 'Data Set INTERLEAVING2: Duplicate BY Values';
run;
```

Output 24.4 Interleaved Data Sets with Duplicate Values of the BY Variable

```
                 Data Set INTERLEAVING2: Duplicate BY Values               1

                      Obs    Common    Animal1    Plant1

                        1      a        Ant
                        2      a        Ape
                        3      a                   Apple
                        4      b        Bird
                        5      b                   Banana
                        6      c        Cat
                        7      c                   Coconut
                        8      c                   Celery
                        9      d        Dog
                       10      d                   Dewberry
                       11      e        Eagle
                       12      e                   Eggplant
```

The number of observations in the new data set is the sum of the observations in all the data sets. The observations are written to the new data set in the order in which they occur in the original data sets.

Example 3: Interleaving with Different BY Values in Each Data Set

The data sets ANIMAL2 and PLANT2 both contain BY values that are present in one data set but not in the other. The following shows the ANIMAL2 and the PLANT2 input data sets:

```
       ANIMAL2                         PLANT2

OBS  Common  Animal2           OBS  Common  Plant2

 1     a      Ant               1     a     Apple
 2     c      Cat               2     b     Banana
 3     d      Dog               3     c     Coconut
 4     e      Eagle             4     e     Eggplant
                                5     f     Fig
```

This program uses SET and BY statements to interleave these data sets, and prints the results:

```
data example.interleaving3;
   set example.animal2 example.plant2;
   by Common;
run;

proc print data=example.interleaving3;
   title 'Data Set INTERLEAVING3: Different BY Values';
```

```
run;
```

Output 24.5 Interleaving Data Sets with Different BY Values

```
                  Data Set INTERLEAVING3: Different BY Values                  1

                      Obs    Common    Animal2    Plant2

                       1       a        Ant
                       2       a                  Apple
                       3       b                  Banana
                       4       c        Cat
                       5       c                  Coconut
                       6       d        Dog
                       7       e        Eagle
                       8       e                  Eggplant
                       9       f                  Fig
```

The resulting data set has nine observations arranged by the values of the BY variable.

Comments and Comparisons

- □ In other languages, the term *merge* is often used to mean *interleave*. SAS reserves the term *merge* for the operation in which observations from two or more data sets are combined into one observation. The observations in interleaved data sets are not combined; they are copied from the original data sets in the order of the values of the BY variable.
- □ If one table has multiple rows with the same BY value, the DATA step preserves the order of those rows in the result.
- □ To use the DATA step, the input tables must be appropriately sorted or indexed. SQL does not require the input tables to be in order.

One-to-One Reading

Definition

One-to-one reading combines observations from two or more data sets into one observation by using two or more SET statements to read observations independently from each data set. This process is also called *one-to-one matching*. The new data set contains all the variables from all the input data sets. The number of observations in the new data set is the number of observations in the smallest original data set. If the data sets contain common variables, the values that are read in from the last data set replace the values that were read in from earlier data sets.

Syntax

Use this form of the SET statement for one-to-one reading:

SET *data-set-1*;

SET *data-set-2*;

where

data-set-1
: specifies a one-level name, a two-level name, or one of the special SAS data set names. *data-set-1* is the first file that the DATA step reads.

data-set-2
: specifies a one-level name, a two-level name, or one of the special SAS data set names. *data-set-2* is the second file that the DATA step reads.

CAUTION:
Use care when you combine data sets with multiple SET statements. Using multiple SET statements to combine observations can produce undesirable results. Test your program on representative samples of the data sets before using this method to combine them. △

For a complete description of the SET statement, see *SAS Language Reference: Dictionary*.

DATA Step Processing During a One-to-One Reading

Compilation phase
: SAS reads the descriptor information of each data set named in the SET statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step.

Execution — Step 1
: When SAS executes the first SET statement, SAS reads the first observation from the first data set into the program data vector. The second SET statement reads the first observation from the second data set into the program data vector. If both data sets contain the same variables, the values from the second data set replace the values from the first data set, even if the value is missing. After reading the first observation from the last data set and executing any other statements in the DATA step, SAS writes the contents of the program data vector to the new data set. The SET statement does not reset the values in the program data vector to missing, except for those variables that were created or assigned values during the DATA step.

Execution — Step 2
: SAS continues reading from one data set and then the other until it detects an end-of-file indicator in one of the data sets. SAS stops processing with the last observation of the shortest data set and does not read the remaining observations from the longer data set.

Example 1: One-to-One Reading: Processing an Equal Number of Observations

The SAS data sets ANIMAL and PLANT both contain the variable Common, and are arranged by the values of that variable. The following shows the ANIMAL and the PLANT input data sets:

```
       ANIMAL                  PLANT

OBS  Common  Animal      OBS  Common  Plant

 1     a     Ant          1     a     Apple
 2     b     Bird         2     b     Banana
 3     c     Cat           3     c     Coconut
 4     d     Dog           4     d     Dewberry
 5     e     Eagle         5     e     Eggplant
 6     f     Frog          6     g     Fig
```

The following program uses two SET statements to combine observations from ANIMAL and PLANT, and prints the results:

```
data twosets;
   set animal;
   set plant;
run;

proc print data=twosets;
   title 'Data Set TWOSETS - Equal Number of Observations';
run;
```

Output 24.6 Data Set Created from Two Data Sets That Have Equal Observations

```
                 Data Set TWOSETS - Equal Number of Observations               1

                       Obs    Common    Animal    Plant

                        1       a       Ant       Apple
                        2       b       Bird      Banana
                        3       c       Cat       Coconut
                        4       d       Dog       Dewberry
                        5       e       Eagle     Eggplant
                        6       g       Frog      Fig
```

Each observation in the new data set contains all the variables from all the data sets. Note, however, that the Common variable value in observation 6 contains a "g." The value of Common in observation 6 of the ANIMAL data set was overwritten by the value in PLANT, which was the data set that SAS read last.

Comments and Comparisons

- The results that are obtained by reading observations using two or more SET statements are similar to those that are obtained by using the MERGE statement with no BY statement. However, with one-to-one reading, SAS stops processing before all observations are read from all data sets if the number of observations in the data sets is not equal.
- Using multiple SET statements with other DATA step statements makes the following applications possible:
  - merging one observation with many
  - conditionally merging observations
  - reading from the same data set twice.

One-to-One Merging

Definition

One-to-one merging combines observations from two or more SAS data sets into a single observation in a new data set. To perform a one-to-one merge, use the MERGE statement without a BY statement. SAS combines the first observation from all data sets in the MERGE statement into the first observation in the new data set, the second observation from all data sets into the second observation in the new data set, and so on. In a one-to-one merge, the number of observations in the new data set equals the number of observations in the largest data set that was named in the MERGE statement.

If you use the MERGENOBY= SAS system option, you can control whether SAS issues a message when MERGE processing occurs without an associated BY statement.

Syntax

Use this form of the MERGE statement to merge SAS data sets:

MERGE *data-set(s)*;

where

data-set
names at least two existing SAS data sets.

CAUTION:
Avoid using duplicate values or different values of common variables. One-to-one merging with data sets that contain duplicate values of common variables can produce undesirable results. If a variable exists in more than one data set, the value from the last data set that is read is the one that is written to the new data set. The variables are combined exactly as they are read from each data set. Using a one-to-one merge to combine data sets with different values of common variables can also produce undesirable results. If a variable exists in more than one data set, the value from the last data set read is the one that is written to the new data set even if the value is missing. Once SAS has processed all observations in a data set, all subsequent observations in the new data set have missing values for the variables that are unique to that data set. △

For a complete description of the MERGE statement, see *SAS Language Reference: Dictionary*.

DATA Step Processing During One-to-One Merging

Compilation phase
SAS reads the descriptor information of each data set that is named in the MERGE statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step.

Execution — Step 1
SAS reads the first observation from each data set into the program data vector, reading the data sets in the order in which they appear in the MERGE statement. If two data sets contain the same variables, the values from the second data set

replace the values from the first data set. After reading the first observation from the last data set and executing any other statements in the DATA step, SAS writes the contents of the program data vector to the new data set. Only those variables that are created or assigned values during the DATA step are set to missing.

Execution — Step 2
SAS continues until it has read all observations from all data sets.

Example 1: One-to-One Merging with an Equal Number of Observations

The SAS data sets ANIMAL and PLANT both contain the variable Common, and the observations are arranged by the values of Common. The following shows the ANIMAL and the PLANT input data sets:

```
       ANIMAL                    PLANT

OBS  Common  Animal       OBS   Common  Plant

 1     a     Ant           1      a     Apple
 2     b     Bird          2      b     Banana
 3     c     Cat           3      c     Coconut
 4     d     Dog           4      d     Dewberry
 5     e     Eagle         5      e     Eggplant
 6     f     Frog          6      g     Fig
```

The following program merges these data sets and prints the results:

```
data combined;
    merge animal plant;
run;

proc print data=combined;
    title 'Data Set COMBINED';
run;
```

Output 24.7 Merged Data Sets That Have an Equal Number of Observations

```
                        Data Set COMBINED                        1

               Obs    Common    Animal    Plant

                1       a       Ant       Apple
                2       b       Bird      Banana
                3       c       Cat       Coconut
                4       d       Dog       Dewberry
                5       e       Eagle     Eggplant
                6       g       Frog      Fig
```

Each observation in the new data set contains all variables from all data sets. If two data sets contain the same variables, the values from the second data set replace the values from the first data set, as shown in observation 6.

Example 2: One-to-One Merging with an Unequal Number of Observations

The SAS data sets ANIMAL1 and PLANT1 both contain the variable Common, and the observations are arranged by the values of Common. The PLANT1 data set has fewer observations than the ANIMAL1 data set. The following shows the ANIMAL1 and the PLANT1 input data sets:

```
        ANIMAL1                    PLANT1

OBS  Common  Animal       OBS   Common  Plant

 1     a     Ant           1      a     Apple
 2     b     Bird          2      b     Banana
 3     c     Cat           3      c     Coconut
 4     d     Dog
 5     e     Eagle
 6     f     Frog
```

The following program merges these unequal data sets and prints the results:

```
data combined1;
    merge animal1 plant1;
run;

proc print data=combined1;
    title 'Data Set COMBINED1';
run;
```

Output 24.8 Merged Data Sets That Have an Unequal Number of Observations

```
                    Data Set COMBINED1                          1

            Obs    Common    Animal     Plant

             1       a       Ant       Apple
             2       b       Bird      Banana
             3       c       Cat       Coconut
             4       d       Dog
             5       e       Eagle
             6       f       Frog
```

Note that observations 4 through 6 contain missing values for the variable Plant.

Example 3: One-to-One Merging with Duplicate Values of Common Variables

The following example shows the undesirable results that you can obtain by using one-to-one merging with data sets that contain duplicate values of common variables. The value from the last data set that is read is the one that is written to the new data set. The variables are combined exactly as they are read from each data set. In the following example, the data sets ANIMAL1 and PLANT1 contain the variable Common, and each data set contains observations with duplicate values of Common. The following shows the ANIMAL1 and the PLANT1 input data sets:

```
       ANIMAL1                  PLANT1

OBS  Common  Animal       OBS   Common  Plant

 1     a     Ant           1      a     Apple
 2     a     Ape           2      b     Banana
 3     b     Bird          3      c     Coconut
 4     c     Cat           4      c     Celery
 5     d     Dog           5      d     Dewberry
 6     e     Eagle         6      e     Eggplant
```

The following program produces the data set MERGE1 data set and prints the results:

```
   /* This program illustrates undesirable results. */
data merge1;
   merge animal1 plant1;
run;

proc print data=merge1;
   title 'Data Set MERGE1';
run;
```

Output 24.9 Undesirable Results with Duplicate Values of Common Variables

```
                          Data Set MERGE1                              1

                  Obs    Common    Animal1    Plant1

                   1       a       Ant       Apple
                   2       b       Ape       Banana
                   3       c       Bird      Coconut
                   4       c       Cat       Celery
                   5       d       Dog       Dewberry
                   6       e       Eagle     Eggplant
```

The number of observations in the new data set is six. Note that observations 2 and 3 contain undesirable values. SAS reads the second observation from data set ANIMAL1. It then reads the second observation from data set PLANT1 and replaces the values for the variables Common and Plant1. The third observation is created in the same way.

Example 4: One-to-One Merging with Different Values of Common Variables

The following example shows the undesirable results obtained from using the one-to-one merge to combine data sets with different values of common variables. If a variable exists in more than one data set, the value from the last data set that is read is the one that is written to the new data set even if the value is missing. Once SAS processes all observations in a data set, all subsequent observations in the new data set have missing values for the variables that are unique to that data set. In this example, the data sets ANIMAL2 and PLANT2 have different values of the Common variable. The following shows the ANIMAL2 and the PLANT2 input data sets:

```
        ANIMAL2                    PLANT2

OBS  Common  Animal       OBS   Common  Plant

 1     a     Ant           1      a     Apple
 2     c     Cat           2      b     Banana
 3     d     Dog           3      c     Coconut
 4     e     Eagle         4      e     Eggplant
                           5      f     Fig
```

The following program produces the data set MERGE2 and prints the results:

```
   /* This program illustrates undesirable results. */
data merge2;
   merge animal2 plant2;
run;

proc print data=merge2;
   title 'Data Set MERGE2';
run;
```

Output 24.10 Undesirable Results with Different Values of Common Variables

```
                            Data Set MERGE2                          1

                  Obs    Common    Animal2    Plant2

                   1       a        Ant       Apple
                   2       b        Cat       Banana
                   3       c        Dog       Coconut
                   4       e        Eagle     Eggplant
                   5       f                  Fig
```

Comments and Comparisons

The results from a one-to-one merge are similar to the results obtained from using two or more SET statements to combine observations. However, with the one-to-one merge, SAS continues processing all observations in all data sets that were named in the MERGE statement.

Match-Merging

Definition

Match-merging combines observations from two or more SAS data sets into a single observation in a new data set according to the values of a common variable. The number of observations in the new data set is the sum of the largest number of observations in each BY group in all data sets. To perform a match-merge, use the MERGE statement with a BY statement. Before you can perform a match-merge, all data sets must be sorted by the variables that you specify in the BY statement or they must have an index.

Syntax

Use this form of the MERGE statement to match-merge data sets:

MERGE *data-set(s)*;

BY *variable(s)*;

where

data-set
: names at least two existing SAS data sets from which observations are read.

variable
: names each variable by which the data set is sorted or indexed. These variables are referred to as BY variables.

For a complete description of the MERGE and the BY statements, see *SAS Language Reference: Dictionary*.

DATA Step Processing During Match-Merging

Compilation phase
: SAS reads the descriptor information of each data set that is named in the MERGE statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step. SAS creates the FIRST.*variable* and LAST.*variable* for each variable that is listed in the BY statement.

Execution – Step 1
: SAS looks at the first BY group in each data set that is named in the MERGE statement to determine which BY group should appear first in the new data set. The DATA step reads into the program data vector the first observation in that BY group from each data set, reading the data sets in the order in which they appear in the MERGE statement. If a data set does not have observations in that BY group, the program data vector contains missing values for the variables unique to that data set.

Execution – Step 2
: After processing the first observation from the last data set and executing other statements, SAS writes the contents of the program data vector to the new data set. SAS retains the values of all variables in the program data vector except those variables that were created by the DATA step; SAS sets those values to missing. SAS continues to merge observations until it writes all observations from the first BY group to the new data set. When SAS has read all observations in a BY group from all data sets, it sets all variables in the program data vector to missing. SAS looks at the next BY group in each data set to determine which BY group should appear next in the new data set.

Execution – Step 3
: SAS repeats these steps until it reads all observations from all BY groups in all data sets.

Example 1: Combining Observations Based on a Criterion

The SAS data sets ANIMAL and PLANT each contain the BY variable Common, and the observations are arranged in order of the values of the BY variable. The following shows the ANIMAL and the PLANT input data sets:

```
        ANIMAL                    PLANT

OBS  Common  Animal        OBS  Common  Plant

 1     a     Ant            1     a     Apple
 2     b     Bird           2     b     Banana
 3     c     Cat            3     c     Coconut
 4     d     Dog            4     d     Dewberry
 5     e     Eagle          5     e     Eggplant
 6     f     Frog           6     f     Fig
```

The following program merges the data sets according to the values of the BY variable Common, and prints the results:

```
data combined;
    merge animal plant;
    by Common;
run;

proc print data=combined;
   title 'Data Set COMBINED';
run;
```

Output 24.11 Data Sets Combined by Match-Merging

```
                         Data Set COMBINED                            1

                 Obs    Common    Animal    Plant

                  1       a       Ant       Apple
                  2       b       Bird      Banana
                  3       c       Cat       Coconut
                  4       d       Dog       Dewberry
                  5       e       Eagle     Eggplant
                  6       f       Frog      Fig
```

Each observation in the new data set contains all the variables from all the data sets.

Example 2: Match-Merge with Duplicate Values of the BY Variable

When SAS reads the last observation from a BY group in one data set, SAS retains its values in the program data vector for all variables that are unique to that data set until all observations for that BY group have been read from all data sets. In the following example, the data sets ANIMAL1 and PLANT1 contain duplicate values of the BY variable Common. The following shows the ANIMAL1 and the PLANT1 input data sets:

```
       ANIMAL1                     PLANT1
```

```
OBS  Common  Animal1         OBS  Common  Plant1

 1     a      Ant             1     a     Apple
 2     a      Ape             2     b     Banana
 3     b      Bird            3     c     Coconut
 4     c      Cat             4     c     Celery
 5     d      Dog             5     d     Dewberry
 6     e      Eagle           6     e     Eggplant
```

The following program produces the merged data set MATCH1, and prints the results:

```
data match1;
   merge animal1 plant1;
   by Common;
run;

proc print data=match1;
   title 'Data Set MATCH1';
run;
```

Output 24.12 Match-Merged Data Set with Duplicate BY Values

```
                          Data Set MATCH1                             1

                  Obs    Common    Animal1    Plant1

                   1       a        Ant       Apple
                   2       a        Ape       Apple
                   3       b        Bird      Banana
                   4       c        Cat       Coconut
                   5       c        Cat       Celery
                   6       d        Dog       Dewberry
                   7       e        Eagle     Eggplant
```

In observation 2 of the output, the value of the variable Plant1 is retained until all observations in the BY group are written to the new data set. Match-merging also produced duplicate values in ANIMAL1 for observations 4 and 5.

Example 3: Match-Merge with Nonmatched Observations

When SAS performs a match-merge with nonmatched observations in the input data sets, SAS retains the values of all variables in the program data vector even if the value is missing. The data sets ANIMAL2 and PLANT2 do not contain all values of the BY variable Common. The following shows the ANIMAL2 and the PLANT2 input data sets:

```
       ANIMAL2                    PLANT2

OBS  Common  Animal2       OBS  Common  Plant2

 1     a      Ant           1     a     Apple
 2     c      Cat           2     b     Banana
 3     d      Dog           3     c     Coconut
 4     e      Eagle         4     e     Eggplant
                            5     f     Fig
```

The following program produces the merged data set MATCH2, and prints the results:

```
data match2;
   merge animal2 plant2;
   by Common;
run;

proc print data=match2;
   title 'Data Set MATCH2';
run;
```

Output 24.13 Match-Merged Data Set with Nonmatched Observations

```
                        Data Set MATCH2                              1

               Obs    Common    Animal2    Plant2

                1       a        Ant       Apple
                2       b                  Banana
                3       c        Cat       Coconut
                4       d        Dog
                5       e        Eagle     Eggplant
                6       f                  Fig
```

As the output shows, all values of the variable Common are represented in the new data set, including missing values for the variables that are in one data set but not in the other.

Updating with the UPDATE and the MODIFY Statements

Definitions

Updating a data set refers to the process of applying changes to a master data set. To update data sets, you work with two input data sets. The data set containing the original information is the *master data set*, and the data set containing the new information is the *transaction data set*.

You can update data sets by using the UPDATE statement or the MODIFY statement:

UPDATE	uses observations from the transaction data set to change the values of corresponding observations from the master data set. You must use a BY statement with the UPDATE statement because all observations in the transaction data set are keyed to observations in the master data set according to the values of the BY variable.
MODIFY	can replace, delete, and append observations in an existing data set. Using the MODIFY statement can save disk space because it modifies data in place, without creating a copy of the data set.

The number of observations in the new data set is the sum of the number of observations in the master data set and the number of unmatched observations in the transaction data set.

For complete information about the UPDATE and the MODIFY statements, see "Statements" in *SAS Language Reference: Dictionary*.

Syntax of the UPDATE Statement

Use this form of the UPDATE statement to update a master data set:

UPDATE *master-data-set transaction-data-set*;

BY *variable-list*;

where

master-data-set
names the SAS data set that is used as the master file.

transaction-data-set
names the SAS data set that contains the changes to be applied to the master data set.

variable-list
specifies the variables by which observations are matched.

If the transaction data set contains duplicate values of the BY variable, SAS applies both transactions to the observation. The last values that are copied into the program data vector are written to the new data set. If your data is in this form, use the MODIFY statement instead of the UPDATE statement to process your data.

CAUTION:
Values of the BY variable must be unique for each observation in the master data set. If the master data set contains two observations with the same value of the BY variable, the first observation is updated and the second observation is ignored. SAS writes a warning message to the log when the DATA step executes. △

For complete information about the UPDATE statement, see *SAS Language Reference: Dictionary*.

Syntax of the MODIFY Statement

This form of the MODIFY statement is used in the examples that follow:

MODIFY *master-data–set*;

BY *variable-list*;

where

master-data–set
specifies the SAS data set that you want to modify.

variable-list
names each variable by which the data set is ordered.

Note: The MODIFY statement does not support changing the descriptor portion of a SAS data set, such as adding a variable. △

For complete information about the MODIFY statement, see *SAS Language Reference: Dictionary*.

DATA Step Processing with the UPDATE Statement

Compilation phase

- □ SAS reads the descriptor information of each data set that is named in the UPDATE statement and creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step.
- □ SAS creates the FIRST.*variable* and LAST.*variable* for each variable that is listed in the BY statement.

Execution – Step 1

SAS looks at the first observation in each data set that is named in the UPDATE statement to determine which BY group should appear first. If the transaction BY value precedes the master BY value, SAS reads from the transaction data set only and sets the variables from the master data set to missing. If the master BY value precedes the transaction BY value, SAS reads from the master data set only and sets the unique variables from the transaction data set to missing. If the BY values in the master and transaction data sets are equal, it applies the first transaction by copying the nonmissing values into the program data vector.

Execution – Step 2

After completing the first transaction, SAS looks at the next observation in the transaction data set. If SAS finds one with the same BY value, it applies that transaction too. The first observation then contains the new values from both transactions. If no other transactions exist for that observation, SAS writes the observation to the new data set and sets the values in the program data vector to missing. SAS repeats these steps until it has read all observations from all BY groups in both data sets.

Updating with Nonmatched Observations, Missing Values, and New Variables

In the UPDATE statement, if an observation in the master data set does not have a corresponding observation in the transaction data set, SAS writes the observation to the new data set without modifying it. Any observation from the transaction data set that does not correspond to an observation in the master data set is written to the program data vector and becomes the basis for an observation in the new data set. The data in the program data vector can be modified by other transactions before it is written to the new data set. If a master data set observation does not need updating, the corresponding observation can be omitted from the transaction data set.

SAS does not replace existing values in the master data set with missing values if those values are coded as periods (for numeric variables) or blanks (for character variables) in the transaction data set. To replace existing values with missing values, you must either create a transaction data set in which missing values are coded with the special missing value characters, or use the UPDATEMODE=NOMISSINGCHECK statement option.

With UPDATE, the transaction data set can contain new variables to be added to all observations in the master data set.

To view a sample program, see "Example 3: Using UPDATE for Processing Nonmatched Observations, Missing Values, and New Variables" on page 354.

Sort Requirements for the UPDATE Statement

If you do not use an index, both the master data set and the transaction data set must be sorted by the same variable or variables that you specify in the BY statement that accompanies the UPDATE statement. The values of the BY variable should be unique for each observation in the master data set. If you use more than one BY variable, the combination of values of all BY variables should be unique for each observation in the master data set. The BY variable or variables should be ones that you never need to update.

Note: The MODIFY statement does not require sorted files. However, sorting the data improves efficiency. △

Using an Index with the MODIFY Statement

The MODIFY statement maintains the index. You do not have to rebuild the index like you do for the UPDATE statement.

Choosing between UPDATE or MODIFY with BY

Using the UPDATE statement is comparable to using MODIFY with BY to apply transactions to a data set. While MODIFY is a more powerful tool with several other applications, UPDATE is still the tool of choice in some cases. The following table helps you choose whether to use UPDATE or MODIFY with BY.

Table 24.3 MODIFY with BY versus UPDATE

Issue	MODIFY with BY	UPDATE
Disk space	saves disk space because it updates data in place	requires more disk space because it produces an updated copy of the data set
Sort and index	sorted input data sets are not required, although for good performance, it is strongly recommended that both data sets be sorted and that the master data set be indexed	requires only that both data sets be sorted
When to use	use only when you expect to process a SMALL portion of the data set	use if you expect to need to process most of the data set
Where to specify the modified data set	specify the updated data set in both the DATA and the MODIFY statements	specify the updated data set in the DATA and the UPDATE statements
Duplicate BY-values	allows duplicate BY-values in both the master and the transaction data sets	allows duplicate BY-values in the transaction data set only (If duplicates exist in the master data set, SAS issues a warning.)
Scope of changes	cannot change the data set descriptor information, so changes such as adding or deleting variables, variable labels, and so on, are not valid	can make changes that require a change in the descriptor portion of a data set, such as adding new variables, and so on

Issue	MODIFY with BY	UPDATE
Error checking	has error-checking capabilities using the _IORC_ automatic variable and the SYSRC autocall macro	needs no error checking because transactions without a corresponding master record are not applied but are added to the data set
Data set integrity	data may only be partially updated due to an abnormal task termination	no data loss occurs because UPDATE works on a copy of the data

For more information about tools for combining SAS data sets, see Table 24.2 on page 327.

Primary Uses of the MODIFY Statement

The MODIFY statement has three primary uses:

- □ modifying observations in a single SAS data set.
- □ modifying observations in a single SAS data set directly, either by observation number or by values in an index.
- □ modifying observations in a master data set, based on values in a transaction data set. MODIFY with BY is similar to using the UPDATE statement.

Several of the examples that follow demonstrate these uses.

Example 1: Using UPDATE for Basic Updating

In this example, the data set MASTER contains original values of the variables Animal and Plant. The data set NEWPLANT is a transaction data set with new values of the variable Plant. The following shows the MASTER and the NEWPLANT input data sets:

```
       MASTER                          NEWPLANT

OBS Common Animal Plant          OBS Common Plant

 1     a    Ant    Apple          1     a    Apricot
 2     b    Bird   Banana         2     b    Barley
 3     c    Cat    Coconut        3     c    Cactus
 4     d    Dog    Dewberry       4     d    Date
 5     e    Eagle  Eggplant       5     e    Escarole
 6     f    Frog   Fig            6     f    Fennel
```

The following program updates MASTER with the transactions in the data set NEWPLANT, writes the results to UPDATE_FILE, and prints the results:

```
data update_file;
   update master newplant;
   by common;
run;
```

```
proc print data=update_file;
   title 'Data Set Update_File';
run;
```

Output 24.14 Master Data Set Updated by Transaction Data Set

```
                     Data Set Update_File                        1

                Obs    Common    Animal    Plant

                 1       a       Ant       Apricot
                 2       b       Bird      Barley
                 3       c       Cat       Cactus
                 4       d       Dog       Date
                 5       e       Eagle     Escarole
                 6       f       Frog      Fennel
```

Each observation in the new data set contains a new value for the variable Plant.

Example 2: Using UPDATE with Duplicate Values of the BY Variable

If the master data set contains two observations with the same value of the BY variable, the first observation is updated and the second observation is ignored. SAS writes a warning message to the log. If the transaction data set contains duplicate values of the BY variable, SAS applies both transactions to the observation. The last values copied into the program data vector are written to the new data set. The following shows the MASTER1 and the DUPPLANT input data sets.

```
         MASTER1                               DUPPLANT

OBS Common Animal1 Plant1             OBS Common Plant1

 1    a     Ant     Apple              1    a    Apricot
 2    b     Bird    Banana             2    b    Barley
 3    b     Bird    Banana             3    c    Cactus
 4    c     Cat     Coconut            4    d    Date
 5    d     Dog     Dewberry           5    d    Dill
 6    e     Eagle   Eggplant           6    e    Escarole
 7    f     Frog    Fig                7    f    Fennel
```

The following program applies the transactions in DUPPLANT to MASTER1 and prints the results:

```
data update1;
   update master1 dupplant;
   by Common;
run;

proc print data=update1;
   title 'Data Set Update1';
run;
```

Output 24.15 Updating Data Sets with Duplicate BY Values

```
                         Data Set Update1                              1

               Obs    Common    Animal1    Plant1

                1       a        Ant       Apricot
                2       b        Bird      Barley
                3       b        Bird      Banana
                4       c        Cat       Cactus
                5       d        Dog       Dill
                6       e        Eagle     Escarole
                7       f        Frog      Fennel
```

When this DATA step executes, SAS generates a warning message stating that there is more than one observation for a BY group. However, the DATA step continues to process, and the data set UPDATE1 is created.

The resulting data set has seven observations. Observations 2 and 3 have duplicate values of the BY variable Common. However, the value of the variable PLANT1 was not updated in the second occurrence of the duplicate BY value.

Example 3: Using UPDATE for Processing Nonmatched Observations, Missing Values, and New Variables

In this example, the data set MASTER2 is a master data set. It contains a missing value for the variable Plant2 in the first observation, and not all of the values of the BY variable Common are included. The transaction data set NONPLANT contains a new variable Mineral, a new value of the BY variable Common, and missing values for several observations. The following shows the MASTER2 and the NONPLANT input data sets:

```
           MASTER2                                NONPLANT

OBS  Common  Animal2  Plant2        OBS  Common  Plant2   Mineral

 1     a     Ant                     1     a     Apricot  Amethyst
 2     c     Cat      Coconut        2     b     Barley   Beryl
 3     d     Dog      Dewberry       3     c     Cactus
 4     e     Eagle    Eggplant       4     e
 5     f     Frog     Fig            5     f     Fennel
                                     6     g     Grape    Garnet
```

The following program updates the data set MASTER2 and prints the results:

```
data update2_file;
   update master2 nonplant;
   by Common;
run;

proc print data=update2_file;
   title 'Data Set Update2_File';
run;
```

Output 24.16 Results of Updating with New Variables, Nonmatched Observations, and Missing Values

```
                    Data Set Update2_File                          1

        Obs    Common    Animal2    Plant2      Mineral

         1       a        Ant       Apricot     Amethyst
         2       b                  Barley      Beryl
         3       c        Cat       Cactus
         4       d        Dog       Dewberry
         5       e        Eagle     Eggplant
         6       f        Frog      Fennel
         7       g                  Grape       Garnet
```

As shown, all observations now include values for the variable Mineral. The value of Mineral is set to missing for some observations. Observations 2 and 6 in the transaction data set did not have corresponding observations in MASTER2, and they have become new observations. Observation 3 from the master data set was written to the new data set without change, and the value for Plant2 in observation 4 was not changed to missing. Three observations in the new data set have updated values for the variable Plant2.

The following program uses the UPDATEMODE statement option on the UPDATE statement, and prints the results:

```
data update2_file;
   update master2 nonplant updatemode=nomissingcheck;
   by Common;
run;

proc print data=update2_file;
   title 'Data Set Update2_File - UPDATEMODE Option';
run;
```

Output 24.17 Results of Updating with the UPDATEMODE Option

```
               Data Set Update2_File - UPDATEMODE Option           1

        Obs    Common    Animal2    Plant2      Mineral

         1       a        Ant       Apricot     Amethyst
         2       b                  Barley      Beryl
         3       c        Cat       Cactus
         4       d        Dog       Dewberry
         5       e        Eagle
         6       f        Frog      Fennel
         7       g                  Grape       Garnet
```

The value of Plant2 in observation 5 is set to missing because the UPDATEMODE=NOMISSINGCHECK option is in effect.

For detailed examples for updating data sets, see *Combining and Modifying SAS Data Sets: Examples.*

Example 4: Updating a MASTER Data Set by Adding an Observation

If the transaction data set contains an observation that does not match an observation in the master data set, you must alter the program. The Year value in observation 5 of TRANSACTION has no match in MASTER. The following shows the MASTER and the TRANSACTION input data sets:

```
          MASTER                          TRANSACTION

OBS  Year   VarX   VarY        OBS  Year   VarX   VarY

  1  1985    x1     y1           1  1991    x2
  2  1986    x1     y1           2  1992    x2      y2
  3  1987    x1     y1           3  1993    x2
  4  1988    x1     y1           4  1993            y2
  5  1989    x1     y1           5  1995    x2      y2
  6  1990    x1     y1
  7  1991    x1     y1
  8  1992    x1     y1
  9  1993    x1     y1
 10  1994    x1     y1
```

You must use an explicit OUTPUT statement to write a new observation to a master data set. (The default action for a DATA step using a MODIFY statement is REPLACE, not OUTPUT.) Once you specify an explicit OUTPUT statement, you must also specify a REPLACE statement. The following DATA step updates data set MASTER, based on values in TRANSACTION, and adds a new observation. This program also uses the _IORC_ automatic variable for error checking. (For more information about error checking, see "Error Checking When Using Indexes to Randomly Access or Update Data" on page 357.

```
data master;
   modify master transaction;
   by Year;
   if _iorc_=%sysrc(_sok) then replace;
   else if _iorc_=%sysrc(_dsenmr) then
      do;
         output;
         _error_=0;
      end;
   else
      do;
         put "Unexpected error at Observation: " _n_;
         _error_=0;
         stop;
      end;
run;

proc print data=master;
   title 'Updated Master Data Set -- MODIFY';
   title2 'One Observation Added';
run;
```

Output 24.18 Modified MASTER Data Set

```
                     Updated Master Data Set -- MODIFY                        1
                           One Observation Added

                        Obs    Year    VarX    VarY

                          1    1985     x1      y1
                          2    1986     x1      y1
                          3    1987     x1      y1
                          4    1988     x1      y1
                          5    1989     x1      y1
                          6    1990     x1      y1
                          7    1991     x2      y1
                          8    1992     x2      y2
                          9    1993     x2      y2
                         10    1994     x1      y1
                         11    1995     x2      y2
```

SAS added a new observation, observation 11, to the MASTER data set and updated observations 7, 8, and 9.

Error Checking When Using Indexes to Randomly Access or Update Data

The Importance of Error Checking

When reading observations with the SET statement and KEY= option or with the MODIFY statement, error checking is imperative for several reasons. The most important reason is that these tools use nonsequential access methods, and so there is no guarantee that an observation will be located that satisfies the request. Error checking enables you to direct execution to specific code paths, depending on the outcome of the I/O operation. Your program will continue execution for expected conditions and terminate execution when unexpected results occur.

Error-Checking Tools

Two tools have been created to make error checking easier when you use the MODIFY statement or the SET statement with the KEY= option to process SAS data sets:

- _IORC_ automatic variable
- SYSRC autocall macro.

IORC is created automatically when you use the MODIFY statement or the SET statement with KEY=. The value of _IORC_ is a numeric return code that indicates the status of the I/O operation from the most recently executed MODIFY or SET statement with KEY=. Checking the value of this variable enables you to detect abnormal I/O conditions and to direct execution down specific code paths instead of having the application terminate abnormally. For example, if the KEY= variable value does match between two observations, you might want to combine them and output an observation. If they don't match, however, you may want only to write a note to the log.

Because the values of the _IORC_ automatic variable are internal and subject to change, the SYSRC macro was created to enable you to test for specific I/O conditions while protecting your code from future changes in _IORC_ values. When you use

SYSRC, you can check the value of _IORC_ by specifying one of the mnemonics listed in the following table.

Table 24.4 Most Common Mnemonic Values of _IORC_ for DATA Step Processing

Mnemonic value	Meaning of return code	This return code occurs when ...
_DSENMR	The TRANSACTION data set observation does not exist in the MASTER data set.	MODIFY with BY is used and no match occurs.
_DSEMTR	Multiple TRANSACTION data set observations with the same BY variable value do not exist in the MASTER data set.	MODIFY with BY is used and consecutive observations with the same BY values do not find a match in the first data set. In this situation, the first observation that fails to find a match returns _DSENMR. The subsequent observations return _DSEMTR.
_DSENOM	No matching observation was found in the MASTER data set.	SET or MODIFY with KEY= finds no match.
_SENOCHN	The output operation was unsuccessful.	the KEY= option in a MODIFY statement contains duplicate values.
_SOK	The I/O operation was successful.	a match is found.

Example 1: Routing Execution When an Unexpected Condition Occurs

Overview

This example shows how to prevent an unexpected condition from terminating the DATA step. The goal is to update a master data set with new information from a transaction data set. This application assumes that there are no duplicate values for the common variable in either data set.

Note: This program works as expected only if the master and transaction data sets contain no consecutive observations with the same value for the common variable. For an explanation of the behavior of MODIFY with KEY= when duplicates exist, see the MODIFY statement in *SAS Language Reference: Dictionary*. △

Input Data Sets

The TRANSACTION data set contains three observations: two updates to information in MASTER and a new observation about PartNumber value 6 that needs to be added. MASTER is indexed on PartNumber. There are no duplicate values of PartNumber in MASTER or TRANSACTION. The following shows the MASTER and the TRANSACTION input data sets:

```
            MASTER                                  TRANSACTION
```

```
OBS   PartNumber   Quantity      OBS    PartNumber    AddQuantity

 1        1           10          1         4              14
 2        2           20          2         6              16
 3        3           30          3         2              12
 4        4           40
 5        5           50
```

Original Program

The objective is to update the MASTER data set with information from the TRANSACTION data set. The program reads TRANSACTION sequentially. MASTER is read directly, not sequentially, using the MODIFY statement and the KEY= option. Only observations with matching values for PartNumber, which is the KEY= variable, are read from MASTER.

```
data master;  ❶
   set transaction;  ❷
   modify master key=PartNumber;  ❸
   Quantity = Quantity + AddQuantity;  ❹
run;
```

❶ Open the MASTER data set for update.

❷ Read an observation from the TRANSACTION data set.

❸ Match observations from the MASTER data set based on the values of PartNumber.

❹ Update the information on Quantity by adding the new values from the TRANSACTION data set.

Resulting Log

This program has correctly updated one observation but it stopped when it could not find a match for PartNumber value 6. The following lines are written to the SAS log:

```
ERROR: No matching observation was found in MASTER data set.
PartNumber=6 AddQuantity=16 Quantity=70 _ERROR_=1
_IORC_=1230015 _N_=2
NOTE: The SAS System stopped processing this step because
      of errors.
NOTE: The data set WORK.MASTER has been updated.  There were
      1 observations rewritten, 0 observations added and 0
      observations deleted.
```

Resulting Data Set

The MASTER file was incorrectly updated. The updated master has five observations. One observation was updated correctly, a new one was not added, and a second update was not made. The following shows the incorrectly updated MASTER data set:

```
          MASTER

OBS    PartNumber    Quantity
```

```
 1          1            10
 2          2            20
 3          3            30
 4          4            54
 5          5            50
```

Revised Program

The objective is to apply two updates and one addition to MASTER, preventing the DATA step from stopping when it does not find a match in MASTER for the PartNumber value 6 in TRANSACTION. By adding error checking, this DATA step is allowed to complete normally and produce a correctly revised version of MASTER. This program uses the _IORC_ automatic variable and the SYSRC autocall macro in a SELECT group to check the value of the _IORC_ variable and execute the appropriate code based on whether or not a match is found.

```
data master; ❶
   set transaction;  ❷
   modify master key=PartNumber;  ❸
select(_iorc_);  ❹
      when(%sysrc(_sok)) do;
         Quantity = Quantity + AddQuantity;
         replace;
      end;
      when(%sysrc(_dsenom)) do;
         Quantity = AddQuantity;
         _error_ = 0;
         output;
      end;
      otherwise do;
         put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
         put 'Program terminating. Data step iteration # ' _n_;
         put _all_;
         stop;
      end;
   end;
run;
```

❶ Open the MASTER data set for update.

❷ Read an observation from the TRANSACTION data set.

❸ Match observations from the MASTER data set based on the value of PartNumber.

❹ Take the correct course of action based on whether a matching value for PartNumber is found in MASTER. Update Quantity by adding the new values from TRANSACTION. The SELECT group directs execution to the correct code. When a match occurs (_SOK), update Quantity and replace the original observation in MASTER. When there is no match (_DSENOM), set Quantity equal to the AddQuantity amount from TRANSACTION, and append a new observation. _ERROR_ is reset to 0 to prevent an error condition that would write the contents of the program data vector to the SAS log. When an unexpected condition occurs, write messages and the contents of the program data vector to the log, and stop the DATA step.

Resulting Log

The DATA step executed without error and observations were appropriately updated and added. The following lines are written to the SAS log:

```
NOTE: The data set WORK.MASTER has been updated.  There were
      2 observations rewritten, 1 observations added and 0
      observations deleted.
```

Correctly Updated MASTER Data Set

MASTER contains updated quantities for PartNumber values 2 and 4 and a new observation for PartNumber value 6. The following shows the correctly updated MASTER data set:

```
            MASTER

OBS    PartNumber    Quantity
 1          1           10
 2          2           32
 3          3           30
 4          4           54
 5          5           50
 6          6           16
```

Example 2: Using Error Checking on All Statements That Use KEY=

Overview

This example shows how important it is to use error checking on all statements that use the KEY= option when reading data.

Input Data Sets

The MASTER and DESCRIPTION data sets are both indexed on PartNumber. The ORDER data set contains values for all parts in a single order. Only ORDER contains the PartNumber value 8. The following shows the MASTER, ORDER, and DESCRIPTION input data sets:

```
            MASTER                              ORDER

OBS    PartNumber    Quantity          OBS    PartNumber

 1         100          10              1         200
 2         200          20              2         400
 3         300          30              3         100
 4         400          40              4         300
 5         500          50              5         800
                                        6         500
                                        7         600
```

```
              DESCRIPTION

  OBS    PartNumber    PartDescription

   1        400          Nuts
   2        300          Bolts
   3        200          Screws
   4        600          Washers
```

Original Program with Logic Error

The objective is to create a data set that contains the description and number in stock for each part in a single order, except for the parts that are not found in either of the two input data sets, MASTER and DESCRIPTION. A transaction data set contains the part numbers of all parts in a single order. One data set is read to retrieve the description of the part and another is read to retrieve the quantity that is in stock.

The program reads the ORDER data set sequentially and then uses SET with the KEY= option to read the MASTER and DESCRIPTION data sets directly, based on the key value of PartNumber. When a match occurs, an observation is written that contains all the necessary information for each value of PartNumber in ORDER. This first attempt at a solution uses error checking for only one of the two SET statements that use KEY= to read a data set.

```
data combine; ❶
   length PartDescription $ 15;
   set order;  ❷
   set description key=PartNumber;  ❷
   set master key=PartNumber;  ❷
   select(_iorc_); ❸
         when(%sysrc(_sok)) do;
            output;
         end;
         when(%sysrc(_dsenom)) do;
            PartDescription = 'No description';
            _error_ = 0;
            output;
         end;
         otherwise do;
            put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
            put 'Program terminating.';
            put _all_;
            stop;
         end;
   end;
run;
```

❶ Create the COMBINE data set.

❷ Read an observation from the ORDER data set. Read an observation from the DESCRIPTION and the MASTER data sets based on a matching value for PartNumber, the key variable. Note that no error checking occurs after an observation is read from DESCRIPTION.

❸ Take the correct course of action, based on whether a matching value for PartNumber is found in MASTER or DESCRIPTION. (This logic is based on the erroneous assumption that this SELECT group performs error checking for both of the preceding SET statements that contain the KEY= option. It actually performs error checking for only the most recent one.) The SELECT group directs execution to the correct code. When a match occurs (_SOK), the value of PartNumber in the observation that is being read from MASTER matches the current PartNumber value from ORDER. So, output an observation. When there is no match (_DSENOM), no observations in MASTER contain the current value of PartNumber, so set the value of PartDescription appropriately and output an observation. _ERROR_ is reset to 0 to prevent an error condition that would write the contents of the program data vector to the SAS log. When an unexpected condition occurs, write messages and the contents of the program data vector to the log, and stop the DATA step.

Resulting Log

This program creates an output data set but executes with one error. The following lines are written to the SAS log:

```
PartNumber=1 PartDescription=Nuts Quantity=10 _ERROR_=1
_IORC_=0 _N_=3
PartNumber=5 PartDescription=No description Quantity=50
_ERROR_=1 _IORC_=0 _N_=6
NOTE: The data set WORK.COMBINE has 7 observations and 3 variables.
```

Resulting Data Set

The following shows the incorrectly created COMBINE data set. Observation 5 should not be in this data set. PartNumber value 8 does not exist in either MASTER or DESCRIPTION, so no Quantity should be listed for it. Also, observations 3 and 7 contain descriptions from observations 2 and 6, respectively.

```
                  COMBINE

OBS    PartNumber    PartDescription   Quantity
 1         2          Screws              20
 2         4          Nuts                40
 3         1          Nuts                10
 4         3          Bolts               30
 5         8          No description      30
 6         5          No description      50
 7         6          No description      50
```

Revised Program

To create an accurate output data set, this example performs error checking on both SET statements that use the KEY= option:

```
data combine(drop=Foundes); ❶
   length PartDescription $ 15;
   set order;  ❷
   Foundes = 0;  ❸
   set description key=PartNumber;  ❹
   select(_iorc_);  ❺
      when(%sysrc(_sok)) do;
         Foundes = 1;
      end;
      when(%sysrc(_dsenom)) do;
         PartDescription = 'No description';
         _error_ = 0;
      end;
      otherwise do;
         put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
         put 'Program terminating. Data set accessed is DESCRIPTION';
         put _all_;
         _error_ = 0;
         stop;
      end;
   end;
set master key=PartNumber;  ❻
select(_iorc_); ❼
      when(%sysrc(_sok)) do;
         output;
      end;
      when(%sysrc(_dsenom)) do;
         if not Foundes then do;
            _error_ = 0;
            put 'WARNING: PartNumber ' PartNumber 'is not in'
                ' DESCRIPTION or MASTER.';
         end;
         else do;
            Quantity = 0;
            _error_ = 0;
            output;
         end;
      end;
      otherwise do;
         put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
         put 'Program terminating. Data set accessed is MASTER';
         put _all_;
         _error_ = 0;
         stop;
      end;
   end;      /* ends the SELECT group */
```

❶ Create the COMBINE data set.

❷ Read an observation from the ORDER data set.

❸ Create the variable Foundes so that its value can be used later to indicate when a PartNumber value has a match in the DESCRIPTION data set.

❹ Read an observation from the DESCRIPTION data set, using PartNumber as the key variable.

❺ Take the correct course of action based on whether a matching value for PartNumber is found in DESCRIPTION. The SELECT group directs execution to the correct code based on the value of _IORC_. When a match occurs (_SOK), the value of PartNumber in the observation that is being read from DESCRIPTION matches the current value from ORDER. Foundes is set to 1 to indicate that DESCRIPTION contributed to the current observation. When there is no match (_DSENOM), no observations in DESCRIPTION contain the current value of PartNumber, so the description is set appropriately. _ERROR_ is reset to 0 to prevent an error condition that would write the contents of the program data vector to the SAS log. Any other _IORC_ value indicates that an unexpected condition has been met, so messages are written to the log and the DATA step is stopped.

❻ Read an observation from the MASTER data set, using PartNumber as a key variable.

❼ Take the correct course of action based on whether a matching value for PartNumber is found in MASTER. When a match is found (_SOK) between the current PartNumber value from ORDER and from MASTER, write an observation. When a match isn't found (_DSENOM) in MASTER, test the value of Foundes. If Foundes is not true, then a value wasn't found in DESCRIPTION either, so write a message to the log but do not write an observation. If Foundes is true, however, the value is in DESCRIPTION but not MASTER. So write an observation but set Quantity to 0. Again, if an unexpected condition occurs, write a message and stop the DATA step.

Resulting Log

The DATA step executed without error. Six observations were correctly created and the following message was written to the log:

```
WARNING: PartNumber 8 is not in DESCRIPTION or MASTER.
NOTE: The data set WORK.COMBINE has 6 observations
      and 3 variables.
```

Correctly Created COMBINE Data Set

The following shows the correctly updated COMBINE data set. Note that COMBINE does not contain an observation with the PartNumber value 8. This value does not occur in either MASTER or DESCRIPTION.

```
                   COMBINE

OBS    PartNumber    PartDescription   Quantity

 1          2        Screws               20
 2          4        Nuts                 40
 3          1        No description       10
 4          3        Bolts                30
 5          5        No description       50
 6          6        Washers               0
```

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