Chapter 7

Working with SAS Data Sets

Contents

Overview

SAS/IML software has many statements for passing data from SAS data sets to matrices and from matrices to SAS data sets. You can create matrices from the variables and observations of a SAS data set in several

ways. You can create a column vector for each data set variable, or you can create a matrix where columns correspond to data set variables. You can use all the observations in a data set or use a subset of them.

You can also create a SAS data set from a matrix. The columns correspond to data set variables and the rows correspond to observations. Data management commands enable you to edit, append, rename, or delete SAS data sets from within the SAS/IML environment.

When reading a SAS data set, you can read any number of observations into a matrix either sequentially, directly by record number, or conditionally according to conditions in a WHERE clause. You can also index a SAS data set. The indexing capability facilitates retrievals by the indexed variable.

Operations on SAS data sets are performed with straightforward, consistent, and powerful statements. For example, the LIST statement can perform the following tasks:

- list the next record
- list a specified record
- list any number of specified records
- list the whole file
- list records satisfying one or more conditions
- list specified variables or all variables

If you want to read values into a matrix, use the READ statement instead of the LIST statement with the same operands and features as the LIST statement. You can specify operands that control which records and variables are used indirectly, as matrices, so that you can dynamically program the records, variables, and conditional values you want.

In this chapter, you use the SAS data set CLASS, which contains the variables NAME, SEX, AGE, HEIGHT, and WEIGHT, to learn about the following:

- opening a SAS data set
- examining the contents of a SAS data set
- displaying data values with the LIST statement
- reading observations from a SAS data set into matrices
- editing a SAS data set
- creating a SAS data set from a matrix
- displaying matrices with row and column headings
- producing summary statistics
- sorting a SAS data set
- indexing a SAS data set

- similarities and differences between the data set and the SAS DATA step

Throughout this chapter, the right angle brackets (>) indicate statements that you submit; responses from Interactive Matrix Language follow.

First, invoke the IML procedure by using the following statement:

```
> proc iml;

IML Ready
```

Opening a SAS Data Set

Before you can access a SAS data set, you must first submit a command to open it. There are three ways to open a SAS data set:

- To simply read from an existing data set, submit a USE statement to open it for Read access. The general form of the USE statement is as follows:

  USE *SAS-data-set* < *VAR operand* > < *WHERE(expression)* > **;**

  With Read access, you can use the FIND, INDEX, LIST, and READ statements with the data set.

- To read and write to an existing data set, use the EDIT statement. The general form of the EDIT statement is as follows:

  EDIT *SAS-data-set* < *VAR operand* > < *WHERE(expression)* > **;**

  This statement enables you to use both the reading statements (LIST, READ, INDEX, and FIND) and the writing statements (REPLACE, APPEND, DELETE, and PURGE).

- To create a new data set, use the CREATE statement to open a new data set for both output and input. The general form of the CREATE statement is as follows:

  CREATE *SAS-data-set* < *VAR operand* > **;**

  CREATE *SAS-data-set FROM from-name* **;**

  < *[COLNAME=column-name ROWNAME=row-name]* > *;* **;**

  Use the APPEND statement to place the matrix data into the newly created data set. If you do not use the APPEND statement, the new data set has no observations.

If you want to list observations and create matrices from the data in the SAS data set named CLASS, you must first submit a statement to open the CLASS data set. Because CLASS already exists, specify the USE statement.

Making a SAS Data Set Current

IML data processing commands work on the current data set. This feature makes it unnecessary for you to specify the data set as an operand each time. There are two current data sets, one for input and one for output. IML makes a data set the current one as it is opened. You can also make a data set current by using two setting statements, SETIN and SETOUT:

- The USE and SETIN statements make a data set current for input.
- The SETOUT statement makes a data set current for output.
- The CREATE and EDIT statements make a data set current for both input and output.

If you issue a USE, EDIT, or CREATE statement for a data set that is already open, the data set is made the current data set. To find out which data sets are open and which are current input and current output data sets, use the SHOW DATASETS statement.

The current observation is set by the last operation that performed input/output (I/O). If you want to set the current observation without doing any I/O, use the SETIN (or SETOUT) statement with the POINT option. After a data set is opened, the current observation is set to 0. If you attempt to list or read the current observation, the current observation is converted to 1. You can make the CLASS data set current for input and position the pointer at the 10th observation with the following statement:

```
> setin class point 10;
```

Displaying SAS Data Set Information

You can use SHOW statements to display information about your SAS data sets. The SHOW DATASETS statement lists all open SAS data sets and their status. The SHOW CONTENTS statement displays the variable names and types, the size, and the number of observations in the current input data set. For example, to get information for the CLASS data set, issue the following statements:

```
> use class;
> show datasets;

             LIBNAME MEMNAME   OPEN MODE   STATUS
             ------- -------   ---------   ------
             WORK    .CLASS    Input       Current Input

> show contents;

             VAR NAME   TYPE   SIZE
             NAME       CHAR     8
             SEX        CHAR     8
```

```
AGE         NUM      8
HEIGHT      NUM      8
WEIGHT      NUM      8
Number of Variables:    5
Number of Observations: 19
```

As you can see, CLASS is the only data set open. The USE statement opens it for input, and it is the current input data set. The full name for CLASS is WORK.CLASS. The libref is the default, WORK. The next section tells you how to change the libref to another name.

Referring to a SAS Data Set

The USE, EDIT, and CREATE statements take as their first operand the data set name. This name can have either one or two levels. If it is a two-level name, the first level refers to the name of the SAS data library; the second name is the data set name. If the libref is WORK, the data set is stored in a directory for temporary data sets; these are automatically deleted at the end of the session. Other librefs are associated with SAS data libraries by using the LIBNAME statement.

If you specify only a single name, then IML supplies a default libref. At the beginning of an IML session, the default libref is SASUSER if SASUSER is defined as a libref or WORK otherwise. You can reset the default libref by using the RESET DEFLIB statement. If you want to create a permanent SAS data set, you must specify a two-level name by using the RESET DEFLIB statement (see the chapter on SAS files in *SAS Language Reference: Concepts* for more information about permanent SAS data sets):

```
> reset deflib=name;
```

Listing Observations

You can list variables and observations in a SAS data set with the LIST statement. The general form of the LIST statement is as follows:

LIST < *range* > < *VAR operand* > < *WHERE(expression)* > **;**

where

range specifies a range of observations.

operand selects a set of variables.

expression is an expression that is evaluated as being true or false.

The next three sections discuss how to use each of these clauses with the CLASS data set.

Specifying a Range of Observations

You can specify a range of observations with a keyword or by record number by using the POINT option. You can use the *range* operand with the data management statements DELETE, FIND, LIST, READ, and REPLACE.

You can specify *range* with any of the following keywords:

ALL	specifies all observations.
CURRENT	specifies the current observation.
NEXT < *number* >	specifies the next observation or next *number* of observations.
AFTER	specifies all observations after the current one.
POINT *operand*	specifies observations by number, where *operand* can be one of the following:

Operand	Example
a single record number	`point 5`
a literal giving several record numbers	`point {2 5 10}`
the name of a matrix that contains record numbers	`point p`
an expression in parentheses	`point (p+1)`

If you want to list all observations in the CLASS data set, use the keyword ALL to indicate that the range is all observations. The following example demonstrates the use of this keyword:

```
> list all;
```

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	JOYCE	F	11.0000	51.3000	50.5000
2	THOMAS	M	11.0000	57.5000	85.0000
3	JAMES	M	12.0000	57.3000	83.0000
4	JANE	F	12.0000	59.8000	84.5000
5	JOHN	M	12.0000	59.0000	99.5000
6	LOUISE	F	12.0000	56.3000	77.0000
7	ROBERT	M	12.0000	64.8000	128.0000
8	ALICE	F	13.0000	56.5000	84.0000
9	BARBARA	F	13.0000	65.3000	98.0000
10	JEFFREY	M	13.0000	62.5000	84.0000
11	CAROL	F	14.0000	62.8000	102.5000
12	HENRY	M	14.0000	63.5000	102.5000
13	ALFRED	M	14.0000	69.0000	112.5000
14	JUDY	F	14.0000	64.3000	90.0000
15	JANET	F	15.0000	62.5000	112.5000
16	MARY	F	15.0000	66.5000	112.0000
17	RONALD	M	15.0000	67.0000	133.0000
18	WILLIAM	M	15.0000	66.5000	112.0000
19	PHILIP	M	16.0000	72.0000	150.0000

Without a *range* specification, the LIST statement lists only the current observation, which in this example is now the last observation because of the previous LIST statement. Here is the result of using the LIST statement:

```
> list;

        OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
         19 PHILIP   M          16.0000   72.0000  150.0000
```

Use the POINT keyword with record numbers to list specific observations. You can follow the keyword POINT with a single record number or with a literal giving several record numbers. Here are two examples:

```
> list point 5;

        OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
          5 JOHN     M          12.0000   59.0000   99.5000

> list point {2 4 9};

        OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
          2 THOMAS   M          11.0000   57.5000   85.0000
          4 JANE     F          12.0000   59.8000   84.5000
          9 BARBARA  F          13.0000   65.3000   98.0000
```

You can also indicate the range indirectly by creating a matrix that contains the records you want to list, as in the following example:

```
> p={2 4 9};
> list point p;

        OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
          2 THOMAS   M          11.0000   57.5000   85.0000
          4 JANE     F          12.0000   59.8000   84.5000
          9 BARBARA  F          13.0000   65.3000   98.0000
```

The *range* operand is usually listed first when you are using the access statements DELETE, FIND, LIST, READ, and REPLACE. The following table shows access statements and their default ranges:

Statement	Default Range
LIST	current
READ	current
FIND	all
REPLACE	current
APPEND	always at end
DELETE	current

Selecting a Set of Variables

You can use the VAR clause to select a set of variables. The general form of the VAR clause is as follows:

VAR *operand* **;**

where *operand* can be specified by using one of the following items:

- a literal that contains variable names
- the name of a matrix that contains variable names
- an expression in parentheses yielding variable names
- one of the following keywords:

  ALL for all variables

  CHAR for all character variables

  NUM for all numeric variables

The following examples show all possible ways you can use the VAR clause:

```
var {time1 time5 time9};  /* a literal giving the variables   */
var time;                 /* a matrix that contains the names */
var('time1':'time9');     /* an expression                    */
var _all_;                /* a keyword                        */
```

For example, to list students' names from the CLASS data set, use the VAR clause with a literal, as in the following statement:

```
> list point p var{name};

            OBS NAME
         ------ --------
              2 THOMAS
              4 JANE
              9 BARBARA
```

To list AGE, HEIGHT, and WEIGHT, you can use the VAR clause with a matrix giving the variables, as in the following statements:

```
> v={age height weight};
> list point p var v;

           OBS       AGE    HEIGHT    WEIGHT
        ------ --------- --------- ---------
             2   11.0000   57.5000   85.0000
             4   12.0000   59.8000   84.5000
             9   13.0000   65.3000   98.0000
```

The VAR clause can be used with the following statements for the tasks described:

Statement	VAR Clause Function
APPEND	specifies which IML variables contain data to append to the data set
CREATE	specifies the variables to go in the data set
EDIT	limits which variables are accessed
LIST	specifies which variables to list
READ	specifies which variables to read
REPLACE	specifies which data set variable's data values to replace with corresponding IML variable data values
USE	limits which variables are accessed

Selecting Observations

The WHERE clause conditionally selects observations, within the *range* specification, according to conditions given in the *expression*. The general form of the WHERE clause is as follows:

WHERE *variable comparison-op operand* **;**

where

variable is a variable in the SAS data set.

comparison-op is one of the following comparison operators:

<	less than
<=	less than or equal to
=	equal to
>	greater than
>=	greater than or equal to
^ =	not equal to
?	contains a given string
^ ?	does not contain a given string
=:	begins with a given string
=*	sounds like or is spelled like a given string

operand is a literal value, a matrix name, or an expression in parentheses.

WHERE comparison arguments can be matrices. For the following operators, the WHERE clause succeeds if *all* the elements in the matrix satisfy the condition:

^ = ^ ? < <= > >=

For the following operators, the WHERE clause succeeds if *any* of the elements in the matrix satisfy the condition:

= ? =: =*

Logical expressions can be specified within the WHERE clause by using the AND (&) and OR (|) operators. The general form is as follows:

clause&*clause* (for an AND clause)
clause | *clause* (for an OR clause)

where *clause* can be a comparison, a parenthesized clause, or a logical expression clause that is evaluated by using operator precedence.

For example, to list the names of all males in the data set CLASS, use the following statement:

```
> list all var{name} where(sex='M');
```

OBS	NAME
2	THOMAS
3	JAMES
5	JOHN
7	ROBERT
10	JEFFREY
12	HENRY
13	ALFRED
17	RONALD
18	WILLIAM
19	PHILIP

The WHERE comparison arguments can be matrices. In the following cases that use the =* operator, the comparison is made to each name to find a string that sounds like or is spelled like the given string or strings:

```
> n={name sex age};
> list all var n where(name=*{"ALFRED","CAROL","JUDY"});
```

OBS	NAME	SEX	AGE
11	CAROL	F	14.0000
13	ALFRED	M	14.0000
14	JUDY	F	14.0000

```
> list all var n where(name=*{"JON","JAN"});
```

OBS	NAME	SEX	AGE
4	JANE	F	12.0000
5	JOHN	M	12.0000

To list AGE, HEIGHT, and WEIGHT for all students in their teens, use the following statement:

```
> list all var v where(age>12);
```

```
   OBS       AGE     HEIGHT     WEIGHT
------ --------- --------- ---------
     8   13.0000   56.5000   84.0000
     9   13.0000   65.3000   98.0000
    10   13.0000   62.5000   84.0000
    11   14.0000   62.8000  102.5000
    12   14.0000   63.5000  102.5000
    13   14.0000   69.0000  112.5000
    14   14.0000   64.3000   90.0000
    15   15.0000   62.5000  112.5000
    16   15.0000   66.5000  112.0000
    17   15.0000   67.0000  133.0000
    18   15.0000   66.5000  112.0000
    19   16.0000   72.0000  150.0000
```

NOTE: In the WHERE clause, the expression on the left side refers to values of the data set variables, and the expression on the right side refers to matrix values. You cannot use comparisons that involve more than one data set variable in a single comparison; for example, you cannot use either of the following expressions:

```
list all where(height>weight);
list all where(weight-height>0);
```

You could use the first statement if WEIGHT were a matrix name already defined rather than a variable in the SAS data set.

Reading Observations from a SAS Data Set

Transferring data from a SAS data set to a matrix is done by using the READ statement. The SAS data set you want to read data from must already be open. You can open a SAS data set with either the USE or the EDIT statement. If you already have several data sets open, you can point to the one you want with the SETIN statement, making it the current input data set. The general form of the READ statement is as follows:

READ < *range* > < *VAR operand* > < *WHERE(expression)* > **;**

< *INTO name* > **;**

where

range specifies a range of observations.

operand selects a set of variables.

expression is an expression that is evaluated as being true or false.

name names a target matrix for the data.

Using the READ Statement with the VAR Clause

Use the READ statement with the VAR clause to read variables from the current SAS data set into column vectors of the VAR clause. Each variable in the VAR clause becomes a column vector with the same name as the variable in the SAS data set. The number of rows is equal to the number of observations processed, depending on the range specification and the WHERE clause. For example, to read the numeric variables AGE, HEIGHT, and WEIGHT for all observations in the CLASS data set, use the following statements:

```
> read all var {age height weight};
```

Now use the SHOW NAMES statement to display all the matrices you have created so far in this chapter:

```
> show names;

          AGE          19 rows      1 col  num      8
          HEIGHT       19 rows      1 col  num      8
          N             1 row       3 cols char     4
          P             1 row       3 cols num      8
          V             1 row       3 cols char     6
          WEIGHT       19 rows      1 col  num      8
          Number of symbols = 8  (includes those without values)
```

You see that, with the READ statement, you have created the three numeric vectors **AGE**, **HEIGHT**, and **WEIGHT**. (Notice that the matrices you created earlier, **N**, **P**, and **V**, are also listed.) You can select the variables that you want to access with a VAR clause in the USE statement. The two previous statements can also be written as follows:

```
use class var{age height weight};
read all;
```

Using the READ Statement with the VAR and INTO Clauses

Sometimes you want to have all of the numeric variables in the same matrix so that you can determine correlations. Use the READ statement with the INTO clause and the VAR clause to read the variables listed in the VAR clause into the single matrix named in the INTO clause. Each variable in the VAR clause becomes a column of the target matrix. If there are p variables in the VAR clause and n observations are processed, the target matrix in the INTO clause is an $n \times p$ matrix.

The following statement creates a matrix **X** that contains the numeric variables of the CLASS data set. Notice the use of the keyword _NUM_ in the VAR clause to specify that all numeric variables be read.

```
> read all var _num_ into x;
> print x;
```

```
                    X
                11      51.3      50.5
                11      57.5        85
                12      57.3        83
                12      59.8      84.5
                12        59      99.5
                12      56.3        77
                12      64.8       128
                13      56.5        84
                13      65.3        98
                13      62.5        84
                14      62.8     102.5
                14      63.5     102.5
                14        69     112.5
                14      64.3        90
                15      62.5     112.5
                15      66.5       112
                15        67       133
                15      66.5       112
                16        72       150
```

Using the READ Statement with the WHERE Clause

Use the WHERE clause as you did with the LIST statement, to conditionally select observations from within the specified range. If you want to create a matrix **FEMALE** that contains the variables AGE, HEIGHT, and WEIGHT for females only, use the following statements:

```
> read all var _num_ into female where(sex="F");
> print female;
```

```
              FEMALE
                11      51.3      50.5
                12      59.8      84.5
                12      56.3        77
                13      56.5        84
                13      65.3        98
                14      62.8     102.5
                14      64.3        90
                15      62.5     112.5
                15      66.5       112
```

Now try some special features of the WHERE clause to find values that begin with certain characters (the =: operator) or that contain certain strings (the ? operator). To create a matrix **J** that contains the students whose names begin with the letter "J," use the following statements:

```
> read all var{name} into j where(name=:"J");
```

```
> print j;
```

```
J
JOYCE
JAMES
JANE
JOHN
JEFFREY
JUDY
JANET
```

To create a matrix **AL** of children with names that contains the string "AL," use the following statement:

```
> read all var{name} into al where(name?"AL");
> print al;
```

```
AL
ALICE
ALFRED
RONALD
```

Editing a SAS Data Set

You can edit a SAS data set by using the EDIT statement. You can update values of variables, mark observations for deletion, delete the marked observations, and save the changes you make. The general form of the EDIT statement is as follows:

EDIT *SAS-data-set* < *VAR operand* > < *WHERE(expression)* > **;**

where

SAS-data-set names an existing SAS data set.

operand selects a set of variables.

expression is an expression that is evaluated as being true or false.

Updating Observations

Suppose you have updated data and want to change some values in the CLASS data set. For instance, suppose the student named Henry has had a birthday since the data were added to the CLASS data set. You can do the following:

- make the CLASS data set current for input and output
- read the data

- change the appropriate data value
- replace the changed data in the data set

First, submit an EDIT statement to make the CLASS data set current for input and output. Then use the FIND statement, which finds observation numbers and stores them in a matrix, to find the observation number of the data for Henry and store it in the matrix **d**. Here are the statements:

```
> edit class;
> find all where(name={'HENRY'}) into d;
> print d;

                              D
                             12
```

The following statement lists the observation that contains the data for Henry:

```
> list point d;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
    12 HENRY    M           14.0000   63.5000  102.5000
```

As you see, the observation number is 12. Now read the value for AGE into a matrix and update its value. Finally, replace the value in the CLASS data set and list the observation that contains the data for Henry again. Here are the statements:

```
> age=15;
> replace;

                      1 observations replaced.

> list point 12;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
    12 HENRY    M           15.0000   63.5000  102.5000
```

Deleting Observations

Use the DELETE statement to mark an observation to be deleted. The general form of the DELETE statement is as follows:

DELETE < *range* > < *WHERE(expression)* > **;**

where

range specifies a range of observations.

expression is an expression that is evaluated as being true or false.

The following are examples of valid uses of the DELETE statement:

Statement	Description
`delete;`	deletes the current observation
`delete point 10;`	deletes observation 10
`delete all where (age>12);`	deletes all observations where AGE is greater than 12

If a file accumulates a number of observations marked as deleted, you can clean out these observations and renumber the remaining observations by using the PURGE statement.

Suppose the student named John has moved and you want to update the CLASS data set. You can remove the observation by using the EDIT and DELETE statements. First, find the observation number of the data for John and store it in the matrix **d** by using the FIND statement. Then submit a DELETE statement to mark the record for deletion. A deleted observation is still physically in the file and still has an observation number, but it is excluded from processing. The deleted observations appear as gaps when you list the file by observation number, as in the following example:

```
> find all where(name={'JOHN'}) into d;
> print d;

                          D
                          5

> delete point d;

                    1 observation deleted.
> list all;

        OBS NAME     SEX            AGE    HEIGHT    WEIGHT
     ------ -------- -------- --------- --------- ---------
          1 JOYCE    F          11.0000   51.3000   50.5000
          2 THOMAS   M          11.0000   57.5000   85.0000
          3 JAMES    M          12.0000   57.3000   83.0000
          4 JANE     F          12.0000   59.8000   84.5000
          6 LOUISE   F          12.0000   56.3000   77.0000
          7 ROBERT   M          12.0000   64.8000  128.0000
          8 ALICE    F          13.0000   56.5000   84.0000
          9 BARBARA  F          13.0000   65.3000   98.0000
         10 JEFFREY  M          13.0000   62.5000   84.0000
         11 CAROL    F          14.0000   62.8000  102.5000
         12 HENRY    M          15.0000   63.5000  102.5000
         13 ALFRED   M          14.0000   69.0000  112.5000
         14 JUDY     F          14.0000   64.3000   90.0000
         15 JANET    F          15.0000   62.5000  112.5000
         16 MARY     F          15.0000   66.5000  112.0000
         17 RONALD   M          15.0000   67.0000  133.0000
         18 WILLIAM  M          15.0000   66.5000  112.0000
         19 PHILIP   M          16.0000   72.0000  150.0000
```

Notice that there is a gap in the data where the deleted observation was (observation 5). To renumber the observations and close the gaps, submit the PURGE statement. Note that the PURGE statement deletes any indexes associated with a data set. Here is the statement:

```
> purge;
```

Creating a SAS Data Set from a Matrix

SAS/IML software provides the capability to create a new SAS data set from a matrix. You can use the CREATE and APPEND statements to create a SAS data set from a matrix, where the columns of the matrix become the data set variables and the rows of the matrix become the observations. Thus, an $n \times m$ matrix produces a SAS data set with m variables and n observations. The CREATE statement opens the new SAS data set for both input and output, and the APPEND statement writes to (outputs to) the data set.

Using the CREATE Statement with the FROM Option

You can create a SAS data set from a matrix by using the CREATE statement with the FROM option. This form of the CREATE statement is as follows:

CREATE *SAS-data-set FROM matrix* **;**

< *[COLNAME=column-name ROWNAME=row-name]*> **;**

where

SAS-data-set	names the new data set.
matrix	names the matrix that contains the data.
column-name	names the variables in the data set.
row-name	adds a variable that contains row titles to the data set.

Suppose you want to create a SAS data set named RATIO that contains a variable with the height-to-weight ratios for each student. You first create a matrix that contains the ratios from the matrices **HEIGHT** and **WEIGHT** that you have already defined. Next, use the CREATE and APPEND statements to open a new SAS data set called RATIO and append the observations, naming the data set variable HTWT instead of COL1.

```
htwt=height/weight;
create ratio from htwt[colname='htwt'];
append from htwt;
```

Now submit the SHOW DATASETS and SHOW CONTENTS statements:

```
> show datasets;

  LIBNAME MEMNAME   OPEN MODE   STATUS
  ------- -------   ---------   ------
  WORK    .CLASS    Update
  WORK    .RATIO    Update      Current Input    Current Output

> show contents;

  VAR NAME   TYPE   SIZE
  HTWT       NUM       8
  Number of Variables:    1
  Number of Observations: 18

> close ratio;
```

As you can see, the new SAS data set RATIO has been created. It has 18 observations and 1 variable (recall that you deleted 1 observation earlier).

Using the CREATE Statement with the VAR Clause

You can use a VAR clause with the CREATE statement to select the variables you want to include in the new data set. In the previous example, the new data set RATIO had one variable. If you want to create a similar data set but include the second variable NAME, you use the VAR clause. You could not do this with the FROM option because the variable HTWT is numeric and the variable NAME is character. The following statements create a new data set RATIO2 having the variables NAME and HTWT:

```
> create ratio2 var{name htwt};
> append;
> show contents;

            VAR NAME   TYPE   SIZE
            NAME       CHAR      8
            HTWT       NUM       8
            Number of Variables:    2
            Number of Observations: 18

> close ratio2;
```

Notice that now the variable NAME is in the data set.

Understanding the End-of-File Condition

If you try to read past the end of a data set or point to an observation greater than the number of observations in the data set, you create an end-of-file condition. If an end-of-file condition occurs inside a DO DATA iteration group, IML transfers control to the next statement outside the current DO DATA group.

The following example uses a DO DATA loop while reading the CLASS data set. It reads the variable WEIGHT in one observation at a time and accumulates the weights of the students in the IML matrix SUM. When the data are read, the total class weight is stored in the matrix SUM.

```
setin class point 0;
sum=0;
do data;
   read next var{weight};
   sum=sum+weight;
end;
print sum;
```

Producing Summary Statistics

Summary statistics on the numeric variables of a SAS data set can be obtained with the SUMMARY statement. These statistics can be based on subgroups of the data by using the CLASS clause in the SUMMARY statement. The SAVE option in the OPT clause enables you to save the computed statistics in matrices for later perusal. For example, consider the following statement.

```
> summary var {height weight} class {sex} stat{mean std} opt{save};
```

SEX	Nobs	Variable	MEAN	STD
F	9	HEIGHT	60.58889	5.01833
		WEIGHT	90.11111	19.38391
M	9	HEIGHT	64.45556	4.90742
		WEIGHT	110.00000	23.84717
All	18	HEIGHT	62.52222	5.20978
		WEIGHT	100.05556	23.43382

This summary statement gives the mean and standard deviation of the variables HEIGHT and WEIGHT for the two subgroups (male and female) of the data set CLASS. Since the SAVE option is set, the statistics of the variables are stored in matrices under the name of the corresponding variables: each column corresponds to a statistic and each row corresponds to a subgroup. Two other vectors, SEX and _NOBS_, are created. The

vector SEX contains the two distinct values of the CLASS variable SEX used in forming the two subgroups. The vector _NOBS_ has the number of observations in each subgroup.

Note that the combined means and standard deviations of the two subgroups are displayed but not saved.

More than one CLASS variable can be used, in which case a subgroup is defined by the combination of the values of the CLASS variables.

Sorting a SAS Data Set

The observations in a SAS data set can be ordered (sorted) by specific key variables. To sort a SAS data set, close the data set if it is currently open, and issue a SORT statement for the variables by which you want the observations to be ordered. Specify an output data set name if you want to keep the original data set. For example, the following statement creates a new SAS data set named SORTED:

```
> sort class out=sorted by name;
```

The new data set has the observations from the data set CLASS, ordered by the variable NAME.

The following statement sorts in place the data set CLASS by the variable NAME:

```
> sort class by name;
```

However, when the SORT statement is finished executing, the original data set is replaced by the sorted data set.

You can specify as many key variables as needed, and, optionally, each variable can be preceded by the keyword DESCENDING, which denotes that the variable that follows is to be sorted in descending order.

Indexing a SAS Data Set

Searching through a large data set for information about one or more specific observations can take a long time because the procedure must read each record. You can reduce this search time by first indexing the data set by a variable. The INDEX statement builds a special companion file that contains the values and record numbers of the indexed variables. Once the index is built, IML can use the index for queries with WHERE clauses if it decides that indexed retrieval is more efficient. Any number of variables can be indexed, but only one index is in use at a given time. Note that purging a data set with the PURGE statement results in the loss of all associated indexes.

Once you have indexed a data set, IML can use this index whenever a search is conducted with respect to the indexed variables. The indexes are updated automatically whenever you change values in indexed variables. When an index is in use, observations cannot be randomly accessed by their physical location

numbers. This means that the POINT range cannot be used when an index is in effect. However, if you purge the observations marked for deletion, or sort the data set in place, the indexes become invalid and IML automatically deletes them.

For example, if you want a list of all female students in the CLASS data set, you can first index CLASS by the variable SEX. Then use the LIST statement with a WHERE clause. Of course, the CLASS data set is small, and indexing does little if anything to speed queries with the WHERE clause. If the data set had thousands of students, though, indexing could save search time.

To index the data set by the variable SEX, submit the following statement:

```
> index sex;

  NOTE: Variable SEX indexed.
  NOTE: Retrieval by SEX.
```

Now list all students by using the following statement. Notice the ordering of the special file built by indexing by the variable SEX. Retrievals by SEX will be quick.

```
> list all;
```

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	JOYCE	F	11.0000	51.3000	50.5000
4	JANE	F	12.0000	59.8000	84.5000
6	LOUISE	F	12.0000	56.3000	77.0000
8	ALICE	F	13.0000	56.5000	84.0000
9	BARBARA	F	13.0000	65.3000	98.0000
11	CAROL	F	14.0000	62.8000	102.5000
14	JUDY	F	14.0000	64.3000	90.0000
15	JANET	F	15.0000	62.5000	112.5000
16	MARY	F	15.0000	66.5000	112.0000
2	THOMAS	M	11.0000	57.5000	85.0000
3	JAMES	M	12.0000	57.3000	83.0000
7	ROBERT	M	12.0000	64.8000	128.0000
10	JEFFREY	M	13.0000	62.5000	84.0000
12	HENRY	M	15.0000	63.5000	102.5000
13	ALFRED	M	14.0000	69.0000	112.5000
17	RONALD	M	15.0000	67.0000	133.0000
18	WILLIAM	M	15.0000	66.5000	112.0000
19	PHILIP	M	16.0000	72.0000	150.0000

Data Set Maintenance Functions

Two functions and two subroutines are provided to perform data set maintenance:

DATASETS function — obtains members in a data library. This function returns a character matrix that contains the names of the SAS data sets in a library.

CONTENTS function	obtains variables in a member. This function returns a character matrix that contains the variable names for the SAS data set specified by *libname* and *memname*. The variable list is returned in alphabetical order.
RENAME subroutine	renames a SAS data set member in a specified library.
DELETE subroutine	deletes a SAS data set member in a specified library.

See Chapter 23 for details and examples of these functions and routines.

Summary of Commands

You have seen that IML has an extensive set of commands that operate on SAS data sets. Table 7.1 summarizes the data management commands you can use to perform management tasks for which you might normally use the SAS DATA step.

Table 7.1 Data Management Commands

Command	Description
APPEND	adds observations to the end of a SAS data set
CLOSE	closes a SAS data set
CREATE	creates and opens a new SAS data set for input and output
DELETE	marks observations for deletion in a SAS data set
EDIT	opens an existing SAS data set for input and output
FIND	finds observations
INDEX	indexes variables in a SAS data set
LIST	lists observations
PURGE	purges all deleted observations from a SAS data set
READ	reads observations into IML variables
REPLACE	writes observations back into a SAS data set
RESET DEFLIB	names default libname
SAVE	saves changes and reopens a SAS data set
SETIN	selects an open SAS data set for input
SETOUT	selects an open SAS data set for output
SHOW CONTENTS	shows contents of the current input SAS data set
SHOW DATASETS	shows SAS data sets currently open
SORT	sorts a SAS data set
SUMMARY	produces summary statistics for numeric variables
USE	opens an existing SAS data set for input

Comparison with the SAS DATA Step

If you want to remain in the IML environment and mimic DATA step processing, you need to learn the basic differences between IML and the SAS DATA step:

- With SAS/IML software, you start with a CREATE statement instead of a DATA statement. You must explicitly set up all your variables with the correct attributes before you create a data set. This means that you must define character variables to have the desired string length beforehand. Numeric variables are the default, so any variable not defined as character is assumed to be numeric. In the DATA step, the variable attributes are determined from context across the whole step.
- With SAS/IML software, you must use an APPEND statement to output an observation; in the DATA step, you either use an OUTPUT statement or let the DATA step output it automatically.
- With SAS/IML software, you iterate with a DO DATA loop. In the DATA step, the iterations are implied.
- With SAS/IML software, you have to close the data set with a CLOSE statement unless you plan to exit the IML environment with a QUIT statement. The DATA step closes the data set automatically at the end of the step.
- The DATA step usually executes faster than IML.

In short, the DATA step treats the problem with greater simplicity, allowing shorter programs. However, IML has more flexibility because it is interactive and has a powerful matrix-handling capability.

Summary

In this chapter, you have learned many ways to interact with SAS data sets from within the IML environment. You learned how to open and close a SAS data set, how to make it current for input and output, how to list observations by specifying a range of observations to process, a set of variables to use, and a condition for subsetting observations. You also learned summary statistics. You also know how to read observations and variables from a SAS data set into matrices as well as create a SAS data set from a matrix of values.