How to Prepare Your Data Sets

Determining the Structure and Contents of Data Sets

Typically, data comes from multiple sources and might be in different formats. Many applications require input data to be in a specific format before the data can be processed. Although application requirements vary, there are common factors for all applications that access, combine, and process data. You can identify these common factors for your data. Here are tasks to help you start:

- 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.

You can use the CONTENTS, DATASETS, and PRINT procedures to review the structure of your data.

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 falls 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

Finally, to obtain the desired results, you should understand how each of these methods combines observations and how each treats duplicate, missing, or unmatched values of common variables. Some of the methods require that you preprocess your data sets by sorting or creating indexes. Testing is a good first step.

Testing Your Program

Create small temporary data sets that contain a sample of rows that test all of your program's logic. If your logic is faulty and you get unexpected output, you can debug your program.

Looking at Sources of Common Problems

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

- columns that have the same name but that represent different data

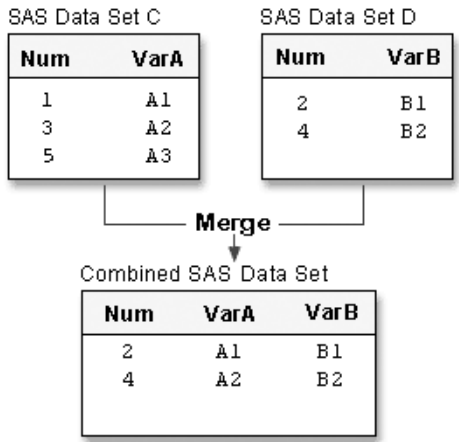
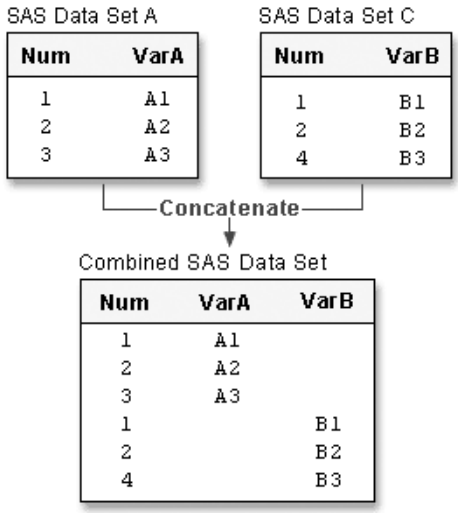
  To correct the error, you can rename columns before you combine the data sets by using the RENAME= table option in the SET or MERGE statement. As an alternative, use the DATASETS procedure to display all library management functions for all member types (except catalogs).

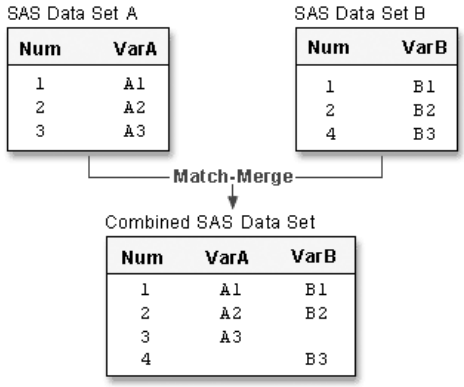
- common columns that have the same data but different attributes

Methods of Combining SAS Data Sets: The Basics

A common task in SAS programming 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.

Table 10.1 *Quick-Reference Overview of Data-Combining Methods*

Method of Combining	Illustration
One-to-one reading 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	

Method of Combining	Illustration
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	

TIP You can also use PROC SQL to join data sets according to common values.

One-to-One Reading: Details

One-to-One Reading Syntax

Use multiple SET statements in a DATA step to combine data sets. One-to-one reading combines rows from two or more data sets by creating rows that contain all of the columns from each contributing data set. Rows are combined based on their relative position in each data set. That is, the first row in one data set is combined with the first in the other, and so on. The data program stops after it has read the last row from the smallest data set.

Syntax, DATA step for one-to-one reading:

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

- *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.

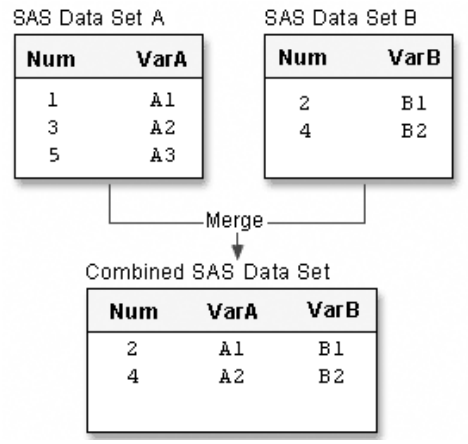
How One-to-One Reading Selects Data

The following statements are true when you perform one-to-one reading:

- 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;
  set b;
run;
```

Figure 10.1 *One-to-One Reading*



How One-to-One Reading Works

Here is a simple example of one-to-one reading.

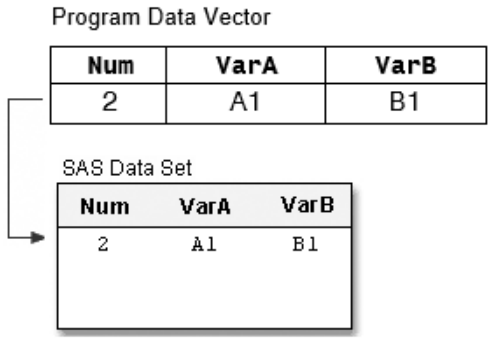
```
data one2one;
  set a;
  set b;
run;
```

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

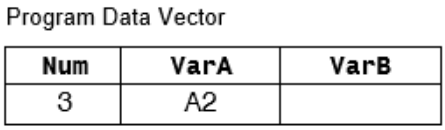
   Program Data Vector

Num	VarA
1	A1

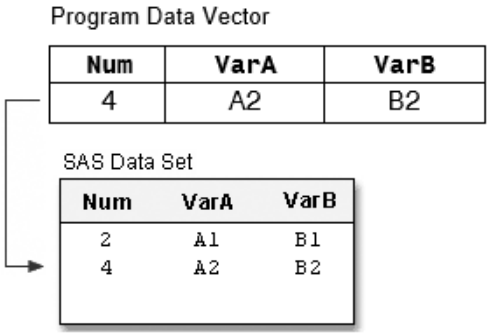
2. The second SET statement reads the first observation from data set B into the PDV, and SAS writes the contents of the PDV to the new data set. The value for Num from data set B overwrites the value for Num from data set A.



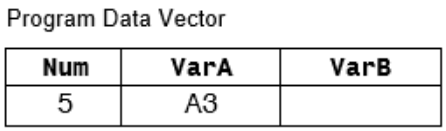
3. The first SET statement reads from data set A into the PDV.



4. The second SET statement reads the second observation from data set B, and SAS writes the contents of the PDV 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 PDV.



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 PDV, but it is not written to the output data set.

Num	VarA	VarB
2	A1	B1
4	A2	B2

Example: Using One-to-One Reading to Combine Data Sets

In the following example, you have basic patient data in Cert.Patients that you want to combine with other patient data that is stored in Cert.Measure. The height and weight data is stored in the data set Cert.Measure. Both data sets are sorted by the variable ID.

Notice that Cert.Patients contains eleven observations in which the patient age is less than 60, and Cert.Measure contains six observations.

Figure 10.2 *Example: One-to-One Reading*

SAS Data Set Cert.Patients

	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
10	8045	M	40
11	8125	M	39

SAS Data Set Cert.Measure

	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

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 work.one2one;
  set cert.patients;
  if age<60;
  set cert.measure;
run;
```

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

Figure 10.3 *The Resulting Data Set for One-to-One Reading Example*

	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

Concatenating: Details

Concatenating Syntax

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.

Syntax, DATA step for concatenating:

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

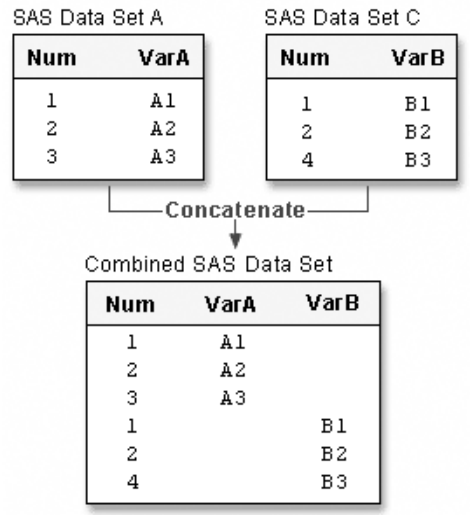
- *output-SAS-data-set* names the data set to be created.
- *SAS-data-set-1* and *SAS-data-set-2* specify the data sets to concatenate.

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 10.4 *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: Using Concatenating to Combine Data Sets

The following DATA step creates Work.Concat by concatenating Cert.Therapy2012 and Cert.Therapy2013. Each data set contains 12 observations.

```
data work.concat;
  set cert.therapy2012 cert.therapy2013;
run;
proc print data=work.concat;
run;
```

The first 12 observations in the new output data set Work.Concat were read from Cert.Therapy2012, and the last 12 observations were read from Cert.Therapy2013.

Figure 10.5 *Example: Concatenating (partial output)*

Obs	Month	Year	AerClass	WalkJogRun	Swim
1	1	2012	26	78	14
2	2	2012	32	109	19
3	3	2012	15	106	22
4	4	2012	47	115	24
5	5	2012	95	121	31
. . . *more observations* . . .					
20	8	2013	63	65	53
21	9	2013	60	49	68
22	10	2013	78	70	41
23	11	2013	82	44	58
24	12	2013	93	57	47

Match-Merging: Details

Match-Merging Syntax

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

When match-merging, use the MERGE statement rather than the SET statement to combine data sets.

Syntax, DATA step for match-merging:

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

- *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. The default sort order is ASCENDING.

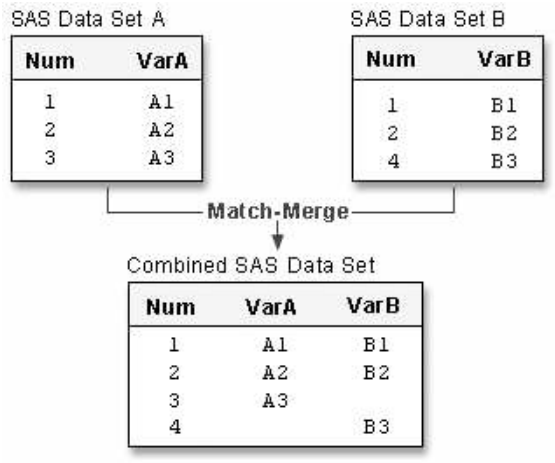
TIP Each input data set in the MERGE statement must be sorted in order of the values of the BY variable or variables, or it must have an appropriate index. Each BY variable must have the same type in all data sets to be merged.

How Match-Merging Selects Data

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 10.6 *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 your input data set does not have any observations for a value of the BY variable, then the observations in the output data set will contain missing values. The missing values are for the variables that are unique to the input data set.

TIP 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: Using Match-Merging to Combine Data Sets

The data sets Cert.Demog and Cert.Visit have been sorted as follows:

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

Figure 10.7 *HTML Output: Sorting Cert.Demog*

Obs	ID	Age	Sex	Date
1	A001	21	m	05/22/07
2	A002	32	m	06/15/06
3	A003	24	f	08/17/07
4	A004	.		01/27/06
5	A005	44	f	02/24/05
6	A007	39	m	11/11/05

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

Figure 10.8 *HTML Output: Sorting Cert.Visit*

Obs	ID	Visit	SysBP	DiasBP	Weight	Date
1	A001	1	140	85	195	11/05/09
2	A001	2	138	90	198	10/13/09
3	A001	3	145	95	200	07/04/09
4	A002	1	121	75	168	04/14/09
5	A003	1	118	68	125	08/12/09
6	A003	2	112	65	123	08/21/09
7	A004	1	143	86	204	03/30/09
8	A005	1	132	76	174	02/27/09
9	A005	2	132	78	175	07/11/09
10	A005	3	134	78	176	04/16/09
11	A008	1	126	80	182	05/22/09

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

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

Note: All observations, including unmatched observations and observations that have missing data, are written to the output data set.

Figure 10.9 *HTML Output: Match-Merging Output*

Obs	ID	Age	Sex	Date	Visit	SysBP	DiasBP	Weight
1	A001	23	M	11/05/09	1	140	85	195
2	A001	38	M	10/13/09	2	138	90	198
3	A001	35	M	07/04/09	3	145	95	200
4	A002	22	M	04/14/09	1	121	75	168
5	A003	38	F	08/12/09	1	118	68	125
6	A003	41	F	08/21/09	2	112	65	123
7	A004	26		03/30/09	1	143	86	204
8	A005	33	F	02/27/09	1	136	76	174
9	A005	31	F	07/11/09	2	132	78	175
10	A005	29	F	04/16/09	3	134	78	176
11	A007	39	M	11/11/05	.	.	.	.
12	A008	26		05/22/09	1	126	80	182

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

Note: 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.

Figure 10.10 *HTML Output: Merge in Descending Order*

Obs	ID	Age	Sex	Date	Visit	SysBP	DiasBP	Weight
1	A008	26		05/22/09	1	126	80	182
2	A007	39	M	11/11/05	.	.	.	.
3	A005	44	F	02/24/05	.	.	.	.
4	A005	33		02/27/09	1	136	76	174
5	A005	31		07/11/09	2	132	78	175
6	A005	29		04/16/09	3	134	78	176
7	A004	.		01/27/06	.	.	.	.
8	A004	26		03/30/09	1	143	86	204
9	A003	24	F	08/17/07	.	.	.	.
10	A003	38		08/12/09	1	118	68	125
11	A003	41		08/21/09	2	112	65	123
12	A002	32	M	06/15/06	.	.	.	.
13	A002	22		04/14/09	1	121	75	168
14	A001	21	M	05/22/07	.	.	.	.
15	A001	23		11/05/09	1	140	85	195
16	A001	38		10/13/09	2	138	90	198
17	A001	35		07/04/09	3	145	95	200

Match-Merge Processing

The Basics of Match-Merge Processing

The match-merging examples in this book 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 PDV.
- 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 PDV. From here, the variables are written to the output data set as a single observation.

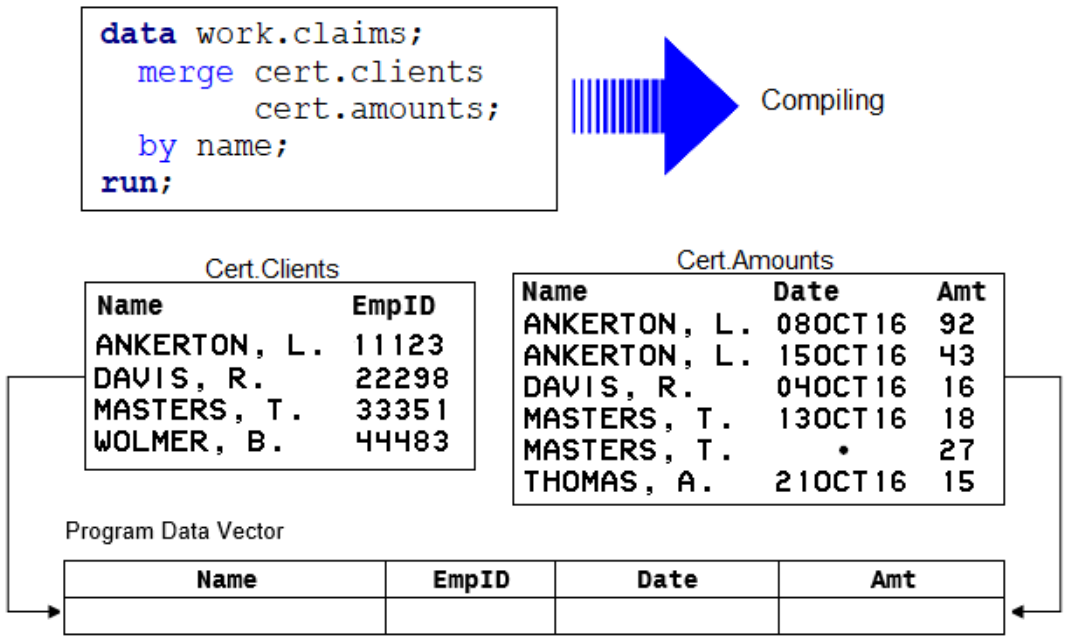
The Compilation Phase: Setting Up a New Data Set

To prepare to merge data sets, SAS does the following:

- 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 PDV for the merged data set
- assigns a tracking pointer to each data set that is listed in the MERGE statement

If there are variables with the same name in more than one data set, then the variable from the first data set (the order in which the data sets are listed in the MERGE statement) determines the length of the variable.

Figure 10.11 *The Compilation Phase: Setting Up the New Data Set*



After reading the descriptor portions of the data sets Clients and Amounts, SAS does the following:

1. creates a PDV for the new Claims data set. The PDV contains all variables from the two data sets. Note that although Name appears in both input data sets, it appears in the PDV 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 the BY values match, 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 observations. 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.

```
data work.claims;
  merge cert.clients
        cert.amounts;
  by name;
run;
```

Executing

Cert.Clients

Name	EmpID
ANKERTON, L.	11123
DAVIS, R.	22298
MASTERS, T.	33351
WOLMER, B.	44483

Cert.Amounts

Name	Date	Amt
ANKERTON, L.	08OCT16	92
ANKERTON, L.	15OCT16	43
DAVIS, R.	04OCT16	16
MASTERS, T.	13OCT16	18
MASTERS, T.	.	27
THOMAS, A.	21OCT16	15

Program Data Vector

Name	EmpID	Date	Amt
ANKERTON, L.	11123	08OCT16	92

Work.Claims

Name	EmpID	Date	Amt
ANKERTON, L.	11123	08OCT16	92

- If the BY values do not match, 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.

Cert.Clients

Name	EmpID
ANKERTON, L.	11123
DAVIS, R.	22298
MASTERS, T.	33351
WOLMER, B.	44483

Cert.Amounts

Name	Date	Amt
ANKERTON, L.	08OCT16	92
ANKERTON, L.	15OCT16	43
DAVIS, R.	04OCT16	16
MASTERS, T.	13OCT16	18
MASTERS, T.	.	27
THOMAS, A.	21OCT16	15

Program Data Vector

Name	EmpID	Date	Amt
ANKERTON, L.	11123	15OCT16	43

Work.Claims

Name	EmpID	Date	Amt
ANKERTON, L.	11123	08OCT16	92
ANKERTON, L.	11123	15OCT16	43

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

Cert.Clients

Name	EmpID
ANKERTON, L.	11123
DAVIS, R.	22298
MASTERS, T.	33351
WOLMER, B.	44483

Cert.Amounts

Name	Date	Amt
ANKERTON, L.	08OCT16	92
ANKERTON, L.	15OCT16	43
DAVIS, R.	04OCT16	16
MASTERS, T.	13OCT16	18
MASTERS, T.	.	27
THOMAS, A.	21OCT16	15

Program Data Vector

Name	EmpID	Date	Amt
		.	

Work.Claims

Name	EmpID	Date	Amt
ANKERTON, L.	11123	08OCT16	92
ANKERTON, L.	11123	15OCT16	43

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.

```
data work.claims;
  merge cert.clients
        cert.amounts;
  by name;
run;
```

Executing

Cert.Clients

Name	EmpID
ANKERTON, L.	11123
DAVIS, R.	22298
MASTERS, T.	33351
WOLMER, B.	44483

Cert.Amounts

Name	Date	Amt
ANKERTON, L.	08OCT16	92
ANKERTON, L.	15OCT16	43
DAVIS, R.	04OCT16	16
MASTERS, T.	13OCT16	18
MASTERS, T.	.	27
THOMAS, A.	21OCT16	15

Program Data Vector

Name	EmpID	Date	Amt
MASTERS, T.	33351	.	27

Work.Claims

Name	EmpID	Date	Amt
ANKERTON. L.	11123	08OCT16	92
ANKERTON. L.	11123	15OCT16	43
DAVIS, R.	22298	04OCT16	16
MASTERS, T.	33351	13OCT16	18
MASTERS, T.	33351	.	27

- If an input data set does not 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.

Cert.Clients

Name	EmpID
ANKERTON, L.	11123
DAVIS, R.	22298
MASTERS, T.	33351
WOLMER, B.	44483

Cert.Amounts

Name	Date	Amt
ANKERTON, L.	08OCT16	92
ANKERTON, L.	15OCT16	43
DAVIS, R.	04OCT16	16
MASTERS, T.	13OCT16	18
MASTERS, T.	.	27
THOMAS, A.	21OCT16	15

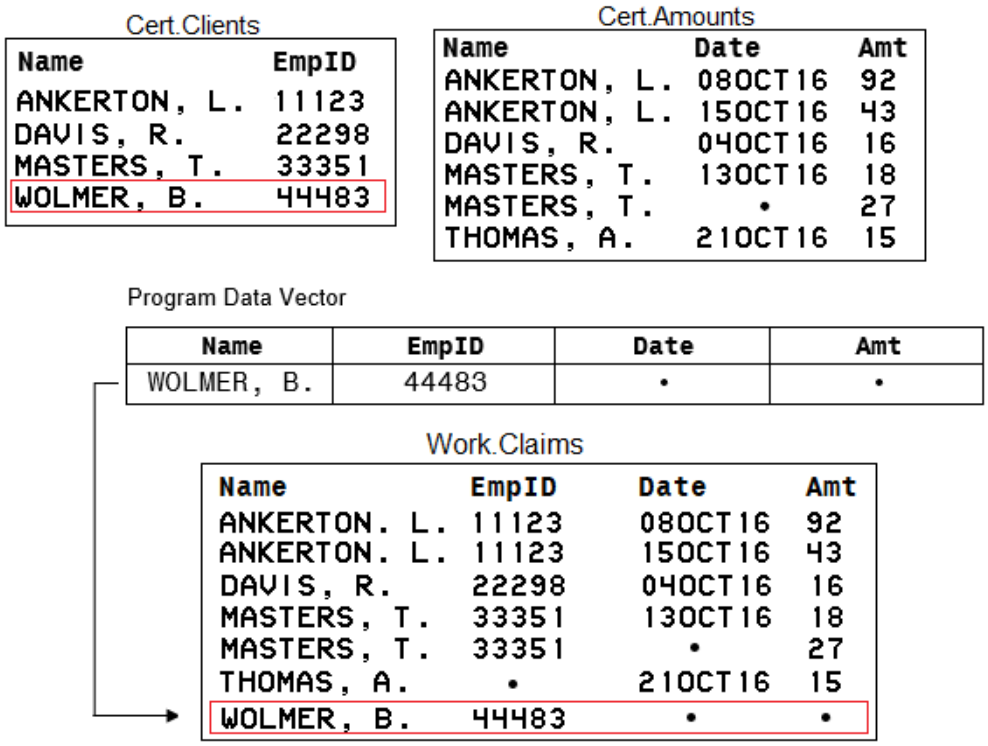
Program Data Vector

Name	EmpID	Date	Amt
THOMAS, A		21OCT16	15

Work.Claims

Name	EmpID	Date	Amt
ANKERTON. L.	11123	08OCT16	92
ANKERTON. L.	11123	15OCT16	43
DAVIS, R.	22298	04OCT16	16
MASTERS, T.	33351	13OCT16	18
MASTERS, T.	33351	.	27
THOMAS, A.	.	21OCT16	15

- The last observation in Cert.Clients would be added after the last observation in Cert.Amounts.



The PROC PRINT output is displayed below. Use the FORMAT statement for the date variable in the PRINT procedure. To learn how to apply a format, see Chapter 12, "SAS Formats and Informats," on page 205.

```
proc print data=work.claims noobs;
  format date date9.;
run;
```

Figure 10.12 *PROC PRINT Output of Merged Data*

Obs	Name	EmpID	Date	Amount
1	ANKERTON, L.	11123	08OCT2016	92
2	ANKERTON, L.	11123	15OCT2016	43
3	DAVIS, R.	22298	04OCT2016	16
4	MASTERS, T.	33351	13OCT2016	18
5	MASTERS, T.	33351	.	27
6	THOMAS, A.	.	21OCT2016	15
7	WOLMER, B.	44483	.	.

Renaming Variables

The Basics of Renaming Variables

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.

Consider Cert.Demog, which contains the variable Date (date of birth), and Cert.Visit, which also contains Date (date of the clinic visit in 2009). The DATA step below overwrites the date of birth with the date of the clinic visit.

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

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

Figure 10.13 *Renaming Variables*

Obs	ID	Age	Sex	Date	Visit	SysBP	DiasBP	Weight
1	A001	23	M	11/05/09	1	140	85	195
2	A001	38	M	10/13/09	2	138	90	198
3	A001	35	M	07/04/09	3	145	95	200
4	A002	22	M	04/14/09	1	121	75	168
5	A003	38	F	08/12/09	1	118	68	125
6	A003	41	F	08/21/09	2	112	65	123
7	A004	26		03/30/09	1	143	86	204
8	A005	33	F	02/27/09	1	136	76	174
9	A005	31	F	07/11/09	2	132	78	175
10	A005	29	F	04/16/09	3	134	78	176
11	A007	39	M	11/11/66	.	.	.	.
12	A008	26		05/22/09	1	126	80	182

RENAME Statement Syntax

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

Syntax, RENAME= data set option:

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

- 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.

TIP Use RENAME= to rename variables in the SET statement or in the output data set that is specified in the DATA statement.

Example: Renaming Variables

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

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

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

Figure 10.14 *Output for RENAME= Option*

Obs	ID	Age	Sex	BirthDate	Visit	SysBP	DiasBP	Weight	VisitDate
1	A001	23	M	05/22/86	1	140	85	195	11/05/09
2	A001	38	M	05/22/86	2	138	90	198	10/13/09
3	A001	35	M	05/22/86	3	145	95	200	07/04/09
4	A002	22	M	06/15/74	1	121	75	168	04/14/09
5	A003	38	F	08/17/83	1	118	68	125	08/12/09
6	A003	41	F	08/17/83	2	112	65	123	08/21/09
7	A004	26		01/27/70	1	143	86	204	03/30/09
8	A005	33	F	02/24/61	1	136	76	174	02/27/09
9	A005	31	F	02/24/61	2	132	78	175	07/11/09
10	A005	29	F	02/24/61	3	134	78	176	04/16/09
11	A007	39	M	11/11/66	.	.	.	.	
12	A008	26			1	126	80	182	05/22/09

Excluding Unmatched Observations

Overview

By default, DATA step match-merging combines all observations in all input data sets.

To exclude unmatched observations from your output data set, use the following in your DATA step:

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

Identifying Observation in Both Data Sets

To match-merge the data sets Cert.Demog and Cert.Visit and select only observations that appear in both data sets, 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 output SAS data set.

Syntax, IN= data set option:

(IN= *variable***)**

- 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 Cert.Demog contributes to the current observation. Otherwise, it is set to 0. Likewise, the value of Invisit depends on whether Cert.Visit contributes to an observation or not.

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

TIP To specify multiple data set options for a given data set, enclose the options in a single set of parentheses.

Selecting Matching Observations

To select only observations that appear in both Cert.Demog and Cert.Visit, specify a subsetting IF statement in the DATA step.

The subsetting IF statement checks the values of Indemog and Invisit and continues processing only those observations that meet the condition of the expression. The condition is that both Cert.Demog and Cert.Visit contribute to the observation. If the condition is met, the new observation is written to Work.Merged. Otherwise, the observation is deleted.

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

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

Figure 10.15 *Selecting Matching Observations*

Obs	ID	Age	Sex	BirthDate	Visit	SysBP	DiasBP	Weight	VisitDate
1	A001	23	M	05/22/86	1	140	85	195	11/05/09
2	A001	38	M	05/22/86	2	138	90	198	10/13/09
3	A001	35	M	05/22/86	3	145	95	200	07/04/09
4	A002	22	M	06/15/74	1	121	75	168	04/14/09
5	A003	38	F	08/17/83	1	118	68	125	08/12/09
6	A003	41	F	08/17/83	2	112	65	123	08/21/09
7	A004	26		01/27/70	1	143	86	204	03/30/09
8	A005	33	F	02/24/61	1	136	76	174	02/27/09
9	A005	31	F	02/24/61	2	132	78	175	07/11/09
10	A005	29	F	02/24/61	3	134	78	176	04/16/09

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.

It is possible to 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;
```