Chapter 11

Reading SAS Data Sets

Overview

You can create a new SAS data set from an existing SAS data set. This chapter shows you how to use the DATA step to achieve this task. When you create your new data set, you can choose variables, select observations based on one or more conditions, and assign values conditionally. You can also assign variable attributes such as formats and labels.

Topics

- Create a new data set from an existing data set
- Use BY groups to process observations
- Read observations by observation number
- Stop processing when necessary
- Explicitly write observations to an output data set
- Detect the last observation in a data set
- Identify differences in DATA step processing for raw data and SAS data sets.

Reading a Single Data Set

General Form of a Basic DATA Step:

```
DATA SAS-data-set;      /*data set to be created*/
     SET SAS-data-set; /*data set to be read*/
     <more SAS statements>
RUN;
```

```
libname Men50 "C:\Users\name\sasuser\Men50";
data Men50.males;
     set sasuser.admit;
     where sex='M' and age>50;
run;
proc print data=Men50.males;
     title "Men Over 50";
run;
```

Men Over 50

Obs	ID	Name	Sex	Age	Date	Height	Weight	ActLevel	Fee
1	2539	LaMance, K	M	51	4	71	158	LOW	124.80
2	2579	Underwood, K	M	60	22	71	191	LOW	149.75
3	2595	Warren, C	M	54	7	71	183	MOD	149.75

Manipulating Data with DROP= or KEEP= options

You have learned that you can use the DROP= and KEEP= options to select variables. Moreover, you can use these options either in the **DATA statement** or the **SET statement**, depending on whether you want to drop (or keep) variables onto output or input.

- If you never reference certain variables and you don't want them to appear in the new data set, use a DROP= (or KEEP=) option in the SET statement.

```
data lab23.drug1h;
    set research.cltrials (drop=triglycerides uricacid);
    if placebo='YES';
run;
```

- If you do need to reference a variable in the original data set, you use the DROP= (or KEEP=) option in the DATA statement, e.g., variable placebo is used in the IF statement but you do not want to include it in the new data set.

```
data lab23.drug1h (drop=placebo);
    set research.cltrials(drop=triglycerides uricacid);
    if placebo='YES';
run;
```

Detecting the End of a Data Set

To detect the last observation, you can use the **END=** option in the SET statement.

```
data work.addtoend(drop=timemin timesec);
      set sasuser.stress2(keep=timemin timesec) end=last;
      TotalMin+timemin;
      TotalSec+timesec;
      TotalTime=totalmin*60+totalsec;
      if last;
run;
proc print data=work.addtoend noobs;
run;
```

TotalMin	TotalSec	TotalTime
272	539	16859

Here **last** is a temporary variable, whose value is 1 when the data step is processing the last observation or 0 otherwise.

Using BY-Group Processing

We have used the BY statement in

- PROC SORT to sort observations,
- PROC PRINT to group observations.

We can also use the BY statement along with the SET statement in the DATA step to group observations.

```
data temp;
    set salary;
    by dept;
run;
```

- The data set(s) listed in the SET statement must be sorted by the values of the BY variable(s), or they must have an index.
- The DATA step creates two temporary variables for each **BY variable**: FIRST.variable (where variable is the name of the BY variable) and LAST.variable. Their values are either 1 or 0, which identify the first and last observation in each BY group.

This variable …	Equals …
FIRST.variable	1 for the first observation in a BY group 0 for any other observation in a BY group
LAST.variable	1 for the last observation in a BY group 0 for any other observation in a BY group

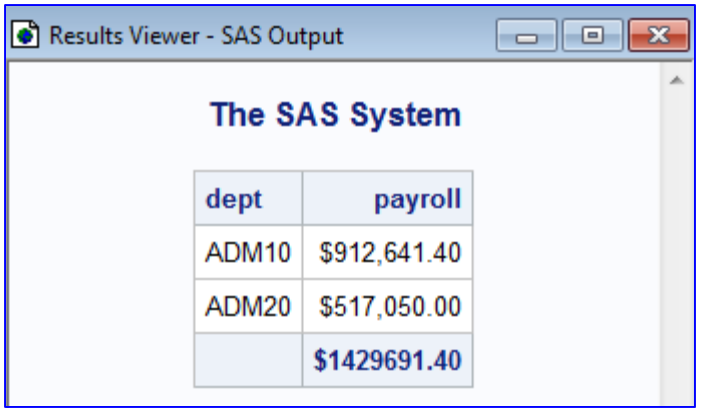
Example: Using BY-Group Processing

Compute the annual payroll by department

```
proc sort data=company.usa out=work.temp;
      by dept;
run;
data company.budget(keep=dept payroll);
      set work.temp;
      by dept;
      if wagecat='S' then yearly=wagerate*12;
      else if wagecat='H' then yearly=wagerate*2000;
      if first.dept then payroll=0;
      payroll+yearly;
      if last.dept;
run;
proc print data=company.budget noobs;
      sum payroll;
      format payroll dollar12.2;
run;
```

Selected PDV Variables

Dept	Payroll	FIRST.Dept	LAST.Dept
ADM10	70929.0	1	0
ADM10	119479.2	0	0
ADM10	173245.2	0	0
ADM10	255516.0	0	0
ADM10	293472.0	0	1
ADM20	40710.0	1	0
ADM20	68010.0	0	0
ADM20	94980.0	0	0
ADM20	136020.0	0	0
ADM20	177330.0	0	1



Specifying Multiple BY Variables

Compute the annual payroll by job type for each manager

```
proc sort data=company.usa out=work.temp2;
      by manger jobtype;
run;
data company.budget2(keep= manager jobtype payroll);
      set work.temp2;
      by manger jobtype;
      if wagecat='S' then yearly=wagerate*12;
      else if wagecat='H' then yearly=wagerate*2000;
      if first.jobtype then payroll=0;
      payroll+yearly;
      if last.jobtype;
run;
proc print data=company.budget2 noobs;
      by manager;
      var jobtype;
      sum payroll;
      where manager in ('Coxe', 'Delgado');
      format payroll dollar12.2;
run;
```

Manager=Coxe

JobType	Payroll
3	$40,710.00
50	$123,390.00
240	$81,570.00
Manager	**$245,670.00**

Manager=Delgado

JobType	Payroll
240	$98,640.00
420	$64,700.00
440	$21,759.00
Manager	**$185,099.00**

Specifying Multiple BY Variables

Selected PDV Variables							
_N_	Manager	JobType	Payroll	FIRST.Manager	LAST.Manager	FIRST.JobType	LAST.Jobtype
1	Coxe	3	40710.0	1	0	1	1
2	Coxe	50	41040.0	0	0	1	0
3	Coxe	50	82350.0	0	0	0	1
4	Coxe	240	27300.0	0	0	1	0
5	Coxe	240	54270.0	0	1	0	1
6	Delgado	240	35520.0	1	0	1	0
7	Delgado	240	63120.0	0	0	0	1
8	Delgado	420	18870.0	0	0	1	0
9	Delgado	420	45830.0	0	0	0	1
10	Delgado	440	21759.6	0	1	1	1
11	Overby	1	82270.8	1	0	1	1
12	Overby	5	48550.2	0	0	1	1
13	Overby	10	53766.0	0	0	1	1
14	Overby	20	70929.0	0	0	1	0
15	Overby	20	108885.0	0	1	0	1

Reading Observations Using Direct Access

- So far we have read the observations in an input data set *sequentially*.
- You can also access observations directly, by going straight to an observation in a SAS data set without having to process each observation that precedes it by using the POINT= option in the SET statement along with the STOP and OUTPUT statements.

```
/*To read only the fifth observation from the data set*/
data work.getobs5;
      obsnum=5; /*a temporary numeric variable to hold the observation #*/
      set company.usa(keep=manager payroll) point=obsnum;
      output;
      stop;
run;
```

Note:

- output: to write the observation explicitly to a data set
- stop: stop processing the current DATA step immediately

More About the OUTPUT Statement

- Using an output statement without following a data set name causes the current observation to be written to **all** data sets that are named in the DATA statement.
- If a DATA statement contains two data set names, and you include an OUTPUT statement that references only one of the data sets. The DATA step will create both data sets, but only the data set that is named in the OUTPUT statement will contain output.

  For example, the program below creates two temporary data sets, Empty and Full. The data set Empty is created but contains no observations, and the data set Full contains all of the observations from Company.usa.

```
data empty full;
    set company.usa;
    output full;
run;
```

More About the OUTPUT Statement: Example

```
DATA qul errors;
   input subj qul3a qul3b qul3c qul3d qul3e
              qul3f qul3g qul3h qul3i qul3j;
   flag = 0;
   if qul3a not in (1, 2, 3) then flag = 1;
   if qul3b not in (1, 2, 3) then flag = 1;
   if qul3c not in (1, 2, 3) then flag = 1;
   if qul3d not in (1, 2, 3) then flag = 1;
   if qul3e not in (1, 2, 3) then flag = 1;
   if qul3f not in (1, 2, 3) then flag = 1;
   if qul3g not in (1, 2, 3) then flag = 1;
   if qul3h not in (1, 2, 3) then flag = 1;
   if qul3i not in (1, 2, 3) then flag = 1;
   if qul3j not in (1, 2, 3) then flag = 1;
   if flag = 1 then output errors;
               else output qul;
   drop flag;
   DATALINES;
   110011 1 2 3 3 3 3 2 1 1 3
   210012 2 3 4 1 2 2 3 3 1 1
   211011 1 2 3 2 1 2 3 2 1 3
   310017 1 2 3 3 3 3 3 2 2 1
   411020 4 3 3 3 3 2 2 2 2 2
   510001 1 1 1 1 1 1 2 1 2 2
   ;
RUN;
PROC PRINT data = qul;
   TITLE 'Observations in Qul data set with no errors';
RUN;
PROC PRINT data = errors;
   TITLE 'Observations in Errors data set with errors';
RUN;
```

Observations in Qul data set with no errors

Obs	subj	qul3a	qul3b	qul3c	qul3d	qul3e	qul3f	qul3g	qul3h	qul3i	qul3j
1	110011	1	2	3	3	3	3	2	1	1	3
2	211011	1	2	3	2	1	2	3	2	1	3
3	310017	1	2	3	3	3	3	3	2	2	1
4	510001	1	1	1	1	1	1	2	1	2	2

Observations in Errors data set with errors

Obs	subj	qul3a	qul3b	qul3c	qul3d	qul3e	qul3f	qul3g	qul3h	qul3i	qul3j
1	210012	2	3	4	1	2	2	3	3	1	1
2	411020	4	3	3	3	3	2	2	2	2	2

Understanding How Data Sets Are Read

DATA step processing for reading existing SAS data sets is very similar to that of reading raw data. One of the differences is that while reading an existing data set with the SET statement, in PDV, SAS retains the values of existing variables from one observation to the next.

```
data finance.duejan;
    set finance.loans;
    Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans				
Account	Amount	Rate	Months	Payment
101-1092	22000	0.100	60	467.43
101-1731	114000	0.095	360	958.57
101-1289	10000	0.105	36	325.02
101-3144	3500	0.105	12	308.52

Compilation Phase

1. The program data vector is created and contains the automatic variables _N_ and _ERROR_.

Program Data Vector

_N_	_ERROR_	

2. SAS also scans each statement in the DATA step, looking for syntax errors.
3. When the SET statement is compiled, a slot is added to the program data vector (PDV) for each variable in the input data set. The input data set supplies the variable names, as well as attributes such as type and length.

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment

4. Any variables that are created in the DATA step are also added to the PDV. The attributes of these variables are determined by the expression in the statement.

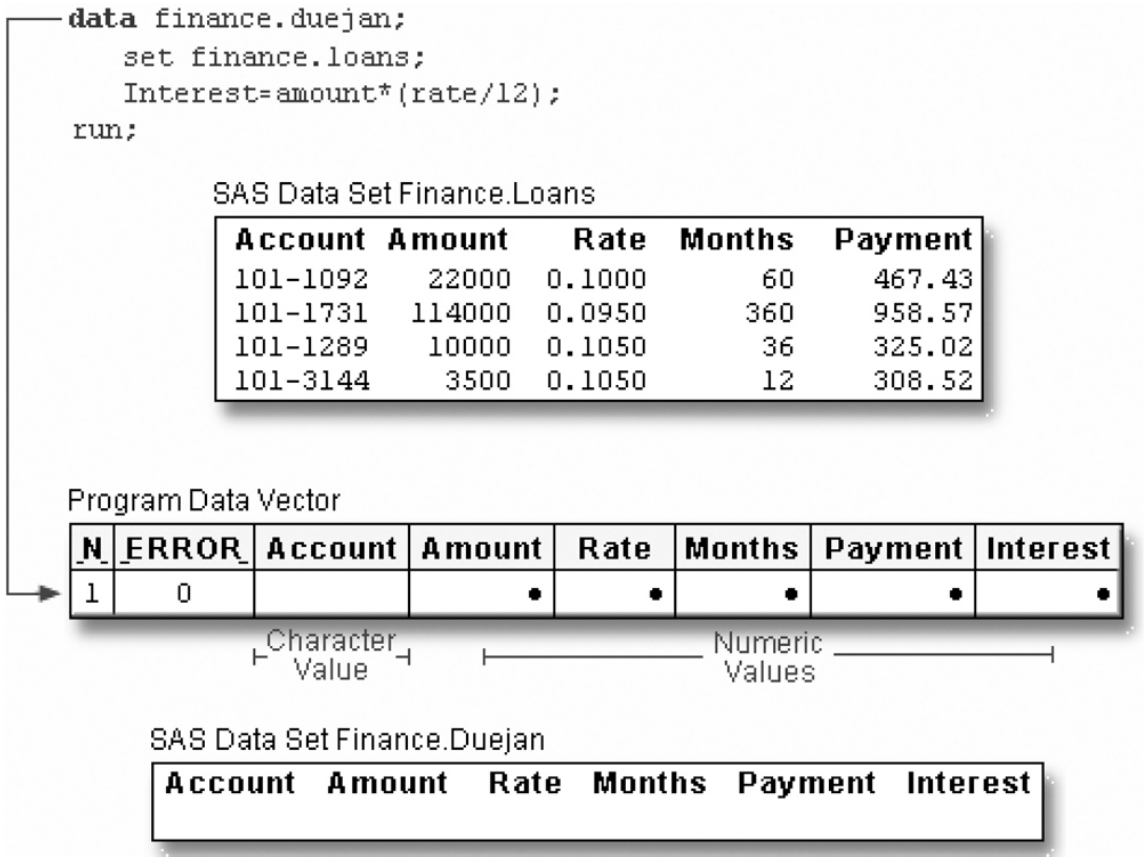
Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest

5. At the bottom of the DATA step, the compilation phase is complete, and the descriptor portion of the new SAS data set is created. There are no observations because the DATA step has not yet executed.

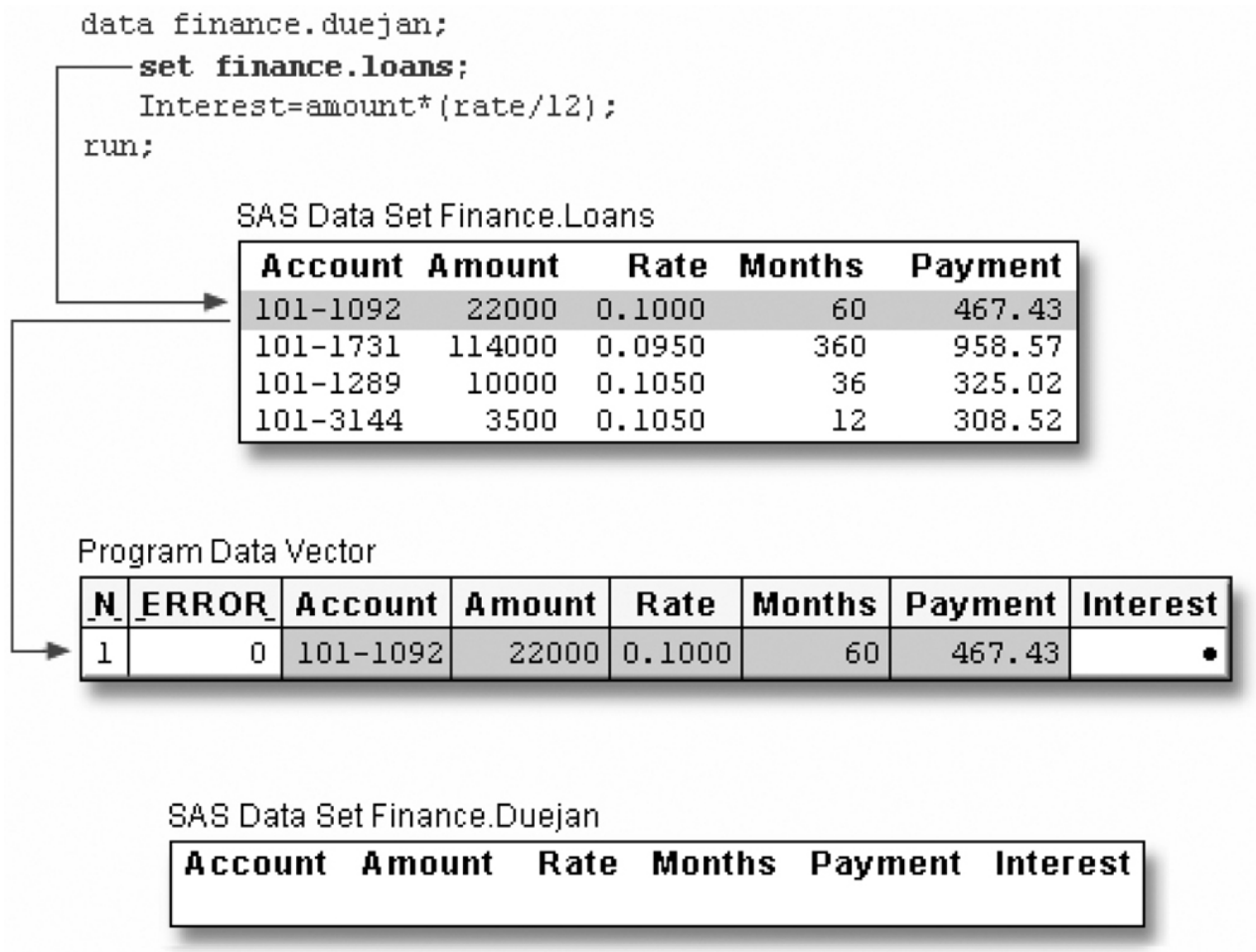
Execution Phase

1. The DATA step executes once for each observation in the input data set. For example, this DATA step will execute four times because there are four observations in the input data set Finance.Loans.
2. At the beginning of the execution phase, the value of _N_ is 1. Because there are no data errors, the value of _ERROR_ is 0. The remaining variables are initialized to missing.



Execution Phase

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



Execution Phase

4. Then, the assignment statement executes to compute the value for Interest.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
1	0	101-1092	22000	0.1000	60	467.43	183.333

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest

Execution Phase

5. At the end of the first iteration of the DATA step, the values in the PDV are written to the new data set as the first observation.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
1	0	101-1092	22000	0.1000	60	467.43	183.333

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

Execution Phase

6. The value of _N_ increments from 1 to 2, and control returns to the top of the DATA step.

```
data finance.duejan;
    set finance.loans;
    Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1092	22000	0.1000	60	467.43	183.333

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

Execution Phase

7. SAS **retains** the values of variables that were read from a SAS data set with the SET statement, or that were created by a sum statement. All other variable values, such as the values of the variable Interest, are set to missing.

```
data finance.duejan;
    set finance.loans;
    Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1092	22000	0.1000	60	467.43	.

Values Retained — Reset to Missing

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

Execution Phase

Note: When SAS reads **raw data**, the situation is different. In that case, SAS sets the value of each variable in the DATA step to missing at the beginning of each iteration, with these exceptions:

- variables named in a RETAIN statement
- variables created in a sum statement
- elements in a _TEMPORARY_ array
- any variables created by using options in the FILE or INFILE statements
- automatic variables.

Execution Phase

8. At the beginning of the second iteration, the value of _ERROR_ is reset to 0.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

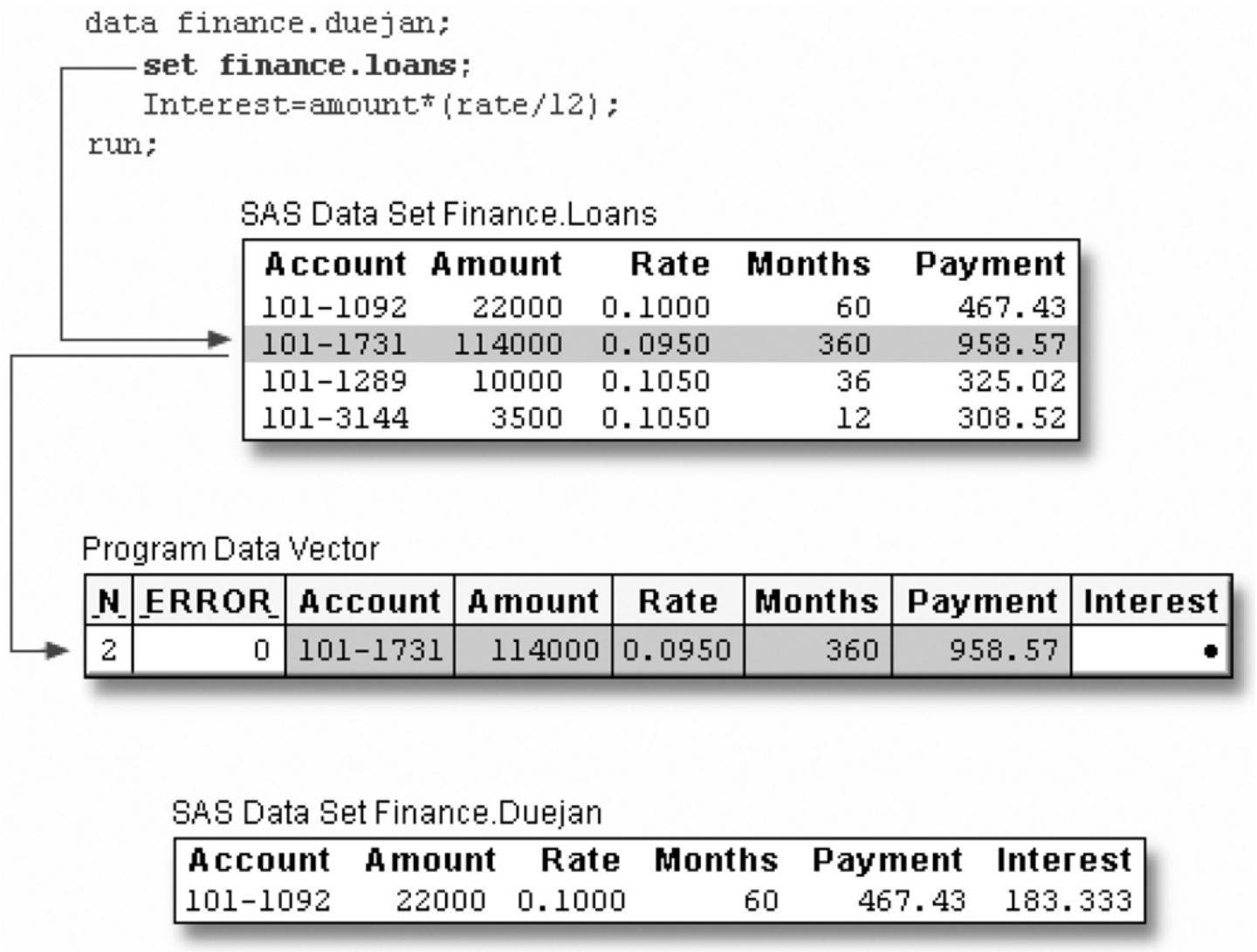
_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1092	22000	0.1000	60	467.43	.

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

Execution Phase

9. As the SET statement executes, the values from the second observation are read into the PDV.



Execution Phase

10. The assignment statement executes again to compute the value for Interest for the second observation.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1731	114000	0.0950	360	958.57	902.5

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

Execution Phase

11. At the bottom of the DATA step, the values in the PDV are written to the data set as the second observation.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1731	114000	0.0950	360	958.57	902.5

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333
101-1731	114000	0.0950	360	958.57	902.5

Execution Phase

12. The value of _N_ increments from 2 to 3, and control returns to the top of the DATA step. SAS retains the values of variables that were read from a SAS data set with the SET statement and the value of the variable Interest is set to missing.

```
data finance.duejan;
    set finance.loans;
    Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
3	0	101-1731	114000	0.0950	360	958.57	.

Values Retained — Reset to Missing

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333
101-1731	114000	0.0950	360	958.57	902.5

This process continues until all of the observations are read.