

CHAPTER 1

Introducing Data Relationships, Techniques for Data Manipulation, and Access Methods

<i>Overview</i>	1
<i>Determining Data Relationships</i>	1
<i>Understanding the Methods for Combining SAS Data Sets</i>	3
<i>Understanding Access Methods: Sequential versus Direct</i>	7
<i>Understanding the Tools for Combining SAS Data Sets</i>	8
<i>Understanding the Tools for Processing Information in Groups</i>	10
<i>Choosing between the DATA Step and PROC SQL</i>	10
<i>Choosing between MODIFY and UPDATE</i>	11

Overview

Many applications require input data to be in a specific format before it can be processed to produce meaningful results. Even if all of your data are already in SAS data sets, the data might come from multiple sources and might be in different formats. Therefore, you often have to take intermediate steps to logically relate and process the data before you can analyze them or create reports from them.

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

- discover how the input data are related
- select the appropriate SAS tools to complete the task

Determining Data Relationships

Relationships among multiple sources of input data exist when each source 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. Once data relationships exist, they fall into one of four categories:

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

The categories are characterized by how observations relate among the data sets. All related data fall into one of these categories. You must be able to identify the existing relationships in your data because this knowledge is crucial to understanding how input data can be processed to produce desired results.

The remainder of this section illustrates the four categories.

One-to-One

In a one-to-one relationship, typically a single observation in one data set is related to a single observation from another based on the values of one or more selected variables. A one-to-one relationship implies that each value of the selected variable occurs no more than once in each data set. When working with multiple selected variables, this relationship implies that each combination of values occurs no more than once in each data set. Figure 1.1 presents an example of two data sets with a one-to-one relationship.

Figure 1.1 One-to-One
Observations in SALARY and TAXES are related by common values for EMPNUM.

SALARY		TAXES	
EMPNUM	SALARY	EMPNUM	TAX BRACKET
1234	125000	1111	0.28
3333	85000	1234	0.33
4876	54000	3333	0.28
5489	29000	4222	0.15
		4876	0.25

One-to-Many and Many-to-One

A one-to-many or many-to-one relationship between input data sets implies that one data set has at most one observation with a specific value of the selected variable, but the other input data set might have more than one occurrence of each value. When working with multiple selected variables, this relationship implies that each combination of values occurs no more than once in one data set, but might occur more than once in the other data set. The order in which the input data sets are processed determines whether the relationship is one-to-many or many-to-one. Figure 1.2 presents an example of two data sets with a one-to-many relationship.

Figure 1.2 One-to-Many
Observations in PATIENTS and APPOINTMENTS are related by common values for ID.

PATIENTS		APPOINTMENTS	
ID	DOB	ID	APPT DATE
5MROTF	04/01/1973	AV0LYM	11/14/2008
AV0LYM	12/29/1944	AV0LYM	12/01/2008
G7DQTB	2/20/1982	AV0LYM	12/15/2008
R89CJ7	06/03/1961	R0VZWO	10/08/2008

Figure 1.3 presents an example of three related data sets. Data sets AGENTS and SALES have a one-to-many relationship. Data sets SALES and QUOTA have a many-to-one relationship.

Figure 1.3 One-to-Many and Many-to-One

Observations in data sets AGENTS, SALES, and QUOTA are related by common values for variable ID. Values of ID are unique in AGENTS and QUOTA, but not in SALES. For ID values HE01 and HH01, a one-to-many relationship exists between observations in data sets AGENTS and SALES, and a many-to-one relationship exists between observations in data sets SALES and QUOTA. Additionally, a one-to-one relationship exists between AGENTS and QUOTA.

AGENTS		SALES		QUOTA	
ID	NAME	ID	SALES	ID	QUOTA
GE01	Graham, Emily	GE01	28000	GE01	15000
HE01	Hall, Elizabeth	HE01	30000	HE01	7000
HH01	Harris, Hannah	HE01	40000	HH01	15000
RA01	Ross, Angela	HH01	15000	RA01	5000
WD01	Wright, Donald	HH01	20000	WD01	8000
		HH01	25000		
		RA01	35000		
		WD01	40000		

Many-to-Many

The many-to-many relationship implies that multiple observations from each input data set might be related based on values of one or more common variables. Figure 1.4 presents an example of two data sets with a many-to-many relationship.

Figure 1.4 Many-to-Many

Observations in data sets REPAIRS and MAINTENANCE are related by common values for variable VEHICLE. Values of VEHICLE are not unique in either data set. A many-to-many relationship exists between observations in these data sets for values 139 and 593 of VEHICLE.

REPAIRS		MAINTENANCE	
VEHICLE	REPAIR_DATE	VEHICLE	MAINT_DATE
139	08/15/2005	139	03/01/2006
139	11/03/2005	139	09/15/2006
139	07/16/2006	139	04/22/2007
139	03/10/2008	593	01/08/2008
414	09/27/2007	593	12/15/2008
593	07/03/2008	684	04/15/2008
593	09/09/2008	684	06/24/2008
		684	08/15/2008

Understanding the Methods for Combining SAS Data Sets

Generally SAS data sets are combined either vertically or horizontally.

- When combined *vertically*, you concatenate the data sets one after another, or you interleave observations from the data sets in order by one or more variables. Observations are not combined side-by-side horizontally nor are they overlaid.
- When combined *horizontally*, you usually match data sets by the values of key variables in common or by programmatically aligning observations when they do not have variables in common. The observations in the data sets can be aligned side-by-side and data in one data set can be overlaid by data from another.

You can use these methods to combine SAS data sets vertically:

- concatenating
- interleaving

You can use these methods to combine SAS data sets horizontally:

- one-to-one reading
- one-to-one merging
- match-merging
- updating

Figures 1.5 through 1.11 show basic illustrations of these methods.

The DATA step and PROC SQL can combine SAS data sets. The methods are demonstrated in the following figures primarily with basic DATA steps. A few include PROC SQL code as well.

Although not described in this section, the APPEND procedure and the APPEND statement in the DATASETS procedure can also concatenate SAS data sets. For more information, see examples throughout this book and SAS documentation.

Figure 1.5 shows vertically combining two data sets one after the other with the DATA step and with PROC SQL.

Figure 1.5 Vertical: Concatenating SAS Data Sets

Concatenating appends the observations from one data set to another data set.

ONE YEAR		TWO YEAR		ALL YEAR
2006		2006		2006
2007		2007		2007
2008	+	2008	=	2008
2009		2009		2009
2010		2010		2010
				2006
				2007
				2008
				2009
				2010

The DATA step reads ONE sequentially until all observations have been processed. Then it reads TWO sequentially until all its observations have been processed.

```
DATA Step
data all;
  set one two;
run;
```

The OUTER UNION CORR set operator in PROC SQL concatenates the two tables.

```
PROC SQL
proc sql;
  create table all as
  select * from one
  outer union corr
  select * from two;
quit;
```

Figure 1.6 presents an example of vertically combining two data sets by interleaving the values of one variable. It shows how to do this with the DATA step and with PROC SQL.

Figure 1.6 Vertical: Interleaving SAS Data Sets

Interleaving intersperses observations from two or more data sets based on values of one or more common variables.

ONE	+	TWO	=	ALL
YEAR		YEAR		YEAR
2006		2006		2006
2007		2007		2007
2008		2008		2007
2009		2009		2007
2010		2010		2008
				2008
				2009
				2009
				2010
				2010

Assume that data sets ONE and TWO were sorted by YEAR prior to the DATA step.

```
DATA Step
data all;
  set one two;
  by year;
run;
```

The ORDER BY clause arranges the observations in the table ALL in the PROC SQL step. It is not necessary to sort the table before interleaving the rows.

```
PROC SQL
proc sql;
  create table all as
  select * from one
  outer union corr
  select * from two
  order by year;
quit;
```

The DATA step in Figure 1.7 combines two data sets horizontally by doing a one-to-one reading of two data sets.

Figure 1.7 Horizontal: One-to-One Reading

One-to-one reading combines observations from two or more data sets by creating observations that contain all of the variables from each contributing data set. Observations are combined based on their relative position in each data set. That is, the first observation in one data set is aligned with the first observation in the other data set, and so on.

X	+	Y	=	XY	
XCOORD		YCOORD		XCOORD	YCOORD
25		110		25	110
35		115		35	115
45		120		45	120
55		125		55	125
		130			

```
DATA Step
data xy;
  set x;
  set y;
run;
```

The DATA step *stops* after it has read the *last* observation from the *smallest* data set. The fifth observation in Y is not present in XY.

The DATA step in Figure 1.8 combines two data sets horizontally by performing a one-to-one merge of two data sets.

Figure 1.8 Horizontal: One-to-One Merging

One-to-one merging is the same as a one-to-one reading, except that all observations from the input data sets are read. Compared to Figure 1.7, the fifth observation from data set Y is now present in XY.

X			Y			XY	
XCOORD		+	YCOORD		=	XCOORD	YCOORD
25			110			25	110
35			115			35	115
45			120			45	120
55			125			55	125
			130			.	130

DATA Step

```
data xy;
  merge x y;
run;
```

The DATA step uses the MERGE statement instead of the multiple SET statements as in Figure 1.7.

Figures 1.7 and 1.8 show only DATA steps to perform one-to-one reading and one-to-one merging and do not show equivalent PROC SQL code. Conceptually, when PROC SQL joins two tables, every row in the first table is combined with every row in the second table. The result is called a Cartesian product. This kind of join can produce a very large table or report when joining tables in this way. More likely you will want to include ON clauses, WHERE clauses, and set operators in your SELECT statements that subset the Cartesian product.

Figures 1.7 and 1.8 do not contain any common variables that can be used to subset the results. Therefore, no equivalent PROC SQL code is included. Figure 1.9 presents a PROC SQL step that produces the Cartesian product of the two data sets used in both Figures 1.7 and 1.8.

Figure 1.9 Cartesian Product

A *Cartesian product* is produced when tables are combined with PROC SQL and no conditions are specified to subset the results.

X			Y			XY	
XCOORD		+	YCOORD		=	XCOORD	YCOORD
25			110			25	110
35			115			25	115
45			120			25	120
55			125			25	125
			130			25	130
						35	110
						35	115
						35	120
						35	125
						35	130
						45	110
						45	115
						45	120
						45	125
						45	130
						55	110
						55	115
						55	120
						55	125
						55	130
						.	110
						.	115
						.	120
						.	125
						.	130

PROC SQL Step

```
proc sql;
  create table xy as
  select * from x,y;
quit;
```

Figure 1.10 presents an example of combining two data sets horizontally by match-merging them by the values of a common variable. It shows how to do this with the DATA step and with PROC SQL.

Figure 1.10 Horizontal: Match-Merging

Match-merging combines observations from two or more data sets into a single observation in a new data set based on the values of one or more common variables.

X		+	Y		=	XY		
DAY	XCOORD		DAY	YCOORD		DAY	XCOORD	YCOORD
1	25		1	110		1	25	110
2	35		3	120		2	35	.
3	45		4	125		3	45	120
4	55		5	130		4	55	125
						5	.	130

Assume that data sets X and Y were sorted by DAY prior to the DATA step.

```
DATA Step
data xy;
  merge x y;
  by day;
run;
```

Tables X and Y do not have to be sorted before submitting the PROC SQL step. The COALESCE function saves the first nonmissing value of DAY from each of the two tables and the resulting column is named DAY.

```
PROC SQL Step
proc sql;
  create table xy as
  select coalesce(x.day,y.day) as day, xcoord, ycoord
  from x full join y
  on x.day=y.day;
```

The DATA step in Figure 1.11 combines two data sets horizontally by updating one data set with information in another.

Figure 1.11 Horizontal: Updating

Updating uses information from observations in a transaction data set to delete, add, or alter information in observations in a master data set.

Assume that MASTER and TRANSACTIONS were sorted by DT prior to the DATA step. Updating a data set with the DATA step requires that the data be sorted or indexed by the values of the common variable. You can update a master data set with the UPDATE or the MODIFY statements.

MASTER			+	TRANSACTIONS			=
DT	MANAGER	SALES		DT	MANAGER	SALES	
02/01/2008	JHT	10521		02/02/2008	LKJ	10845	
02/02/2008	JHT	10761		02/05/2008	LKJ	10976	
02/03/2008	IRW	10796		02/08/2008	UTR	10754	
02/04/2008	TRP	10457		02/13/2008	LKJ	10754	
02/05/2008	JHT	10729		02/14/2008	IRW	10381	
02/06/2008	LKJ	10850					
02/07/2008	JHT	10468					
02/08/2008		10646					
02/09/2008	UTR	10302					
02/10/2008	LKJ	10971					
02/11/2008	LKJ	10757					
02/12/2008	IRW	10838					

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

MASTER		
DT	MANAGER	SALES
02/01/2008	JHT	10521
02/02/2008	JHT	10845
02/03/2008	IRW	10796
02/04/2008	TRP	10457
02/05/2008	LKJ	10976
02/06/2008	LKJ	10850
02/07/2008	JHT	10468
02/08/2008	UTR	10646
02/09/2008	UTR	10302
02/10/2008	LKJ	10971
02/11/2008	LKJ	10757
02/12/2008	IRW	10838
02/13/2008	LKJ	10754
02/14/2008	IRW	10381

DATA Step

```
data master;
  update master transactions;
  by dt;
run;
```

The PROC SQL step performs a full join of the two tables. The COALESCE function saves the first nonmissing value of the columns that are supplied to it. Note that the order of the columns that are supplied to the COALESCE function represents the column from TRANSACTIONS first so that a nonmissing value updates the value in MASTER.

PROC SQL

```
proc sql;
  create table master as
  select coalesce(transactions.dt, master.dt) as date
         format=mmdyy10.,
         coalesce(transactions.manager, master.manager) as
         manager,
         coalesce(transactions.sales, master.sales) as sales
  from master full join transactions
  on transactions.dt=master.dt;
quit;
```

Understanding Access Methods: Sequential versus Direct

SAS can access your data either sequentially or directly. When writing a SAS DATA step, your code might enable you to specify which of the two access methods SAS should use. PROC SQL decides for you which method is more efficient to use based on the code and data sets that you have specified.

- Sequential access* means that SAS accesses the observations in your data set in the order in which they appear in the physical file.
- Direct access* means that SAS goes straight to an observation in a SAS data set without having to process each observation that precedes it.

Computer resources such as CPU time, I/O, and disk storage can be conserved based on the access method that you choose.

Using Sequential Access

The simpler and perhaps more common way to process data with a DATA step is to read observations in a data set sequentially. You can read observations sequentially using the SET, MERGE, MODIFY, or UPDATE statement.

Using Direct Access

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

- by an observation number in the DATA step only
- by the value of one or more variables through a simple or composite index in the DATA step or PROC SQL

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

To access observations directly based on the values of one or more specified variables, you must first create an index for the variables. An index is a separate structure that contains the data values of the key variable or variables paired with a location identifier for the observations that contain the value.

If you are using a DATA step and an index, you would then read the data set by using the KEY= option with the SET or MODIFY statement.

With PROC SQL, specific clauses and types of queries and joins can use indexes to access data in your tables. In these situations, unless you specify otherwise, the PROC SQL internal optimizer will determine whether it is more efficient to use an index or not before executing the PROC SQL step.

Understanding the Tools for Combining SAS Data Sets

Once you understand the basics of establishing relationships among data and the ways you can combine SAS data sets, you can choose from a variety of SAS tools for accessing, combining, and processing your data. Table 1.1 lists and briefly describes the primary tools that are featured in this book. The remainder of the section describes in more detail some of the choices to make in determining how to combine and modify your data sets.

Table 1.1 Tools for Combining SAS Data Sets

Class of Tool	Statement, PROC, or Other Tool	Action Performed	Sequential	Direct	Can Use with BY Statement	Comments
DATA Step Statements	SET	Reads an observation from one or more SAS data sets.	X	X	X	Use KEY= or POINT= to access data directly.
	MERGE	Reads observations from two or more SAS data sets and joins them into single observations.	X		X	When using MERGE with BY, the data must be sorted or indexed on the BY variable.
	MODIFY	Manipulates observations in a SAS data set in place.	X	X	X	Sorted and indexed data are not required for direct access or usage with BY, but are recommended for performance.
	UPDATE	Applies transactions to observations in a master SAS data set. UPDATE does not update observations in place; it produces an updated copy of the current data set.	X		X	Both the master and transaction data sets must be sorted or indexed on the BY variable.
	BY	Controls the operation of a SET, MERGE, UPDATE, or MODIFY statement in the DATA step and sets up special grouping variables.	N/A	N/A	N/A	BY-group processing is a means of processing observations that have the same values of one or more variables.
PROCs	PROC APPEND	Adds the observations from one SAS data set to the end of another SAS data set.	X			This procedure is limited to appending one data set to another.
	PROC DATASETS with APPEND Statement	Adds the observations from one SAS data set to the end of another SAS data set.	X			The APPEND statement in this procedure is limited to appending one data set to another.
	PROC SQL	Joins rows from one or more tables and can manipulate the rows in a table in place. The maximum number of tables that PROC SQL can read is 256.	X	X		The access method is chosen by the PROC SQL internal optimizer.
Other	DATA Step Hash Objects	Enables you to quickly and efficiently store, search, and retrieve data based on lookup keys. Consists of two predefined component objects for use in the DATA step: the hash object and the hash iterator object.	N/A	N/A	N/A	Component objects are data elements that consist of attributes and methods. Attributes are the properties that specify the information that is associated with an object. Methods define the operations that an object can perform.
	IORC	An automatic variable that is created when you use the MODIFY statement or when you use the SET statement with the KEY= option.	N/A	N/A	N/A	The value of this variable is a numeric return code that indicates the status of the most recent I/O operation that used MODIFY or KEY=.
	%SYSRC	An autocall macro program that you use in conjunction with _IORC_ to test for specific I/O conditions.	N/A	N/A	N/A	

Understanding the Tools for Processing Information in Groups

Processing BY Groups in the DATA Step

When combining SAS data sets in a DATA step, it is often convenient or necessary to process observations in BY groups (that is, groups of observations that have the same value for one or more selected variables). Many examples in this book use BY-group processing with one or more SAS data sets to create a new data set.

The BY statement identifies one or more BY variables. When using the BY statement with the SET, MERGE, or UPDATE statement, your data must be sorted or indexed on the BY variable or variables.

In a DATA step, SAS identifies the beginning and end of each BY group by creating two temporary variables for each BY variable: *FIRST.variable* and *LAST.variable*. These variables are set to 1 if true and 0 if false to indicate whether that observation is the first or last in the current BY group. Using programming logic, you can test *FIRST.variable* and *LAST.variable* to determine whether the current observation is the first or last (or both first and last, or neither first nor last) in the current BY group. Testing the values of these variables in conditional processing enables you to perform certain operations at the beginning or end of a BY group.

Processing Grouped Data in PROC SQL

The same programming functionality that BY-group processing offers in the DATA step is not replicated in PROC SQL. The GROUP BY clause processes data in groups, similar to the way a BY statement in a PROC step processes data. Tables do not have to be sorted by the columns that are specified in the GROUP BY clause. The ORDER BY clause can be added to arrange the results.

Understanding BY-Group Processing with the MODIFY and BY Statements

Internally, the MODIFY statement handles BY-group processing differently from the SET, MERGE, and UPDATE statements. MODIFY creates a dynamic WHERE clause, making it possible for you to use BY-group processing without either sorting or indexing your data first. However, processing based on *FIRST.variables* and *LAST.variables* can result in multiple BY groups for the same BY values if your data are not sorted. Therefore, you might not get the expected results unless you use sorted data. And even though sorting is not required, it is often useful for improved performance.

Processing Groups with Arrays in the DATA Step

When you want to process several variables in the same way, you might be able to use arrays. Processing variables in arrays can save you time and simplify your code. Use an ARRAY statement to define a temporary grouping of variables as an array. Then use a DO loop to perform a task repetitively on all or selected elements in the array.

Choosing between the DATA Step and PROC SQL

As illustrated earlier, you can sometimes use either the DATA step or PROC SQL to combine data sets. A good understanding of your input data sets and the results you want to obtain will help you determine which tool is the better one to use to combine your data sets or tables. An additional important consideration is which method is easier for you to code and support.

In general, PROC SQL requires more computer resources than the equivalent DATA step. However, you might find coding PROC SQL in certain situations to be much simpler. What you lose in computer resources you might gain in your time to write and support the code.

The DATA step and PROC SQL follow different processes when combining data sets. It is important to understand how the two methods process the input data sets because the results of each method can produce different results.

Many of the examples in this book show how to use both tools. Discussion of efficiency and applicability of both methods is included.

Table 1.2 lists some advantages and disadvantages of the DATA step.

Table 1.2 Advantages and Disadvantages of Using the DATA Step

Advantages	Disadvantages
There is no limit to the number of input data sets, other than memory.	Data sets must be sorted by or indexed on the BY variables prior to match-merging.
Multiple data sets can be created in one DATA step.	When match-merging, the BY variables must be present in all data sets, and the names of the BY variables must match exactly.
With the SAS language in the DATA step, complex logic can be programmed that can include arrays and DO loops, and options in the SET, MERGE, UPDATE, and MODIFY statements.	An exact match on the key values must be found. Sometimes inexact matches can be made if additional programming logic is added.
Multiple BY variables enable lookups that depend on more than one variable.	

Table 1.3 lists some advantages and disadvantages of PROC SQL.

Table 1.3 Advantages and Disadvantages of Using PROC SQL

Advantages	Disadvantages
Tables do not have to be sorted or indexed, but an index might improve performance.	Complex logic can be difficult to code.
Multiple tables can be joined in one step without having common columns in all tables.	PROC SQL might require more resources for a simple join than a DATA step that uses the MERGE statement. For example, even though input tables do not have to be sorted, PROC SQL might perform a sort behind the scenes, which could require more resources than PROC SORT and a DATA step.
You can create tables, views, or query reports with the combined data.	PROC SQL can create only one table from a query.
Combining tables when variable values do not match exactly can often be much easier to accomplish than with a DATA step.	The maximum number of tables that can be joined at one time is 256.
Matching tables based on the values of variables with different names does not require renaming of the variables as it does in the DATA step.	

Choosing between MODIFY and UPDATE

You can use either the MODIFY or UPDATE statement to update a master data set with information in a transaction data set. Chapter 6 includes examples that use the UPDATE statement. Chapter 7 includes examples that use the MODIFY statement.

The MODIFY statement has many applications while the UPDATE statement is limited to updating a master data set. You can use the MODIFY statement to perform the following tasks:

- process a file sequentially to apply updates in place (without a BY statement)
- make changes to a master data set in place by applying transactions from a transaction data set
- update the values of variables by directly accessing observations based on observation numbers

- update the values of variables by directly accessing observations based on the values of one or more key variables

Only one application of MODIFY is comparable to UPDATE: using MODIFY with the BY statement to apply transactions to a data set. While MODIFY is a more powerful tool than UPDATE, UPDATE is still the tool of choice in some cases. Table 1.3 helps you choose whether to use UPDATE or MODIFY with BY.

Table 1.3 UPDATE versus MODIFY with BY

Issue	MODIFY with BY	UPDATE
Disk space	Saves disk space because it updates data in place.	Requires more disk space because it produces an updated copy of the data set.
Sort and index	For good performance, it is strongly recommended that both data sets be sorted and that the master data set be indexed.	Requires that both data sets be sorted.
When to use	Use only when you expect to process a small portion of the data set.	Use if you expect to process most of the data set.
Duplicate BY values	Allows duplicate BY values in both the master and transaction data sets.	Allows duplicate BY values in only the transaction data set.
Scope of changes	Cannot change the data set descriptor information, so changes such as adding or deleting variables or variable labels are not valid.	Can make changes that require a change in the descriptor portion of a data set, such as adding new variables.
Error checking	Automatically generates the <code>_IORC_</code> return code variable whose value can be examined for error checking.	Needs no error checking because transactions without a corresponding master record are not applied, but are added to the data set.
Data set integrity	Data can only be partially updated due to an abnormal task termination.	No data loss occurs because UPDATE works on a copy of the data.