Chapter 1: Basic Concepts

Objectives : learning about
- the structure and components of SAS programs
- the steps involved in processing SAS programs
- SAS libraries and the types of SAS files that they contain
- temporary and permanent SAS libraries
- The structure and components of SAS data sets.

SAS Programs

You can use SAS programs to access, manage, analyze, or present your data. Let's begin by looking at a simple SAS program.

```
data sasuser.admit2;
        set sasuser.admit;
        where age>39;
run;
proc print data=sasuser.admit2;
run;
```

This program creates a new SAS data set from an existing SAS data set and then prints a listing of the new data set. A SAS data set is a data file that is formatted in a way that SAS can understand.

■ Components of SAS Programs

DATA steps typically create or modify SAS data sets. They can also be used to produce custom-designed reports. For example, you can use DATA steps to
- put your data into a SAS data set
- compute values check for and correct errors in your data
- produce new SAS data sets by subsetting, merging, and updating existing data sets.

PROC (procedure) steps are pre-written routines that enable you to analyze and process the data in a SAS data set and to present the data in the form of a report. PROC steps sometimes create new SAS data sets that contain the results of the procedure. PROC steps can list, sort, and summarize data. For example, you can use PROC steps to
- create a report that lists the data
- produce descriptive statistics
- create a summary report
- produce plots and charts.

■ Processing SAS Programs

When you submit a SAS program, SAS begins reading the statements and

checking them for errors.

DATA and PROC statements signal the beginning of a new step. When SAS encounters a subsequent DATA, PROC, or RUN statement (for DATA steps and most procedures) or a QUIT statement (for some procedures), SAS stops reading statements and executes the previous step in the program. In our sample program, each step ends with a RUN statement.

```
data sasuser.admit2;
          set sasuser.admit;
          where age>39;
run;
proc print data=sasuser.admit2;
run;
```

Note that the beginning of a new step (DATA or PROC) implies the end of the previous step. Though the RUN statement is not always required between steps in a SAS program, using it can make the SAS program easier to read and debug, and it makes the SAS log easier to read.

■ Log Messages

Each time a step is executed, SAS generates a log of the processing activities and the results of the processing. The **SAS log** collects **messages about the processing** of SAS programs and **about any errors** that occur. When SAS processes our sample program, you see the log messages shown below. Notice that you get separate sets of messages for each step in the program.

```
10084   data sasuser.admit2;
10085   set sasuser.admit;
10086   where age>39;
10087   run;

NOTE: There were 10 observations read from the data set SASUSER.ADMIT.
        WHERE age>39;
NOTE: The data set SASUSER.ADMIT2 has 10 observations and 9 variables.
NOTE: DATA statement used (Total process time):
        real time               0.00 seconds
        cpu time                0.01 seconds


10088   proc print data=sasuser.admit2;
10089   run;

NOTE: There were 10 observations read from the data set SASUSER.ADMIT2.
NOTE: PROCEDURE PRINT used (Total process time):
        real time               0.06 seconds
        cpu time                0.00 seconds
```

■ Results of Processing

When the program is processed, it creates the SAS data set sasuser.Admit2 in

the DATA step. The DATA step produces messages in the SAS log, but it does not create a report or other output. It creates the following HTML report of the SAS data set Clinic.Admit2:

The SAS System

Obs	ID	Name	Sex	Age	Date	Height	Weight	ActLevel	Fee
1	2523	Johnson, R	F	43	31	63	137	MOD	149.75
2	2539	LaMance, K	M	51	4	71	158	LOW	124.80
3	2568	Eberhardt, S	F	49	27	64	172	LOW	124.80
4	2571	Nunnelly, A	F	44	19	66	140	HIGH	149.75
5	2575	Quigley, M	F	40	8	69	163	HIGH	124.80
6	2578	Cameron, L	M	47	5	72	173	MOD	124.80
7	2579	Underwood, K	M	60	22	71	191	LOW	149.75
8	2584	Takahashi, Y	F	43	29	65	123	MOD	124.80
9	2589	Wilcox, E	F	41	16	67	141	HIGH	149.75
10	2595	Warren, C	M	54	7	71	183	MOD	149.75

SAS Libraries

You've learned about SAS programs and SAS data sets. Now let's look at SAS libraries to see how SAS data sets and other SAS files are organized and stored. Every SAS file is stored in a **SAS library**, which is a collection of SAS files.

■ Storing Files Temporarily or Permanently

Depending on the library name that you use when you create a file, you can store SAS files temporarily or permanently.

Temporary SAS libraries last only for the current SAS session.

- Storing files temporarily:
  If you don't specify a library name when you create a file (or if you specify the library name Work), the file is stored in the temporary SAS data library. When you end the session, the temporary library and all of its files are deleted.

- Storing files permanently:
  To store files permanently in a SAS data library, you specify a library name other than the default library name Work. For example, by specifying the library name Clinic when you create a file, you specify that the file is to be stored in a permanent SAS data library until you delete it.

Referencing SAS Files

■ Two-Level Names

To reference a permanent SAS data set in your SAS programs, you use a two-level name:

libref.filename

ex] sasuser.admit

In the two-level name, **libref** is the name of the SAS data library that contains the file, and **filename** is the name of the file itself. A period separates the libref and filename. For example, in our sample program, **sasuser.admit** is the two-level name for the SAS data set **Admit**, which is stored in the library named **sasuser**.

■ Referencing Temporary SAS Files

To reference temporary SAS files, you can specify the default libref **Work**, a period, and the filename. For example, the two-level name **Work.Test** references the SAS data set named Test that is stored in the temporary SAS library **Work**.

Alternatively, you can use a **one-level name** (the filename only) to reference a file in a temporary SAS library. When you specify a one-level name, the default libref **Work** is assumed. For example, the one-level name **Test** also references the SAS data set named **Test** that is stored in the temporary SAS library Work.

■ Referencing Permanent SAS Files

You can see that **Sasuser.Admit** and **Sasuser.Admit2** are permanent SAS data sets because the library name is **Sasuser**, not **Work**.

So referencing a SAS file in any library **except Work** indicates that the SAS file is stored permanently. For example, when our sample program creates **Sasuser.Admit2**, it stores the new **Admit2** data set permanently in the SAS library **Sasuser**.

■ Rules for SAS Names

SAS data set names
- can be 1 to 32 characters long
- must begin with a letter (A--Z, either uppercase or lowercase) or an underscore (_)
- can continue with any combination of numbers, letters, or underscores.

These are examples of valid data set names:
- Payroll
- LABDATA1995_1997
- _EstimatedTaxPayments3

SAS Data Sets

So far, you've seen the components and characteristics of SAS programs, including how they reference SAS data sets. Data sets are one type of SAS file. There are other types of SAS files (such as catalogs), but this chapter focuses on SAS data sets. For many procedures and for some DATA step statements, data must be in the form of a SAS data set to be processed. Now let's take a closer look at SAS data sets.

■ Overview of Data Sets

As you saw in our sample program, for many of the data processing tasks that you perform with SAS, you

- access data in the form of a SAS data set
- analyze, manage, or present the data.

Conceptually, a SAS data set is a file that consists of two parts: a **descriptor portion** and a **data portion.** Sometimes a SAS data set also points to one or more indexes, which enable SAS to locate records in the data set more efficiently. (The data sets that you see in this chapter do not contain indexes.)

■ Descriptor Portion

The descriptor portion of a SAS data set contains information about the data set, including

- the name of the data set
- the date and time that the data set was created
- the number of observations
- the number of variables.

Let's look at another SAS data set. The table below lists part of the descriptor portion of the data set Sasuser.Insure, which contains insurance information for patients who are admitted to a wellness clinic. (It's a good idea to give your data set a name that is descriptive of the contents.)

Data Set Name	SASUSER.INSURE	Observations	21
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	Monday, March 07, 2016 03:52:52 PM	Observation Length	64
Last Modified	Monday, March 07, 2016 03:52:52 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

■ Data Portion

The data portion of a SAS data set is a collection of data values that are arranged in a rectangular table. In the example below, the name Johnson, R is a data value, the weight 158 is a data value, and so on.

Name	Sex	Age	Date	Height	Weight
Johnson, R	F	43	31	63	137
LaMance, K	M	51	4	71	158
Eberhardt, S	F	49	27	64	172

■ Observations (Rows)

Rows (called observations) in the data set are collections of data values that usually relate to a single object. The values Johnson, R, F, 43, 31, 63 and 137 constitute a single observation in the data set.

■ Variables (Columns)

Columns (called variables) in the data set are collections of values that describe a particular characteristic. The values Johnson, R, LaMance, K, and Eberhardt, S constitute the variable Name in the data set shown below. This data set contains 6 variables for each observation: Name, Sex, Age, Date, Height and Weight. A SAS data set can store thousands of variables.

■ Variable Attributes

In addition to general information about the data set, the descriptor portion contains information about the attributes of each variable in the data set. The attribute information includes the variable's name, type, length, format and so on.

When you write SAS programs, it's important to understand the attributes of the variables that you use. For example, you might need to combine SAS data sets that contain same-named variables. In this case, the variables must be the same type (character or numeric).

The following is a partial listing of the attribute information in the descriptor portion of the SAS data set Sasuser.Insure. Let's look at the name, type, and length variable attributes.

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
7	BalanceDue	Num	8	6.2
4	Company	Char	11	
1	ID	Char	4	
2	Name	Char	14	

5	PctInsured	Num	8	
3	Policy	Char	5	
6	Total	Num	8	

■ Name

Each variable has a name that conforms to SAS naming conventions. Variable names follow exactly the same rules as SAS data set names. Like data set names, variable names

- can be 1 to 32 characters long
- must begin with a letter (A--Z, either uppercase or lowercase) or an underscore (_)
- can continue with any combination of numbers, letters, or underscores.

■ Type

A variable's type is either **character** or **numeric**.

- Character variables, such as Name (shown below), can contain any values.
- Numeric variables, such as Policy and Total (shown below), can contain only numeric values (the digits 0 through 9, +, -, ., and E for scientific notation).

A variable's type determines how missing values for a variable are displayed. In the following data set, Name and Sex are character variables, and Age and Weight are numeric variables.

- For character variables such as Name, a **blank** represents a missing value.
- For numeric variables such as Age, a **period** represents a missing value.

■ Length

A variable's length (the number of bytes used to store it) is related to its type.

- Character variables can be up to 32,767 bytes long.
- All numeric variables have a default length of 8. Numeric values (no matter how many digits they contain) are stored as floating-point numbers in 8 bytes of storage, unless you specify a different length.

You've seen that each SAS variable has a name, type, and length. In addition, you can define format, informat, and label attributes for variables.

■ Missing Values

The rectangular arrangement of rows and columns in a SAS data set implies that every variable must exist for each observation. If a data value is unknown for a particular observation, a missing value is recorded in the SAS data set.

Managing SAS Libraries

By default, SAS defines several libraries for you:

- **Sashelp** is a permanent library that contains sample data and other files that control how SAS works at your site. This is a read-only library.
- **Sasuser** is a permanent library that contains SAS files in the Profile catalog that store your personal settings. This is also a convenient place to store your own files.
- **Work** is a temporary library for files that do not need to be saved from session to session.

You can also define **additional libraries**. In fact, often the first step in setting up your SAS session is to define the libraries.

To define a library, you assign a **library name** (a libref) to it and specify a **path**, such as a directory path. You will use the libref as the first part of the file's two-level name (libref.filename) to reference the file within the library. You can use programming statements to assign library names.

■ Assigning Librefs

To define libraries, you can use a LIBNAME statement. You can store the LIBNAME statement with any SAS program so that the SAS data library is assigned each time the program is submitted.

General form, basic LIBNAME statement:

```
LIBNAME libref 'SAS-data-library';
```

where

- libref is 1 to 8 characters long, begins with a letter or underscore, and contains only letters, numbers, or underscores.
- SAS-data-library is the name of a SAS data library in which SAS data files are stored. The specification of the physical name of the library differs by operating environment.

The LIBNAME statement below assigns the libref Clinic to the SAS data library D:\Users\Qtr\Reports in the Windows environment.

```
libname clinic 'd:\users\qtr\reports';
```

After assigning a libref, it is a good idea to check the Log window to verify that the libref has been assigned successfully.

■ How Long Librefs Remain in Effect

The LIBNAME statement is global, which means that the librefs remain in effect until you modify them, cancel them, or end your SAS session.

Therefore, the LIBNAME statement assigns the libref for the current SAS session only. Each time you begin a SAS session, you must assign a libref to each permanent SAS data library that contains files that you want to access in that session. (Remember that Work is the default libref for a temporary SAS data library.)

When you end your SAS session or delete a libref, SAS no longer has access to the files in the library. However, the contents of the library still exist on your operating system.

Viewing the Contents of SAS Libraries

■ The CONTENTS Procedure

You can use SAS windows to view the contents of a SAS library or of a SAS file. Alternatively, you can use the CONTENTS procedure to create SAS output that describes either of the following:

- the contents of a library
- the descriptor information for an individual SAS data set.

General form, basic PROC CONTENTS step:

```
PROC CONTENTS DATA=libref._ALL_ NODETAILS;
RUN;
```

where

- libref is the libref that has been assigned to the SAS library.
- _ALL_ requests a listing of all files in the library. Use a period (.) to append _ALL_ to the libref.
- NODETAILS (NODS) suppresses the printing of detailed information about each file when you specify _ALL_. You can specify NODS only when you specify _ALL_.

```
proc contents data=sasuser._all_ nods;
run;
proc contents data=sasuser.insure;
run;
```

■ The DATASETS Procedure

In addition to PROC CONTENTS, you can also use PROC DATASETS with the CONTENTS statement to view the contents of a SAS library or a SAS data set.

For example, the following PROC steps produce essentially the same output (with minor formatting differences):

```
proc datasets;
contents data=sasuser._all_ nods;
quit;
```

As with PROC CONTENTS, you can also use PROC DATASETS to display the descriptor information for a specific SAS data set.

By default, PROC CONTENTS and PROC DATASETS list variables alphabetically. To list variable names in the order of their logical position (or creation order) in the data set, you can specify the VARNUM option in PROC CONTENTS or in the CONTENTS statement in PROC DATASETS.

```
proc datasets;
contents data=sasuser.admit varnum;
quit;
proc contents data=sasuser.admit varnum;
run;
```

Setting SAS System Options

■ SAS Output

Next, let's consider the appearance and format of your SAS output. You can specify result formats to create your output as

- an HTML document
- a listing (traditional SAS output)
- both of the above.

If you create your procedure output as a SAS listing, you can also control the appearance of your output by setting system options such as

- line size (the maximum width of the log and output)
- page size (the number of lines per printed page of output)
- the display of page numbers
- the display of date and time.

The above options do not affect the appearance of HTML output.

All SAS system options have default settings that are used unless you specify otherwise. For example, page numbers are automatically displayed (unless your site modifies this default).

To modify system options, you submit an OPTIONS statement. You can place an OPTIONS statement anywhere in a SAS program to change the settings from that point onward. However, it is good programming practice to place OPTIONS statements outside of DATA or PROC steps so that your programs are easier to read and debug. Because the OPTIONS statement is global, the settings remain in effect until you modify them, or until you end your SAS session.
General form, OPTIONS statement:

OPTIONS options;

where options specifies one or more system options to be changed.

■ Examples

```
options nonumber nodate;
proc print data=sasuser.admit;
run;

options date;
proc print data=sasuser.admit;
run;

options nodate number pageno=3;
proc print data=sasuser.admit;
run;

options number pageno=1 pagesize=30;
proc print data=sasuser.admit;
run;

options number pageno=1 linesize=64;
proc print data=sasuser.y2000;
run;

options number pageno=1 linesize=200;
proc print data=sasuser.y2000;
ru ;
```

■ Handling Two-Digit Year Values: Year 2000 Compliance

If you use two-digit year values in your data lines, external files, or programming statements, you should consider another important system option, the YEARCUTOFF= option. This option specifies which 100-year span is used to interpret two-digit year values.

All versions of SAS represent dates correctly from 1582 A.D. to 20,000 A.D. (Leap years, century, and fourth-century adjustments are made automatically. Leap seconds are ignored, and SAS does not adjust for daylight saving time.) However, you should be aware of the YEARCUTOFF=value to ensure that you are properly interpreting two-digit years in data lines.

As with other system options, you specify the YEARCUTOFF= option in the OPTIONS statement:

options yearcutoff=1925;

When a two-digit year value is read, SAS interprets it based on a 100-year span that starts with the YEARCUTOFF= value. The default value of YEARCUTOFF= is 1920.

12/07/41 is interpreted as 12/07/1941

However, you can override the default and change the value of YEARCUTOFF=

to the first year of another 100-year span. For example, if you specify YEARCUTOFF=1950, then the 100-year span will be from 1950 to 2049.

options yearcutoff=1950;

Using YEARCUTOFF=1950,

12/07/41 is interpreted as 12/07/2041

Remember, the value of the YEARCUTOFF= system option affects only two-digit year values. A date value that contains a four-digit year value will be interpreted correctly even if it does not fall within the 100-year span set by the YEARCUTOFF= system option.

■ Using System Options to Specify Observations

You can also use the OBS= and FIRSTOBS= system options to specify the observations to process from SAS data sets.

You can specify either or both of these options as needed. That is, you can use
- OBS= to specify the last observation to be processed
- FIRSTOBS= to specify the first observation to be processed
- FIRSTOBS= and OBS= together to specify a range of observations to be processed.

General form, FIRSTOBS= and OBS= options in an OPTIONS statement:

OPTIONS FIRSTOBS=n;

OPTIONS OBS=n

where n is a positive integer. For FIRSTOBS=, n specifies the number of the first observation to process. For OBS=, n specifies the number of the last observation to process. By default, FIRSTOBS=1. The default value for OBS= is MAX, which is the largest signed, four-byte integer that is representable in your operating environment.

```
options firstobs=10;
proc print data=sasuser.heart;
run;

options firstobs=2 obs=10;
proc print data=sasuser.heart;
run;

options obs=max;
```

sas-code

```
data sasuser.admit2;
set sasuser.admit;
where age>39;
run;

proc print data=sasuser.admit2;
run;

proc print data=sasuser.insure;
run;

proc contents data=sasuser._all_ nods;
run;

proc datasets;
contents data=sasuser._all_ nods;
quit;

proc contents data=sasuser.insure;
run;

proc datasets;
contents data=sasuser.admit varnum;
quit;
proc contents data=sasuser.admit varnum;
run;

options nonumber nodate;
proc print data=sasuser.admit;
run;

options date;
proc print data=sasuser.admit;
run;

options nodate number pageno=3;
proc print data=sasuser.admit;
run;

options number pageno=1 pagesize=30;
proc print data=sasuser.admit;
run;

options number pageno=1 linesize=64;
proc print data=sasuser.y2000;
run;

options number pageno=1 linesize=200;
proc print data=sasuser.y2000;
run;

options firstobs=10;
proc print data=sasuser.heart;
run;

options firstobs=2 obs=10;
proc print data=sasuser.heart;
run;
```