SAS Certification Prep Guide: Base Programming for SAS 9, Third Edition

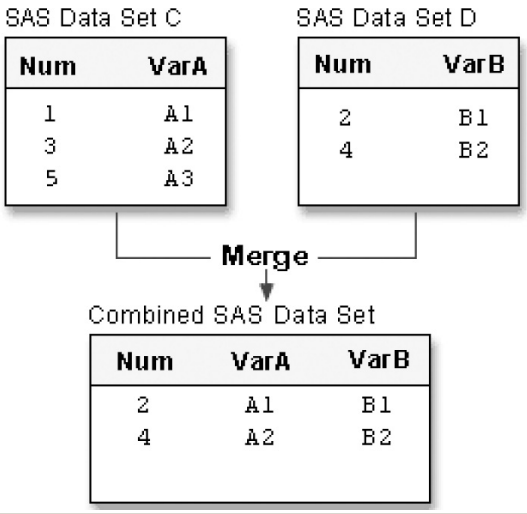
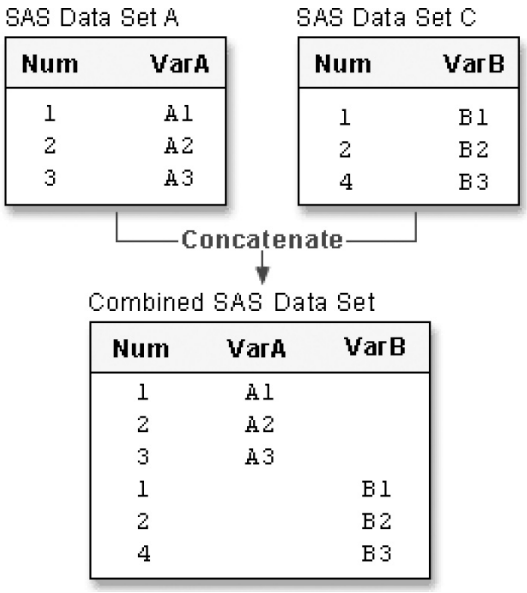
by SAS Institute

Chapter 12: Combining SAS Data Sets

Overview

Introduction

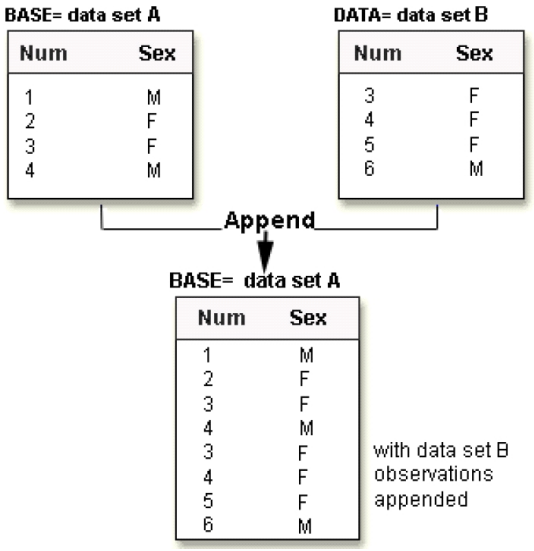
In SAS programming, a common task is to combine observations from two or more data sets into a new data set. Using the DATA step, you can combine data sets in several ways, including the following:

Method of Merging (Combining)	Illustration
One-to-one merging Creates observations that contain all of the variables from each contributing data set. Combines observations based on their relative position in each data set. Statement: SET	
Concatenating Appends the observations from one data set to another. Statement: SET	

Appending

Appending adds the observations in the second data set directly to the end of the original data set.

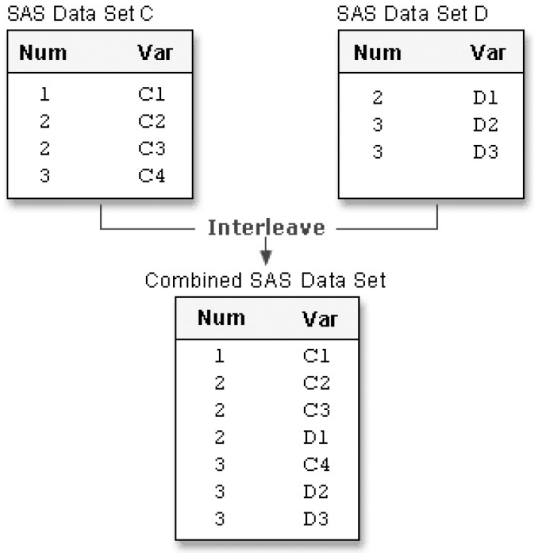
Procedure: APPEND



Interleaving

Intersperses observations from two or more data sets, based on one or more common variables.

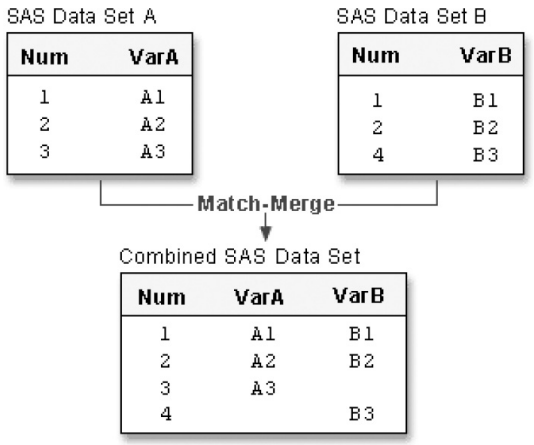
Statements: SET, BY



Match-merging

Matches observations from two or more data sets into a single observation in a new data set according to the values of a common variable.

Statements: MERGE, BY



Additional Note You can also use PROC SQL to join data sets according to common values. PROC SQL enables you to perform many other types of data set joins. See the *SQL Processing with SAS* e-learning course for additional training.

This chapter shows you how to combine SAS data sets using one-to-one combing or merging, concatenating, appending, interleaving, and match-merging. When you use the DATA step to combine data sets, you have a high degree of control in creating and manipulating data sets.

Objectives

In this chapter, you learn to

- perform one-to-one merging (combining) of data sets
- concatenate data sets
- append data sets
- interleave data sets
- match-merge data sets
- rename any like-named variables to avoid overwriting values
- select only matched observations, if desired
- predict the results of match-merging.

One-to-One Merging

Overview

In "Reading SAS Data Sets" on page 334, you learned how to use the SET statement to read an existing SAS data set. You can also use multiple SET statements in a DATA step to combine data sets. This is called one-to-one merging (or combining). In one-to-one merging you can read different data sets, or you can read the same data set more than once, as if you were reading from separate data sets.

General form, basic DATA step for one-to-one reading:

```
DATA output-SAS-data-set;
     SET SAS-data-set-1;
     SET SAS-data-set-2;
RUN;
```

where

- *output-SAS-data-set* names the data set to be created
- *SAS-data-set-1* and *SAS-data-set-2* specify the data sets to be read.

How One-to-One Merging Selects Data

When you perform one-to-one merging,

- the new data set contains all the variables from all the input data sets. If the data sets contain variables that have the same names, the values that are read from the last data set overwrite the values that were read from earlier data sets.
- the number of observations in the new data set is the number of observations in the smallest original data set. Observations are combined based on their relative position in each data set. That is, the first observation in one data set is joined with the first observation in the other, and so on. The DATA step stops after it has read the last observation from the smallest data set.

```
data one2one;
   set a;
   setb;
run;
```

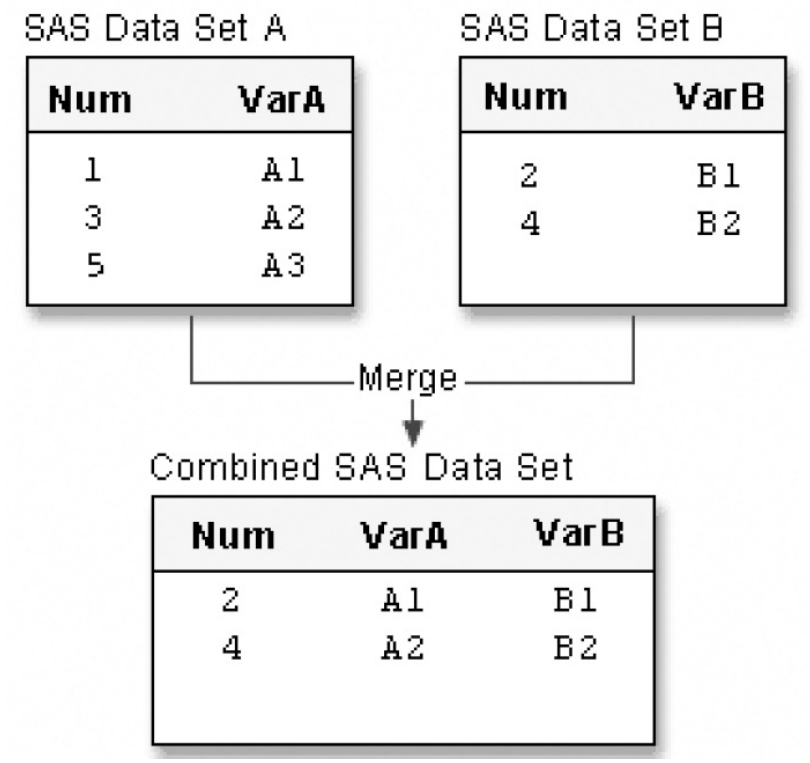


Figure 12.1: One-to-One Merging

How One-to-One Reading Works

Let's look at a simple case of one-to-one merging.

```
data one2one;
   set a;
   set b;
run;
```

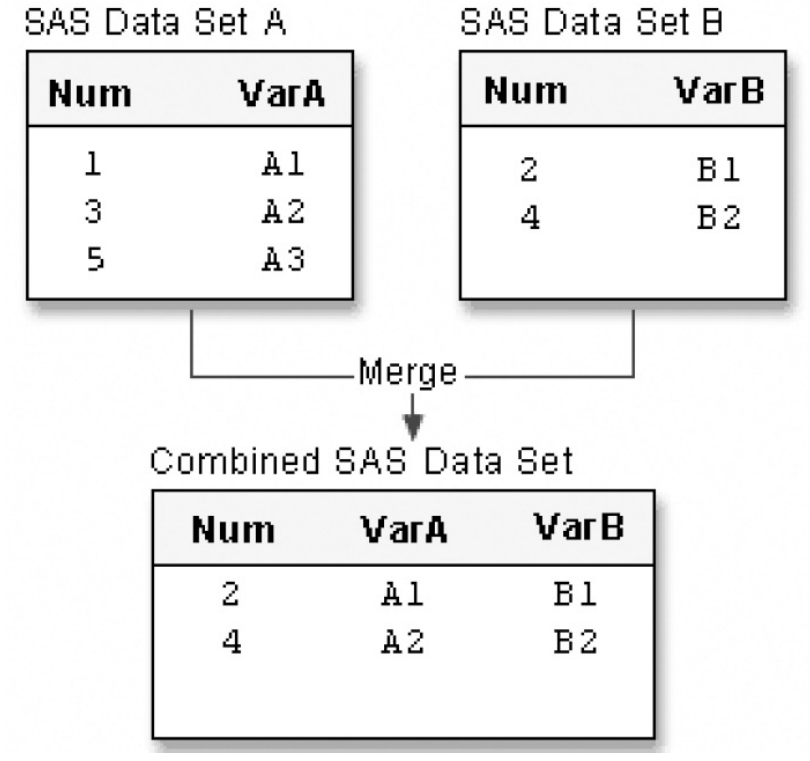
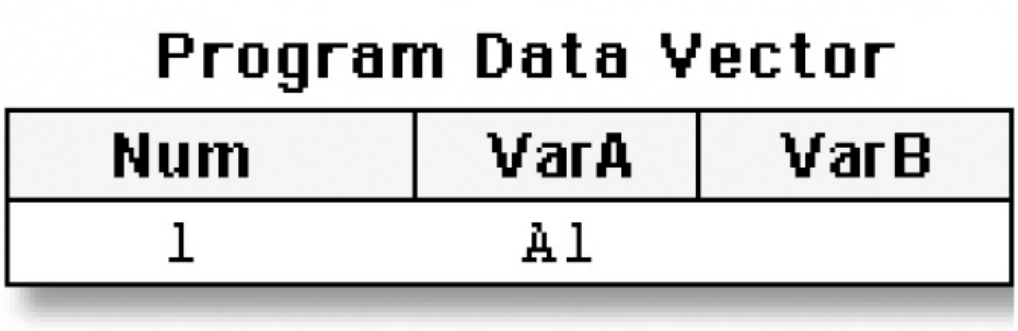
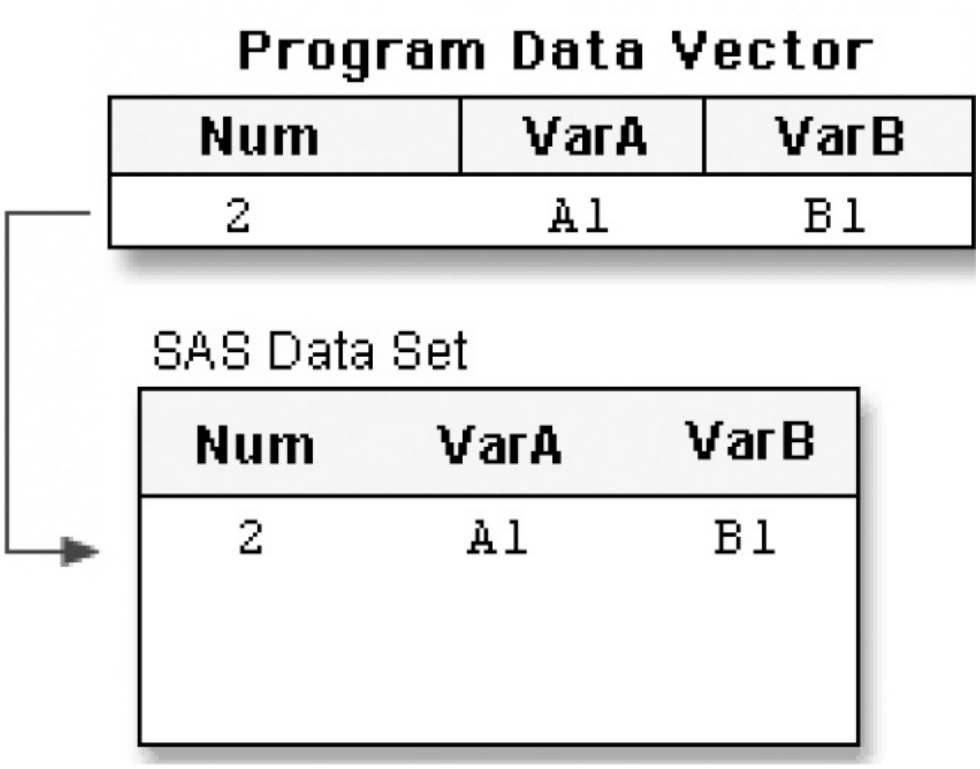


Figure 12.2: How One-to-One Reading Works

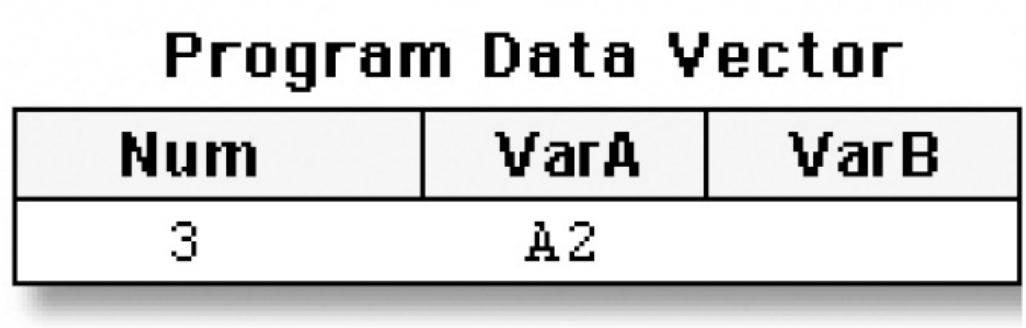
1. The first SET statement reads the first observation from data set A into the program data vector.



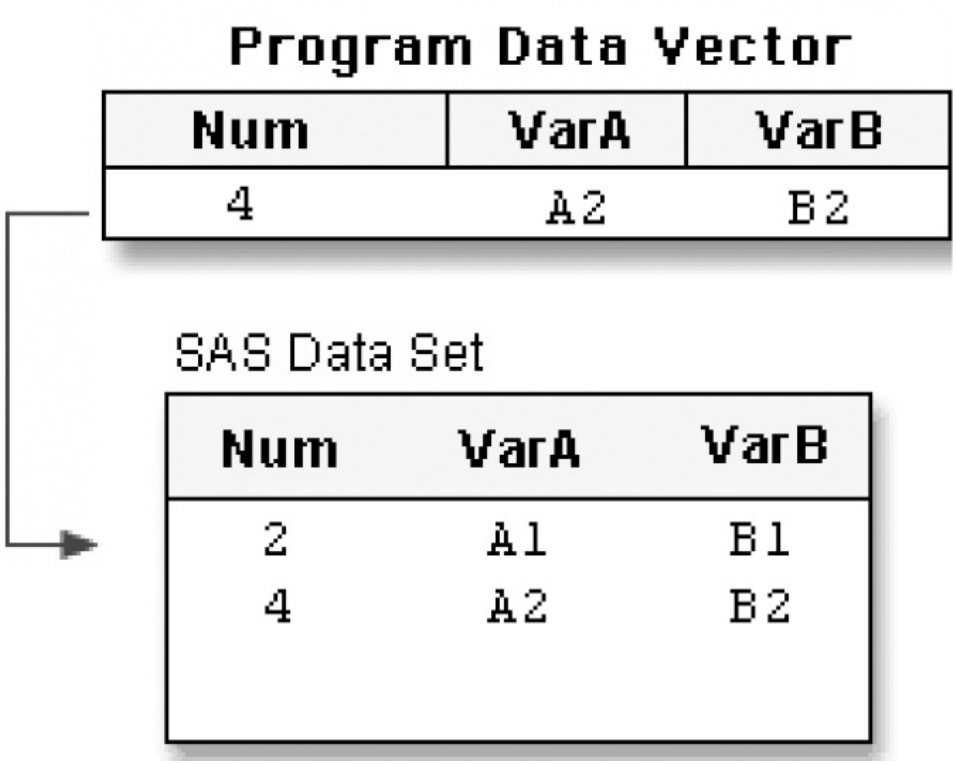
2. The second SET statement reads the first observation from data set B into the program data vector and SAS writes the contents of the program data vector to the new data set. The value for Num from data set B overwrites the value for Num from data set A.



3. The first SET statement reads the second observation from data set A into the program data vector.



4. The second SET statement reads the second observation from data set B, and SAS writes the contents of the program data vector to the new data set. The value for Num from data set B overwrites the value for Num from data set A.



5. The first SET statement reads the third observation from data set A into the program data vector.

Program Data Vector

Num	VarA	VarB
5	A3	

6. The second SET statement reads the end of file in data set B, which stops the DATA step processing with no further output written to the data set. The last observation in data set A is read into the program data vector, but it is not written to the output data set.

Num	VarA	VarB
2	Al	B1
4	A2	B2

Figure 12.3: Final Combined SAS Data Set

Here is an example of how you might use one-to-one merging.

Example: One-to-One Merging

Suppose you have basic patient data (ID, sex, and age) in the data set Clinic.Patients and want to combine it with other patient data (height and weight) for patients under age 60. The height and weight data is stored in the data set Clinic.Measure. Both data sets are sorted by the variable ID.

Notice that Clinic.Patients contains 7 observations in which the patient age is less than 60, and Clinic.Measure contains 6 observations.

SAS Data Set Clinic.Patients

Obs	ID	Sex	Age
1	1129	F	48
2	1387	F	57
3	2304	F	16
4	2486	F	63
5	4759	F	60
6	5438	F	42
7	6488	F	59
8	9012	F	39
9	9125	F	56

SAS Data Set Clinic.Measure

Obs	ID	Height	Weight
1	1129	61	137
2	1387	64	142
3	2304	61	102
4	5438	62	168
5	6488	64	154
6	9012	63	157

Figure 12.4: Example: One-to-One Merging

To subset observations from the first data set and combine them with observations from the second data set, you can submit the following program:

```
data clinic.one2one;
   set clinic.patients;
   if age<60;
```

```
   set clinic.measure;
run;
```

The resulting data set, Clinic.One2one, contains 6 observations (the number of observations read from the smallest data set, which is Clinic.Measure). The last observation in Clinic.Patients is not written to the data set because the second SET statement reaches an end-of-file, which stops the DATA step processing.

SAS Data Set Clinic.One2one

Obs	ID	Sex	Age	Height	Weight
1	1129	F	48	61	137
2	1387	F	57	64	142
3	2304	F	16	61	102
4	5438	F	42	62	168
5	6488	F	59	64	154
6	9012	F	39	63	157

Figure 12.5: The Resulting Data Set for One-to-One Reading Example

Concatenating

Overview

Another way to combine SAS data sets with the SET statement is concatenating, which appends the observations from one data set to another data set. To concatenate SAS data sets, you specify a list of data set names in the SET statement.

General form, basic DATA step for concatenating:

```
DATA output-SAS-data-set;
     SET SAS-data-set-1 SAS-data-set-2;
RUN;
```

where

- *output-SAS-data-set* names the data set to be created
- *SAS-data-set-1* and *SAS-data-set-2* specify the data sets to be read.

How Concatenating Selects Data

When a program concatenates data sets, all of the observations are read from the first data set listed in the SET statement. Then all of the observations are read from the second data set listed, and so on, until all of the listed data sets have been read. The new data set contains all of the variables and observations from all of the input data sets.

```
data concat;
   set a c;
run;
```

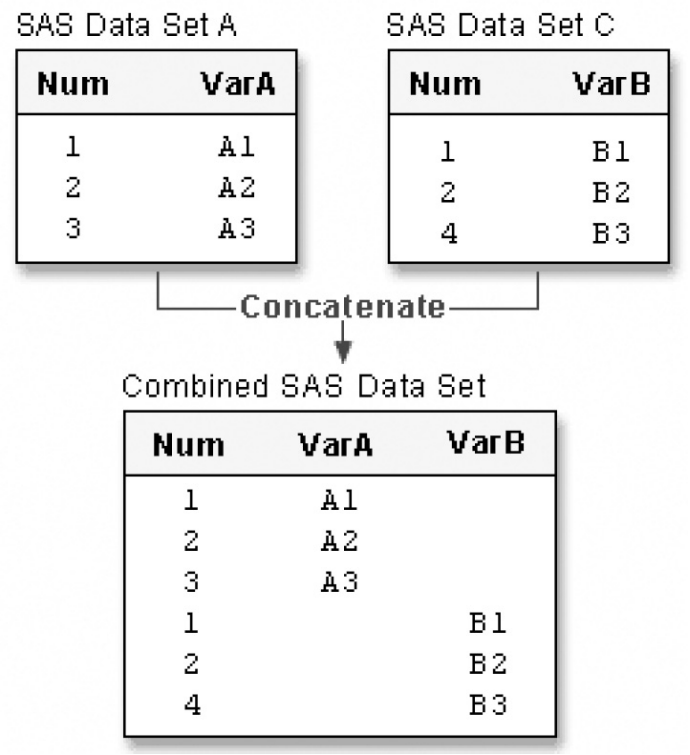


Figure 12.6: How Concatenating Selects

Notice that A and C contain a common variable named Num:

- Both instances of Num (or any common variable) must have the same type attribute, or SAS stops processing the DATA step and issues an error message stating that the variables are incompatible.
- However, if the length attribute is different, SAS takes the length from the first data set that contains the variable. In this case, the length of Num in A determines the length of Num in Concat.
- The same is true for the 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.

Example

The following DATA step creates Clinic.Concat by concatenating Clinic.Therapy1999 and Clinic.Therapy2000.

```
data clinic.concat;
   set clinic.therapy1999 clinic.therapy2000;
run;
```

Below is the listing of Clinic.Concat. The first 12 observations were read from Clinic.Therapy1999, and the last 12 observations were read from Clinic.Therapy2000.

Obs	Month	Year	AerClass	WalkJogRun	Swim
1	01	1999	26	78	14
2	02	1999	32	109	19
3	03	1999	16	106	22
4	04	1999	47	115	24
5	06	1999	95	121	31
6	06	1999	61	114	67
7	07	1999	67	102	72
8	08	1999	24	76	77
9	09	1999	78	77	54

10	10	1999	81	62	47
11	11	1999	84	31	52
12	12	1999	92	44	65
13	01	2000	37	91	83
14	02	2000	41	102	27
15	03	2000	52	98	19
16	04	2000	61	118	22
17	06	2000	49	88	29
18	08	2000	24	101	54
19	07	2000	45	91	69
20	08	2000	63	65	63
21	09	2000	60	49	68
22	10	2000	78	70	41
23	11	2000	82	44	68
24	12	2000	93	57	47

Figure 12.7: Example: Concatenating

Appending

Overview

Another way to combine SAS data sets is to append one data set to another using the APPEND procedure. Although appending and concatenating are similar, there are some important differences between the two methods. Whereas the DATA step creates an entirely new data set when concatenating, PROC APPEND simply adds the observations of one data set to the end of a "master" (or BASE) data set. SAS does not create a new data set nor does it read the base data set when executing the APPEND procedure.

To append SAS data sets, you specify a BASE= data set, which is the data set to which observations are added and then specify a DATA= data set, which is the data set containing the observations that are added to the base data set. The data set specified with DATA= is the only one of the two data sets that SAS actually reads.

General form of the APPEND procedure:

```
PROC APPEND BASE=SAS-data-set
     DATA=SAS-data-set;
RUN;
```

where

- *BASE*= names the data set to which observations are added
- *DATA*= names the data set containing observations that are added to the base data set

---

For example, the following PROC APPEND statement appends the observations in data set B to the end of data set A:

```
proc append base=A
            data=B;
run;
```

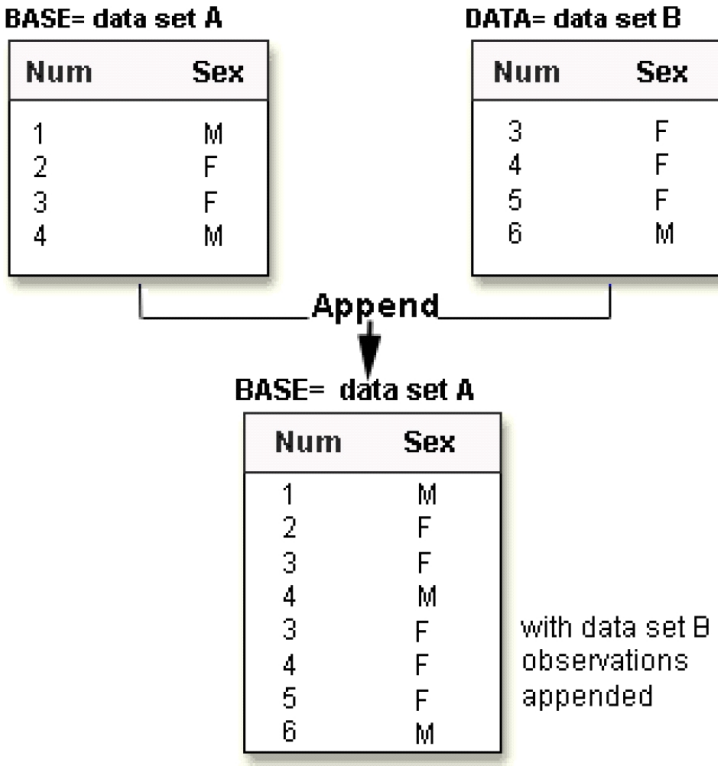


Figure 12.8: How PROC APPEND Appends Data

Requirements for the APPEND Procedure

The requirements for appending one data set to another are as follows:

- Only two data sets can be used at a time in one step.
- The observations in the base data set are not read.
- The variable information in the descriptor portion of the base data set cannot change.

Notice that the final data set is the original data set with appended observations and that no new data set was created.

Example

The following PROC APPEND statement appends the data set totals2011 to the base data set totals2005. The two data sets are like-structured data sets: that is, both data sets have the same variable information.

```
proc append base=totals2005 data=totals2011;
run;
```

Below is the listing of totals2005. The first 4 observations already existed in totals2005, and the last 4 observations were read from totals2011 and added to totals2005.

Obs	Month	Therapy	NewAdmit	Treadmill
1	01	220	27	11
2	02	209	S3	43
3	03	189	11	2
4	04	211	52	36
5	01	210	27	13
6	02	219	63	23
7	03	199	11	21
8	04	251	52	30

Figure 12.9: Example: PROC APPEND

Using the FORCE Option with Unlike-Structured Data Sets

In order to use PROC APPEND with data sets that have unmatching variable definitions, you can use the FORCE option in the PROC APPEND statement.

General form of the APPEND procedure with the FORCE option:

```
PROC APPEND BASE=SAS-data-set
  DATA=SAS-data-set FORCE;
RUN;
```

The FORCE option is needed when the DATA= data set contains variables that meet any one of the following criteria:

- They are not in the BASE= data set.
- They are variables of a different type (for example, character or numeric).
- They are longer than the variables in the BASE= data set.

If the length of a variable is longer in the DATA= data set than in the BASE= data set, SAS truncates values from the DATA= data set to fit them into the length that is specified in the BASE= data set.

If the type of a variable in the DATA= data set is different than in the BASE= data set, SAS replaces all values for the variable in the DATA= data set with missing values and keeps the variable type of the variable specified in the BASE= data set.

If the BASE= data set contains a variable that is not in the DATA= data set, the observations are appended, but the observations from the DATA= data set have a missing value for the variable that was not present in the DATA= data set. If the DATA= data set contains a variable that is not in the BASE= data set, the variable is dropped from the output.

Example

For example:

```
proc append base=sasuser.patients
            data=clinic.append force;
run;
```

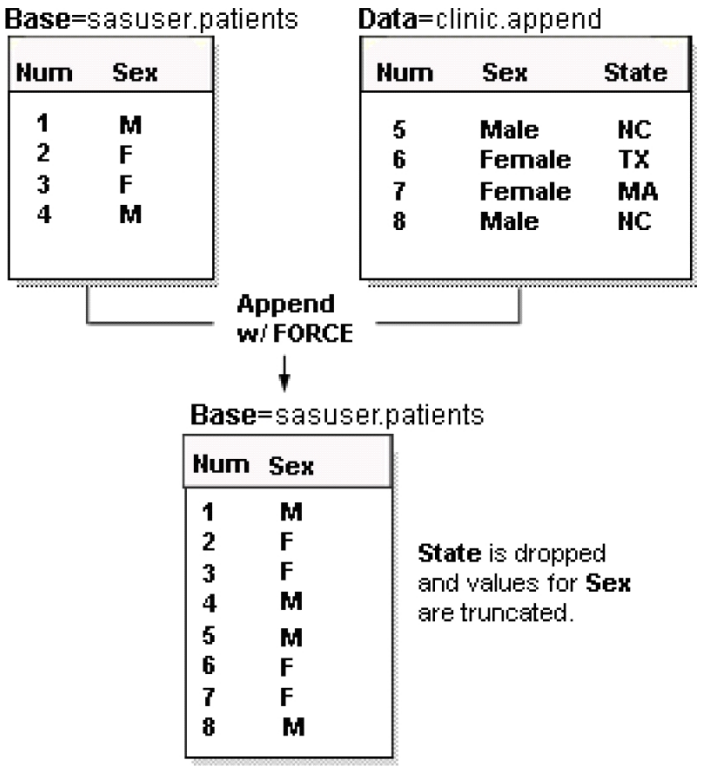


Figure 12.10: Example: PROC APPEND with FORCE Option

Interleaving

Overview

If you use a BY statement when you concatenate data sets, the result is interleaving. Interleaving intersperses observations from two or more data sets, based on one or more common variables.

To interleave SAS data sets, specify a list of data set names in the SET statement, and specify one or more BY variables in the BY statement.

General form, basic DATA step for interleaving:

```
DATA output-SAS-data-set;
     SET SAS-data-set-1 SAS-data-set-2;
     BY variable(s);
RUN;
```

where

- *output-SAS-data-set* names the data set to be created
- *SAS-data-set-1* and *SAS-data-set-2* specify the data sets to be read.
- *variable(s)* specifies one or more variables that are used to interleave observations.

Additional Note You can specify any number of data sets in the SET statement. Each input data set *must* be sorted or indexed in ascending order based on the BY variable(s).

How Interleaving Selects Data

When SAS interleaves data sets, observations in each BY group in each data set in the SET statement are read sequentially, in the order in which the data sets and BY variables are listed, until all observations have been processed. The new data set includes all the variables from all the input data sets, and it contains the total number of observations from all input data sets.

```
data interlv;
     set c d;
     by num;
run;
```

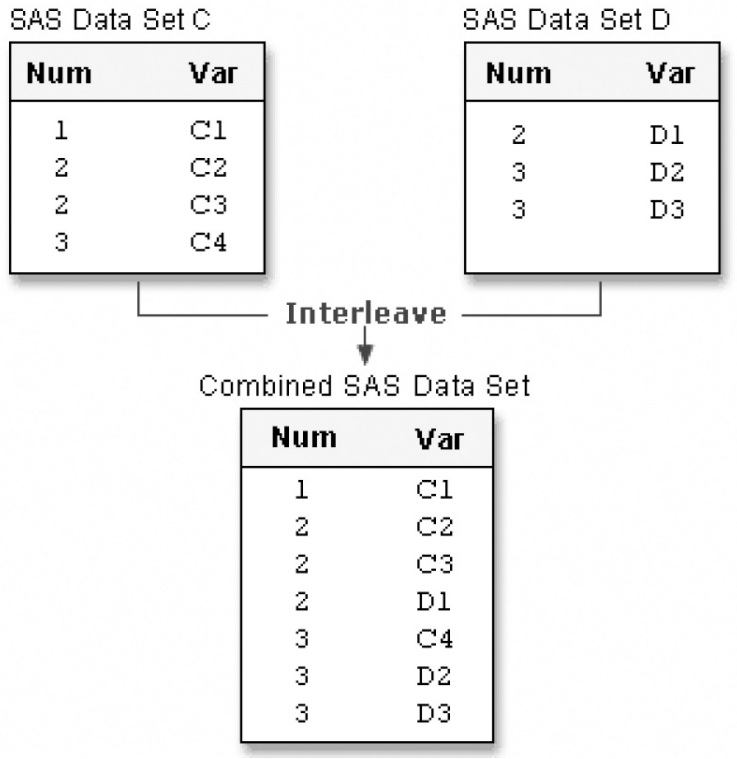


Figure 12.11: How Interleaving Selects Data

Example

The following DATA step creates Clinic.Interlv by interleaving Clinic.Therapy1999 and Clinic.Therapy2000.

```
data clinic.interlv;
   set clinic.therapy1999 clinic.therapy2000;
   by month;
run;
```

Below is the listing of Clinic.Interlv. Notice that, unlike the previous example, observations are interleaved by month instead of being concatenated.

SAS Data Set Clinic.Interlv

Obs	Month	Year	AerClass	WalkJogRun	Swim
1	01	1999	26	78	14
2	01	2000	37	91	83
3	02	1999	32	109	19
4	02	2000	41	102	27
5	03	1999	15	106	22
6	03	2000	52	98	19
7	04	1999	47	115	24
8	04	2000	61	118	22
9	05	1999	95	121	31
10	05	2000	49	88	29
11	06	1999	61	114	67
12	06	2000	24	101	54
13	07	1999	67	102	72
14	07	2000	45	91	69
15	08	1999	24	76	77
16	08	2000	63	65	53
17	09	1999	78	77	54
18	09	2000	60	49	68
19	10	1999	81	62	47
20	10	2000	78	70	41
21	11	1999	84	31	52
22	11	2000	82	44	58
23	12	1999	92	44	55
24	12	2000	93	57	47

Figure 12.12: Example: How Interleaving Selects Data

Match-Merging

Overview

So far in this chapter, we've combined data sets based on the order of the observations in the input data sets. But sometimes you need to combine observations from two or more data sets into a single observation in a new data set according to the values of a common variable. This is called match-merging.

When you match-merge, you use a MERGE statement rather than a SET statement to combine data sets.

General form, basic DATA step for match-merging:

```
DATA output-SAS-data-set;
     MERGE SAS-data-set-1 SAS-data-set-2;
     BY <DESCENDING> variable(s);
RUN;
```

where

- *output-SAS-data-set* names the data set to be created.
- *SAS-data-set-1* and *SAS-data-set-2* specify the data sets to be read.
- *variable(s)* in the **BY** statement specifies one or more variables whose values are used to match observations.
- DESCENDING indicates that the input data sets are sorted in descending order (largest to smallest numerically, or reverse alphabetical for character variables) by the variable that is specified. If you have more that one variable in the BY statement, DESCENDING applies only to the variable that immediately follows it.

Additional Note Each input data set in the MERGE statement *must* be sorted in order of the values of the BY variable (s), or it must have an appropriate index. Each BY variable must have the same type in all data sets to be merged.

Additional Note You cannot use the DESCENDING option with indexed data sets because indexes are always stored in ascending order.

How Match-Merging Selects Data

Generally speaking, during match-merging, SAS sequentially checks each observation of each data set to see whether the BY values match, and then writes the combined observation to the new data set.

```
data merged;
   merge a b;
   by num;
run;
```

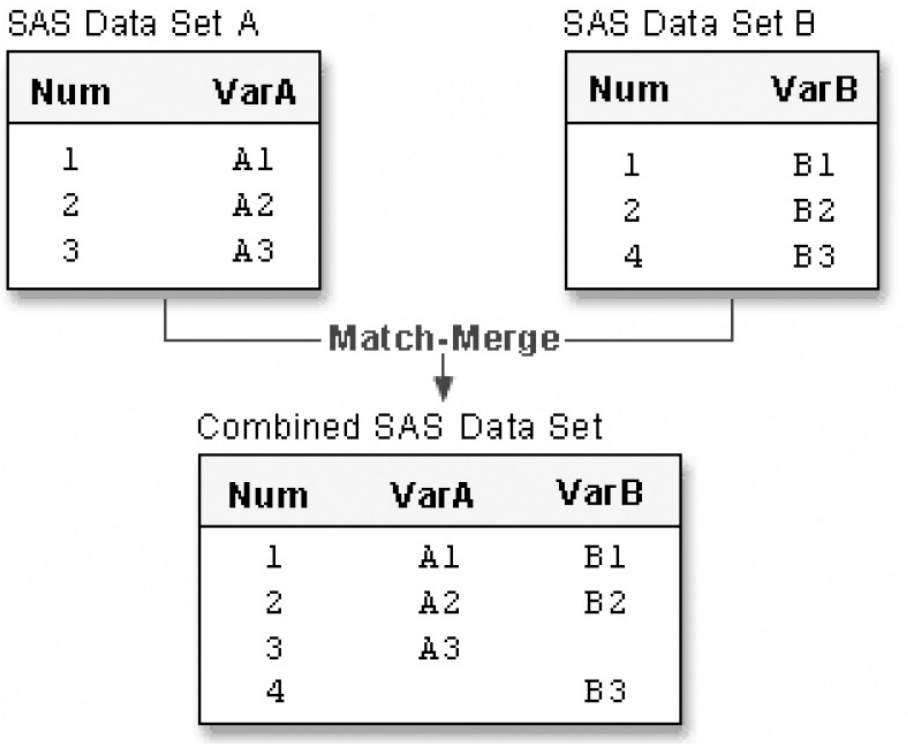


Figure 12.13: How Match-Merging Selects Data

Basic DATA step match-merging produces an output data set that contains values from all observations in all input data sets. You can add statements and options to select only matching observations.

If an input data set doesn't have any observations for a particular value of the by-variable, then the observation in the output data set contains missing values for the variables that are unique to that input data set.

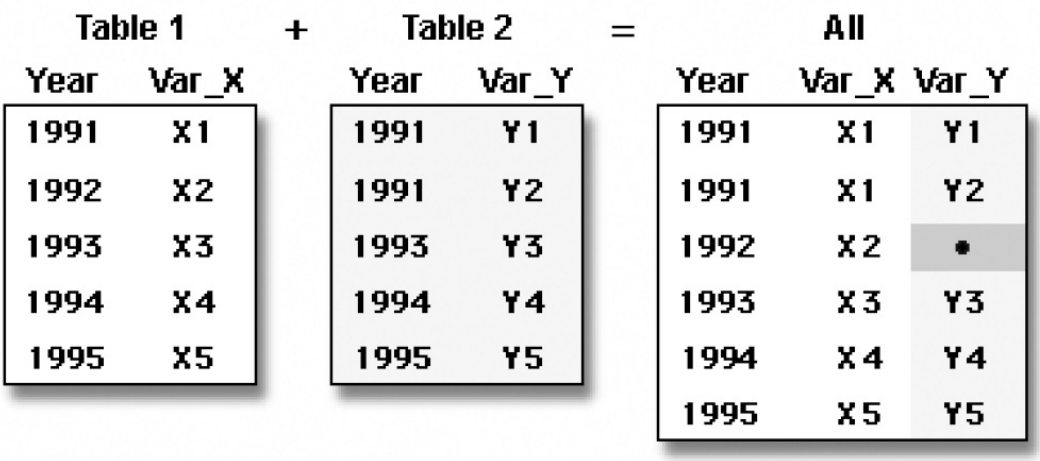


Figure 12.14: Match-Merging with Missing Data

Additional Note In match-merging, often one data set contains unique values for the BY-variable and other data sets contain multiple values for the BY-variable.

Example

Suppose you have sorted the data sets Clinic.Demog and Clinic.Visit as follows:

```
proc sort data=clinic.demog;
   by id;
run;
proc print data=clinic.demog;
run;
```

Obs	ID	Age	Sex	Date
1	A001	21	m	05/22/75
2	A002	32	m	06/15/63
3	A003	24	f	08/17/72
4	AO04	.		03/27/69
5	A005	44	f	02/24/52
6	A007	39	m	11/11/57

Figure 12.15: Example: Sorting clinic.demog

```
proc sort data=clinic.visit;
   by id;
run;
proc print data=clinic.visit;
run;
```

Obs	ID	Visit	SysBP	DiasBP	Weight	Date
1	A001	1	140	85	195	11/05/98
2	A001	2	138	90	198	10/13/98
3	A001	3	145	95	200	07/04/98
4	A002	1	121	75	168	04/14/98
5	A003	1	118	68	125	08/12/98
6	A003	2	112	65	123	08/21/98
7	AO04	1	143	86	204	03/30/98
8	A005	1	132	76	174	02/27/98
9	A005	2	132	78	175	07/11/98
10	A005	3	134	78	176	04/16/98
11	A008	1	126	80	182	05/22/98

Figure 12.16: Example: Sorting clinic.visit

You can then submit this DATA step to create Clinic.Merged by merging Clinic.Demog and Clinic.Visit according to values of the variable ID.

```
data clinic.merged;
   merge clinic.demog clinic.visit;
   by id;
run;
proc print data=clinic.merged;
run;
```

Notice that all observations, including unmatched observations and observations that have missing data, are written to the output data set.

Obs	ID	Age	Sex	Date	Visit	SysBP	DiasBP	Weight
1	A001	21	m	11/05/98	1	140	85	195
2	A001	21	m	10/13/98	2	138	90	198
3	A001	21	m	07/04/98	3	145	95	200
4	A002	32	m	04/14/98	1	121	75	168
5	A003	24	f	08/12/98	1	118	68	125
6	A003	24	f	08/21/98	2	112	65	123
7	AO04	.		03/30/98	1	143	86	204
8	A005	44	f	02/27/98	1	132	76	174
9	A005	44	f	07/11/98	2	132	78	175
10	A005	44	f	04/16/98	3	134	78	176
11	A007	39	m	11/11/57		.	.	.
12	A008	.		05/22/98	1	126	80	182

Figure 12.17: Match-merging Output

Example: Merge in Descending Order

The example above illustrates merging two data sets that are sorted in ascending order of the BY variable ID. To sort the data sets in descending order and then merge them, you can submit the following program.

```
proc sort data=clinic.demog;
   by descending id;
run;
proc sort data=clinic.visit;
   by descending id;
run;
data clinic.merged;
   merge clinic.demog clinic.visit;
   by descending id;
run;
proc print data=clinic.merged;
run;
```

Notice that you specify the DESCENDING option in the BY statements in both the PROC SORT steps and the DATA step. If you omit the DESCENDING option in the DATA step, you generate error messages about improperly sorted BY variables.

Now the data sets are merged in descending order of the BY variable ID.

Obs	ID	Age	Sex	Date	Visit	SysBP	DiasBP	Weight
1	A008	.		05/22/98	1	126	80	182
2	A007	39	m	11/11/57		.	.	.
3	A005	44	f	02/27/98	1	132	76	174
4	A005	44	f	07/11/98	2	132	78	175
5	A005	44	f	04/16/98	3	134	78	176
6	AO04	.		03/30/98	1	143	86	204
7	A003	24	f	08/12/98	1	118	68	125
8	A003	24	f	08/21/98	2	112	65	123
9	A002	32	m	04/14/98	1	121	75	168
10	A001	21	m	11/05/98	1	140	85	195
11	A001	21	m	10/13/98	2	138	90	198

12	A001	21	m	07/04/98	3	145	95	200

Figure 12.18: Example: Merge in Descending Order

Match-Merge Processing

Overview

The match-merging examples in this chapter are straightforward. However, match-merging can be more complex, depending on your data and on the output data set that you want to create. To predict the results of match-merges correctly, you need to understand how the DATA step performs match-merges.

When you submit a DATA step, it is processed in two phases:

- the compilation phase, in which SAS checks the syntax of the SAS statements and compiles them (translates them into machine code). During this phase, SAS also sets up descriptor information for the output data set and creates the program data vector (PDV), an area of memory where SAS holds one observation at a time.

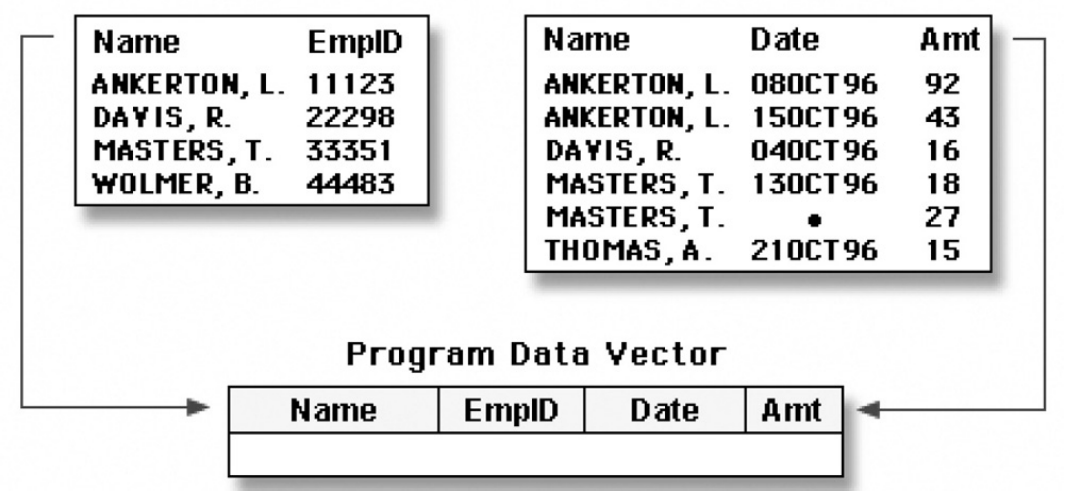


Figure 12.19: Match-Merge Processing: Compilation Phase

- the execution phase, in which the DATA step reads data and executes any subsequent programming statements. When the DATA step executes, data values are read into the appropriate variables in the program data vector. From here, the variables are written to the output data set as a single observation.

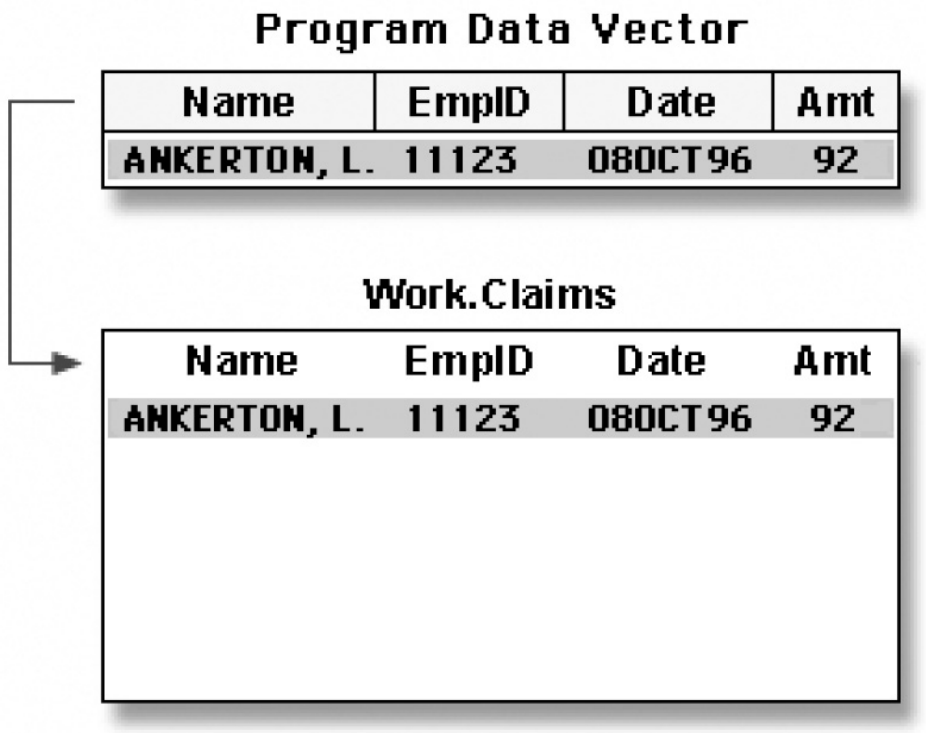


Figure 12.20: Match-Merge Processing: Execution Phase

The following pages cover DATA step processing in greater detail. In those pages, you learn

- how the DATA step sets up the new output data set
- what happens when variables in different data sets have the same name
- how the DATA step matches observations in input data sets
- what happens when observations don't match
- how missing values are handled.

The Compilation Phase: Setting Up the New Data Set

To prepare to merge data sets, SAS

- reads the descriptor portions of the data sets that are listed in the MERGE statement
- reads the rest of the DATA step program
- creates the program data vector (PDV) for the merged data set
- assigns a tracking pointer to each data set that is listed in the MERGE statement.

If variables that have the same name appear in more than one data set, the variable from the first data set that contains the variable (in the order listed in the MERGE statement) determines the length of the variable.

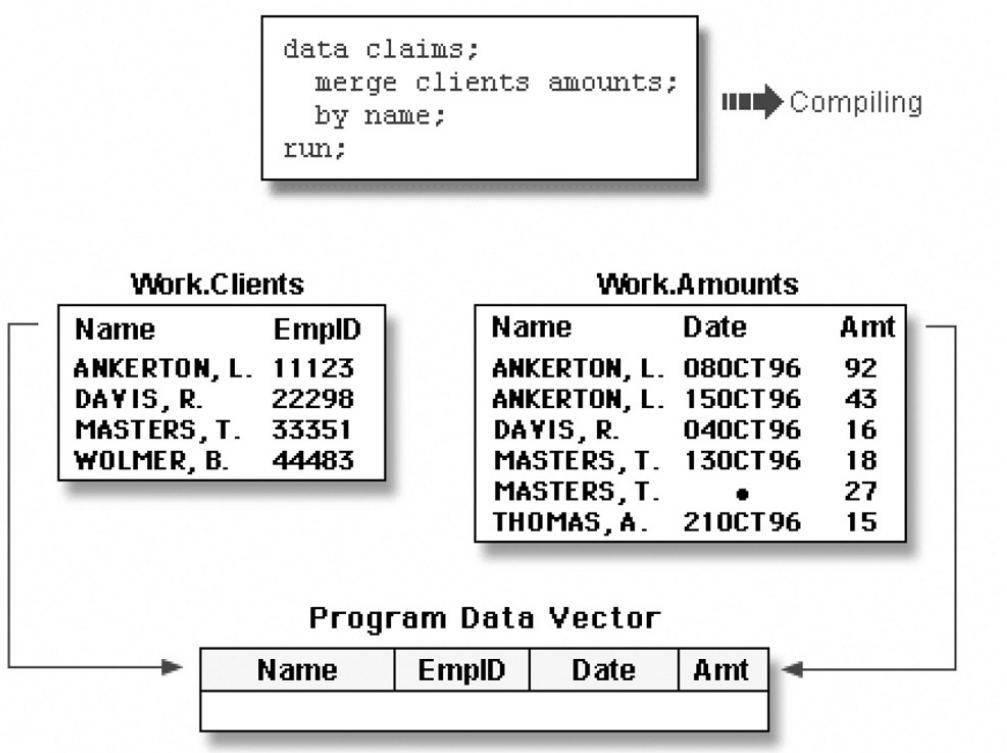


Figure 12.21: The Compilation Phase: Setting Up the New Data Set

The illustration above shows match-merging during the compilation phase. After reading the descriptor portions of the data sets Clients and Amounts, SAS

1. creates a program data vector for the new Claims data set. The program data vector contains all variables from the two data sets. Note that although Name appears in both input data sets, it appears in the program data vector only once.

2. assigns tracking pointers to Clients and Amounts.

The Execution Phase: Match-Merging Observations

After compiling the DATA step, SAS sequentially match-merges observations by moving the pointers down each observation of each data set and checking to see whether the BY values match.

- If *Yes*, the observations are read into the PDV in the order in which the data sets appear in the MERGE statement. Values of any same-named variable are overwritten by values of the same-named variable in subsequent data sets. SAS writes the combined observation to the new data set and retains the values in the PDV until the BY value changes in all the data sets.

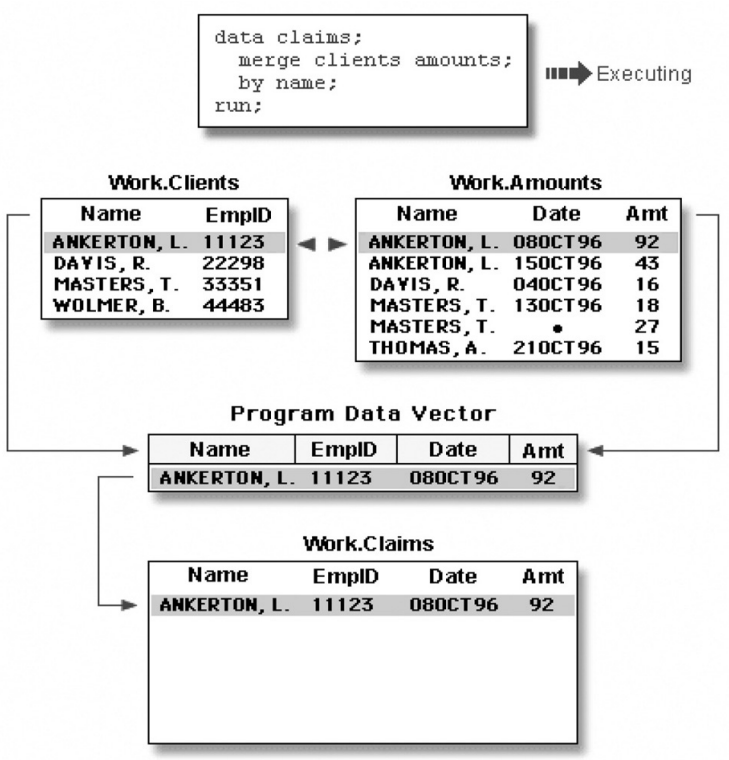


Figure 12.22: The Execution Phase: Match-Merging Observations

- If *No*, SAS determines which BY value comes first and reads the observation that contains this value into the PDV. Then the contents of the PDV are written

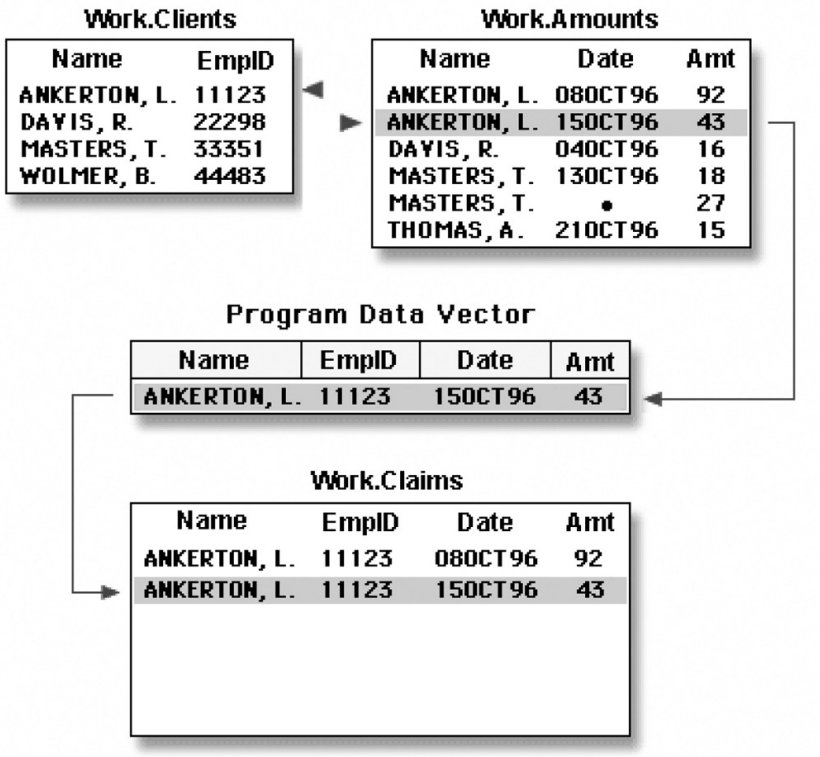


Figure 12.23: The Execution Phase: PDV

When the BY value changes in all the input data sets, the PDV is initialized to missing.

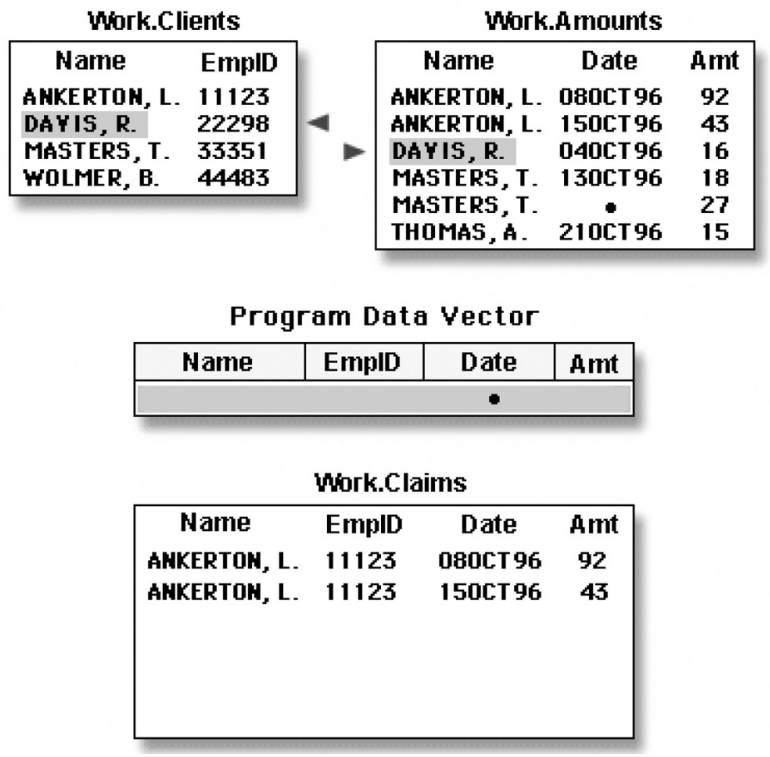


Figure 12.24: Initializing the PDV to Missing

The DATA step merge continues to process every observation in each data set until it has processed all observations in all data sets.

Handling Unmatched Observations and Missing Values

By default, all observations that are read into the PDV, including observations that have missing data and no matching BY values, are written to the output data set. (If you specify a subsetting IF statement to select observations, then only those that meet the IF condition are written.)

- If an observation contains missing values for a variable, then the observation in the output data set contains the missing values as well. Observations that have missing values for the BY variable appear at the top of the output data set because missing values sort first in ascending order.

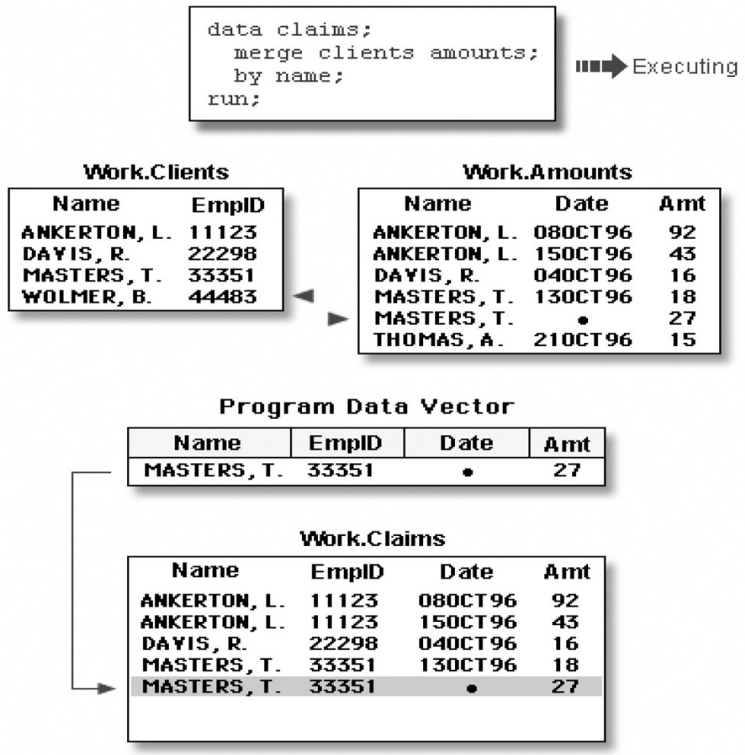


Figure 12.25: Handling Unmatched Observations and Missing Values

- If an input data set doesn't have a matching BY value, then the observation in the output data set contains missing values for the variables that are unique to that input data set.

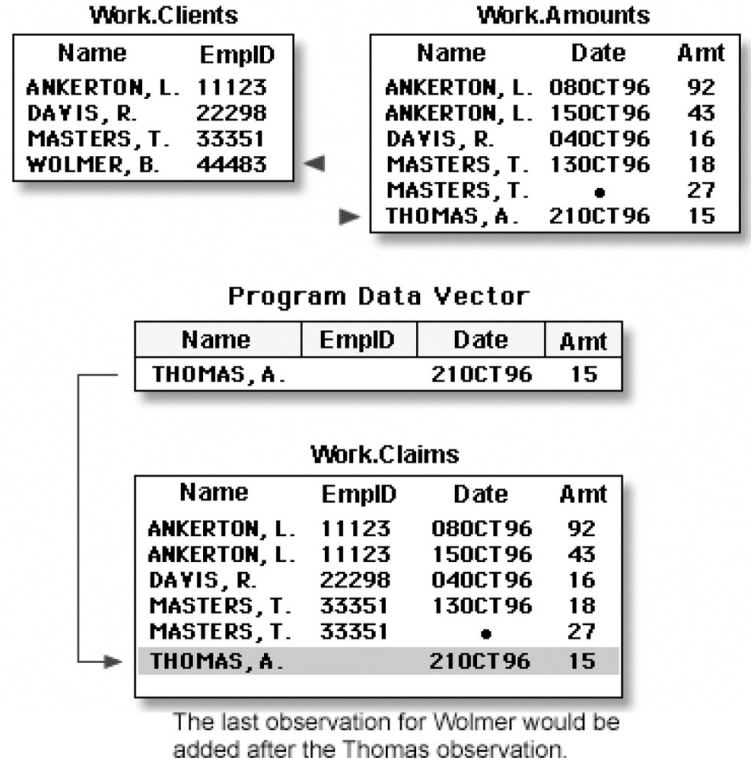


Figure 12.26: Handling Unmatched Observations and Missing Values: Unique Missing Values

Renaming Variables

Sometimes you might have same-named variables in more than one input data set. In this case, DATA step match-merging overwrites values of the like-named variable in the first data set in which it appears with values of the like-named variable

in subsequent data sets.

For example, Clinic.Demog contains the variable Date (date of birth), and Clinic.Visit also contains Date (date of the clinic visit in 1998). The DATA step below overwrites the date of birth with the date of the clinic visit.

```
data clinic.merged;
   merge clinic.demog clinic.visit;
   by id;
run;
proc print data=clinic.merged;
run;
```

The following output shows the effects of overwriting the values of a variable in the Clinic.Merged data set. In most observations, the date is now the date of the clinic visit. In observation 11, the date is still the birth date because Clinic.Visit did not contain a matching ID value and did not contribute to the observation.

Obs	ID	Age	Sex	Date	Visit	SysBP	DiasBP	Weight
1	A001	21	m	11/05/98	1	140	85	195
2	A001	21	m	10/13/98	2	138	90	198
3	A001	21	m	07/04/98	3	145	95	200
4	A002	32	m	04/14/98	1	121	75	168
5	A003	24	f	08/12/98	1	118	68	125
6	A003	24	f	08/21/98	2	112	65	123
7	AO04	.		03/30/98	1	143	86	204
8	A005	44	f	02/27/98	1	132	76	174
9	A005	44	f	07/11/98	2	132	78	175
10	A005	44	f	04/16/98	3	134	78	176
11	A007	39	m	11/11/57		.	.	.
12	A008	.		05/22/98	1	126	80	182

Figure 12.27: Renaming Variables

You now have a data set with values for Date that mean two different things: date of birth and date of clinic visit. Let's see how to prevent this problem.

To prevent overwriting, you can rename variables by using the RENAME= data set option in the MERGE statement.

General form, RENAME= data set option:

(RENAME=(*old-variable-name=new-variable-name*))

where

- the **RENAME=** option, in parentheses, follows the name of each data set that contains one or more variables to be renamed
- *old-variable-name* specifies the variable to be renamed
- *new-variable-name* specifies the new name for the variable.

Additional Note You can rename any number of variables in each occurrence of the RENAME= option.

You can also use RENAME= to rename variables in the SET statement or in the output data set that is specified in the DATA statement.

In the following example, the RENAME= option renames the variable Date in Clinic.Demog to BirthDate, and it renames the

variable Date in Clinic.Visit to VisitDate.

```
data clinic.merged;
   merge clinic.demog (rename=(date=BirthDate))
         clinic.visit (rename=(date=VisitDate));
   by id;
run;
proc print data=clinic.merged;
run;
```

The following output shows the effect of the RENAME= option.

Obs	ID	Age	Sex	BirthDate	Visit	SysBP	DiasBP	Weight	VisitDate
1	A001	21	m	05/22/75	1	140	85	195	11/05/98
2	A001	21	m	05/22/75	2	138	90	198	10/13/98
3	A001	21	m	05/22/75	3	145	95	200	07/04/98
4	A002	32	m	06/15/63	1	121	75	168	04/14/98
5	A003	24	f	08/17/72	1	118	68	125	08/12/98
6	A003	24	f	08/17/72	2	112	65	123	08/21/98
7	AO04	.		03/27/69	1	143	86	204	03/30/98
8	A005	44	f	02/24/52	1	132	76	174	02/27/98
9	A005	44	f	02/24/52	2	132	78	175	07/11/98
10	A005	44	f	02/24/52	3	134	78	176	04/16/98
11	A007	39	m	11/11/57			.	.	.
12	A008	.		.	1	126	80	182	05/22/98

Figure 12.28: Output for RENAME= Option

Excluding Unmatched Observations

Overview

By default, DATA step match-merging combines all observations in all input data sets. However, you may want to select only observations that match for two or more specific input data sets.

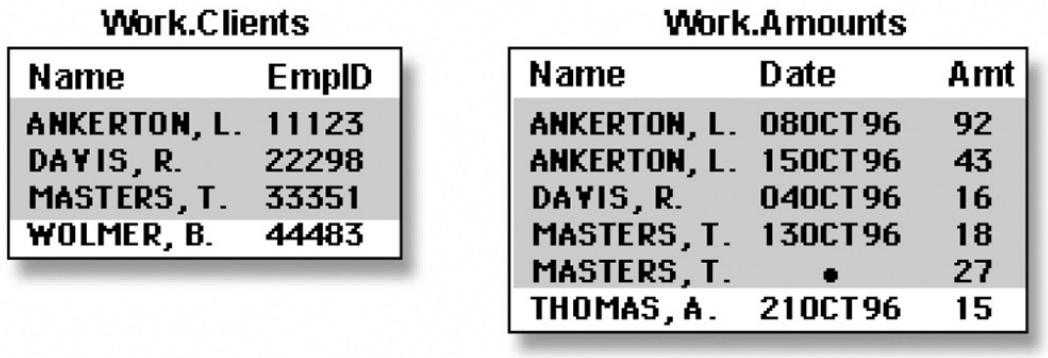


Figure 12.29: Excluding Unmatched Observations

To exclude unmatched observations from your output data set, you can use the IN= data set option and the subsetting IF statement in your DATA step. In this case, you use

- the IN= data set option to create and name a variable that indicates whether the data set contributed data to the current observation
- the subsetting IF statement to check the IN= values and write to the merged data set only matching observations.

Creating Temporary IN= Variables

Suppose you want to match-merge the data sets Clinic.Demog and Clinic.Visit and select only observations that appear in both data sets.

First, you use IN= to create two temporary variables, indemog and invisit. The IN= variable is a temporary variable that is available to program statements during the DATA step, but it is not included in the SAS data set that is being created.

General form, IN= data set option:

(IN= *variable)*

where

- the **IN=** option, in parentheses, follows the data set name
- variable names the variable to be created.

Within the DATA step, the value of the variable is **1** if the data set contributed data to the current observation. Otherwise, its value is **0**.

The DATA step that contains the IN= options appears below. The first IN= creates the temporary variable indemog, which is set to **1** when an observation from Clinic.Demog contributes to the current observation. Otherwise, it is set to **0**. Likewise, the value of invisit depends on whether Clinic.Visit contributes to an observation or not.

```
data clinic.merged;
   merge clinic.demog(in=indemog)
         clinic.visit(in=invisit
                      rename=(date=BirthDate));
   by id;
run;
```

Additional Note When you specify multiple data set options for a given data set, enclose them in a single set of parentheses.

Selecting Matching Observations

Next, to select only observations that appear in both Clinic.Demog and Clinic.Visit, you specify a subsetting IF statement in the DATA step.

In the DATA step below, the subsetting IF statement checks the values of indemog and invisit and continues processing only those observations that meet the condition of the expression. Here the condition is that both Clinic.Demog and Clinic.Visit contribute to the observation. If the condition is met, the new observation is written to Clinic.Merged. Otherwise, the observation is deleted.

```
data clinic.merged;
   merge clinic.demog(in=indemog
                      rename=(date=BirthDate))
         clinic.visit(in=invisit
                      rename=(date=VisitDate));
   by id;
   if indemog=1 and invisit=1;
run;
proc print data=clinic.merged;
run;
```

In previous examples, Clinic.Merged contained 12 observations. In the output below, notice that only 10 observations met the condition in the IF expression.

Obs	ID	Age	Sex	BirthDate	Visit	SysBP	DiasBP	Weight	VisitDate
1	A001	21	m	05/22/75	1	140	85	195	11/05/98
2	A001	21	m	05/22/75	2	138	90	198	10/13/98

3	A001	21	m	05/22/75	3	145	95	200	07/04/98
4	A002	32	m	06/15/63	1	121	75	168	04/14/98
5	A003	24	f	08/17/72	1	118	68	125	08/12/98
6	A003	24	f	08/17/72	2	112	65	123	08/21/98
7	AO 04	.		03/27/69	1	143	86	204	03/30/98
8	A005	44	f	02/24/52	1	132	76	174	02/27/98
9	A005	44	f	02/24/52	2	132	78	175	07/11/98
10	A005	44	f	02/24/52	3	134	78	176	04/16/98

Figure 12.30: Selecting Matching Observations

SAS evaluates the expression within an IF statement to produce a result that is either nonzero, zero, or missing. A nonzero and nonmissing result causes the expression to be true; a zero or missing result causes the expression to be false.

Thus, you can specify the subsetting IF statement from the previous example in either of the following ways. The first IF statement checks specifically for a value of **1**. The second IF statement checks for a value that is neither missing nor **0** (which for IN= variables is always **1)**.

```
if indemog=1 and invisit=1;
```

```
if indemog and invisit;
```

Selecting Variables

Overview

As with reading raw data or reading SAS data sets, you can specify the variables you want to drop or keep by using the DROP= and KEEP= data set options.

For example, the DATA step below reads all variables from Clinic.Demog and all variables except Weight from Clinic.Visit. It then excludes the variable ID from Clinic.Merged after the merge processing is complete.

```
data clinic.merged (drop=id);
   merge clinic.demog(in=indemog
                      rename=(date=BirthDate))
         clinic.visit(drop=weight in=invisit
                      rename=(date=VisitDate));
   by id;
   if indemog and invisit;
run;
proc print data=clinic.merged;
run;
```

Obs	Age	Sex	BirthDate	Visit	SysBP	DiasBP	VisitDate
1	21	m	05/22/75	1	140	85	11/05/98
2	21	m	05/22/75	2	138	90	10/13/98
3	21	m	05/22/75	3	145	95	07/04/98
4	32	m	06/15/63	1	121	75	04/14/98
5	24	f	08/17/72	1	118	68	08/12/98
6	24	f	08/17/72	2	112	65	08/21/98
7	.		03/27/69	1	143	86	03/30/98
8	44	f	02/24/52	1	132	76	02/27/98
9	44	f	02/24/52	2	132	78	07/11/98
10	44	f	02/24/52	3	134	78	04/16/98

Figure 12.31: Selecting Variables

Where to Specify DROP= and KEEP=

As you've seen in previous chapters, you can specify the DROP= and KEEP= options wherever you specify a SAS data set. When match-merging, you can specify these options in either the DATA statement or the MERGE statement, depending on whether you want to reference the variables in that DATA step:

- If you *don't* reference certain variables and you don't want them to appear in the new data set, specify them in the DROP= option in the MERGE statement.

```
merge clinic.demog(in=indemog
                   rename=(date=BirthDate))
      clinic.visit(drop=weight in=invisit
                   rename=(date=VisitDate));
```

- If you *do* need to reference a variable in the original data set (in a subsetting IF statement, for example), then you must specify the variable in the DROP= option in the DATA statement. Otherwise, you may get unexpected results and your variable will be uninitialized.

```
data clinic.merged (drop=id);
```

When used in the DATA statement, the DROP= option simply drops the variables from the new data set. However, the variables are still read from the original data set and are available for processing within the DATA step.

Additional Features

The DATA step provides a large number of other programming features for manipulating data when you combine data sets. For example, you can

- use IF-THEN/ELSE logic to control processing based on one or more conditions
- specify additional data set options
- perform calculations
- create new variables
- process variables in arrays
- use SAS functions
- use special variables such as FIRST. and LAST. to control processing.

You can also combine SAS data sets in other ways:

- You can perform one-to-one merging, which creates a data set that contains all of the variables and observations from each contributing data set. Observations are combined based on their relative position in each data set.

  One-to-one merging is the same as one-to-one reading, with two exceptions:

  - You use the MERGE statement instead of multiple SET statements.
  - The DATA step reads all observations from all data sets.

  ```
  data work.onemerge;
     merge clinic.demog clinic.visit;
  run;
  ```

- You can perform a conditional merge, using DO loops or other conditional statements:

```
data work.combine;
   set sales.pounds;
   do while(not(begin le date le last));
      set sales.rate;
   end;
```

```
   Dollars=(sales*1000)*rate;
run;
```

Additional Note You can learn about DO loops in "Generating Data with DO Loops" on page 463 .

- You can read the same data set in more than one SET statement:

```
data work.combine(drop=totpay);
   if _n_=1 then do until(last);
      set sales.budget(keep=payroll) end=last;
      totpay+payroll;
      end;
   set sales.budget;
   Percent=payroll/totpay;
run;
```

Chapter Summary

Text Summary

One-to-One Merging

You can combine data sets with one-to-one merging (combining) by including multiple SET statements in a DATA step. When you perform one-to-one merging, the new data set contains all the variables from all the input data sets. If the data sets contain same-named variables, the values that are read in from the last data set replace those that were read in from earlier ones. The number of observations in the new data set is the number of observations in the smallest original data set.

```
data one2one;
   set a;
   set b;
run;
```

Concatenating

To append the observations from one data set to another data set, you concatenate them by specifying the data set names in the SET statement. When SAS concatenates, data sets in the SET statement are read sequentially, in the order in which they are listed. The new data set contains all the variables and the total number of observations from all input data sets.

```
data concat;
   set a b;
run;
```

Appending

Another way to combine SAS data sets is to append one data set to another using the APPEND procedure. Although appending and concatenating are similar, there are some important differences between the two methods. The DATA step creates a new data set when concatenating. PROC APPEND adds the observations of one data set to the end of a "master" (or BASE) data set. SAS does not create a new data set nor does it read the base data set when executing the APPEND procedure.

Interleaving

If you use a BY statement when you concatenate data sets, the result is interleaving. Interleaving intersperses observations from two or more data sets, based on one or more common variables. Each input data set must be sorted or indexed based on the BY variable(s). Observations in each BY group in each data set in the SET statement are read sequentially, in the order in which the data sets and BY variables are listed, until all observations have been processed. The new data set contains all the variables and the total number of observations from all input data sets.

```
data interlv;
   set a b;
   by num;
run;
```

Match-Merging

Sometimes you need to combine observations from two or more data sets into a single observation in a new data set according to the values of a BY variable. This is match-merging, which uses a MERGE statement rather than a SET

statement to combine data sets. Each input data set must be sorted or indexed in ascending order based on the BY variable(s). During match-merging, SAS sequentially checks each observation of each data set to see whether the BY values match, and then writes the combined observation to the new data set.

```
data merged;
   merge a b;
   by num;
run;
```

Match-Merge Processing

To predict the results of match-merging correctly, you need to understand how the DATA step processes data in match-merges.

Compiling

To prepare to merge data sets, SAS

1. reads the descriptor portions of the data sets that are listed in the MERGE statement
2. reads and compiles the rest of the DATA step program
3. creates the program data vector (PDV), an area of memory where SAS holds one observation at a time
4. assigns a tracking pointer to each data set that is listed in the MERGE statement.

If variables with the same name appear in more than one data set, the variable from the first data set that contains the variable (in the order listed in the MERGE statement) determines the length of the variable.

Executing

After compiling the DATA step, SAS sequentially match-merges observations by moving the pointers down each observation of each data set and checking to see whether the BY values match.

- If *Yes*, the observations are read into the PDV in the order in which the data sets appear in the MERGE statement. Values of any same-named variable are overwritten by values of the same-named variable in subsequent data sets. SAS writes the combined observation to the new data set and retains the values in the PDV until the BY value changes in all the data sets.
- If *No*, SAS determines which BY value comes first and reads the observation that contains this value into the PDV. Then the observation is written to the new data set.

When the BY value changes in all the input data sets, the PDV is initialized to missing. The DATA step merge continues to process every observation in each data set until it has processed all observations in all data sets.

Handling Unmatched Observations and Missing Values

All observations that are written to the PDV, including observations that have missing data and no matching BY values, are written to the output data set.

- If an observation contains missing values for a variable, then the observation in the output data set contains the missing values as well. Observations that have missing values for the BY variable appear at the top of the output data set.
- If an input data set doesn't have a matching BY value, then the observation in the output data set contains missing values for the variables that are unique to that input data set.

Renaming Variables

Sometimes you might have same-named variables in more than one input data set. In this case, match-merging overwrites values of the same-named variable in the first data set with values of the same-named variable in subsequent data sets. To prevent overwriting, use the RENAME= data set option in the MERGE statement to rename variables.

Excluding Unmatched Observations

By default, match-merging combines all observations in all input data sets. However, you might want to select only observations that match for two or more input data sets. To exclude unmatched observations, use the IN= data set option

and the subsetting IF statement in your DATA step. The IN= data set option creates a variable to indicate whether the data set contributed data to the current observation. The subsetting IF statement then checks the IN= values and writes to the merged data set only matching observations.

Selecting Variables

You can specify the variables you want to drop or keep by using the DROP= and KEEP= data set options. When match-merging, you can specify these options in either the DATA statement or the MERGE statement, depending on whether or not you want to reference values of the variables in that DATA step. When used in the DATA statement, the DROP= option simply drops the variables from the new data set. However, they are still read from the original data set and are available within the DATA step.

Syntax

One-to-One Merging

```
LIBNAME libref 'SAS-data-library';
DATA output-SAS-data-set;
        SET SAS-data-set-1;
        SET SAS-data-set-2;
RUN;
```

Concatenating

```
DATA output-SAS-data-set;
        SET SAS-data-set-1 SAS-data-set-2;
RUN;
```

Interleaving

```
PROC SORT DATA=SAS-data-set OUT=SAS-data-set;
       BY variable(s);
RUN;
DATA output-SAS-data-set;
        SET SAS-data-set-1 SAS-data-set-2;
        BY variable(s);
RUN;
```

Match-Merging

```
PROC SORT DATA=SAS-data-set OUT=SAS-data-set;
       BY variable(s);
RUN;
DATA output-SAS-data-set (DROP=variable(s) | KEEP=variable(s));
        MERGE SAS-data-set-1 SAS-data-set-2
        (RENAME=(old-variable-name=new-variable-name)
        IN= variable DROP=variable(s) | KEEP=variable(s));
     BY variable(s);
     IF expression;
RUN;
```

Sample Programs

One-to-One Reading

```
data clinic.one2one;
   set clinic.patients;
   if age<60;
   set clinic.measure;
run;
```

Concatenating

```
data clinic.concat;
   set clinic.therapy1999 clinic.therapy2000;
run;
```

Interleaving

```
data clinic.intrleav;
   set clinic.therapy1999 clinic.therapy2000;
   by month;
```

```
run;
```

Match-Merging

```
data clinic.merged(drop=id);
   merge clinic.demog(in=indemog
                      rename=(date=BirthDate))
         clinic.visit(drop=weight in=invisit
                      rename=(date=VisitDate));
   by id;
   if indemog and invisit;
run;
```

Points to Remember

- You can rename any number of variables in each occurrence of the RENAME= option.
- In match-merging, the IN= data set option can apply to any data set in the MERGE statement. The RENAME=, DROP=, and KEEP= options can apply to any data set in the DATA or MERGE statements.
- Use the KEEP= option instead of the DROP= option if more variables are dropped than kept.
- When you specify multiple data set options for a particular data set, enclose them in a single set of parentheses.

Chapter Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which program will combine Brothers.One and Brothers.Two to produce Brothers.Three? ?

Brothers.One

VarX	VarY
1	Groucho
3	Harpo
5	Karl

+

Brothers.Two

VarX	VarZ
2	Chico
4	Zeppo

=

Brothers.Three

VarX	VarY	VarZ
2	Groucho	Chico
4	Harpo	Zeppo

a.
```
data brothers.three;
   set brothers.one;
   set brothers.two;
run;
```

b.
```
data brothers.three;
   set brothers.one brothers.two;
run;
```

c.
```
data brothers.three;
   set brothers.one brothers.two;
   by varx;
run;
```

d.
```
data brothers.three;
   merge brothers.one brothers.two;
   by varx;
run;
```

2. Which program will combine Actors.Props1 and Actors.Props2 to produce Actors.Props3? ?

Actors.Props1

Actor	Prop
Curly	Anvil
Larry	Ladder
Moe	Poker

+

Actors.Props2

Actor	Prop
Curly	Ladder
Moe	Pliers

=

Actors.Props3

Actor	Prop
Curly	Anvil
Curly	Ladder
Larry	Ladder
Moe	Poker
Moe	Pliers

a.
```
data actors.props3;
   set actors.props1;
   set actors.props2;
run;
```

b.
```
data actors.props3;
   set actors.props1 actors.props2;
run;
```

c.
```
data actors.props3;
   set actors.props1 actors.props2;
   by actor;
run;
```

d.
```
data actors.props3;
   merge actors.props1 actors.props2;
   by actor;
run;
```

3. If you submit the following program, which new data set is created? ?

Work.Dataone

Career	Supervis	Finance
72	26	9
63	76	7
96	31	7
96	98	6
84	94	6

Work.Datatwo

Variety	Feedback	Autonomy
10	11	70
85	22	93
83	63	73
82	75	97
36	77	97

```
data work.jobsatis;
   set work.dataone work.datatwo;
run;
```

Career	Supervis	Finance	Variety	Feedback	Autonomy
72	26	9	.	.	.
63	76	7	.	.	.
96	31	7	.	.	.
96	98	6	.	.	.

a.

84	94	6	.	.	.
.	.	.	10	11	70
.	.	.	85	22	93
.	.	.	83	63	73
.	.	.	82	75	97
.	.	.	36	77	97

b.

Career	Supervis	Finance	Variety	Feedback	Autonomy
72	26	9	10	11	70
63	76	7	85	22	93
96	31	7	83	63	73
96	98	6	82	75	97
84	94	6	36	77	97

c.

Career	Supervis	Finance
72	26	9
63	76	7
96	31	7
96	98	6
84	94	6
10	11	70
85	22	93
83	63	73
82	75	97
36	77	97

d. none of the above

4. If you concatenate the data sets below in the order shown, what is the value of Sale in observation 2 of the new data set? ?

Sales.Reps

ID	Name
1	Nay Rong
2	Kelly Windsor
3	Julio Meraz
4	Richard Krabill

Sales.Close

ID	Sale
1	$28,000
2	$30,000
2	$40,000
3	$15,000
3	$20,000
3	$25,000
4	$35,000

Sales.Bonus

ID	Bonus
1	$2,000
2	$4,000
3	$3,000
4	$2,500

a. missing

b. **$30,000**

c. **$40,000**

d. you cannot concatenate these data sets

5. What happens if you merge the following data sets by the variable SSN? ?

1st

SSN	Age
029-46-9261	39
074-53-9892	34
228-88-9649	32
442-21-8075	12
446-93-2122	36
776-84-5391	28
929-75-0218	27

2nd

SSN	Age	Date
029-46-9261	37	02/15/95
074-53-9892	32	05/22/97
228-88-9649	30	03/04/96
442-21-8075	10	11/22/95
446-93-2122	34	07/08/96
776-84-5391	26	12/15/96
929-75-0218	25	04/30/97

a. The values of Age in the 1st data set overwrite the values of Age in the 2nd data set.

b. The values of Age in the 2nd data set overwrite the values of Age in the 1st data set.

c. The DATA step fails because the two data sets contain same-named variables that have different values.

d. The values of Age in the 2nd data set are set to missing.

6. Suppose you merge data sets Health.Set1 and Health.Set2 below: ?

Health.Set1

ID	Sex	Age
1129	F	48
1274	F	50
1387	F	57
2304	F	16
2486	F	63
4425	F	48
4759	F	60
5438	F	42
6488	F	59
9012	F	39
9125	F	56

Health.Set2

ID	Height	Weight
1129	61	137
1387	64	142
2304	61	102
5438	62	168
6488	64	154
9012	63	157
9125	64	159

Which output does the following program create?

```
data work.merged;
   merge health.set1(in=in1) health.set2(in=in2);
   by id;
   if in1 and in2;
run;
proc print data=work.merged;
run;
```

a.

Obs	ID	Sex	Age	Height	Weight
1	1129	F	48	61	137
2	1274	F	50	.	.
3	1387	F	57	64	142
4	2304	F	16	61	102
5	2486	F	63	.	.
6	4425	F	48	.	.
7	4759	F	60	.	.
8	5438	F	42	62	168
9	6488	F	59	64	154
10	9012	F	39	63	157
11	9125	F	56	64	159

Obs	ID	Sex	Age	Height	Weight
1	1129	F	48	61	137
2	1387	F	50	64	142
3	2304	F	57	61	102
4	5438	F	16	62	168
5	6488	F	63	64	154
6	9012	F	48	63	157
7	9125	F	60	64	159

b.

8	5438	F	42	.	.
9	6488	F	59	.	.
10	9012	F	39	.	.
11	9125	F	56	.	.

c.

Obs	ID	Sex	Age	Height	Weight
1	1129	F	48	61	137
2	1387	F	57	64	142
3	2304	F	16	61	102
4	5438	F	42	62	168
5	6488	F	59	64	154
6	9012	F	39	63	157
7	9125	F	56	64	159

d. none of the above

7. The data sets Ensemble.Spring and Ensemble.Sum both contain a variable named Blue. How do you prevent the values of the variable Blue from being overwritten when you merge the two data sets? ?

a.
```
data ensemble.merged;
   merge ensemble.spring(in=blue)
         ensemble.summer;
   by fabric;
run;
```

b.
```
data ensemble.merged;
   merge ensemble.spring(out=blue)
         ensemble.summer;
   by fabric;
run;
```

c.
```
data ensemble.merged;
   merge ensemble.spring(blue=navy)
         ensemble.summer;
   by fabric;
run;
```

d.
```
data ensemble.merged;
   merge ensemble.spring(rename=(blue=navy))
         ensemble.summer;
   by fabric;
run;
```

8. What happens if you submit the following program to merge Blood.Donors1 and Blood.Donors2, shown below? ?

```
data work.merged;
   merge blood.donors1 blood.donors2;
   by id;
run
```

Blood.Donors1

ID	Type	Units
2304	O	16
1129	A	48
1129	A	50
1129	A	57
2486	B	63

Blood.Donors2

ID	Code	Units
6488	65	27
1129	63	32
5438	62	39
2304	61	45
1387	64	67

a. The Merged data set contains some missing values because not all observations have matching observations in the other data set.

b. The Merged data set contains eight observations.

c. The DATA step produces errors.

d. Values for Units in Blood.Donors2 overwrite values of Units in Blood.Donors1.

9. If you merge Company.Staff1 and Company.Staff2 below by ID, how many observations does the new data set contain? ?

Company.Staff1

ID	Name	Dept	Project
000	Miguel	A12	Document
111	Fred	B45	Survey
222	Diana	B45	Document
888	Monique	A12	Document
999	Vien	D03	Survey

Company.Staff2

ID	Name	Hours
111	Fred	35
222	Diana	40
777	Steve	0
888	Monique	37

a. 4

b. 5

c. 6

d. 9

10. If you merge data sets Sales.Reps, Sales.Close, and Sales.Bonus by ID, what is the value of Bonus in the third observation in the new data set? ?

Sales.Reps

ID	Name
1	Nay Rong
2	Kelly Windsor
3	Julio Meraz
4	Richard Krabill

Sales.Close

ID	Sale
1	$28,000
2	$30,000
2	$40,000
3	$15,000
3	$20,000
3	$25,000
4	$35,000

Sales.Bonus

ID	Bonus
1	$2,000
2	$4,000
3	$3,000
4	$2,500

a. **`$4,000`**

b. **`$3,000`**

c. missing

d. can't tell from the information given

Answers

1. Correct answer: a

This example is a case of one-to-one matching, which requires multiple SET statements. Where same-named variables occur, values that are read from the second data set replace those read from the first data set. Also, the number of observations in the new data set is the number of observations in the smallest original data set.

2. Correct answer: c

This is a case of interleaving, which requires a list of data set names in the SET statement and one or more BY variables in the BY statement. Notice that observations in each BY group are read sequentially, in the order in which the data sets and BY variables are listed. The new data set contains all the variables from all the input data sets, as well as the total number of records from all input data sets.

3. Correct answer: a

Concatenating appends the observations from one data set to another data set. The new data set contains the total number of records from all input data sets, so b is incorrect. All the variables from all the input data sets appear in the new data set, so c is incorrect.

4. Correct answer: a

The concatenated data sets are read sequentially, in the order in which they are listed in the SET statement. The second observation in Sales.Reps does not contain a value for Sale, so a missing value appears for this variable. (Note that if you merge the data sets, the value of Sale for the second observation is **`$30,000`**.)

5. Correct answer: b

If you have variables with the same name in more than one input data set, values of the same-named variable in the first data set in which it appears are overwritten by values of the same-named variable in subsequent data sets.

6. Correct answer: c

The DATA step uses the IN= data set option and the subsetting IF statement to exclude unmatched observations from the output data set. So a and b, which contain unmatched observations, are incorrect.

7. Correct answer: d

Match-merging overwrites same-named variables in the first data set with samenamed variables in subsequent data sets. To prevent overwriting, rename variables by using the RENAME= data set option in the MERGE statement.

8. Correct answer: c

The two input data sets are not sorted by values of the BY variable, so the DATA step produces errors and stops processing.

9. Correct answer: c

In this example, the new data set contains one observation for each unique value of ID. The merged data set is shown below.

ID	Name	Dept	Project	Hours
000	Miguel	A12	Document	.
111	Fred	B45	Survey	35
222	Diana	B45	Document	40
777	Steve			0
888	Monique	A12	Document	37
999	Vien	D03	Survey	.

10. Correct answer: a

In the new data set, the third observation is the second observation for ID number 2 (Kelly Windsor). The value for Bonus is retained from the previous observation because the BY variable value didn't change. The new data set is shown below.

ID	Name	Sale	Bonus
1	Nay Rong	$28,000	$2,000
2	Kelly Windsor	$30,000	$4,000
2	Kelly Windsor	$40,000	$4,000
3	Julio Meraz	$15,000	$3,000
3	Julio Meraz	$20,000	$3,000
3	Julio Meraz	$25,000	$3,000
4	Richard Krabill	$35,000	$2,500