SAS® Programming III: Advanced Techniques

Course Notes

SAS® Programming III: Advanced Techniques Course Notes was developed by Linda Jolley and Jane Stroupe. Additional contributions were made by Bill Brideson, George Berg, Ted Meleky, Rich Papel, Dr. Sue Rakes, Kent Reeve, Christine Riddiough, and Roger Staum. Editing and production support was provided by the Curriculum Development and Support Department.

SAS® Programming III: Advanced Techniques Course Notes

Book code E70041, course code PROG3, prepared date 14Oct05.

Table of Contents

Course Description

This course builds on the concepts presented in the *SAS Programming II: Manipulating Data with the DATA Step* course. This course focuses on reading data with direct access; combining data; sorting; using multidimensional arrays, hash tables, and formats for table lookups; efficiently storing data; utilizing best practices; and creating tables with the SAS Scalable Performance Data Engine.

This course is a combination of the previously offered *SAS Programming III: Advanced Techniques* and *Optimizing SAS Programs* courses.

To learn more…

SAS Education

A full curriculum of general and statistical instructor-based training is available at any of the Institute's training facilities. Institute instructors can also provide on-site training.

For information on other courses in the curriculum, contact the SAS Education Division at 1-919-531-7321, or send e-mail to training@sas.com. You can also find this information on the Web at support.sas.com/training/ as well as in the Training Course Catalog.

SAS Publishing

For a list of other SAS books that relate to the topics covered in this Course Notes, USA customers can contact our SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the Publications Catalog on the Web at support.sas.com/pubs for a complete list of books and a convenient order form.

Prerequisites

This course is **not** appropriate for beginning SAS software users. Before attending this course, you should have at least nine months of SAS programming experience and should have completed the *SAS Programming II: Manipulating Data with the DATA Step* course. Specifically, you should be able to do the following:

- understand your operating system file structures and perform basic operating system tasks
- understand programming logic concepts
- understand the compilation and execution process of the DATA step
- use different kinds of input to create SAS data sets from external files
- use SAS software to access SAS data libraries
- create and use SAS date values
- read, concatenate, merge, match-merge, and interleave SAS data sets
- use the DROP=, KEEP=, and RENAME= data set options
- create multiple output data sets
- use array processing and DO loops to process data iteratively
- use SAS functions to perform data manipulation and transformations.

Chapter 1 Introduction

1.1 Introduction of Course Topics

General Business Scenario

International Airlines has several data files that must be manipulated before they can be used for report production.

The to-do list includes the items on the following slides:

continued...

3

General Business Scenario

- appending
  - raw data files
  - SAS data sets

Date	Expenses	Origin	Destination
02DEC1999	58907	RDU	LHR
03DEC1999	108543	RDU	LHR
04DEC1999	21963	RDU	LHR
05DEC1999	31517	RDU	LHR
06DEC1999	105682	RDU	LHR
07DEC1999	66992	RDU	LHR
08DEC1999	92873	RDU	LHR
09DEC1999	59560	RDU	LHR
10DEC1999	41096	RDU	LHR
11DEC1999	10272	RDU	LHR
12DEC1999	35121	RDU	LHR
13DEC1999	65836	RDU	LHR
14DEC1999	73350	RDU	LHR
15DEC1999	58539	RDU	LHR
16DEC1999	64191	RDU	LHR
17DEC1999	116839	RDU	LHR
18DEC1999	82369	RDU	LHR
19DEC1999	109908	RDU	LHR
20DEC1999	2439	RDU	LHR
21DEC1999	36700	RDU	LHR
12DEC1999	35121	RDU	LHR
13DEC1999	65836	RDU	LHR
14DEC1999	73350	RDU	LHR
15DEC1999	58539	RDU	LHR
16DEC1999	64191	RDU	LHR
17DEC1999	116839	RDU	LHR
18DEC1999	82369	RDU	LHR
19DEC1999	109908	RDU	LHR
20DEC1999	2439	RDU	LHR
21DEC1999	36700	RDU	LHR
14DEC1999	73350	RDU	LHR
15DEC1999	58539	RDU	LHR
16DEC1999	64191	RDU	LHR
17DEC1999	116839	RDU	LHR
18DEC1999	82369	RDU	LHR
19DEC1999	109908	RDU	LHR
20DEC1999	2439	RDU	LHR
21DEC1999	36700	RDU	LHR

- combining
  - three SAS data sets without common BY variables
  - a summary data set with a detail data set
  - a small data set with a large data set

Expenses	Origin	Destination	Date	Class	Business	Economy	Profit	AirportCity	AirportName
58907	RDU	LHR	02DEC1999	19200	31610	79650	71553	London, England	Heathrow Airport
108543	RDU	LHR	03DEC1999	17600	25070	80181	14308	London, England	Heathrow Airport
21963	RDU	LHR	04DEC1999	17600	28340	84960	108937	London, England	Heathrow Airport
31517	RDU	LHR	05DEC1999	17600	32700	72216	90999	London, England	Heathrow Airport
105682	RDU	LHR	06DEC1999	22400	29430	74871	21019	London, England	Heathrow Airport
66992	RDU	LHR	07DEC1999	22400	29430	84960	69798	London, England	Heathrow Airport
92873	RDU	LHR	08DEC1999	20800	27250	82305	37482	London, England	Heathrow Airport
59560	RDU	LHR	09DEC1999	22400	32700	84429	79969	London, England	Heathrow Airport
41096	RDU	LHR	10DEC1999	20800	32700	67968	80372	London, England	Heathrow Airport
10272	RDU	LHR	11DEC1999	22400	29430	78588	120146	London, England	Heathrow Airport
35121	RDU	LHR	12DEC1999	17600	30520	67968	80967	London, England	Heathrow Airport
65836	RDU	LHR	13DEC1999	22400	31610	84960	73134	London, England	Heathrow Airport
73350	RDU	LHR	14DEC1999	22400	32700	74340	56090	London, England	Heathrow Airport
58539	RDU	LHR	15DEC1999	20800	29430	72747	64438	London, England	Heathrow Airport
64191	RDU	LHR	16DEC1999	20800	28340	82836	67785	London, England	Heathrow Airport
116839	RDU	LHR	17DEC1999	20800	25070	83898	12929	London, England	Heathrow Airport
82369	RDU	LHR	18DEC1999	20800	32700	72747	43878	London, England	Heathrow Airport
109908	RDU	LHR	19DEC1999	20800	27250	70092	8234	London, England	Heathrow Airport
2439	RDU	LHR	20DEC1999	17600	30520	65844	111525	London, England	Heathrow Airport
36700	RDU	LHR	21DEC1999	22400	32700	75933	94333	London, England	Heathrow Airport

continued...

4

General Business Scenario

- creating random samples to use for various analyses
- creating indexes for quick retrieval of subsets
- updating a master table with a transaction table
- performing table lookups
- sorting data sets
- accessing current data in frequently changing files

continued...

5

General Business Scenario

Perform these tasks as efficiently as possible, and optimize the following:

- I/O
- CPU
- memory
- data storage space

6

1.2 Measuring Efficiencies

Objectives

- Identify the resources used by a SAS program.
- Use SAS system options to measure computer resources.
- Interpret resource usage statistics in your operating environment.
- Benchmark resource usage.

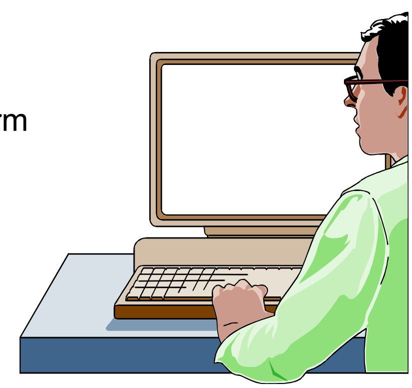
8

Running a SAS Program

What resources are required to run a SAS program?

The programmer must perform the following tasks:

- write the program
- execute the program
- maintain the program



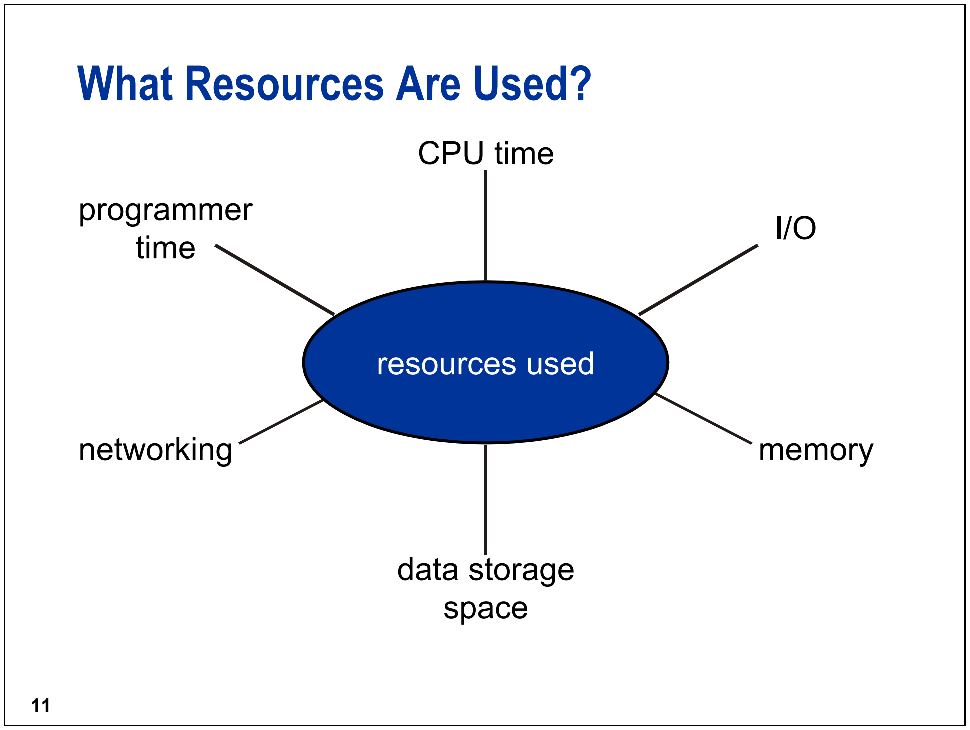
9

Running a SAS Program

The computer must perform the following actions:

- load the required SAS software components and the program into memory
- compile the program
- locate data required by the program
- execute the program
- store output data files
- store printed reports

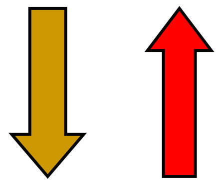
10



CPU	measures the amount of time that the Central Processing Unit uses to perform requested tasks such as calculations, reading and writing data, conditional and iterative logic, and so on.
I/O	provides a measurement of the read-and-write operations performed as data and programs are moved from a storage device to memory (input) or from memory to a storage or display device (output).
Memory	is the size of the work area required to hold executable program modules, data, and buffers.
Data storage space	is the amount of space on a disk or tape required to store data.
Programmer time	is the amount of time required for the programmer to write and maintain the program. This can be decreased through well documented, logical programming practices.
Networking	is the amount of time required to transfer data across your computer network. This can be decreased by performing as much of the subsetting and summarizing as possible on the remote computer before transferring the data to the local computer. The networking time is dependent on the bandwidth of your I/O controller.

Understanding Efficiency Trade-offs

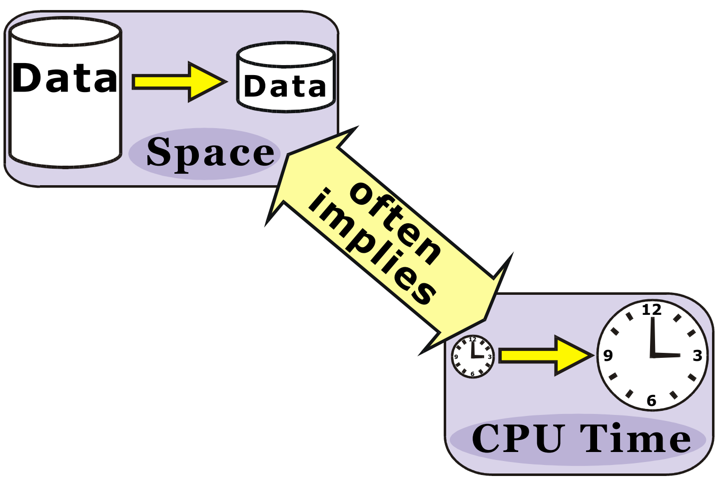
When you decrease the use of one resource, the use of another resource frequently increases.



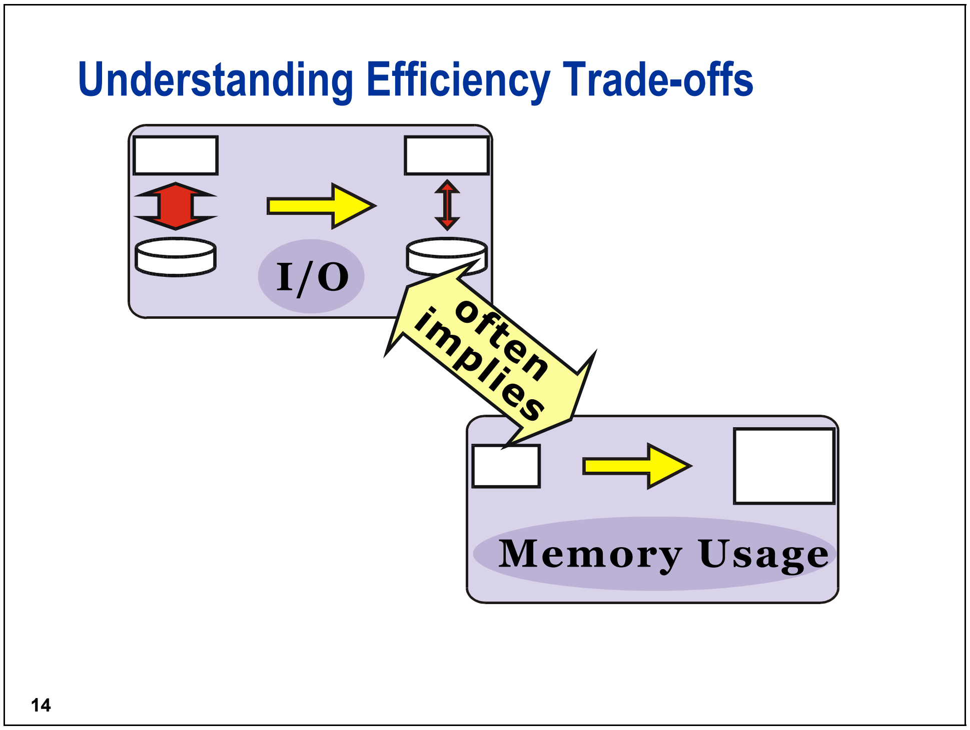
12

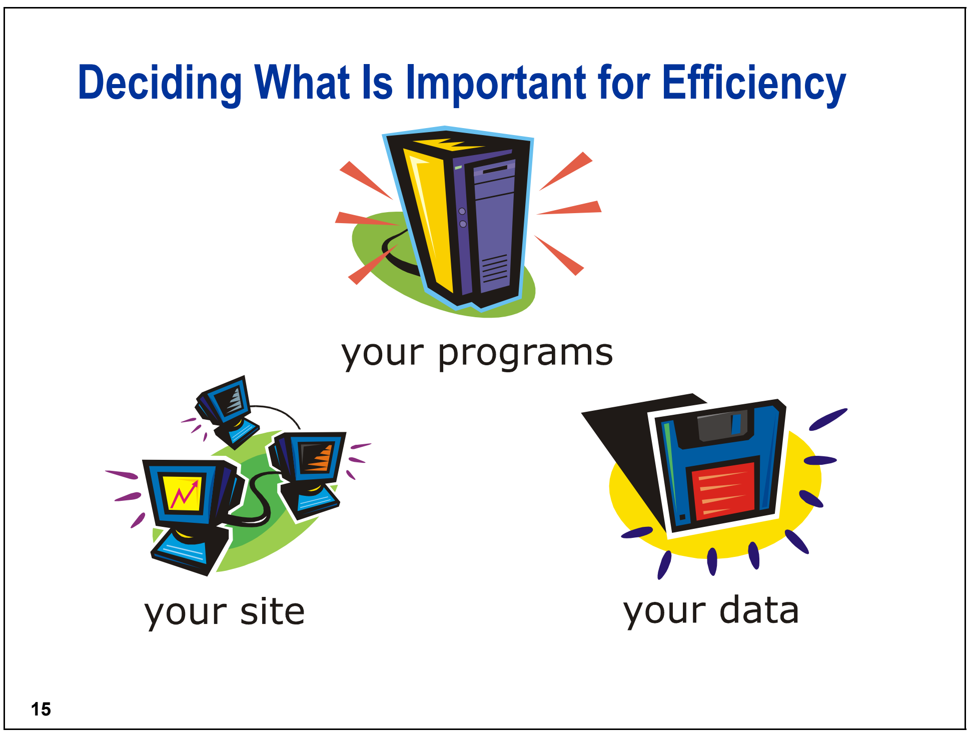
...

Understanding Efficiency Trade-offs



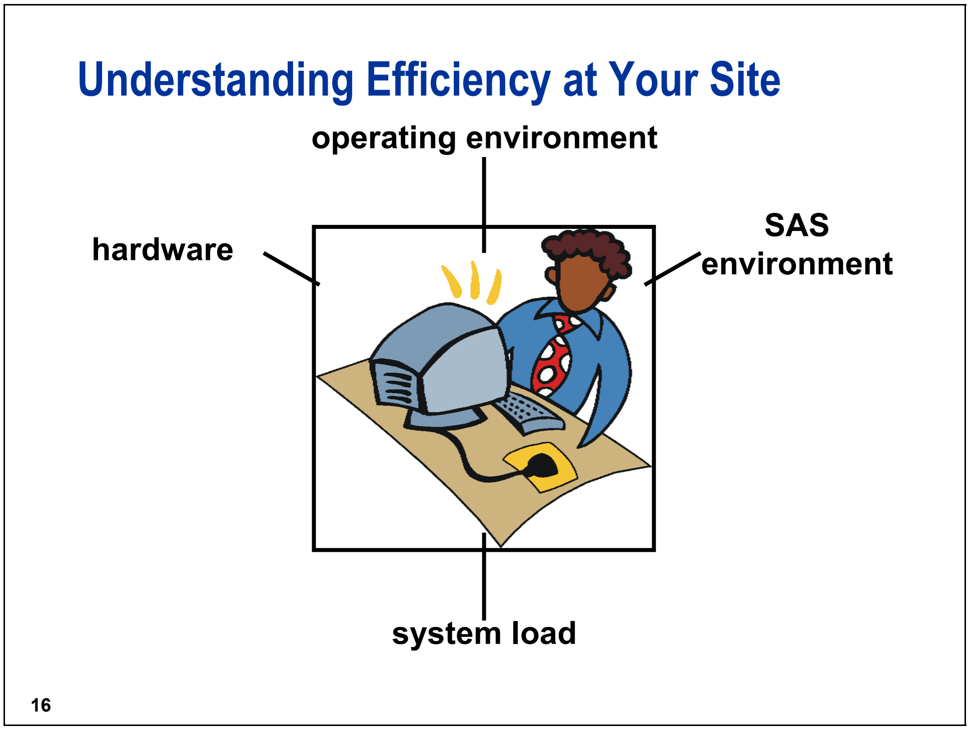
13





You must decide which factors are the most important for improving resource usage at your site. To make this decision, you must know the following:

- which resources are scarce or costly at your site
- how and when your programs will be used
- the type and volume of data your programs will process



Environmental factors that affect the efficiency of SAS programs include the following:

Hardware	the amount of available memory, the number of peripheral devices attached to the CPU, and the communications hardware in use
Operating environment	resource allocation, scheduling algorithms, and I/O methods
System load	the number of users or jobs sharing system resources including network bandwidth along with the traffic.
SAS environment	determined by which SAS software products are installed, how they were installed, and which methods are available to run SAS programs at your site

In most cases, one or two resources are the most limited or most expensive for your programs. You can usually decrease the amount of critical resources that are used if you are willing to sacrifice some efficiency of the resources that are less critical at your site.

Knowing How Your Program Will Be Used

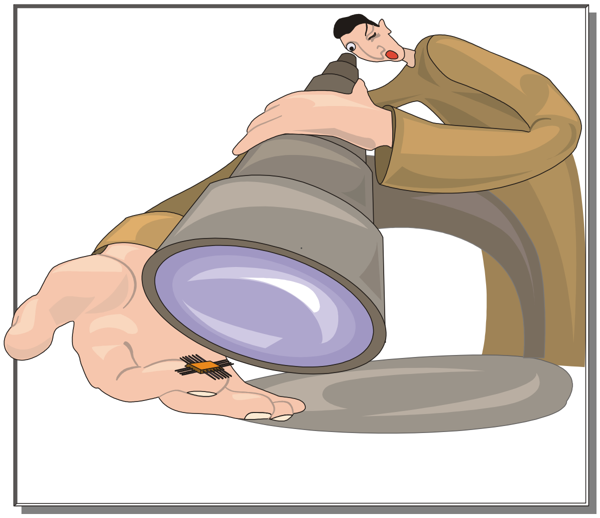
The importance of efficiency increases with the following:

- the size of the program or the files being processed
- the number of times the program will be executed

17

- Developing an efficient program requires time and thought. The first question to address is whether the additional amount of resources saved is worth the time and effort spent to achieve the savings.
- Consider the size of the program or the files that are processed. As the programs or files increase in size, the potential for savings increases. Therefore, devote your effort to improve the efficiency of large programs.
- Also consider the number of times the program will run. The difference in the resources used by an inefficient program and an efficient program that run one time or a few times is relatively small, whereas the cumulative difference for a program that is run frequently is large.

Knowing Your Data



18

The effectiveness of any efficiency technique depends greatly on the data with which you use it. When you know the characteristics of your data, you can select the techniques that take advantage of those characteristics.

Considering Trade-Offs

In this class, each task will be performed using one or more techniques.

You should benchmark with your own data to determine which technique is the most efficient.

19

Deciding Which Technique Is Most Efficient

To decide which technique is most efficient for a given task, *benchmark*, or measure and compare, the resource usage of each technique.

20

Running Benchmarks: Guidelines

To benchmark your programming techniques, do the following:

- Turn on the appropriate options to report resource usage.
- Test each technique in a separate SAS session.
- Test only one technique or change at a time, with as little additional code present as possible.

continued...

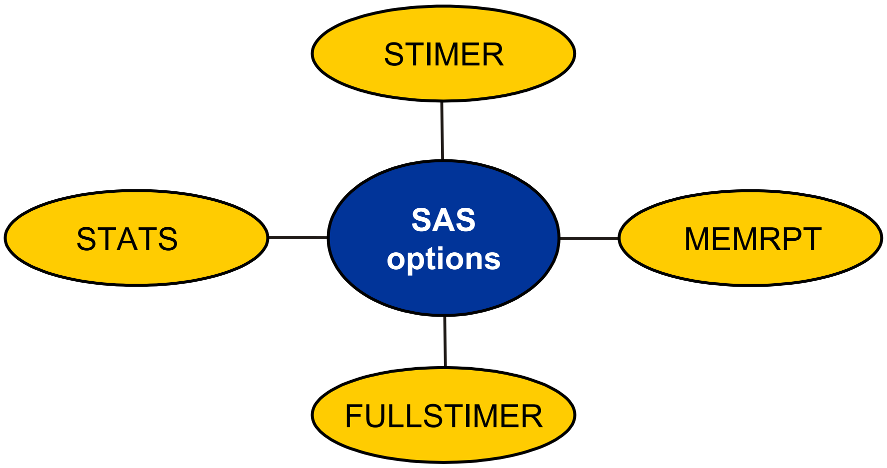
21

Running Benchmarks: Guidelines

- Run your tests and use the conditions that your final program will use (for example, batch execution, large data sets, and so on).
- Turn off the options that report resource usage after testing is finished, because they consume resources.
- Run each program several times and base your conclusions on averages, not on an individual execution, if you are benchmarking elapsed time.
- Average resource usage data only if the results are in the same *ballpark*. Do not average very diverse resource usages because that data might lead you to tune your program to run less efficiently.

22 ...

Tracking Resource Usage



23

There are four SAS system options that you can use to track and report on resource utilization:

STIMER	tracks the CPU time used to perform a task (DATA or PROC step). CPU time can be divided into System CPU time and User CPU time.
MEMRPT	tracks memory used while performing a task.
FULLSTIMER	tracks usage of additional resources. This option is ignored unless STIMER or MEMRPT is in effect. It can also be specified by the alias FULLSTATS.
STATS	writes information tracked by the above options to the SAS log.

✎ The availability and usage of these options are specific to the operating environment.

Syntax (default listed first):

```
OPTIONS NOFULLSTIMER | FULLSTIMER;
```

```
OPTIONS STIMER | NOSTIMER;
```

```
OPTIONS STATS | NOSTATS;
```

```
OPTIONS MEMRPT | NOMEMRPT;
```

Tracking Resources with SAS Options

	z/OS	Windows	UNIX
STIMER	**I**	**B D**	**B D**
MEMRPT	**B D**	**N/A**	**N/A**
FULLSTIMER	**B**	**B**	**B**
STATS	**B D**	**N/A**	**N/A**

24

I Invocation option only

B Can be set at invocation or by using an OPTIONS statement

N/A Not available (The functionality is part of the STIMER option under UNIX and Windows.)

D Default

Use the OPTIONS procedure with the HOST option to determine the default settings of these options at your site.

```
proc options host;
run;
```

You can find more information on operating environment dependencies in the SAS documentation for your operating environment.

Tracking SAS/ACCESS Resources (Self-Study)

In addition to the traditional four SAS system options for tracking resource usage, the SASTRACE= system option is a powerful tool to use when you want to see the commands that are sent to your database management system (DBMS) by the SAS/ACCESS engine.

SASTRACE= output is DBMS-specific.

General form of the SASTRACE= system option:

```
OPTIONS SASTRACE = ',,,d ' | ',,t, ' | ',,t,s ';
```

Notice the use of the commas as placeholders.

25

Selected values for SASTRACE= are shown below:

',,,d' specifies that all SQL statements sent to the DBMS are sent to the log.

',,t,' specifies that all threading information is sent to the log.

',,t,s' specifies that all threading information and a summary of timing information for calls made to the DBMS are sent to the log.

The following details can help you manage SASTRACE= output in your DBMS:

- When using SASTRACE= on PC platforms, you must also specify the following option:

```
sastraceloc = stdout | saslog
```

- In order to turn SAS tracing off, you can specify the following option:

```
options sastrace=off;
```

- Log output is much easier to read if you specify **`nostsuffix`**.

Tracking SAS/ACCESS Resources (Self-Study)

```
7    options ls = 64 sastrace = ',,,d' sastraceloc = saslog
             nostsuffix;

9    proc print data = oralib.flightdelays;
10      where destination = 'CPH';
11      title 'Flights to Copenhagen';
12   run;

ORACLE_2: Prepared:
SELECT  "DESTINATION", "FLIGHTNUMBER", "FLIGHTDATE", "ORIGIN",
"DELAYCATEGORY", "DESTINATIONTYPE", "DAYOFWEEK", "DELAY" FROM
educ.FLIGHTDELAYS  WHERE  ("DESTINATION" = 'CPH' )

ORACLE_3: Executed:
SELECT statement  ORACLE_2

NOTE: There were 27 observations read from the data set
      ORALIB.FLIGHTDELAYS.
      WHERE destination='CPH';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.58 seconds
      cpu time            0.07 seconds
```

26

c01s2d1

The following code was used to generate this output:

```
/*  Using a WHERE statement to subset an Oracle table.  */

libname oralib oracle user = edu001 pw = xxxxxx
                      path = dbmssrv schema = educ;

/*  Use SASTRACE= and SASTRACELOC= to write the */
/*  generated Oracle SQL statements to the log. */
options ls = 64 sastrace = ',,,d' sastraceloc = saslog
        nostsuffix;

/*  Subset for Copenhagen destination */
proc print data = oralib.flightdelays;
   where destination = 'CPH';
   title 'Flights to Copenhagen';
run;
```

1.3 SAS Processing

Objectives

- Investigate the concept of a data set page and how it relates to the structure of SAS data sets.
- Review how SAS reads and writes data.

28

SAS Data Set Pages

A SAS *data set page* has the following attributes:

- is the unit of data transfer between the operating system buffers and SAS buffers in memory
- includes the number of bytes used by the descriptor portion, the data values, and the overhead
- is fixed in size when the data set is created, either to a default value or to a value specified by the programmer

29

Using PROC CONTENTS to Report Page Size

```
proc contents data = ia.sales;
run;
```

Partial Output

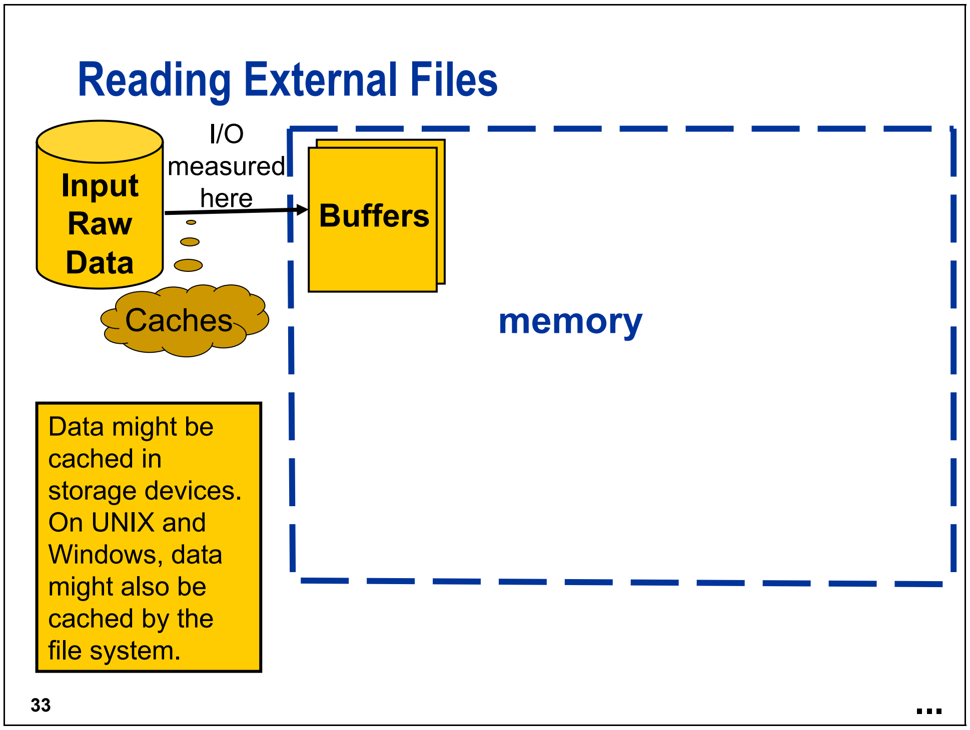
```
                Engine/Host Dependent Information

                  Data Set Page Size            16384
                  Number of Data Set Pages      3396
                  First Data Page               1
                  Max Obs per Page              97
                  Obs in First Data Page        76
                  Index File Page Size          4096
                  Number of Index File Pages    2552
                  Number of Data Set Repairs    0
                  File Name                     sales.sas7bdat
                  Release Created               9.0101M3
                  Host Created                  XP_PRO
```

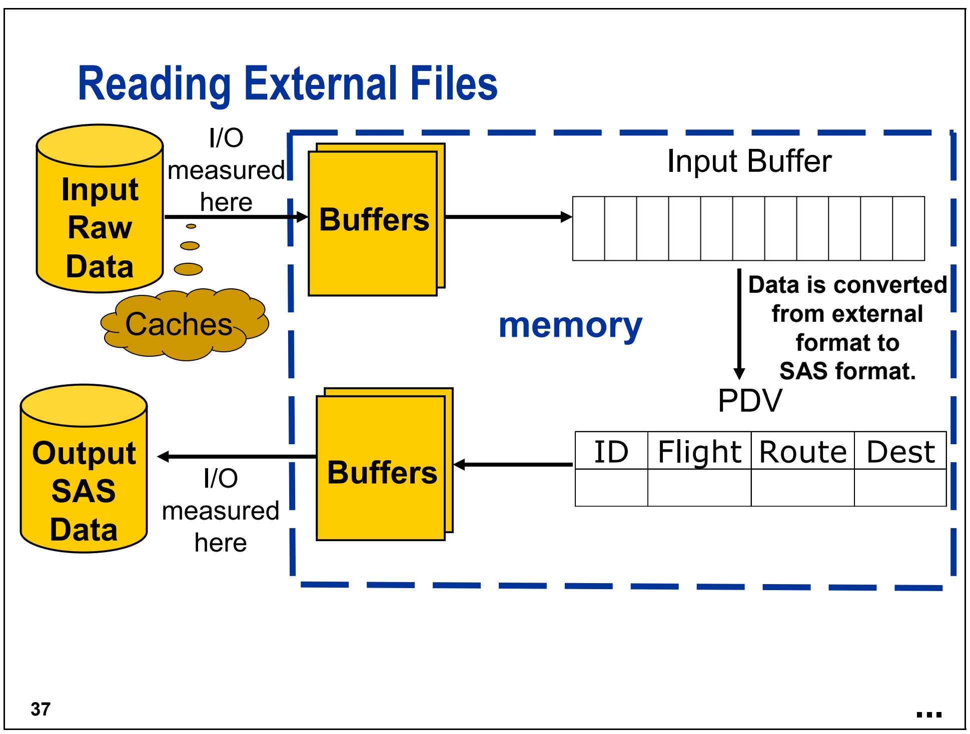
30 c01s3d1

The total number of bytes occupied by **ia.sales** can be calculated as shown below:

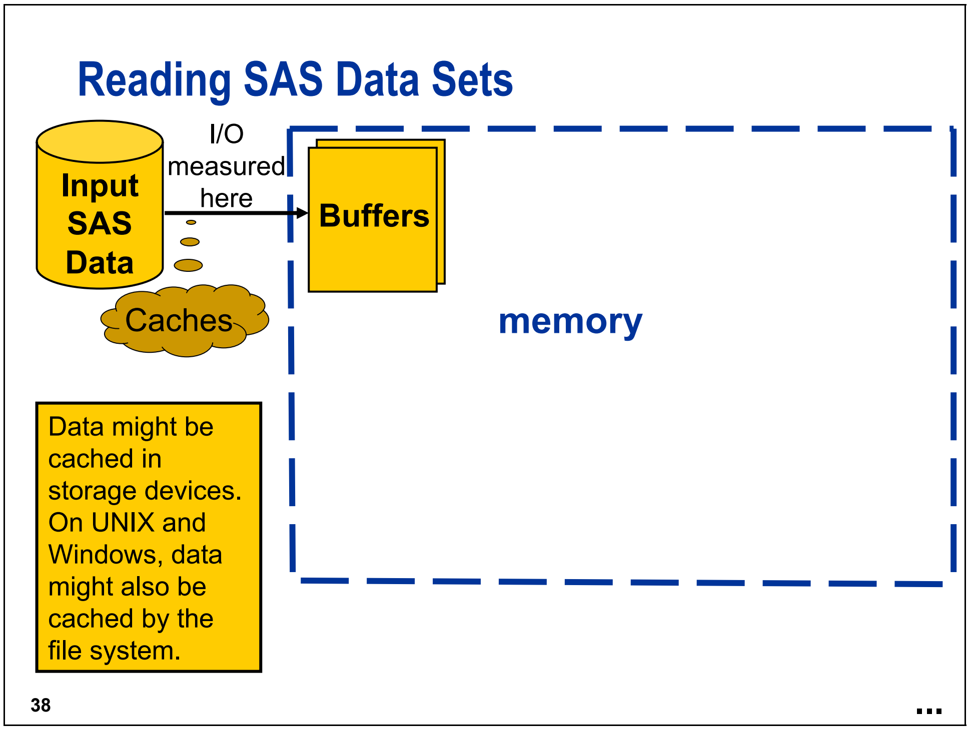
(16,384 * 3,396) + (4,096 * 2,552) = 66,093,056 bytes

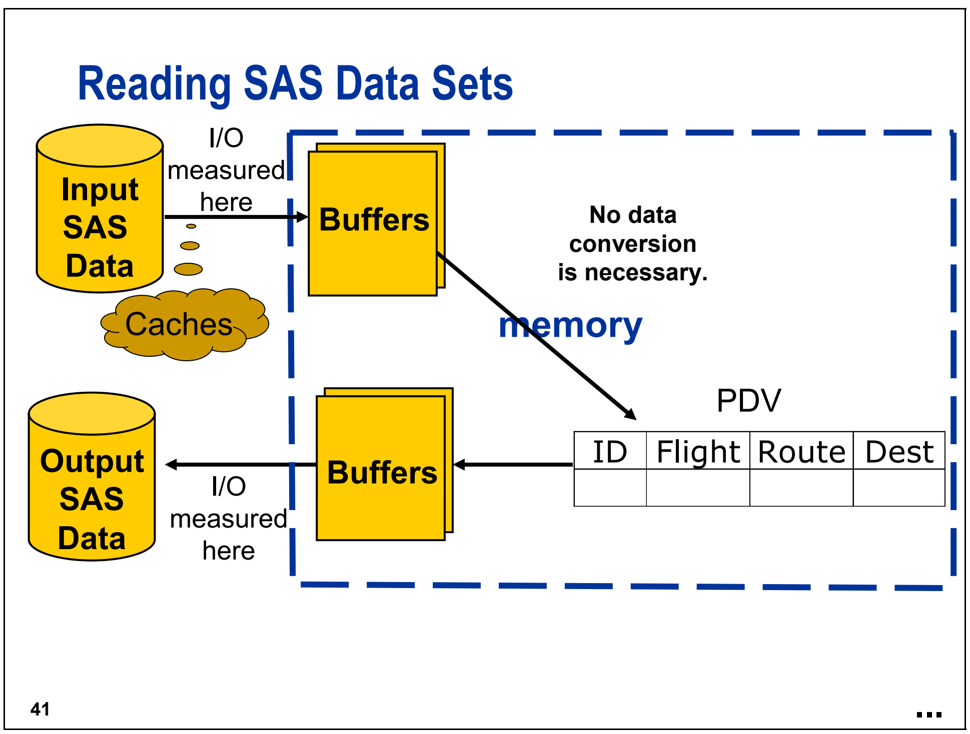
The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.





- The Input Buffer contains one record of raw data.
- The PDV contains one observation of SAS data.


Reading SAS Data Sets
Input SAS Data
I/O measured here
Buffers
Caches
memory
Data might be cached in storage devices. On UNIX and Windows, data might also be cached by the file system.
38
...


Reading SAS Data Sets
Input SAS Data
I/O measured here
Buffers
No data conversion is necessary.
Caches
memory
PDV
ID
Flight
Route
Dest
Output SAS Data
Buffers
I/O measured here
41
...

1.4 Controlling Memory and I/O Resources

Objectives

- Change the page size of a SAS data set.
- Use system and data set options to control memory usage.
- Use the SASFILE statement when you read small SAS data sets.
- Use the Scatter/Gather I/O feature in the Windows operating environment.

43

Controlling Page Size and Memory Usage

- You can use the BUFSIZE= system option or data set option to control the page size of an output SAS data set.
- You can use the BUFNO= system option or data set option to control the number of SAS buffers open simultaneously in memory.

```
BUFSIZE= n | nK | nM | nG | nT | hexX | MIN | MAX
```

```
BUFNO= n
```

44

Increasing the BUFSIZE= option is useful for SAS data sets that are read sequentially (top to bottom). Using small BUFSIZE= and larger BUFNO= options is useful for SAS data sets that are read randomly. Random access to SAS data is discussed in Chapter 2.

Reference Information

```
BUFSIZE=n| nK | nM | nG | nT |hexX | MIN | MAX
```

n | nK | nM | nG | nT

specifies the page size in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of 8 specifies 8 bytes, and a value of 3m specifies 3,145,728 bytes.

The default is 0, which causes SAS to use the minimum optimal page size for the operating environment.

hexX

specifies the page size as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value 2dx sets the page size to 45 bytes.

MIN

sets the page size to the smallest possible number in your operating environment, down to the smallest four-byte, signed integer, which is -231-1, or approximately -2 billion bytes.

CAUTION: This setting might cause unexpected results and should be avoided.

Use BUFSIZE=0 in order to reset the buffer page size to the default value in your operating environment.

MAX

sets the page size to the maximum possible number in your operating environment, up to the largest four-byte, signed integer, which is 231-1, or approximately 2 billion bytes.

Windows:

n | nK | nM | nG

specifies the buffer page size in multiples of 1; 1,024 (kilobytes); 1,048,576 (megabytes), and 1,073,741,824 (gigabytes), respectively. You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 bytes, a value of .782k specifies 801 bytes, and a value of 3m specifies 3,145,728 bytes.

hexX

specifies the buffer page size as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value 2dx sets the buffer page size to 45 bytes.

MIN

sets the buffer page size to -2,147,483,648 and requires SAS to use a default value. Under Windows, the default value is 0. The minimum number is -2,147,483,648.

MAX

sets the buffer page size to 2,147,483,647 bytes.

UNIX:

n | nK | nM | nG

specifies the buffer page size in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 bytes, a value of .782k specifies 801 bytes, and a value of 3m specifies 3,145,728 bytes.

hexX

specifies the buffer page size as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by hex digits (0-9, A-F), and then followed by an X. For example, 2dx sets the buffer page size to 45 bytes.

MIN

sets the buffer page size to 0. When the buffer size is 0, the BASE engine calculates a buffer size to optimize CPU and I/O use. This size is the smallest multiple of 8K that can hold 80 observations but is not larger than 64K.

MAX

sets the buffer page size to 2,147,483,647.

Reference Information

z/OS:

```
BUFSIZE=0 | n | nK
```

0

specifies that SAS choose the optimal page size of the data set based on the characteristics of the library and the type of data set.

n | nK

specifies the permanent buffer size (page size) in bytes or kilobytes, respectively. For libraries other than HFS, the value specified will be rounded up to the block size (BLKSIZE) of the library data set, because a block is the smallest unit of a data set that may be transferred in a single I/O operation.

Windows and Unix:

```
BUFNO= MIN | MAX | n| nK | nM | nG | nT | hex
```

Windows:

n | nK | nM | nG

specifies the number of buffers in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 buffers, a value of .782k specifies 801 buffers, and a value of 3m specifies 3,145,728 buffers.

For values greater than 1G, use the nM option or specify MAX.

hexX

specifies the number of buffers as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value 2dx specifies 45 buffers.

MIN

sets the number of buffers to 0, and requires SAS to use the default value of 1.

MAX

sets the number of buffers to 2,147,483,647.

UNIX:

n | nK | nM | nG

specifies the number of buffers in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 buffers, a value of .782k specifies 801 buffers, and a value of 3m specifies 3,145,728 buffers.

hexX

specifies the number of buffers as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by hex digits (0-9, A-F), and then followed by an X. For example, 2dx specifies 45 buffers.

MIN

sets the number of buffers to 0, and requires SAS to use the default value of 1.

MAX

sets the number of buffers to 2,147,483,647.

✎ For more information, consult SAS OnlineDoc 9.1.3. Expand **Base SAS**, and select **SAS Language Reference: Dictionary** and **Operating Environment Specific Information**.

Controlling Page Size and Memory Usage

The product of BUFNO= and BUFSIZE= determines how much data can be transferred in a read operation.

BUFSIZE	BUFNO	Bytes transferred in one I/O
6144	2	12,288

Increasing either BUFSIZE= or BUFNO= increases the amount of data that can be transferred in a read operation.

45 ...

Controlling Page Size

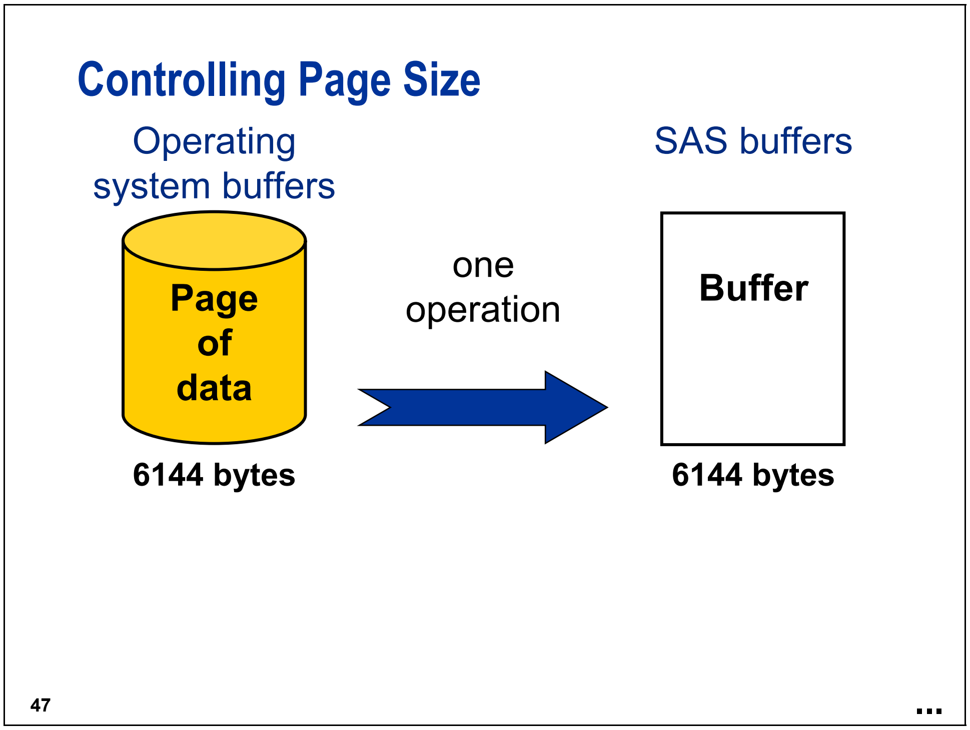
In order to select a default page size, SAS software uses an algorithm based on observation length, engine, and operating environment.

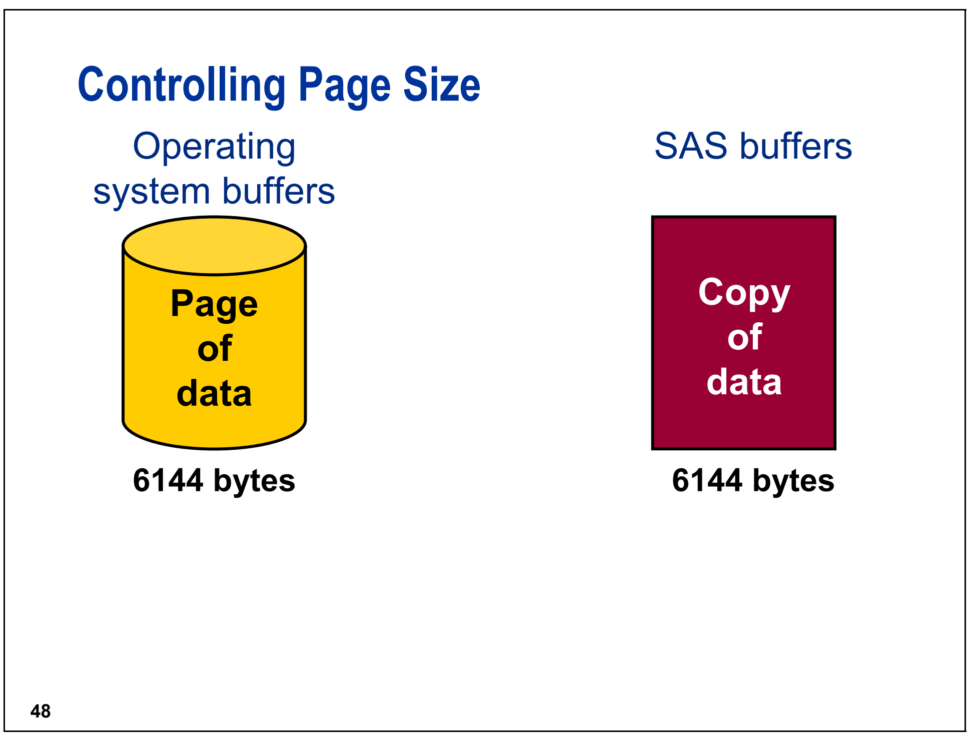
You can use the BUFSIZE= system or data set option to override the default page size.

BUFSIZE= specifies not only the page size (in bytes), but also the size of each buffer used to read or write the SAS data set.

```
data ia.times(bufsize = 30720);
   infile rtetimes;
   input @1 RouteID $7.
         @8 Origin $3.
         @11 Dest $3.
         @14 Distance 8.
         @24 Depart time5.
         @32 Arrival time5.;
run;
```

46 c01s4d1


Controlling Page Size
Operating system buffers
SAS buffers
Page of data
one operation
Buffer
6144 bytes
6144 bytes
47
...


Controlling Page Size
Operating system buffers
SAS buffers
Page of data
Copy of data
6144 bytes
6144 bytes
48

Controlling Page Size

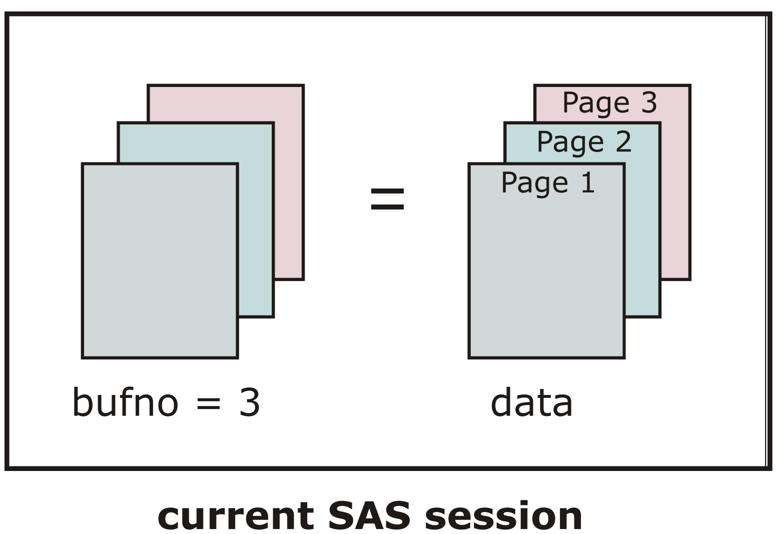
After it is specified, page size is a permanent attribute of the data set, and is used whenever the data set is processed.

Choosing a page size that is larger than the default can reduce execution time by reducing the number of times that SAS must read from or write to the operating system buffers.

The reduction in I/O comes at the cost of increased memory consumption.

49

Controlling Memory Usage



50

Controlling Memory Usage

The buffer number is not a permanent attribute of the data set and is valid only for the current step or SAS session. As more buffers are available, more pages can be transferred in a single move operation.

The reduction in number of moves comes at the cost of increased memory consumption.

```
data _null_;
   set ia.times(bufno = 2);
run;
```

51 c01s4d2

SASFILE Global Statement

- The SASFILE statement requests that a SAS data set be opened and loaded into SAS memory in its entirety instead of a few pages at a time.
- After it is read, data is held in memory for subsequent DATA and PROC steps to process.
- A second SASFILE statement closes the file and frees the SAS buffers.

52

The SASFILE statement can reduce execution time by taking advantage of large amounts of memory. The SASFILE statement became available in SAS Release 8.1.

SASFILE Global Statement

General form of the SASFILE statement:

SASFILE <*libref.*>*member-name*
<(*password-data-set-option*(*s*))>
OPEN | LOAD | CLOSE;

53

OPEN	opens the file and allocates the buffers, but defers reading the data into memory until a procedure or a statement that references the file is executed.
LOAD	opens the file, allocates the buffers, and reads the data into memory.
CLOSE	frees the buffers and closes the file.

Buffer Allocation

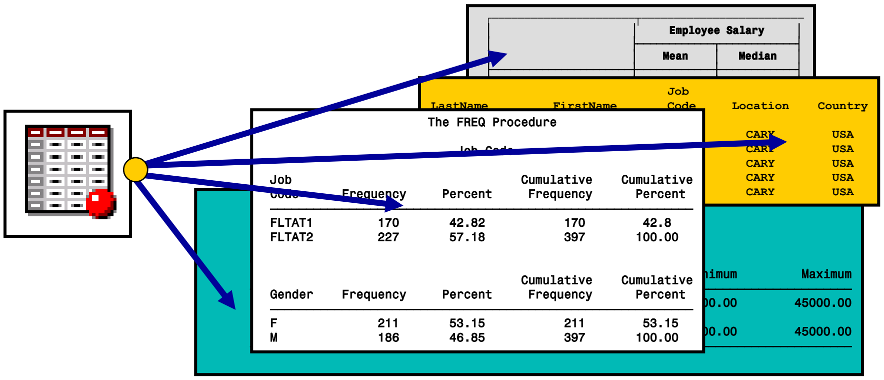
When the SASFILE statement executes, SAS allocates the number of buffers based on the number of pages of the SAS data set and index file.

If the file in memory increases in size during processing by editing or appending data, the number of buffers also increases.

54

Using the SASFILE Statement

Create reports using the PRINT, TABULATE, MEANS, and FREQUENCY procedures against a single SAS data set.



55

Using the SASFILE Statement

```
sasfile ia.fltaten load;
proc print data = ia.fltaten;
   var LastName FirstName JobCode
       Country Location;
   sum Salary;
run;
proc tabulate data = ia.fltaten;
   class Gender;
   var Salary;
   table Gender, Salary*(mean median);
run;
proc means data = ia.fltaten;
   var Salary;
   class Gender;
   output out = summary sum =;
run;
proc freq data = ia.fltaten;
   tables JobCode Gender;
run;
sasfile ia.fltaten close;
```

`ia.fltaten` is read into memory only once instead of four times. This results in one-fourth as many I/O operations, increased memory usage, and probably reduced elapsed time.

56

c01s4d3

The SASFILE statement is good for small SAS data sets.

Using the SGIO System Option in Windows (Self-Study)

The SGIO system option performs the following functions:

- activates the Scatter-Read/Gather-Write I/O feature
- improves I/O performance for SAS I/O files when the PC has a large amount of RAM

General form of the SGIO system option:

```
NOSGIO | SGIO;
```

NOSGIO | SGIO is an invocation option.

57

The default value is NOSGIO.

- With SAS I/O files (data sets, catalogs, indexes, utility files, and so on), normal sequential reads and writes go through the Windows File Cache.
- The Windows File Cache provides a great benefit in most cases, but for large SAS I/O files, Scatter-Read or Gather-Write usually improves performance.

Scatter-Read/Gather-Write is available in Windows 2000 and Windows XP.

For Windows NT users, you must install Service Pack 4.

Using the SGIO System Option in Windows (Self-Study)

When SGIO is active, SAS does the following:

- uses the number of buffers that are specified by the BUFNO= system option to transfer data between disk and RAM
- bypasses intermediate buffer transfers when reading or writing data
- reads ahead the number of pages specified by the BUFNO= system option and places the data in memory before it is needed

When the data is needed, it is already in memory and is, in effect, a direct memory access.

Try different values of the BUFNO system option to tune each SAS job or DATA step.

58

The Scatter-Read/Gather-Write feature is active only for SAS I/O files that have the following attributes:

- contain a 4K-multiple pagesize (for example, 4096 or 8192) on 32-bit systems
- contain a 8K-multiple pagesize (for example, 8192 or 16384) on 64-bit systems

If an I/O file does not meet these criteria, SGIO is inactive for that file even though the SGIO option is specified.

To learn more, visit this page: http://support.sas.com/techsup/technote/ts710.html.

Exercises

1. **Recording Resource Statistics**

   a. Open the program, c01ex1Start, and add the appropriate OPTIONS statement to report the following statistics. Record your results.

   1) CPU ____________________

   2) I/O ____________________

   3) Memory ____________________

   b. Turn off the option after you record the statistics.

2. **Using the SASFILE Statement**

   Open the program, c01ex2Start, and add the appropriate statement(s) to open and load the entire data set **`ia.UK_fltat`** into memory. At the end of the program, close the data set.

1.5 Solutions to Exercises

1. Recording Resource Statistics

a. Open the program, c01ex1Start, and add the appropriate OPTIONS statement to report the following statistics. Record your results.

✎ Each student's results will vary depending on the individual PC.

1) CPU ______________________________

2) I/O ______________________________

3) Memory ______________________________

```
options fullstimer;

filename rawdata 'saledata.dat';

data sales(keep = FlightID Num1st
                  NumBus NumEcon NumPassTotal);
   infile rawdata;
   input FlightID $7.  RouteID $7.
         Origin $3.  Dest $3.
         DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
run;

options nofullstimer;
```

b. Turn off the option after you record the statistics.

2. Using the SASFILE Statement

Open the program, c01ex2Start, and add the appropriate statement(s) to open and load the entire data set **ia.UK_fltat** into memory. At the end of the program, close the data set.

```
sasfile ia.uk_fltat load;

proc print data = ia.uk_fltat;
run;

proc means data = ia.uk_fltat;
   var Salary;
run;

proc freq data = ia.uk_fltat;
   tables JobCode Gender;
run;

proc tabulate data = ia.uk_fltat;
   class Gender JobCode;
   var Salary;
   tables JobCode,Gender*Salary*(Mean Median);
run;

sasfile ia.uk_fltat close;
```

Chapter 2 Accessing Observations

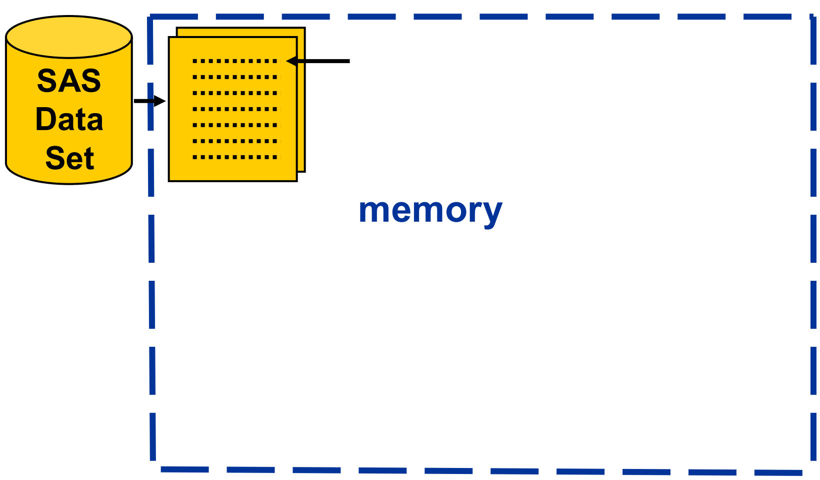
2.1 Introduction

Objectives

- Review sequential processing.
- Investigate methods for direct access.

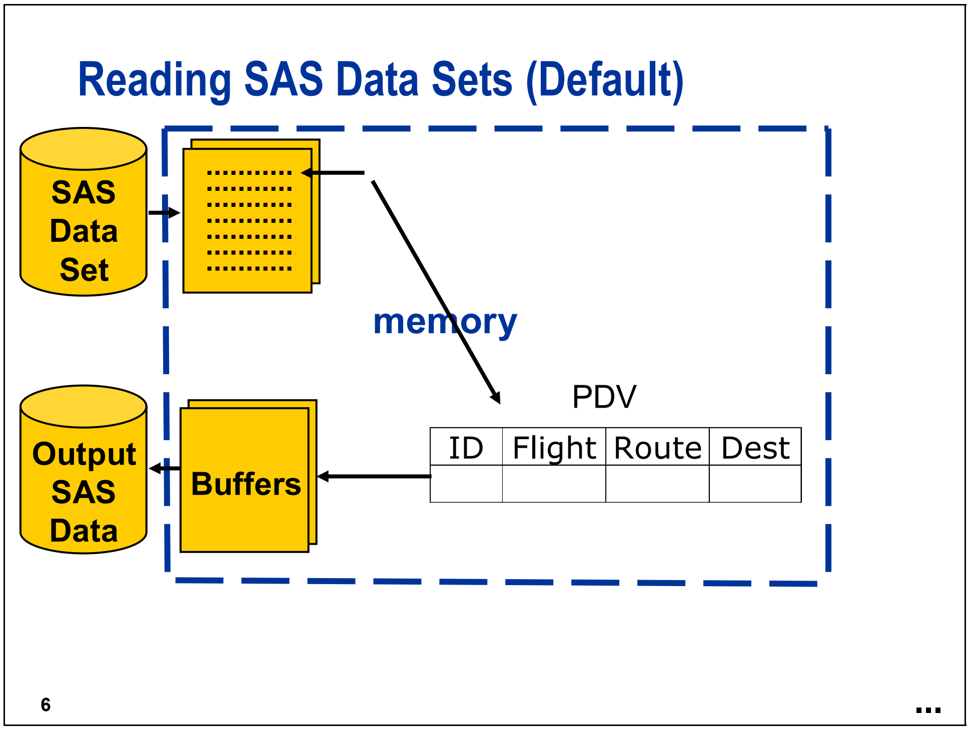
3

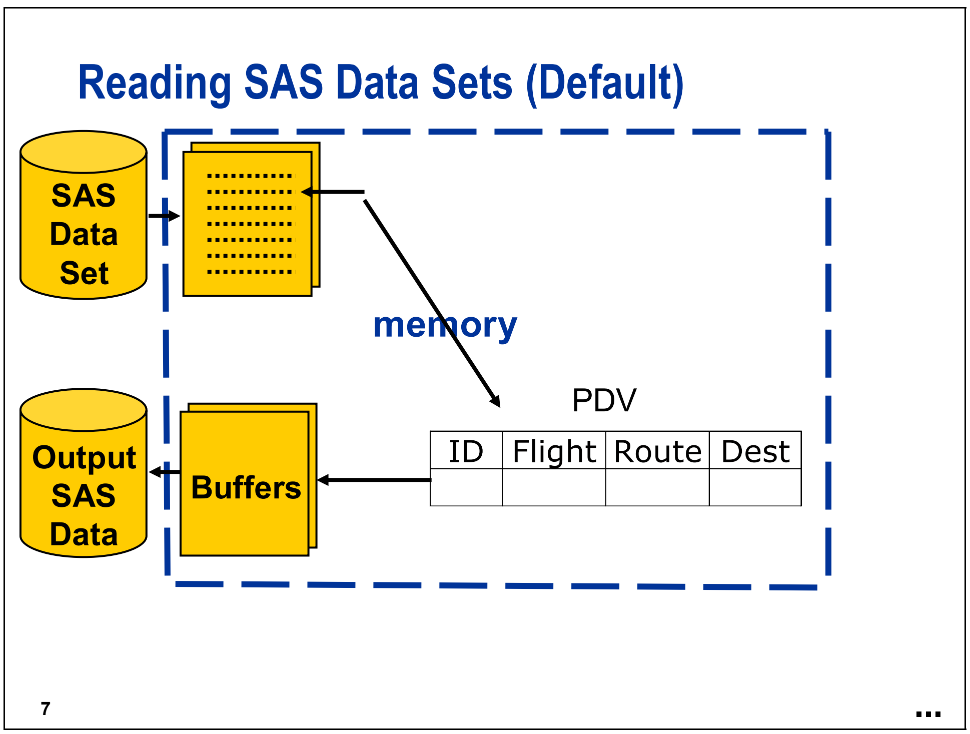
Reading SAS Data Sets (Default)

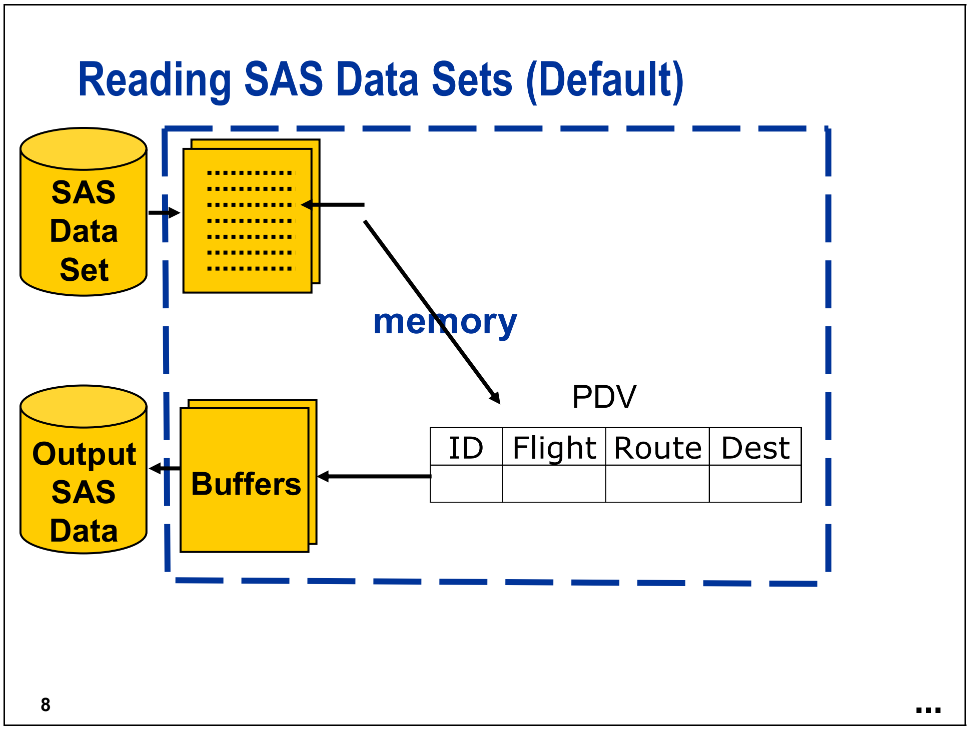


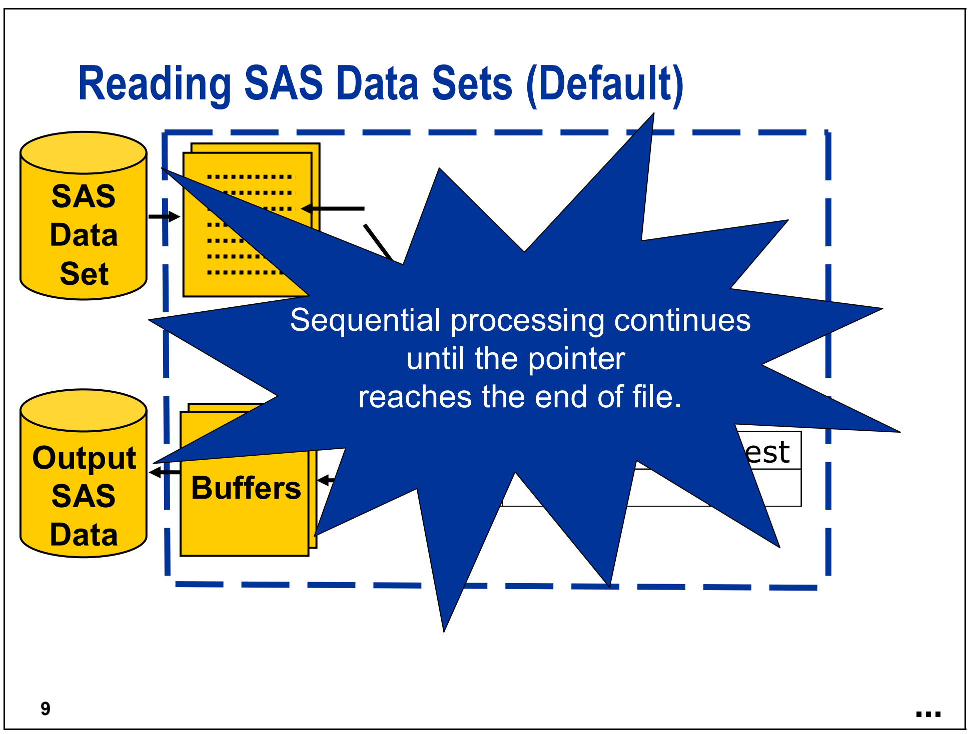
4

...


Reading SAS Data Sets (Default)
SAS Data Set
memory
PDV
ID
Flight
Route
Dest
Output SAS Data
Buffers
6
...


Reading SAS Data Sets (Default)
SAS Data Set
memory
PDV
ID
Flight
Route
Dest
Output SAS Data
Buffers
7
...


Reading SAS Data Sets (Default)
SAS Data Set
memory
PDV
ID
Flight
Route
Dest
Output SAS Data
Buffers
8
...


Reading SAS Data Sets (Default)
SAS Data Set
Sequential processing continues until the pointer reaches the end of file.
Output SAS Data
Buffers
9
...

Using Direct Access Methods

To change the default sequentially processing, you can use direct access methods.

Method:	Possible use:	How does it work?
POINT= SET statement option	creating a sample of data from a SAS data set	Locates an observation by observation number
Indexing	creating a subset of data with a WHERE clause	Locates an observation by variable value(s)

10

2.2 Creating a Sample Data Set

Objectives

- Create a systematic sample that contains five observations.
- Create a systematic sample that contains an unknown number of observations.
- Create a random sample with replacement.
- Create a random sample without replacement.

12

Selecting Observations

International Airlines (IA) is concerned with the accuracy of the data in **`ia.sales`** that contains revenue figures for 2004 and 2005. The size of the data set makes auditing all of the data difficult. IA first wants to audit a small sample to determine if a full audit is necessary.

Partial Output

Flight ID	RouteID	Origin	Dest	DestType	FltDate	Cap1st	CapBus	CapEcon	Cap Pass Total	CapCargo	Num1st	Num Bus	Num Econ	Num Pass Total
IA10700	0000107	WLG	AKL	International	01JAN2005	12	.	138	150	36900	11	.	126	137
IA10701	0000107	WLG	AKL	International	01JAN2005	12	.	138	150	36900	12	.	136	148
IA10702	0000107	WLG	AKL	International	01JAN2005	12	.	138	150	36900	10	.	112	122
IA10703	0000107	WLG	AKL	International	01JAN2005	12	.	138	150	36900	12	.	113	125
IA10704	0000107	WLG	AKL	International	01JAN2005	12	.	138	150	36900	10	.	118	128
IA10705	0000107	WLG	AKL	International	01JAN2005	12	.	138	150	36900	11	.	117	128
IA10700	0000107	WLG	AKL	International	02JAN2005	12	.	138	150	36900	10	.	131	141
IA10701	0000107	WLG	AKL	International	02JAN2005	12	.	138	150	36900	11	.	113	124
IA10702	0000107	WLG	AKL	International	02JAN2005	12	.	138	150	36900	10	.	134	144
IA10703	0000107	WLG	AKL	International	02JAN2005	12	.	138	150	36900	11	.	114	125
IA10704	0000107	WLG	AKL	International	02JAN2005	12	.	138	150	36900	11	.	128	139
IA10705	0000107	WLG	AKL	International	02JAN2005	12	.	138	150	36900	12	.	131	143
IA10700	0000107	WLG	AKL	International	03JAN2005	12	.	138	150	36900	10	.	124	134
IA10701	0000107	WLG	AKL	International	03JAN2005	12	.	138	150	36900	12	.	135	147
IA10702	0000107	WLG	AKL	International	03JAN2005	12	.	138	150	36900	12	.	127	139

13 ...

✎ The data set **`ia.sales`** used for demonstrations and exercises contains fewer observations than the data set **`ia.sales`** used for the course notes.

Creating a Systematic Sample

Select a five-observation subset by reading every hundredth observation from observation number 100 to observation number 500.

```
data work.subset;
   do PickIt = 100 to 500 by 100; ❶
      set ia.sales
          point = PickIt; ❷
      output; ❸
   end;
   stop; ❹
run;
```

14

c02s2d1

① The DO loop assigns a value to the variable **PickIt**.

② **PickIt** is used by the POINT= option to select an observation from the SAS data set.

③ The OUTPUT statement writes the PDV values to the SAS data set.

④ The STOP statement stops the DATA step from continuing to execute after the five observations are selected. Without a STOP statement, the DATA step continues in an infinite loop

Using the POINT= Option

To create a sample, use the POINT= option in the SET statement.

General form of the POINT= option:

SET *data-set-name* POINT = *point-variable*;

The *point-variable* has the following attributes:

- names a temporary numeric variable that contains the observation number of the observation to read
- must be given a value before the execution of the SET statement
- must be a variable (for example, X) and not a constant value (for example, 12)

15 ...

The POINT= option value should be an integer greater than zero and less than or equal to the number of observations in the SAS data set. If the value is not integral, the SET statement effectively applies the FLOOR function to the value.

Using the STOP Statement

The POINT= option has the following features:

- uses direct-access read mode
- does not detect the end-of-file

To prevent the DATA step from looping continuously, use the STOP statement.

General form of the STOP statement:

STOP;

17

c02s2d1

```
data work.subset;
   do PickIt = 100 to 500 by 100;
      set ia.sales
          point = PickIt;
      output;
   end;
   stop;
run;
```

The PROC PRINT output of **work.subset** is shown below.

Creating a Systematic Sample of 5 Observations

Obs	Flight ID	RouteID	Origin	Dest	DestType	FltDate	Cap1st	CapBus
1	IA09200	0000092	CCU	DEL	International	01JAN2004	12	.
2	IA02501	0000025	RDU	IND	Domestic	01JAN2004	12	.
3	IA01101	0000011	RDU	ORD	Domestic	01JAN2004	12	.
4	IA04203	0000042	PWM	RDU	Domestic	01JAN2004	12	.
5	IA04901	0000049	LHR	BRU	International	02JAN2004	14	.

Obs	CapEcon	Cap Pass Total	CapCargo	Num1st	Num Bus	Num Econ	Num Pass Total	Rev1st	RevBus
1	138	150	36900	10	.	110	120	$3,360.00	.
2	138	150	36900	12	.	134	146	$2,472.00	.
3	138	150	36900	11	.	126	137	$2,915.00	.
4	138	150	36900	10	.	116	126	$2,890.00	.
5	125	139	39700	12	.	124	136	$1,020.00	.

Obs	RevEcon	CargoRev	RevTotal	Cargo Weight
1	$12,210.00	$6,708.00	$22,278	12900
2	$9,112.00	$2,464.00	$14,048	7700
3	$11,088.00	$3,895.00	$17,898	9500
4	$11,136.00	$5,148.00	$19,174	11700
5	$3,472.00	$1,625.00	$6,117	12500

✎ The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Using the Number of Observations

You must select a subset by reading every hundredth observation from observation number 100 to the end of the SAS data set.

```
data work.subset;
   do PickIt = 100 to TotObs by 100; ❷
      set ia.sales point = PickIt
                          nobs = TotObs; ❶
      output;
   end;
   stop;
run;
```

18

c02s2d2

① The NOBS= option creates a temporary variable that contains the total number of observations in the input data files. During compilation, SAS reads the descriptor portion of the data file and assigns the value of the NOBS= variable.

✎ The total includes deleted observations. Rebuild the data set to remove deleted observations.

② You can refer to the NOBS= variable in executable statements that appear before the SET statement.

Using the Number of Observations

You can use the NOBS= option in the SET statement to determine how many observations there are in a SAS data set.

General form of the SET statement:

SET *SAS-data-set* NOBS = *variable*;

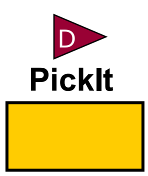
The NOBS= option creates a temporary variable whose value has the following characteristics:

- is the number of observations in the input data set(s)
- assigned during compilation
- retained
- should not be modified during execution

19

Compilation

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```



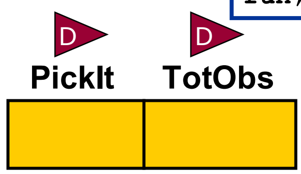
20

c02s2d2

...

Compilation

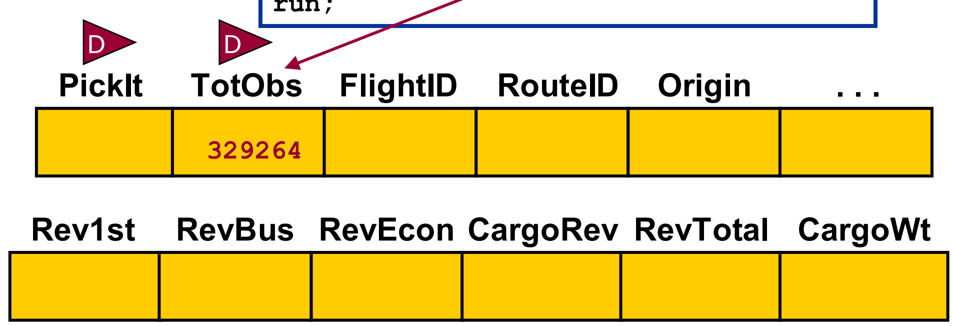
```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```



21 ...

Compilation

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```



22 ...

✎ The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
100	329264	IA10703	0000107	WLG	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1524.00	.	4956.00	2180.00	8660.00	10900.00

24 ...

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```

Explicit Output

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
100	329264	IA10703	0000107	WLG	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1524.00	.	4956.00	2180.00	8660.00	10900.00

25 ...

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```

PickIt = 200

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
200	329264	IA10703	0000107	WLG	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1524.00	.	4956.00	2180.00	8660.00	10900.00

26 ...

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                   nobs = TotObs;
      output;
   end;
   stop;
run;
```

Explicit Output

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
200	329264	IA10701	0000107	WLG	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1270.00	.	5796.00	1460.00	8526.00	7300.00

28 ...

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                    nobs = TotObs;
      output;
   end;
   stop;
run;
```

PickIt > TotObs

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
329300	329264	IA10801	0000108	AKL	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1524.00	.	4998.00	2140.00	8662.00	10700.00

30 ...

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales point = PickIt
                    nobs = TotObs;
      output;
   end;
   stop;
run;
```

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
329300	329264	IA10801	0000108	AKL	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1524.00	.	4998.00	2140.00	8662.00	10700.00

31 ...

Execution

```
data work.subset;
   do PickIt = 100 to TotObs by 100;
      set ia.sales poin  = PickIt
                       bs = TotObs;
      output;
   end;
   stop;
run;
```

Execution STOPS

PickIt (D)	TotObs (D)	FlightID	RouteID	Origin	. . .
329300	329264	IA10801	0000108	AKL	

Rev1st	RevBus	RevEcon	CargoRev	RevTotal	CargoWt
1524.00	.	4998.00	2140.00	8662.00	10700.00

32

c02s2d2

Partial PROC PRINT Output of **work.subset**

```
                                   A Systematic Sample of Fares

    Flight
Obs   ID    RouteID Origin Dest DestType        FltDate   Cap1st   CapBus

  1 IA09200 0000092  CCU   DEL  International 01JAN2004      12         .
  2 IA02501 0000025  RDU   IND  Domestic      01JAN2004      12         .
  3 IA01101 0000011  RDU   ORD  Domestic      01JAN2004      12         .
  4 IA04203 0000042  PWM   RDU  Domestic      01JAN2004      12         .
  5 IA04901 0000049  LHR   BRU  International 02JAN2004      14         .
  6 IA06405 0000064  FBU   FRA  International 02JAN2004      14         .
  7 IA05203 0000052  GVA   LHR  International 02JAN2004      14         .
  8 IA02000 0000020  BOS   RDU  Domestic      02JAN2004      12         .
  9 IA10802 0000108  AKL   WLG  International 02JAN2004      12         .
 10 IA08900 0000089  JRS   DEL  International 03JAN2004      14        30
 11 IA01305 0000013  RDU   IAD  Domestic      03JAN2004      12         .
 12 IA03705 0000037  RDU   MSY  Domestic      03JAN2004      12         .

                Cap                                      Num
                Pass                      Num    Num     Pass
Obs   CapEcon   Total   CapCargo  Num1st  Bus   Econ    Total           Rev1st

  1       138     150      36900     10     .    110     120         $3,360.00
  2       138     150      36900     12     .    134     146         $2,472.00
  3       138     150      36900     11     .    126     137         $2,915.00
  4       138     150      36900     10     .    116     126         $2,890.00
  5       125     139      39700     12     .    124     136         $1,020.00
  6       125     139      39700     14     .    101     115         $3,976.00
  7       125     139      39700     12     .    109     121         $2,280.00
  8       138     150      36900     11     .    120     131         $2,772.00
  9       138     150      36900     11     .    108     119         $1,397.00
 10       163     207      82400     12    26    145     183        $12,372.00
 11       138     150      36900     12     .    130     142         $1,140.00
 12       138     150      36900     11     .    122     133         $3,520.00

                                                                            Cargo
Obs          RevBus           RevEcon          CargoRev         RevTotal   Weight

  1               .        $12,210.00         $6,708.00          $22,278    12900
  2               .         $9,112.00         $2,464.00          $14,048     7700
  3               .        $11,088.00         $3,895.00          $17,898     9500
  4               .        $11,136.00         $5,148.00          $19,174    11700
  5               .         $3,472.00         $1,625.00           $6,117    12500
  6               .         $9,494.00         $7,181.00          $20,651    16700
  7               .         $6,867.00         $4,495.00          $13,642    15500
  8               .         $9,960.00         $4,173.00          $16,905    10700
  9               .         $4,536.00         $2,620.00           $8,553    13100
 10      $18,278.00        $49,590.00        $72,364.00         $152,604    45800
 11               .         $4,160.00         $1,275.00           $6,575     8500
 12               .        $12,932.00         $5,047.00          $21,499    10300
```

✐ The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Creating a Random Sample

There are several random number functions to generate random numbers from various distributions.

General form of the RANUNI function:

```
RANUNI(seed)
```

33

The UNIFORM function is an alias for the RANUNI function.

The *seed* is an initial starting point that the RANUNI function uses to generate streams of random numbers.

The seed must be an integer with a value less than 2^{31}-1 (2,147,483,647).

✎ A 0 argument for the RANUNI function uses the system clock time, resulting in a different stream of random numbers each time that the program is run.

Using the RANUNI Function

The RANUNI function returns a rational number between 0 and 1 (non-inclusive) generated from a uniform distribution.

ranuni(*seed*)

Examples:

Random number

.01253689

.95196500

34 ...

Using the RANUNI Function

If you want a number between 0 and 5 (non-inclusive), use the following:

ranuni(*seed*) * 5

Examples:

Random number * 5

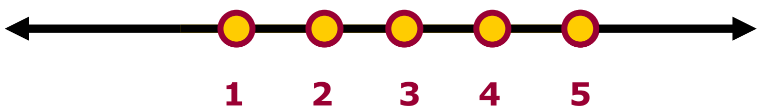
.01253689 → 0.06268445

.95196500 → 4.75982500

35 ...

Using the RANUNI and CEIL Functions

If you want an integer between 1 and 5 (inclusive), use the following:



CEIL(ranuni(*seed*) * 5)

Examples:

Random number	* 5	CEIL( )
.01253689 →	0.06268445 →	1
.95196500 →	4.75982500 →	5

36

The CEIL function returns the smallest integer that is greater than or equal to the argument.

Creating a Random Sample

c02s2d3

Create a random sample **with replacement**. A sample with replacement can contain duplicate observations because an observation can be selected more than one time.

```
data work.subset (drop = i SampSize);
   SampSize = 10;
   do i = 1 to SampSize;
      PickIt = ceil(ranuni(0)*TotObs);
      set ia.sales point = PickIt nobs = TotObs;
      output;
   end;
   stop;
run;

proc print data = work.subset;
   title 'A Random Sample with Replacement';
run;
```

Output

A Random Sample with Replacement

Obs	Flight ID	RouteID	Origin	Dest	DestType	FltDate	Cap1st	CapBus
1	IA04604	0000046	GLA	LHR	International	04APR2005	14	.
2	IA06302	0000063	FRA	FBU	International	29NOV2005	14	.
3	IA01003	0000010	LAX	RDU	Domestic	28JUL2004	16	.
4	IA01502	0000015	RDU	SEA	Domestic	26APR2005	16	.
5	IA09000	0000090	DEL	JRS	International	05DEC2005	14	30
6	IA02003	0000020	BOS	RDU	Domestic	09JAN2004	12	.
7	IA03000	0000030	HNL	SFO	Domestic	28MAY2005	14	30
8	IA01302	0000013	RDU	IAD	Domestic	20FEB2004	12	.
9	IA01602	0000016	SEA	RDU	Domestic	06MAY2005	16	.
10	IA06802	0000068	PRG	LHR	International	21FEB2004	14	.

Obs	CapEcon	Cap Pass Total	CapCargo	Num1st	Num Bus	Num Econ	Num Pass Total	Rev1st
1	125	139	39700	13	.	106	119	$1,846.00
2	125	139	39700	14	.	95	109	$3,976.00
3	251	267	77400	16	.	227	243	$14,816.00
4	251	267	77400	15	.	208	223	$14,610.00
5	163	207	82400	13	24	150	187	$13,403.00
6	138	150	36900	10	.	111	121	$2,520.00
7	163	207	82400	13	27	132	172	$12,844.00
8	138	150	36900	11	.	129	140	$1,045.00
9	251	267	77400	13	.	241	254	$12,662.00
10	125	139	39700	12	.	124	136	$3,192.00

(Continued on the next page.)

Obs	RevBus	RevEcon	CargoRev	RevTotal	Cargo Weight
1	.	$4,982.00	$3,498.00	$10,326	15900
2	.	$8,930.00	$7,697.00	$20,603	17900
3	.	$69,689.00	$40,896.00	$125,401	28800
4	.	$67,184.00	$48,872.00	$130,666	32800
5	$16,872.00	$51,300.00	$71,100.00	$152,675	45000
6	.	$9,213.00	$4,953.00	$16,686	12700
7	$18,171.00	$43,296.00	$72,960.00	$147,271	48000
8	.	$4,128.00	$1,335.00	$6,508	8900
9	.	$77,843.00	$39,634.00	$130,139	26600
10	.	$10,912.00	$5,125.00	$19,229	12500

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

c02s2d4 (Self-Study)

Create a random sample **without** replacement. A sample without replacement cannot contain duplicate observations because after an observation is output to **work.subset**, programmatically it cannot be selected again.

✐ The following program can be used as a template. Replace the following:

- **work.subset** with the name of your resulting SAS data set
- **ia.sales** with the name of the data set from which to sample
- the 10 in the **SampSize = 10** statement with the number of observations to read

```
data work.subset(drop = ObsLeft SampSize);
①  SampSize = 10;
②  ObsLeft = TotObs;
   do while(SampSize > 0 and ObsLeft > 0);
    ③ PickIt + 1;
      if ranuni(0) < SampSize/ObsLeft then
         do;
            set ia.sales point = PickIt
                         nobs = TotObs;
            output;
            SampSize = SampSize - 1;
         end;
      ObsLeft = ObsLeft - 1;
   end;
   stop;
run;

proc print data = work.subset;
   title 'A Random Sample without Replacement';
run;
```

① **SampSize** is the number of observations wanted in the sample.

② **ObsLeft** is the number of observations still needed to be selected. The start value is equal to **TotObs**, the total number of observations in the data set being sampled.

③ **PickIt** is the number of the observation to be read in the sample data set. Because it is used in a SUM statement, its starting value is 0.

✐ The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

In each iteration of the DO loop, the following occurs:

1. **`PickIt`** is incremented by 1.
2. The IF expression **`ranuni(0) < Sampsize/ObsLeft`** is evaluated:
   a. If true, these actions occur:
      1) The observation **`PickIt`** is selected in the sample.
      2) **`SampSize`** is decreased by 1.
   b. If false, the observation **`PickIt`** is skipped.
3. **`ObsLeft`** is decreased by 1.

The process ends when **`SampSize`** is 0; no additional observations are needed.

Take note of the following:

- Each observation is considered for selection.
- An observation number is considered only once.
- The data set is read-only when an observation number is selected.

This is an adaptation of a sampling routine that has been used by statisticians for many years.

- The sample size is fixed.
- An observation can be selected only once.
- Each observation has an equal probability of being selected.
- The selection probability for an observation is independent of the selection of another observation.

Output

```
                                A Random Sample without Replacement

    Flight
Obs   ID    RouteID Origin Dest DestType        FltDate   Cap1st   CapBus

  1 IA02000 0000020  BOS   RDU  Domestic      08JAN2004       12        .
  2 IA06502 0000065  FRA   ARN  International 08FEB2004       14        .
  3 IA11201 0000112  SFO   HND  International 23JUN2004       19       35
  4 IA01804 0000018  SFO   SEA  Domestic      15JUL2004       12        .
  5 IA04605 0000046  GLA   LHR  International 08SEP2004       14        .
  6 IA01803 0000018  SFO   SEA  Domestic      09SEP2004       12        .
  7 IA02203 0000022  DFW   RDU  Domestic      18JAN2005       12        .
  8 IA05205 0000052  GVA   LHR  International 23MAR2005       14        .
  9 IA03904 0000039  RDU   MCI  Domestic      23JUN2005       12        .
 10 IA04200 0000042  PWM   RDU  Domestic      10DEC2005       12        .

                Cap                                  Num
                Pass                    Num   Num    Pass
Obs  CapEcon    Total   CapCargo  Num1st  Bus  Econ   Total           Rev1st

  1      138     150      36900     11      .   133    144         $2,772.00
  2      125     139      39700     11      .   100    111         $3,377.00
  3      201     255     105500     17     32   193    242        $36,091.00
  4      138     150      36900     12      .   134    146         $3,360.00
  5      125     139      39700     11      .    97    108         $1,562.00
  6      138     150      36900     10      .   113    123         $2,800.00
  7      138     150      36900     10      .   137    147         $4,350.00
  8      125     139      39700     14      .   106    120         $2,660.00
  9      138     150      36900     11      .   125    136         $4,092.00
 10      138     150      36900     12      .   116    128         $3,468.00

                                                                       Cargo
Obs          RevBus           RevEcon          CargoRev         RevTotal  Weight

  1               .        $11,039.00         $3,159.00          $16,970    8100
  2               .        $10,200.00         $8,225.00          $21,802   17500
  3      $46,304.00       $136,065.00       $186,146.00         $404,606   57100
  4               .        $12,462.00         $3,311.00          $19,133    7700
  5               .         $4,559.00         $3,982.00          $10,103   18100
  6               .        $10,509.00         $5,289.00          $18,598   12300
  7               .        $19,728.00         $5,025.00          $29,103    7500
  8               .         $6,678.00         $4,553.00          $13,891   15700
  9               .        $15,500.00         $5,529.00          $25,121    9700
 10               .        $11,136.00         $4,972.00          $19,576   11300
```

With a seed value of 0, you get different results each time that the program is executed, but it is possible that some of the same observations will be selected as were selected in previous executions.

Using the SURVEYSELECT Procedure (Self-Study)

The SURVEYSELECT procedure has the following attributes:

- provides a variety of methods for selecting probability-based random samples
- can select a simple random sample or can sample according to a complex multistage sample design that includes stratification, clustering, and unequal probabilities of selection
- is part of SAS/STAT

38

Using the SURVEYSELECT Procedure (Self-Study)

This program creates a SAS data set, **sample**, containing 100 observations randomly selected from the **ia.sales** SAS data set.

```
proc surveyselect data = ia.sales
   method = srs n = 100
   out = sample;
run;
```

39

c02s2d5

Using the SURVEYSELECT Procedure (Self Study)

General form of the SURVEYSELECT procedure:

```
PROC SURVEYSELECT options;
      STRATA variables;
      CONTROL variables;
      SIZE variable;
      ID variables;
RUN;
```

40

The STRATA statement	partitions the input data set into non-overlapping groups defined by the STRATA variables. PROC SURVEYSELECT then selects independent samples from these strata, according to the selection method and design parameters specified in the PROC SURVEYSELECT statement. PROC SURVEYSELECT expects the input data set to be sorted in the order of the STRATA variables.
The CONTROL statement	names variables for sorting the input data set. The CONTROL variables can be character or numeric. PROC SURVEYSELECT sorts the input data set by the CONTROL variables before selecting the sample. If you also specify a STRATA statement, PROC SURVEYSELECT sorts by the CONTROL variables within the strata.
The SIZE statement	names one and only one size measure variable, which contains the size measures to be used when sampling with probability proportional to size. The SIZE variable must be numeric. When the value of an observation's SIZE variable is missing or non-positive, that observation has no chance of being selected for the sample.
The ID statement	names variables from the DATA= input data set to be included in the OUT= data set of selected units. If there is no ID statement, PROC SURVEYSELECT includes all variables from the DATA= data set in the OUT= data set. The ID variables can be character or numeric.

Using the SURVEYSELECT Procedure (Self-Study)

The PROC SURVEYSELECT statement performs the following tasks:

- invokes the procedure
- optionally identifies input and output data sets
- specifies the sample selection method, the sample size, and other sample design parameters

The PROC SURVEYSELECT statement is the only statement required to create a simple random sample.

41

Options for the SURVEYSELECT Procedure (Self-Study)

The following options can be specified in the PROC SURVEYSELECT statement:

To do this:	Use this option:
Specify the input data set	DATA=
Specify output data sets	OUT=
Suppress displayed output	NOPRINT
Specify selection method	METHOD=
Specify sample size	SAMPSIZE=
Specify random number seed	SEED=

42

Methods Used by the SURVEYSELECT Procedure (Self-Study)

Selected values for the METHOD= option are as follows:

METHOD=	
SYS	The method of systematic random sampling selects units at a fixed interval throughout the sampling frame or stratum after a random start.
URS	The method of unrestricted random sampling selects units with equal probability and with replacement. Because units are selected with replacement, a unit can be selected for the sample more than once.
SRS	The method of simple random sampling selects units with equal probability and without replacement. The selection probability for each individual unit equals n/N.

43

These methods correspond to the DATA step examples at the beginning of this section.

Reviewing the SURVEYSELECT Procedure Example (Self-Study)

This program creates a SAS data set, **`sample`**, containing 100 observations randomly selected from the **`ia.sales`** SAS data set.

```
proc surveyselect data = ia.sales
     method = srs n = 100
     out = sample;
run;
```

44

c02s2d5

The SURVEYSELECT procedure step produces similar output to the **`c02s2d3`** example earlier in this chapter, except that it selects more samples (100 versus 10).

Using the SURVEYSELECT Procedure (Self-Study)

In addition to creating the SAS data set, **`Sample`**, PROC SURVEYSELECT provides the following information in the Output window:

```
         The SURVEYSELECT Procedure

Selection Method    Simple Random Sampling

Input Data Set                  SALES
Random Number Seed          955326001
Sample Size                       100
Selection Probability        0.000304
Sampling Weight               3292.64
Output Data Set                SAMPLE
```

Because the SEED= option is not specified in the PROC SURVEYSELECT statement, the seed value is obtained using the time of day from the computer's clock.

45

To specify a seed so that you can replicate a sample, use the SEED= option on the PROC SURVEYSELECT statement.

```
proc surveyselect data = ia.sales
                  method = srs n = 100
                  out = sample
                  seed = 12345;
run;
```

Using the SURVEYSELECT Procedure (Self-Study)

In addition to creating the SAS data set, **`Sample`**, PROC SURVEYSELECT provides the following information in the log:

```
        The SURVEYSELECT Procedure

Selection Method     Simple Random Sampling

Input Data Set                  SALES
Random Number Seed          955326001
Sample Size                       100
Selection Probability        0.000304
Sampling Weight               3292.64
Output Data Set                SAMPLE
```

The Selection Probability for each individual unit is calculated as 100/329264 (sample size/number of observations in the input data set).

46

Using the SURVEYSELECT Procedure (Self-Study)

In addition to creating the SAS data set, **`Sample`**, PROC SURVEYSELECT provides the following information in the log:

```
        The SURVEYSELECT Procedure

Selection Method     Simple Random Sampling

Input Data Set                  SALES
Random Number Seed          955326001
Sample Size                       100
Selection Probability        0.000304
Sampling Weight               3292.64
Output Data Set                SAMPLE
```

The Sampling Weight is the inverse of the selection probability, 329264/100.

47

Partial Output from the SAS Data Set **SAMPLE**

```
              Using PROC SURVEYSELECT to create a Random Sample without Replacement

    Flight
Obs   ID    RouteID Origin Dest DestType        FltDate   Cap1st   CapBus

  1 IA06900 0000069  LHR   AMS  International 29OCT2005       14        .
  2 IA01905 0000019  RDU   BOS  Domestic      14FEB2005       12        .
  3 IA01904 0000019  RDU   BOS  Domestic      22MAY2005       12        .
  4 IA04901 0000049  LHR   BRU  International 26JAN2005       14        .
  5 IA10303 0000103  SYD   CBR  International 30OCT2005       12        .
  6 IA09103 0000091  DEL   CCU  International 08MAR2005       12        .
  7 IA09801 0000098  PEK   CCU  International 23DEC2005       28       52
  8 IA04301 0000043  LHR   CDG  International 15NOV2005       14        .
  9 IA06001 0000060  MAD   CDG  International 23NOV2005       14        .
 10 IA06000 0000060  MAD   CDG  International 26NOV2005       14        .
 11 IA05701 0000057  FRA   CPH  International 05APR2005       14        .
 12 IA08500 0000085  FRA   CPT  International 12JUL2005       19       56

            Cap                                Num
            Pass                     Num  Num   Pass
Obs  CapEcon  Total   CapCargo  Num1st  Bus  Econ  Total           Rev1st

  1      125    139      39700     13     .   106   119         $1,170.00
  2      138    150      36900     11     .   115   126         $2,772.00
  3      138    150      36900     12     .   137   149         $3,024.00
  4      125    139      39700     14     .   101   115         $1,190.00
  5      138    150      36900     12     .   118   130           $768.00
  6      138    150      36900     12     .   131   143         $4,032.00
  7      157    237      85900     28    48   146   222        $23,324.00
  8      125    139      39700     14     .   106   120         $1,274.00
  9      125    139      39700     14     .   115   129         $3,710.00
 10      125    139      39700     13     .   112   125         $3,445.00
 11      125    139      39700     12     .   106   118         $2,088.00
 12      163    238     105500     18    50   124   192        $43,344.00

                                                                  Cargo
Obs          RevBus          RevEcon         CargoRev         RevTotal    Wt

  1               .        $3,074.00        $2,226.00        $6,470.00  15900
  2               .        $9,545.00        $4,563.00       $16,880.00  11700
  3               .       $11,371.00        $2,769.00       $17,164.00   7100
  4               .        $2,828.00        $2,171.00        $6,189.00  16700
  5               .        $2,478.00        $1,090.00        $4,336.00  10900
  6               .       $14,541.00        $4,316.00       $22,889.00   8300
  7      $27,264.00       $40,442.00       $53,120.00      $144,150.00  41500
  8               .        $3,180.00        $2,198.00        $6,652.00  15700
  9               .       $10,120.00        $5,699.00       $19,529.00  13900
 10               .        $9,856.00        $6,027.00       $19,328.00  14700
 11               .        $6,042.00        $4,347.00       $12,477.00  16100
 12      $82,050.00       $99,076.00      $247,599.00      $472,069.00  67100
```

Comparison of the DATA Step and the SURVEYSELECT Procedure (Self-Study)

DATA Step	PROC SURVEYSELECT
Full power of DATA step processing	Less coding
Can create multiple output data sets	One output data set with additional statistics
Part of Base SAS	Part of SAS/STAT

48

Exercises

1. Generating a Random Sample with Replacement

Generate a random sample **with** replacement of 50 employees from **`ia.salcomps`** to analyze their current salaries.

If the current salary is over $30,000, then place the employee's information in the **`work.over30`** SAS data set.

If the current salary is $30,000 or less, then place the employee's information in the **`work.ltoreq30`** SAS data set.

✎ If you obtain zero observations in one of the data sets, run the program again. It is possible that the selected observations might all be over $30,000 or all $30,000 or less.

2. Generating a Random Sample without Replacement (Optional)

Generate a random sample **without** replacement of ten flights from **`ia.cap2000`**.

2.3 Creating and Using an Index

Objectives

- Define indexes.
- List the uses of indexes.
- Use the DATA step to create indexes.
- Use PROC DATASETS to create and maintain indexes.
- Use PROC SQL to create and maintain indexes.

51

Using Indexes

To decrease the time used to query a heavily used SAS data set, create an index on **ia.sales**.

```
         Flight
  Obs      ID      RouteID   Origin   Dest     DestType        FltDate . . .

    1    IA10700   0000107    WLG     AKL    International   01JAN2004 . . .
    2    IA10701   0000107    WLG     AKL    International   01JAN2004 . . .
    3    IA10702   0000107    WLG     AKL    International   01JAN2004 . . .
    4    IA10703   0000107    WLG     AKL    International   01JAN2004 . . .
    5    IA10704   0000107    WLG     AKL    International   01JAN2004 . . .
           .          .         .       .           .
           .          .         .       .           .
           .          .         .       .           .

          Flight
  Obs      ID      RouteID   Origin   Dest     DestType        FltDate . . .

329259   IA10800   0000108    AKL     WLG    International   30DEC2005 . . .
329260   IA10801   0000108    AKL     WLG    International   30DEC2005 . . .
329261   IA10802   0000108    AKL     WLG    International   30DEC2005 . . .
329262   IA10803   0000108    AKL     WLG    International   30DEC2005 . . .
329263   IA10804   0000108    AKL     WLG    International   30DEC2005 . . .
329264   IA10805   0000108    AKL     WLG    International   30DEC2005 . . .
```

52

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Using Indexes

Indexed SAS Data Set

obs	Flight ID	RouteID	Origin	Dest	DestType	FltDate . . .
329259	IA10800	0000108	AKL	WLG	International	30DEC2005 . . .
329260	IA10801	0000108	AKL	WLG	International	30DEC2005 . . .
329261	IA10802	0000108	AKL	WLG	International	30DEC2005 . . .
329262	IA10803	0000108	AKL	WLG	International	30DEC2005 . . .

Simplified Index File

Key Variable=Origin

Key Value	Record Identifiers Page(obs,obs...)
AKL	25(1,2,3,...) 32(...)...
AMS	82(22,23,...) 96(...)...
ANC	75(18,34,...) 96(...)...
. . .	. . .

53

The index is stored with the key values in sorted order.

Using Indexes

An *index* is an optional file that you can create for a SAS data file that does the following:

- points to observations based on the values of one or more key variables
- provides direct access to specific observations

In other words, index usage locates an observation by value.

54

This section discusses indexes for Base SAS data files. A discussion of indexes for Scalable Performance Data Engine (SPDE) data files is presented in a later chapter.

The Purpose of Indexes

Indexes can provide direct access to observations in SAS data sets to accomplish the following:

- yield faster access to small subsets (WHERE)
- return observations in sorted order (BY)
- perform table lookup operations (SET with KEY=)
- join observations (PROC SQL)
- modify observations (MODIFY with KEY=)

55

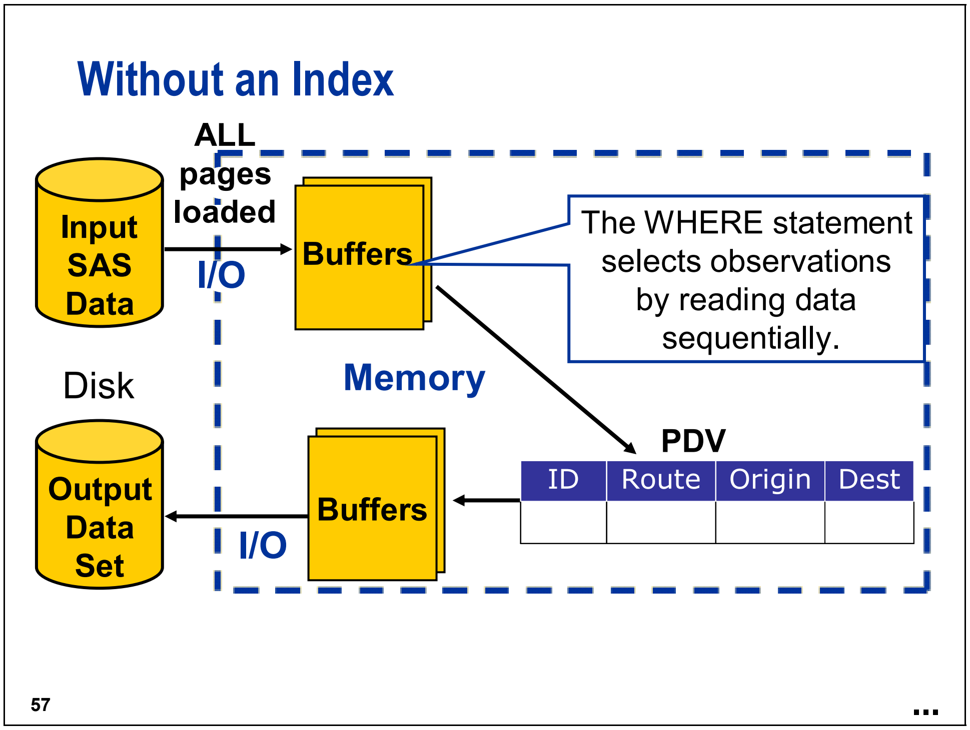
Why Use an Index?

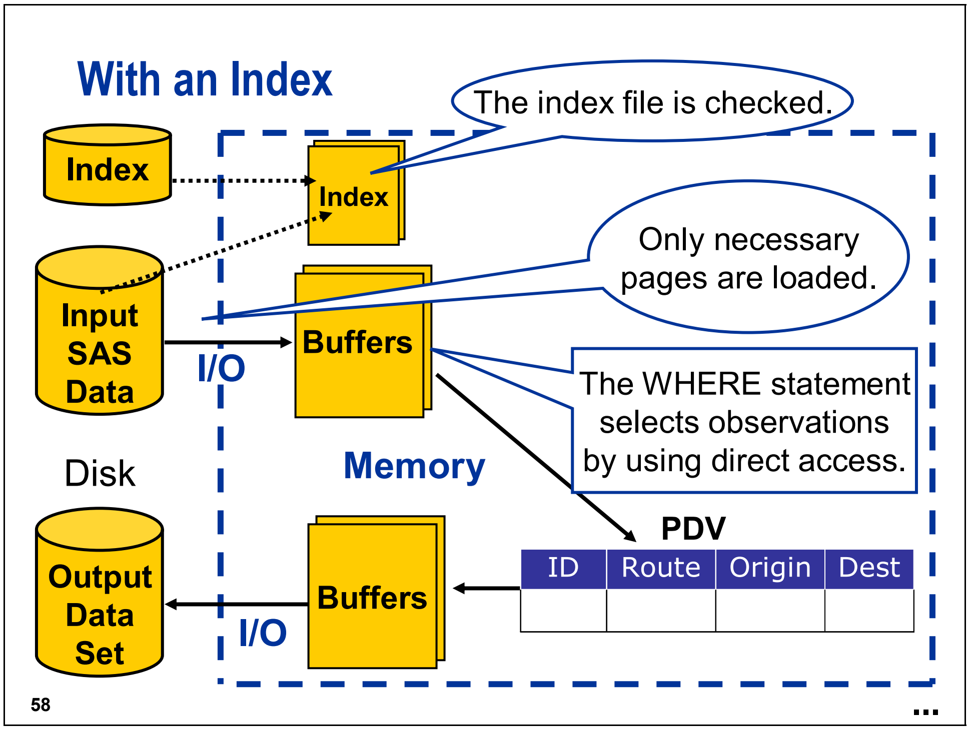
```
data _null_;
   set ia.sales;
   where FltDate = '02JUL2004'd;
run;
```

What happens when you submit this program?

56

...


Without an Index
ALL pages loaded
Input SAS Data
I/O
Buffers
The WHERE statement selects observations by reading data sequentially.
Disk
Memory
PDV
ID
Route
Origin
Dest
Output Data Set
I/O
Buffers
57
...



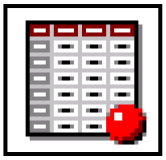
When SAS uses an index to process data, SAS accomplishes the following:

- performs a binary search on the index file
- positions the index to the first entry containing a qualified value
- transfers a page of data containing the first record identifier for the qualified value to a buffer
- directly accesses the value specified by the record identifier
- positions the index to the next entry containing a qualified value
- transfers the page of data, if it is not already in the buffer
- directly accesses the value specified by the record identifier
- continues to process the data until there is no more data that satisfies the WHERE expression

✎ If the data values are sorted in ascending order by the indexed variables, fewer I/O operations are required. In addition, if observations with the same key values are near each other in the file, for whatever reason, I/O will be minimized.

Using Indexes

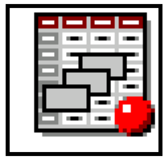
- The index file consists of entries that are organized in a tree structure, and connected by pointers.
- When an index is used to process a request, such as for WHERE processing, SAS searches the index file in order to locate the requested record(s) rapidly.



FlightID
FltDate
Origin

Key variables in **ia.sales**

sales.sas7bdat



Origin
DteFlt

Indexes in the index file for **ia.sales**

sales.sas7bndx

Directory-based Index File Naming Conventions

59

Index Terminology

There are two types of indexes.

Type	Based On	Name	Example
Simple	the value of only one variable	Automatically given the same name as its key variable	**Origin**
Composite	the values of more than one variable concatenated to form a single value	Must be given a name that is not the same as any variable or existing index	**DteFlt**

60

Index Terminology

Index options include the following:

UNIQUE Values of the key variable(s) must be unique. The option prevents an observation with a duplicate value for the key variable(s) from being added to the data set.

Flight ID	RouteID	Origin	Dest	DestType	FltDate	. . .
IA10800	0000108	AKL	WLG	International	30DEC2005	. . .
IA10801	0000108	AKL	WLG	International	30DEC2005	. . .
IA10802	0000108	AKL	WLG	International	30DEC2005	. . .
IA10803	0000108	AKL	WLG	International	30DEC2005	. . .

The concatenation of the values for **`FlightID`** and **`FltDate`** forms a unique identifier for a row of data.

61

In an existing data set, if the variable(s) on which you attempt to create a unique index has duplicate values, the index is not created and an error message is written to the SAS log.

Creating Indexes

- To create indexes at the same time that you create a data set, use the INDEX= data set option on the output data set.
- To create or delete indexes in existing data sets, use the one of the following:
  - DATASETS procedure
  - SQL procedure

62

Indexes can also be created using the SAS Management Console that is part of SAS Business Intelligence Architecture.

Creating Indexes

When creating the index, you can do the following:

- designate the key variable(s)
- select a valid SAS name for the index (composite index only)
- specify the UNIQUE index option if appropriate

A data set can have these features:

- multiple simple and composite indexes
- character and numeric key variables

63

For increased efficiency, use the INDEX= option to create indexes when you initially create a SAS data set.

Creating an Index with the DATA Step

c02s3d1

```
options msglevel=i;

data ia.Sales(index = (Origin
            DteFlt = (FltDate FlightID)/unique));
   infile 'sales.dat'  lrecl=162;  * PC and Unix;
   *infile '.prog3.rawdata(sales)' lrecl=162;   * mainframe ;
   input FlightID $7.  RouteID $7.  Origin $3.  Dest $3.
         DestType $13.  FltDate date9. Cap1st 8. CapBus 8.
         CapEcon 8.  CapPassTotal 8.  CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8. NumPassTotal 8. Rev1st comma8.
         RevBus comma8.  RevEcon comma8. CargoRev comma8.
         RevTotal comma8.  CargoWeight comma8.;
   format FltDate date9.;
run;
```

Log

```
679  options msglevel=i;
680
681  data ia.Sales(index = (Origin
682              DteFlt = (FltDate FlightID)/unique));
683     infile 'sales.dat'  lrecl=162;  * PC and Unix;
684     *infile '.prog3.rawdata(sales)' lrecl=162;   * mainframe ;
685     input FlightID $7.  RouteID $7.  Origin $3.  Dest $3.
686           DestType $13.  FltDate date9. Cap1st 8. CapBus 8.
687           CapEcon 8.  CapPassTotal 8.  CapCargo 8.  Num1st 8.
688           NumBus 8. NumEcon 8. NumPassTotal 8. Rev1st comma8.
689           RevBus comma8.  RevEcon comma8. CargoRev comma8.
690           RevTotal comma8.  CargoWeight comma8.;
691     format FltDate date9.;
692  run;

NOTE: The infile 'C:\workshop\winsas\prog3\sales.dat' is:
      File Name=C:\workshop\winsas\prog3\sales.dat,
      RECFM=V,LRECL=162

NOTE: 329264 records were read from the infile 'C:\workshop\winsas\prog3\sales.dat'
      The minimum record length was 162.
      The maximum record length was 162.
NOTE: The data set IA.SALES has 329264 observations and 21 variables.
NOTE: Composite index DteFlt has been defined.
NOTE: Simple index Origin has been defined.
NOTE: DATA statement used (Total process time):
      real time           10.76 seconds
      cpu time            3.85 seconds
```

✎ The external file **sales** used for demonstrations and exercises contains fewer observations than the external file **sales** used for the course notes.

Creating Indexes with the DATA Step

When creating a data set in a DATA step, use the INDEX= data set option to create an index at the same time.

General form of the INDEX= data set option:

```
DATA SAS-data-file-name(INDEX =
          (index-specification-1</option>
     …<index-specification-n</option>>));
```

65

The following are conditions for an *index-specification*

simple index	is the name of the key variable.
composite index	is *index-name* = (list of key variables).

You can specify the UNIQUE option with the INDEX= data set option.

The INDEX= data set option can also be used in procedures with OUT= options and also with ODS OUTPUT statements.

Viewing Information about Indexes

To display information in the log concerning index creation or index usage, change the value of the MSGLEVEL= system option from its default value of N to I.

General form of the MSGLEVEL= system option:

OPTIONS MSGLEVEL = N | I;

66

N only prints notes, warnings, and error messages. This is the default.

I also prints informational or INFO notes that pertain to index creation and usage, merge processing, and host sort utilities.

Managing Indexes with PROC DATASETS

c02s3d2

```
proc datasets library = ia nolist;
   modify Sales;
      index delete Origin;
      index delete DteFlt;

      index create Origin;
      index create DteFlt = (FltDate FlightID) / unique;
quit;
```

The NOLIST option prevents a list of library members from being printed in the log.

Log

```
703  options msglevel = i;
704
705  proc datasets library = ia nolist;
706     modify Sales;
707        index delete Origin;
NOTE: Index Origin deleted.
708        index delete DteFlt;
NOTE: All indexes defined on IA.SALESDATA.DATA have been deleted.
709
710        index create Origin;
NOTE: Simple index Origin has been defined.
711        index create DteFlt = (FltDate FlightID) / unique;
NOTE: Composite index DteFlt has been defined.
712  quit;

NOTE: MODIFY was successful for IA.SALES.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           0.84 seconds
      cpu time            0.80 seconds
```

Managing Indexes with PROC DATASETS

You can use the DATASETS procedure on existing data sets to create or delete indexes.

General form of the PROC DATASETS step to delete or create indexes:

PROC DATASETS LIBRARY = *libref* ;
    MODIFY *SAS-data-set-name*;
        INDEX DELETE *index-name*;
        INDEX CREATE *index-specification*
                < / *options*>;
QUIT;

68

The INDEX CREATE statement in PROC DATASETS cannot be used if the index to be created already exists.

If the index to be created already exists, you must do the following:

- Delete the existing index of the same name.
- Create the new index to avoid an error.

If you delete and create indexes in the same step, delete indexes first so that the newly created indexes can reuse the space of the deleted indexes.

You can specify the UNIQUE option on the INDEX CREATE statement.

Managing Indexes with PROC SQL

c02s3d3

```
options msglevel = n;

proc sql;
   drop index Origin
      from ia.Sales;
   drop index DteFlt
      from ia.Sales;

   create index Origin
         on ia.Sales(Origin);
   create unique index DteFlt
         on ia.Sales(FltDate,FlightID);
quit;
```

Log

```
739  options msglevel = n;
740
741  proc sql;
742     drop index Origin
743        from ia.Sales;
NOTE: Index Origin has been dropped.
744     drop index DteFlt
745        from ia.Sales;
NOTE: Index DteFlt has been dropped.
746
747     create index Origin
748           on ia.Sales(Origin);
NOTE: Simple index Origin has been defined.
749     create unique index DteFlt
750           on ia.Sales(FltDate,FlightID);
NOTE: Composite index DteFlt has been defined.
751  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.88 seconds
      cpu time            0.77 seconds
```

Managing Indexes with the SQL Procedure

You can use PROC SQL on existing data sets to create or delete indexes.

General form of the PROC SQL step to create or delete indexes:

```
PROC SQL;
   DROP INDEX index-name
         FROM table-name;
   CREATE <option> INDEX  index-name
         ON table-name(column-name-1,...
                        column-name-n);
```

70

PROC SQL cannot be used if the index to be created already exists.

If the index to be created already exists, you must do the following:

1. Drop the existing index of the same name.
2. Create the new index.

In most data processing situations, SAS maintains an index automatically.

The SQL procedure CREATE|DROP INDEX syntax is ANSI standard syntax.

You can specify the UNIQUE option in the CREATE INDEX statement.

Index Documentation

- PROC CONTENTS
- PROC DATASETS
- SAS Explorer
- SAS Management Console

71

Documenting Indexes

c02s3d4

```
proc contents data = ia.sales;
run;
```

Partial Output

```
                              The CONTENTS Procedure

Data Set Name        IA.SALES                       Observations          329264
Member Type          DATA                           Variables             21
Engine               V9                             Indexes               2
Created              Monday, March 28,              Observation Length    168
                     2005 05:55:43 PM
Last Modified        Monday, March 28,              Deleted Observations  0
                     2005 06:06:25 PM
Protection                                          Compressed            NO
Data Set Type                                       Sorted                NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)

< lines of output removed >


             Alphabetic List of Indexes and Attributes

                                # of
                     Unique    Unique
#    Index           Option    Values    Variables

1    DteFlt          YES       329264    FltDate FlightID
2    Origin                        52
```

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Exercises

1. Creating Indexes with the DATA Step

Open the program, c02ex3Start, and add the INDEX= option to create two indexes:

- a simple index **`Depart`**, based on the **`Depart`** variable
- a unique composite index **`FltDte`**, based on the **`Flight`** and **`Date`** variables

2. Deleting Indexes with the SQL Procedure

Use PROC SQL to delete the **`Depart`** index from the **`ia.schedule`** data set.

3. Creating Indexes with the DATASETS Procedure

Use PROC DATASETS to create a simple index **`Date`** based on the **`Date`** variable for the **`ia.schedule`** data set.

4. Viewing Index Information

Use PROC CONTENTS or PROC DATASETS to look at the index information.

Index Usage Possible

An index might be used when a WHERE expression references one of the following:

- a simple index key variable
- the primary key variable of a composite index

Using an index to process a WHERE expression might improve performance, and is referred to as *optimizing* the WHERE expression.

In a compound expression using the logical operator AND, only one simple index can be used.

74

Index Usage Possible

Condition	Examples

Comparison operators and the IN operator	`where FlightID eq 'IA07903';`
	`where EconomyRev < 5000;`
	`where Origin in ('LHR','CDG');`

Comparison operators with NOT	`where FlightID ne 'IA07903';` `where Origin not in ('LHR','CDG');`
Comparison operators with the colon modifier	`where Origin =:'L';`

continued...

75

There are simple indexes on the variables **FlightID**, **EconomyRev**, and **Origin**.

The colon modifier indicates a *starts with* condition. It cannot be used in the SQL procedure.

Index Usage Possible

Condition	Examples
CONTAINS operator	`where Origin contains 'L';`
Fully bounded range conditions specifying both an upper and lower limit, which includes the BETWEEN-AND operator	`where 5000 < EconomyRev < 10000;` `where EconomyRev between 5000 and 10000;`

continued...

76

There are simple indexes on the variables **EconomyRev** and **Origin**.

Index Usage Possible

Condition	Examples
Pattern-matching operator LIKE	`where Origin like 'L%';` `where Origin like 'YY_';`
IS NULL or IS MISSING operator	`where Origin is null;` `where Origin is missing;`
TRIM function	`where trim(City)='London';`

continued...

77

There are simple indexes on the variables **Origin** and **City**.

Index Usage Possible

Condition	Examples
The SUBSTR function with the conditions that the starting position = 1 and the length is less than or equal to the length of the string variable.	`where substr(City,1,2)='Ca';`

78

There is a simple index on the variable **`City`**.

General form of the SUBSTR function:

SUBSTR (*variable,position,<length>*)

When Is an Index Not Used?

An index is **not** used in the following circumstances:

- with a subsetting IF statement in a DATA step
- with particular WHERE expressions
- if SAS determines that it is more efficient to read the data sequentially

79

The conditions listed here apply to indexed Base SAS data files only. A discussion of when an index is used with Scalable Performance Data Engine data files is contained in a later chapter.

No Index Usage

SAS does not use an index when a WHERE expression references an indexed variable if the following conditions exist:

- No single index could supply all required observations.

```
where RouteID = '000035' or FlightID = '202';
```

- Any function other than TRIM or SUBSTR appears in the WHERE expression.

```
where weekday(FlightDate) = 6;
```

continued...

80

No Index Usage

- The SUBSTR function does not search a string beginning at the first position.

```
where substr(Destination,2,1) = 'F';
```

- The sounds-like operator (=*) is used.

```
where Destination =* 'lacks';
```

81

Compound Optimization

When you write a WHERE expression using all the key variables in a composite index, you can take advantage of compound optimization.

Compound optimization means that SAS can use a composite index to optimize some WHERE expressions that involve multiple variables.

```
where FlightID = 'IA10703' and
      FltDate = '03DEC2004'd;
```

82

There is a composite index, **DteFlt**, on the variables **FlightID** and **FltDate**.

Compound Optimization

For compound optimization to occur, all of the following must be true:

- At least the first two key variables in the composite index must be used in the WHERE conditions.
- The conditions are connected using the AND operator.
- At least one condition must be the EQ or IN operator.

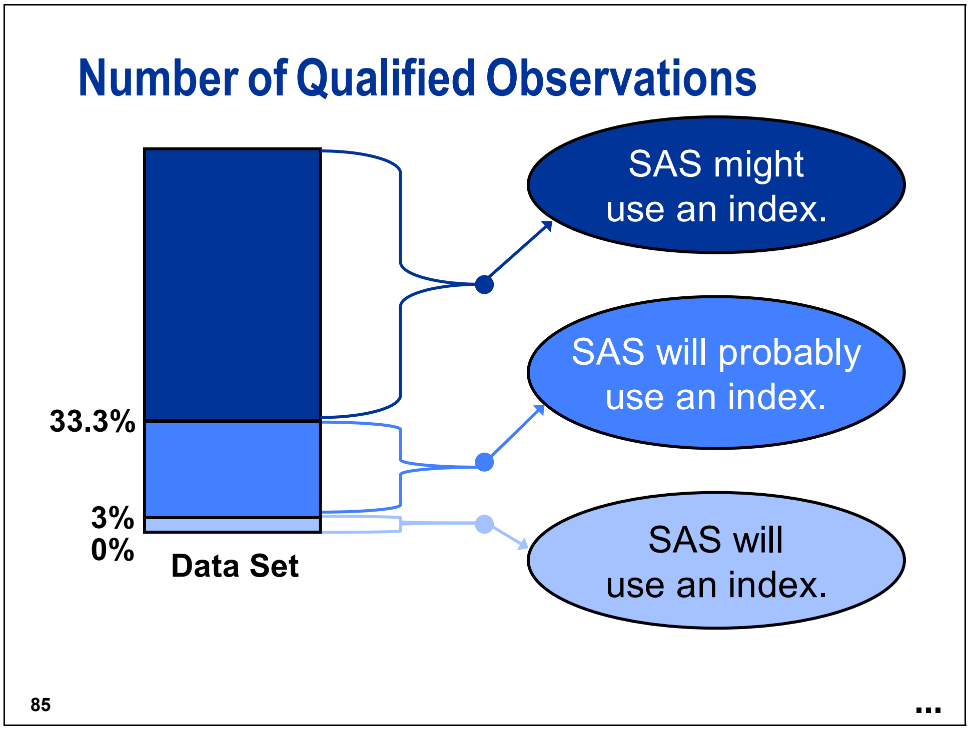
83

WHERE Expression Index Usage

To decide whether to use indexed or sequential access, SAS must do the following:

- determine whether the WHERE expression can be satisfied by an existing index
- select the best index if several indexes are available
- estimate the number of observations that qualify
- compare probable resource usage for both methods

84



To determine whether it is more efficient to satisfy the WHERE expression by using the index or reading the data sequentially, SAS uses these guidelines:

- If only a few observations are qualified, it is more efficient to use the index than to do a sequential search of the entire data file.
- If most or all of the observations qualify, then it is more efficient to read the data file sequentially.

Number of Qualified Observations

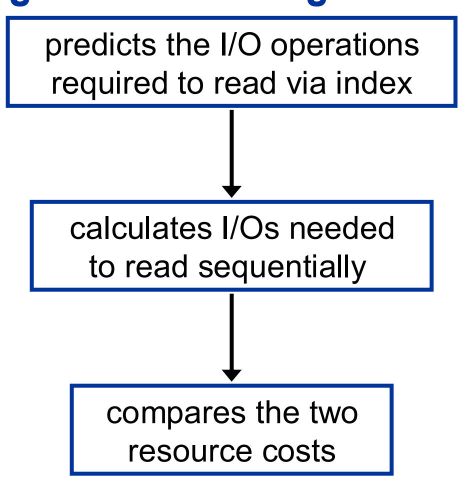
To help SAS estimate the number of observations that would be selected by a WHERE expression, each index stores 21 statistics called cumulative percentiles or centiles.

Centiles provide information about the distribution of values in an index.

86

For information on updating and viewing the centile information, see the centiles information in the SAS documentation for the CONTENTS and DATASETS procedures.

Comparing Resource Usage



87

...

Factors Affecting I/O

- Size of the subset relative to the size of the data file
- Order of data
- Page size of the data file
- Number of buffers allocated
- Cost to uncompress a compressed file for a sequential read

88

Data Order

Sort order can affect the number of I/O operations required for indexed access.

Obs	Flight ID	RouteID	Origin	. .
450	IA10803	0000108	AKL .	. .
451	IA10804	0000108	AKL .	. .
452	IA10805	0000108	AKL .	. .
.	.	.	.	
.	.	.	.	
898	IA10800	0000108	AKL .	. .
899	IA10801	0000108	AKL .	. .
900	IA10802	0000108	AKL .	. .
901	IA10803	0000108	AKL .	. .
902	IA10804	0000108	AKL .	. .
903	IA10805	0000108	AKL .	. .
.	.	.	.	
.	.	.	.	
1350	IA10800	0000108	AKL .	. .
1351	IA10801	0000108	AKL .	. .
1352	IA10802	0000108	AKL .	. .
1353	IA10803	0000108	AKL .	. .

unsorted data set

Obs	Flight ID	RouteID	Origin	. .
1	IA10800	0000108	AKL .	. .
2	IA10801	0000108	AKL .	. .
3	IA10802	0000108	AKL .	. .
4	IA10803	0000108	AKL .	. .
Obs	Flight ID	RouteID	Origin	. .
4367	IA10804	0000108	AKL .	. .
4368	IA10805	0000108	AKL .	. .
4369	IA07000	0000070	AMS .	. .
4370	IA07001	0000070	AMS .	. .
4371	IA07002	0000070	AMS .	. .
4372	IA07003	0000070	AMS .	. .

sorted data set

89

If the data set is sorted on the indexed variable(s), the qualified observations are adjacent to each other. Fewer pages must be read into the input buffers.

Controlling WHERE Processing Index Usage

You can control index usage for WHERE processing with these data set options:

- **IDXWHERE=YES | NO**

  overrides the software's decision regarding whether to use an index.

- **IDXNAME=*index-name***

  directs SAS to use a specific index.

90

IDXWHERE = YES | NO

YES SAS uses the best available index to process the WHERE expression, even if SAS estimates that processing sequentially is faster.

NO SAS processes the data sequentially, even if SAS estimates that processing with an index is faster.

You cannot use IDXWHERE= to override the use of an index to process a BY statement.

Using the IDXWHERE= Option

Suppose that the variable **`Country`** in the data set **`ia.freqflyers`** has the value **`'USA'`** in 71% of the observations.

To insure that SAS does not use an index when printing the data for **`Country = 'USA'`**, use the following code:

```
options msglevel = i;
proc print data = ia.freqflyers
          (idxwhere = no);
   where Country = 'USA';
run;
```

91

c02s3d5

Using the IDXWHERE= Option

Partial Log

```
18   proc print data = ia.freqflyers
19              (idxwhere = no);
20      where Country = 'USA';
INFO: Data set option (IDXWHERE=NO) forced a sequential pass
of the data rather than use of an index for where-clause
processing.
21   run;

NOTE: There were 65935 observations read from the data set
IA.FREQFLYERS.
      WHERE Country='USA';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           4.86 seconds
      cpu time            0.89 seconds
```

92

Guidelines for Indexing

Suggested guidelines for creating indexes:

- Minimize the number of indexes to reduce disk storage and update costs. Create indexes only on variables that are often used in queries or BY-group processing (when data cannot be sorted).
- Do not create an index if the data file page count is less than three pages. It is faster to access the data sequentially.
- Consider the cost of an index for a data file that is frequently changed.
- Create indexes on variables that are discriminating. These variables precisely identify observations that satisfy WHERE expressions.

continued...

93

A variable such as **Gender** is not discriminating. A *discriminating variable* is one that enables you to break the data into many small groups or subsets.

Guidelines for Indexing

- When you create a composite index, make the first key variable the most discriminating.
- Create an index when you intend to retrieve a small subset of observations from a large data file.
- To reduce the number of I/Os performed when you create an index, first sort the data by the key variable. Then, to improve performance, maintain the data file in sorted order by the key variable.
- Consider how often your applications use an index. An index must be used often in order to compensate for the resources used in creating and maintaining it.
- When you create an index to process a WHERE expression, do not try to create one index that is used to satisfy every conceivable query.

94

Index Trade-offs

BENEFITS

- Fast access to a small subset of observations
- Values returned in sorted order
- Can enforce uniqueness

COSTS

- Extra CPU cycles and I/O operations to create and maintain an index
- Increased CPU to read the data
- Extra disk space to store the index file
- Extra memory to load index pages and SAS C code to use the index

95

Maintaining Indexes

Data Management Tasks	Index Action Taken
Copy the data set with the COPY procedure or the DATASETS procedure.	Index file constructed for new data file
Move the data set with the MOVE option in the COPY procedure.	Index file deleted from IN= library; rebuilt in OUT= library
Copy the data set with drag-and-drop in SAS Explorer.	Index file constructed for new file

96

Maintaining Indexes

Data Management Tasks	Index Action Taken
Rename data set.	Index file renamed
Rename variable.	Variable renamed to new name in index file
Add observations.	Value/identifier pairs added
Delete observations.	Value/identifier pairs deleted; space recovered for re-use
Update observations.	Value/identifier pairs updated if values change

97

Indexes are maintained by updates in place, such as using the Viewtable window to update, add, or delete observations, and the APPEND or SQL procedures to append data. Using the Explorer window or the DATASETS procedure maintains indexes when data sets or variables are renamed. However, recreating a data set with the SET, MERGE, or UPDATE statements does **not** automatically maintain indexes.

Maintaining Indexes

Data Management Tasks	Index Action Taken

Delete a data set.
`proc datasets lib = work;`
`   delete a;`
`run;`
	Index file deleted

Rebuild a data set with a DATA step.
`data a;`
`   set a;`
`run;`
	Index file deleted

Sort the data set in place with the FORCE option in the SORT procedure.
`proc sort data = a force;`
`   by var;`
`run;`
	Index file deleted

98

If you use the UPLOAD procedure or the DOWNLOAD procedure, the index is re-created by default when you upload or download a single data set and omit the OUT= option, or when you upload or download a SAS data library. Use the INDEX=NO data set option to upload or download without re-creating the index.

Index re-created:

```
proc upload data = schedule;
run;
```

Index not re-created:

```
proc download data = Sales(index = no);
run;
```

Exercises

5. Using an Index

Open the program, c02ex7Start, and submit it. Consult the log and answer the questions following the program code listed here.

c02ex7Start

```
options msglevel=I obs = 500;

*** Example 1;

data rdu;
   set ia.Sales;
   if Origin = 'RDU';
run;

*** Example 2;

proc print data=ia.Sales;
   where Origin = 'RDU' or FltDate = '01dec2004'd;
run;

*** Example 3;

proc print data=ia.Sales;
   where Origin ne 'RDU';
run;

*** Example 4;

proc print data=ia.Sales;
   where Origin='ATH';
run;

**** Example 5;

proc print data=ia.Sales;
   where FltDate='24mar2005'd;
run;

*****Example 6;

data SalesCopy;
   set ia.Sales;
run;
```

Questions:

a. Does Example 1 use an index? Why or why not?

b. Does Example 2 use an index? Why or why not?

c. Does Example 3 use an index? Why or why not?

d. Does Example 4 use an index? Why or why not?

e. Does Example 5 use an index? Why or why not?

f. In Example 6, does the data set **`SalesCopy`** have an index?

2.4 Solutions to Exercises

1. Generating a Random Sample with Replacement

Generate a random sample **with** replacement of 50 employees from **ia.salcomps** to analyze their current salaries.

If the current salary is over $30,000, then place the employee's information in the **work.over30** SAS data set.

If the current salary is $30,000 or less, then place the employee's information in the **work.ltoreq30** SAS data set.

```
data over30 ltoreq30;
   SampSize = 50;
   do i = 1 to SampSize;
      PickIt = ceil(ranuni(0)*TotObs);
      set ia.salcomps point = PickIt nobs = TotObs;
      if Salary > 30000 then output over30;
      else output ltoreq30;
   end;
   stop;
run;
```

2. Generating a Random Sample without Replacement (Optional)

Generate a random sample **without** replacement of ten flights from **ia.cap2000**.

DATA Step Solution:

```
data work.CapSample(drop = ObsLeft SampSize);
   SampSize = 10;
   ObsLeft = TotObs;
   do while(SampSize > 0 and ObsLeft > 0);
      PickIt + 1;
      if ranuni(0) < SampSize/ObsLeft then
         do;
            set ia.cap2000 point = PickIt
                           nobs = TotObs;
            output;
            SampSize = SampSize - 1;
         end;
      ObsLeft = ObsLeft - 1;
   end;
   stop;
run;
```

SURVEYSELECT Procedure Solution:

```
proc surveyselect data=ia.cap2000
      method=srs n=10
      out=CapSample;
run;
```

3. Creating Indexes with the DATA Step

Open the program, c02ex3Start, and add the INDEX= option to create two indexes:

- a simple index **Depart**, based on the **Depart** variable
- a unique composite index **FltDte**, based on the **Flight** and **Date** variables

```
data ia.schedule(index = (Depart
                          FltDte = (Flight Date)/unique));
   infile 'schedule.dat';  *PC/Unix;
   *infile '.prog3.rawdata(schedule)';  *z/OS;
   input Flight $7. Depart time5. Date date9.;
   format Depart time5. Date date9.;
run;
```

4. Deleting Indexes with the SQL Procedure

Use PROC SQL to delete the **Depart** index from the **ia.schedule** data set.

```
proc sql;
   drop index Depart
      from ia.schedule;
quit;
```

5. Creating Indexes with the DATASETS Procedure

Use PROC DATASETS to create a simple index **Date** based on the **Date** variable for the **ia.schedule** data set.

```
proc datasets library = ia nolist;
   modify schedule;
     index create Date;
quit;
```

6. Viewing Index Information

Use PROC CONTENTS to look at the index information.

```
proc contents data = ia.schedule;
run;
```

7. Using Indexes

Open the program, c02ex7Start, and submit it. Consult the log and answer the questions following the program code listed here.

Questions:

a. Does Example 1 use an index? Why or why not?

   No, Example 1 does not use an index because the example uses a subsetting IF statement instead of a WHERE statement.

b. Does Example 2 use an index? Why or why not?

   No, Example 2 does not use an index because the WHERE statement uses the OR operator.

c. Does Example 3 use an index? Why or why not?

No, Example 3 does not use an index because the subset is too large for an index to be appropriate.

d. Does Example 4 use an index? Why or why not?

Yes, Example 4 uses an index because the WHERE statement selects a small subset.

e. Does Example 5 use an index? Why or why not?

Yes, Example 5 uses an index because the WHERE statement selects a small subset. The WHERE statement is using the composite index, **`DteFlt`**, because the subset is on the primary key variable.

f. In Example 6, does the data set **`SalesCopy`** have an index?

No, the data set **`ia.sales`** maintains its index, but **`SalesCopy`** does not retain the index from **`ia.sales`**.

Chapter 3 Combining Data Horizontally

3.1 Joining Data Sets by Value

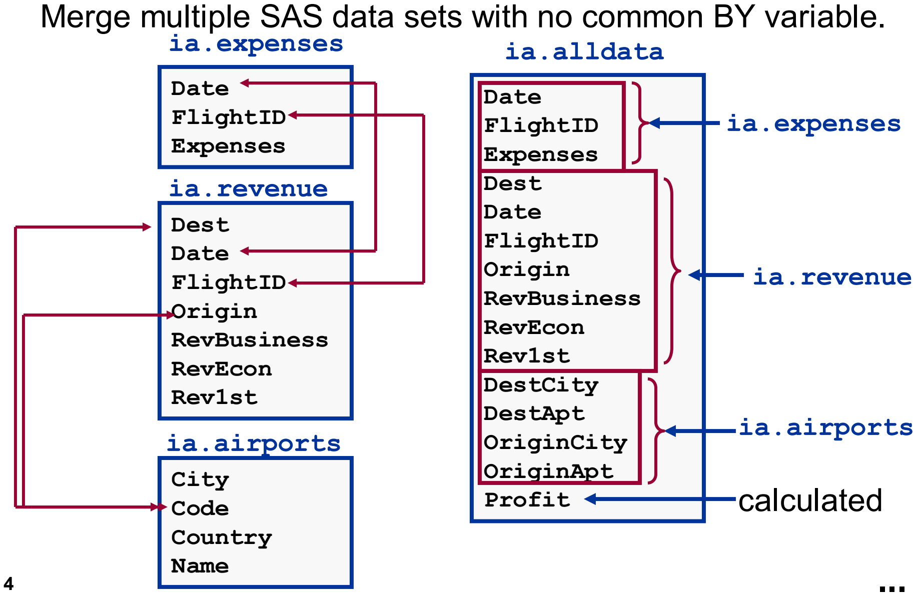
Objectives

- Use the DATA step with a MERGE statement to join more than two SAS data sets.
- Use the SQL procedure to join SAS data sets without a common variable.
- Investigate the differences between the DATA step MERGE and PROC SQL.
- Combine data conditionally.

3

Business Task

Merge multiple SAS data sets with no common BY variable.



4 ...

Methods for the Match-Merge

You can perform a match-merge of two or more SAS data sets with the following:

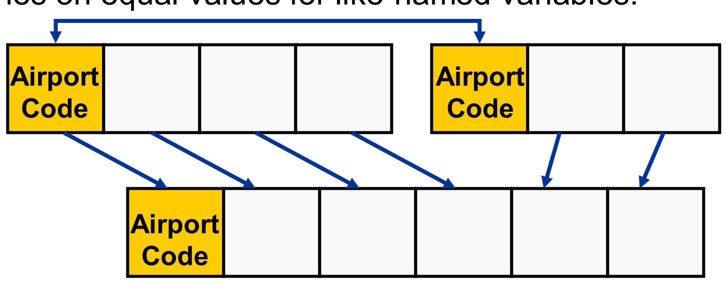
- DATA step with the MERGE statement and a BY statement
- PROC SQL join

5

DATA Step MERGE Statement

```
DATA data-set-name;
     MERGE SAS-data-sets;
     BY variables;
RUN;
```

Matches on equal values for like-named variables:



6

Using the DATA Step to Perform a Match-Merge

c03s1d1

```
proc sort data = ia.expenses out = expenses;
   by FlightID Date;
run;

proc sort data = ia.revenue out = revenue;
   by FlightID Date;
run;

data exprev;
   merge expenses(in = e) revenue(in = r);
   by FlightID Date;
   if e and r;
   Profit = sum(Rev1st, RevBusiness, RevEcon, -Expenses);
run;

proc sort data = exprev;
   by Dest;
run;

proc sort data = ia.airports out = airports;
   by Code;
run;

data destinfo;  ①
   merge exprev(in = exp)
         airports(keep = City Name Code
                  rename = (Code = Dest City = DestCity
                            Name = DestApt));
   by Dest;
   if exp;
run;

proc sort data = destinfo;
   by Origin;
run;
```

(Continued on the next page.)

```
data alldata;  ②
   merge destinfo(in = des)
             airports(keep = City Name Code
                      rename = (Code = Origin City = OriginCity
                                Name = OriginApt));
   by Origin;
   if des;
run;

proc print data = alldata(obs=5);
   title 'Result of Merging Three Data Sets';
   format Date date9.;
run;
```

① This DATA step creates the **`city`** variable for the *destination*.

② This DATA step creates the **`city`** variable for the *origin*.

Partial Output

```
                                Result of Merging Three Data Sets

      Flight                                        Rev     Rev
  Obs   ID        Date Expenses Origin Dest Rev1st Business  Econ Profit      DestCity

    1 IA03400 02DEC2005   89155   ANC   RDU   15829   28420  68688  23782 Raleigh-Durham, NC
    2 IA03400 03DEC2005   22008   ANC   RDU   15829   26460  68688  88969 Raleigh-Durham, NC
    3 IA03400 04DEC2005   71609   ANC   RDU   18707   23520  77751  48369 Raleigh-Durham, NC
    4 IA03400 05DEC2005   82454   ANC   RDU   15829   27440  64872  25687 Raleigh-Durham, NC
    5 IA03400 06DEC2005   85174   ANC   RDU   17268   27440  67257  26791 Raleigh-Durham, NC

  Obs              DestApt                   OriginCity                OriginApt

    1 Raleigh-Durham International Airport    Anchorage, AK    Anchorage International Airport
    2 Raleigh-Durham International Airport    Anchorage, AK    Anchorage International Airport
    3 Raleigh-Durham International Airport    Anchorage, AK    Anchorage International Airport
    4 Raleigh-Durham International Airport    Anchorage, AK    Anchorage International Airport
    5 Raleigh-Durham International Airport    Anchorage, AK    Anchorage International Airport
```

Advantages of DATA Step MERGE

- Multiple values can be returned.
- There is no limit to the size of the table, other than disk space.
- Multiple BY variables enable lookups that depend on more than one variable.
- Multiple data sets can be used to provide access to different tables.
- A merge enables complex business logic to be incorporated into the new data set by using DATA step processing, such as arrays and DO loops, in addition to merging features.

continued...

8

Advantages of DATA Step MERGE

- The IN= data set option and subsequent IF-THEN/ELSE logic afford comprehensive control over whether to accept, reject, or process differently depending on which data set contributed each observation.
- Observations with duplicate BY values are joined one-to-one instead of being expanded into a Cartesian product, as SQL does.

9

Disadvantages of DATA Step MERGE

- Data sets must be sorted by or indexed based on the BY variable(s).
- An exact match on the key value(s) must be found.
- The BY variable(s) must be present in all data sets.
- When more than one data set contributes variables with the same name, the values from the variable in the rightmost data set overwrite the other like-named variables, and no warning is printed.

10

Example:

Data set ONE

X	Y	Z
1	2	3

Data set TWO

X	Y	W
1	8	9

```
data three;
   merge one two;
   by x;
run;
```

Data set THREE

X	Y	Z	W
1	8	3	9

To avoid this behavior, merge on all common BY variables or use the RENAME input data set option.

The SQL Procedure

General form of the SQL procedure CREATE TABLE statement:

PROC SQL;
    CREATE TABLE *SAS-data-set* **AS**
    SELECT *column-1, column-2,… ,column-n*
        FROM *table-1, table-2,…,table-n*
        WHERE *joining criteria*
        ORDER BY *sorting criteria*;

11

Using a PROC SQL Join to Perform a Match-Merge

c03s1d2

```
proc sql;
   create table usesql as
      select revenue.FlightID, revenue.Date,
             Expenses,
             Origin, Dest,
             Rev1st, RevBusiness, RevEcon,
             sum(Rev1st, RevBusiness, RevEcon, -Expenses)
                as Profit,
             d.City as DestCity, d.Name as DestApt,  ①
             o.City as OriginCity, o.Name as OriginApt  ①
      from ia.expenses, ia.revenue,
           ia.airports as d, ia.airports as o  ①
      where expenses.FlightID = revenue.FlightID
            and expenses.Date = revenue.Date
            and d.Code = revenue.Dest  ①
            and o.Code = revenue.Origin  ①
      order by revenue.FlightID, revenue.Date;
quit;

proc print data = usesql(obs=5);
   title 'Result of Joining Three Data Sets';
   format Date date9.;
run;
```

① The data set `ia.airports` is named twice in the FROM clause so that the airport `Code` variable can be used twice in the code and the airport `City` can be extracted twice: once for the destination city and once for the city of origin. An alias is required on the duplicated data set names to distinguish which of the duplicate column names is requested.

Partial Output

```
                              Result of Joining Three Data Sets

      Flight                                              Rev     Rev
 Obs    ID         Date Expenses Origin Dest Rev1st Business  Econ Profit    DestCity

   1 IA00100 02DEC2005   58907   RDU   LHR   19200   31610  79650  71553 London, England
   2 IA00100 03DEC2005  108543   RDU   LHR   17600   25070  80181  14308 London, England
   3 IA00100 04DEC2005   21963   RDU   LHR   17600   28340  84960 108937 London, England
   4 IA00100 05DEC2005   31517   RDU   LHR   17600   32700  72216  90999 London, England
   5 IA00100 06DEC2005  105682   RDU   LHR   22400   29430  74871  21019 London, England

 Obs     DestApt             OriginCity                    OriginApt

   1 Heathrow Airport    Raleigh-Durham, NC    Raleigh-Durham International Airport
   2 Heathrow Airport    Raleigh-Durham, NC    Raleigh-Durham International Airport
   3 Heathrow Airport    Raleigh-Durham, NC    Raleigh-Durham International Airport
   4 Heathrow Airport    Raleigh-Durham, NC    Raleigh-Durham International Airport
   5 Heathrow Airport    Raleigh-Durham, NC    Raleigh-Durham International Airport
```

Advantages of PROC SQL Joins

- Multiple data sets can be joined without having common variables in all data sets.
- Data sets do not have to be sorted or indexed.
- Inequality joins can be performed.
- You can create data files (tables), views, or reports.
- PROC SQL follows ANSI standard language definitions, so that you can use knowledge gained from other languages.

13

Disadvantages of PROC SQL Joins

- The maximum number of tables that can be joined at one time is 32.
- PROC SQL might require more resources than the DATA step with the MERGE statement for simple joins.
- Complex business logic is difficult to incorporate into the join.
- Duplicate BY values are combined into a Cartesian product, which can produce an extremely large output data set.

14

Comparison Programs

The following programs are used to generate the results for the next four result sets.

```
data three;
   merge one two;
   by x;
run;
```

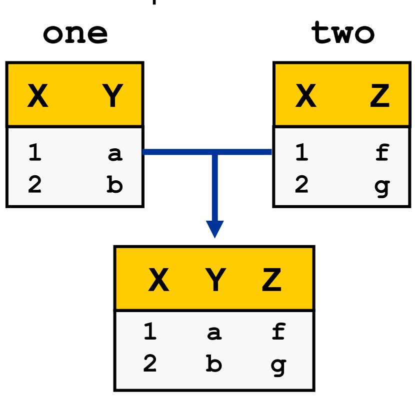
```
proc sql;
   select one.x, one.y, two.z
      from one, two
      where one.x = two.x;
quit;
```

15

✎ The DATA step and SQL procedure code remain constant. The data values change in the following examples.

MERGE and SQL Join Comparison

ONE-TO-ONE matches produce identical results:



16

The X values are unique in both data sets **one** and **two**.

MERGE and SQL Join Comparison

ONE-TO-MANY matches produce identical results:

one

X	Y
1	a
2	b

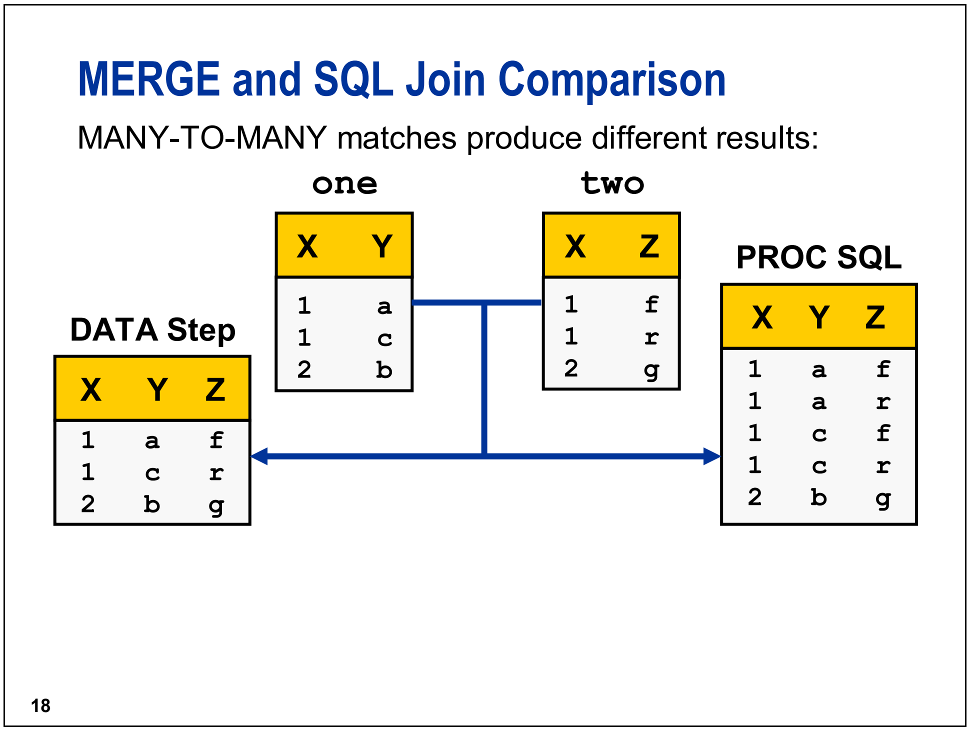
two

X	Z
1	f
1	r
2	g

X	Y	Z
1	a	f
1	a	r
2	b	g

17

The X values are unique in **one** but not in **two**.



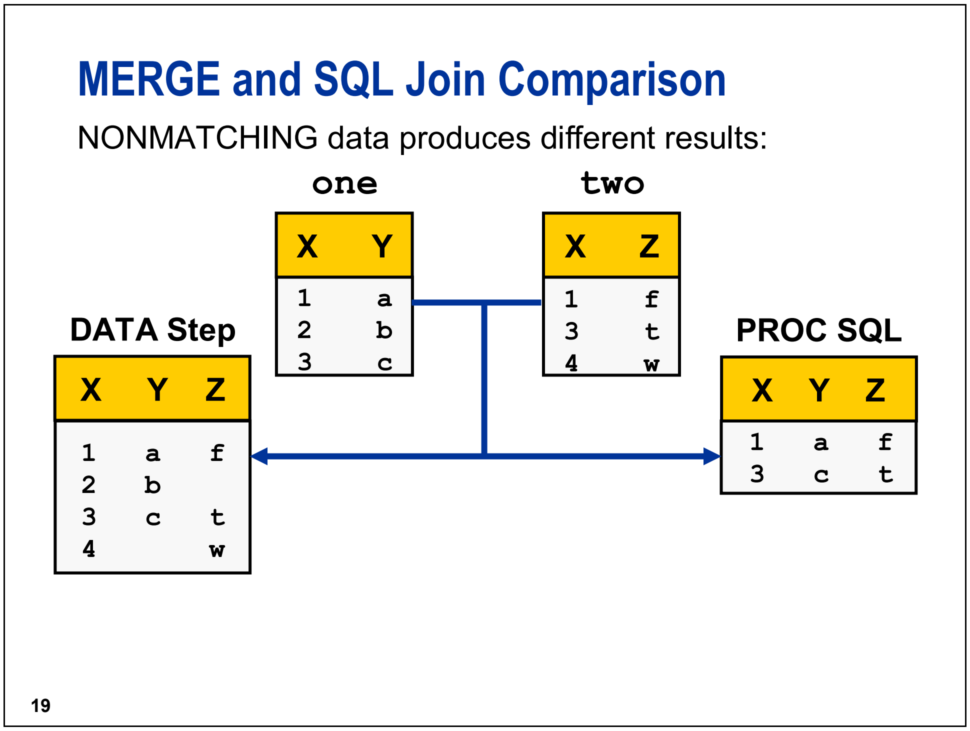
The X values in data sets **one** and **two** are not unique.

Many-to-many joins are problematic. The question is not efficiency of the technique; rather, the question is which output do you want? Do you want two or four observations for a 2-to-2 match?

Reference Information

The following DATA step creates a Cartesian product.

```
data three(drop = temp);
   set one;
   do I = 1 to totobs;
      set two(rename = (x = temp))
              nobs=totobs point = i;
      if x = temp then output;
   end;
run;
```



Reference Information

The following SQL step produces results that are identical to those of the DATA step when there is non-matching data.

```
proc sql;
  select coalesce(one.x, two.x) as x, y, z
     from one full join two
     on one.x = two.x;
quit;
```

.

MERGE and SQL Join Comparison

How does the DATA step perform a match-merge?

one

X	Y
1	a
1	d
3	c

two

X	Z
1	f
1	r
3	t
4	w

PDV

X	Y	Z

20 ...

The DATA step MERGE statement processes sequentially, top to bottom, by default.

MERGE and SQL Join Comparison

How does the DATA step perform a match-merge?

one

X	Y
1	a
1	d
3	c

two

X	Z
1	f
1	r
3	t
4	w

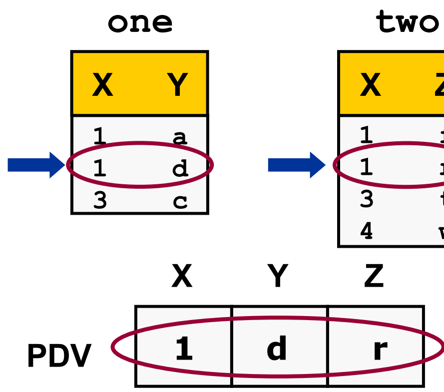
PDV

X	Y	Z
1	a	f

21 ...

MERGE and SQL Join Comparison

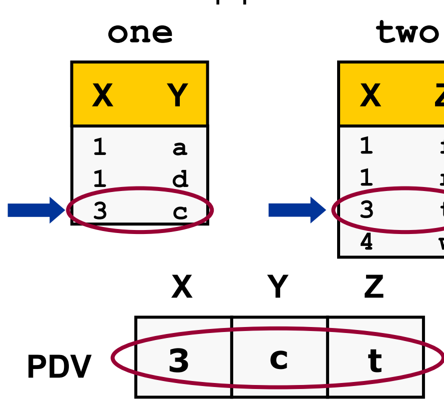
How does the DATA step perform a match-merge?



22 ...

MERGE and SQL Join Comparison

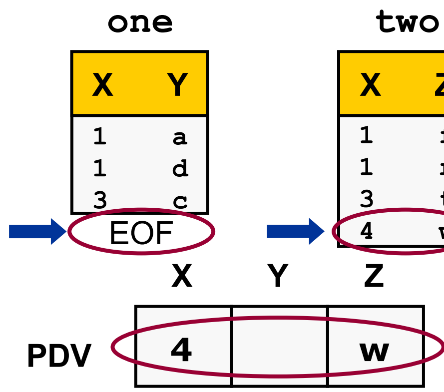
How does the DATA step perform a match-merge?



23 ...

MERGE and SQL Join Comparison

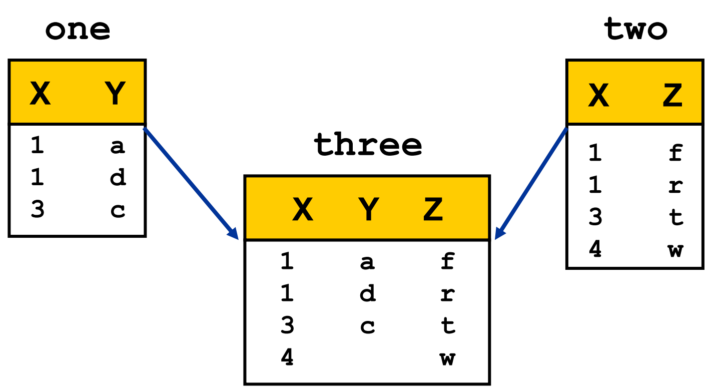
How does the DATA step perform a match-merge?



24

MERGE and SQL Join Comparison

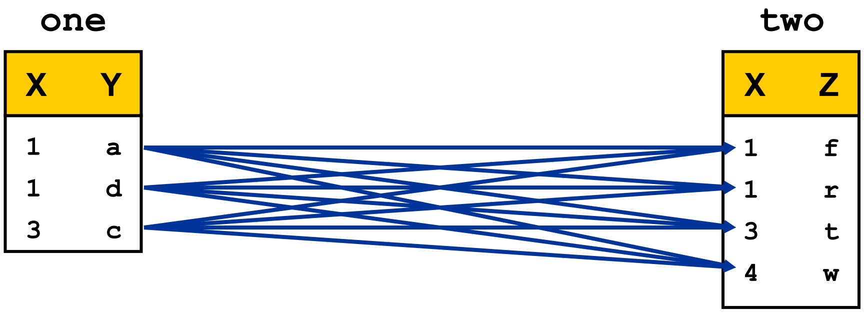
Both the matches and the non-matches on X remain.



25

MERGE and SQL Join Comparison

How does PROC SQL perform a join?



Without a WHERE clause

26 ...

PROC SQL processes by creating a Cartesian product.

MERGE and SQL Join Comparison

How does PROC SQL perform a join?

one

X	Y
1	a
1	d
3	c

X	Y	X	Z
1	a	1	f
1	a	1	r
1	a	3	t
1	a	4	w
1	d	1	f
1	d	1	r
1	d	3	t
1	d	4	w
3	c	1	f
3	c	1	r
3	c	3	t
3	c	4	w

two

X	Z
1	f
1	r
3	t
4	w

Without a WHERE clause

27

Conceptually, PROC SQL creates the result set pictured above. There are optimization routines that make the process more efficient.

MERGE and SQL Join Comparison

How does PROC SQL perform a join?

one

X	Y
1	a
1	d
3	c

where one.x = two.x

X	Y	X	Z
1	a	1	f
1	a	1	r
~~1~~	~~a~~	~~3~~	~~t~~
~~1~~	~~a~~	~~4~~	~~w~~
1	d	1	f
1	d	1	r
~~1~~	~~d~~	~~3~~	~~t~~
~~1~~	~~d~~	~~4~~	~~w~~
~~3~~	~~c~~	~~1~~	~~f~~
~~3~~	~~c~~	~~1~~	~~r~~
3	c	3	t
~~3~~	~~c~~	~~4~~	~~w~~

two

X	Z
1	f
1	r
3	t
4	w

With a WHERE clause

28

The non-matches on X are eliminated.

MERGE and SQL Join Comparison

How does PROC SQL perform a join?

one

X	Y
1	a
1	d
3	c

three

X	Y	Z
1	a	f
1	a	r
1	d	f
1	d	r
3	c	t

two

X	Z
1	f
1	r
3	t
4	w

All combinations of observations
from ONE and TWO with matches on X remain.

29

Exercises

1. Joining Data Sets to Create a New Data Set

Using PROC SQL, join **`ia.employees`**, **`ia.jcodedat`**, and **`ia.newsals`** to create a data set that contains employee IDs, employee job codes, job code descriptions, current salaries, and new salaries. Print the resulting data set.

There is no variable common to all three SAS data sets. Use PROC CONTENTS, PROC DATASETS, or the SAS Explorer to determine the columns on which to join the rows.

Partial Output

EmpID	Job Code	Descript	Salary	NewSalary
E00001	FLTAT3	FLIGHT ATTENDANT GRADE 3	$25,000	$27,420.04
E00003	VICEPR	VICE PRESIDENT	$120,000	$143,789.80
E00005	GRCREW	GROUND CREW	$19,000	$20,757.68
E00008	OFFMGR	OFFICE MANAGER	$85,000	$93,811.78
E00012	MKTCLK	MARKETING CLERK	$33,000	$38,481.44
E00013	RECEPT	RECEPTIONIST	$22,000	$23,243.79
E00014	MECH02	MECHANIC GRADE 2	$19,000	$20,434.78
E00017	RESCLK	RESERVATIONS CLERK	$36,000	$36,241.64
E00018	FACMNT	FACILITIES MAINTENANCE OPERATIVE	$33,000	$35,947.80
E00022	FACCLK	FACILITIES CLERK	$27,000	$27,530.65

2. Combining Data with the DATA Step MERGE Statement

Repeat the same task using the DATA step MERGE statement to merge all three data sets. Print the resulting data set.

Partial Output

EmpID	Job Code	Descript	Salary	NewSalary
E00001	FLTAT3	FLIGHT ATTENDANT GRADE 3	$25,000	$27,420.04
E00003	VICEPR	VICE PRESIDENT	$120,000	$143,789.80
E00005	GRCREW	GROUND CREW	$19,000	$20,757.68
E00008	OFFMGR	OFFICE MANAGER	$85,000	$93,811.78
E00012	MKTCLK	MARKETING CLERK	$33,000	$38,481.44
E00013	RECEPT	RECEPTIONIST	$22,000	$23,243.79
E00014	MECH02	MECHANIC GRADE 2	$19,000	$20,434.78
E00017	RESCLK	RESERVATIONS CLERK	$36,000	$36,241.64
E00018	FACMNT	FACILITIES MAINTENANCE OPERATIVE	$33,000	$35,947.80
E00022	FACCLK	FACILITIES CLERK	$27,000	$27,530.65

The results should be identical to the previous exercise.

Conditionally Combining Data

Some combinations of data are based on a condition.

For example, the data set **ia.madrid** contains the flights from Madrid in March 2005. The revenue amounts are in dollars.

Partial Data Set

Obs	Flight ID	FltDate	Rev1st	RevBus
1	IA05900	01MAR2005	$3,445.00	.
2	IA05901	01MAR2005	$2,915.00	.
3	IA05902	01MAR2005	$2,915.00	.
4	IA05903	01MAR2005	$2,915.00	.

Obs	RevEcon	CargoRev	RevTotal
1	$8,360.00	$7,421.00	$19,226
2	$10,824.00	$5,289.00	$19,028
3	$8,448.00	$7,503.00	$18,866
4	$9,416.00	$6,601.00	$18,932

31

Conditionally Combining Data

The data set **ia.rates** has the conversion rate for converting from dollars to euros.

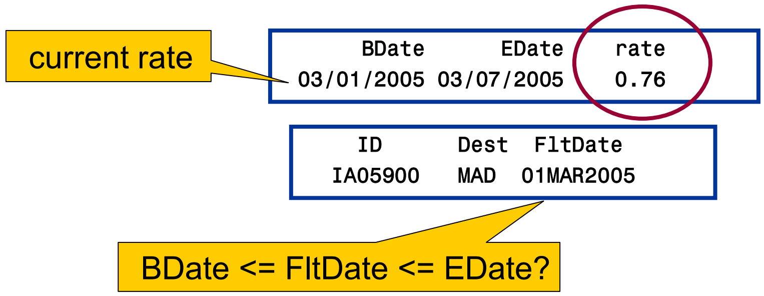
Obs	BDate	EDate	rate
1	03/01/2005	03/07/2005	0.76
2	03/08/2005	03/10/2005	0.75
3	03/11/2005	03/13/2005	0.74
4	03/14/2005	03/15/2005	0.75
5	03/16/2005	03/16/2005	0.74
6	03/17/2005	03/20/2005	0.75
7	03/21/2005	03/22/2005	0.76
8	03/23/2005	03/27/2005	0.77
9	03/28/2005	03/28/2005	0.78
10	03/29/2005	03/31/2005	0.77

32

Conditionally Combining Data

What needs to be done:

- Use the current value of **rate** when **FltDate** is between **BDate** and **EDate**.



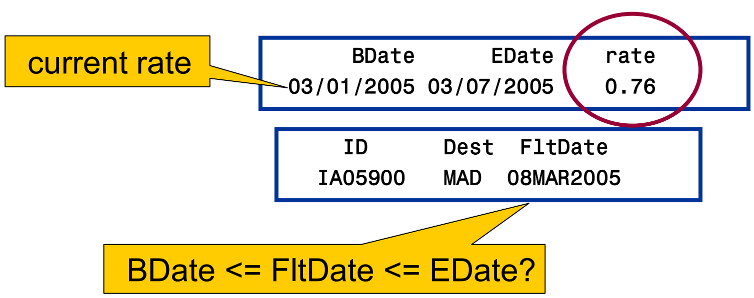
continued...

33 ...

Conditionally Combining Data

What needs to be done:

- Read a new value for **rate** when **FltDate** is not between **BDate** and **EDate**.



34 ...

Conditionally Combining Data

The MERGE statement cannot be used in this example. It can only be used to join data when one of the following conditions are met:

- The data can be joined by comparing values of a common BY value.

or

- The data can be combined by observation number. In this case, there is no BY statement in the DATA step.

36

Conditionally Combining Data

You can use multiple SET statements to combine observations from several SAS data sets.

When you use multiple SET statements, the following occurs:

- Processing stops when SAS encounters the end-of-file marker on either data set.
- The variables in the PDV are not reinitialized when a second SET statement is executed.

Example:

```
data Euros;
   set ia.madrid;
   set ia.rates;
run;
```

37

Conditionally Combining Data

```
data Euros;
   set ia.Madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le       ❶
                           EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

`ia.madrid` must be sorted by **`FltDate`**.

`ia.rates` must be sorted by **`BDate`**.

c03s1d3

38

① The DO WHILE statement executes statements in a DO loop while a condition is true. The expression is evaluated at the top of the loop. The statements in the loop never execute if the expression is initially false.

Execution

`ia.rates`

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/20	03/15/2005	0.75
03/16/20		

`ia.madrid`

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                         EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	01MAR2005	19226	.	.	.	.

40

...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20…		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                           EDate))
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

False
. <= 01MAR2005 <= .

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	01MAR2005	19226	.	.	.	.

41 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20…		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

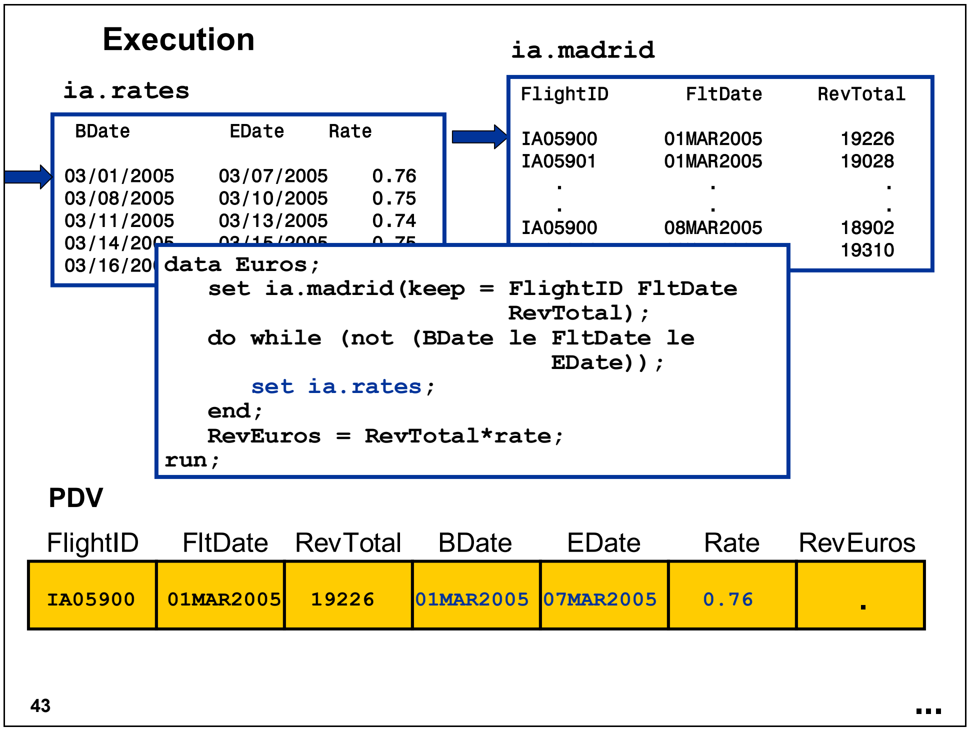
```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                      RevTotal);
   do while (not (BDate le FltDate le
                           EDate))
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

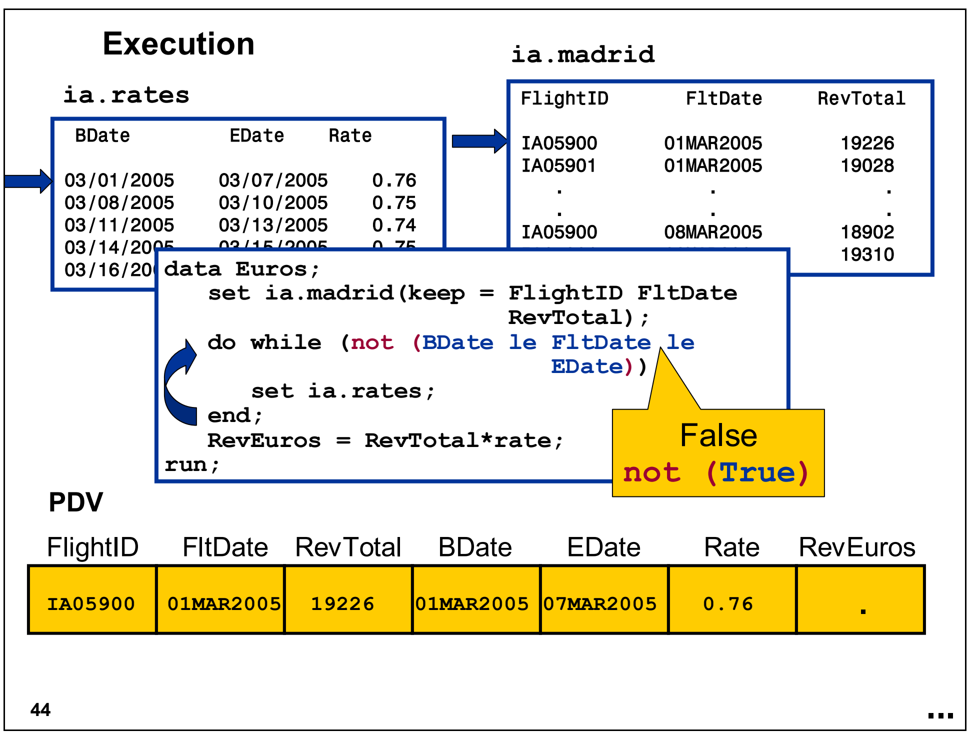
True
not (False)

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	01MAR2005	19226	.	.	.	.

42 ...


Execution
ia.rates
BDate EDate Rate
03/01/2005 03/07/2005 0.76
03/08/2005 03/10/2005 0.75
03/11/2005 03/13/2005 0.74
03/14/20
03/16/20
ia.madrid
FlightID FltDate RevTotal
IA05900 01MAR2005 19226
IA05901 01MAR2005 19028
IA05900 08MAR2005 18902
19310
data Euros;
set ia.madrid(keep = FlightID FltDate
RevTotal);
do while (not (BDate le FltDate le
EDate));
set ia.rates;
end;
RevEuros = RevTotal*rate;
run;
PDV
FlightID FltDate RevTotal BDate EDate Rate RevEuros
IA05900 01MAR2005 19226 01MAR2005 07MAR2005 0.76 .
43
...


Execution
ia.rates
BDate EDate Rate
03/01/2005 03/07/2005 0.76
03/08/2005 03/10/2005 0.75
03/11/2005 03/13/2005 0.74
03/14/20
03/16/20
ia.madrid
FlightID FltDate RevTotal
IA05900 01MAR2005 19226
IA05901 01MAR2005 19028
IA05900 08MAR2005 18902
19310
data Euros;
set ia.madrid(keep = FlightID FltDate
RevTotal);
do while (not (BDate le FltDate le
EDate));
set ia.rates;
end;
RevEuros = RevTotal*rate;
run;
False
not (True)
PDV
FlightID FltDate RevTotal BDate EDate Rate RevEuros
IA05900 01MAR2005 19226 01MAR2005 07MAR2005 0.76 .
44
...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                           EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	01MAR2005	19226	01MAR2005	07MAR2005	0.76	14611.76

45 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                           EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

Implied Output

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	01MAR2005	19226	01MAR2005	07MAR2005	0.76	14611.76

46 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/20...	...	...
03/16/20...		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                        EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	01MAR2005	19226	01MAR2005	07MAR2005	0.76	.

47 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/20...	...	...
03/16/20...		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                        EDate));
      set ia.rates;
   end;
   RevEuros = Rev...
run;
```

True
01MAR2005 <= 01MAR2005 <= 07MAR2005

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05901	01MAR2005	19028	01MAR2005	07MAR2005	0.76	.

49 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/200	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                  EDate))
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

False
not (True)

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05901	01MAR2005	19028	01MAR2005	07MAR2005	0.76	.

50 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/200	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                  EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

Implied Output

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05901	01MAR2005	19028	01MAR2005	07MAR2005	0.76	14461.28

52 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20[illegible]		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = Flight[illegible] FltDate
                 RevT[illegible]);
   [illegible]while (not [illegible]te le [illegible]te le
   [illegible]
   end;
   [illegible]
ru[illegible]
```

Continue reading **ia.madrid** until **FltDate**=08MAR2005.

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA06005	07MAR2005	[illegible]8968	[illegible]R2005	[illegible]R2005	0.[illegible]6	14415.68

54 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20[illegible]		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                 RevTotal);
   do while (not (BDate le FltDate le
                  EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	08MAR2005	18902	01MAR2005	07MAR2005	0.76	.

55 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                  EDate));
      set ia.rates;
   end;
   RevEuros
run;
```

False
01MAR2005 <= 08MAR2005 <= 07MAR52005

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	08MAR2005	18902	01MAR2005	07MAR2005	0.76	.

56 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                  EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

True
not (False)

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	08MAR2005	18902	01MAR2005	07MAR2005	0.76	.

57 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                          EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	08MAR2005	18902	08MAR2005	10MAR2005	0.75	.

58 ...

Execution

ia.rates

BDate	EDate	Rate
03/01/2005	03/07/2005	0.76
03/08/2005	03/10/2005	0.75
03/11/2005	03/13/2005	0.74
03/14/2005	03/15/2005	0.75
03/16/20		

ia.madrid

FlightID	FltDate	RevTotal
IA05900	01MAR2005	19226
IA05901	01MAR2005	19028
.	.	.
.	.	.
IA05900	08MAR2005	18902
		19310

```
data Euros;
   set ia.madrid(keep = FlightID FltDate
                        RevTotal);
   do while (not (BDate le FltDate le
                          EDate));
      set ia.rates;
   end;
   RevEuros = RevTotal*rate;
run;
```

Implied Output

PDV

FlightID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
IA05900	08MAR2005	18902	08MAR2005	10MAR2005	0.75	14176.50

60 ...

Conditional Merge Output

Obs	Flight ID	FltDate	RevTotal	BDate	EDate	Rate	RevEuros
1	IA05900	01MAR2005	$19,226	03/01/2005	03/07/2005	0.76	14611.76
2	IA05901	01MAR2005	$19,028	03/01/2005	03/07/2005	0.76	14461.28
3	IA05902	01MAR2005	$18,866	03/01/2005	03/07/2005	0.76	14338.16
4	IA05903	01MAR2005	$18,932	03/01/2005	03/07/2005	0.76	14388.32
5	IA05904	01MAR2005	$19,232	03/01/2005	03/07/2005	0.76	14616.32
6	IA05905	01MAR2005	$18,950	03/01/2005	03/07/2005	0.76	14402.00
7	IA05900	02MAR2005	$19,340	03/01/2005	03/07/2005	0.76	14698.40
8	IA05901	02MAR2005	$19,370	03/01/2005	03/07/2005	0.76	14721.20
	<Rows Removed>						
40	IA05903	07MAR2005	$19,061	03/01/2005	03/07/2005	0.76	14486.36
41	IA05904	07MAR2005	$19,322	03/01/2005	03/07/2005	0.76	14684.72
42	IA05905	07MAR2005	$19,139	03/01/2005	03/07/2005	0.76	14545.64
43	IA05900	08MAR2005	$18,902	03/08/2005	03/10/2005	0.75	14176.50
44	IA05901	08MAR2005	$19,310	03/08/2005	03/10/2005	0.75	14482.50
45	IA05902	08MAR2005	$19,589	03/08/2005	03/10/2005	0.75	14691.75
46	IA05903	08MAR2005	$19,346	03/08/2005	03/10/2005	0.75	14509.50
47	IA05904	08MAR2005	$18,998	03/08/2005	03/10/2005	0.75	14248.50
48	IA05905	08MAR2005	$19,547	03/08/2005	03/10/2005	0.75	14660.25
49	IA05900	09MAR2005	$18,896	03/08/2005	03/10/2005	0.75	14172.00

61

The secret to using multiple SET statements in this fashion is to have both data sets in order (ascending or descending) by the variables tested in the DO WHILE statement.

Using The SQL Procedure

```
proc sql;
   create table combinesql as
      select FlightID, FltDate, RevTotal,
             rate, RevTotal*rate as
              RevEuros
      from ia.Madrid, ia.rates
      where FltDate between BDate and EDate
      order by FltDate, FlightID;
quit;
```

PROC SQL can use the BETWEEN operator in the WHERE clause.

Neither data set must be sorted.

62

c03s1d4

...

Exercises

3. Combining Two Data Sets Conditionally

The data set **`ia.options`** has the number of stock options awarded to the crew employees based on the date they were hired. The hired dates for the crew are stored in the data set **`ia.crew`**. Create a data set named **`crewshrs`** that combines the two data sets. The data set **`crewshrs`** should contain only the variables **`LastName`**, **`FirstName`**, **`HireDate`**, and **`NumShares`** and should be in order by **`HireDate`**.

`ia.options`

Obs	BeginDte	EndDte	Num Shares
1	01JAN1980	31DEC1984	500
2	01JAN1985	31DEC1987	550
3	01JAN1988	31DEC1992	600
4	01Jan1993	31DEC1996	700

`ia.crew` (First 5 observations)

Obs	HireDate	LastName	FirstName	Location	Phone	EmpID	Job Code	Salary	JobCat
1	15JAN1982	CHRISTIAN	JOHN G.	LONDON	1369	E01146	FLTAT1	28000	Flight Attendant
2	23FEB1981	ELLIS	GREGORY	FRANKFURT	1595	E00364	FLTAT1	25000	Flight Attendant
3	15APR1994	EUNICE	ROBERT N.	CARY	1157	E03022	FLTAT1	23000	Flight Attendant
4	23DEC1990	FITZGERALD	JAMES V.	CARY	1168	E03511	FLTAT1	21000	Flight Attendant
5	11JUN1983	GOODWIN	CYNTHIA Q.	CARY	1752	E03510	FLTAT1	29000	Flight Attendant

`crewshrs` (First 10 observations)

Obs	LastName	FirstName	HireDate	Num Shares
1	WAKELIN	DAVE	14JAN1980	500
2	WASCHK	ROBERT	18FEB1980	500
3	GODFREY	GERALD T.	13AUG1980	500
4	WHITE	RUTH M.	25SEP1980	500
5	MEEKS	KRAIG E.	11OCT1980	500
6	WHITMEYER	ROBERTA J.	02JAN1981	500
7	WILDER	TODD C.	09JAN1981	500
8	ELLIS	GREGORY	23FEB1981	500
9	PIERCE	STEVEN W.	19AUG1981	500
10	STRAUSS	REINHARD	09OCT1981	500

3.2 Combining Summary and Detail Data

Objectives

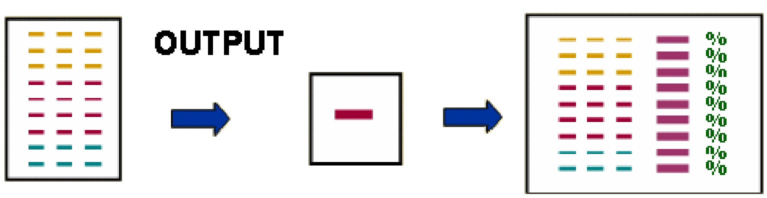
- Create an output SAS data set that contains summary statistics from PROC MEANS.
- Combine PROC MEANS summary statistics in a SAS data set with a detail SAS data set.

65

Combining Summary and Detail Data

The following are common business tasks:

- Create a summary statistic from a data set variable.
- Combine the summary information with detail rows of the original data set.
- Calculate percentages.



66 ...

Combining Summary and Detail Data

The data set **ia.empcount** has one row for every value of **JobCode**.

Partial Output

Obs	Job Code	Num Emps
1	BAGCLK	140
2	BAGSUP	18
3	CHKCLK	125
4	CHKSUP	18
5	FACCLK	124
6	FACMGR	17
7	FACMNT	60
8	FINACT	36
9	FINCLK	53
10	FINMGR	20

67

Combining Summary and Detail Data

Summarize the data to get the total number of employees at International Airlines.

TotalEmps
2070

68

Combining Summary and Detail Data

Combine the summary data with the detail data in **`ia.empcount`** to calculate the percentage of employees in each job code.

Resulting SAS Data Set

Percentage of Each Job Code

Obs	Total Emps	Job Code	Num Emps	PctEmps
1	2070	BAGCLK	140	6.76%
2	2070	BAGSUP	18	0.87%
3	2070	CHKCLK	125	6.04%
4	2070	CHKSUP	18	0.87%
5	2070	FACCLK	124	5.99%
6	2070	FACMGR	17	0.82%
7	2070	FACMNT	60	2.90%
8	2070	FINACT	36	1.74%
9	2070	FINCLK	53	2.56%
10	2070	FINMGR	20	0.97%

69

Creating a Summary Data Set

Some methods used to create a summary data set are as follows:

- the Output Delivery System (ODS)
- ➡ the SUMMARY or MEANS procedure with an OUTPUT statement
- the DATA step
- ➡ the SQL procedure

70 ...

The MEANS Procedure

For numeric variables within a SAS data set, the Means procedure computes descriptive statistics such as the following:

- mean
- minimum
- maximum
- number of non-missing values
- standard deviation

71

The default statistics generated by PROC MEANS are listed. For a complete list of statistics, please refer to the SAS documentation.

Using the OUTPUT Statement

The following program creates the summary data set.

```
proc means data = ia.empcount;
   var NumEmps;
   output out = summary sum = TotalEmps;
run;
```

PROC PRINT Output of **Summary**

Obs	_TYPE_	_FREQ_	Total Emps
1	0	42	2070

72

c03s2d1

PROC MEANS OUTPUT Statement

By default, PROC MEANS generates a report that contains the descriptive statistics.

The report can be routed to a SAS data set using an OUTPUT statement.

PROC MEANS DATA = *SAS-data-set* NOPRINT;
    OUTPUT OUT = *SAS-data-set*
        output-statistic-specification(s);

73

✎ The NOPRINT option suppresses the printing of the PROC MEANS report.

For a complete listing of PROC MEANS statements and options, see the SAS documentation.

The output data set contains variables that contain the requested statistics plus the following:

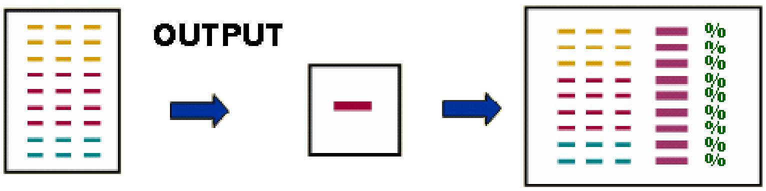
- _TYPE_ contains information about the class variables.
- _FREQ_ contains the number of observations that an output level represents.

PROC SUMMARY can also be used to generate a data set that contains summary statistics.

Combining Summary and Detail Data

After creating the summary statistic, perform the following tasks:

- Combine the summary information with the detail rows.
- Calculate the percentages.



74 ...

Combining Summary and Detail Data (Review)

You can use multiple SET statements to combine observations from several SAS data sets.

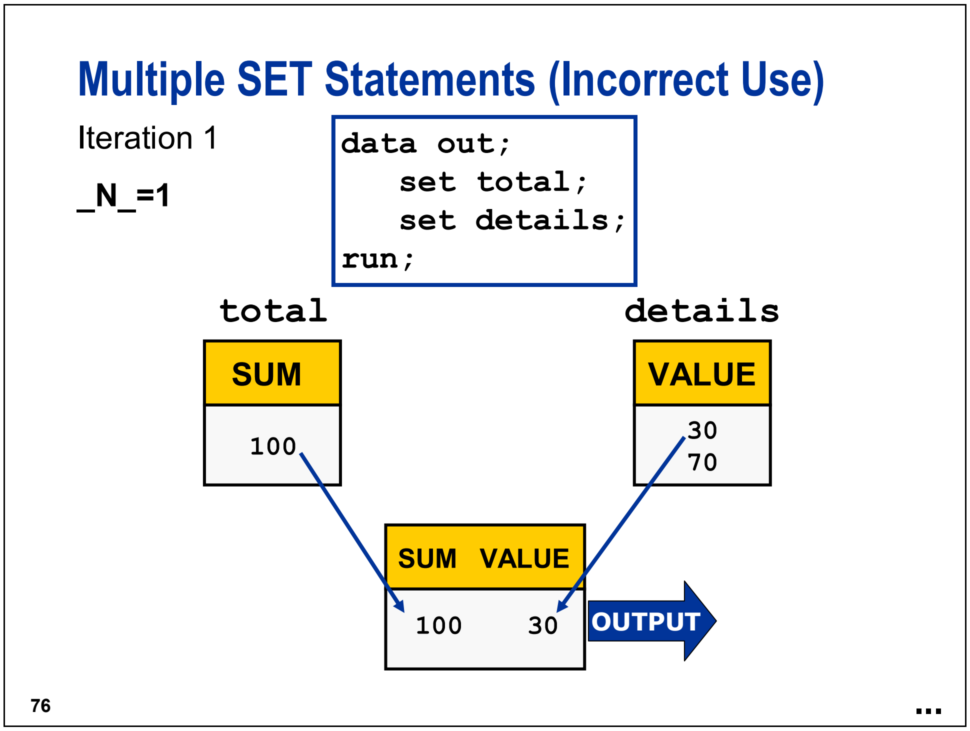
When you use multiple SET statements, the following events occur:

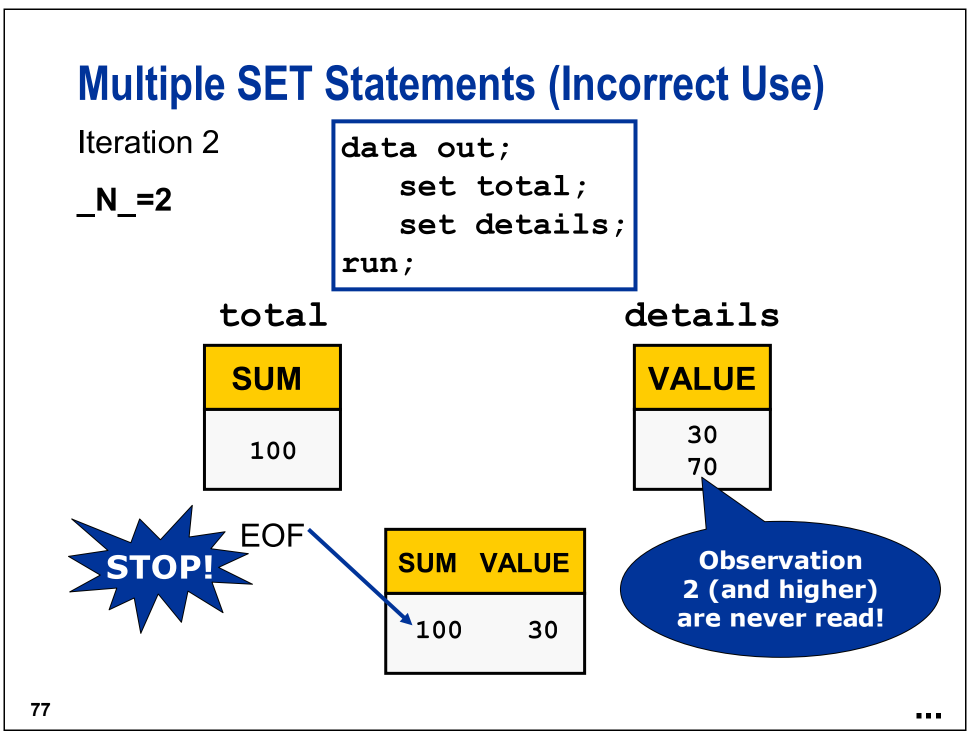
- Processing stops when SAS encounters the end-of-file marker on either data set.
- The variables in the PDV are not reinitialized when a second SET statement is executed.

Example:

```
data out;
   set total;
   set details;
run;
```

75


Multiple SET Statements (Incorrect Use)
Iteration 1
_N_=1
data out;
set total;
set details;
run;
total
details
SUM
100
VALUE
30
70
SUM VALUE
100 30
OUTPUT
76
...


Multiple SET Statements (Incorrect Use)
Iteration 2
_N_=2
data out;
set total;
set details;
run;
total
details
SUM
100
VALUE
30
70
STOP!
EOF
SUM VALUE
100 30
Observation 2 (and higher) are never read!
77
...

Using _N_

During the execution of a DATA step, the automatic variable **_N_** has the following features:

- is set to 1 initially
- is incremented by 1 as the DATA step loops past the DATA statement
- is dropped automatically from the data set that is created
- can be used in the DATA step to control when statements are executed

78

Combining Summary and Detail Data

```
data percent;
   if _n_ = 1 then
        set summary(keep = TotalEmps);      ❶
   set ia.empcount;                   ❷
   PctEmps = NumEmps / TotalEmps;
run;
```

c03s2d2

79

① The **_n_ = 1** condition causes the **summary** data set to be read only during the first iteration of the DATA step. Without it, the DATA step reaches the end of file of **summary** on the second iteration of the DATA step, and the DATA step terminates with one observation in the data set **percent1**.

② The data set **ia.empcount** is read for each iteration of the DATA step.

Compilation

summary

TotalEmps
2070

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
CHKSUP	18

```
data percent;
   if _n_ = 1 then set summary (keep = TotalEmps);
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	TotalEmps	JobCode	NumEmps	PctEmps
1	.	.	.	.

80 ...

Execution

summary

TotalEmps
2070

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
CHKSUP	18

```
data percent;
   if _n_ = 1 then set summary (keep = TotalEmps);
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

True

PDV

D _N_	TotalEmps	JobCode	NumEmps	PctEmps
1	.	.	.	.

81 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
CHKSUP	18

summary

TotalEmps
2070

```
data percent;
   if _n_ = 1 then set summary (keep = TotalEmps);
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	TotalEmps	JobCode	NumEmps	PctEmps
1	2070	.	.	.

82 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
CHKSUP	18

summary

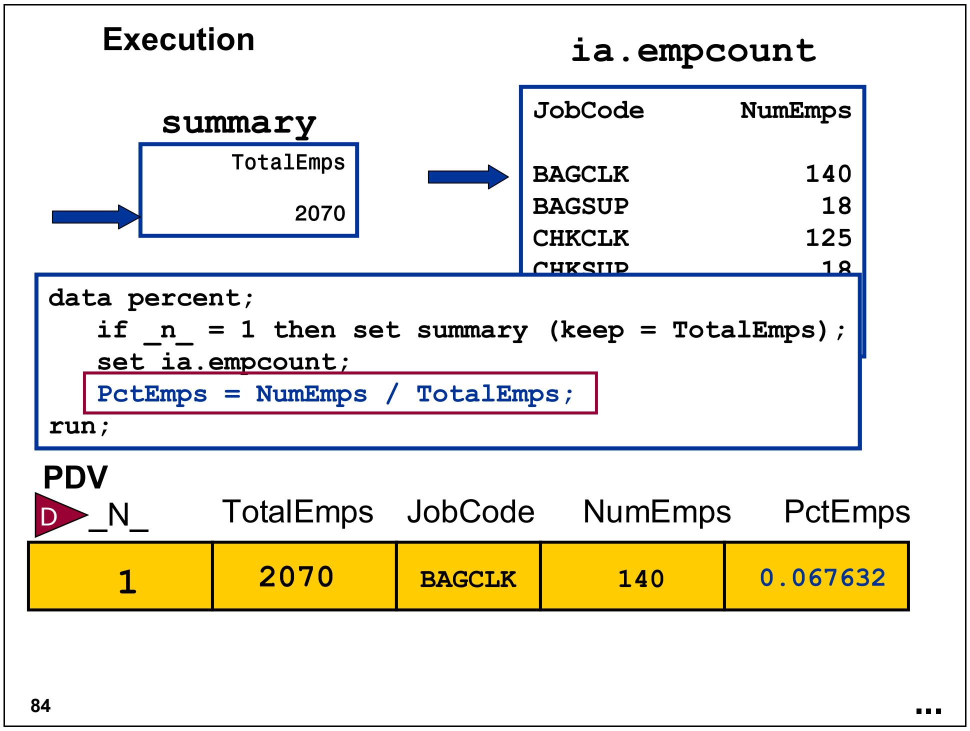
TotalEmps
2070

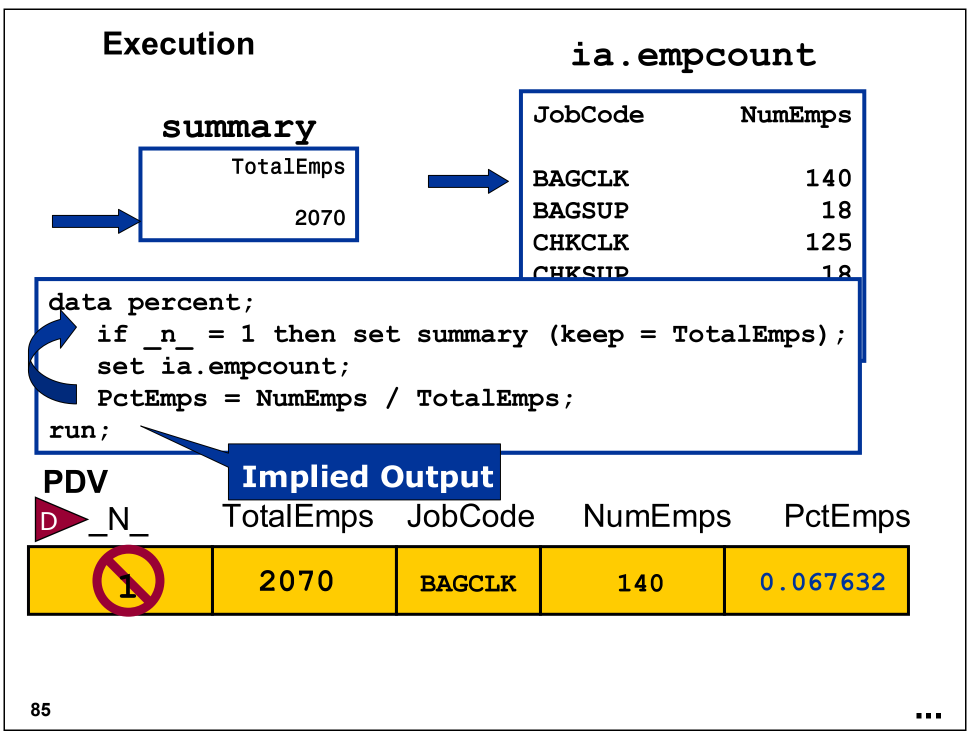
```
data percent;
   if _n_ = 1 then set summary (keep = TotalEmps);
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

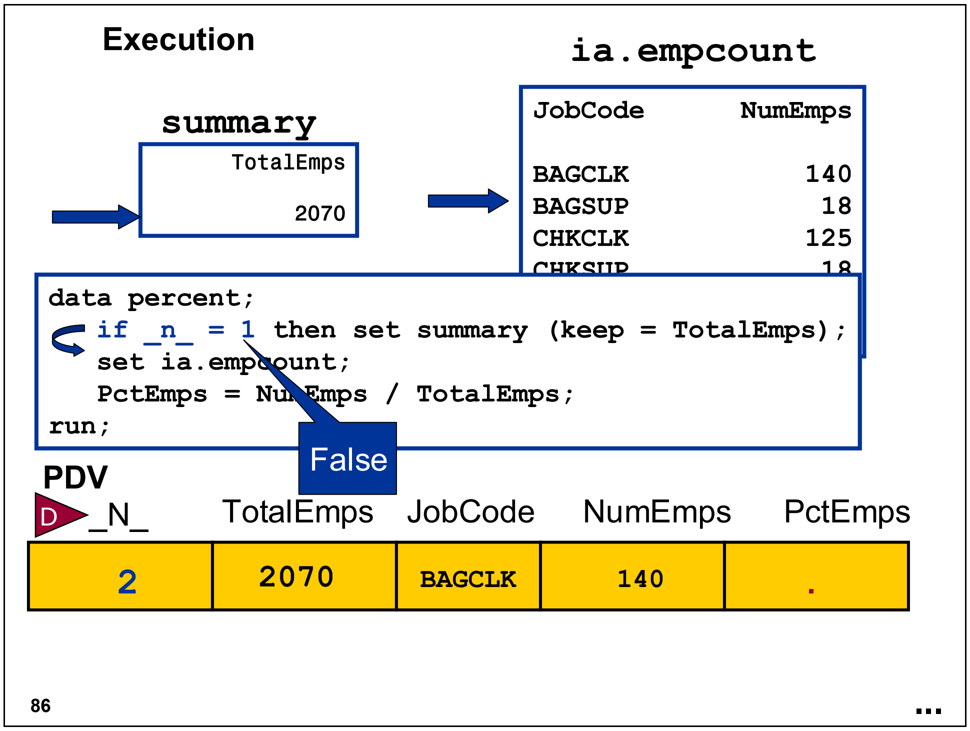
PDV

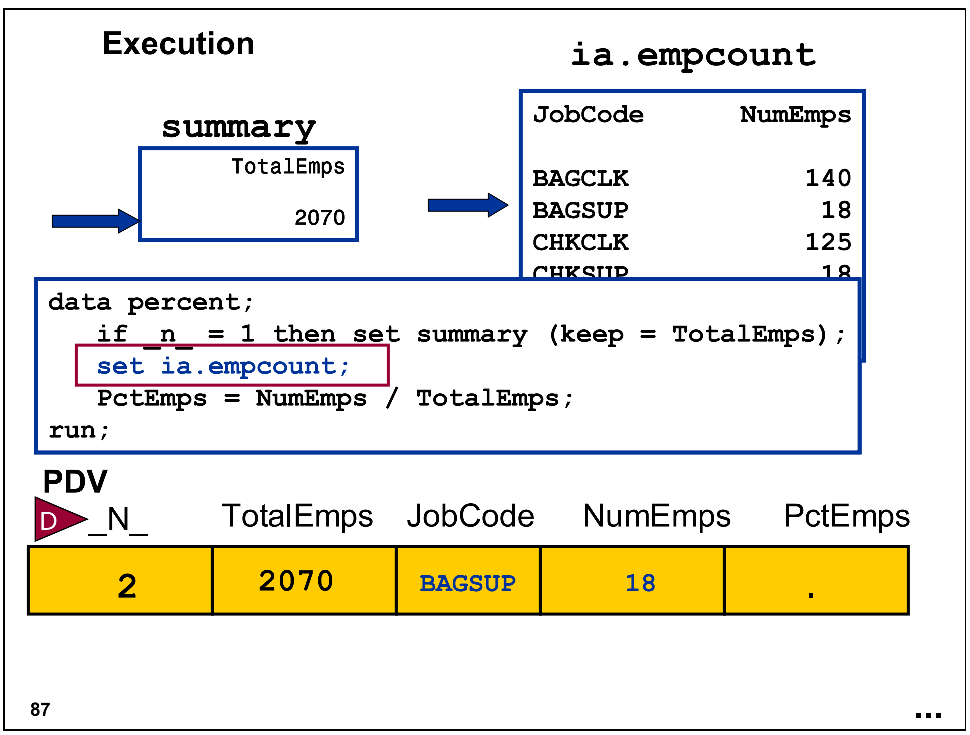
D _N_	TotalEmps	JobCode	NumEmps	PctEmps
1	2070	BAGCLK	140	.

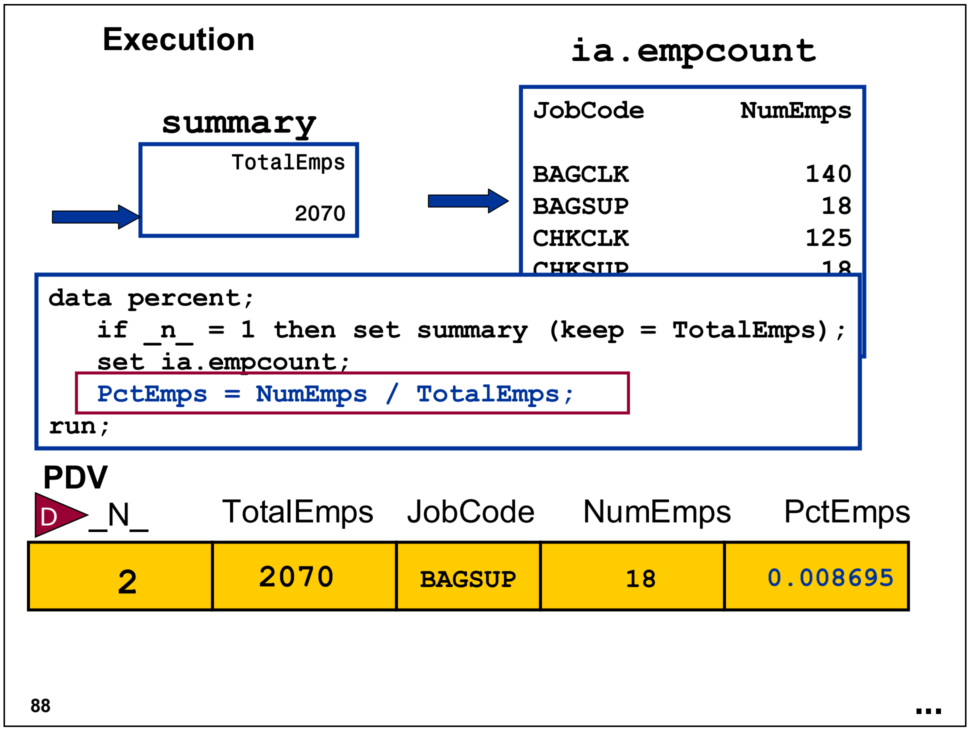
83 ...

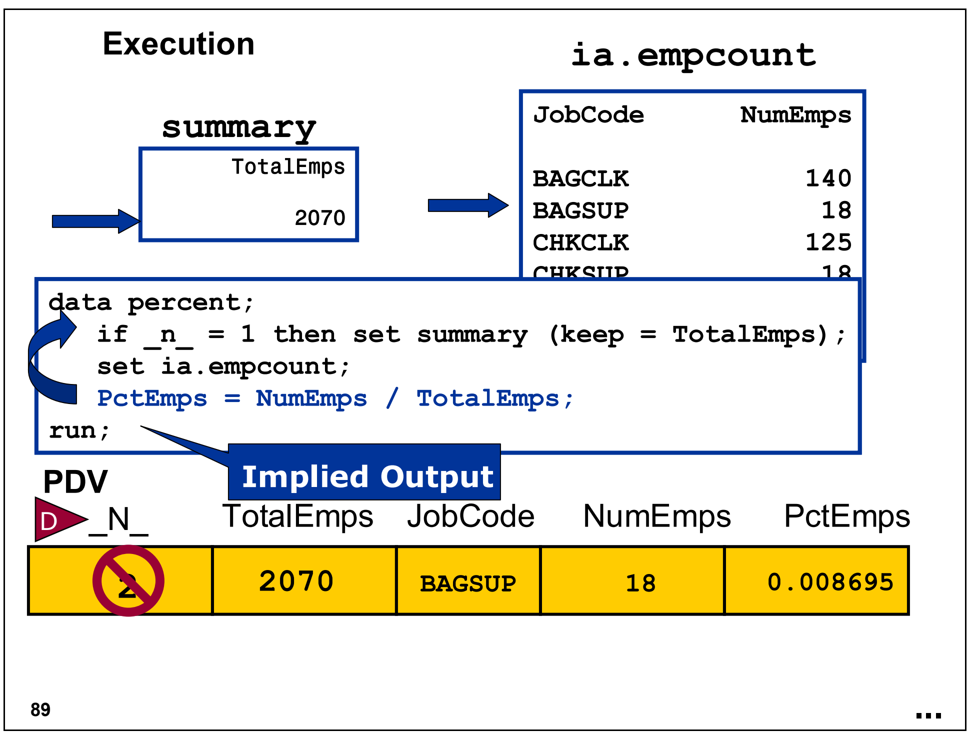

Execution
ia.empcount
summary
TotalEmps
2070
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
CHKSUP 18
data percent;
if _n_ = 1 then set summary (keep = TotalEmps);
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
PDV
D _N_
TotalEmps JobCode NumEmps PctEmps
1 2070 BAGCLK 140 0.067632
84
...


Execution
ia.empcount
summary
TotalEmps
2070
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
CHKSUP 18
data percent;
if _n_ = 1 then set summary (keep = TotalEmps);
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
Implied Output
PDV
D _N_
TotalEmps JobCode NumEmps PctEmps
1 2070 BAGCLK 140 0.067632
85
...


Execution
ia.empcount
summary
TotalEmps
2070
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
CHKSUP 18
data percent;
if _n_ = 1 then set summary (keep = TotalEmps);
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
False
PDV
D
_N_ TotalEmps JobCode NumEmps PctEmps
2 2070 BAGCLK 140 .
86
...


Execution
ia.empcount
summary
TotalEmps
2070
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
CHKSUP 18
data percent;
if _n_ = 1 then set summary (keep = TotalEmps);
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
PDV
D
_N_ TotalEmps JobCode NumEmps PctEmps
2 2070 BAGSUP 18 .
87
...


Execution
ia.empcount
summary
TotalEmps
2070
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
data percent;
if _n_ = 1 then set summary (keep = TotalEmps);
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
PDV
D _N_ TotalEmps JobCode NumEmps PctEmps
2 2070 BAGSUP 18 0.008695
88
...


Execution
ia.empcount
summary
TotalEmps
2070
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
data percent;
if _n_ = 1 then set summary (keep = TotalEmps);
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
Implied Output
PDV
D _N_ TotalEmps JobCode NumEmps PctEmps
2 2070 BAGSUP 18 0.008695
89
...

Partial Output

The previous program creates the following data:

Percentage of Each Job Code

Obs	Total Emps	Job Code	Num Emps	PctEmps
1	2070	BAGCLK	140	6.76%
2	2070	BAGSUP	18	0.87%
3	2070	CHKCLK	125	6.04%
4	2070	CHKSUP	18	0.87%
5	2070	FACCLK	124	5.99%
6	2070	FACMGR	17	0.82%
7	2070	FACMNT	60	2.90%
8	2070	FINACT	36	1.74%
9	2070	FINCLK	53	2.56%
10	2070	FINMGR	20	0.97%

90 c03s2d2

The report was created by the following program:

```
proc print data = percent noobs;
   title 'Percentage of Each Job Code';
   format PctEmps percent8.2;
run;
```

Combining Data Using SQL

You can join a summary data set and a detail data set using SQL.

```
proc sql;
   title 'Percentage of Each Job Code';
   create table percent as
   select JobCode, NumEmps,
          NumEmps / TotalEmps as PctEmps
      from summary, ia.empcount;
quit;
```

This program takes advantage of the Cartesian product that SQL forms when BY values repeat.

91 c03s2d3

Combining Data Using SQL

Partial Output from SQL Join

Percentage of Each Job Code

Obs	Job Code	Num Emps	PctEmps
1	BAGCLK	140	6.76%
2	BAGSUP	18	0.87%
3	CHKCLK	125	6.04%
4	CHKSUP	18	0.87%
5	FACCLK	124	5.99%
6	FACMGR	17	0.82%
7	FACMNT	60	2.90%
8	FINACT	36	1.74%
9	FINCLK	53	2.56%
10	FINMGR	20	0.97%

92 c03s2d3

The report was created by the following program:

```
proc print data = percent noobs;
   title 'Percentage of Each Job Code';
   format PctEmps percent8.2;
run;
```

Combining Data Using SQL

You can remerge overall summary results, such as grand totals, with detail data using SQL.

```
proc sql;
   title 'Remerging Summary Data with Detail Data';
   create table percent as
      select JobCode, NumEmps,
             NumEmps / sum(NumEmps) as PctEmps
         from ia.empcount;
quit;
```

93 c03s2d4

When SQL remerges summary data, it puts a note in the SAS log:

```
7    proc sql;
8       title 'Remerging Summary Data with Detail Data';
9       create table percent as
10         select JobCode, NumEmps,
11                NumEmps / sum(NumEmps) as PctEmps
12            from ia.empcount;
NOTE: The query requires remerging summary statistics back with the original data.
13   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.33 seconds
      cpu time            0.05 seconds
```

Combining Data using SQL

Partial Output from SQL Remerge

Remerging Summary Data with Detail Data

Obs	Job Code	Num Emps	PctEmps
1	BAGCLK	140	6.76%
2	BAGSUP	18	0.87%
3	CHKCLK	125	6.04%
4	CHKSUP	18	0.87%
5	FACCLK	124	5.99%
6	FACMGR	17	0.82%
7	FACMNT	60	2.90%
8	FINACT	36	1.74%
9	FINCLK	53	2.56%
10	FINMGR	20	0.97%

c03s2d4

94

The report was created by the following program:

```
proc print data = percent noobs;
   title 'Percentage of Each Job Code';
   format PctEmps percent8.2;
run;
```

Exercises

4. Creating a Summary Data Set

Using PROC MEANS, create a SAS data set named **ia.mean** that contains the overall average employee contribution stored in **ia.contrib**. Name the summary variable **AvgAmt**.

Partial Listing of **ia.contrib**

```
                         ia.contrib

                                Qtr
             Obs    EmpID       Num     Amount

               1    E00224      qtr1      12
               2    E00224      qtr2      33
               3    E00224      qtr3      22
               4    E00224      qtr4       .
               5    E00367      qtr1      35
               6    E00367      qtr2      48
               7    E00367      qtr3      40
               8    E00367      qtr4      30
               9    E00441      qtr1       .
              10    E00441      qtr2      63
              11    E00441      qtr3      89
              12    E00441      qtr4      90
              13    E00587      qtr1      16
              14    E00587      qtr2      19
              15    E00587      qtr3      30
              16    E00587      qtr4      29
              17    E00598      qtr1       4
              18    E00598      qtr2       8
              19    E00598      qtr3       6
```

Output

```
                          ia.mean

                     Obs     AvgAmt

                      1     28.9667
```

5. Combining a Summary Data Set with a Detail Data Set

Combine **`ia.mean`** from the previous exercise with **`ia.contrib`** to determine the difference between the overall average contribution and each individual employee contribution.

- Create a new SAS data set named **`diffs`** that contains the differences.
- Round the difference to the nearest cent.
- Print the resulting data set.

Partial Output

```
                                   Qtr
         Obs      AvgAmt    EmpID     Num      Amount       Diff

           1     28.9667    E00224    qtr1       12       -16.97
           2     28.9667    E00224    qtr2       33         4.03
           3     28.9667    E00224    qtr3       22        -6.97
           4     28.9667    E00224    qtr4        .           .
           5     28.9667    E00367    qtr1       35         6.03
           6     28.9667    E00367    qtr2       48        19.03
           7     28.9667    E00367    qtr3       40        11.03
           8     28.9667    E00367    qtr4       30         1.03
           9     28.9667    E00441    qtr1        .           .
          10     28.9667    E00441    qtr2       63        34.03
```

6. Combining Summary and Detail Data Using PROC SQL (Optional)

Repeat the previous exercise and use PROC SQL to achieve the same result.

3.3 Using an Index to Combine Data

Objectives

- Use the SET statement with the KEY= option to combine two SAS data sets.
- Use _IORC_ to determine whether the index search was successful.

97

Combining a Large Data Set with a Small One

Use the index on **ia.sales** to match observations from a small SAS data set, **ia.dnunder**, that contains information about New Zealand and Australia.

ia.dnunder

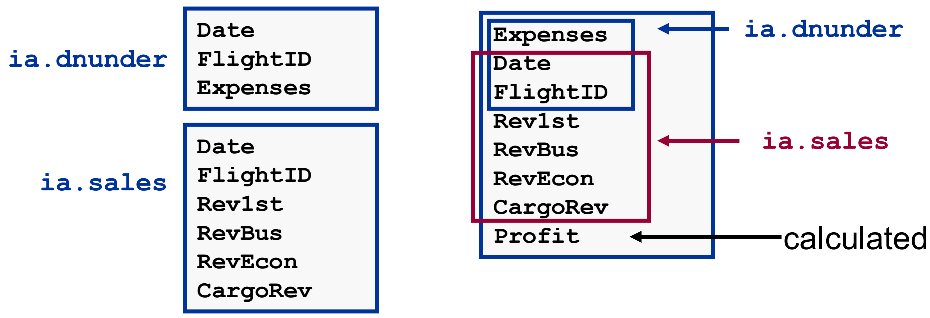
Flight ID	Expenses	FltDate
IA10200	154269	01DEC2005
IA10201	71165	01DEC2005
IA10200	65188	02DEC2005
IA10201	14259	02DEC2005
IA10200	161419	03DEC2005
IA10201	194320	03DEC2005
IA10200	140349	04DEC2005
IA10201	34894	04DEC2005
IA10200	149703	05DEC2005
IA10201	129356	05DEC2005
:	:	:

98

The data set **ia.dnunder** used for demonstrations and exercises contains fewer observations than the data set **ia.dnunder** used for the course notes.

Business Task

Build a data set with the following variables:



ia.dnunder has 900 observations.

ia.sales has 329,264 observations and a composite index, **DteFlt** on **Date** and **FlightID**.

99

✎ The data sets **ia.sales** and **ia.dnunder** used for demonstrations and exercises contain fewer observations than the data sets **ia.sales** and **ia.dnunder** used for the course notes.

Indexes on ia.sales

Partial PROC CONTENTS Output for **ia.sales**

Alphabetic List of Indexes and Attributes

#	Index	Unique Option	# of Unique Values	Variables
1	DteFlt	YES	329264	FltDate FlightID
2	Origin		52	

100

Using the KEY= Option

An index is always used when a SET or MODIFY statement contains the KEY= option.

Specify the KEY= option in the SET statement to use an index to retrieve observations that have key values equal to the current value of the key variable(s).

General form of the KEY= option:

SET *SAS-data-file-name* KEY = *index-name*;

101

- Assign a value to the index key variable(s) before the SET statement is executed.
- The index is then used to retrieve an observation with the key value.
- WHERE processing is not allowed for a data set read with the KEY= option.

Using the KEY= Option

```
data profit;
   set ia.dnunder;  ❶
   set ia.sales(keep = FlightID FltDate Rev1st
                       RevBus RevEcon RevCargo)
                  key = DteFlt;  ❷
   Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
                - Expenses);
run;
```

102 c03s3d1

① The data set **ia.dnunder** is read sequentially.

② The data set **ia.sales** is read by direct access.

```
data profit;
   set ia.dnunder;
   set ia.sales(keep = FlightID FltDate Rev1st
                       RevBus RevEcon CargoRev)
                key = DteFlt;
   Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
               - Expenses);
run;
```

Execution

FlightID	FltDate	Expenses	Rev1st	RevBus
IA10200	01DEC2005	154269	.	.

RevEcon	CargoRev	Profit	_N_
.	.	.	1

103 ...

An observation is read from **`ia.dnunder`** sequentially by the first SET statement.

```
data profit;
   set ia.dnunder;
   set ia.sales(keep = FlightID FltDate Rev1st
                       RevBus RevEcon CargoRev)
                key = DteFlt;
   Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
               - Expenses);
run;
```

Execution

FlightID	FltDate	Expenses	Rev1st	RevBus
IA10200	01DEC2005	154269	35967	37410

RevEcon	CargoRev	Profit	_N_
98124	188277	.	1

104 ...

The KEY= option causes the second SET statement to use the current PDV values for **`FlightID`** and **`FltDate`** to access an observation through the **`DteFlt`** index.

Execution

```
data profit;
   set ia.dnunder;
   set ia.sales(keep = FlightID FltDate Rev1st
                       RevBus RevEcon CargoRev)
                key = DteFlt;
   Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
                - Expenses);
run;
```

FlightID	FltDate	Expenses	Rev1st	RevBus
IA10200	01DEC2005	154269	35967	37410

RevEcon	CargoRev	Profit	D _N_
98124	188277	205509	1

105 ...

The assignment statement calculates values for **`Profit`**.

Execution

```
data profit;
   set ia.dnunder;
   set ia.sales(keep = FlightID FltDate Rev1st
                       RevBus RevEcon CargoRev)
                key = DteFlt;
   Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
                - Expenses);
run;
```

Implied Output

FlightID	FltDate	Expenses	Rev1st	RevBus
IA10200	01DEC2005	154269	35967	37410

RevEcon	CargoRev	Profit	D _N_
98124	188277	205509	1

106 ...

The observation is written to **`profit`**.

Partial Output

Partial PROC PRINT Output from **profit**

Profit for the Flights
to Australia and New Zealand

Obs	Flight ID	FltDate	Expenses	Rev1st	RevBus
1	IA10200	01DEC2005	154269	$35,967.00	$37,410.00
2	IA10201	01DEC2005	71165	$34,074.00	$42,570.00
3	IA10200	02DEC2005	65188	$30,288.00	$41,280.00
4	IA10201	02DEC2005	14259	$28,395.00	$43,860.00
5	IA10200	03DEC2005	161419	$28,395.00	$38,700.00

Obs	RevEcon	CargoRev	Profit
1	$98,124.00	$188,277.00	$205,509.00
2	$106,301.00	$178,965.00	$290,745.00
3	$96,237.00	$190,023.00	$292,640.00
4	$97,495.00	$188,277.00	$343,768.00
5	$120,139.00	$169,653.00	$195,468.00

107 c03s3d1

Partial Output

Partial PROC PRINT Output from **profit**

Profit for the Flights
to Australia and New Zealand

Obs	Flight ID	FltDate	Expenses	Rev1st	RevBus
898	IA10803	30DEC2005	1204	$1,270.00	.
899	IA10804	30DEC2005	2084	$1,397.00	.
900	IA11805	30DEC2005	4548	$1,397.00	.

Obs	RevEcon	CargoRev	Profit
898	$5,376.00	$1,860.00	$7,302.00
899	$4,872.00	$2,300.00	$6,485.00
900	$4,872.00	$2,300.00	$4,021.00

108 c03s3d1

Observation 899 is correct, but because the data values are retained when SAS reads observation 900 from **ia.dnunder**, observation 900 is incorrect.

The observation number and the data are different in the data set created during the demonstration than the one created in the course notes.

Log

```
data profit;
    set ia.dnunder;
    set ia.sales(keep =  FlightID FltDate Rev1st
                         RevBus RevEcon CargoRev)
                         key = DteFlt;
    Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
                 - Expenses);
 run;

FlightID=IA11805 FltDate=30DEC2005 Expenses=4548 Rev1st=$1,397.00
RevBus=. RevEcon=$4,872.00 CargoRev=$2,300.00 Profit=4021
_ERROR_=1 _IORC_=1230015 _N_=900
NOTE: There were 900 observations read from the data set IA.DNUNDER.
NOTE: The data set WORK.PROFIT has 900 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds
```

109 c03s3d1

The observation that appears in the log is the result of having an observation in **`ia.dnunder`** that does not match an observation in **`ia.sales`**.

The last observation in **`profit`** is incorrect because there is no flight on December 30, 2005 in the SAS data set **`ia.sales`**.

Execution

```
data profit;
   set ia.dnunder;
   set ia.sales(keep = FlightID FltDate Rev1st
                       RevBus RevEcon CargoRev)
                key = DteFlt;
   Profit = sum(Rev1st, RevBus, RevEcon, CargoRev,
                - Expenses);
run;
```

Implied Output

FlightID	FltDate	Expenses	Rev1st	RevBus
IA10804	30DEC2005	2084	1397	.

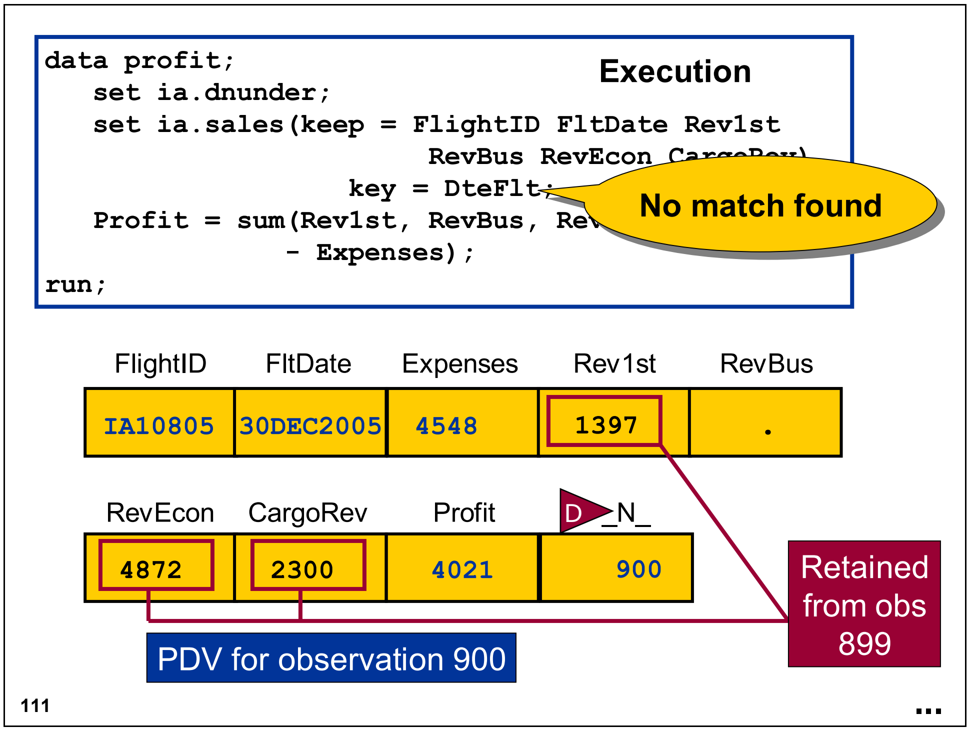
RevEcon	CargoRev	Profit	D _N_
4872	2300	6485	899

PDV for observation 899

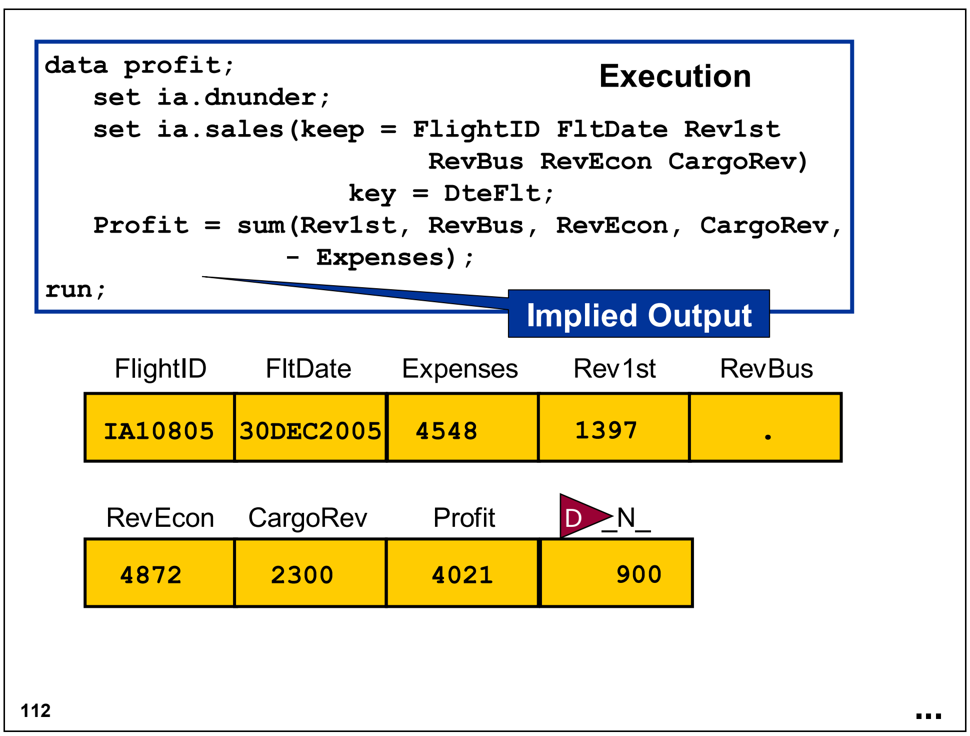
110 ...

At the next iteration of the DATA step, only **`Profit`** is reinitialized to missing.

✎ The observation number is different in the data set created during the demonstration than the one created in the course notes.



Profit is recalculated using the new value of **Expenses** and the retained values of **Rev1st**, **RevBus**, **RevEcon**, and **CargoRev**.



Using the _IORC_ Automatic Variable

When you use the KEY= option, SAS creates an automatic variable named _IORC_, which is an acronym for INPUT/OUTPUT Return Code.

You can use _IORC_ to determine whether the index search was successful.

IORC=0	indicates that SAS found a matching observation.
IORC ne 0	the SET statement did not successfully execute. One possible cause is that SAS did **not** find a matching observation.

113

For values of the _IORC_ automatic variable, see the %SYSRC autocall macro in the Macro Language Dictionary in the Base SAS Documentation.

Using the _IORC_ Automatic Variable

To prevent writing the data error to the log, perform the following tasks:

- Check the value of _IORC_.
- Set _ERROR_ to 0, if there is no match.
- Delete the non-matching data or write the non-matching data to an errors data set.

114

The automatic variable **`_error_`** controls the writing of the PDV contents to the SAS log if there is a data error. Setting **`_error_ = 0`** prevents writing to the log, even if a data error is encountered.

Using _IORC_

```
data profit errors;
   set ia.dnunder;
   set ia.sales(keep = FlightID FltDate Rev1st
                     RevBus RevEcon CargoRev)
          ❶        key = DteFlt;
   if _IORC_ = 0 then do;
      Profit = sum(Rev1st, RevBus, RevEcon,
                   CargoRev, - Expenses);
      output profit; ❷
   end;
   else do;
      _error_ = 0; ❸
      output errors;  ❹
   end;
run;
```

c03s3d2

116 ...

① Finds a match

② Outputs to **profit**

③ Prevents the non-match from appearing in the log

④ Outputs to **errors**

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Partial Output

Partial PROC PRINT Output from **`profit`**

The PROFIT Data

Obs	Flight ID	FltDate	Expenses	Rev1st	RevBus
895	IA10800	30DEC2005	2934	$1,524.00	.
896	IA10801	30DEC2005	5488	$1,524.00	.
897	IA10802	30DEC2005	3720	$1,397.00	.
898	IA10803	30DEC2005	1204	$1,270.00	.
899	IA10804	30DEC2005	2084	$1,397.00	.

Obs	RevEcon	CargoRev	Profit
895	$4,704.00	$2,420.00	5714
896	$5,586.00	$1,580.00	3202
897	$5,292.00	$1,900.00	4869
898	$5,376.00	$1,860.00	7302
899	$4,872.00	$2,300.00	6485

c03s3d2

118

The observation number and the data are different in the data set created during the demonstration than the one created in the course notes.

Output

PROC PRINT Output from **`errors`**

The ERRORS data

Obs	Flight ID	FltDate	Expenses	Rev1st	RevBus
1	IA11805	30DEC2005	4548	$1,397.00	.

Obs	RevEcon	CargoRev	Profit
1	$4,872.00	$2,300.00	.

c03s3d2

119

Log

```
249  data profit errors;
250     set ia.dnunder;
251     set ia.sales(keep = FlightID FltDate Rev1st
252                         RevBus RevEcon CargoRev)
253          key = DteFlt;
254     if _IORC_ = 0 then do;
255        Profit = sum(Rev1st, RevBus, RevEcon,
256                     CargoRev, - Expenses);
257        output profit;
258     end;
259     else do;
260        _error_ = 0;
261        output errors;
262     end;
263  run;

NOTE: There were 900 observations read from the data set IA.DNUNDER.
NOTE: The data set WORK.PROFIT has 899 observations and 8 variables.
NOTE: The data set WORK.ERRORS has 1 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds
```

c03s3d2

120

The non-matching record does not appear in the log.

For a discussion regarding duplicate observations in either the master or transaction data set, see the course section titled "Modifying SAS Data Sets in Place."

Advantages of SET/SET with the KEY= Option

- Only the necessary observations are read.
- An existing index is used.
- Multiple values can be returned.
- Availability of DATA step syntax provides the full power of the DATA step.
- Exact matches are returned.
- _IORC_ can be used to control non-matching data.

121

Disadvantages of SET/SET with the KEY= Option

- An index on one data set is required.
- Creating and maintaining an index use resources.
- Useful only for data with exact matches.

122

Exercises

7. Combining Data Sets Using an Index

Combine the **`ia.newtimes`** data set with the **`ia.schedule`** data set using the **`FltDte`** index. The data set **`ia.newtimes`** contains a column named **`TimeDiff`** that has the number of minutes later that the flight will depart.

Locate the flight using the **`FltDte`** index in the **`ia.schedule`** data set that was created in a previous exercise. If the **`FltDte`** index does not exist, create it as a composite unique index of **`Flight`** and **`Date`**.

The flight times are stored as SAS time (the number of seconds since midnight).

Create the variable **`NewDepart`** that is the new departure time for the flights. Apply the TIME5. format to **`NewDepart`**. (**Hint:** Use the expression **`sum(TimeDiff*60,depart)`**.)

Print the resulting data set.

Partial Output

```
                              work.newsched

                                         time              new
     Obs    flight          date         diff    depart    depart

      20    IA10803    30JUN2000          60     15:35     16:35
      21    IA10804    26JUN2000          75     18:35     19:50
      22    IA10804    27JUN2000          75     18:35     19:50
      23    IA10804    28JUN2000          75     18:35     19:50
      24    IA10804    29JUN2000          75     18:35     19:50
      25    IA10804    30JUN2000          75     18:35     19:50
      26    IA10805    26JUN2000          90     21:35     23:05
      27    IA10805    27JUN2000          90     21:35     23:05
      28    IA10805    28JUN2000          90     21:35     23:05
      29    IA10805    29JUN2000          90     21:35     23:05
      30    IA10805    30JUN2000          90     21:35     23:05
      31    IS10800    26JUN2000          65     21:35     22:40
```

The flight value for observation 31 is invalid.

8. Removing Erroneous Data

If you receive any non-matching data errors in your SAS log, repeat the above exercise using **`_IORC_`**. Direct data errors to a temporary error data set.

NewSched Output

work.newsched

flight	date	Time Diff	depart	New Depart
IA10800	26JUN2000	15	6:35	6:50
IA10800	27JUN2000	15	6:35	6:50
IA10800	28JUN2000	15	6:35	6:50
IA10800	29JUN2000	15	6:35	6:50
IA10800	30JUN2000	15	6:35	6:50
IA10801	26JUN2000	30	9:35	10:05
IA10801	27JUN2000	30	9:35	10:05
IA10801	28JUN2000	30	9:35	10:05
IA10801	29JUN2000	30	9:35	10:05
IA10801	30JUN2000	30	9:35	10:05
IA10802	26JUN2000	45	12:35	13:20
IA10802	27JUN2000	45	12:35	13:20
IA10802	28JUN2000	45	12:35	13:20
IA10802	29JUN2000	45	12:35	13:20
IA10802	30JUN2000	45	12:35	13:20
IA10803	26JUN2000	60	15:35	16:35
IA10803	27JUN2000	60	15:35	16:35
IA10803	28JUN2000	60	15:35	16:35
IA10803	29JUN2000	60	15:35	16:35
IA10803	30JUN2000	60	15:35	16:35
IA10804	26JUN2000	75	18:35	19:50
IA10804	27JUN2000	75	18:35	19:50
IA10804	28JUN2000	75	18:35	19:50
IA10804	29JUN2000	75	18:35	19:50
IA10804	30JUN2000	75	18:35	19:50
IA10805	26JUN2000	90	21:35	23:05
IA10805	27JUN2000	90	21:35	23:05
IA10805	28JUN2000	90	21:35	23:05
IA10805	29JUN2000	90	21:35	23:05
IA10805	30JUN2000	90	21:35	23:05

Errors Output

Errors data

Obs	flight	date	Time Diff	depart	New Depart
1	IS10800	26JUN2000	65	21:35	.

3.4 Updating Data

Objectives

- Update a master data set with a transaction data set.
- Use special missing values when updating.
- Compare the MERGE statement with the UPDATE statement.

125

Using the UPDATE Statement

Use the UPDATE statement in the DATA step to update a master data set with data in a transaction data set.

The UPDATE statement can do the following:

- change the values of variables in the master data set
- add observations to the master data set
- add variables to the master data set

126

Although the technique is not discussed in this course, the UPDATE statement can also delete observations from the master data set. See the documentation for the UPDATE statement for details.

Updating with the UPDATE Statement

Some of the Human Resources employees have changes to the employee information data stored in a data set named **ia.hrempsu**.

Transaction Data Set: **ia.hrempsu**

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

The missing values indicate that the values for those variables did not change.

127

Updating with the UPDATE Statement

Apply these changes to the master data set **ia.hremps**.

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00010	CARY	RESCLK	5153	$20,000
E00017	CARY	RESCLK	2821	$36,000
E00018	CARY	FACMNT	1459	$33,000
E00020	CARY	FACCLK	1256	$21,000
E00022	CARY	FACCLK	1255	$27,000
E00038	CARY	FACCLK	2853	$20,000
E00039	TORONTO	FACCLK	1053	$38,000
E00066	NASHVILLE	TELOP	1010	$39,000
E00068	SYDNEY	FACMNT	1060	$23,000
E00070	OSLO	RESCLK	1029	$24,000
E00076	CARY	RESMGR	1030	$36,000
E00087	FRANKFURT	HRCLK	1019	$45,000
E00088	LONDON	FACCLK	1053	$33,000
E00090	CARY	FACCLK	2657	$29,000
E00094	COPENHAGEN	HRCLK	1019	$23,000
E00105	AUSTIN	FACMNT	1062	$19,000
E00122	TORONTO	RESMGR	1030	$24,000
E00131	OSLO	RESCLK	1024	$42,000
E00133	FRANKFURT	FACCLK	1056	$38,000

Master data set:
ia.hremps
first 21 rows

Add a row.

128

...

Using the UPDATE Statement

- If an observation is in the master data set and not in the transaction data set, the observation is written to the new data set without modifying it.
- If an observation is in the transaction data set and not in the master data set, the observation is written to the new data set.
- Multiple transactions can be applied to the master data set before it is written to the new data set.
- By default, SAS does not replace existing values in the master data set with missing values if those values are coded as periods (for numeric variables) or blanks (for character variables) in the transaction data set.

129

Using a Transaction Data Set to Update

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

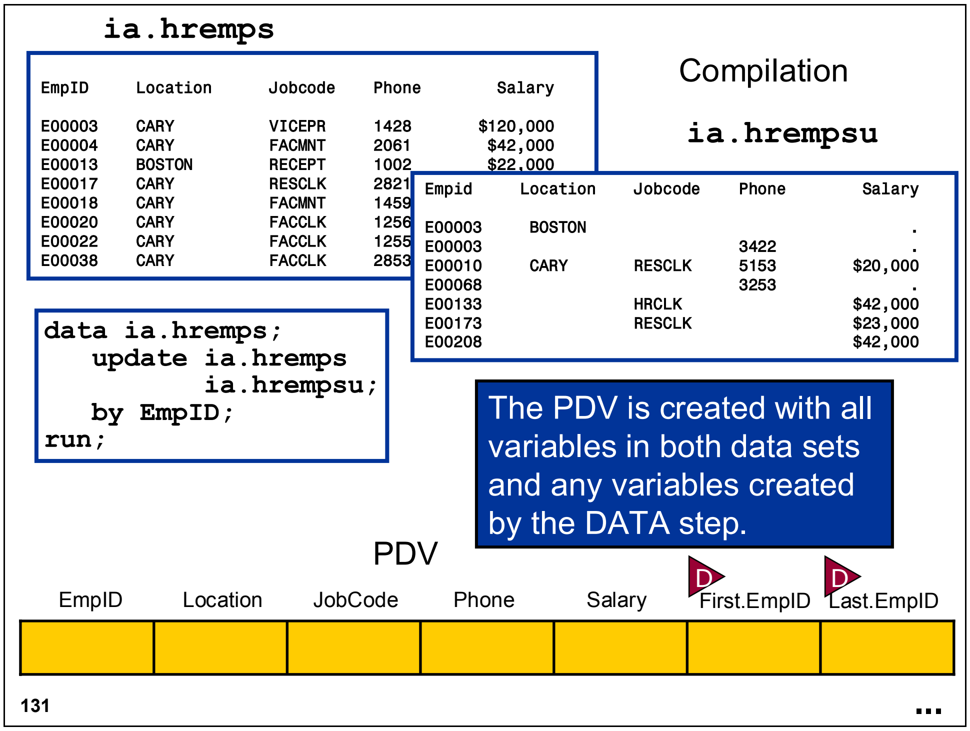
Master data set (ia.hremps)

Transaction data set (ia.hrempsu)

- The UPDATE statement requires a BY statement.
- Both data sets must be sorted or indexed.
- The BY value must be unique in the master data set.
- The transaction data set can add new BY values.

c03s4d1

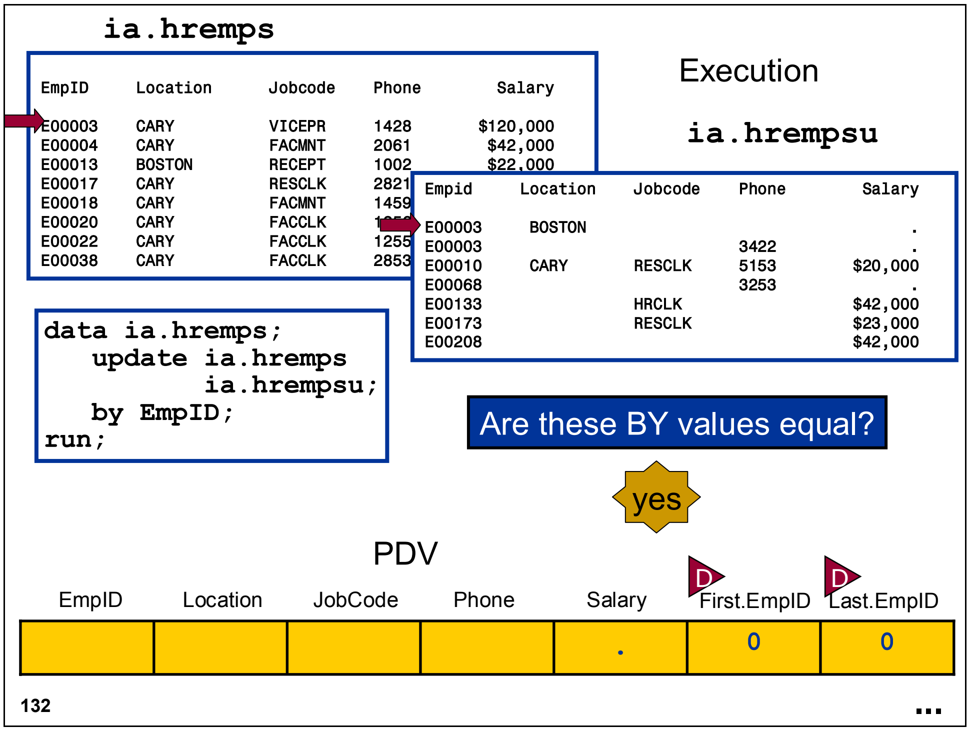
130



During compilation, the following occurs:

- SAS reads the descriptor information of each data set that is named in the UPDATE statement and creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step.
- SAS creates **`FIRST.variable`** and **`LAST.variable`** for each variable that is listed in the BY statement.

✎ **`FIRST.variable`** and **`LAST.variable`** are utilized to provide information for applying multiple transactions to an observation.



- SAS looks at the first observation in each data set that is named in the UPDATE statement to determine which BY group should appear first.
- If the transaction BY value precedes the master BY value, SAS reads from the transaction data set only and sets the variables from the master data set to missing.
- If the master BY value precedes the transaction BY value, SAS reads from the master data set only and sets the unique variables from the transaction data set to missing.
- If the BY values in the master and transaction data sets are equal, SAS reads from the master data set first and then applies the first transaction by copying the non-missing values into the program data vector.
- If the transaction data set contains multiple observations with the same BY value, non-missing values on all of those observations are applied to the data that was read from the master data set.

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	2853	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Read **ia.hremps**.

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00003	CARY	VICEPR	1428	120000	1	0

133 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	2853	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

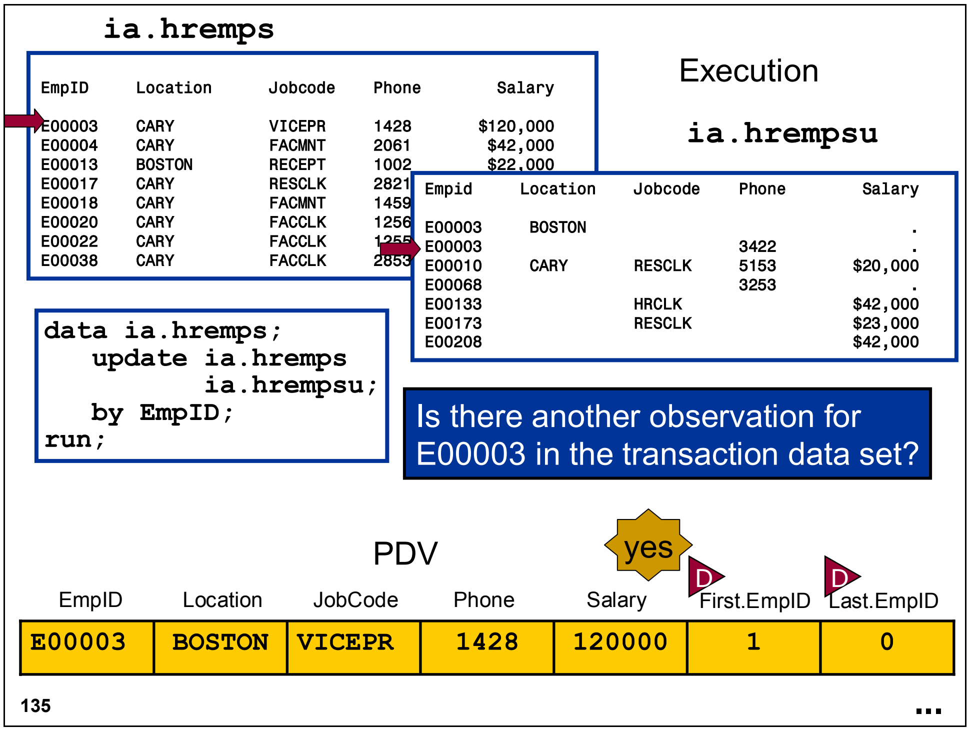
```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Apply the transactions from **ia.hrempsu**.

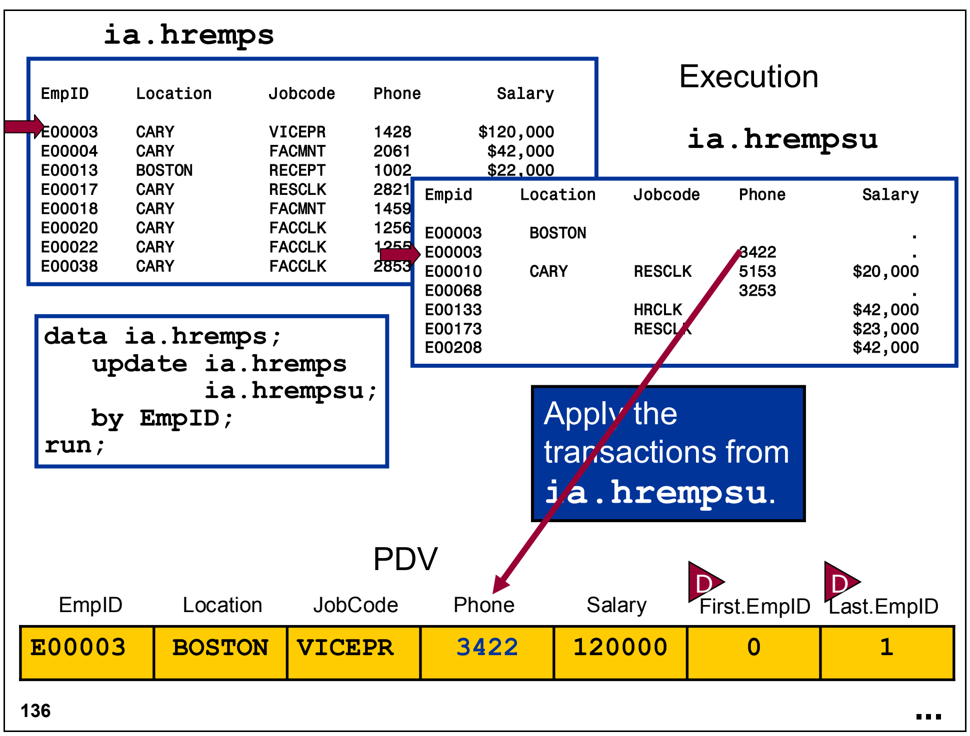
PDV

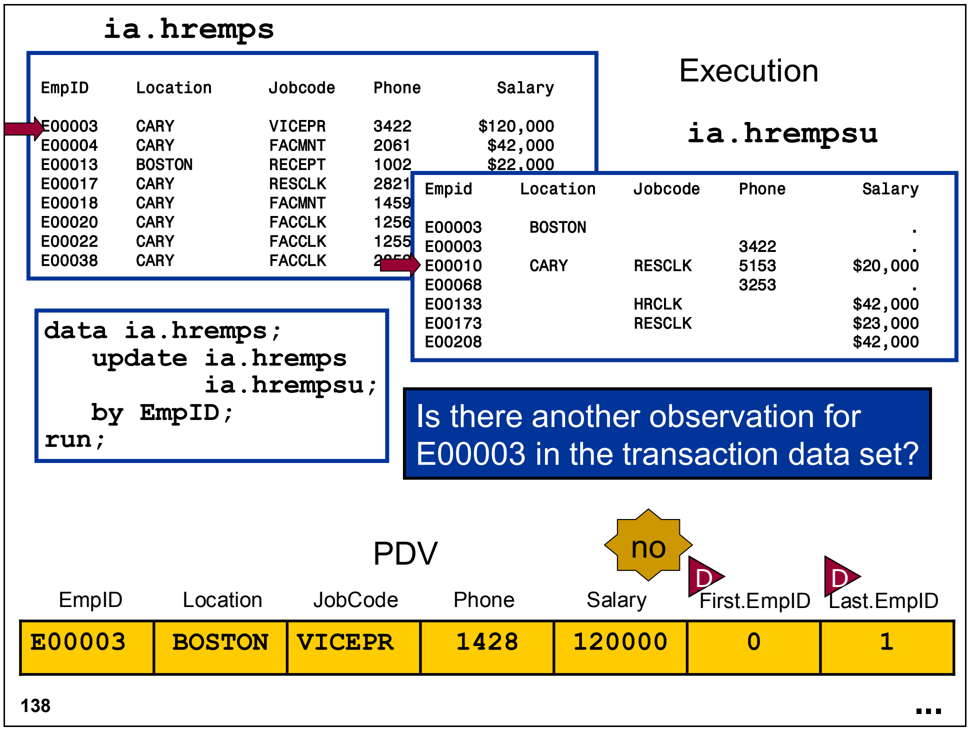
EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00003	BOSTON	VICEPR	1428	120000	1	0

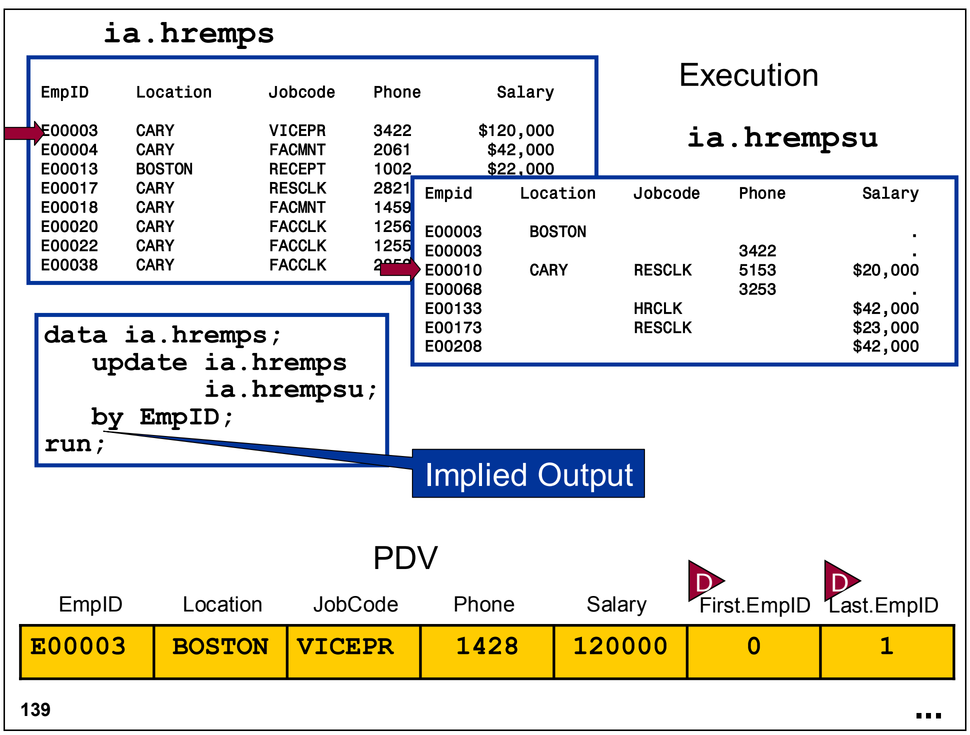
134 ...



- After completing the first transaction, SAS looks at the next observation in the transaction data set. If SAS finds one with the same BY value, it applies that transaction, too.
- The first observation then contains the new values from both transactions.







- If no other transactions exist for that observation, SAS writes the observation to the new data set and sets the values in the program data vector to missing.
- SAS repeats these steps until it reads all observations from all BY groups in both data sets.

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Which comes first?

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00003	BOSTON	VICEPR	3422	120000	0	1

140 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Reinitialize any variables unique to the transaction data set to missing.

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00003	BOSTON	VICEPR	3422	120000	0	1

141 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Read **ia.hremps**.

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00004	CARY	FACMNT	2061	42000	1	1

142 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Is there an observation for E00004 in the transaction data set?

no

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00004	CARY	FACMNT	2061	42000	1	1

143 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Implied Output

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00004	CARY	FACMNT	2061	42000	1	1

144 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Which comes first?

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00004	CARY	FACMNT	2061	42000	1	1

145 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Reinitialize any variables unique to the master data set to missing.

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00004	CARY	FACMNT	2061	42000	1	1

146 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	[illegible]	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Read **ia.hrempsu**.

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00010	CARY	RESCLK	5153	20000	1	1

147 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	2853	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

Implied Output

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00010	CARY	RESCLK	5153	20000	1	1

148 ...

ia.hremps

EmpID	Location	Jobcode	Phone	Salary
E00003	CARY	VICEPR	1428	$120,000
E00004	CARY	FACMNT	2061	$42,000
E00013	BOSTON	RECEPT	1002	$22,000
E00017	CARY	RESCLK	2821	
E00018	CARY	FACMNT	1459	
E00020	CARY	FACCLK	1256	
E00022	CARY	FACCLK	1255	
E00038	CARY	FACCLK	2853	

Execution

ia.hrempsu

Empid	Location	Jobcode	Phone	Salary
E00003	BOSTON			.
E00003			3422	.
E00010	CARY	RESCLK	5153	$20,000
E00068			3253	.
E00133		HRCLK		$42,000
E00173		RESCLK		$23,000
E00208				$42,000

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu;
   by EmpID;
run;
```

The DATA step continues until the end of file in both data sets.

PDV

EmpID	Location	JobCode	Phone	Salary	D First.EmpID	D Last.EmpID
E00531	SINGAPORE	RECEPT	1002	39000	1	1

149

Partial Output of UPDATE

Obs	EmpID	Location	Jobcode	Phone	Salary	
1	E00003	BOSTON	VICEPR	3422	$120,000	*
2	E00004	CARY	FACMNT	2061	$42,000	
3	E00010	CARY	RESCLK	5153	$20,000	**
4	E00013	BOSTON	RECEPT	1002	$22,000	
5	E00017	CARY	RESCLK	2821	$36,000	
6	E00018	CARY	FACMNT	1459	$33,000	
7	E00020	CARY	FACCLK	1256	$21,000	
8	E00022	CARY	FACCLK	1255	$27,000	
9	E00038	CARY	FACCLK	2853	$20,000	
10	E00039	TORONTO	FACCLK	1053	$38,000	
11	E00066	NASHVILLE	TELOP	1010	$39,000	
12	E00068	SYDNEY	FACMNT	3253	$23,000	*
13	E00070	OSLO	RESCLK	1029	$24,000	
14	E00076	CARY	RESMGR	1030	$36,000	
15	E00087	FRANKFURT	HRCLK	1019	$45,000	
16	E00088	LONDON	FACCLK	1053	$33,000	
17	E00090	CARY	FACCLK	2657	$29,000	
18	E00094	COPENHAGEN	HRCLK	1019	$23,000	
19	E00105	AUSTIN	FACMNT	1062	$19,000	
20	E00122	TORONTO	RESMGR	1030	$24,000	
21	E00131	OSLO	RESCLK	1024	$42,000	
22	E00133	FRANKFURT	HRCLK	1056	$42,000	*

150 * updated ** added c03s4d1

Using the UPDATE Statement

General form of the DATA step UPDATE and BY statements:

DATA *master-data-set***;**
    UPDATE *master-data-set transaction-data-set*
        <END=*variable*>
        <UPDATEMODE=
         MISSINGCHECK|NOMISSINGCHECK>;
    BY *by-variables*;
RUN;

151

END=*variable* creates and names a temporary variable that contains an end-of-file indicator. This variable is initialized to 0 and is set to 1 when the UPDATE statement processes the last observation in both data sets. This variable is not added to any data set.

UPDATE Statement Restriction Summary

UPDATE restrictions:

- Only two data set names can appear in the UPDATE statement.
- The master data set must be listed first.
- A BY statement that gives the matching variable(s) must be used.
- Both data sets must be sorted by or have indexes based on the matching variables.
- The master data set must not contain more than one observation with the same BY-variable value.

152

Missing Values in the Transaction Data Set

- By default the UPDATEMODE=MISSINGCHECK option is in effect, so missing values in the transaction data set do not replace existing values in the master data set.
- If you want missing values in the transaction data set to replace existing values in the master data set, use UPDATEMODE=NOMISSINGCHECK.

General form for the UPDATEMODE= option:

```
UPDATEMODE = MISSINGCHECK
UPDATEMODE = NOMISSINGCHECK
```

153

Missing Values in the Transaction Data Set

```
data ia.hremps;
   update ia.hremps
          ia.hrempsu
          updatemode = nomissingcheck;
   by EmpID;
run;
```

Obs	EmpID	Location	Jobcode	Phone	Salary
1	E00003			3422	.
2	E00004	CARY	FACMNT	2061	$42,000
3	E00010	CARY	RESCLK	5153	$20,000
4	E00013	BOSTON	RECEPT	1002	$22,000
5	E00017	CARY	RESCLK	2821	$36,000
6	E00018	CARY	FACMNT	1459	$33,000
7	E00020	CARY	FACCLK	1256	$21,000
8	E00022	CARY	FACCLK	1255	$27,000
9	E00038	CARY	FACCLK	2853	$20,000
10	E00039	TORONTO	FACCLK	1053	$38,000
11	E00066	NASHVILLE	TELOP	1010	$39,000
12	E00068			3253	.
13	E00070	OSLO	RESCLK	1029	$24,000
14	E00076	CARY	RESMGR	1030	$36,000
15	E00087	FRANKFURT	HRCLK	1019	$45,000

154

c03s4d2

Special Missing Values

Even when UPDATEMODE=MISSINGCHECK is in effect, you can do the following:

- retain the original value of some variables
- replace existing values of other variables with missing values by using special missing value characters in the transaction data set

If you need to update an existing value in the master data set to missing, include a special missing value in the transaction data set.

155

Special Missing Values

To create the transaction data set with special missing values, use the MISSING statement in the DATA or procedure step that creates the transaction data set.

- For character values, an underscore (_) represents the special missing value.
- For numeric values, a special missing value can be represented by an underscore (_) or any letter (A-Z, a-z). To use special numeric missing values, you must declare them in a MISSING statement.

General form of the MISSING statement:

MISSING *special-value special-value* . . . ;

156

Special Missing Values

The data set **ia.empupdates** contains new addresses, phone numbers, and birthdates of employees. One employee has a new address. One wants his address and birth date excluded. All employees want the telephone numbers excluded.

ia.empupdates

Obs	EmpID	Add1	Telephone	DOB
1	1352		_	.
2	212	12 Main St.	_	.
3	2512	_	_	_

157

The program, c03s4d3, created the transaction data set **ia.empupdates**, which contains special missing values:

```
data ia.empupdates;
   missing _;
   infile cards missover;
   input EmpID $4. Add1 $12. Telephone $ DOB ;
   cards;
1352              _
212  12 Main St.  _
2512          _   _ _
;
run;
```

Special Missing Values

The data set **ia.empinfo** has home address, telephone, and date of birth. This data needs to be updated.

ia.empinfo

Obs	EmpID	Add1	Telephone	DOB
1	1352	15 Greenwood St.	467-7753	03/03/1947
2	161	1623 N. Avon Pl.	635-5535	12/31/1945
3	212	42 Glenwood Ave.	634-2570	05/22/1953
4	2512	249 Brady St.	624-8868	04/13/1952
5	2532	2947 Arbor Lane	625-2257	11/12/1957

158

Special Missing Values

```
data ia.empinfo;
   update ia.empinfo ia.empupdates;
   by EmpID;
run;
```

Output

Obs	EmpID	Add1	Telephone	DOB
1	1352	15 Greenwood St.		03/03/47
2	161	1623 N. Avon Pl.	635-5535	12/31/45
3	212	12 Main St.		05/22/53
4	2512			.
5	2532	2947 Arbor Lane	625-2257	11/12/57

c03s4d4

159

Using UPDATE versus MERGE

UPDATE	MERGE
Two data sets at a time	Unlimited number of data sets
Can update and add observations to the data	Can update and add observations to the data
Outputs observation at the end of the BY group	Outputs each observation at the bottom of a DATA step or explicit OUTPUT statement
Does not replace existing values in the master data set with missing values in the transaction data set unless you use the UPDATEMODE = NOMISSINGCHECK UPDATE statement option or special missing characters	Automatically replaces existing values in the first data set with missing values in the second data set if the variables have the same name.

160

The output at the end of a BY group used by the UPDATE statement is called *conditional output*, where the condition is that the step reached the last observation in the BY group.

Using UPDATE versus MERGE

```
data ia.hremps;
   merge ia.hremps
         ia.hrempsu;
   by EmpID;
run;
```

Partial Output of the MERGE

Result of Merge

Obs	EmpID	Location	Jobcode	Phone	Salary
1	E00003	BOSTON			.
2	E00003			3422	.
3	E00004	CARY	FACMNT	2061	$42,000
4	E00010	CARY	RESCLK	5153	$20,000
5	E00013	BOSTON	RECEPT	1002	$22,000
6	E00017	CARY	RESCLK	2821	$36,000
7	E00018	CARY	FACMNT	1459	$33,000
8	E00020	CARY	FACCLK	1256	$21,000
9	E00022	CARY	FACCLK	1255	$27,000
10	E00038	CARY	FACCLK	2853	$20,000
11	E00039	TORONTO	FACCLK	1053	$38,000
12	E00066	NASHVILLE	TELOP	1010	$39,000
13	E00068			3253	.
14	E00070	OSLO	RESCLK	1029	$24,000

161

c03s4d5

Partial Output of UPDATE:

The Master Data Set after Updates are Applied

Obs	EmpID	Location	Jobcode	Phone	Salary
1	E00003	BOSTON	VICEPR	3422	$120,000
2	E00004	CARY	FACMNT	2061	$42,000
3	E00010	CARY	RESCLK	5153	$20,000
4	E00013	BOSTON	RECEPT	1002	$22,000
5	E00017	CARY	RESCLK	2821	$36,000
6	E00018	CARY	FACMNT	1459	$33,000
7	E00020	CARY	FACCLK	1256	$21,000
8	E00022	CARY	FACCLK	1255	$27,000
9	E00038	CARY	FACCLK	2853	$20,000
10	E00039	TORONTO	FACCLK	1053	$38,000
11	E00066	NASHVILLE	TELOP	1010	$39,000
12	E00068	SYDNEY	FACMNT	3253	$23,000
13	E00070	OSLO	RESCLK	1029	$24,000
14	E00076	CARY	RESMGR	1030	$36,000

3.5 Combining Summary and Detail Data Using Two SET Statements (Self-Study)

Combining Summary and Detail Data in the DATA Step

To create the summary statistic in the DATA step and combine it with the detail data, you must do the following:

- read the data once and calculate the summary statistic
- re-read the data to combine the summary statistic with the detail data and calculate the percentages

163

Combining Data in the DATA Step

```
data percent;
   if _n_ = 1 then do until(LastObs); ❶
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps; ❷
   end;
   set ia.empcount; ❸
   PctEmps = NumEmps / TotalEmps; ❹
run;
```

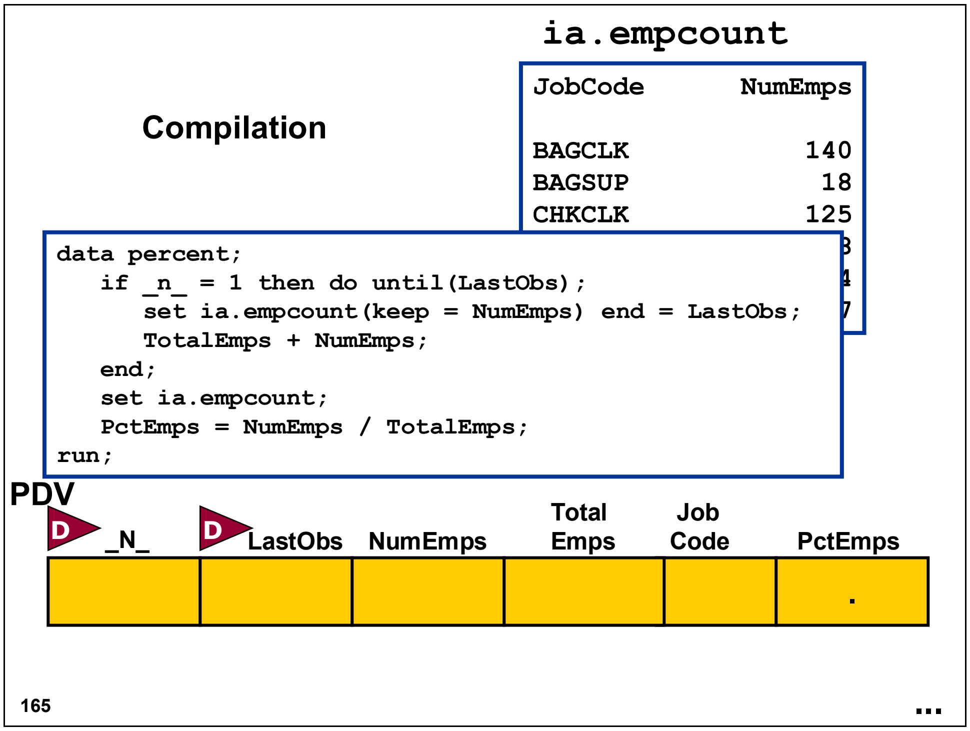
164 c03s5d1

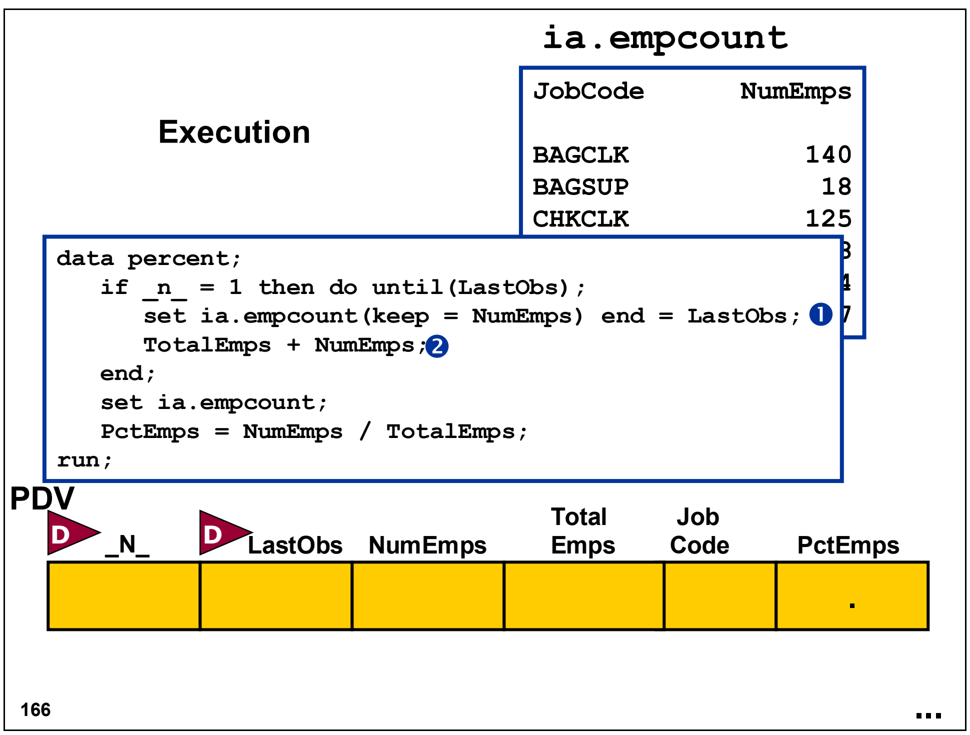
① The DO UNTIL loop is used to read through the entire data set **ia.empcount** once, in order to calculate the summary statistics.

② The SUM statement calculates the summary variable **TotalEmps**.

③ When the DO LOOP completes execution, the second SET statement reads the **ia.empcount** data set a second time.

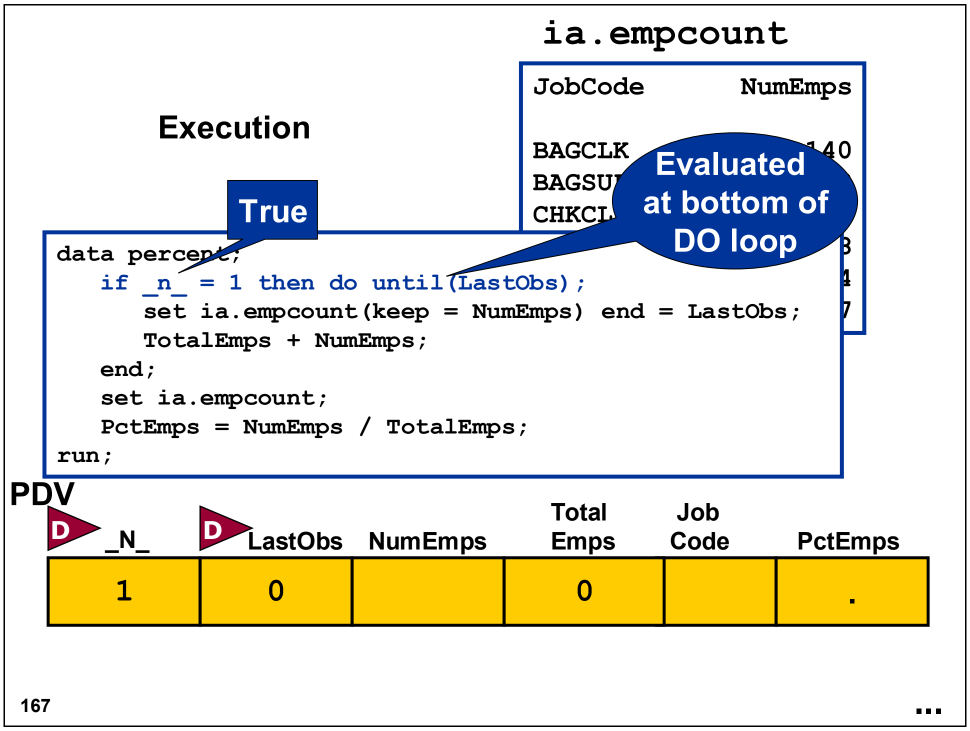
④ **PctEmps** is calculated using the **TotalEmps** summary variable.

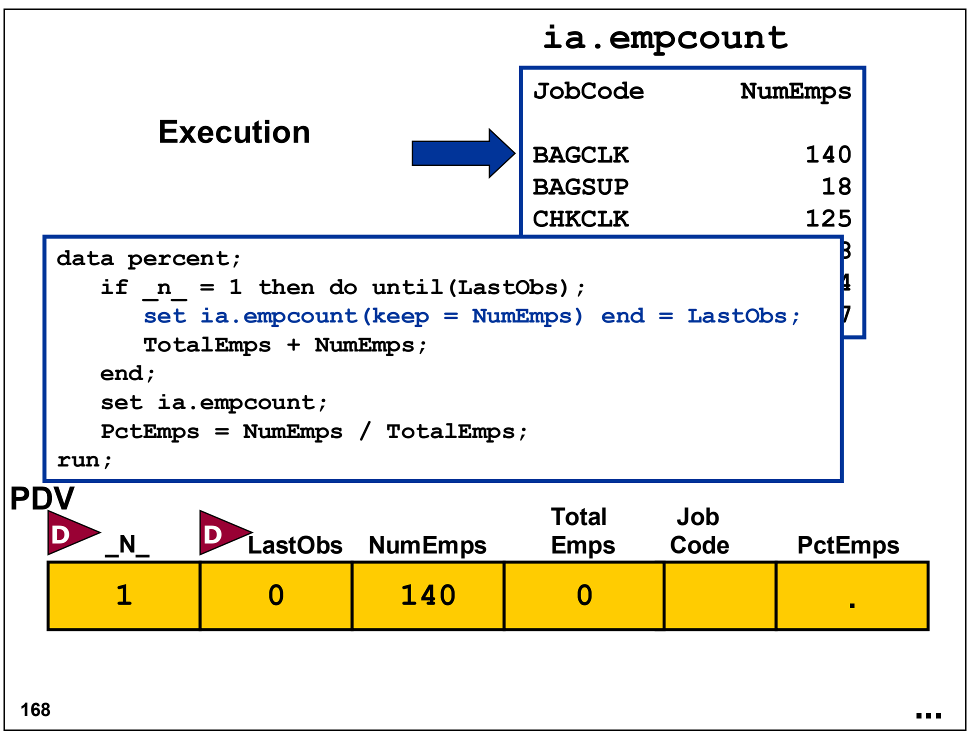




① The value for the END = variable is 0 when reading all observations from a data set except for the last one, when the value changes to 1.

② The SUM statement creates a variable that is initialized to 0 prior to the execution of the DATA step and retained across iterations of the DATA step.


ia.empcount
JobCode NumEmps
BAGCLK
BAGSU
CHKCL
Execution
True
Evaluated at bottom of DO loop
data percent;
if _n_ = 1 then do until(LastObs);
set ia.empcount(keep = NumEmps) end = LastObs;
TotalEmps + NumEmps;
end;
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
PDV
_N_
LastObs
NumEmps
Total Emps
Job Code
PctEmps
1
0
0
.
167
...


ia.empcount
JobCode NumEmps
BAGCLK 140
BAGSUP 18
CHKCLK 125
Execution
data percent;
if _n_ = 1 then do until(LastObs);
set ia.empcount(keep = NumEmps) end = LastObs;
TotalEmps + NumEmps;
end;
set ia.empcount;
PctEmps = NumEmps / TotalEmps;
run;
PDV
_N_
LastObs
NumEmps
Total Emps
Job Code
PctEmps
1
0
140
0
.
168
...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	0	140	140		.

169 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

LastObs ne 1

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	0	140	140		.

170 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSU	
CHKCL	
	3
	4
	7

Execution

Evaluated at bottom of DO loop

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	0	140	140		.

171 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	0	18	140		.

172 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	0	18	158		.

173 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

LastObs ne 1

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	0	18	158		.

174 ...

ia.empcount

JobCode	NumEmps
	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

Execution

Continuing until the last observation ...

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	1	6	2070		.

175 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	1	6	2070		.

176 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

LastObs = 1

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	1	6	2070		.

177 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	1	140	2070	BAGCLK	.

178 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	1	140	2070	BAGCLK	0.06763

179 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

Implied Output

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
1	1	140	2070	BAGCLK	0.06763

180 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125

Execution

False

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
2	1	140	2070	BAGCLK	.

181 ...

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125

Execution

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
2	1	18	2070	BAGCLK	.

182 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
2	1	18	2070	BAGCLK	0.008696

183 ...

Execution

ia.empcount

JobCode	NumEmps
BAGCLK	140
BAGSUP	18
CHKCLK	125
	3
	4
	7

```
data percent;
   if _n_ = 1 then do until(LastObs);
      set ia.empcount(keep = NumEmps) end = LastObs;
      TotalEmps + NumEmps;
   end;
   set ia.empcount;
   PctEmps = NumEmps / TotalEmps;
run;
```

Implied Output

PDV

D _N_	D LastObs	NumEmps	Total Emps	Job Code	PctEmps
2	1	18	2070	BAGCLK	0.008696

184 ...

Partial Output

Partial PROC PRINT Output from **percent**

Reading through the data twice

Obs	Num Emps	Total Emps	Job Code	PctEmps
1	140	2070	BAGCLK	6.76%
2	18	2070	BAGSUP	0.87%
3	125	2070	CHKCLK	6.04%
4	18	2070	CHKSUP	0.87%
5	124	2070	FACCLK	5.99%
6	17	2070	FACMGR	0.82%
7	60	2070	FACMNT	2.90%
8	36	2070	FINACT	1.74%
9	53	2070	FINCLK	2.56%
10	20	2070	FINMGR	0.97%

c03s5d1

185

3.6 Solutions to Exercises

1. Joining Data Sets to Create a New Data Set

Using PROC SQL, join **ia.employees**, **ia.jcodedat**, and **ia.newsals** to create a data set that contains employee IDs, employee job codes, job code descriptions, current salaries, and new salaries. Print the resulting data set.

There is no variable common to all three SAS data sets. Use PROC CONTENTS, PROC DATASETS, or the SAS Explorer to determine the columns on which to join the rows.

```
/*  DATASETS solution */
proc datasets lib = ia nolist;
   contents data = newsals;
   contents data = jcodedat;
   contents data = employees;
run;
quit;

/*  CONTENTS solution  */
proc contents data=ia.newsals;
run;

proc contents data=ia.jcodedat;
run;

proc contents data=ia.employees;
run;

/* PROC SQL solution */
proc sql;
   create table usesql as
      select e.EmpID, j.JobCode,
             Descript, Salary, NewSalary
         from ia.newsals n, ia.jcodedat j,
              ia.employees e
         where e.EmpID = n.EmpID and
               j.JobCode = e.JobCode
         order by e.EmpID;
quit;

proc print data = usesql noobs;
run;
```

2. Combining Data Using the DATA Step MERGE Statement

Repeat the same task using the DATA step MERGE statement to merge all three data sets. Print the resulting data set.

```
proc sort data = ia.jcodedat out=jcodedat;
   by JobCode;
run;

proc sort data = ia.employees out=employees;
   by JobCode;
run;

data temp1;
   merge employees(in = e) jcodedat(in = j);
   by JobCode;
   if e and j;
run;

proc sort data = ia.newsals out=newsals;
   by EmpID;
run;

proc sort data = temp1;
   by EmpID;
run;

data final;
   merge newsals(in = n) temp1(in = t);
   by EmpID;
   if n and t;
run;

proc print data=final noobs;
title;
  var EmpID JobCode Descript Salary NewSalary;
run;
```

3. Combining Two Data Sets Conditionally

The data set **ia.options** has the number of stock options awarded to the crew employees based on the date they were hired. The hired dates for the crew are stored in the data set **ia.crew**. Create a data set named **crewshrs** that combines the two data sets. The data set **crewshrs** should contain only the variables **LastName**, **FirstName**, **HireDate**, and **NumShares NumShares** and should be in order by **HireDate**.

```
/*  SQL solution  */

proc sql;
   create table crewshrs as
       select LastName, FirstName, HireDate, NumShares
       from ia.crew, ia.options
       where crew.HireDate between BeginDte and EndDte
       order by HireDate;
   select *
       from crewshrs;
quit;

/*  DATA step solution */

proc sort data = ia.crew out = crew;
   by HireDate;
run;

data crewshrs;
   keep LastName FirstName HireDate NumShares;
   set crew;
   do while (not (BeginDte le HireDate le
                           EndDte));
      set ia.options;
   end;
run;

proc print data = crewshrs;
run;
```

4. Creating a Summary Data Set

Using PROC MEANS, create a SAS data set named **ia.mean** that contains the overall average employee contribution stored in **ia.contrib**. Name the summary variable **AvgAmt**.

```
proc means data = ia.contrib noprint;
   var Amount;
   output out = ia.mean mean=AvgAmt;
run;

proc print data = ia.mean;
   title 'ia.mean';
   var avgamt;
run;
```

5. **Combining a Summary Data Set with a Detail Data Set**

Combine **ia.mean** from the previous exercise with **ia.contrib** to determine the difference between the overall average contribution and each individual employee contribution.

- Create a new SAS data set named **diffs** that contains the differences.
- Round the difference to the nearest cent.
- Print the resulting data set.

```
data diffs;
   if _n_ = 1 then set ia.mean(keep = AvgAmt);
   set ia.contrib;
   Diff = round(Amount - AvgAmt,.01);
run;

proc print data = diffs;
run;
```

6. **Combining Summary and Detail Data Using PROC SQL (Optional)**

Repeat the previous exercise using PROC SQL to achieve the same result.

```
proc sql;
   create table diffs as
     select avgamt, empid, qtrnum, amount,
            round(amount-avgamt, .01) as diff
     from ia.contrib, ia.mean;
   select * from diffs;
quit;

/* A remerge solution is also feasible */
proc sql;
   create table diffs as
     select mean(amount), empid, qtrnum, amount,
          round(amount-mean(amount), .01) as diff
      from ia.contrib;
   select * from diffs;
quit;
```

7. Combining Data Sets Using an Index

Combine the **ia.newtimes** data set with the **ia.schedule** data set using the **FltDte** index. The data set **ia.newtimes** contains a column named **TimeDiff** that has the number of minutes later that the flight will depart.

Locate the flight using the **FltDte** index in the **ia.schedule** data set that was created in a previous exercise. If the **FltDte** index does not exist, create it as a composite unique index of **Flight** and **Date**.

The flight times are stored as a SAS time (the number of seconds since midnight).

Create the variable **NewDepart** that is the new departure time for the flights. Apply the `time5.` format to **NewDepart**. (**Hint:** Use the expression **sum(TimeDiff*60,depart)**.)

Print the resulting data set.

```
data work.newsched;
   set ia.newtimes;
   set ia.schedule key = FltDte;
   NewDepart = sum(TimeDiff*60,depart);
   format NewDepart time5.;
run;

proc print data = work.newsched;
   title 'work.newsched';
run;
```

8. Removing Erroneous Data

If you receive any nonmatching data errors in your SAS log, repeat the above exercise using **_IORC_**. Direct data errors to a temporary error data set.

```
data work.newsched work.errors;
   set ia.newtimes;
   set ia.schedule key = FltDte;
   if _IORC_ = 0 then do;
       NewDepart = sum(TimeDiff*60,depart);
       output work.newsched;
   end;
   else do;
       _error_ = 0;
       output work.errors;
   end;
   format NewDepart time5.;
run;
proc print data = work.newsched;
   title 'work.newsched';
run;
proc print data = work.errors;
   title 'Errors data';
run;
```

Chapter 4 Using Lookup Tables to Match Data

4.1 Introduction to Lookup Techniques

Objectives

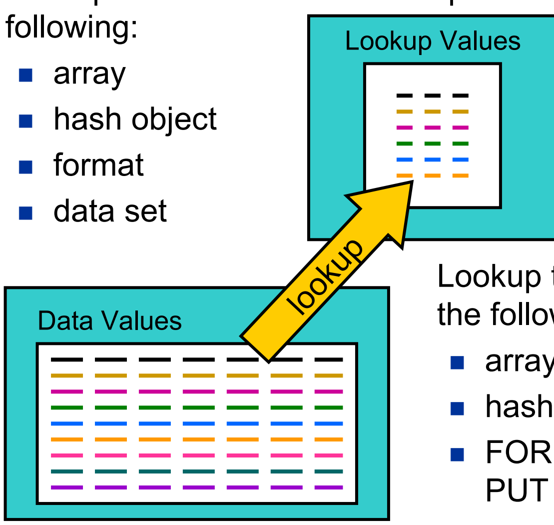
- Define table lookup.
- Investigate table look up techniques.

3

Table Lookups

Lookup values for a table lookup can be stored in the following:

- array
- hash object
- format
- data set



Lookup techniques include the following:

- array index value
- hash object key value
- FORMAT statement, PUT function
- merge, join

4

✎ The hash object is new in SAS®9.

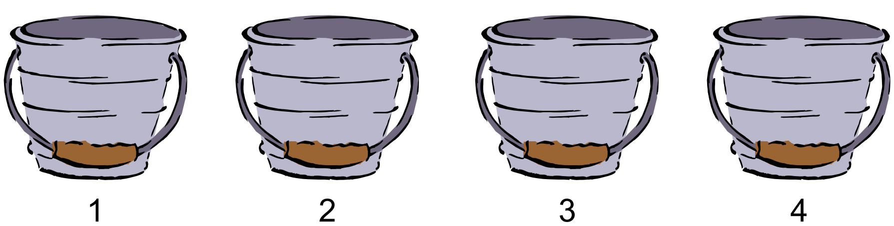
Overview of Table Look Up Techniques

- Arrays, hash objects, and formats provide an in-memory lookup table.
- The merge and join use lookup values that are stored on disk.

5

Overview of Arrays

An array is similar to a row of buckets.

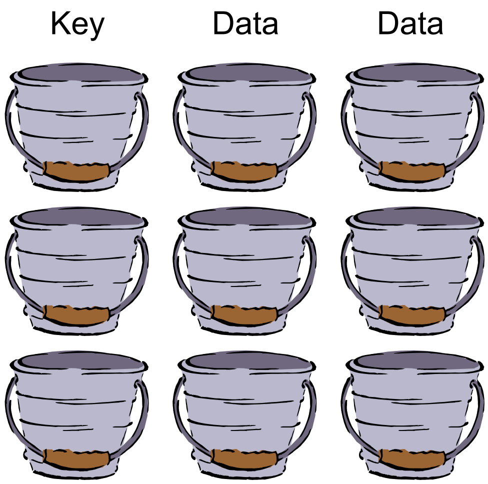


- SAS puts a value in a bucket based on the bucket number.
- Values are retrieved from a bucket based on the bucket number.

6 ...

Overview of a Hash Object

A hash object is similar to stacks of buckets that are referred to by the value of a key.



- SAS puts value(s) in the **data** bucket(s) based on the value(s) in the **key** bucket.
- Value(s) are retrieved from the **data** bucket(s) based on the value(s) in the **key** bucket.

7 ...

Overview of a Format

A format is similar to stacks of buckets that are referred to by the value of a variable.

- SAS puts data values and label values in the buckets when the format is used in a FORMAT statement, PUT function, or PUT statement.
- SAS uses a binary search on the **data value** bucket in order to return the value in the **label** bucket.

8 ...

4.2 Using Arrays as Lookup Tables

Objectives

- Review one dimensional arrays.
- Write an ARRAY statement for a multidimensional array.
- Process a multidimensional array.
- Load a multidimensional array from a SAS data set.
- Use a multidimensional array to compare values.

10

Overview of Arrays

An array is similar to a row of buckets.

- SAS puts a value in a bucket based on the bucket number.
- Values are retrieved from a bucket based on the bucket number.

11

Reviewing Arrays

An *array*

- is a temporary grouping of SAS variables that are arranged in a particular order and identified by an array name
- exists only for the duration of the current DATA step.

An *array* can

- perform repetitive calculations on a group of variables
- create many variables with the same attributes
- restructure data
- perform a table lookup with one or more numeric factors.

12

Using One-dimensional Arrays (Review)

To use an array in a DATA step, declare the array by using an ARRAY statement.

General form for the one-dimensional ARRAY statement:

> **ARRAY** *array-name* {*number-of-elements*} <$> <*length*>
> <*list-of-variables*> <(*initial-values*)>;

```
array numarray{3} num1 - num3;
```

```
array char{4} $ 6;
```

```
array num{5} _temporary_ (5, 6, 7, 8, 9);
```

13

array-name is a SAS name that identifies the group of variables.

number-of-elements is the number of variables in the group. You must enclose this value in parentheses, braces, or brackets.

$ indicates that the elements in the array are character elements

length specifies the length of elements in the array that were not previously assigned a length.

list-of-variables is a list of the names of the variables in the group. All variables that are defined in a given array must be of the same type, either all character or all numeric.

initial-values gives initial values for the corresponding positional elements in the array.

The keyword _TEMPORARY_ can be used instead of *list-of-variables* to avoid creating new data set variables.

Using One-dimensional Arrays (Review)

To use an array in a DATA step, declare the array by using an ARRAY statement.

General form for the one-dimensional ARRAY statement:

> **ARRAY** *array-name* {*number-of-elements*} <$> <*length*>
> <*list-of-variables*> <(*initial-values*)>;

Array name

```
array numarray{3} num1 - num3;
array char{4} $ 6;
array num{5} _temporary_ (5, 6, 7, 8, 9);
```

14

Using One-dimensional Arrays (Review)

To use an array in a DATA step, declare the array by using an ARRAY statement.

General form for the one-dimensional ARRAY statement:

> **ARRAY** *array-name* {*number-of-elements*} <$> <*length*>
> <*list-of-variables*> <(*initial-values*)>;

```
array numarray{3} num1 - num3;
array char{4} $ 6;
array num{5} _temporary_ (5, 6, 7, 8, 9);
```

number of elements

15

Using One-dimensional Arrays (Review)

To use an array in a DATA step, declare the array by using an ARRAY statement.

General form for the one-dimensional ARRAY statement:

ARRAY *array-name* {*number-of-elements*} <$> <*length*>
<*list-of-variables*> <(*initial-values*)>;

```
array numarray{3} num1 - num3;
```

List of numeric variables (num1 - num3)

```
array char{4} $ 6;
```

Creates four character variables, char1 – char4, each a length of 6 ($ 6)

```
array num{5} _temporary_ (5, 6, 7, 8, 9);
```

Creates temporary numeric variables, and stores the numeric values 5, 6, 7, 8, 9

16

Using a One-dimensional Array

The data set **ia.rdudelay** contains the actual number of minutes that the January 2004 flights to Raleigh were delayed.

Partial Listing of **ia.rdudelay**

Obs	Flight ID	FltDate	Delay
1	IA00201	01JAN2004	11
2	IA00200	01JAN2004	22
3	IA00400	01JAN2004	25
4	IA00401	01JAN2004	8
5	IA00600	01JAN2004	6
6	IA00601	01JAN2004	22

17

Using a One-dimensional Array

The data set **ia.delaystats** contains delay statistics for all flights in January 2004 with each day stored in a variable named **JAN01** to **JAN31**.

First 10 Variables in **ia.delaystats**

Statistic	JAN01	JAN02	JAN03	JAN04	JAN05	JAN06	JAN07	JAN08	JAN09
AvgDelay	4.708	4.760	5.842	6.571	4.645	6.0714	5.500	5.080	4.692
SumDelay	113.000	119.000	111.000	184.000	144.000	85.0000	121.000	127.000	122.000
StdDelay	2.971	3.140	3.420	4.316	3.508	4.5987	4.373	4.252	4.688
MedianDelay	5.000	4.000	6.000	6.500	4.000	4.5000	4.000	3.000	2.500

The two data sets must be combined to calculate the difference between the average delay and the actual delay for each day.

18

Desired Results

Obs	Flight ID	FltDate	Delay	Average	Delay Dif
1	IA00201	01JAN2004	11	4.70833	6.2917
2	IA00200	01JAN2004	22	4.70833	17.2917
3	IA00400	01JAN2004	25	4.70833	20.2917
<lines removed>					
87	IA00201	02JAN2004	8	4.76000	3.2400
88	IA00200	02JAN2004	30	4.76000	25.2400
89	IA00400	02JAN2004	23	4.76000	18.2400
<lines removed>					
174	IA00200	03JAN2004	12	5.84211	6.1579
175	IA00400	03JAN2004	11	5.84211	5.1579
176	IA00401	03JAN2004	9	5.84211	3.1579
<lines removed>					

from **ia.rdudelay**

Load an array from **ia.delaystats**

calculated

19

Stored Array Values

Array values should be stored in a SAS data set when the following conditions exist:

- too many values to initialize easily in the ARRAY
- values changing frequently
- the same values used in many programs

20

Using a One-dimensional Array

```
data compare;
   keep FlightID FltDate Delay Average
        DelayDif;
   if _n_ = 1 then do; ❶
      set ia.delaystats(where =
                (Statistic = 'AvgDelay'));
      array jan{31} Jan01 - Jan31; ❷
   end;
   set ia.rdudelay;
   day = day(FltDate);
   Average = Jan{day}; ❸
   DelayDif = Delay - Average;
run;
```

21 c04s2d1

① During the first time through the DATA step, the data set **ia.delaystats** is read into the PDV.

② The array **JAN** is associated with the variables **Jan01**, **Jan02**, **Jan03**, and so forth. The ARRAY statement that defines the array **JAN** appears after the SET statement for the data set that contains the variables **JAN01** – **JAN31**. The array statement does not have to be inside the DO loop because it is a non-executable statement.

③ The value of the **JAN** array referenced positionally by the value of the variable **day** is given to the variable **Average**.

Statistic	JAN01	JAN02	JAN03	JAN04	JAN05	JAN06	JAN07	JAN08	JAN09 . . .
AvgDelay	4.708	4.760	5.842	6.571	4.645	6.0714	5.500	5.080	4.692 . . .

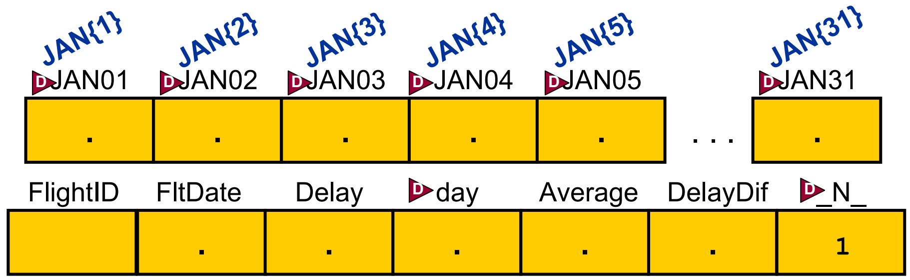
```
data compare;
   keep FlightID FltDate Delay Average
        DelayDif;
   if _n_ = 1 then do;
      set ia.delaystats(where =
              (Statistic = 'AvgDelay'));
      array jan{31} Jan01 - Jan31;
   end;
   set ia.rdudelay;
   day = day(FltDate);
   Average = Jan{day};
   DelayDif = Delay - Average;
run;
```

```
ia.delaystats (where =
    (Statistic = 'AvgDelay'));
```

Execution

ia.rdudelay

Flight ID	FltDate	Delay
IA00201	01JAN2004	11
IA00200	01JAN2004	22
IA00400	01JAN2004	25



22 ...

Statistic	JAN01	JAN02	JAN03	JAN04	JAN05	JAN06	JAN07	JAN08	JAN09 . . .
AvgDelay	4.708	4.760	5.842	6.571	4.645	6.0714	5.500	5.080	4.692 . . .

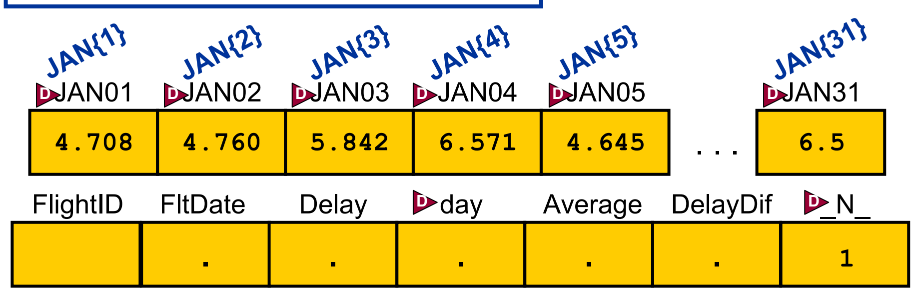
```
data compare;
   keep FlightID FltDate Delay Average
        DelayDif;
   if _n_ = 1 then do;
      set ia.delaystats(where =
              (Statistic = 'AvgDelay'));
      array jan{31} Jan01 - Jan31;
   end;
   set ia.rdudelay;
   day = day(FltDate);
   Average = Jan{day};
   DelayDif = Delay - Average;
run;
```

```
ia.delaystats (where =
    (Statistic = 'AvgDelay'));
```

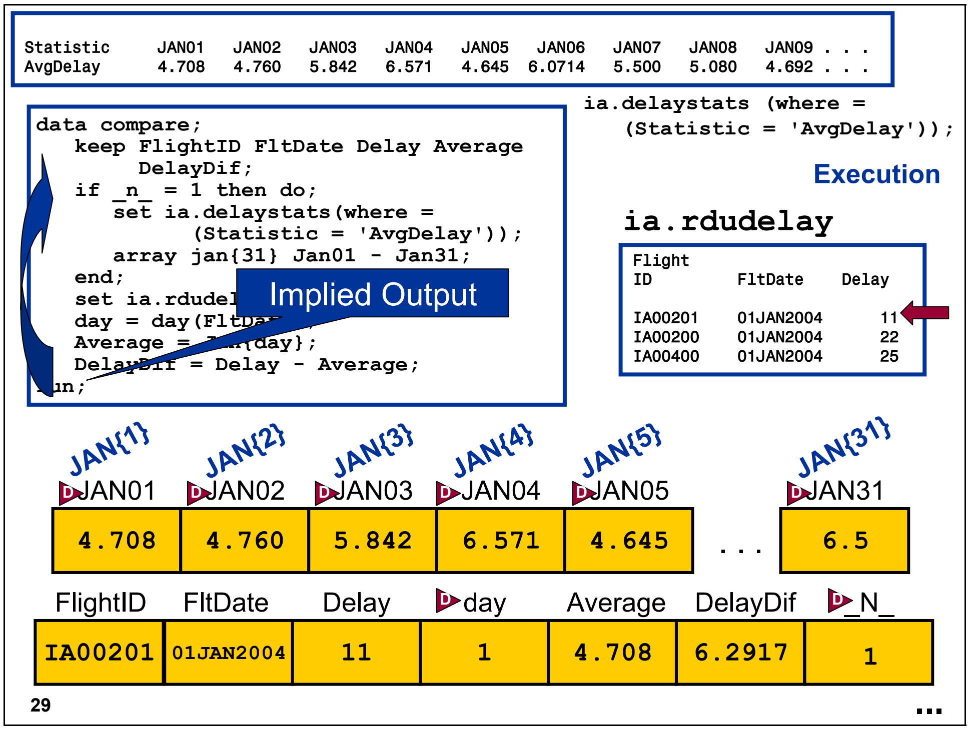
Execution

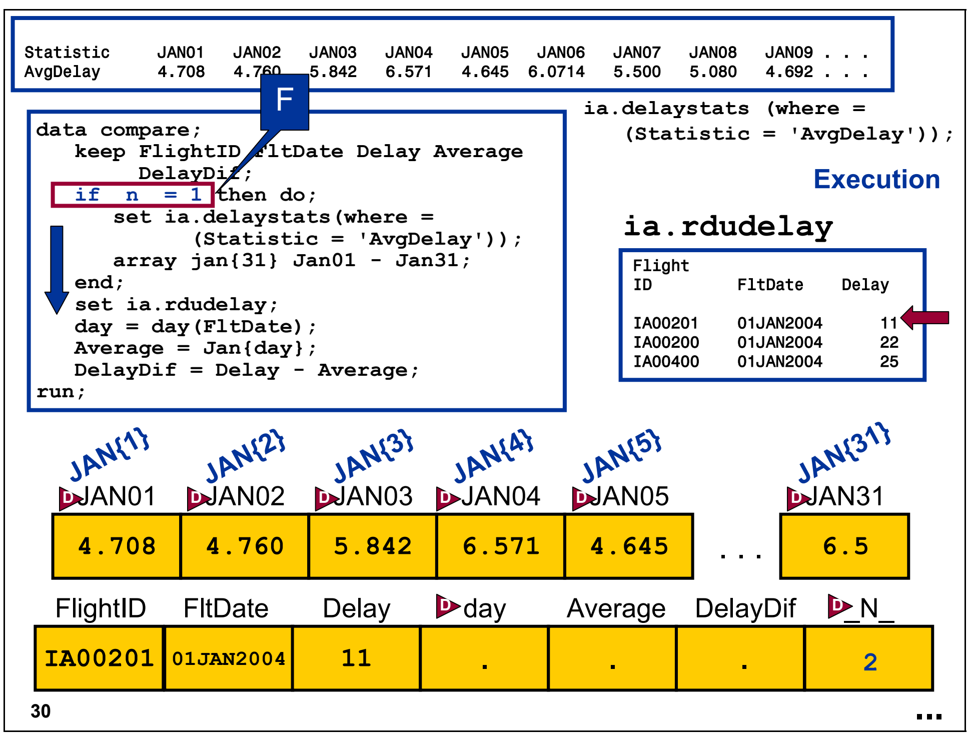
ia.rdudelay

Flight ID	FltDate	Delay
IA00201	01JAN2004	11
IA00200	01JAN2004	22
IA00400	01JAN2004	25



24 ...


Statistic JAN01 JAN02 JAN03 JAN04 JAN05 JAN06 JAN07 JAN08 JAN09 . . .
AvgDelay 4.708 4.760 5.842 6.571 4.645 6.0714 5.500 5.080 4.692 . . .
data compare;
keep FlightID FltDate Delay Average
DelayDif;
if _n_ = 1 then do;
set ia.delaystats(where =
(Statistic = 'AvgDelay'));
array jan{31} Jan01 - Jan31;
end;
set ia.rdudel
day = day(FltDa
Average = Jan{day};
DelayDif = Delay - Average;
run;
Implied Output
ia.delaystats (where =
(Statistic = 'AvgDelay'));
Execution
ia.rdudelay
Flight ID FltDate Delay
IA00201 01JAN2004 11
IA00200 01JAN2004 22
IA00400 01JAN2004 25
JAN{1} JAN{2} JAN{3} JAN{4} JAN{5} JAN{31}
JAN01 JAN02 JAN03 JAN04 JAN05 JAN31
4.708 4.760 5.842 6.571 4.645 . . . 6.5
FlightID FltDate Delay day Average DelayDif _N_
IA00201 01JAN2004 11 1 4.708 6.2917 1
29
...


Statistic JAN01 JAN02 JAN03 JAN04 JAN05 JAN06 JAN07 JAN08 JAN09 . . .
AvgDelay 4.708 4.760 5.842 6.571 4.645 6.0714 5.500 5.080 4.692 . . .
F
data compare;
keep FlightID FltDate Delay Average
DelayDif;
if _n_ = 1 then do;
set ia.delaystats(where =
(Statistic = 'AvgDelay'));
array jan{31} Jan01 - Jan31;
end;
set ia.rdudelay;
day = day(FltDate);
Average = Jan{day};
DelayDif = Delay - Average;
run;
ia.delaystats (where =
(Statistic = 'AvgDelay'));
Execution
ia.rdudelay
Flight ID FltDate Delay
IA00201 01JAN2004 11
IA00200 01JAN2004 22
IA00400 01JAN2004 25
JAN{1} JAN{2} JAN{3} JAN{4} JAN{5} JAN{31}
JAN01 JAN02 JAN03 JAN04 JAN05 JAN31
4.708 4.760 5.842 6.571 4.645 . . . 6.5
FlightID FltDate Delay day Average DelayDif _N_
IA00201 01JAN2004 11 . . . 2
30
...

```
Statistic     JAN01   JAN02   JAN03   JAN04   JAN05   JAN06    JAN07   JAN08   JAN09 . . .
AvgDelay      4.708   4.760   5.842   6.571   4.645   6.0714   5.500   5.080   4.692 . . .
```

```
data compare;
   keep FlightID FltDate Delay Average
        DelayDif;
   if _n_ = 1 then do;
      set ia.delaystats(where =
            (Statistic = 'AvgDelay'));
      array jan{31} Jan01 - Jan31;
   end;
   set ia.rdudelay;
   day = day(FltDate);
   Average = Jan{day};
   DelayDif = Delay - Average;
run;
```

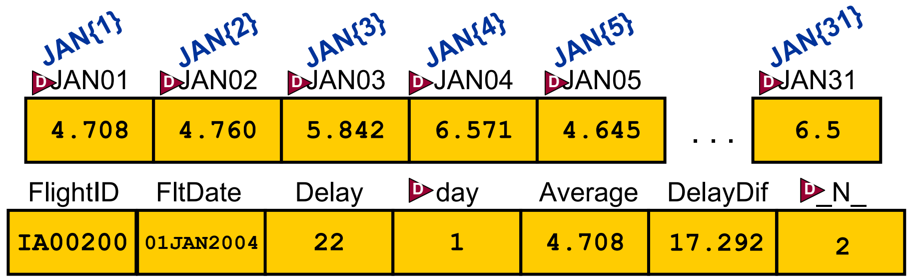
Implied Output

```
ia.delaystats (where =
    (Statistic = 'AvgDelay'));
```

Execution

ia.rdudelay

Flight ID	FltDate	Delay
IA00201	01JAN2004	11
IA00200	01JAN2004	22
IA00400	01JAN2004	25



33 ...

Using a One-dimensional Array

Partial Output

Obs	Flight ID	FltDate	Delay	Average	Delay Dif
1	IA00201	01JAN2004	11	4.70833	6.2917
2	IA00200	01JAN2004	22	4.70833	17.2917
3	IA00400	01JAN2004	25	4.70833	20.2917
4	IA00401	01JAN2004	8	4.70833	3.2917
5	IA00600	01JAN2004	6	4.70833	1.2917
6	IA00601	01JAN2004	22	4.70833	17.2917
7	IA00602	01JAN2004	2	4.70833	-2.7083
8	IA00603	01JAN2004	22	4.70833	17.2917
9	IA00604	01JAN2004	21	4.70833	16.2917
10	IA00605	01JAN2004	23	4.70833	18.2917
11	IA00800	01JAN2004	15	4.70833	10.2917

34

Using Multidimensional Arrays

International Airlines needs to determine the windchill values for each flight. Windchill values are derived from the air temperature and wind speed.

Temperature

Wind Speed

	-10	-5	0	5	10	15	20	25	30
5	-22	-16	-11	-5	1	7	13	19	25
10	-28	-22	-16	-10	-4	3	9	15	21
15	-32	-26	-19	-13	-7	0	6	13	19
20	-35	-29	-22	-15	-9	-2	4	11	17
25	-37	-31	-24	-17	-11	-4	3	9	16
30	-39	-33	-26	-19	-12	-5	1	8	15
35	-41	-34	-27	-21	-14	-7	0	7	14
40	-43	-36	-29	-22	-15	-8	-1	6	13

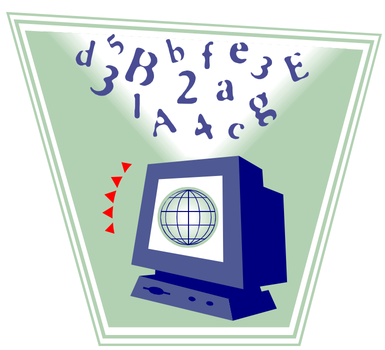
35

For more information regarding the windchill index, see www.weatherimages.org/data/windchill.html.

Using Multidimensional Arrays

When the lookup operation depends on more than one numerical factor, you can use a multidimensional array.

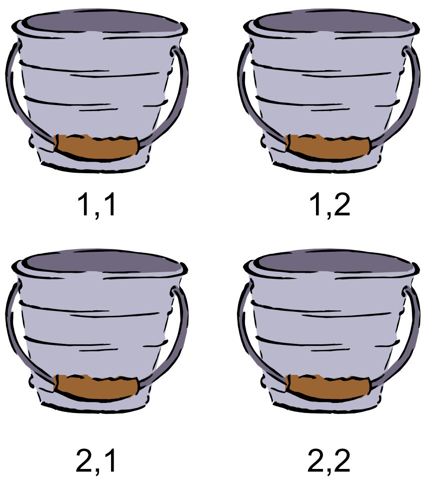
You can use a two-dimensional array to determine the windchill based on temperature and wind speed.



36

Overview of Arrays

A two-dimensional array is similar to a stack of buckets.



- SAS puts a value in a bucket based on the bucket row and column pair.
- Values are retrieved from a bucket based on the bucket row and column pair.

37 ...

Using Multidimensional Arrays

General form for the multidimensional ARRAY statement:

> **ARRAY** *array-name* {…,*rows, cols*} $ *length*
> *elements* (*initial values*);

rows

specifies the number of array elements in a row arrangement.

cols

specifies the number of array elements in a column arrangement.

Example:

```
array W{2,3} W1-W6;
```

38

The keyword _TEMPORARY_ can be used instead of *elements* to avoid creating new data set variables.

Using Multidimensional Arrays

```
array W{4,2} (-22,-16,-28,-22,-32,-26,-35,-29);
```

Temperature

Wind Speed

	-10	-5	0	5	10	15	20	25	30
5	-22	-16	-11	-5	1	7	13	19	25
10	-28	-22	-16	-10	-4	3	9	15	21
15	-32	-26	-19	-13	-7	0	6	13	19
20	-35	-29	-22	-15	-9	-2	4	11	17
25	-37	-31	-24	-17	-11	-4	3	9	16
30	-39	-33	-26	-19	-12	-5	1	8	15
35	-41	-34	-27	-21	-14	-7	0	7	14
40	-43	-36	-29	-22	-15	-8	-1	6	13

The values in the windchill table can be typed as initial values in an array named W.

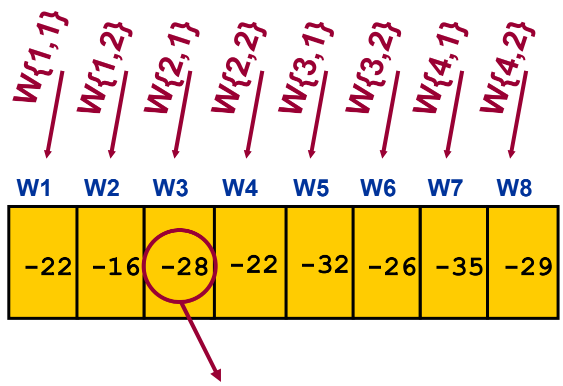
39

For this example, only the first two columns and four rows are included in the array.

The initial values fill all the columns in a row before moving on to the next row.

Using Multidimensional Arrays

```
array W{4,2} (-22,-16,-28,-22,-32,-26,-35,-29);
```



Corresponds to a wind speed of 10 and a temperature of -10

40 ...

When you use a multidimensional array, you

- must supply an index value for each dimension to process a specific array element
- can use a DO loop to process elements in a given dimension
- use nested DO loops to process elements in more than one dimension.

Flights Data

Find the windchill for the flights based on the temperature and wind speed.

First Two Observations of **`ia.flights`**

```
          ia.flights

flight      temp      wspeed

 IA2736       -8           9
 IA6352       -4          16
```

41

Desired Results

wndchill

Flight	Temp	WSpeed	Chill
IA2736	-8	9	-28
IA6352	-4	16	-26

ia.flights

constants loaded into an array

42

Using Multidimensional Arrays

If you want to know the windchill based on wind speed and temperature, you can use the array as a table lookup.

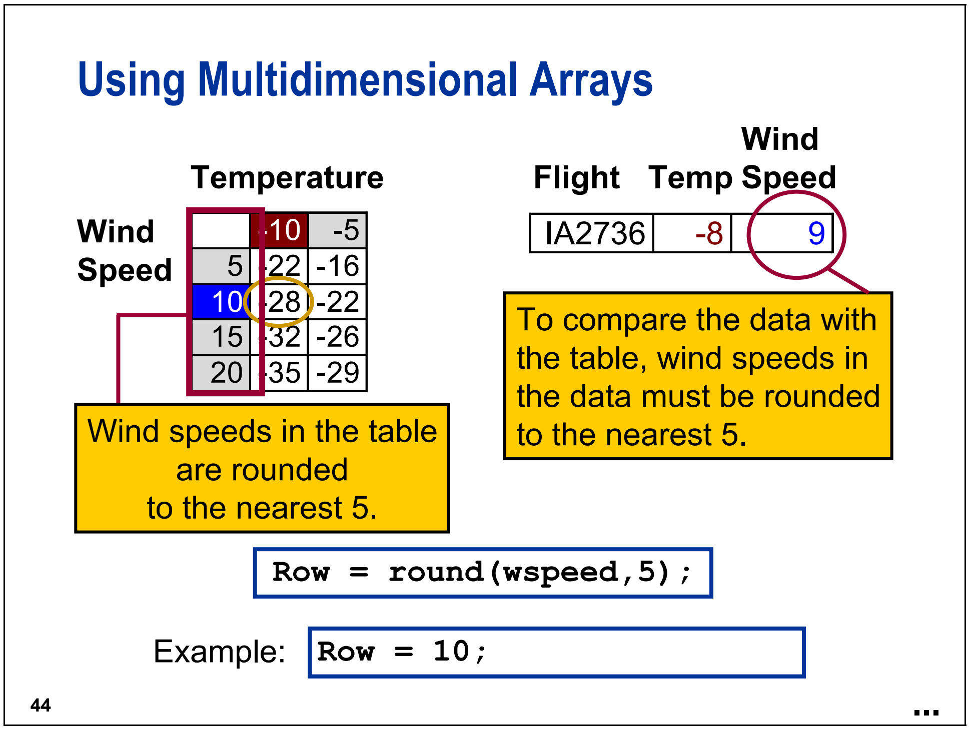
Temperature

Wind Speed

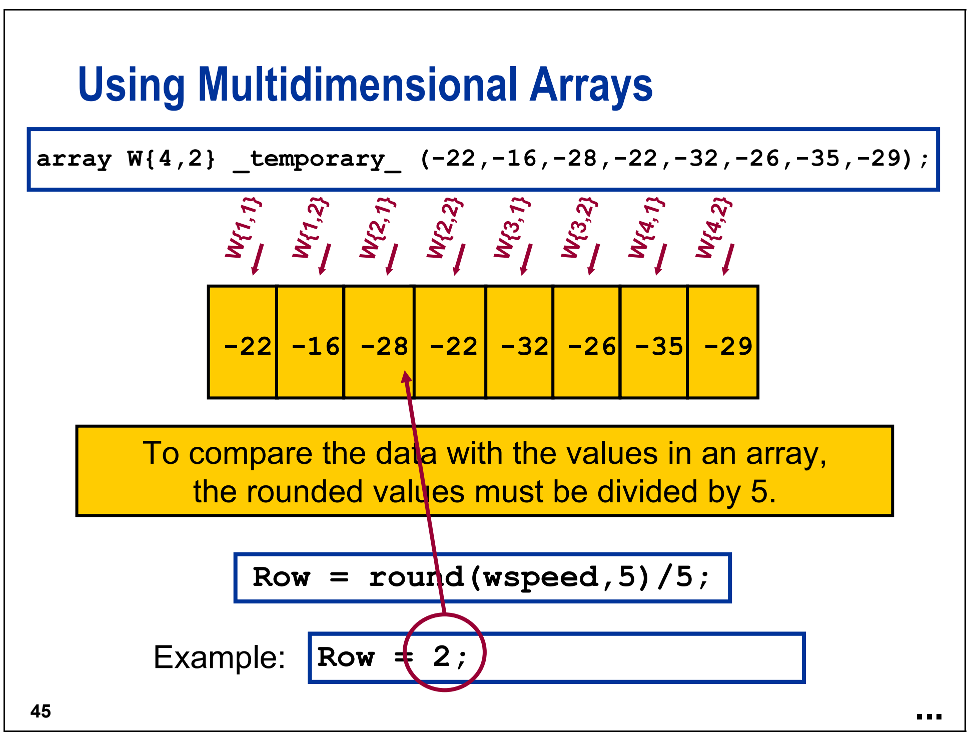
	-10	-5
5	-22	-16
10	-28	-22
15	-32	-26
20	-35	-29

Flight	**Temp**	**Wind Speed**
IA2736	-8	9

43 ...



Using Multidimensional Arrays
Temperature
Wind Speed
-10 -5
5 -22 -16
10 -28 -22
15 -32 -26
20 -35 -29
Flight Temp Wind Speed
IA2736 -8 9
To compare the data with the table, wind speeds in the data must be rounded to the nearest 5.
Wind speeds in the table are rounded to the nearest 5.
Row = round(wspeed,5);
Example: Row = 10;
44
...



Using Multidimensional Arrays
array W{4,2} _temporary_ (-22,-16,-28,-22,-32,-26,-35,-29);
W{1,1} W{1,2} W{2,1} W{2,2} W{3,1} W{3,2} W{4,1} W{4,2}
-22 -16 -28 -22 -32 -26 -35 -29
To compare the data with the values in an array, the rounded values must be divided by 5.
Row = round(wspeed,5)/5;
Example: Row = 2;
45
...

Using Multidimensional Arrays

Temperature

Wind Speed	-10	-5
5	-22	-16
10	-28	-22
15	-32	-26
20	-35	-29

Flight	Temp	Wind Speed
IA2736	-8	9

Temperatures in the table are rounded to the nearest 5.

To compare the data with the table, temperatures in the data must be rounded to the nearest 5.

```
Column = round(temp,5);
```

Example:

```
Column = -10;
```

46 ...

Using Multidimensional Arrays

```
array W{4,2} _temporary_ (-22,-16,-28,-22,-32,-26,-35,-29);
```

W{1,1}	W{1,2}	W{2,1}	W{2,2}	W{3,1}	W{3,2}	W{4,1}	W{4,2}
-22	-16	-28	-22	-32	-26	-35	-29

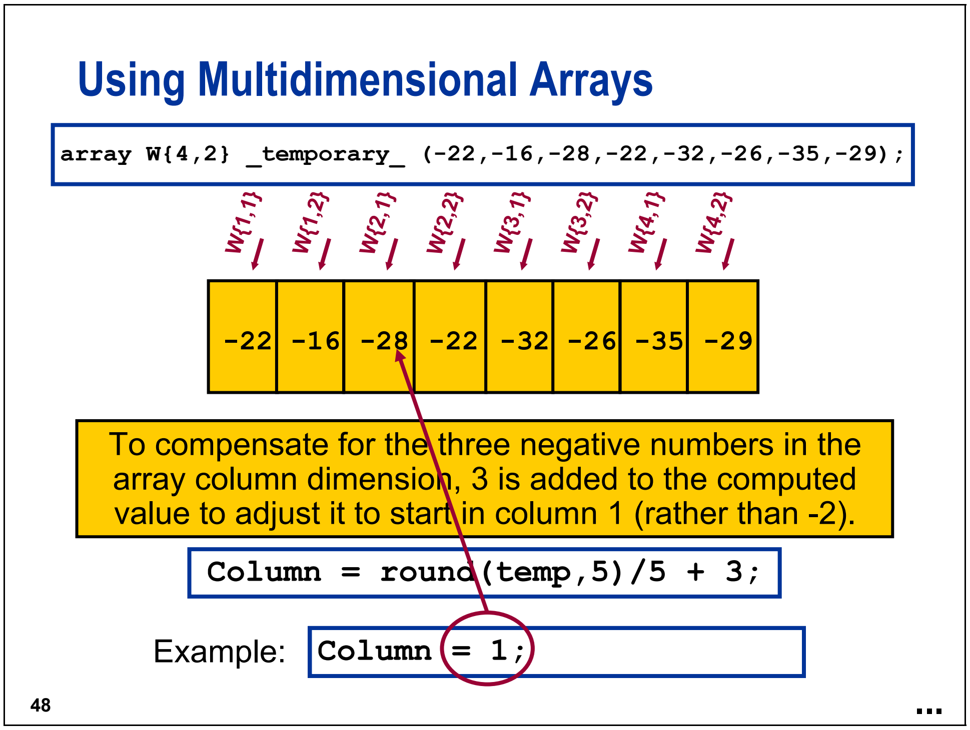
To compare the data with the values in an array, the rounded values must be divided by 5.

```
Column = round(temp,5)/5;
```

Example:

```
Column = -2;
```

47 ...


Using Multidimensional Arrays
array W{4,2} _temporary_ (-22,-16,-28,-22,-32,-26,-35,-29);
W{1,1} W{1,2} W{2,1} W{2,2} W{3,1} W{3,2} W{4,1} W{4,2}
-22 -16 -28 -22 -32 -26 -35 -29
To compensate for the three negative numbers in the array column dimension, 3 is added to the computed value to adjust it to start in column 1 (rather than -2).
Column = round(temp,5)/5 + 3;
Example: Column = 1;
48
...

Using Multidimensional Arrays

```
data wndchill(drop = Column Row);
   array W{4,2} _Temporary_
     (-22,-16,-28,-22,-32,-26,-35,-29); ❶
   set ia.flights (obs = 2);
   Row = round(WSpeed,5)/5; ❷
   Column = (round(Temp,5)/5)+3; ❸
   Chill = W{Row,Column};   ❹
run;
```

49

c04s2d2

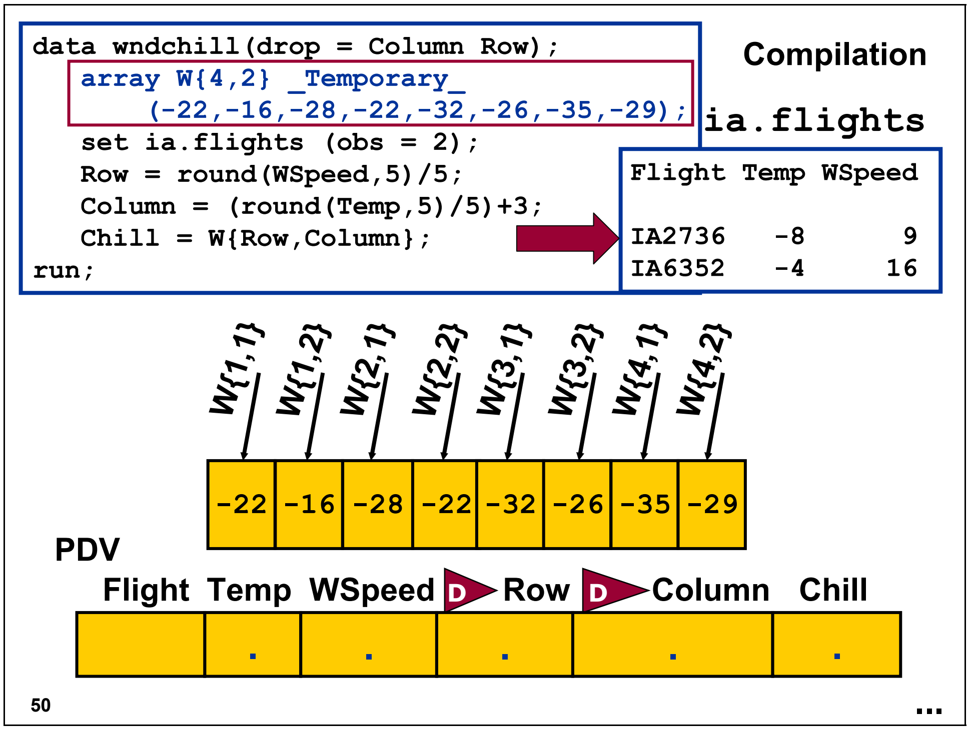
✎ In this example, **WSpeed** must be at least 2.5 and less than 22.5, and **Temp** must be at least –12.5 and less than –2.5.

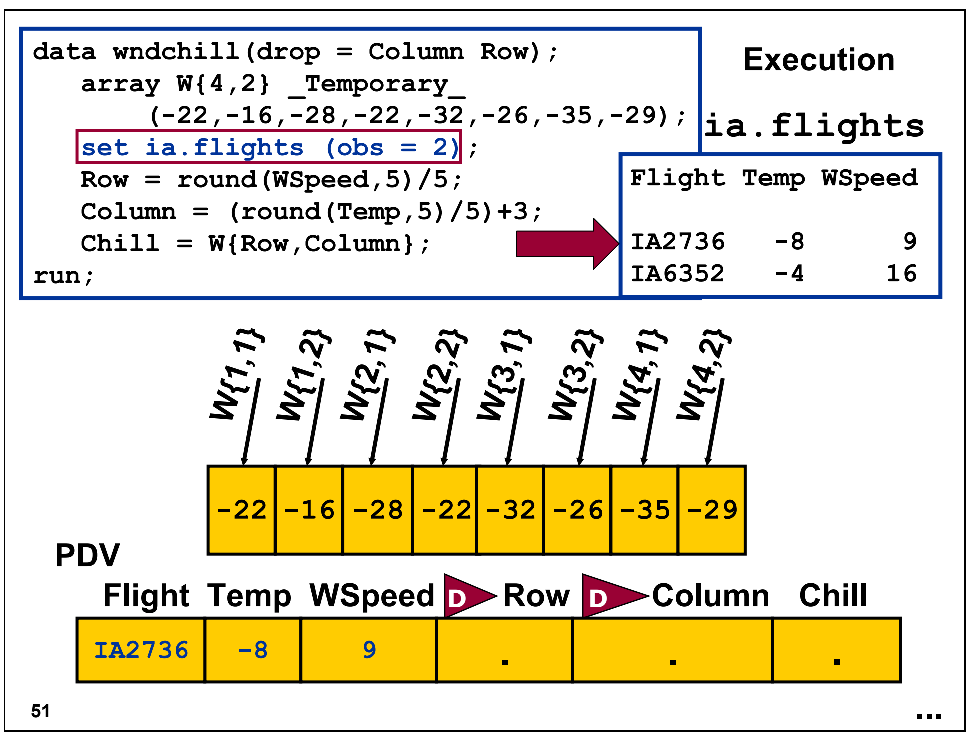
① Eight values are typed into the array initial values. The _TEMPORARY_ keyword creates a list of temporary data elements. They behave in the same way as DATA step variables except that they do not have names and they do not appear in the output data set.

② **WSpeed** is rounded to the nearest fifth unit because the lookup table only contains wind speeds rounded to every 5 units. The value is divided by 5 to derive the row position in the windchill lookup table.

③ The offset of 3 is used because the third column in the windchill lookup table represents zero degrees.

④ The **W** array is used to look up the windchill values using the **row** and **column** variables.


data wndchill(drop = Column Row);
array W{4,2} _Temporary_
(-22,-16,-28,-22,-32,-26,-35,-29);
set ia.flights (obs = 2);
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
Compilation
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{2,1} W{2,2} W{3,1} W{3,2} W{4,1} W{4,2}
-22 -16 -28 -22 -32 -26 -35 -29
PDV
Flight Temp WSpeed D Row D Column Chill
. . . . .
50
...

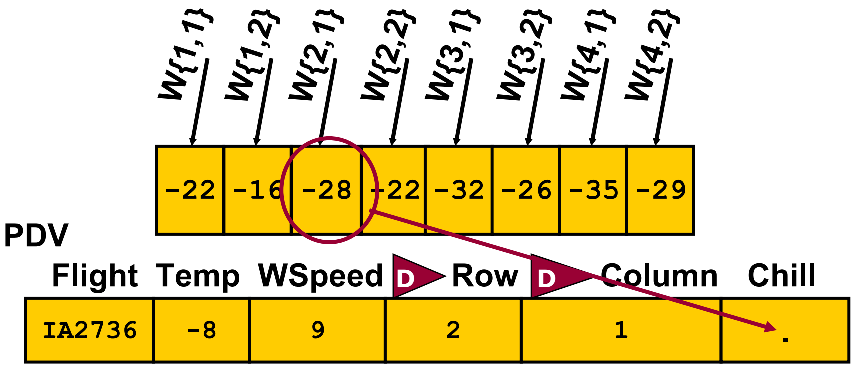

data wndchill(drop = Column Row);
array W{4,2} _Temporary_
(-22,-16,-28,-22,-32,-26,-35,-29);
set ia.flights (obs = 2);
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{2,1} W{2,2} W{3,1} W{3,2} W{4,1} W{4,2}
-22 -16 -28 -22 -32 -26 -35 -29
PDV
Flight Temp WSpeed D Row D Column Chill
IA2736 -8 9 . . .
51
...

```
data wndchill(drop = Column Row);
   array W{4,2} _Temporary_
      (-22,-16,-28,-22,-32,-26,-35,-29);
   set ia.flights (obs = 2);
   Row = round(WSpeed,5)/5;
   Column = (round(Temp,5)/5)+3;
   Chill = W{Row,Column};
run;
```

Execution

ia.flights

Flight	Temp	WSpeed
IA2736	-8	9
IA6352	-4	16



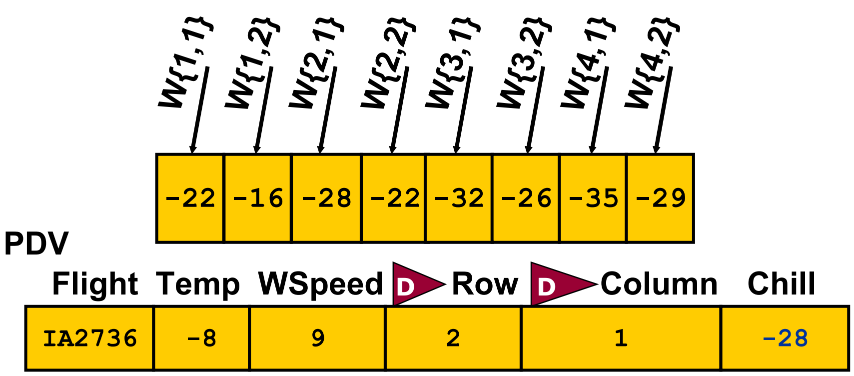
54 ...

```
data wndchill(drop = Column Row);
   array W{4,2} _Temporary_
      (-22,-16,-28,-22,-32,-26,-35,-29);
   set ia.flights (obs = 2);
   Row = round(WSpeed,5)/5;
   Column = (round(Temp,5)/5)+3;
   Chill = W{Row,Column};
run;
```

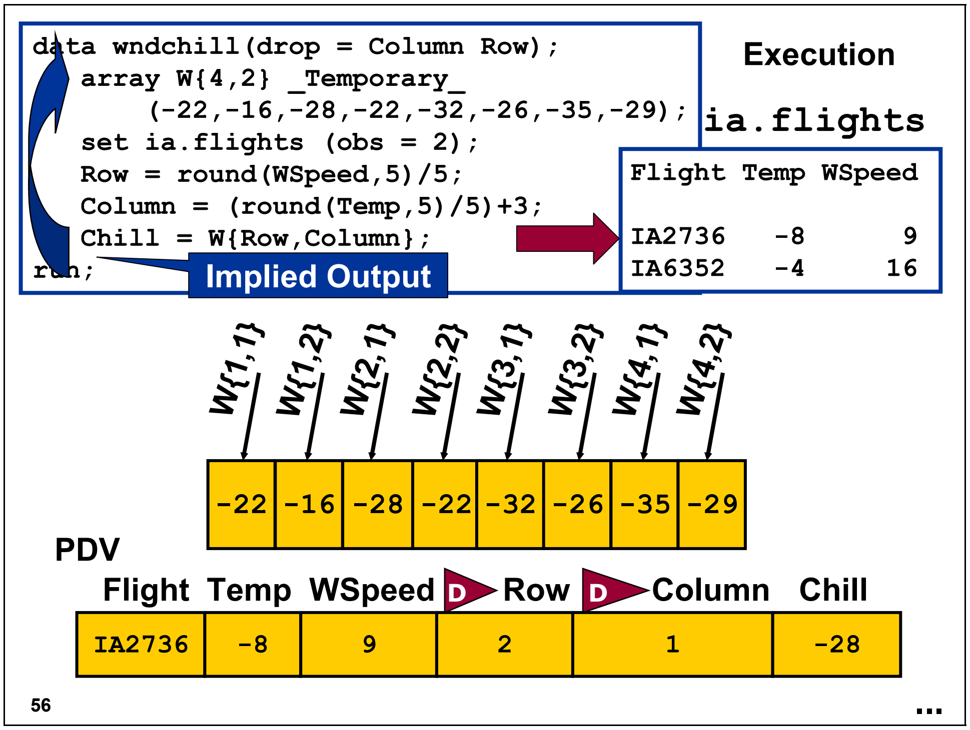
Execution

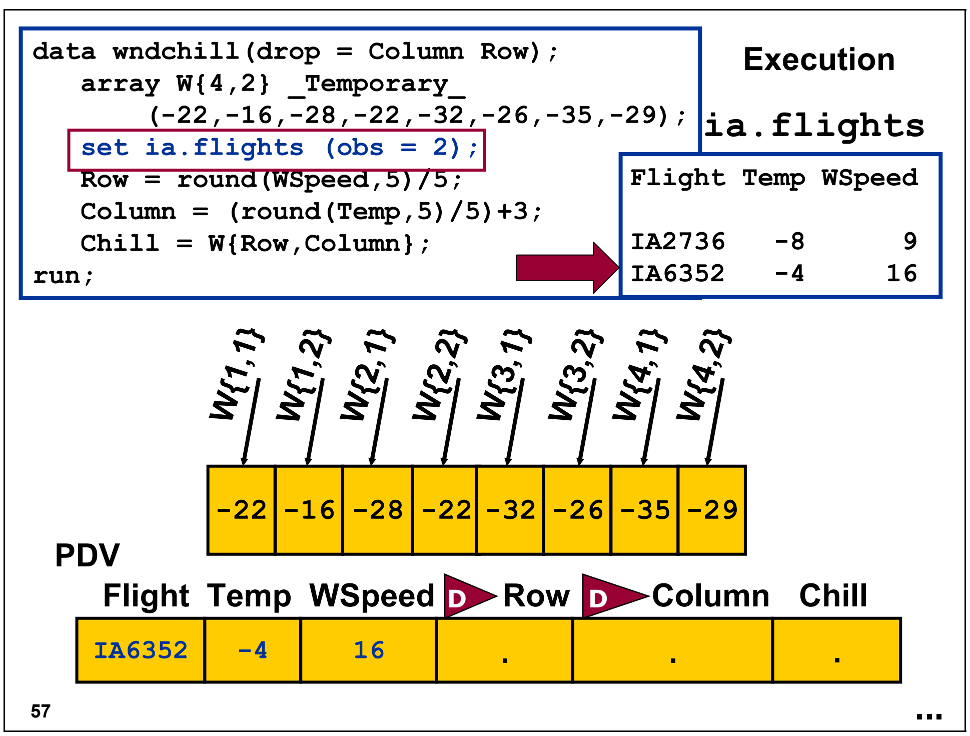
ia.flights

Flight	Temp	WSpeed
IA2736	-8	9
IA6352	-4	16



55 ...


data wndchill(drop = Column Row);
array W{4,2} _Temporary_
(-22,-16,-28,-22,-32,-26,-35,-29);
set ia.flights (obs = 2);
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
Implied Output
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{2,1} W{2,2} W{3,1} W{3,2} W{4,1} W{4,2}
-22 -16 -28 -22 -32 -26 -35 -29
PDV
Flight Temp WSpeed D Row D Column Chill
IA2736 -8 9 2 1 -28
56
...

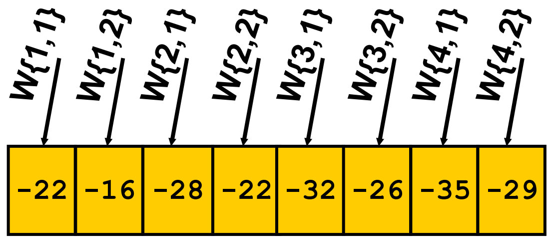

data wndchill(drop = Column Row);
array W{4,2} _Temporary_
(-22,-16,-28,-22,-32,-26,-35,-29);
set ia.flights (obs = 2);
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{2,1} W{2,2} W{3,1} W{3,2} W{4,1} W{4,2}
-22 -16 -28 -22 -32 -26 -35 -29
PDV
Flight Temp WSpeed D Row D Column Chill
IA6352 -4 16 . . .
57
...

```
data wndchill(drop = Column Row);
   array W{4,2} _Temporary_
      (-22,-16,-28,-22,-32,-26,-35,-29);
   set ia.flights (obs = 2);
   Row = round(WSpeed,5)/5;
   Column = (round(Temp,5)/5)+3;
   Chill = W{Row,Column};
run;
```

Execution

ia.flights

Flight	Temp	WSpeed
IA2736	-8	9
IA6352	-4	16



PDV

Flight	Temp	WSpeed	D Row	D Column	Chill
IA6352	-4	16	3	2	-26

58

Output

PROC PRINT Output from **work.wndchill**

wndchill

Flight	Temp	WSpeed	Chill
IA2736	-8	9	-28
IA6352	-4	16	-26

c04s2d2

59

Exercises

1. Using a Two-dimensional Array

The company recently sponsored a triathlon involving bicycling (**`EVENT = 1`**), swimming (**`EVENT = 2`**), and running (**`EVENT = 3`**). The finish order of the top four contestants in all events is stored in **`ia.compete`**. Use the following table and a two-dimensional array to determine the scores received for each event. The newly created SAS data set should be named **`results`**.

Event	1st Place	2nd Place	3rd Place	4th Place
1	65	55	45	35
2	80	70	60	50
3	70	60	50	40

Output

```
                              work.results

                   Frst
     LastName      Name       Event    Finish    Score

     Tuttle        Thomas       1         1        65
     Gomez         Alan         1         2        55
     Chapman       Neil         1         3        45
     Welch         Darius       1         4        35
     Vandeusen     Richard      2         1        80
     Tuttle        Thomas       2         2        70
     Venter        Vince        2         3        60
     Morgan        Mel          2         4        50
     Chapman       Neil         3         1        70
     Gomez         Alan         3         2        60
     Morgan        Mel          3         3        50
     Tuttle        Thomas       3         4        40
```

Using Multidimensional Arrays

Suppose the windchill values are stored in a SAS data set named **ia.wchill** where the rows represent wind speeds and the columns represent temperatures.

Obs	Neg10	Neg5	Tmp0	Tmp5	Tmp10	Tmp15	Tmp20	Tmp25	Tmp30
1	-22	-16	-11	-5	1	7	13	19	25
2	-28	-22	-16	-10	-4	3	9	15	21
3	-32	-26	-19	-13	-7	0	6	13	19
4	-35	-29	-22	-15	-9	-2	4	11	17
5	-37	-31	-24	-17	-11	-4	3	9	16
6	-39	-33	-26	-19	-12	-5	1	8	15
7	-41	-34	-27	-21	-14	-7	0	7	14
8	-43	-36	-29	-22	-15	-8	-1	6	13

You can load the array
from the values in the SAS data set.

61

Stored Array Values (Review)

Array values should be stored in a SAS data set when the following conditions exist:

- too many values to initialize easily in the array
- values changing frequently
- the same values used in many programs

62

Using Multidimensional Arrays

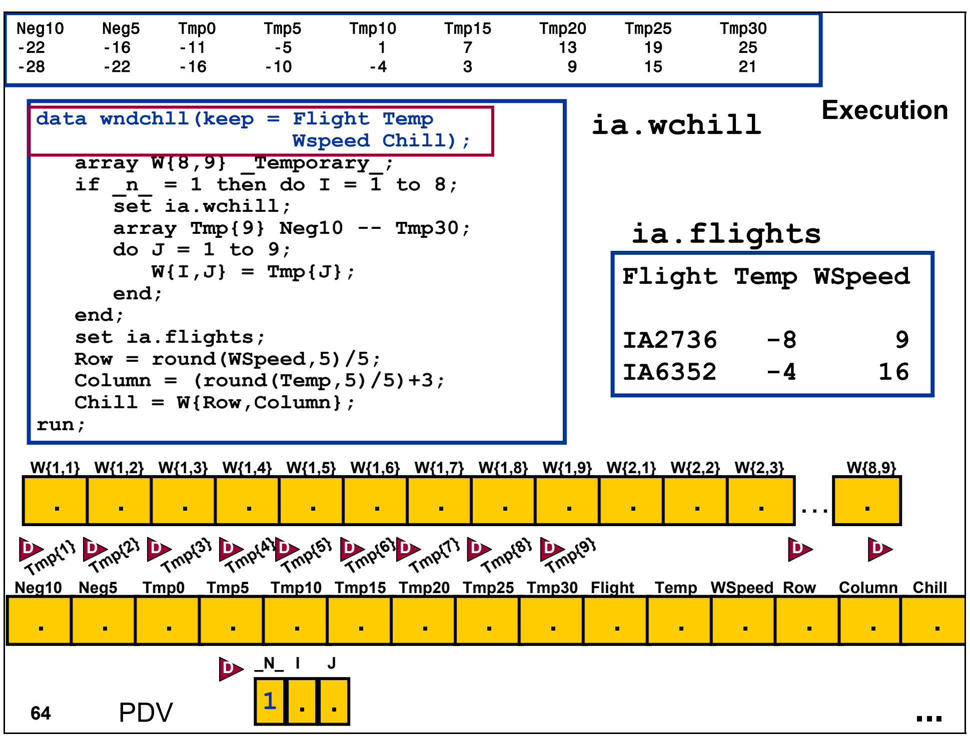
```
data wndchll(keep = Flight Temp
                    Wspeed Chill);
   array W{8,9} _Temporary_;
   if _n_ = 1 then do I = 1 to 8; ❶
      set ia.wchill;
      array Tmp{9} Neg10 -- Tmp30; ❷
      do J = 1 to 9;
         W{I,J} = Tmp{J};  ❸
      end;
   end;
   set ia.flights;
   Row = round(WSpeed,5)/5;
   Column = (round(Temp,5)/5)+3;
   Chill = W{Row,Column};
run;
```

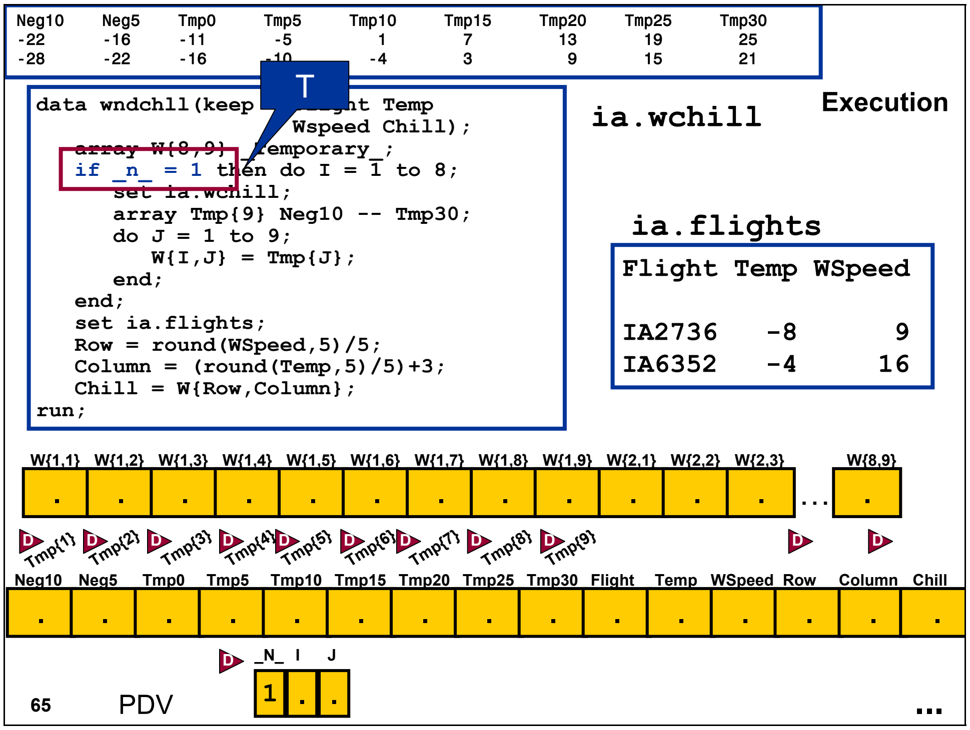
63 c04s2d3

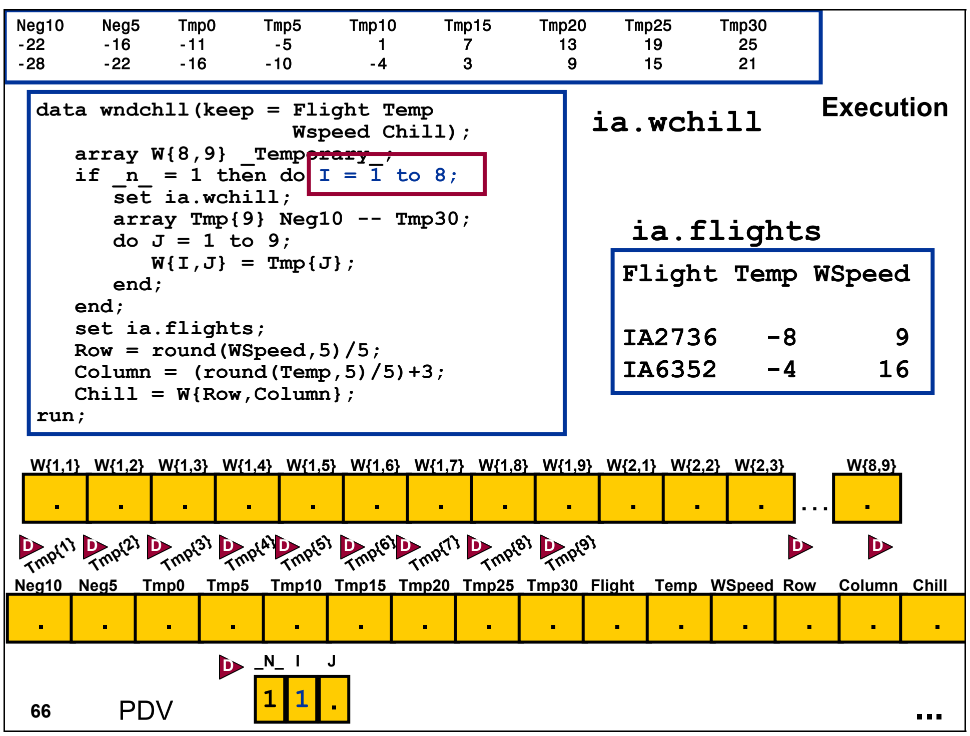
① The index variable, **I**, is used so that the SET statement is executed for each observation in **ia.wchill**.

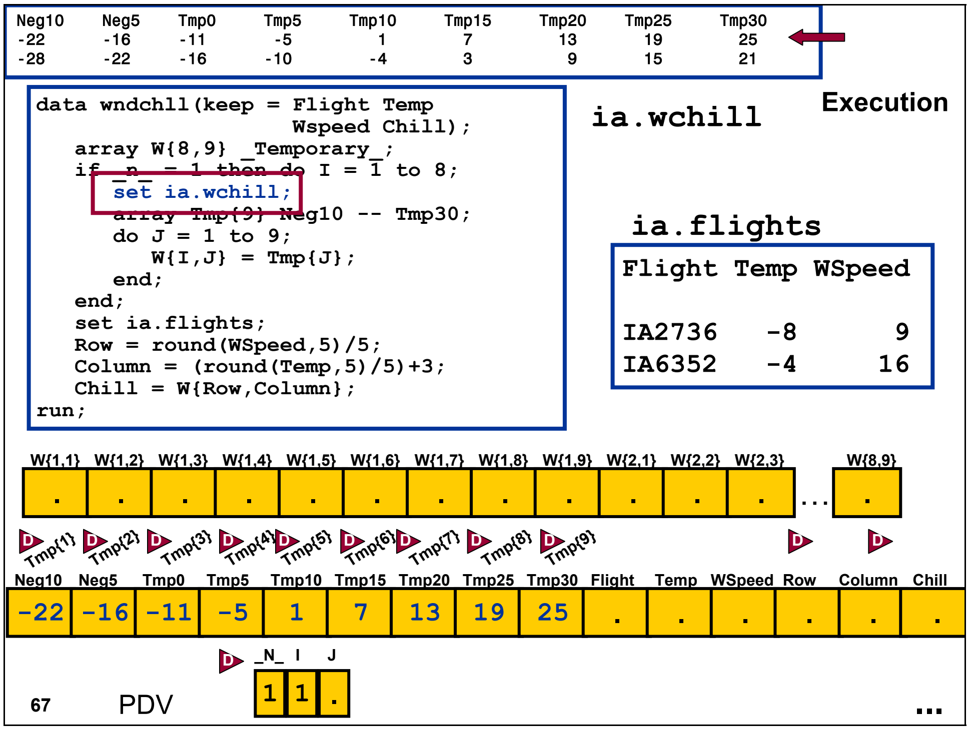
② The array, **Tmp**, is associated with the variables **Neg10** through **Tmp30**.

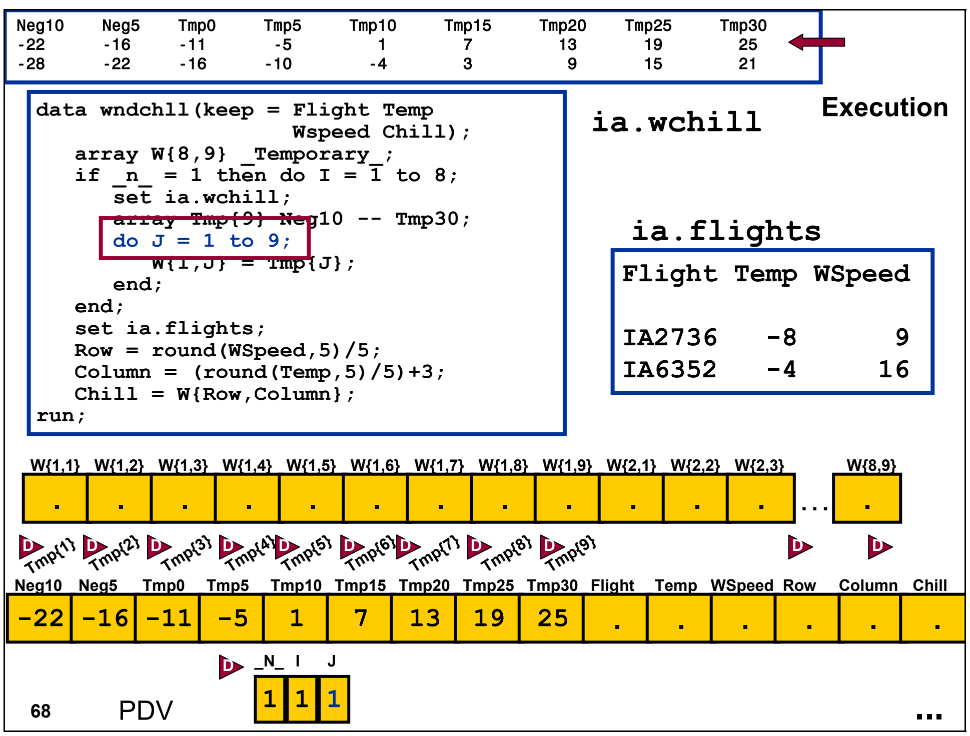
③ The two-dimensional array **W** is loaded with the values of the **Tmp** array.

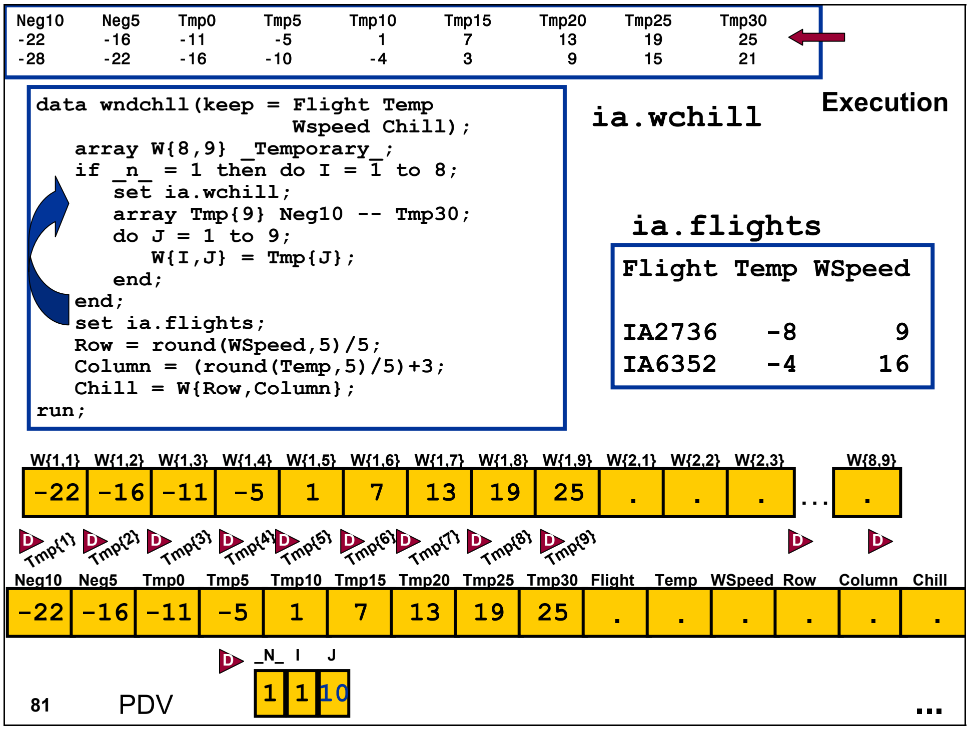


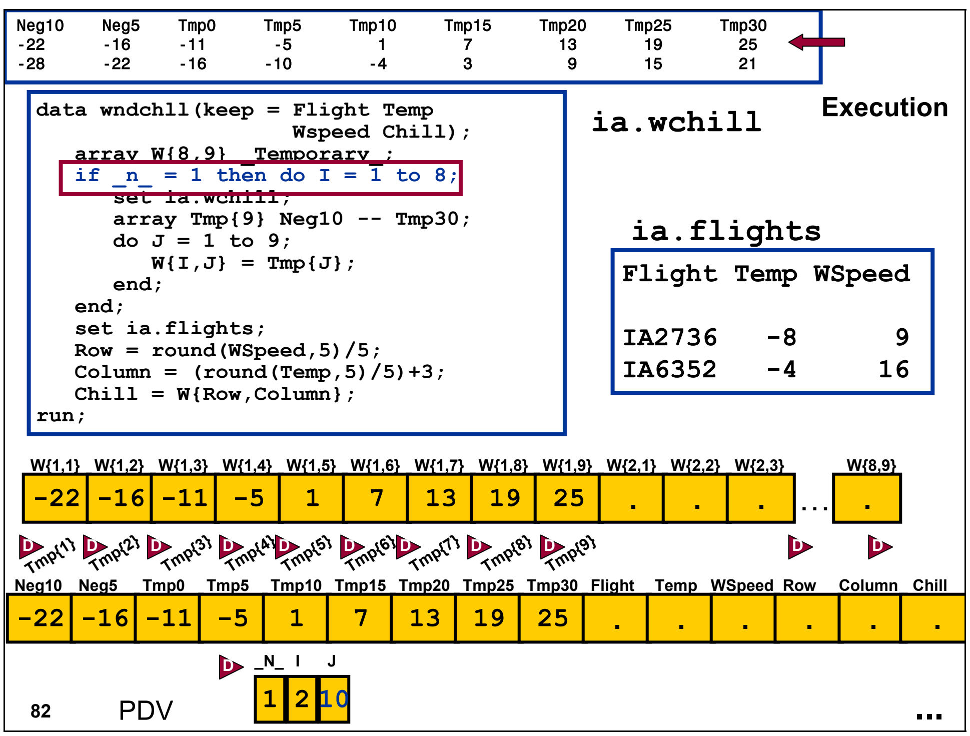

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
T
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
_N_ I J
1
65 PDV

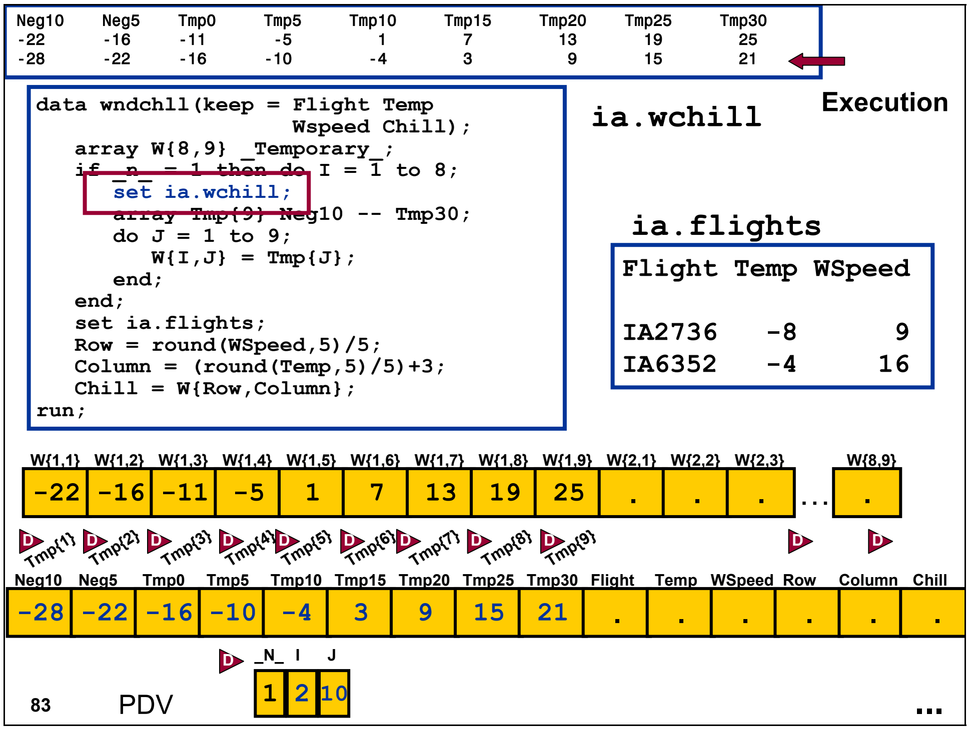

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
_N_ I J
1 1
66 PDV

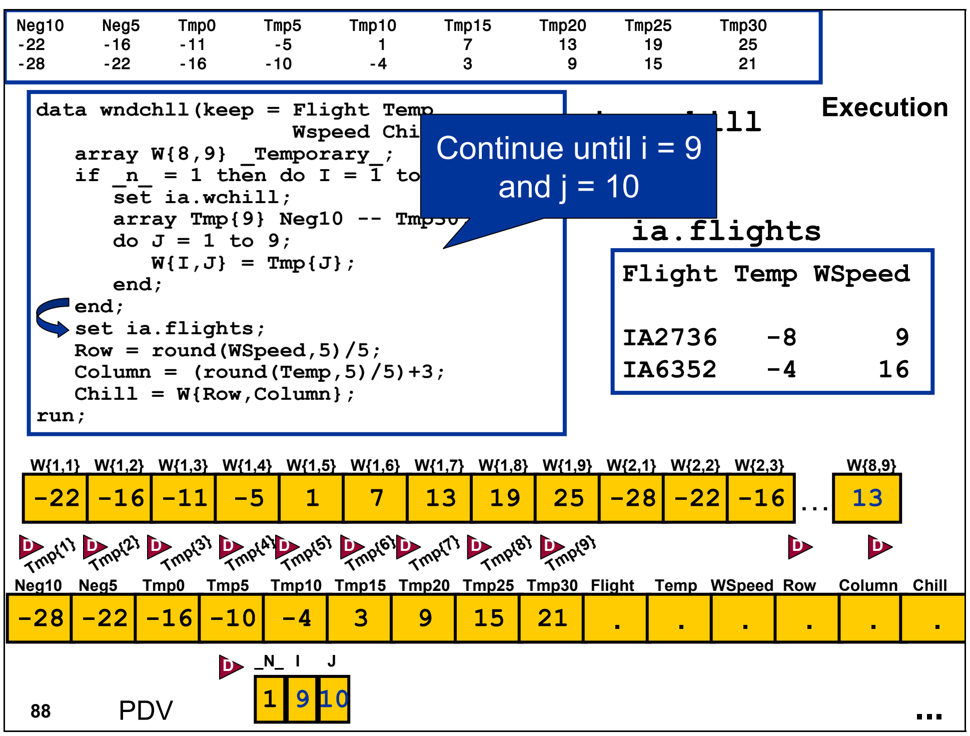

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-22 -16 -11 -5 1 7 13 19 25
_N_ I J
1 1 .
67
PDV
...

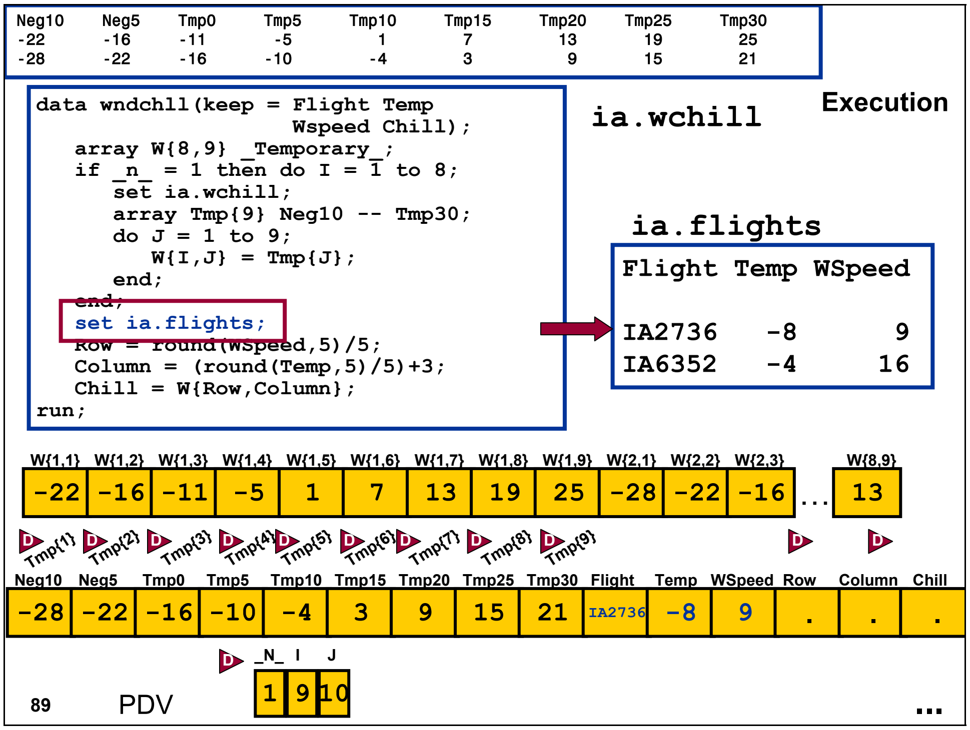

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-22 -16 -11 -5 1 7 13 19 25
_N_ I J
1 1 1
68
PDV
...

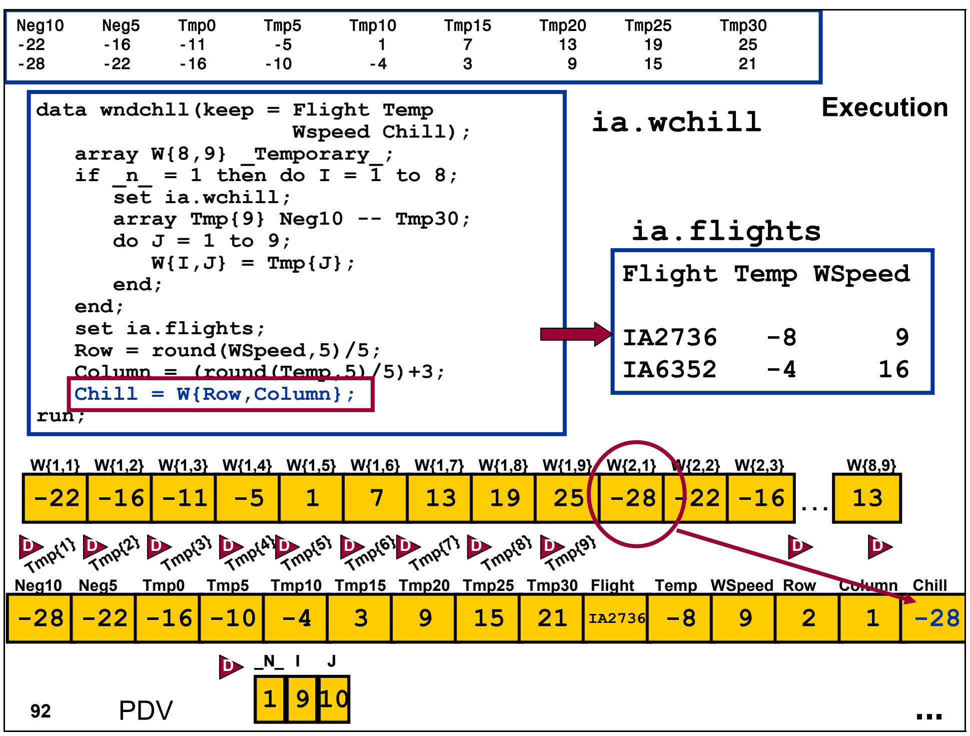

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 . . . .
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-22 -16 -11 -5 1 7 13 19 25 . . . . . .
_N_ I J
1 1 10
81
PDV
...

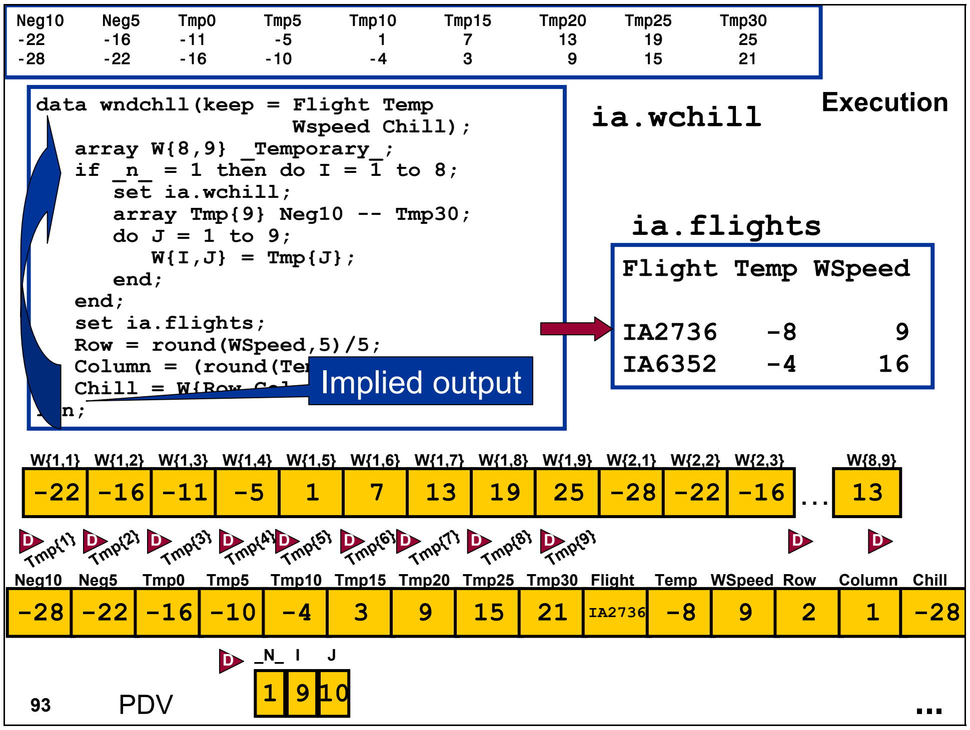

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 . . . .
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-22 -16 -11 -5 1 7 13 19 25 . . . . . .
_N_ I J
1 2 10
82
PDV
...

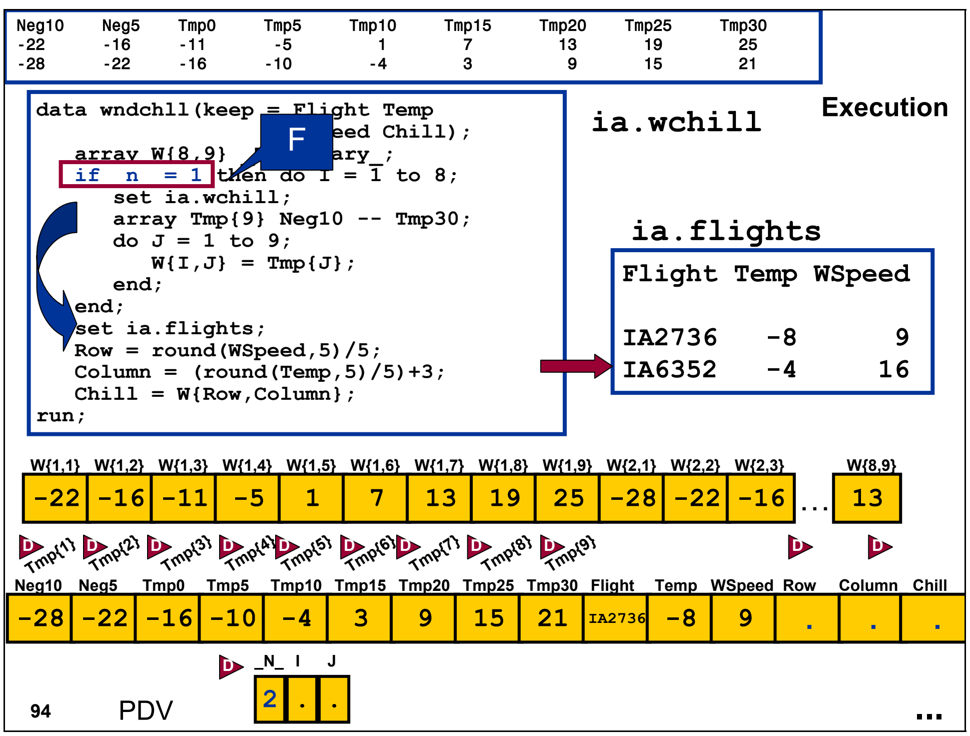

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 . . . .
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 . . . . . .
_N_ I J
1 2 10
83
PDV
...

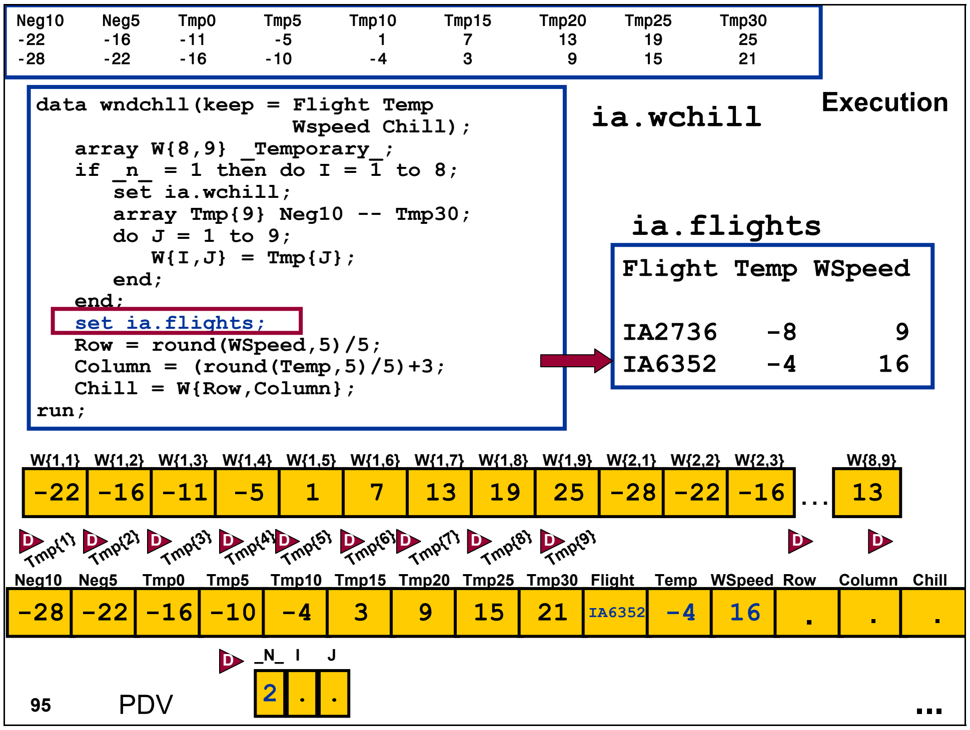

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chi
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
Continue until i = 9
and j = 10
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 . . . . . .
_N_ I J
1 9 10
88
PDV
...

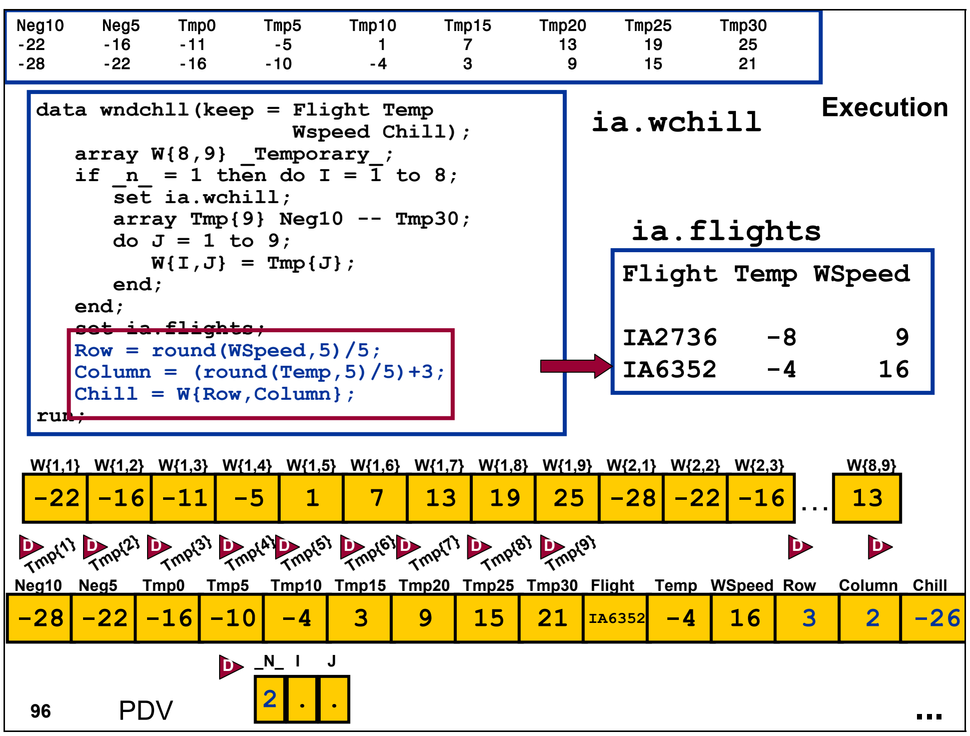

Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 IA2736 -8 9 . . .
_N_ I J
1 9 10
89
PDV
...


Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 IA2736 -8 9 2 1 -28
_N_ I J
1 9 10
92
PDV
...


Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Te
Chill = W{Row,Col
n;
Implied output
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 IA2736 -8 9 2 1 -28
_N_ I J
1 9 10
93
PDV
...


Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
eed Chill);
array W{8,9}
ary_;
if n = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
F
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 IA2736 -8 9 . . .
_N_ I J
2 . .
94
PDV
...


Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 IA6352 -4 16 . . .
_N_ I J
2 . .
95 PDV
...


Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30
-22 -16 -11 -5 1 7 13 19 25
-28 -22 -16 -10 -4 3 9 15 21
data wndchll(keep = Flight Temp
Wspeed Chill);
array W{8,9} _Temporary_;
if _n_ = 1 then do I = 1 to 8;
set ia.wchill;
array Tmp{9} Neg10 -- Tmp30;
do J = 1 to 9;
W{I,J} = Tmp{J};
end;
end;
set ia.flights;
Row = round(WSpeed,5)/5;
Column = (round(Temp,5)/5)+3;
Chill = W{Row,Column};
run;
ia.wchill
Execution
ia.flights
Flight Temp WSpeed
IA2736 -8 9
IA6352 -4 16
W{1,1} W{1,2} W{1,3} W{1,4} W{1,5} W{1,6} W{1,7} W{1,8} W{1,9} W{2,1} W{2,2} W{2,3} W{8,9}
-22 -16 -11 -5 1 7 13 19 25 -28 -22 -16 ... 13
Tmp{1} Tmp{2} Tmp{3} Tmp{4} Tmp{5} Tmp{6} Tmp{7} Tmp{8} Tmp{9}
Neg10 Neg5 Tmp0 Tmp5 Tmp10 Tmp15 Tmp20 Tmp25 Tmp30 Flight Temp WSpeed Row Column Chill
-28 -22 -16 -10 -4 3 9 15 21 IA6352 -4 16 3 2 -26
_N_ I J
2 . .
96 PDV
...

Advantages of an Array

Advantages of using an array include the following:

- use of positional order
- use of multiple values to determine the array element to be returned
- ability to use a non-sorted and non-indexed base data set
- use of numeric mathematical expressions to determine which element of the array to be looked up; exact match not required

An array will always be faster than a hash object or format if you can use it.

98

Disadvantages of an Array

Disadvantages of using an array include the following:

- memory requirements to load the entire array
- requirement that you must have a numeric value as pointer to the array elements
- return of only a single value from the lookup operation
- dimensions supplied at compile time by either hard-coding or macro variables

An array requests a contiguous chunk of memory requested at compile time.

99

Exercises

2. Loading an Array from a SAS Data Set

The company recently sponsored a triathlon involving bicycling (**EVENT = 1**), swimming (**EVENT = 2**), and running (**EVENT = 3**). The finish order of the top four contestants in all events is stored in **ia.compete**. Use the **ia.events** data set, which contains the points awarded for each event and finish, and a two-dimensional array to determine the scores received for each event. The newly created SAS data set should be named **results**.

Output

```
                                  work.results

                      Frst
         LastName     Name       Event    Finish    Score

         Tuttle       Thomas       1         1        65
         Gomez        Alan         1         2        55
         Chapman      Neil         1         3        45
         Welch        Darius       1         4        35
         Vandeusen    Richard      2         1        80
         Tuttle       Thomas       2         2        70
         Venter       Vince        2         3        60
         Morgan       Mel          2         4        50
         Chapman      Neil         3         1        70
         Gomez        Alan         3         2        60
         Morgan       Mel          3         3        50
         Tuttle       Thomas       3         4        40
```

3. Loading an Array from a SAS Data Set (Optional)

The **ia.mealplan** data set contains information on which meals, if any, are served on flights. Meal service is based on the day of the week (1 to 7), **DOW**, and the hour of the day of the flight, **Hour**.

a. Produce a SAS data set named **meals** that contains the meal service code for each flight.

b. Use **ia.schedule** to obtain the flight information.

c. Create a two-dimensional array from **ia.mealplan**.

d. Look up the meal for each flight using the WEEKDAY function on **Date** and the HOUR function on **Depart**.

> ✎ The HOUR function returns values between 0 and 23. The **Hour** variable in **ia.mealplan** contains the values 1 to 24.

e. Print only the first 15 observations. The expected output is below.

Output

```
                          meals

  Obs    flight     depart         date    Service

    1    IA10800      6:35    01JUN2000    Breakfast
    2    IA10801      9:35    01JUN2000    None
    3    IA10802     12:35    01JUN2000    Snack
    4    IA10803     15:35    01JUN2000    None
    5    IA10804     18:35    01JUN2000    Dinner
    6    IA10805     21:35    01JUN2000    None
    7    IA10800      6:35    02JUN2000    Breakfast
    8    IA10801      9:35    02JUN2000    Snack
    9    IA10802     12:35    02JUN2000    Lunch
   10    IA10803     15:35    02JUN2000    Snack
   11    IA10804     18:35    02JUN2000    Dinner
   12    IA10805     21:35    02JUN2000    None
   13    IA10800      6:35    03JUN2000    Breakfast
   14    IA10801      9:35    03JUN2000    Snack
   15    IA10802     12:35    03JUN2000    Lunch
```

4.3 Using Hash Objects as Lookup Tables

Objectives

- Define the DATA step hash object.
- Use the hash object as a lookup table.
- Use the hash object to match records.

102

DATA Step Hash Objects

The DATA step hash object

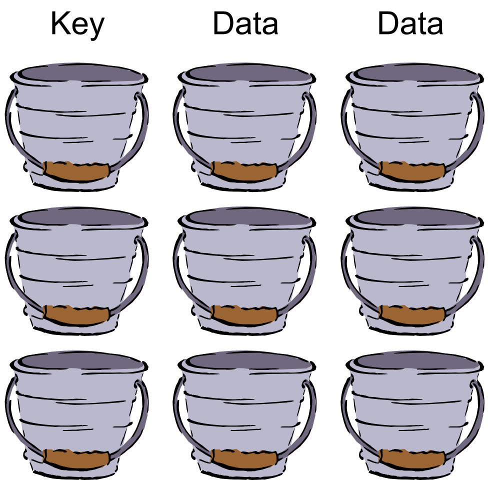
- provides in-memory data storage and retrieval
- has a data component and a key component
- uses the key for quick data retrieval
- can store multiple data items per key
- does not require the data to be sorted
- is sized dynamically.

The hash object is a good choice for lookups using unordered data that can fit into memory.

103

Overview of a Hash Object

A hash object is similar to stacks of buckets that are referred to by the value of a key.



- SAS puts value(s) in the **data** bucket(s) based on the value(s) in the **key** bucket.
- Value(s) are retrieved from the **data** bucket(s) based on the value(s) in the **key** bucket.

104

...

DATA Step Hash Objects

The hash object

- resembles a table with rows and columns
- might have numeric columns and character columns
- can be loaded from hard-coded values
- can be loaded from a SAS data set
- exists for the duration of the DATA step.

105

DATA Step Hash Objects

The data component

- can contain multiple data values per key value
- can consist of numeric and character values.

The key component

- might consist of numeric and character values
- maps key values to data rows
- must be unique
- can be composite.

Data and keys are DATA step variables.

106

Using Hash Objects

The data set **`ia.Contrib`** contains quarterly contributions to a retirement fund. Calculate the difference between the actual contribution and the goal amount.

`ia.Contrib`
(Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35
E00367	qtr2	48
E00367	qtr3	40
E00367	qtr4	30
E00441	qtr1	.
E00441	qtr2	63

Quarter	Goal amount
1	10
2	15
3	5
4	15

107

Using Hash Objects for Table Lookups

When a lookup operation depends on one or more key values, you can use a hash object.

ia.Contrib
(Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35
E00367	qtr2	48
E00367	qtr3	40
E00367	qtr4	30
E00441	qtr1	.
E00441	qtr2	63

hash object (single key)

Key	Data
qtr1	10
qtr2	15
qtr3	5
qtr4	15

108

Using Hash Objects for Table Lookups

The hash object is the lookup table.

ia.Contrib
(Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35
E00367	qtr2	48
E00367	qtr3	40
E00367	qtr4	30
E00441	qtr1	.
E00441	qtr2	63

hash object (single key)

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

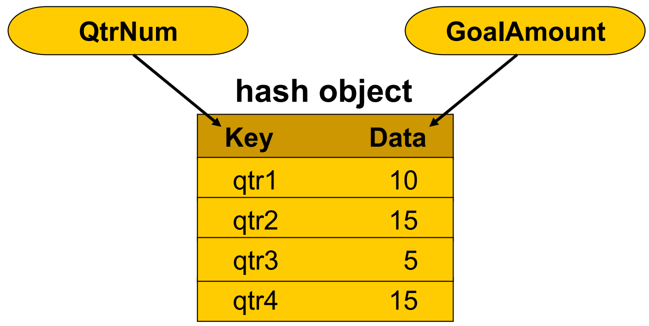
Actual contribution
– Goal amount
Calculate the difference

109

A set of lookup values can be stored in a hash object. Whereas an array uses a series of consecutive integers to address array elements, a hash object can use any combination of numeric and character values as addresses.

Using a Hash Object as Lookup Table

Load the goal amounts into a hash object.



110

Creating a Hash Object

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

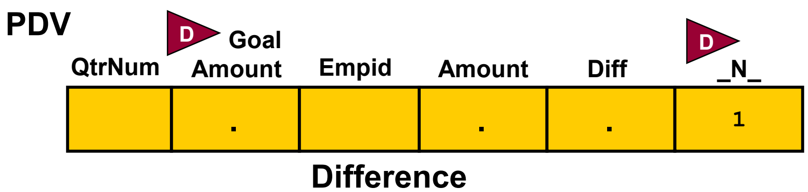
111 c04s3d1

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

T

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35



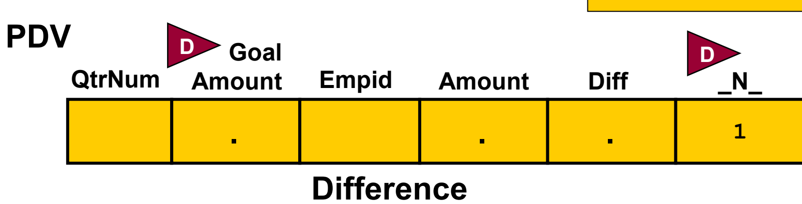
112 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum



113 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount

PDV

QtrNum	D Goal Amount	Empid	Amount	Diff	D _N_
	.		.	.	1

Difference

114 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount

Prevents the Note in the Log:

```
NOTE: Variable GoalAmount is uninitialized.
```

PDV

QtrNum	D Goal Amount	Empid	Amount	Diff	D _N_
	.		.	.	1

Difference

115 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	(D) GoalAmount	Empid	Amount	Diff	(D) _N_
	.		.	.	1

Difference

119 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	(D) GoalAmount	Empid	Amount	Diff	(D) _N_
qtr1	.	E00224	12	.	1

Difference

120 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	Goal Amount (D)	Empid	Amount	Diff	_N_ (D)
qtr1	.	E00224	12	.	1

Difference

121 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	Goal Amount (D)	Empid	Amount	Diff	_N_ (D)
qtr1	10	E00224	12	.	1

Difference

122 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	D Goal Amount	Empid	Amount	Diff	D _N_
qtr1	10	E00224	12	2	1

Difference

123 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

Implied Output

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	D Goal Amount	Empid	Amount	Diff	D _N_
qtr1	10	E00224	12	2	1

Difference

124 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

F

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	D GoalAmount	Empid	Amount	Diff	D _N_
qtr1	.	E00224	12	.	2

Difference

125 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	D GoalAmount	Empid	Amount	Diff	D _N_
qtr2	.	E00224	33	.	2

Difference

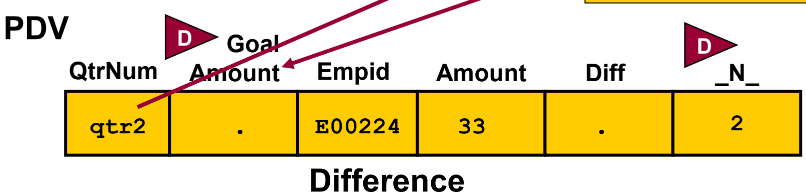
126 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15



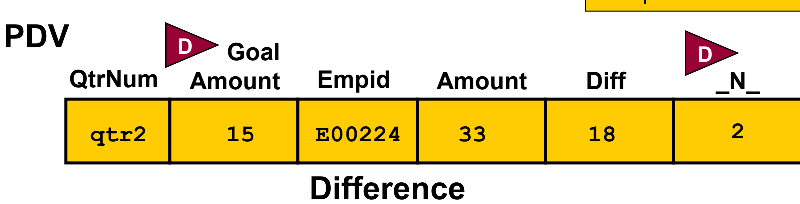
127 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15



129 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount);
      Goal.add(key:'qtr1', data:10 );
      Goal.add(key:'qtr2', data:15 );
      Goal.add(key:'qtr3', data: 5 );
      Goal.add(key:'qtr4', data:15 );
   end;
   set ia.Contrib;
   Goal.find();
   Diff = Amount - GoalAmount;
run;
```

Implied Output

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	Goal Amount (D)	Empid	Amount	Diff	_N_ (D)
qtr2	15	E00224	33	18	2

Difference

130 ...

```
data Difference(drop = GoalAmount);
   length GoalAmount 8;
   if _N_ = 1 then do;
      declare hash Goal();
      Goal.definekey ("QtrNum");
      Goal.definedata("GoalAmount");
      Goal.definedone();
      call missing(GoalAmount)
      Goal.add(key:'qtr1', da  :10 );
      Goa  add(key:'q  r2', d  :15 );
      Goal.   (key:'  3',   a: 5 );
      Goal.ad      :'             ;
   end;
   set ia.Co
   Goal.f
   Diff = A
run;
```

Continues to execute until the DATA step encounters the end of the file.

ia.Contrib (Partial Listing)

EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35

Key: QtrNum	Data: GoalAmount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

PDV

QtrNum	Goal Amount (D)	Empid	Amount	Diff	_N_ (D)
qtr2	15	E00224	33	18	2

Difference

131 ...

The `Difference` Data Set

Partial Output

EmpID	Qtr Num	Amount	Diff
E00224	qtr1	12	2
E00224	qtr2	33	18
E00224	qtr3	22	17
E00224	qtr4	.	.
E00367	qtr1	35	25
E00367	qtr2	48	33
E00367	qtr3	40	35
E00367	qtr4	30	20
E00441	qtr1	.	.
E00441	qtr2	63	48

132

Using Hash Objects

The DATA step hash object can be defined as follows:

- is a DATA step component object
- has attributes and methods
- is created with a DECLARE statement
- is manipulated with object dot syntax

An *attribute* is a property. A *method* is a function.

133

✎ When a DATA step hash object is created, it is said to be *instantiated*.

Declaring a Hash Object

```
declare hash Goal();
```

General form for the DECLARE statement:

DECLARE *object variable (<arg_tag-1: value-1 <,…arg_tag-n: value-n>>)*;

object specifies the component object.

variable specifies the variable name for the component object.

arg_tag specifies the information that is used to create an instance of the component object.

value specifies the value for an argument tag.

134

Declaring a Hash Object

Valid values for *object* are as follows:

hash indicates a hash object.

hiter indicates a hash iterator object.

The hash iterator object retrieves data from the hash object in ascending or descending key order.

135

Valid values for the *argument_tag* depend on the component object.

Hash Object Argument Tags

Argument_tag	Value Description
dataset : *'dataset_name'*	The name of a SAS data set to load into the hash object
hashexp : *n*	The hash object's table size, where the size of the hash table is 2^n (default n = 8, max n= 16)
ordered : 'NO' | 'ascending' | 'descending' | 'YES' | 'Y'	The sort order for the OUTPUT method or the iterator object (default = 'NO')

136

The table in a hash object is an array of buckets. The default hash table size (the default number of buckets) is 256 (2^8) and the maximum size is 65,536 (2^{16}). When multiple key values hash to the same index (same bucket), the key values are stored in a binary tree in the bucket for rapid retrieval. The size of the tree is limited only by the available memory.

Declaring a Hash Object

```
declare hash Goal();
```

creates a hash object named **Goal**.

```
declare hash Goal(dataset: 'ia.ideal');
```

creates the **Goal** hash object and loads it from **ia.ideal**.

```
declare hash Goal(hashexp: 10,
                  ordered: 'ascending');
```

creates the **Goal** hash object, assigns a size, and specifies a return order.

The DECLARE statement is an executable statement.

137

Using Object Dot Syntax

```
Goal.definekey ("QtrNum");
Goal.definedata("GoalAmount");
Goal.definedone();
```

General form for object dot **method** syntax:

OBJECT.METHOD(<*arg_tag-1*: *value-1*<
,…*arg_tag-n*: *value-n*>>);

object	name of the object
method	method to invoke
arg_tag	name of an argument to be passed
value	value of the argument

138

Defining Key and Data Variables

Use the DEFINEKEY, DEFINEDATA, and DEFINEDONE methods to specify variables that hold the hash object's key and data values.

```
Goal.definekey ("QtrNum");
Goal.definedata("GoalAmount");
Goal.definedone();
```

The DEFINEDONE method must be called to complete the initialization of the hash object.

139

Selected hash object methods include the following:

DEFINEKEY	defines key variables for the hash object.
DEFINEDATA	defines data variables for the hash object.
DEFINEDONE	completes the initialization of the hash object.
ADD	adds key and data values to the hash object.
FIND	searches the hash object for a key value, and returns a zero if successful.
OUTPUT	outputs the hash object’s data values to a SAS data set.
DELETE	deletes a hash object.
REPLACE	replaces the data for a key in the hash object.
REMOVE	removes a key and its associated data from the hash object.

For more information on using the DATA step object attributes and methods, see "Using DATA Step Component Objects" in the DATA Step Contents section of the SAS Language Reference: Concepts chapter of the SAS documentation for SAS®9.

Loading Key and Data Values

Use the ADD method to load key and data values into the hash object.

```
Goal.add(key:'qtr1', data:10 );
Goal.add(key:'qtr2', data:15 );
Goal.add(key:'qtr3', data: 5 );
Goal.add(key:'qtr4', data:15 );
```

140

Retrieving Matching Data

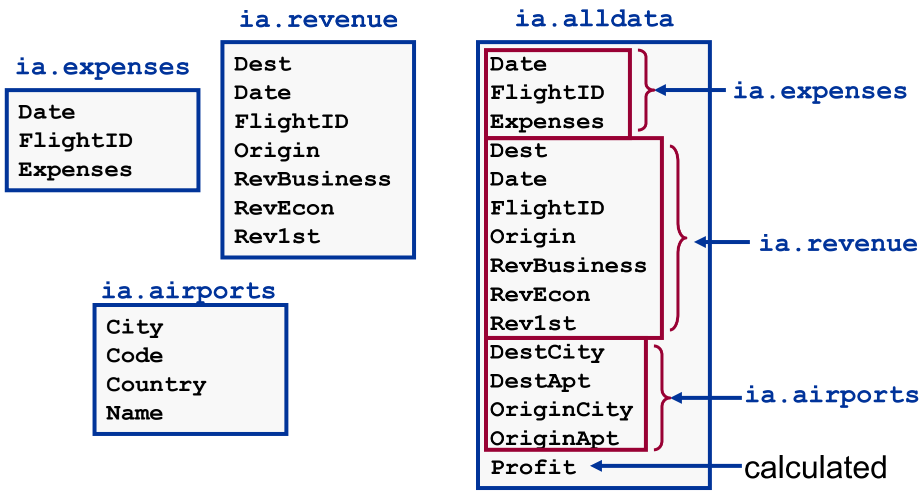
Use the FIND method to retrieve matching data from the hash object.

```
Goal.find();
```

141

Business Task

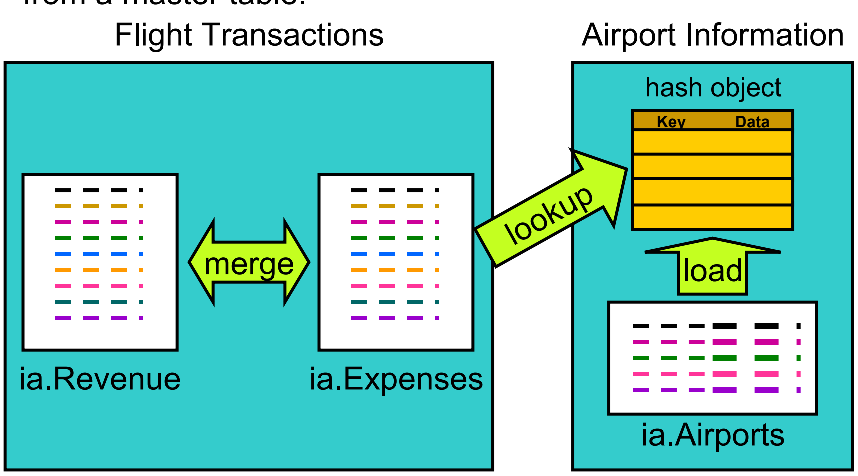
Combine three data sets to create a report showing revenues, expenses, profits, and airport information.



142 ...

Using Hash Objects for Table Lookups

You can use a hash object to retrieve matching records from a master table.



143

Using a Hash Object as Lookup Table

Load the **ia.airport** data set into a hash object.

Code **City** **Name**

hash object

Key	Data	Data
AKL	Auckland	International
AMS	Amsterdam	Schiphol
ANC	Anchorage	Anchorage International Airport
ARN	Stockholm	Arlanda
ATH	Athens	Hellinikon International Airport
BHM	Birmingham	Birmingham International Airport

144

Preview of Program

```
data Alldata_hash;
   if _N_ = 1 then do;
      if 0 then
         set ia.Airports(keep=Code City Name);
      declare hash airports(dataset: "ia.Airports");
      airports.definekey ("Code");
      airports.definedata("City", "Name");
      airports.definedone();
   end;
   merge Expenses(in = e) Revenue(in = r);
   by FlightID Date;
   if e and r;
   Profit = sum(Rev1st, RevBusiness, RevEcon, -Expenses);
   rc = airports.find(key:origin);
   OriginCity = city;
   OriginAirport = name;
   rc=airports.find(key:dest);
   DestCity = city;
   DestAirport = name;
run;
```

145 c04s3d2

Preview of Program

```
data Alldata_hash;
   if _N_ = 1 then do;
      if 0 then
         set ia.Airports(keep=Code City Name);
      declare hash airports(dataset: "ia.Airports");
      airports.definekey ("Code");
      airports.definedata("City", "Name");
      airports.definedone();
   end;
   merge Expenses(in = e) Revenue(in = r);
   by FlightID Date;
   if e and r;
   Profit = sum(Rev1st, RevBusiness, RevEcon, -Expenses);
   rc = airports.find(key:origin);
   OriginCity = city;
   OriginAirport = name;
   rc=airports.find(key:dest);
   DestCity = city;
   DestAirport = name;
run;
```

146 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
data alldata_hash;

   if _N_ = 1 then do;
      if 0 then
         set ia.airports(keep = Code City Name);     ❶
      declare hash airports(dataset: "ia.Airports");❷
      airports.definekey ("Code"); ❸
      airports.definedata("City", "Name");❹
      airports.definedone();
   end;
```

147

c04s3d2

① To initialize the attributes of hash variables that originate from an existing SAS data set, you can use a non-executing SET statement. When you use this technique, the MISSING routine is not required.

IF 0 **THEN SET** *data-set-name* (KEEP=*hash-variables*)**;**

② Creates a hash object named **`airports`** and loads it from **`ia.airports`**.

③ Defines the key to be the value of the variable **`Code`**.

④ Defines the data to be the value of the variables **`City`** and **`Name`**.

Creating a Hash Object from a SAS Data Set

Partial Program

```
data alldata_hash;

   if _N_ = 1 then do;
      if 0 then
         set ia.airports(keep = Code City Name);
      declare hash airports(dataset: "ia.Airports");
      airports.definekey ("Code");
      airports.definedata("City", "Name");
      airports.definedone();
   end;
```

Because the IF condition is false during execution, the SET statement is compiled but not executed. The PDV would be created with the variables **Code**, **City**, and **Name** from **ia.airports**.

148 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
data alldata_hash;

   if _N_ = 1 then do;
      if 0 then
         set ia.airports(keep = Code City Name);
      declare hash airports(dataset: "ia.Airports");
      airports.definekey ("Code");
      airports.definedata("City", "Name");
      airports.definedone();
   end;
```

Key: Code	Data: City	Data: Name
AKL	Auckland	International
AMS	Amsterdam	Schiphol
<more keys and data added>		

149 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
rc = airports.find(key:origin);
OriginCity = city;
OriginAirport = name;
rc = airports.find(key:dest);
DestCity = city;
DestAirport = name;
```

Partial Hash Object

Key: Code	Data: City	Data: Name
CDG	Paris	Charles de Gaulle
LHR	London, England	Heathrow
<more keys and data added>		

Partial PDV

rc	Origin	Dest	City	Name	Origin City	Origin Airport	Dest City	Dest Airport
	CDG	LHR						

RC is a variable that contains the return code from the FIND method.

150 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
rc = airports.find(key:origin);
OriginCity = city;
OriginAirport = name;
rc = airports.find(key:dest);
DestCity = city;
DestAirport = name;
```

Partial Hash Object

Key: Code	Data: City	Data: Name
CDG	Paris	Charles de Gaulle
LHR	London, England	Heathrow
<more keys and data added>		

Partial PDV

rc	Origin	Dest	City	Name	Origin City	Origin Airport	Dest City	Dest Airport
0	CDG	LHR	Paris	Charles de Gaulle				

151 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
rc = airports.find(key:origin);
OriginCity = city;
OriginAirport = name;
rc = airports.find(key:dest);
DestCity = city;
DestAirport = name;
```

Partial Hash Object

Key: Code	Data: City	Data: Name
CDG	Paris	Charles de Gaulle
LHR	London, England	Heathrow
<more keys and data added>		

Partial PDV

rc	Origin	Dest	City	Name	Origin City	Origin Airport	Dest City	Dest Airport
0	CDG	LHR	Paris	Charles de Gaulle	Paris	Charles de Gaulle		

152 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
rc = airports.find(key:origin);
OriginCity = city;
OriginAirport = name;
rc = airports.find(key:dest);
DestCity = city;
DestAirport = name;
```

Partial Hash Object

Key: Code	Data: City	Data: Name
CDG	Paris	Charles de Gaulle
LHR	London, England	Heathrow
<more keys and data added>		

Partial PDV

rc	Origin	Dest	City	Name	Origin City	Origin Airport	Dest City	Dest Airport
0	CDG	LHR	London, England	Heathrow	Paris	Charles de Gaulle		

153 c04s3d2 ...

Creating a Hash Object from a SAS Data Set

Partial Program

```
rc = airports.find(key:origin);
OriginCity = city;
OriginAirport = name;
rc = airports.find(key:dest);
DestCity = city;
DestAirport = name;
```

Partial Hash Object

Key: Code	Data: City	Data: Name
CDG	Paris	Charles de Gaulle
LHR	London, England	Heathrow
<more keys and data added>		

Partial PDV

rc	Origin	Dest	City	Name	Origin City	Origin Airport	Dest City	Dest Airport
0	CDG	LHR	London, England	Heathrow	Paris	Charles de Gaulle	London, England	Heathrow

154

c04s3d2 ...

Using the FIND Method

The FIND method creates return code that is a numeric value that specifies whether the FIND method succeeded or failed.

- The return code can be used in conditional logic to insure that the FIND method found a KEY value in the hash object that matches the KEY value from the PDV.
- If the program does not contain a return code variable for the method call and the method fails, then an appropriate error message is written to the log.

Values of the return code variable

zero → success

non-zero → failure

155

Using the Return Code for the FIND Method

Replace this code:

```
rc = airports.find(key:origin);
OriginCity = city;
OriginAirport = name;
rc = airports.find(key:dest);
DestCity = city;
DestAirport = name;
```

With this code:

```
rc = airports.find(key:origin);
if rc = 0 then do;
   OriginCity = city;
   OriginAirport = name;
end;
else do;
   OriginCity = ' ';
   OriginAirport = ' ';
end;
rc = airports.find(key:dest);
if rc = 0 then do;
   DestCity = city;
   DestAirport = name;
end;
else do;
   DestCity = ' ';
   DestAirport = ' ';
end;
```

156

Combining the Three Data Sets

c04s3d2

Use a hash object.

```
proc sort data = ia.Expenses out = Expenses;
   by FlightID Date;
run;

proc sort data = ia.Revenue out = Revenue;
   by FlightID Date;
run;

data Alldata_hash;

   if _N_ = 1 then do;
      if 0 then
         set ia.Airports(keep=Code City Name);
      declare hash airports(dataset: "ia.Airports");
      airports.definekey ("Code");
      airports.definedata("City", "Name");
      airports.definedone();
   end;

   merge Expenses(in = e) Revenue(in = r);
   by FlightID Date;
   if e and r;
   Profit = sum(Rev1st, RevBusiness, RevEcon, -Expenses);

   rc = airports.find(key:origin);
   OriginCity = city;
   OriginAirport = name;
   rc=airports.find(key:dest);
   DestCity = city;
   DestAirport = name;
run;

proc print data = Alldata_hash(obs = 5);
   title 'Result of Merge plus Hash Object Lookup';
   var FlightID Date OriginCity OriginAirport DestCity DestAirport Profit;
   format Date date9.;
run;

title;
```

(Continued on the next page.)

```
/*****************************/
/* Alternate Solution        */
/* Checking the Return Code  */
/*****************************/

proc sort data = ia.Expenses out = Expenses;
   by FlightID Date;
run;

proc sort data = ia.Revenue out = Revenue;
   by FlightID Date;
run;

data Alldata_hash;

   if _N_ = 1 then do;
      if 0 then
         set ia.Airports(keep=Code City Name);
      declare hash airports(dataset: "ia.Airports");
      airports.definekey ("Code");
      airports.definedata("City", "Name");
      airports.definedone();
   end;

   merge Expenses(in = e) Revenue(in = r);
   by FlightID Date;
   if e and r;
   Profit = sum(Rev1st, RevBusiness, RevEcon, -Expenses);

   rc = airports.find(key:origin);
   if rc = 0 then do;
      OriginCity = city;
      OriginAirport = name;
   end;
   else do;
      OriginCity = ' ';
      OriginAirport = ' ';
   end;

   rc = airports.find(key:dest);
   if rc = 0 then do;
      DestCity = city;
      DestAirport = name;
   end;
   else do;
      DestCity = ' ';
      DestAirport = ' ';
   end;
run;

proc print data = Alldata_hash(obs = 5);
   title 'Result of Merge plus Hash Object Lookup';
   var FlightID Date OriginCity OriginAirport DestCity DestAirport Profit;
   format Date date9.;
run;
```

To define all data set variables as data variables for the hash object, use the ALL: "YES" option.

hashobject.DEFINEDATA (ALL: "YES")**;**

```
                          Result of Merge plus Hash Object Lookup

            Flight
     Obs      ID            Date        OriginCity

       1    IA00100    02DEC2005    Raleigh-Durham, NC
       2    IA00100    03DEC2005    Raleigh-Durham, NC
       3    IA00100    04DEC2005    Raleigh-Durham, NC
       4    IA00100    05DEC2005    Raleigh-Durham, NC
       5    IA00100    06DEC2005    Raleigh-Durham, NC


     Obs               OriginAirport                  DestCity

       1    Raleigh-Durham International Airport    London, England
       2    Raleigh-Durham International Airport    London, England
       3    Raleigh-Durham International Airport    London, England
       4    Raleigh-Durham International Airport    London, England
       5    Raleigh-Durham International Airport    London, England


     Obs      DestAirport       Profit

       1    Heathrow Airport     71553
       2    Heathrow Airport     14308
       3    Heathrow Airport    108937
       4    Heathrow Airport     90999
       5    Heathrow Airport     21019
```

Advantages of Hash Objects

Advantages of using hash objects include the following:

- use of character and numeric keys
- use of composite keys
- ability for faster lookup
- ability to be loaded from a SAS data set
- fine level of control (flexibility)
- ability to do chained lookups

158

Disadvantages of Hash Objects

Disadvantages of using a hash object include the following:

- unique keys required
- DATA step only

159

Exercises

4. Using a Hash Object

a. Create a report that shows revenues, expenses, and profits for flights to Australia and New Zealand. Expenses for flights to Australia and New Zealand are in **`ia.Dnunder`** (145 observations). Revenues for all flights are in **`ia.Sales`** (about 50,000 observations).

b. Load the relevant data from **`ia.Sales`** in a hash object and use it as a lookup table for the flights in **`ia.Dnunder`**. Include the variables **`FlightID`**, **`RouteID`**, **`FltDate`**, **`RevTotal`**, **`Expenses`**, and **`Profit`** in the report.

Partial Listing

ia.dnunder

Obs	Flight ID	FltDate	Expenses
1	IA10200	01DEC2005	154269
2	IA10200	02DEC2005	65188
3	IA10200	03DEC2005	161419
4	IA10201	08DEC2005	56839
5	IA10200	13DEC2005	80197

Partial Listing

ia.sales

Obs	Flight ID	RouteID	Origin	Dest	DestType	FltDate	Cap1st	CapBus
1	IA10700	0000107	WLG	AKL	International	01JAN2004	12	.
2	IA10701	0000107	WLG	AKL	International	01JAN2004	12	.
3	IA10702	0000107	WLG	AKL	International	01JAN2004	12	.
4	IA10703	0000107	WLG	AKL	International	01JAN2004	12	.
5	IA10704	0000107	WLG	AKL	International	01JAN2004	12	.

Obs	CapEcon	Cap Pass Total	CapCargo	Num1st	Num Bus	Num Econ	Num Pass Total	Rev1st	RevBus
1	138	150	36900	11	.	126	137	$1,397.00	.
2	138	150	36900	12	.	136	148	$1,524.00	.
3	138	150	36900	10	.	112	122	$1,270.00	.
4	138	150	36900	12	.	113	125	$1,524.00	.
5	138	150	36900	10	.	118	128	$1,270.00	.

Obs	RevEcon	CargoRev	RevTotal	Cargo Weight
1	$5,292.00	$1,900.00	$8,589	9500
2	$5,712.00	$1,460.00	$8,696	7300
3	$4,704.00	$2,500.00	$8,474	12500
4	$4,746.00	$2,380.00	$8,650	11900
5	$4,956.00	$2,260.00	$8,486	11300

Partial Output

```
                   Profit for Flights to Australia and New Zealand

                  Flight                             Rev
              Obs     ID      RouteID     FltDate    Total   Expenses   Profit

                1   IA10200   0000102   01DEC2005   359778    154269    205509
                2   IA10200   0000102   02DEC2005   357828     65188    292640
                3   IA10200   0000102   03DEC2005   356887    161419    195468
                4   IA10201   0000102   08DEC2005   357015     56839    300176
                5   IA10200   0000102   13DEC2005   357543     80197    277346
```

4.4 Using Formats as Lookup Tables

Objectives

- Create permanent formats.
- Access permanent formats.
- Create formats from SAS data sets.
- Maintain formats.
- Use formats as lookup tables.

162

Table Lookup Using Formats

The appearance of values is controlled by formats.

- Use the FORMAT procedure to define tables that store coded values and the definitions of the codes.
- Reference these user-defined formats when a table lookup operation is needed.

163

You can use PROC FORMAT to define the following:

- VALUES
- PICTURES
- INFORMATS

You can code missing values using the following:

- ' ' (missing character)
- . (missing numeric)

You can use the following keywords:

- OTHER
- HIGH
- LOW

You can code non-inclusive ranges:

- <

Overview of a Format

A format is similar to stacks of buckets that are referred to by the value of a variable.

- SAS puts data values and label values in the buckets when the format is used in a FORMAT statement, PUT function, or PUT statement.
- SAS uses a binary search on the **data value** bucket in order to return the value in the **label** bucket.

164

...

Using Permanent Formats as Lookup Tables

c04s4d1

Example 1

```
proc format library = ia.formats;

   value $routes            'Route1' = 'Zone One'
                 'Route2' - 'Route4' = 'Zone Two'
                 'Route5' - 'Route7' = 'Zone Three'
                                 ' ' = 'Missing'
                               other = 'Unknown';

   value $dest  'AKL','AMS','ARN',
                'ATH','BKK','BRU',
                'CBR','CCU','CDG',
                'CPH','CPT','DEL',
                'DXB','FBU','FCO',
                'FRA','GLA','GVA',
                'HEL','HKG','HND',
                'JED','JNB','JRS',
                'LHR','LIS','MAD',
                'NBO','PEK','PRG',
                'SIN','SYD','VIE','WLG' = 'International'
                'ANC','BHM','BNA',
                'BOS','DFW','HNL',
                'IAD','IND','JFK',
                'LAX','MCI','MIA',
                'MSY','ORD','PWM',
                'RDU','SEA','SFO' = 'Domestic';

   value revfmt             . = 'Missing'
                 low - 10000 = 'Up to $10,000'
              10000 <- 20000 = '$10,000+ to $20,000'
              20000 <- 30000 = '$20,000+ to $30,000'
              30000 <- 40000 = '$30,000+ to $40,000'
              40000 <- 50000 = '$40,000+ to $50,000'
              50000 <-  high = 'More than $50,000';
run;
```

Example 2

```
proc catalog cat = ia.FORMATS;
   contents;
run;
quit;

proc format library = ia fmtlib;
   title 'Using the FMTLIB option to view the formats';
run;
```

Output

```
                           Contents of Catalog IA.FORMATS

   #   Name     Type             Create Date         Modified Date   Description
   ______________________________________________________________________________

   1   DATES    FORMAT     26OCT2001:14:29:34    26OCT2001:14:29:34
   2   REVFMT   FORMAT     22JAN2004:11:20:14    22JAN2004:11:20:14
   3   DEST     FORMATC    22JAN2004:11:20:14    22JAN2004:11:20:14
   4   ROUTES   FORMATC    22JAN2004:11:20:14    22JAN2004:11:20:14
```

```
                    Using the FMTLIB option to view the formats

      FORMAT NAME: REVFMT   LENGTH:   18   NUMBER OF VALUES:    7
   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  18  FUZZ: STD

START            |END              |LABEL  (VER. V7|V8   13MAY2005:15:36:19)
                .|                .|Missing
LOW              |            10000|Up to $10,000
           10000<|            20000|$10,000+ to $20000
           20000<|            30000|$20,000+ to $30000
           30000<|            40000|$30,000+ to $40000
           40000<|            50000|$40,000+ to $50000
           50000<HIGH              |More than $50,000
```

(Continued on the next page.)

```
FORMAT NAME: $AIRPORT LENGTH:   28   NUMBER OF VALUES:   52
MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  28  FUZZ:        0
```

START	END	LABEL (VER. V7|V8 20APR2005:13:41:43)
AKL	AKL	Auckland
AMS	AMS	Amsterdam
ANC	ANC	Anchorage, AK
ARN	ARN	Stockholm
ATH	ATH	Athens (Athinai)
BHM	BHM	Birmingham, AL
BKK	BKK	Bangkok
BNA	BNA	Nashville, TN
BOS	BOS	Boston, MA
BRU	BRU	Brussels (Bruxelles)
CBR	CBR	Canberra, Australian Capitol
CCU	CCU	Calcutta
CDG	CDG	Paris
CPH	CPH	Kobenhavn (Copenhagen)
CPT	CPT	Cape Town
DEL	DEL	Delhi
DFW	DFW	Dallas/Fort Worth, TX
DXB	DXB	Dubai
FBU	FBU	Oslo
FCO	FCO	Roma (Rome)
FRA	FRA	Frankfurt
GLA	GLA	Glasgow, Scotland
GVA	GVA	Geneva
HEL	HEL	Helsinki
HKG	HKG	Hong Kong
HND	HND	Tokyo
HNL	HNL	Honolulu, HI
IAD	IAD	Washington, DC
IND	IND	Indianapolis, IN
JED	JED	Jeddah
JFK	JFK	New York, NY
JNB	JNB	Johannesburg
JRS	JRS	Jerusalem
LAX	LAX	Los Angeles, CA
LHR	LHR	London, England
LIS	LIS	Lisboa (Lisbon)
MAD	MAD	Madrid
MCI	MCI	Kansas City, MO
MIA	MIA	Miami, FL
MSY	MSY	New Orleans, LA
NBO	NBO	Nairobi
ORD	ORD	Chicago, IL
PEK	PEK	Beijing (Peking)
PRG	PRG	Praha (Prague)
PWM	PWM	Portland, ME
RDU	RDU	Raleigh-Durham, NC
SEA	SEA	Seattle, WA
SFO	SFO	San Francisco, CA
SIN	SIN	Singapore
SYD	SYD	Sydney, New South Wales
VIE	VIE	Wien (Vienna)
WLG	WLG	Wellington

(Continued on the next page.)

FORMAT NAME: $DEST LENGTH: 13 NUMBER OF VALUES: 52 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 13 FUZZ: 0		
START	END	LABEL (VER. V7|V8 13MAY2005:15:36:19)
AKL	AKL	International
AMS	AMS	International
ANC	ANC	Domestic
ARN	ARN	International
ATH	ATH	International
BHM	BHM	Domestic
BKK	BKK	International
BNA	BNA	Domestic
BOS	BOS	Domestic
BRU	BRU	International
CBR	CBR	International
CCU	CCU	International
CDG	CDG	International
CPH	CPH	International
CPT	CPT	International

(Continued on the next page.)

Using the FMTLIB option to view the formats

FORMAT NAME: $DEST LENGTH: 13 NUMBER OF VALUES: 52
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 13 FUZZ: 0

START	END	LABEL (CONT'D)
DEL	DEL	International
DFW	DFW	Domestic
DXB	DXB	International
FBU	FBU	International
FCO	FCO	International
FRA	FRA	International
GLA	GLA	International
GVA	GVA	International
HEL	HEL	International
HKG	HKG	International
HND	HND	International
HNL	HNL	Domestic
IAD	IAD	Domestic
IND	IND	Domestic
JED	JED	International
JFK	JFK	Domestic
JNB	JNB	International
JRS	JRS	International
LAX	LAX	Domestic
LHR	LHR	International
LIS	LIS	International
MAD	MAD	International
MCI	MCI	Domestic
MIA	MIA	Domestic
MSY	MSY	Domestic
NBO	NBO	International
ORD	ORD	Domestic
PEK	PEK	International
PRG	PRG	International
PWM	PWM	Domestic
RDU	RDU	Domestic
SEA	SEA	Domestic
SFO	SFO	Domestic
SIN	SIN	International
SYD	SYD	International
VIE	VIE	International
WLG	WLG	International

(Continued on the next page.)

```
                    Using the FMTLIB option to view the formats

     FORMAT NAME: $ROUTES  LENGTH:   10   NUMBER OF VALUES:    5
  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  10  FUZZ:       0
```

START	END	LABEL (VER. V7|V8 13MAY2005:15:36:19)
		Missing
Route1	Route1	Zone One
Route2	Route4	Zone Two
Route5	Route7	Zone Three
OTHER	**OTHER**	Unknown

General Form of a PROC FORMAT Step

```
PROC FORMAT LIBRARY = libref.catalog;
      VALUE $charfmt 'value1' = 'formatted-value-1'
                     'value2' = 'formatted-value-2'
                     'valuen' = 'formatted-value-n';
      VALUE numfmt   value1  = 'formatted-value-1'
                     value2 = 'formatted-value-2'
                     valuen = 'formatted-value-n';
RUN;
```

To avoid re-creating formats each time that a job is run, store formats permanently.

166

A VALUE statement is required for each format.

Format names must meet the following conditions:

- cannot duplicate SAS format names, such as DOLLAR and SSN
- cannot end in a number
- must be 32 characters or fewer

For character formats, these are the requirements:

- Format names must begin with a $.
- Input values are quoted.

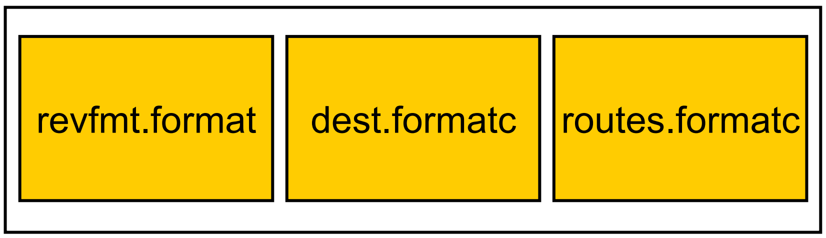
For numeric formats, input values are not quoted.

Format names are limited to eight characters in versions of SAS prior to SAS®9.

How Are Formats Stored?

- Formats are stored as SAS catalog entries.
- SAS catalogs are special SAS files that store many different kinds of information in smaller units called *entries*.
- A single SAS catalog can contain several different catalog entries.

SAS Catalog **`ia.formats`**



167

Catalog entries have four-level names: *libref.catalog.entry-name.type.*

The *type* for character formats is **formatc**. The *type* for numeric formats is **format**.

How Are Formats Stored?

Without the LIBRARY= option, formats are stored in the **`work.formats`** catalog and exist for the duration of the SAS session.

PROC FORMAT;

If the LIBRARY= option specifies only a *libref*, formats are stored permanently in ***libref*.formats**.

PROC FORMAT LIBRARY = *libref* ;

If the LIBRARY= option specifies *libref.catalog*, formats are stored permanently in that catalog.

PROC FORMAT LIBRARY = *libref.catalog*;

168

The CATALOG Procedure

The CATALOG procedure manages entries in SAS catalogs.

Selected functions of PROC CATALOG:

- creating a listing of the contents of a catalog
- copying a catalog or selected entries within a catalog
- renaming or deleting entries within a catalog
- modifying the description of a catalog entry

You can also use the Explorer window in SAS to see the contents of a catalog.

169

General Form of PROC CATALOG

```
PROC CATALOG CATALOG = <libref.>catalog <options>;
     CONTENTS <OUT = SAS-data-set>
                    <FILE = fileref>;
     COPY OUT = <libref.>catalog <options>;
            SELECT entry(s) </ ENTRYTYPE=etype>;
            EXCLUDE entry(s) </ ENTRYTYPE=etype>;
     DELETE entry(s) </ ENTRYTYPE=etype>;
RUN;
QUIT;
```

170

For a complete listing of the CATALOG procedure statements and functionality, see the procedures section of the Base SAS Procedures Guide in the Base SAS documentation.

Documenting Formats

You can use the FMTLIB option in the PROC FORMAT statement to document the format.

General form of the FMTLIB option:

```
PROC FORMAT LIBRARY = libref.catalog
                     FMTLIB;
      <other statements>;
RUN;
```

171

Other statements can include the following:

```
SELECT format-name format-name...;
```

```
EXCLUDE format-name format-name...;
```

You can use either the SELECT or EXCLUDE statement to process specific formats rather than an entire catalog.

Using Permanent Formats

You can reference formats in any of the following:

- FORMAT statements
- PUT statements
- PUT functions in assignment, WHERE, or IF statements
- FORMAT= options

172

When a user-defined format is referenced, SAS does the following:

- loads the format from the catalog entry into memory
- performs a binary search on values in the table to execute a lookup
- returns a single result for each lookup

Using Permanent Formats as Lookup Tables

c04s4d2

```
options fmtsearch = (ia);

proc print data = ia.cargorev(obs = 10);
   where put(Route,$routes.) = 'Zone Two';
   format RevCargo revfmt. Date mmddyyb10.;
   var Date Route RevCargo;
   title 'Revenue Cargo for Zone Two';
   title2 'First Ten Rows';
run;
```

Output

```
                        Revenue Cargo for Zone Two
                              First Ten Rows

              Obs          Date    Route          RevCargo

                1    01 01 2000    Route2    Up to $10,000
                2    01 01 2000    Route3    More than $50,000
                6    01 02 2000    Route3    More than $50,000
                7    01 03 2000    Route3    Up to $10,000
                9    01 03 2000    Route3    Up to $10,000
               11    01 03 2000    Route4    $40,000 to $50000
               12    01 04 2000    Route3    Up to $10,000
               14    01 05 2000    Route3    Up to $10,000
               15    01 05 2000    Route4    Up to $10,000
               20    01 05 2000    Route4    More than $50,000
```

You can use the WHERE statement when the OBS= option is in effect.

✎ The MMDDYYB10. format displays the **Date** variable value using a blank as a separator.

General form:

MMDDYY*xw*.

Value of *x*	Separator
B	blank
C	colon
D	dash
N	no separator
P	period
S	slash

Using the FMTSEARCH= System Option

To use permanent formats or to search multiple catalogs, use the FMTSEARCH= system option to identify the catalog(s) to be searched for the format(s).

General form of the FMTSEARCH= system option:

> **OPTIONS** FMTSEARCH = (*item-1 item-2…item-n*);

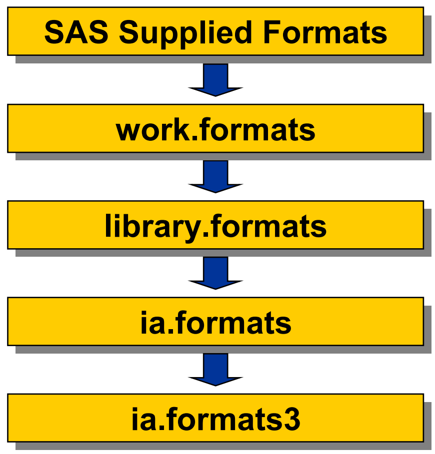
174

By specifying multiple items in the FMTSEARCH= option, you can concatenate format catalogs. This enables you to do the following:

- define personal format catalogs to be used in addition to corporate catalogs
- use test and production format catalogs without duplicating the production catalog
- control the order in which catalogs are searched

Using the FMTSEARCH= System Option

```
options fmtsearch = (ia ia.formats3);
```



175 ...

Because **ia** is a libref without a catalog name, **formats** is assumed to be the catalog name.

SAS-supplied formats are always searched first. The **work.formats** catalog is always searched second, unless it appears in the FMTSEARCH list. If the **library** libref is assigned, the **library.formats** catalog is searched after **work.formats** and before anything else in the FMTSEARCH list, unless it appears in the list. To assign the **library** libref, use the code shown below:

```
libname library 'SAS-data-library-containing-format-catalog';
```

Using the NOFMTERR System Option

By default, the FMTERR system option is in effect. If you use a format that SAS cannot load, SAS issues an error message and stops processing the step.

To prevent the default action, change the system option FMTERR to NOFMTERR.

```
OPTIONS FMTERR | NOFMTERR;
```

176

FMTERR specifies that when SAS cannot find a specified variable format, it generates an error message and does not allow default substitution to occur.

NOFMTERR replaces missing formats with the *w.* or *$w.* default format, issues a note, and continues processing.

Using a Control Data Set to Create a Format

The data set **`ia.acities`** contains airport codes and airport cities. Rather than typing the values in the PROC FORMAT code, you can create a format from the data set and use the format as a lookup table.

```
Airport Airport          Airport                  Airport
Code    City             Name                     Country

AKL     Auckland         International            New Zealand
AMS     Amsterdam        Schiphol                 Netherlands
ANC     Anchorage, AK    Anchorage International  USA
ARN     Stockholm        Arlanda                  Sweden
ATH     Athens           Hellinikon International Greece
BHM     Birmingham, AL   Birmingham International USA
BKK     Bangkok          Don Muang International  Thailand
```

177

The control data set has the following attributes:

- must contain the variables **`FmtName`**, **`Start`**, and **`Label`**
- must contain the variable **`Type`** for character formats, unless the value for **`FmtName`** begins with a $
- does not require a **`Type`** variable for numeric formats
- assumes that the ending value of the format range is equal to the value of **`Start`** if no variable named **`End`** is found
- does not require the other variables created by the CNTLOUT= option that specify optional attributes
- can be created by a DATA step, another PROC step, or an interactive application such as the Viewtable window
- can be used to create new formats, as well as re-create existing formats
- must be grouped by **`FmtName`** if multiple formats are specified

Using a Control Data Set to Create a Format

c04s4d3

Create the CNTLIN data set.

```
data aports;
   keep Start Label FmtName;
   retain FmtName '$airport';
   set ia.acities (rename = (Code = Start
                             City = Label));
run;

proc print data = work.aports(obs = 10) noobs;
   title 'Airports';
run;
```

Output

```
                                    Airports

             fmtname     Label                   Start

             $airport    Auckland                 AKL
             $airport    Amsterdam                AMS
             $airport    Anchorage, AK            ANC
             $airport    Stockholm                ARN
             $airport    Athens (Athinai)         ATH
             $airport    Birmingham, AL           BHM
             $airport    Bangkok                  BKK
             $airport    Nashville, TN            BNA
             $airport    Boston, MA               BOS
             $airport    Brussels (Bruxelles)     BRU
```

Create the format and document its contents:

```
proc format library = ia cntlin = aports;
run;

proc format library = ia fmtlib;
   select $airport;
   title '$airport format';
run;
```

Partial Output

```
                              $airport format

        ----------------------------------------------------------------------------
        |    FORMAT NAME: $AIRPORT LENGTH:   28   NUMBER OF VALUES:   52            |
        |  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  28  FUZZ:        0     |
        |--------------------------------------------------------------------------|
        |START           |END             |LABEL  (VER. V7|V8   20APR2005:13:41:43)|
        |----------------+----------------+----------------------------------------|
        |AKL             |AKL             |Auckland                                |
        |AMS             |AMS             |Amsterdam                               |
        |ANC             |ANC             |Anchorage, AK                           |
        |ARN             |ARN             |Stockholm                               |
        |ATH             |ATH             |Athens (Athinai)                        |
        |BHM             |BHM             |Birmingham, AL                          |
        |BKK             |BKK             |Bangkok                                 |
        |BNA             |BNA             |Nashville, TN                           |
        |BOS             |BOS             |Boston, MA                              |
        |BRU             |BRU             |Brussels (Bruxelles)                    |
        |CBR             |CBR             |Canberra, Australian Capitol            |
        |CCU             |CCU             |Calcutta                                |
        |CDG             |CDG             |Paris                                   |
```

Use the format:

```
options fmtsearch = (ia);

data international;
   set ia.international;
   DestCity = put(dest,$airport.);
   OriginCity = put(Origin,$airport.);
run;

proc print data=international (obs = 10);
   title 'International Cities';
run;
```

```
                               International Cities

                                                          Num
       Flight                                   Num   Num    Pass
 Obs     ID     Origin  Dest     FltDate  Num1st  Bus  Econ   Total  DestCity   OriginCity

   1  IA10700    WLG    AKL   01JAN2005    11      .    126    137   Auckland   Wellington
   2  IA10701    WLG    AKL   01JAN2005    12      .    136    148   Auckland   Wellington
   3  IA10702    WLG    AKL   01JAN2005    10      .    112    122   Auckland   Wellington
   4  IA10703    WLG    AKL   01JAN2005    12      .    113    125   Auckland   Wellington
   5  IA10704    WLG    AKL   01JAN2005    10      .    118    128   Auckland   Wellington
   6  IA10705    WLG    AKL   01JAN2005    11      .    117    128   Auckland   Wellington
   7  IA06900    LHR    AMS   01JAN2005    13      .    102    115   Amsterdam  London, England
   8  IA06901    LHR    AMS   01JAN2005    13      .    105    118   Amsterdam  London, England
   9  IA06902    LHR    AMS   01JAN2005    12      .     95    107   Amsterdam  London, England
  10  IA06903    LHR    AMS   01JAN2005    14      .    119    133   Amsterdam  London, England
```

Using a Control Data Set to Create a Format

You can create a format from a SAS data set that contains value information (called a *control data set*).

Use the CNTLIN= option to read the data and create the format.

General form of CNTLIN= option:

> **PROC FORMAT** LIBRARY = *libref.catalog*
> CNTLIN = *SAS-data-set*;
> **RUN**;

179

Review

The CNTLIN= data set has the following features:

- must contain the variables **FmtName**, **Start**, and **Label**
- must contain the variable **Type** for character formats, unless the value for **FmtName** begins with a $
- does not require a **Type** variable for numeric formats
- assumes that the ending value of the format range is equal to the value of **Start** if no variable named **End** is found
- does not require the other variables created by the CNTLOUT= option that specify optional attributes
- can be created by a DATA step, another PROC step, or an interactive application such as the Viewtable window
- can be used to create new formats, as well as re-create existing formats
- must be grouped by **FmtName** if multiple formats are specified

Maintaining Formats

To maintain formats, perform one of the following tasks:

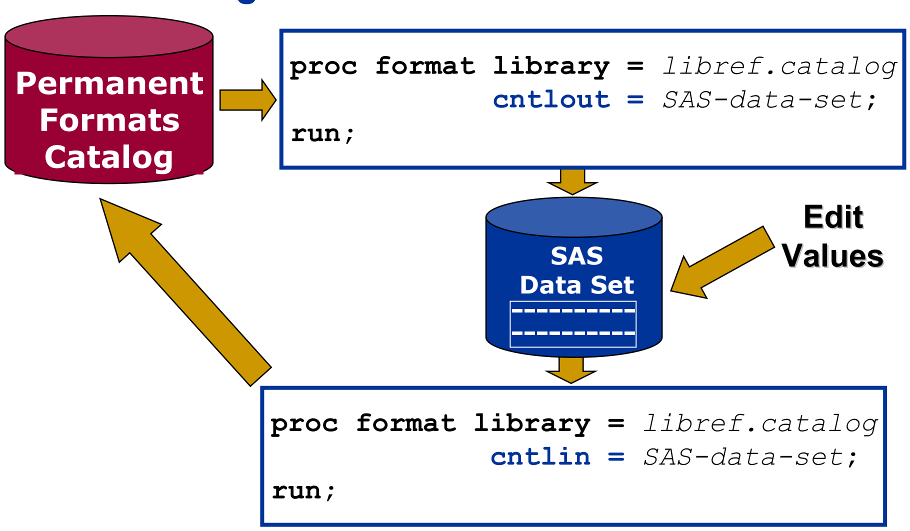
- Edit the PROC FORMAT code that created the original format.

or

- Create a SAS data set from the format, edit the data set, and use the CNTLIN= option to re-create the format.

180

Maintaining Permanent Formats



181 ...

✎ When the data set created by the CNTLOUT= option will be used as a CNTLIN= data set in a subsequent FORMAT procedure step, the minimum variables that must be edited are **START**, **END**, **FMTNAME**, and **LABEL**.

Maintaining Permanent Formats

c04s4d4

```
proc format lib = ia cntlout = fmtdata;
   select $airport;
run;
```

Log

```
295  proc format lib = ia cntlout = fmtdata;
296     select $airport;
297  run;

NOTE: PROCEDURE FORMAT used:
      real time           0.41 seconds
      cpu time            0.04 seconds

NOTE: The data set WORK.FMTDATA has 52 observations and 21 variables.
```

Add the new observations, re-create the format, and document the format:

```
proc fsedit data = work.fmtdata;
run;

proc format library = ia cntlin = fmtdata;
run;

proc format library = ia fmtlib;
   select $airport;
   title 'New values in the $AIRPORT Format';
run;
```

Rather than using an interactive technique to add data, you can use procedures such as PROC SQL.

```
proc format lib = ia cntlout = fmtdata;
   select $airport;
run;

proc sql;
   insert into FmtData
      set FmtName = '$airport',
            Start = 'YQB',
              End = 'YQB',
            Label = 'Quebec, QC'
      set FmtName = '$AIRPORT',
            Start = 'YUL',
              End = 'YUL',
            Label = 'Montreal, QC';
quit;
```

Log

```
proc sql;
   insert into fmtdata
    set FmtName = '$airport',
            Start = 'YQB',
              End = 'YQB',
            Label = 'Quebec, QC'
      set FmtName = '$airport',
            Start = 'YUL',
              End = 'YUL' ,
            Label = 'Montreal, QC';
NOTE: 2 rows were inserted into WORK.FMTDATA.
```

```
proc format library = ia cntlin = fmtdata;
run;

proc format library = ia fmtlib;
   select $airport;
   title 'New values in the $AIRPORT Format';
run;
```

You can also use a DATA step.

```
proc format lib = ia cntlout = fmtdata;
   select $airport;
run;

data work.fmtdata;
   set work.fmtdata end=last;
   output;
   if last then do;
         FmtName = '$airport';
         Start = 'YYC';
         End = 'YYC';
         Label = 'Calgary, AB';
      output;
         Start = 'YYZ';
         End = 'YYZ';
         Label = 'Toronto, ON';
      output;
   end;
run;

proc format library = ia cntlin = fmtdata;
run;

proc format library = ia fmtlib;
   select $airport;
   title 'New values in the $AIRPORT Format';
run;
```

Partial Output

```
                    New values in the $airport Format

     FORMAT NAME: $AIRPORT LENGTH:   28   NUMBER OF VALUES:   56
  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  28  FUZZ:        0
```

START	END	LABEL (CONT'D)
SFO	SFO	San Francisco, CA
SIN	SIN	Singapore
SYD	SYD	Sydney, New South Wales
VIE	VIE	Wien (Vienna)
WLG	WLG	Wellington
YQB	YQB	Quebec, QC
YUL	YUL	Montreal, QC
YYC	YYC	Calgary, AB
YYZ	YYZ	Toronto, ON

Maintaining Permanent Formats

General form of PROC FORMAT with the CNTLOUT= option:

```
PROC FORMAT LIBRARY = libref.catalog
                      CNTLOUT = SAS-data-set;
      <other statements>;
RUN;
```

183

Other statements can include the following:

- SELECT *format-name format-name*...;
- EXCLUDE *format-name format-name*...;

You can use either the SELECT or EXCLUDE statement to process specific formats rather than an entire catalog.

The variables in the output control data set completely describe all aspects of each format or informat, including optional settings.

The output control data set contains one observation per range per format or informat in the specified catalog.

Advantages of Formats

Advantages of using formats include the following:

- familiarity
- no need to create additional data
- can be used with procedures
- range search for both character and numeric
- binary search through lookup table
- centralize maintenance
- use of multiple PUT functions to create multiple variables

184

Disadvantages of Formats

Disadvantages of using formats include the following:

- memory requirements to load the entire format for the binary search
- use of only one variable for the table lookup
- requires more disk space to store a format than to store SAS data

185

Exercises

5. Creating a Format from a SAS Data Set

Use the **`ia.jcodedat`** data set to create a permanent format named **`$jcodes`**. View the new format using the FMTLIB option in PROC FORMAT.

Output

```
                              $jcodes Format

   ----------------------------------------------------------------------------
   |       FORMAT NAME: $JCODES  LENGTH:   32   NUMBER OF VALUES:   42        |
   |   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  32  FUZZ:        0   |
   |--------------------------------------------------------------------------|
   |START           |END             |LABEL  (VER. V7|V8   22JAN2004:11:31:01)|
   |----------------+----------------+----------------------------------------|
   |BAGCLK          |BAGCLK          |BAGGAGE CLERK                           |
   |BAGSUP          |BAGSUP          |BAGGAGE SUPERVISOR                      |
   |CHKCLK          |CHKCLK          |CHECK IN CLERK                          |
   |CHKSUP          |CHKSUP          |CHECK IN SUPERVISOR                     |
   |FACCLK          |FACCLK          |FACILITIES CLERK                        |
   |FACMGR          |FACMGR          |FACILITES MANAGER                       |
   |FACMNT          |FACMNT          |FACILITIES MAINTENANCE OPERATIVE        |
   |FINACT          |FINACT          |FINANCIAL ACCOUNTANT                    |
   |FINCLK          |FINCLK          |FINANCE CLERK                           |
   |FINMGR          |FINMGR          |FINANCE MANAGER                         |
   |FLSCHD          |FLSCHD          |FLIGHT SCHEDULER                        |
   |FLSMGR          |FLSMGR          |FLIGHT SCHEDULING MANAGER               |
   |FLTAT1          |FLTAT1          |FLIGHT ATTENDANT GRADE 1                |
   |FLTAT2          |FLTAT2          |FLIGHT ATTENDANT GRADE 2                |
   |FLTAT3          |FLTAT3          |FLIGHT ATTENDANT GRADE 3                |
   |FSVCLK          |FSVCLK          |FLIGHT SERVICES CLERK                   |
   |FSVMGR          |FSVMGR          |FLIGHT SERVICES MANAGER                 |
   |GRCREW          |GRCREW          |GROUND CREW                             |
   |GRCSUP          |GRCSUP          |GROUND CREW SUPERVISOR                  |
   |HRCLK           |HRCLK           |HUMAN RESOURCES CLERK                   |
   |HRMGR           |HRMGR           |HUMAN RESOURCES MANAGER                 |
   |ITCLK           |ITCLK           |IT CLERK                                |
   |ITMGR           |ITMGR           |IT MANAGER                              |
   |ITPROG          |ITPROG          |COMPUTER PROGRAMMER                     |
   |ITSUPT          |ITSUPT          |IT SUPPORT SPECIALIST                   |
   |MECH01          |MECH01          |MECHANIC GRADE 1                        |
   |MECH02          |MECH02          |MECHANIC GRADE 2                        |
   |MECH03          |MECH03          |MECHANIC GRADE 3                        |
   |MKTCLK          |MKTCLK          |MARKETING CLERK                         |
   |MKTMGR          |MKTMGR          |MARKETING MANAGER                       |
   |OFFMGR          |OFFMGR          |OFFICE MANAGER                          |
   |PILOT1          |PILOT1          |PILOT GRADE 1                           |
   |PILOT2          |PILOT2          |PILOT GRADE 2                           |
   |PILOT3          |PILOT3          |PILOT GRADE 3                           |
   |PRES            |PRES            |COMPANY PRESIDENT                       |
   |RECEPT          |RECEPT          |RECEPTIONIST                            |
   |RESCLK          |RESCLK          |RESERVATIONS CLERK                      |
   |RESMGR          |RESMGR          |RESERVATIONS MANAGER                    |
   |SALCLK          |SALCLK          |SALES CLERK                             |
   |SALMGR          |SALMGR          |SALES MANAGER                           |
   |TELOP           |TELOP           |TELEPHONE SWITCHBOARD OPERATOR          |
   |VICEPR          |VICEPR          |VICE PRESIDENT                          |
   ----------------------------------------------------------------------------
```

6. Updating a Format (Optional)

Update an existing format by following these steps:

a. Add to the permanent **`$jcodes`** format.

b. Use the CNTLOUT= and CNTLIN= options in PROC FORMAT. Add new data for ticket agents using the INSERT statement in PROC SQL or a DATA step program.

TKTAG1	Ticket Agent Grade 1
TKTAG2	Ticket Agent Grade 2
TKTAG3	Ticket Agent Grade 3

c. View the new format using the FMTLIB option in PROC FORMAT. The output is on the next page.

Exercise Output

New values in the $JCODES Format

FORMAT NAME: $JCODES LENGTH: 32 NUMBER OF VALUES: 45 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 32 FUZZ: 0		
START	END	LABEL (VER. V7|V8 22JAN2004:11:50:24)
BAGCLK	BAGCLK	BAGGAGE CLERK
BAGSUP	BAGSUP	BAGGAGE SUPERVISOR
CHKCLK	CHKCLK	CHECK IN CLERK
CHKSUP	CHKSUP	CHECK IN SUPERVISOR
FACCLK	FACCLK	FACILITIES CLERK
FACMGR	FACMGR	FACILITES MANAGER
FACMNT	FACMNT	FACILITIES MAINTENANCE OPERATIVE
FINACT	FINACT	FINANCIAL ACCOUNTANT
FINCLK	FINCLK	FINANCE CLERK
FINMGR	FINMGR	FINANCE MANAGER
FLSCHD	FLSCHD	FLIGHT SCHEDULER
FLSMGR	FLSMGR	FLIGHT SCHEDULING MANAGER
FLTAT1	FLTAT1	FLIGHT ATTENDANT GRADE 1
FLTAT2	FLTAT2	FLIGHT ATTENDANT GRADE 2
FLTAT3	FLTAT3	FLIGHT ATTENDANT GRADE 3
FSVCLK	FSVCLK	FLIGHT SERVICES CLERK
FSVMGR	FSVMGR	FLIGHT SERVICES MANAGER
GRCREW	GRCREW	GROUND CREW
GRCSUP	GRCSUP	GROUND CREW SUPERVISOR
HRCLK	HRCLK	HUMAN RESOURCES CLERK
HRMGR	HRMGR	HUMAN RESOURCES MANAGER
ITCLK	ITCLK	IT CLERK
ITMGR	ITMGR	IT MANAGER
ITPROG	ITPROG	COMPUTER PROGRAMMER
ITSUPT	ITSUPT	IT SUPPORT SPECIALIST
MECH01	MECH01	MECHANIC GRADE 1
MECH02	MECH02	MECHANIC GRADE 2
MECH03	MECH03	MECHANIC GRADE 3
MKTCLK	MKTCLK	MARKETING CLERK
MKTMGR	MKTMGR	MARKETING MANAGER
OFFMGR	OFFMGR	OFFICE MANAGER
PILOT1	PILOT1	PILOT GRADE 1
PILOT2	PILOT2	PILOT GRADE 2
PILOT3	PILOT3	PILOT GRADE 3
PRES	PRES	COMPANY PRESIDENT
RECEPT	RECEPT	RECEPTIONIST
RESCLK	RESCLK	RESERVATIONS CLERK
RESMGR	RESMGR	RESERVATIONS MANAGER
SALCLK	SALCLK	SALES CLERK
SALMGR	SALMGR	SALES MANAGER
TELOP	TELOP	TELEPHONE SWITCHBOARD OPERATOR
TKTAG1	TKTAG1	Ticket Agent Grade 1
TKTAG2	TKTAG2	Ticket Agent Grade 2
TKTAG3	TKTAG3	Ticket Agent Grade 3
VICEPR	VICEPR	VICE PRESIDENT

4.5 Transposing Data to Create a Lookup Table

Objectives

- Use the TRANSPOSE procedure to transpose a SAS data set and prepare it for a table lookup.

188

Another reason for transposing a data set is to restructure a data set to match the requirements of a particular procedure.

Using the TRANSPOSE Procedure

Compare delay values for flights to Raleigh with the average delay statistics for all flights.

ia.rdudelay (First Observation)

Obs	Flight ID	FltDate	Delay
1	IA00201	01JAN2004	22

ia.delaystats (First Ten Variables)

Obs	Statistic	JAN01	JAN02	JAN03	JAN04	JAN05	JAN06	JAN07	JAN08	JAN09
1	AvgDelay	4.708	4.760	5.842	6.571	4.645	6.0714	5.500	5.080	4.692
2	SumDelay	113.000	119.000	111.000	184.000	144.000	85.0000	121.000	127.000	122.000
3	StdDelay	2.971	3.140	3.420	4.316	3.508	4.5987	4.373	4.252	4.688
4	MedianDelay	5.000	4.000	6.000	6.500	4.000	4.5000	4.000	3.000	2.500

189

Using the TRANSPOSE Procedure

An alternate solution to using an array as a lookup table is to transpose **`ia.delaystats`** and merge it with **`ia.rdudelay`**.

`ia.delaystats`
(Partial Output)

Obs	Statistic	JAN01	JAN02	JAN03
1	AvgDelay	4.708	4.760	5.842
2	SumDelay	113.000	119.000	111.000
3	StdDelay	2.971	3.140	3.420
4	MedianDelay	5.000	4.000	6.000

Transpose

`stats`
(Partial Output)

Obs	Statistic	Day	AvgDelay
1	AvgDelay	JAN01	4.70833
2	AvgDelay	JAN02	4.76000
3	AvgDelay	JAN03	5.84211
4	AvgDelay	JAN04	6.57143
5	AvgDelay	JAN05	4.64516
6	AvgDelay	JAN06	6.07143
7	AvgDelay	JAN07	5.50000
8	AvgDelay	JAN08	5.08000
9	AvgDelay	JAN09	4.69231
10	AvgDelay	JAN10	5.11538
11	AvgDelay	JAN11	4.69231

190

Combining Final Results (Self-Study)

Match-Merge

`stats`
(Partial Output)

Obs	Statistic	Day	AvgDelay
1	AvgDelay	JAN01	4.70833
2	AvgDelay	JAN02	4.76000
3	AvgDelay	JAN03	5.84211
4	AvgDelay	JAN04	6.57143
5	AvgDelay	JAN05	4.64516
6	AvgDelay	JAN06	6.07143
7	AvgDelay	JAN07	5.50000
8	AvgDelay	JAN08	5.08000
9	AvgDelay	JAN09	4.69231
10	AvgDelay	JAN10	5.11538
11	AvgDelay	JAN11	4.69231

`ia.rdudelay`
(Partial Output)

Obs	Flight ID	FltDate	Delay
1	IA00201	01JAN2004	22
2	IA00200	01JAN2004	29
3	IA00400	01JAN2004	18
4	IA00401	01JAN2004	24
5	IA00600	01JAN2004	12
6	IA00601	01JAN2004	10
7	IA00602	01JAN2004	15
8	IA00603	01JAN2004	3
9	IA00604	01JAN2004	6

191

Default PROC TRANSPOSE with OUT= Option

```
proc transpose data = ia.delaystats
               out = stats;
run;
```

Partial Output

Default PROC TRANSPOSE

Obs	_NAME_	COL1	COL2	COL3	COL4
1	JAN01	4.70833	113	2.97057	5.0
2	JAN02	4.76000	119	3.13953	4.0
3	JAN03	5.84211	111	3.41993	6.0
4	JAN04	6.57143	184	4.31559	6.5
5	JAN05	4.64516	144	3.50760	4.0
6	JAN06	6.07143	85	4.59873	4.5
7	JAN07	5.50000	121	4.37253	4.0
8	JAN08	5.08000	127	4.25167	3.0
9	JAN09	4.69231	122	4.68845	2.5

192 c04s5d1

The OUT= option provides the name of the new data set.

The default variable names for transposed variables are **_NAME_**, **COL1**, **COL2**, **COL3**, and **COL4**.

The data set is not structured correctly for the merge. More options and statements are needed.

✎ The variable, **Statistic**, does not appear in the PROC TRANSPOSE data set because PROC TRANSPOSE does not automatically transpose character variables.

NAME= Option

```
proc transpose data = ia.delaystats
               out = stats
               name = Day;
run;
```

Partial Output

Using the NAME =

Obs	Day	COL1	COL2	COL3	COL4
1	JAN01	4.70833	113	2.97057	5.0
2	JAN02	4.76000	119	3.13953	4.0
3	JAN03	5.84211	111	3.41993	6.0
4	JAN04	6.57143	184	4.31559	6.5
5	JAN05	4.64516	144	3.50760	4.0
6	JAN06	6.07143	85	4.59873	4.5
7	JAN07	5.50000	121	4.37253	4.0
8	JAN08	5.08000	127	4.25167	3.0
9	JAN09	4.69231	122	4.68845	2.5

193 c04s5d1

The NAME= option specifies the name for the new variable in the output data set that contains the names of the existing variables being transposed.

BY Statement

```
proc sort data = ia.delaystats
          out = delaystats;
   by Statistic;
run;
proc transpose data = delaystats
               out  = stats
               name = Day;
   by Statistic;
run;
```

Partial Output

```
              Using a BY statement

Obs      Statistic        Day        COL1

  1     AvgDelay         JAN01     4.70833
  2     AvgDelay         JAN02     4.76000
  3     AvgDelay         JAN03     5.84211
  <lines removed>
 32     MedianDelay      JAN01     5.00000
 33     MedianDelay      JAN02     4.00000
 34     MedianDelay      JAN03     6.00000
 <lines removed>
 63     StdDelay         JAN01       2.971
 64     StdDelay         JAN02       3.140
 65     StdDelay         JAN03       3.420
```

194 c04s5d1

For each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes. The BY variable is not transposed.

The original SAS data set must be sorted or indexed with the BY statement prior to the PROC TRANSPOSE statement.

The **COL1** variable needs a more descriptive variable name. You can use SAS data set options to rename this variable.

RENAME= Data Set Option

```
proc transpose data = delaystats
               out = stats (rename = (COL1 = AvgDelay))
               name = Day;
   by Statistic;
run;
```

Partial Output

Using the RENAME= option

Obs	Statistic	Day	AvgDelay
1	AvgDelay	JAN01	4.70833
2	AvgDelay	JAN02	4.76000
3	AvgDelay	JAN03	5.84211
4	AvgDelay	JAN04	6.57143
5	AvgDelay	JAN05	4.64516
6	AvgDelay	JAN06	6.07143
7	AvgDelay	JAN07	5.50000
8	AvgDelay	JAN08	5.08000
9	AvgDelay	JAN09	4.69231

195

c04s5d1

Alternate Solution Using the ID Statement

```
proc transpose data = delaystats
               out = stats name = Day;
   id Statistic;
run;
```

Partial Output

Using the ID Statement

Obs	Day	Avg Delay	Sum Delay	Std Delay	Median Delay
1	JAN01	4.70833	113	2.97057	5.0
2	JAN02	4.76000	119	3.13953	4.0
3	JAN03	5.84211	111	3.41993	6.0
4	JAN04	6.57143	184	4.31559	6.5
5	JAN05	4.64516	144	3.50760	4.0
6	JAN06	6.07143	85	4.59873	4.5
7	JAN07	5.50000	121	4.37253	4.0
8	JAN08	5.08000	127	4.25167	3.0
9	JAN09	4.69231	122	4.68845	2.5

196

c04s5d1

The ID statement specifies a variable in the input data set whose formatted values name the transposed variables in the output data set.

The TRANSPOSE Procedure Summary

General form of the TRANSPOSE procedure:

PROC TRANSPOSE <DATA=*input-data-set*>
<OUT=*output-data-set*>
<NAME = *variable-name*>;
<**BY** <DESCENDING> *variable-1*
<...<DESCENDING> *variable-n*>;>
<**VAR** *variable(s)*;>
<**ID** *variable*;>
RUN;

197

- The BY statement is used to transpose each BY group.
- The VAR statement lists the variables to transpose. By default, all numeric variables are transposed. Any character variables that you want to transpose **must** be listed in the VAR statement.
- The ID statement specifies a variable in the input data set whose formatted values name the transposed variables in the output data set.

Advantages of Transposing and Merging

Advantages of transposing and merging include the following:

- transposing data can be used for multiple applications
- no limit to the size of the data sets

198

Disadvantages of Transposing and Merging

Disadvantages of using transposing and merging include the following:

- requires two steps
- requires sorted or indexed data
- requires exact matches
- presence of BY values in all data sets

199

Merging the Transposed Data Set (Self-Study)

c04s5d2

```
proc sort data = ia.rdudelay out = rdudelay;
   by FltDate;
run;

/*****************************
Program assumes that the data set STATS was created
by the TRANSPOSE procedure using the BY statement
and the RENAME= data set option.
*****************************/

data delays;
   set stats;
   FltDate = mdy(1,input(substr(day,4),2.),2004);
   drop day;
   where Statistic = 'AvgDelay';
run;

data combine;
   merge rdudelay delays;
   by FltDate;
   DelayDif = delay - AvgDelay;
run;

proc print data = combine;
   title 'Transposed Average Delays Combined with the Raleigh Delays';
   var FlightID FltDate Delay DelayDif;
run;
```

Partial Output

Transposed Average Delays Combined with the Raleigh Delays

Obs	Flight ID	FltDate	Delay	Delay Dif
1	IA00201	01JAN2004	11	6.2917
2	IA00200	01JAN2004	22	17.2917
3	IA00400	01JAN2004	25	20.2917
4	IA00401	01JAN2004	8	3.2917
5	IA00600	01JAN2004	6	1.2917
6	IA00601	01JAN2004	22	17.2917
7	IA00602	01JAN2004	2	-2.7083
8	IA00603	01JAN2004	22	17.2917
9	IA00604	01JAN2004	21	16.2917
10	IA00605	01JAN2004	23	18.2917

```
/*********************************************************
Alternate Solution if the data set STATS was created with the
TRANSPOSE procedure and the ID statement;
*********************************************************/

proc sort data = ia.rdudelay out = rdudelay;
   by FltDate;
run;

data delays;
   set stats (keep = Day AvgDelay);
   FltDate = mdy(1,input(substr(day,4),2.),2004);
   drop day;
run;

data combine;
   merge rdudelay delays;
   by FltDate;
   DelayDif = delay - AvgDelay;
run;

proc print data = combine;
   title 'Transposed Average Delays Combined with the Raleigh Delays';
   var FlightID FltDate Delay DelayDif;
run;
```

Exercises

7. Using the TRANSPOSE Procedure

Using PROC TRANSPOSE, transpose the data set **ia.econtrib**. Name the new SAS data set **ia.tcontrib**. It should be structured as shown below, with the following features:

- **QtrNum** as the name of the column that contains the quarter number
- one column that contains each unique employee contribution named **Amount**
- printing of the resulting data set

Partial Output

ia.econtrib

Obs	EmpID	Qtr1	Qtr2	Qtr3	Qtr4
1	E00224	$12.00	$33.00	$22.00	.
2	E00367	$35.00	$48.00	$40.00	$30.00
3	E00441	.	$63.00	$89.00	$90.00
4	E00587	$16.00	$19.00	$30.00	$29.00
5	E00598	$4.00	$8.00	$6.00	$1.00

Partial Output

ia.tcontrib

Obs	EmpID	Qtr Num	Amount
1	E00224	Qtr1	$12.00
2	E00224	Qtr2	$33.00
3	E00224	Qtr3	$22.00
4	E00224	Qtr4	.
5	E00367	Qtr1	$35.00
6	E00367	Qtr2	$48.00
7	E00367	Qtr3	$40.00
8	E00367	Qtr4	$30.00
9	E00441	Qtr1	.
10	E00441	Qtr2	$63.00
11	E00441	Qtr3	$89.00
12	E00441	Qtr4	$90.00
13	E00587	Qtr1	$16.00
14	E00587	Qtr2	$19.00
15	E00587	Qtr3	$30.00

4.6 Solutions to Exercises

1. Using a Two-Dimensional Array

The company recently sponsored a triathlon that involved bicycling (EVENT=1), swimming (EVENT=2), and running (EVENT=3). The finish order of the top four contestants in all events is stored in **`ia.compete`**. Use the following table and a two-dimensional array to determine the scores received for each event. The newly created SAS data set should be named **`results`**.

Event	1st Place	2nd Place	3rd Place	4th Place
1	65	55	45	35
2	80	70	60	50
3	70	60	50	40

Output

```
                              work.results

                Frst
   LastName     Name       Event    Finish    Score

   Tuttle       Thomas       1         1        65
   Gomez        Alan         1         2        55
   Chapman      Neil         1         3        45
   Welch        Darius       1         4        35
   Vandeusen    Richard      2         1        80
   Tuttle       Thomas       2         2        70
   Venter       Vince        2         3        60
   Morgan       Mel          2         4        50
   Chapman      Neil         3         1        70
   Gomez        Alan         3         2        60
   Morgan       Mel          3         3        50
   Tuttle       Thomas       3         4        40
```

```
data results;
   array Awards{3,4} _Temporary_  (65,55,45,35,
                                   80,70,60,50,
                                   70,60,50,40);
   set ia.compete;
   Score = Awards{Event,Finish};
run;

proc print data = results;
run;
```

2. Loading an Array from a SAS Data Set

The company recently sponsored a triathlon involving bicycling (**`EVENT = 1`**), swimming (**`EVENT = 2`**), and running (**`EVENT = 3`**). The finish order of the top four contestants in all events is stored in **`ia.compete`**. Use the **`ia.events`** data set, which contains the points awarded for each event and finish, and a two-dimensional array to determine the scores received for each event. The newly created SAS data set should be named **`results`**.

Output

work.results

LastName	Frst Name	Event	Finish	Score
Tuttle	Thomas	1	1	65
Gomez	Alan	1	2	55
Chapman	Neil	1	3	45
Welch	Darius	1	4	35
Vandeusen	Richard	2	1	80
Tuttle	Thomas	2	2	70
Venter	Vince	2	3	60
Morgan	Mel	2	4	50
Chapman	Neil	3	1	70
Gomez	Alan	3	2	60
Morgan	Mel	3	3	50
Tuttle	Thomas	3	4	40

```
data results (drop = i j first second third fourth);
   array awards{3, 4} _temporary_;
   if _n_ = 1 then do i = 1 to 3;
      set ia.events;
      array temp{4} first -- fourth;
      do j = 1 to 4;
      awards{i, j} = temp{j};
      end;
   end;
   set ia.compete;
   Score = Awards{Event,Finish};
run;

proc print data = results;
run;
```

3. Loading an Array from a SAS Data Set (Optional)

The **ia.mealplan** data set contains information on which meals, if any, are served on flights. Meal service is based on the day of the week (1 to 7) , **DOW**, and the hour of the day of the flight, **Hour**.

a. Produce a SAS data set named **meals** that contains the meal service code for each flight.

b. Use **ia.schedule** to obtain the flight information.

c. Create a two-dimensional array from **ia.mealplan**.

d. Look up the meal for each flight using the WEEKDAY function on **Date** and the HOUR function on **Depart**.

> The HOUR function returns values between 0 and 23. The **Hour** variable in **ia.mealplan** contains the values 1 to 24.

e. Print only the first 15 observations.

Output

```
                                   meals

              Obs    flight     depart         date    Service

                1    IA10800      6:35    01JUN2000    Breakfast
                2    IA10801      9:35    01JUN2000    None
                3    IA10802     12:35    01JUN2000    Snack
                4    IA10803     15:35    01JUN2000    None
                5    IA10804     18:35    01JUN2000    Dinner
                6    IA10805     21:35    01JUN2000    None
                7    IA10800      6:35    02JUN2000    Breakfast
                8    IA10801      9:35    02JUN2000    Snack
                9    IA10802     12:35    02JUN2000    Lunch
               10    IA10803     15:35    02JUN2000    Snack
               11    IA10804     18:35    02JUN2000    Dinner
               12    IA10805     21:35    02JUN2000    None
               13    IA10800      6:35    03JUN2000    Breakfast
               14    IA10801      9:35    03JUN2000    Snack
               15    IA10802     12:35    03JUN2000    Lunch
```

```
data meals;
   array food{7,24} $ 10 _Temporary_;
   if _n_ = 1 then do i = 1 to 7*24;
      set ia.mealplan;
      food{dow,hour} = Meal;
   end;
   set ia.schedule;
   Service = food{weekday(Date),hour(Depart)+1};
   keep Flight Date Depart Service;
run;

proc print data = meals(obs = 15);
   title 'meals';
run;
```

4. Using a Hash Object

a. Create a report that shows revenues, expenses, and profits for flights to Australia and New Zealand. Expenses for flights to Australia and New Zealand are in **ia.Dnunder** (900 observations). Revenues for all flights are in **ia.Sales** (about 330,000 observations).

b. Load the relevant data from **ia.Sales** in a hash object and use it as a lookup table for the flights in **ia.Dnunder**. Include the variables **FlightID**, **RouteID**, **FltDate**, **RevTotal**, **Expenses**, and **Profit** in the report. The variable **RevTotal** is the sum of **Rev1st**, **RevBus**, **RevEcon**, and **CargoRev**.

Partial Listing

```
                                    ia.Dnunder

                              Flight
                        Obs     ID         FltDate    Expenses

                          1    IA10200    01DEC2005     154269
                          2    IA10201    01DEC2005      71165
                          3    IA10200    02DEC2005      65188
                          4    IA10201    02DEC2005      14259
                          5    IA10200    03DEC2005     161419
```

Partial Listing

```
                                     ia.sales

      Flight
 Obs    ID       RouteID   Origin   Dest     DestType        FltDate      Cap1st     CapBus

   1  IA10700    0000107    WLG     AKL    International   01JAN2004         12          .
   2  IA10701    0000107    WLG     AKL    International   01JAN2004         12          .
   3  IA10702    0000107    WLG     AKL    International   01JAN2004         12          .
   4  IA10703    0000107    WLG     AKL    International   01JAN2004         12          .
   5  IA10704    0000107    WLG     AKL    International   01JAN2004         12          .

                  Cap                                Num
                 Pass                    Num   Num   Pass
 Obs   CapEcon   Total   CapCargo  Num1st  Bus   Econ   Total           Rev1st       RevBus

   1      138     150      36900     11     .     126    137         $1,397.00          .
   2      138     150      36900     12     .     136    148         $1,524.00          .
   3      138     150      36900     10     .     112    122         $1,270.00          .
   4      138     150      36900     12     .     113    125         $1,524.00          .
   5      138     150      36900     10     .     118    128         $1,270.00          .

                                                           Cargo
 Obs         RevEcon           CargoRev          RevTotal   Weight

   1       $5,292.00          $1,900.00            $8,589     9500
   2       $5,712.00          $1,460.00            $8,696     7300
   3       $4,704.00          $2,500.00            $8,474    12500
   4       $4,746.00          $2,380.00            $8,650    11900
   5       $4,956.00          $2,260.00            $8,486    11300
```

Partial Output

```
                        Profit for Flights to Australia and New Zealand

            Flight                                   Total
    Obs       ID       RouteID          Date       Revenue    Expenses     Profit

      1     IA10200    0000102     01DEC2000        359778      154269     205509
      2     IA10201    0000102     01DEC2000        361910       71165     290745
      3     IA10200    0000102     02DEC2000        357828       65188     292640
      4     IA10201    0000102     02DEC2000        358027       14259     343768
      5     IA10200    0000102     03DEC2000        356887      161419     195468
```

```
data Profit;
   if _n_ = 1 then do;
      if 0 then set ia.Sales
                (keep = FlightID RouteID FltDate RevTotal);
      declare hash ht(dataset: 'ia.Sales');
      ht.definekey ('FlightID', 'FltDate');
      ht.definedata('RouteID', 'RevTotal');
      ht.definedone();
   end;
   set ia.Dnunder;
   if ht.find() = 0 then do;
      Profit = RevTotal - Expenses;
      output;
   end;
   else putlog 'WARNING: _N_=' _N_ 'No match found. '
                FlightID= FltDate=;
run;

proc print data = work.Profit(obs = 5);
   title 'Profit for Flights to Australia and New Zealand';
   var FlightID RouteID FltDate RevTotal Expenses Profit;
run;
```

The PUTLOG statement writes text to the log.

General form of the PUTLOG statement:

PUTLOG '*text*';

Preceding the text with WARNING, ERROR, or NOTE displays the text in the color that SAS-generated warnings, errors, or notes are written to the log.

5. Creating a Format from a SAS Data Set

Use the **ia.jcodedat** data set to create a permanent format named **$jcodes**. View the new format using the FMTLIB option in PROC FORMAT.

```
data jcodes;
   keep Start Label FmtName;
   retain FmtName '$JCodes';
   set ia.jcodedat(rename = (JobCode = Start
                             Descript = Label));
run;

proc format library = ia cntlin = JCodes;
run;

options ls = 80;
proc format library = ia fmtlib;
   select $jcodes;
   title '$jcodes Format';
run;
```

6. Updating a Format (Optional)

Update an existing format by following these steps:

a. Add to the permanent **$jcodes** format.

b. Use the CNTLOUT= and CNTLIN= options in PROC FORMAT. Add new data for ticket agents using the INSERT statement in PROC SQL or a DATA step program.

TKTAG1	Ticket Agent Grade 1
TKTAG2	Ticket Agent Grade 2
TKTAG3	Ticket Agent Grade 3

c. View the new format using the FMTLIB option in PROC FORMAT.

```
proc format lib = ia cntlout = FmtData;
   select $jcodes;
run;

/*  SQL solution  */
proc sql;
   insert into fmtdata
      set FmtName = '$JCODES',
            Start = 'TKTAG1',
              End = 'TKTAG1',
            Label = 'Ticket Agent Grade 1'
      set FmtName = '$JCODES',
            Start = 'TKTAG2',
              End = 'TKTAG2',
            Label = 'Ticket Agent Grade 2'
```

(Continued on the next page.)

```
      set FmtName = '$JCODES',
            Start = 'TKTAG3',
              End = 'TKTAG3',
            Label = 'Ticket Agent Grade 3';
quit;

/*  DATA Step solution  */
data FmtData;
   set FmtData end = last;
   output;
   if last then do;
      FmtName = '$JCODES';
        Start = 'TKTAG1';
          End = 'TKTAG1';
        Label = 'Ticket Agent Grade 1';
   output;
      Start = 'TKTAG2';
        End = 'TKTAG2';
      Label = 'Ticket Agent Grade 2';
   output;
      Start = 'TKTAG3';
        End = 'TKTAG3';
      Label = 'Ticket Agent Grade 3';
   output;
   end;
run;

proc format library = ia cntlin = FmtData;
run;

proc format library = ia fmtlib;
   select $jcodes;
   title 'New values in the $JCODES Format';
run;
```

7. Using the TRANSPOSE Procedure

Using PROC TRANSPOSE, transpose the data set **`ia.econtrib`**. Name the new SAS data set **`ia.tcontrib`**. It should be structured as shown below, with the following features:

- **`QtrNum`** as the name of the column that contains the quarter number
- one column that contains each unique employee contribution named **`Amount`**
- printing of the resulting data set

Partial Output

ia.econtrib

Obs	EmpID	Qtr1	Qtr2	Qtr3	Qtr4
1	E00224	$12.00	$33.00	$22.00	.
2	E00367	$35.00	$48.00	$40.00	$30.00
3	E00441	.	$63.00	$89.00	$90.00
4	E00587	$16.00	$19.00	$30.00	$29.00
5	E00598	$4.00	$8.00	$6.00	$1.00

Partial Output

ia.tcontrib

Obs	EmpID	Qtr Num	Amount
1	E00224	Qtr1	$12.00
2	E00224	Qtr2	$33.00
3	E00224	Qtr3	$22.00
4	E00224	Qtr4	.
5	E00367	Qtr1	$35.00
6	E00367	Qtr2	$48.00
7	E00367	Qtr3	$40.00
8	E00367	Qtr4	$30.00
9	E00441	Qtr1	.
10	E00441	Qtr2	$63.00
11	E00441	Qtr3	$89.00
12	E00441	Qtr4	$90.00
13	E00587	Qtr1	$16.00
14	E00587	Qtr2	$19.00
15	E00587	Qtr3	$30.00

```
proc transpose data = ia.econtrib
   out = ia.tcontrib(rename = (col1 = Amount))
      name = QtrNum;
   by EmpID;
run;

proc print data = ia.tcontrib;
run;
```

Chapter 5 Combining Data Vertically

5.1 Appending SAS Data Sets

Objectives

- Append two SAS data sets using the APPEND procedure.
- Update a SAS data set using an INSERT INTO statement in the SQL procedure.

3

Vertical Combination Methods

SAS data can be combined vertically using one of these four methods:

- PROC APPEND
- the INSERT INTO statement in PROC SQL
- OUTER UNION CORRESPONDING set operator in PROC SQL
- DATA step SET statement

4 ...

This chapter discusses the APPEND procedure and the SQL procedure INSERT INTO statement.

Using the APPEND Procedure

The data set **emps** contains employees who were hired in the 1980s. The data set **newemps** contains employees who were hired in the 1990s.

You can use the APPEND procedure to concatenate two SAS data sets.

```
proc append base = emps
            data = newemps;
run;
```

5 c05s1d1

Log

```
113
114  proc append base = emps
115              data = newemps;
116  run;

NOTE: Appending WORK.NEWEMPS to WORK.EMPS.
NOTE: There were 655 observations read from the data set WORK.NEWEMPS.
NOTE: 655 observations added.
NOTE: The data set WORK.EMPS has 2070 observations and 5 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds

117
118  proc print data = ia.emps;
119     title 'All Employees Created';
120     title2 'by Appending ia.newemps to ia.emps';
121  run;

NOTE: There were 2070 observations read from the data set IA.EMPS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.02 seconds
```

Partial Output

```
                         All Employees Created
                    by Appending ia.newemps to ia.emps

 Obs LastName                  FirstName

1409 ROY                       SHEILA M.
1410 GUEGAN                    JOCELYNE
1411 JENSEN                    PIA
1412 HORTON                    SLAVA J.                  from ia.emps
1413 WARD                      PHILIP R.
1414 SUMMERS II                KAREN H.
1415 MORRIS                    MATTHEW
1416 MILLS                     DOROTHY E
1417 BADINE                    DAVID
1418 LEWIS                     JOSEPH                    from ia.newemps
1419 DBAIBO                    CATHRYN J.
1420 SIMPSON                   ARTHUR P.
<lines removed>
                                            Job
 Obs Division                   HireDate    Code

1409 AIRPORT OPERATIONS        20MAR1986    GRCREW
1410 AIRPORT OPERATIONS        30MAY1989    CHKCLK
1411 AIRPORT OPERATIONS        22MAR1980    CHKCLK
1412 AIRPORT OPERATIONS        06OCT1980    BAGSUP       from ia.emps
1413 FLIGHT OPERATIONS         17DEC1986    MECH02
1414 AIRPORT OPERATIONS        24JUL1985    FSVCLK
1415 FLIGHT OPERATIONS         16JUL1986    MECH02
1416 FLIGHT OPERATIONS         11MAR1992    FLTAT3
1417 CORPORATE OPERATIONS      15FEB1992    OFFMGR
1418 SALES & MARKETING         13JUL1994    MKTCLK       from ia.newemps
1419 HUMAN RESOURCES & FACILITIES 20SEP1991 RECEPT
1420 HUMAN RESOURCES & FACILITIES 13JAN1993 RESCLK
<lines removed>
```

Using the APPEND Procedure

General form of the APPEND procedure:

PROC APPEND BASE=*SAS-data-set*
DATA=*SAS-data-set*
<FORCE>;

Using the APPEND procedure preserves any indexes on the BASE= data set. The indexes are automatically updated with the observations in the DATA= data set after the data is appended.

6

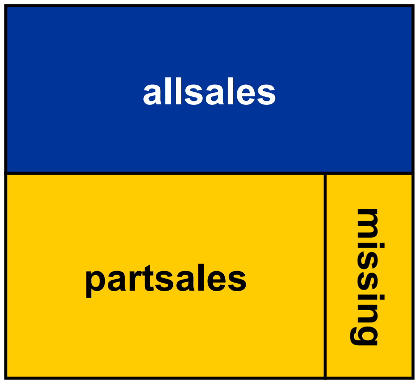
PROC APPEND only reads the data in the DATA= SAS data set, not in the BASE= SAS data set.

The FORCE option forces PROC APPEND to concatenate data sets when the DATA= data set contains variables that have any of the following characteristics:

- are not in the BASE= data set.
- do not have the same type as the variables in the BASE= data set. (For variables with a type mismatch, missing values are assigned in the appended observations when the FORCE option is used.)
- are longer than the variables in the BASE= data set.

Appending Fewer Variables

PROC APPEND concatenates the data sets even though there might be variables in the BASE= data set that do not exist in the DATA= data set.



7

To create **allsales** and **partsales**, execute the following program (c05ref1):

```
data allsales;
   set ia.sales(obs = 25);
run;

data partsales(keep = FlightID RouteID FltDate Rev: Cap: Num:);
   set ia.sales(firstobs = 26 obs = 40);
run;
```

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Partial Log

```
8
9    proc append base=allsales data=partsales;
10   run;

NOTE: Appending WORK.PARTSALES to WORK.ALLSALES.
WARNING: Variable Origin was not found on DATA file.
WARNING: Variable Dest was not found on DATA file.
WARNING: Variable DestType was not found on DATA file.
WARNING: Variable CargoRev was not found on DATA file.
WARNING: Variable CargoWeight was not found on DATA file.
NOTE: There were 15 observations read from the data set WORK.PARTSALES.
NOTE: 15 observations added.
NOTE: The data set WORK.ALLSALES has 40 observations and 21 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.67 seconds
      cpu time            0.06 seconds
```

The FORCE option is not required.

8 c05ref1

The **work.allsales** data set has 21 variables. The **work.partsales** data set has 16 variables.

Partial Output

```
proc print data=allsales(firstobs=23 obs=29);
   var Origin Dest DestType CargoRev CargoWeight;
   title 'Partial ALLSALES Data Set';
run;
```

Partial ALLSALES Data Set

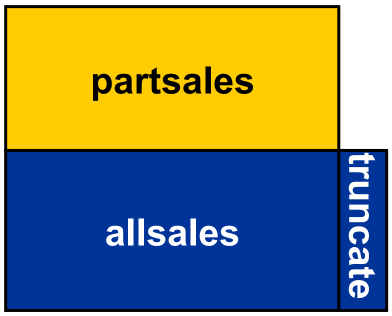
Obs	Origin	Dest	DestType	CargoRev	Cargo Weight
23	FRA	ATH	International	$23,501.00	33100
24	FRA	ATH	International	$23,501.00	33100
25	RDU	BHM	Domestic	$3,813.00	12300
26				.	.
27				.	.
28				.	.
29				.	.

Origin, **Dest**, **DestType**, **CargoRev**, and **CargoWeight** are in **allsales** but not in **partsales**.

9 c05ref1

FORCE Option

The FORCE option enables PROC APPEND to concatenate the data sets even though there might be variables in the DATA= data set that do not exist in the BASE= data set.



10 ...

The FORCE option can cause loss of data due to truncation or dropping variables.

To create **allsales** and **partsales**, execute the following program (c05ref2):

```
data allsales;
   set ia.sales(obs = 25);
run;

data partsales(keep = FlightID RouteID FltDate Rev: Cap: Num:);
   set ia.sales(firstobs = 26 obs = 40 rename = (RouteID = RouteNum));
   RouteID = input(RouteNum,10.);
run;
```

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Partial Log

```
51
52   proc append base=partsales data=allsales force;
53   run;

NOTE: Appending WORK.ALLSALES to WORK.PARTSALES.
WARNING: Variable Origin was not found on BASE file. The variable will not
be added to the BASE file.
WARNING: Variable Dest was not found on BASE file. The variable will not be
added to the BASE file.
WARNING: Variable DestType was not found on BASE file. The variable will not
be added to the BASE file.
WARNING: Variable CargoRev was not found on BASE file. The variable will not
be added to the BASE file.
WARNING: Variable CargoWeight was not found on BASE file. The variable will
not be added to the BASE file.
WARNING: Variable RouteID not appended because of type mismatch.
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 25 observations read from the data set WORK.ALLSALES.
NOTE: 25 observations added.
NOTE: The data set WORK.PARTSALES has 40 observations and 16 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.06 seconds
      cpu time            0.05 seconds
```

11 c05ref2

The **work.allsales** data set has 21 variables. The **work.partsales** data set has 16 variables.

The variable **RouteID** is character in the **work.allsales** data set. The variable **RouteID** is numeric in the **work.partsales** data set.

The type mismatch for **RouteID** and the additional variables present in **work.allsales** require the use of the FORCE option.

Partial Output

Partial PARTSALES Data Set

Obs	Flight ID	FltDate	Cap1st	CapBus	CapEcon	Cap Pass Total	CapCargo
14	IA03504	01JAN2004	12	.	138	150	36900
15	IA03505	01JAN2004	12	.	138	150	36900
16	IA10700	01JAN2004	12	.	138	150	36900
17	IA10701	01JAN2004	12	.	138	150	36900

Obs	Num1st	Num Bus	Num Econ	Num Pass Total	Rev1st	RevBus	RevEcon	RevTotal	Route ID
14	12	.	107	119	$2,232.00	.	$6,634.00	$12,665	35
15	11	.	127	138	$2,046.00	.	$7,874.00	$12,617	35
16	11	.	126	137	$1,397.00	.	$5,292.00	$8,589	.
17	12	.	136	148	$1,524.00	.	$5,712.00	$8,606	.

numeric

character

`Origin`, **`Dest`**, **`DestType`**, **`CargoRev`**, and **`CargoWeight`** are in **`allsales`** but not in **`partsales`**.

12

c05ref2

Appending Variables with Different Attributes

c05s1d2

```
proc contents data = airports;
run;
```

Partial Output

```
                              The CONTENTS Procedure

Data Set Name        WORK.AIRPORTS                     Observations         9397
Member Type          DATA                              Variables            4

                    Alphabetic List of Variables and Attributes

                #    Variable    Type    Len    Label

                2    City        Char     50    City Where Airport is Located
                1    Code        Char      3    Airport Code
                3    Country     Char     40    Country Where Airport is Located
                4    Name        Char     50    Airport Name
```

```
proc contents data = acities;
run;
```

Partial Output

```
                              The CONTENTS Procedure

Data Set Name        WORK.ACITIES                       Observations         52
Member Type          DATA                               Variables            4

                    Alphabetic List of Variables and Attributes

                #    Variable    Type    Len    Label

                1    City        Char     30    City Where Airport is Located
                2    Code        Char      3    Start Point
                4    Country     Char     40    Country Where Airport is Located
                3    Name        Char     50    Airport Name
```

```
proc append base = acities data = airports force;
run;

proc contents data = acities;
run;
```

Log

```
 proc append data=airports base=acities force;
 run;

NOTE: Appending WORK.AIRPORTS to WORK.ACITIES.
WARNING: Variable City has different lengths on BASE and DATA files
         (BASE 30 DATA 50).
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 9397 observations read from the data set WORK.AIRPORTS.
NOTE: 9397 observations added.
NOTE: The data set WORK.ACITIES has 9449 observations and 4 variables.
NOTE: PROCEDURE APPEND used:
      real time           0.04 seconds
      cpu time            0.04 seconds
```

Partial Output

```
                                    The CONTENTS Procedure

Data Set Name        WORK.ACITIES                     Observations          9449
Member Type          DATA                             Variables             4

                          Alphabetic List of Variables and Attributes

                #    Variable    Type    Len    Label

                1    City        Char     30    City Where Airport is Located
                2    Code        Char      3    Start Point
                4    Country     Char     40    Country Where Airport is Located
                3    Name        Char     50    Airport Name
```

```
proc contents data = allemps;
run;
```

Output

```
                                    The CONTENTS Procedure

Data Set Name        WORK.ALLEMPS                      Observations          550
Member Type          DATA                              Variables             5

                          Alphabetic List of Variables and Attributes

                                   #    Variable    Type    Len

                                   5    Division    Char     30
                                   1    EmpID       Char      6
                                   2    LastName    Char     15
                                   4    Location    Char     13
                                   3    Phone       Char      4
```

```
data pilots;
   keep phone Division LastName Location EmpID;
   set pilots(rename = (phone = ophone));
   phone = input(ophone,4.);
run;

proc contents data = pilots;
run;
```

Output

```
                                  The CONTENTS Procedure

Data Set Name        WORK.PILOTS                         Observations          31
Member Type          DATA                                Variables             5

                          Alphabetic List of Variables and Attributes

           #   Variable   Type   Len   Format   Informat   Label

           2   Division   Char    30   $30.     $30.       Division
           5   EmpID      Char     6   $6.      $6.        Employee Identification Number
           3   LastName   Char    32   $32.     $32.       Employee Last Name
           4   Location   Char    16   $16.     $16.       Employee Office Location
           1   Phone      Num      8
```

```
proc append base = allemps data = pilots force;
run;
```

Log

```
proc append base=allemps data=pilots force;
run;

NOTE: Appending WORK.PILOTS to WORK.ALLEMPS.
WARNING: Variable LastName has different lengths on BASE and DATA files
         (BASE 15 DATA 32).
WARNING: Variable Phone not appended because of type mismatch.
WARNING: Variable Location has different lengths on BASE and DATA files
         (BASE 13 DATA 16).
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 31 observations read from the data set WORK.PILOTS.
NOTE: 31 observations added.
NOTE: The data set WORK.ALLEMPS has 581 observations and 5 variables.
NOTE: PROCEDURE APPEND used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

```
proc print data = allemps;
   var phone;
run;
```

Output

```
                                 Obs    Phone

                                 547    1003
                                 548    1028
                                 549    1070
                                 550    1016
                                 551
                                 552
                                 553
                                 554
```

```
proc contents data = allemps;
run;
```

Output

```
                                     The CONTENTS Procedure

Data Set Name        WORK.ALLEMPS                       Observations          581
Member Type          DATA                               Variables             5

                            Alphabetic List of Variables and Attributes

                                   #    Variable    Type    Len

                                   5    Division    Char     30
                                   1    EmpID       Char      6
                                   2    LastName    Char     15
                                   4    Location    Char     13
                                   3    Phone       Char      4
```

Summary of APPEND Procedure

DATA= data set contains variables that …	Force Required?	Consequences
are not in the BASE= data set.	Yes	Extra DATA= data set variables are dropped.
do not have the same type as variables in the BASE= data set.	Yes	Data is not appended. Missing values are assigned to mismatched data.
are longer than the variables in the BASE= data set.	Yes	DATA= data set variable values are truncated.
are in the BASE= data set, but BASE= data set has more variables.	No	Missing values are assigned to extra BASE= data set variables.

14

Advantages of the APPEND Procedure

PROC APPEND has the following advantages:

- reads only the DATA= data set
- uses the FORCE option to concatenate data with different variable attributes
- updates indexes once at the end of the append

15

Disadvantages of the APPEND Procedure

PROC APPEND has the following disadvantages:

- can combine only two data sets
- cannot manipulate data
- cannot create a new (third) data set
- cannot change the descriptor portion of the BASE= data set

16

Using the SQL Procedure

The INSERT INTO statement in the SQL procedure can be used to add rows to a new or existing table or view.

There are three ways that INSERT INTO can be used:

- the SET clause to specify or alter the values of a column.
- the VALUES clause to insert lists of values into a table. (A value for each column in the table or values for only the columns specified in the list of column names must be specified.)
- a query expression to insert the results into a table.

Using the SQL procedure also preserves indexes.

17

When you use the INSERT INTO statement with a view, the view must reference one and only one table. The INSERT INTO statement cannot add rows to a view of joined tables.

The columns are matched positionally when you use the VALUES clause or a query expression to insert the results in a table. If the data types do not match, if there are more values than columns, or if there are fewer values than columns, the row is not inserted. Whether or not other rows are inserted depends on the current value of the UNDO_POLICY SQL statement option.

Using the INSERT INTO Statement

```
proc sql;
   insert into acities
      set City = 'Toronto',Code = 'YYZ',
          Name = 'Pearson International',     ①
          Country = 'Canada'
      set City = 'Montreal', Code = 'YUL',
          Name = 'Montreal Trudeau',
          Country = 'Canada';
quit;
```

```
PROC SQL;
     INSERT INTO table-name<(column<, ... column>)>
          SET column=sql-expression                ①
          <, ... column=sql-expression>
          <SET column=sql-expression
          <, ... column=sql-expression>>;
QUIT;
```

18 c05s1d3

① Each SET clause contains column names and their values separated by commas. The value for a column can be the result of a SELECT clause.

Log

```
proc sql;
   insert into acities
      set City = 'Toronto',Code = 'YYZ',
          Name = 'Pearson International',
          Country = 'Canada'
      set City = 'Montreal', Code = 'YUL',
          Name = 'Montreal Trudeau',
          Country = 'Canada';

NOTE: 2 rows were inserted into WORK.ACITIES.   ①
```

A partial log file is shown above.

Using the INSERT INTO Statement

```
proc sql;
   insert into acities(City, Code, Name, Country)
      values
 ➁  ('Toronto','YYZ','Pearson International','Canada')
      values
    ('Montreal','YUL','Montreal Trudeau','Canada');
quit;
```

```
PROC SQL;
      INSERT INTO table-name <(column<, ... column>)>
      VALUES (value <, ... value>)
      <... VALUES (value <, ... value>)>;        ➁
QUIT;
```

19 c05s1d4

➁ The VALUES clause is positional unless the columns are specified in the INSERT INTO clause.

Log

```
proc sql;
   insert into acities(City, Code, Name, Country)
      values
    ('Toronto','YYZ','Pearson International','Canada')
      values
    ('Montreal','YUL','Montreal Trudeau','Canada');

NOTE: 2 rows were inserted into WORK.ACITIES.   ➁
```

✎ A partial log file is shown above.

Using the INSERT INTO Statement

```
proc sql;
   insert into acities
③     select city, code, name, country from ia.airports
             where code in ('YYZ','YUL');
quit;
```

PROC SQL;
    INSERT INTO *table-name*
      SELECT <(*column*<, ...*column*>)> ③
    FROM *table-name query-expression*;
QUIT;

20 c05s1d5

③ The *query-expression* can be any SELECT clause.

Log

```
proc sql;
   insert into acities
      select city, code, name, country from ia.airports
             where Code in ('YYZ','YUL');
NOTE: 2 rows were inserted into WORK.ACITIES.   ③
```

✎ A partial log file is shown above.

Advantages of the SQL Procedure

PROC SQL with the INSERT INTO statement has the following advantages:

- only reads the data set on the FROM clause
- can manipulate data in the FROM data set only
- uses ANSI standard syntax
- maintains indexes

21

Disadvantages of the SQL Procedure

PROC SQL with the INSERT INTO statement has the following disadvantages:

- can combine only two data sets
- cannot create a new data set

22

Reference Information

Other techniques to concatenate SAS data sets:

DATA Step with SET Statement

Pros:

- This technique enables the full power of the DATA step to manipulate the data.
- Creation of a new data set occurs.
- An unlimited number of SAS data sets can be read.

Cons:

- All of the SAS data sets must be read.

PROC SQL with OUTER UNION CORRESPONDING

Pros:

- Data manipulation occurs in both data sets.
- There is a combination of joins and OUTER UNION CORRESPONDING.
- A new data set is created.
- ANSI standard syntax is used.

Cons:

- All data sets are read.

✎ Only the APPEND procedure and the INSERT INTO statement in the SQL procedure were discussed in this section.

Concatenation

	SET	PROC APPEND	SQL INSERT INTO	SQL OUTER UNION CORR
Data manipulation allowed	X		On second data set	X
Creation of a new data set	X			X
Unlimited number of SAS data sets	X			X
All SAS data sets must be read	X			X
Only one SAS data set must be read		X	X	

Exercises

1. Updating a Data Set Using the APPEND Procedure

Create the **work.quarter4** and **work.y2005** data sets by submitting the code in the **ProcCopy** program file:

```
proc copy in = ia out = work;
   select Quarter4 Y2005;
run;
```

Append **work.quarter4** to **work.y2005**. First, determine if the data sets have the same variables. The resulting data set should be **work.y2005** data with the additional observations from **work.quarter4**.

Partial Output: Added Observations

work.y2005 with Quarter4 Data

Obs	CrgoRev1	CrgoRev2	CrgoRev3	CrgoRev4
265	$3,281,364	$558,698	$2,094,261	$1,814,348
266	$3,296,780	$534,094	$2,403,148	$1,803,004
267	$3,317,456	$567,020	$2,155,557	$1,822,840
268	$3,279,250	$526,076	$1,893,366	$1,801,768
269	$3,260,316	$552,722	$2,133,225	$1,834,500
270	$3,243,090	$559,722	$2,337,188	$1,849,388
271	$3,293,606	$531,262	$2,132,043	$1,824,242
272	$3,268,782	$553,850	$2,114,361	$1,828,158
273	$3,227,646	$545,726	$2,369,204	$1,825,288
274	$3,287,060	$549,280	$2,132,679	$1,817,324
275	$3,281,134	$555,670	$1,917,524	$1,769,740
276	$3,270,620	$572,136	$2,102,609	$1,775,210
277	$3,296,466	$592,800	$2,352,088	$1,797,826
278	$3,299,664	$542,860	$2,102,151	$1,846,074
279	$3,283,118	$538,246	$2,135,697	$1,795,390
280	$3,212,646	$528,154	$2,403,092	$1,800,462

Obs	CrgoRev5	CrgoRev6	Date
265	$216,498	$1,229,390	22SEP2005
266	$233,466	$975,811	23SEP2005
267	$217,542	$943,923	24SEP2005
268	$219,428	$967,185	25SEP2005
269	$214,046	$985,297	26SEP2005
270	$212,828	$949,119	27SEP2005
271	$223,846	$943,461	28SEP2005
272	$219,926	$1,194,524	29SEP2005
273	$219,114	$974,305	30SEP2005
274	$219,792	$972,585	01OCT2005
275	$225,944	$984,625	02OCT2005
276	$215,386	$981,231	03OCT2005
277	$216,650	$941,179	04OCT2005
278	$217,364	$980,101	05OCT2005
279	$225,754	$1,211,148	06OCT2005
280	$213,560	$969,143	07OCT2005

2. Updating a Data Set Using the INSERT INTO Statement in the SQL Procedure (Optional)

Create the **work.quarter4** and **work.y2005** data sets by submitting the code in the **ProcCopy** program file:

```
proc copy in = ia out = work;
   select Quarter4 Y2005;
run;
```

Append **work.quarter4** to **work.y2005** using the INSERT INTO statement in the SQL procedure. First, determine if the data sets have the same variables. The resulting data set should be **work.y2005** data with the additional observations from **work.quarter4**.

Partial Output: Added Observations

work.y2005 with Quarter4 Data

Obs	CrgoRev1	CrgoRev2	CrgoRev3	CrgoRev4
265	$3,281,364	$558,698	$2,094,261	$1,814,348
266	$3,296,780	$534,094	$2,403,148	$1,803,004
267	$3,317,456	$567,020	$2,155,557	$1,822,840
268	$3,279,250	$526,076	$1,893,366	$1,801,768
269	$3,260,316	$552,722	$2,133,225	$1,834,500
270	$3,243,090	$559,722	$2,337,188	$1,849,388
271	$3,293,606	$531,262	$2,132,043	$1,824,242
272	$3,268,782	$553,850	$2,114,361	$1,828,158
273	$3,227,646	$545,726	$2,369,204	$1,825,288
274	$3,287,060	$549,280	$2,132,679	$1,817,324
275	$3,281,134	$555,670	$1,917,524	$1,769,740
276	$3,270,620	$572,136	$2,102,609	$1,775,210
277	$3,296,466	$592,800	$2,352,088	$1,797,826
278	$3,299,664	$542,860	$2,102,151	$1,846,074
279	$3,283,118	$538,246	$2,135,697	$1,795,390
280	$3,212,646	$528,154	$2,403,092	$1,800,462

Obs	CrgoRev5	CrgoRev6	Date
265	$216,498	$1,229,390	22SEP2005
266	$233,466	$975,811	23SEP2005
267	$217,542	$943,923	24SEP2005
268	$219,428	$967,185	25SEP2005
269	$214,046	$985,297	26SEP2005
270	$212,828	$949,119	27SEP2005
271	$223,846	$943,461	28SEP2005
272	$219,926	$1,194,524	29SEP2005
273	$219,114	$974,305	30SEP2005
274	$219,792	$972,585	01OCT2005
275	$225,944	$984,625	02OCT2005
276	$215,386	$981,231	03OCT2005
277	$216,650	$941,179	04OCT2005
278	$217,364	$980,101	05OCT2005
279	$225,754	$1,211,148	06OCT2005
280	$213,560	$969,143	07OCT2005

5.2 Appending Raw Data Files

Objectives

- Create a SAS data set from multiple raw data files using the FILENAME statement.
- Create a SAS data set from multiple raw data files using the FILEVAR= option.

25

Vertical Combination Methods

Raw data might be combined vertically using several methods:

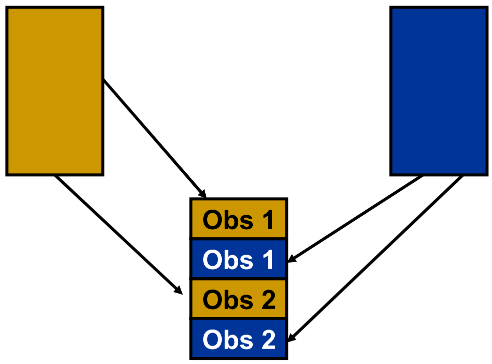
- concatenating files using multiple INFILE statements
- concatenating files using a FILENAME statement
- using the FILEVAR= option to read a list of files
- operating system techniques

26 ...

✎ Only the FILENAME statement and the FILEVAR= option are discussed in this section.

Reading Multiple Raw Data Files

To read multiple raw data files, you can use multiple INFILE statements.



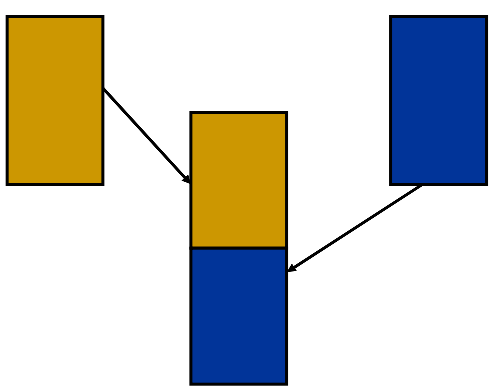
27 ...

Use multiple INFILE statements to read a record from one raw data file, a record from the second raw data file, a record from the third raw data file, and so on (similar to an interleave).

Multiple INFILE statements can be used to concatenate raw data files that have different file layouts.

Reading Multiple Raw Data Files

To read multiple raw data files, you can use the FILENAME statement.



28 ...

Use the FILENAME statement to concatenate multiple raw data files whose names can be hard-coded.

Using the FILENAME Statement

```
filename Q1 ('month1.dat' 'month2.dat'
             'month3.dat');
data firstq;
   infile Q1;
   input Flight $ Origin $ Dest $
         Date : date9.
         RevCargo : comma15.;
run;
```

Partial Listing of **month1.dat**

```
IA10200 SYD HKG 01JAN2005 $191,187.00
IA10201 SYD HKG 01JAN2005 $169,653.00
IA10300 SYD CBR 01JAN2005 $850.00
IA10301 SYD CBR 01JAN2005 $970.00
IA10302 SYD CBR 01JAN2005 $1,030.00
IA10303 SYD CBR 01JAN2005 $1,410.00
IA10304 SYD CBR 01JAN2005 $870.00
```

29 c05s2d1

Under z/OS (OS/390):

```
filename Q1 ('.prog3.rawdata(month1)'
             '.prog3.rawdata(month2)'
             '.prog3.rawdata(month3)');
```

Windows/UNIX Log

```
filename Q1 ('month1.dat' 'month2.dat' 'month3.dat');

 data firstq;
     infile Q1;
     input Flight $ Origin $ Dest $ Date : date9. RevCargo : comma15.;
 run;

NOTE: The infile Q1 is:
      File Name=c:\workshop\winsas\prog3\month1.dat,

      File List=('c:\workshop\winsas\prog3\month1.dat'
      'c:\workshop\winsas\prog3\month2.dat'
      'c:\workshop\winsas\prog3\month3.dat'),
      RECFM=V,LRECL=256

NOTE: The infile Q1 is:
      File Name=c:\workshop\winsas\prog3\month2.dat,

      File List=('c:\workshop\winsas\prog3\month1.dat'
      'c:\workshop\winsas\prog3\month2.dat'
      'c:\workshop\winsas\prog3\month3.dat'),
      RECFM=V,LRECL=256

NOTE: The infile Q1 is:
      File Name=c:\workshop\winsas\prog3\month3.dat,

      File List=('c:\workshop\winsas\prog3\month1.dat'
      'c:\workshop\winsas\prog3\month2.dat'
      'c:\workshop\winsas\prog3\month3.dat'),
      RECFM=V,LRECL=256

NOTE: 2299 records were read from the infile Q1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2090 records were read from the infile Q1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2297 records were read from the infile Q1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: The data set WORK.FIRSTQ has 6686 observations and 5 variables.
NOTE: DATA statement used:
      real time           0.31 seconds
      cpu time            0.12 seconds
```

FILENAME Statement Syntax

General form of the FILENAME statement:

FILENAME *fileref* ('*external-file1*'
'*external-file2*' … '*external-filen*');

fileref
is any SAS name that is eight characters or fewer.

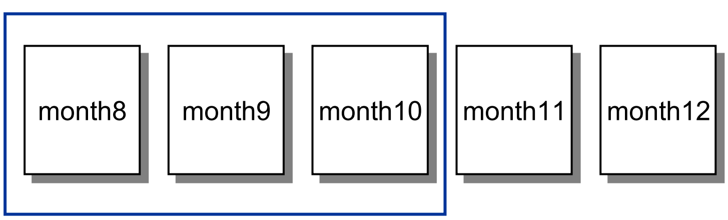
'*external-file*'
is the physical name of an external file. The physical name is the name that is recognized by the operating environment.

30

A FILENAME statement can associate a fileref with multiple physical external files.

Making the Program More Flexible

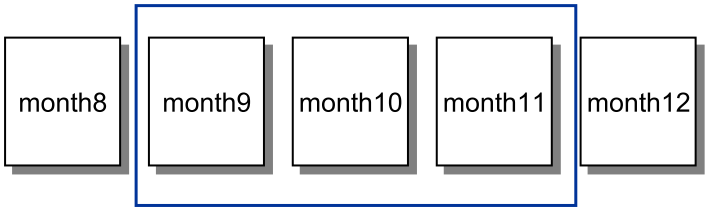
Provide reports of three months of data to IA executives. The three months are the current month and the previous two months (*rolling quarter*).



31 ...

Making the Program More Flexible

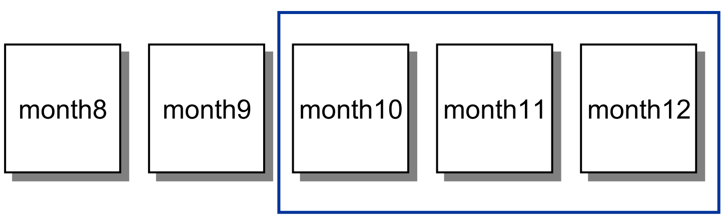
Provide reports of three months of data to IA executives. The three months are the current month and the previous two months (*rolling quarter*).



32 ...

Making the Program More Flexible

Provide reports of three months of data to IA executives. The three months are the current month and the previous two months (*rolling quarter*).



Use the FILEVAR= option in the INFILE statement to provide the name of the raw data file.

33

Creating the File Name

How can you change and assign the names of the three files to be read?

month + 9 + .dat
month + 10 + .dat
month + 11 + .dat

Creating the File Name

```
do I = 11,10,9;
   NextFile = "month"||put(I,2.)||".dat";
   infile zzz filevar = NextFile;
end;
```

When I = 11

NextFile = month11.dat

When I = 10

NextFile = month10.dat

When I = 9

NextFile = month 9.dat

Notice the space!

35 ...

The value of a FILEVAR= variable option is a character string that contains the physical filename of the raw data file to be read. When the next INPUT statement executes, it reads from the new file that the FILEVAR= variable option specifies. Similar to automatic variables, the FILEVAR= variable is not written to the data set.

The FILEVAR= variable option can read raw data files conditionally. You can construct the names of the raw data files programmatically.

✎ The concatenation characters can be !! or ||.

INFILE Statement with FILEVAR= Option

General form of the FILEVAR= variable option:

INFILE *file-specification* FILEVAR = *variable;*

FILEVAR = *variable*

names a variable whose change in value causes the INFILE statement to close the current input file and open a new one.

36

INFILE Statement with FILEVAR= Option

```
infile zzz filevar = NextFile;
```

zzz

is an arbitrarily named placeholder, not an actual filename or a fileref that was assigned to a file previously. SAS uses this placeholder for reporting processing information to the SAS log.

NextFile

contains the name of the raw data file to be read (**`month9.dat`**, **`month10.dat`**, **`month11.dat`**, and so on).

37

The placeholder must be eight characters or fewer, and must begin with an alpha character or underscore, followed by alphanumeric characters or underscores.

COMPRESS Function

To eliminate the space in filenames such as **`month 9.dat`**, use the COMPRESS function.

General form of the COMPRESS function:

COMPRESS(*source,<characters-to-remove>*)

source	specifies a source string that contains the characters to remove.
characters-to-remove	specifies the character or characters that SAS removes from the source string.

Example:

```
NextFile = compress(NextFile,' ');
```

38

If the *characters-to-remove* option is omitted, the COMPRESS function removes blanks from the *source*.

Reading Raw Data

```
data movingq;
   length Dest Origin $ 3 Flight $ 7;
   do I = 11,10,9;
      NextFile = "month"||put(I,2.)||".dat";❶
      NextFile = compress(NextFile,' ');❷
      infile zzz filevar = NextFile;❸
      input Flight $ Origin $ Dest $
            Date : date9.
            RevCargo : comma15.;
      output;❹
   end;
   stop;❺
run;
```

Why is the STOP statement needed?

How many observations are in **movingq**?

39 c05s2d2

① Creates the name of the raw data file.

② Removes any blanks from the name of the raw data file.

③ Names the raw data file. In addition, it closes the current file and opens the new file.

④ Outputs the observation that is created by the INPUT statement.

⑤ Stops the DATA step after all of the observations are written.

In this example, the DATA step does not encounter the end of file. If the STOP statement were not included, the program would continue to execute the DO loop repetitively. Therefore, the STOP statement is needed to prevent an infinite loop of the DATA step.

There are three observations in **movingq**.

Log

```
data movingq;
     length Dest Origin $ 3 Flight $ 7;
     do i = 11,10,9;
         NextFile = "month"||put(I,2.)||".dat";
         NextFile = compress(NextFile,' ');
         infile zzz filevar=NextFile;
         input Flight $ Origin $ Dest $ Date : date9. RevCargo : comma15.;
         output;
     end;
     stop;
run;

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month11.dat,
      RECFM=V,LRECL=256

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month10.dat,
      RECFM=V,LRECL=256

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month9.dat,
      RECFM=V,LRECL=256

NOTE: 1 record was read from the infile ZZZ.
      The minimum record length was 37.
      The maximum record length was 37.
NOTE: 1 record was read from the infile ZZZ.
      The minimum record length was 37.
      The maximum record length was 37.
NOTE: 1 record was read from the infile ZZZ.
      The minimum record length was 37.
      The maximum record length was 37.
NOTE: The data set WORK.MOVINGQ has 3 observations and 6 variables.
NOTE: DATA statement used:
      real time           0.15 seconds
      cpu time            0.01 seconds
```

Reading Raw Data

```
data movingq;
   length Dest Origin $ 3 Flight $ 7;
   do I = 11,10,9;
      NextFile = "month"||put(I,2.)||".dat";
      NextFile = compress(NextFile,' ');
      do until (LastObs); ❶                     ❷
         infile zzz filevar = NextFile end = LastObs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.;
         output;
      end;
   end;
   stop;
run;
```

How can the program always read the **current** month and previous two months?

40 c05s2d3

① The DO UNTIL statement continues to execute the INFILE statement for every record of the raw data file until the value of **`LastObs = 1`**. The DO UNTIL statement checks the condition at the bottom of the loop.

② The END= option creates the variable **`LastObs`** that can be used to determine the end of the raw data file. The END= option names a variable whose value is one of the following:

0 when the current input data record is not the last in the current input file

1 when the current input record is the last in the current input file

Partial Log

```
data movingq;
   length Dest Origin $ 3 Flight $ 7;
   do I = 11,10,9;
      NextFile = "month"||put(I,2.)||".dat";
      NextFile = compress(NextFile,' ');
      do until (LastObs);
         infile zzz filevar = NextFile end = LastObs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month11.dat,
      RECFM=V,LRECL=256

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month10.dat,
      RECFM=V,LRECL=256

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month9.dat,
      RECFM=V,LRECL=256

NOTE: 2195 records were read from the infile ZZZ.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2306 records were read from the infile ZZZ.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2215 records were read from the infile ZZZ.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: The data set WORK.MOVINGQ has 6716 observations and 6
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.18 seconds
      cpu time            0.07 seconds
```

Reading the Current Month

```
data movingq;
   length Dest Origin $ 3 Flight $ 7;
   drop MonNum MidMon LastMon I;
   MonNum = month(today()); ❶
   MidMon = MonNum-1; ❷
   LastMon = MidMon-1; ❷
   do I = MonNum, MidMon, LastMon;
      NextFile = "month"||put(i,2.)||".dat";
      NextFile = compress(NextFile,' ');
      do until (LastObs);
          infile zzz filevar = NextFile end = LastObs;
          input Flight $ Origin $ Dest $
                 Date : date9. RevCargo : comma15.;
          output;
      end;
   end;
   stop;
run;
```

41 c05s2d4

① Obtains the month number of today's date to begin the rolling month range.

② Calculates the month numbers of the two months prior to today's month number.

Calendar Logic

What if the current month is January or February?

42

INTNX Function

The INTNX function increments a date value by a given interval or intervals, and returns a date value.

```
EDate = intnx('interval',BDate, increment)
```

Formatted Value of **BDate**	Using the INTNX function	Formatted Value of **EDate**
04JUL2005	`intnx('year',BDate, -1)`	01JAN2004
04JUL2005	`intnx('year',BDate, 0)`	01JAN2005
04JUL2005	`intnx('year',BDate, 1)`	01JAN2006
04JUL2005	`intnx('year',BDate, 2)`	01JAN2007
04JUL2005	`intnx('month',BDate, -1)`	01JUN2005
04JUL2005	`intnx('month',BDate, 0)`	01JUL2005
04JUL2005	`intnx('month',BDate, 1)`	01AUG2005
04JUL2005	`intnx('month',BDate, 2)`	01SEP2005

43

The INTNX function can increment dates, time, or datetime values by a given interval or intervals, and returns a date, time, or datetime value.

INTNX Function

General form of the INTNX function:

INTNX(*'interval',start-from,increment<,alignment>*)

'interval'
specifies a character constant or variable of date, datetime, or time intervals.

start-from
specifies a SAS expression that represents a SAS date,datetime, or time value identifying a starting point.

increment
specifies a negative or positive integer that represents the specific number of time *intervals*.

44

Optional arguments:

INTNX(*interval<multiple><.shift-index>*, *start-from*, *increment<,alignment>*)

interval specifies a character constant, a variable, or an expression that contains a time interval such as WEEK, SEMIYEAR, QTR, or HOUR. The type of interval (date, datetime, or time) must match the type of value in start-from and increment.

multiple specifies a multiple of the interval. It sets the interval equal to a multiple of the interval type. For example, YEAR2 consists of two-year, or biennial, periods.

shift-index specifies the starting point of the interval. By default, the starting point is 1. A value that is greater than 1 shifts the start to a later point within the interval. The unit for shifting depends on the interval. For example, YEAR.3 specifies yearly periods that are shifted to start on the first of March of each calendar year and to end in February of the following year. The shift index cannot be greater than the number of periods in the entire interval. For example, YEAR2.24 has a valid shift index, but YEAR2.25 is invalid because there is no twenty-fifth month in a two-year interval. If the default shift period is the same as the interval type, then you can shift only multi-period intervals with the shift index. For example, because MONTH type intervals shift by MONTH sub-periods by default, you cannot shift monthly intervals with the shift index. However, you can shift bimonthly intervals with the shift index, because two MONTH intervals exist in each MONTH2 interval. The interval name MONTH2.2, for example, specifies bimonthly periods starting on the first day of even-numbered months.

start-from specifies a SAS expression that represents a SAS date, time, or datetime value that identifies a starting point.

increment specifies a negative, positive, or zero integer that represents the number of date, time, or datetime intervals. Increment is the number of intervals to shift the value of start-from.

alignment controls the position of SAS dates within the interval. Alignment can be one of these values:

BEGINNING | B	specifies that the returned date is aligned to the beginning of the interval. (DEFAULT)
MIDDLE | M	specifies that the returned date is aligned to the midpoint of the interval.
END | E	specifies that the returned date is aligned to the end of the interval.
SAMEDAY | S | SAME	specifies that the date that is returned is aligned to the same calendar date with the corresponding interval increment.

✎ *Alignment* is new in SAS®9.

Reading Multiple Raw Data Files

c05s2d5

```
data movingq;
   drop MonNum MidMon LastMon I;
   MonNum=month(today());
   MidMon=month(intnx('month',today(),-1));
   LastMon=month(intnx('month',today(),-2));
   do i=MonNum, MidMon, LastMon;
      NextFile="month"||put(i,2.)||".dat";
      NextFile=compress(NextFile,' ');
      do until (LastObs);
         infile zzz filevar=NextFile end=LastObs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.;
         output;
      end;
   end;
   stop;
run;
```

For z/OS (OS/390):

```
NextFile = '.prog3.rawdata(month'||put(i,2.)||')';
```

Log

```
data movingq;
   drop MonNum MidMon LastMon I;
   MonNum=month(today());
   MidMon=month(intnx('month',today(),-1));
   LastMon=month(intnx('month',today(),-2));
   do i=MonNum, MidMon, LastMon;
      NextFile="month"||put(i,2.)||".dat";
      NextFile=compress(NextFile,' ');
      do until (LastObs);
         infile zzz filevar=NextFile end=LastObs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.;
         output;
      end;
   end;
   stop;
run;

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month2.dat,
      RECFM=V,LRECL=256

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month1.dat,
      RECFM=V,LRECL=256

NOTE: The infile ZZZ is:
      File Name=c:\workshop\winsas\prog3\month12.dat,
      RECFM=V,LRECL=256

NOTE: 2090 records were read from the infile ZZZ.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2299 records were read from the infile ZZZ.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2190 records were read from the infile ZZZ.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: The data set WORK.MOVINGQ has 6579 observations and 5 variables.
NOTE: DATA statement used:
      real time           0.48 seconds
      cpu time            0.14 seconds
```

✎ This program was run in February.

Considering Efficiency

To make the program more efficient, call the TODAY function only once.

```
today = today();
MonNum = month(today);
MidMon = month(intnx('month',today,-1));
LastMon = month(intnx('month',today,-2));
```

46 c05s2d5a

c05s2d5a

```
data movingq;
  drop MonNum MidMon LastMon I today;
  today = today();
  MonNum = month(today);
  MidMon = month(intnx('month',today,-1));
  LastMon = month(intnx('month',today,-2));
  do i=MonNum, MidMon, LastMon;
     NextFile = "month"||put(i,2.)||".dat"; * PC and Unix;
     *Nextfile = ".prog3.rawdata(month"||put(i,2.)||")"; * mainframe ;
     NextFile=compress(NextFile,' ');
     do until (LastObs);
        infile xxx filevar=NextFile end=LastObs;
        input Flight $ Origin $ Dest $ Date : date9.
              RevCargo : comma15.2;
        output;
     end;
  end;
  stop;
run;
```

Instead of using the concatenate operator (|| or !!), you could use the concatenation functions.

Function	**Use**	**Example**
CAT	concatenates character strings without removing leading or trailing blanks.	`newvar = cat(var1,var);`
CATS	concatenates character strings and removes leading and trailing blanks.	`newvar = cats(var1,var);`
CATT	concatenates character strings and removes trailing blanks only.	`newvar = catt(var1,var);`
CATX	concatenates character strings, removes leading and trailing blanks, and inserts separators.	`newvar = catx(' ',var1,var);`

Caution: Without specifying the LENGTH of the new variable, the value of the new variable returned by any of the CAT functions has a length of up to the following:

- 200 characters in WHERE clauses and in PROC SQL
- 32,767 characters in the DATA step except in WHERE clauses
- 65,534 characters when string is called from the macro processor

Reference Information

Storing the Raw Data Filenames in a SAS Data Set

If raw data files that are to be read are in the SAS data set **ia.rawdata** shown below:

Obs	ReadIt
1	route1.dat
2	route2.dat
3	route3.dat
4	route4.dat
5	route5.dat

then you can use the following code:

c05ref3.sas

```
data route1_5;
   set ia.rawdata; ①
   infile zzz ② filevar = ReadIt ③ end = LastFile ④;
   do while(LastFile = 0); ⑤
      input  @1 RouteID $7.
             @8 Origin $3.
            @11 Dest $3.
            @14 Distance 5.
            @19 Fare1st 4.
            @23 FareBusiness 4.
            @27 FareEcon 4.
            @31 FareCargo 5.;
      output; ⑥
   end;
run;
```

① The data set **ia.rawdata** contains the variable named **Readit** whose value is the name of the raw data files: **month1**, **month2**, **month3**, **month4**, and **month5**.

② The letter grouping **zzz** is a placeholder, not an actual filename or a fileref that was previously assigned to a file. SAS uses this placeholder for reporting processing information to the SAS log. The placeholder is an arbitrary word; however, it must be eight characters or fewer, begin with an alpha character or underscore, followed by alphanumeric characters or underscores.

③ The FILEVAR= option specifies the value for the FILEVAR= variable. The INFILE statement closes the current file and opens a new one if the value of **Readit** changed when the INFILE statement executed.

④ **LastFile** is the arbitrary variable name created by the END= option. **LastFile** is a temporary variable and is set to 1 after each file is finished being read.

⑤ The DO WHILE loop checks the value of the variable **LastFile** at the top of the loop. Therefore, the INPUT statement reads from the current open INPUT file. Use a DO WHILE loop here, not a DO UNTIL loop. The DO UNTIL stops the DATA step if any file is empty.

⑥ The OUTPUT statement writes the contents of the Program Data Vector to create an observation of the SAS data set. The OUTPUT statement is required in this DATA step. Without the OUTPUT statement, the data set **route1_5** contains only six observations, that is, one per external file.

Storing the Raw Data Filenames in an External File

If the raw data files to be read are in the external file **rawfiles.dat** shown below:

```
route1.dat
route2.dat
route3.dat
route4.dat
route5.dat
```

then you can use the following code:

c05ref4.sas

```
data route1_5;
   infile 'rawfiles.dat';
   input ReadIt $ 10.; ①
   infile zzz ② filevar = ReadIt ③ end = LastFile ④;
   do while(LastFile = 0); ⑤
      input  @1 RouteID $7.
             @8 Origin $3.
            @11 Dest $3.
            @14 Distance 5.
            @19 Fare1stclass 4.
            @23 FareBusiness 4.
            @27 FareEcon 4.
            @31 FareCargo 5.;
      output; ⑥
   end;
run;
```

① The raw data file **rawfiles** contains the field whose value is the name of the raw data files, **month1**, **month2**, **month3**, **month4**, and **month5**. The INPUT statement reads the variable **ReadIt** of length 10.

② The letter grouping **zzz** is a placeholder, not an actual filename or a fileref that was previously assigned to a file. SAS uses this placeholder for reporting processing information to the SAS log. The placeholder is an arbitrary word; however, it must be eight characters or fewer, begin with an alpha character or underscore, followed by alphanumeric characters or underscores.

③ The FILEVAR= option specifies the value for the FILEVAR= variable. The INFILE statement closes the current file and opens a new one if the value of **Readit** changed when the INFILE statement executes.

④ **LastFile** is the arbitrary variable name created by the END= option. **LastFile** is a temporary variable and is set to 1 after each file is finished being read.

⑤ The DO WHILE loop checks the value of the variable **LastFile** at the top of the loop. Therefore, the INPUT statement reads from the current open INPUT file. Use a DO WHILE loop here, not a DO UNTIL loop. The DO UNTIL stops the DATA step if any file is empty.

⑥ The OUTPUT statement writes the contents of the Program Data Vector to create an observation of the SAS data set. The OUTPUT statement is required in this DATA step. Without the OUTPUT statement, the data set **route1_5** contains only six observations, that is, one per external file.

Exercises

3. Using the FILEVAR= Option

Concatenate the company's annual raw data files for the current year and previous two years using the FILEVAR= option. Create a SAS data set named **`last3`**.

The raw data files use the naming convention Y*yyyy*. For example:

For directory based: **`y2005.dat`**

For z/OS (OS/390): **`'.prog3.rawdata(y2005)'`**

Open the program **c05ex3Start**, which contains the following INPUT statement:

```
input Flight $ Date : date9. Depart : time5.;
```

Partial Output

```
                Three Years of Data

      Obs    Flight          Date    Depart

        1    IA00100    01JAN2005     7:00
        2    IA00101    01JAN2005    19:00
        3    IA00200    01JAN2005    23:30
        4    IA00201    01JAN2005    11:30
        5    IA00300    01JAN2005     7:30
        6    IA00301    01JAN2005    19:30
        7    IA00400    01JAN2005     1:30
        8    IA00401    01JAN2005    13:30
        9    IA00500    01JAN2005     6:30
       10    IA00501    01JAN2005    10:00
```

4. Using the FILENAME Statement

Use the FILENAME statement to concatenate the **`route3`** and **`route5`** raw data files and create a SAS data set named **`EuropeFlights`**. The raw data files are as follows:

For directory based: **`route3.dat`**

`route5.dat`

For z/OS (OS/390): **`'.prog3.rawdata(route3)'`**

`'.prog3.rawdata(route5)'`

Open the program **c05ex4Start**, which contains the following INPUT statement:

```
input @1 RouteID $7.
      @8 Origin $3.
      @11 Destination $3.
      @14 cargo 5.
      @19 totalpass 4.
      @23 boarded 4.
      @27 transfered 4.;
```

Partial Output

European Flights

Obs	RouteID	Origin	Destination	cargo	totalpass	boarded	transfered
1	0000002	LHR	RDU	3893	1600	1090	531
2	0000004	FRA	RDU	4288	1761	1201	585
3	0000043	LHR	CDG	223	91	62	30
4	0000044	CDG	LHR	223	91	62	30
5	0000045	LHR	GLA	347	142	97	47
6	0000046	GLA	LHR	347	142	97	47
7	0000047	LHR	FRA	397	163	111	54
8	0000048	FRA	LHR	397	163	111	54
9	0000049	LHR	BRU	207	85	57	28
10	0000050	BRU	LHR	207	85	57	28
11	0000051	LHR	GVA	465	190	130	63
12	0000052	GVA	LHR	465	190	130	63
13	0000055	FRA	FCO	595	244	167	81
14	0000056	FCO	FRA	595	244	167	81
15	0000057	FRA	CPH	424	174	118	57
16	0000059	CDG	MAD	644	265	180	88
17	0000060	MAD	CDG	644	265	180	88
18	0000061	CDG	LIS	899	369	251	123
19	0000062	LIS	CDG	899	369	251	123

5.3 Solutions to Exercises

1. **Updating a Data Set Using the APPEND Procedure**

   Create the **work.quarter4** and **work.y2005** data sets by submitting the code in the **ProcCopy** program file:

```
proc copy in = ia out = work;
   select Quarter4 Y2005;
run;
```

   Append **work.quarter4** to **work.y2005**. First, determine if the data sets have the same variables. The resulting data set should be **work.y2005** data with the additional observations from **work.quarter4**.

```
proc append data = quarter4 base = y2005 force;
run;
```

2. **Updating a Data Set Using the INSERT INTO Statement in the SQL Procedure (Optional)**

   Create the **work.quarter4** and **work.y2005** data sets by submitting the code in the **ProcCopy** program file:

```
proc copy in = ia out = work;
   select Quarter4 Y2005;
run;
```

   Append **work.quarter4** to **work.y2005** using the INSERT INTO statement in the SQL procedure. First, determine if the data sets have the same variables. The resulting data set should be **work.y2005** data with the additional observations from **work.quarter4**.

```
proc sql;
   insert into work.y2005(CrgoRev1, CrgoRev2, CrgoRev3,
                          CrgoRev4, CrgoRev5, CrgoRev6,
                          Date)
      select CrgoRev1, CrgoRev2, CrgoRev3, CrgoRev4,
             CrgoRev5, CrgoRev6, Date
      from work.quarter4;
quit;
```

3. Using the FILEVAR= Option

Concatenate the company's annual raw data files for the current year and previous two years using the FILEVAR= option. Create a SAS data set named **last3**.

The raw data files use the following naming convention: Y*yyyy*. For example:

For directory based: **y2005.dat**

For z/OS (OS/390): **'.prog3.rawdata(y2005)'**

Open the program **c05ex3Start**, which contains the following INPUT statement:

```
input Flight $ Date : date9. Depart : time5.;
```

Save your SAS program.

For directory based: ch3ex1.sas

For z/OS (OS/390): '.prog3.sascode(ch3ex1)'

```
data last3(drop=year thisyear);
   thisyear=year(today());
   do year=thisyear to thisyear-2 by -1;
      NextFile="y"||put(year,4.)||".dat";
      do until(Last);
         infile zzz filevar=NextFile end=Last;
         input Flight $ Date : date9. Depart : time5.;
         output;
      end;
   end;
   stop;
run;

proc print data=last3;
   format Date date9. Depart time5.;
   title 'Three Years of Data';
run;
```

4. Using the FILENAME Statement

Use the FILENAME statement to concatenate the **route3** and **route5** raw data files and create a SAS data set named **EuropeFlights**. The raw data files are as follows:

For directory based:	**route3.dat**
	route5.dat
For z/OS (OS/390):	**'.prog3.rawdata(route3)'**
	'.prog3.rawdata(route5)'

Open the program **c05ex4Start**, which contains the following INPUT statement:

```
input @1 RouteID $7.
      @8 Origin $3.
      @11 Destination $3.
      @14 cargo 5.
      @19 totalpass 4.
      @23 boarded 4.
      @27 transfered 4.;
```

```
filename europe ('route3.dat' 'route5.dat');  /* Windows/UNIX */
*filename europe ('.prog3.rawdata(route3)'
                  '.prog3.rawdata(route5)'); /* z/OS */

data EuropeFlights;
   infile europe;
   input @1 RouteID $7.
         @8 Origin $3.
         @11 Destination $3.
         @14 cargo 5.
         @19 totalpass 4.
         @23 boarded 4.
         @27 transfered 4.;
run;

title1 'European Flights';

proc print data=europeflights;
run;
```

Chapter 6 BY-Group Processing and Sorting

6.1 Introduction

Objectives

- Investigate the reasons for sorting data.
- Define BY-group processing.
- List alternatives to the SORT procedure.

3

Reasons for Sorting Data

Data is sorted to accomplish the following:

- reorder the data for reporting

Create a report with employees listed in alphabetical order.

- store ordered data to reduce data retrieval time

A WHERE statement executes faster if data is sorted by the variables used in the WHERE expression.

- enable BY-group processing in both DATA and PROC steps

Create individual reports for each employee.

4 ...

BY-Group Processing

BY-group processing has these characteristics:

- is a method of processing observations that are grouped or ordered by the values of common variables
- can be used in both DATA and PROC steps
- can be used to eliminate observations with duplicate BY values

These techniques can be used to perform BY-group processing:

- use the SORT procedure
- index the data set
- use the NOTSORTED option in the BY statement

5

Alternatives to Sorting

There are several alternatives to sorting data:

- indexing
- using grouped, but not sorted, data
- implementing user-sort assertion
- using a CLASS statement

6

6.2 Eliminating Duplicates

Objectives

- Use the NODUPKEY option.
- Use FIRST. and LAST. processing.
- Create a data set using the DUPOUT= option.

8

Using the NODUPKEY Option

NODUPKEY checks for and eliminates observations with duplicate BY values.

General form of the NODUPKEY option:

PROC SORT DATA = *data-set-name* NODUPKEY;

9

Reference Information

The NODUPRECS option checks for and eliminates duplicate **consecutive** observations.

PROC SORT DATA = *data-set-name* NODUPRECS;

The example below demonstrates the fact that duplicates might remain in the data set.

TABLE_ONE

A	B	C	D
1	3	5	8
1	3	5	8
2	4	6	8
1	2	8	6
1	3	5	8
2	5	7	3

```
proc sort data=table_one noduprecs;
   by a;
run;
```

TABLE_ONE (after the sort, but before NODUPRECS)

A	B	C	D	
1	3	5	8	
1	3	5	8	⇐ Removed
1	2	8	6	
1	3	5	8	
2	4	6	8	
2	5	7	3	

Only one row containing A = 1, B = 3, C = 5, and D = 8 is removed because it is the only consecutive row that contains those values.

```
proc sort data=table_one noduprecs;
   by a b c d;
run;
```

TABLE_ONE (after the sort with NODUPRECS on all variables)

A	B	C	D
1	2	8	6
1	3	5	8
2	4	6	8
2	5	7	3

The DATA step with FIRST. or LAST. has the advantage of additional data processing in the same step.

SORTDUP=PHYSICAL | LOGICAL
is a system option that controls how NODUPRECS processing works.

PHYSICAL removes duplicates based on all variables in the data set. This is the default.

LOGICAL removes duplicates based only on variables remaining after DROP= and KEEP= data set options are processed.

An example of using the SORTDUP= system option is shown below.

TABLE_ONE

A	B	C	D
1	3	5	8
1	3	8	6
1	3	8	6

```
options sortdup = physical; /* This is the default. */
proc sort data = table_one(drop = C D) noduprecs;
   by a b;
run;
```

TABLE_ONE

A	B
1	3
1	3

Because the first two rows are different before columns C and D are dropped, PROC SORT with the NODUPRECS option retains both rows in the output table when SORTDUP=PHYSICAL.

```
options sortdup = logical;
proc sort data = table_one(drop = C D) noduprecs;
   by a b;
run;
```

TABLE_ONE

A	B
1	3

Eliminate Duplicates

The data set **ia.allemps** contains data for both retired and current employees. Because the data was drawn from different sources, multiple observations were accidentally inserted for some employee ID numbers.

Create a new SAS data set that contains only one observation for each employee ID number.

ia.allemps (First Six Observations)

Obs	EmpID	LastName	Phone	Location	Division
1	E00010	FOSKEY	1666	CARY	AIRPORT OPERATIONS
2	E00015	BROWN	1263	CARY	AIRPORT OPERATIONS
3	E00025	BROCKLEBANK	1248	CARY	AIRPORT OPERATIONS
4	E00029	MAROON	1325	CARY	AIRPORT OPERATIONS
5	E00042	ANDERSON	1045	CARY	AIRPORT OPERATIONS
6	E00053	CURTIS	1468	CARY	AIRPORT OPERATIONS

10

DATA Step with FIRST. Processing

```
proc sort data = ia.allemps
          out = allemps;
   by EmpID;
run;

data allemps dups;
   set allemps;
   by EmpID;
   if first.EmpID then output
       allemps;
   else output dups;
run;
```

c06s2d1

11

Using the DUPOUT= Option

The SORT procedure DUPOUT= option specifies the output data set to which duplicate observations are written.

```
proc sort data = ia.allemps nodupkey ❶
          out = allemps
          dupout = dups;    ❷
   by EmpID;
run;
```

12 c06s2d2

① The NODUPKEY option selects duplicate observations based on the key value **EmpID**.

② The DUPOUT= option creates a data set named **dups** that contains the duplicate observations.

✎ The DUPOUT= option is new in SAS®9.

Partial **work.allemps** Data Set

Work.Allemps Data Set

Obs	EmpID	LastName	Phone	Location	Division
1	E00001	MILLS	2380	CARY	FLIGHT OPERATIONS
2	E00002	BOWER	1214	CARY	FINANCE & IT
3	E00003	READING	1428	CARY	HUMAN RESOURCES & FACILITIES
4	E00004	JUDD	2061	CARY	HUMAN RESOURCES & FACILITIES
5	E00005	WONSILD	1086	COPENHAGEN	AIRPORT OPERATIONS
6	E00006	ANDERSON	1007	CARY	SALES & MARKETING
7	E00007	MASSENGILL	2290	CARY	FLIGHT OPERATIONS
8	E00008	BADINE	1000	TORONTO	CORPORATE OPERATIONS
9	E00009	DEMENT	1506	CARY	FINANCE & IT
10	E00010	FOSKEY	1666	CARY	AIRPORT OPERATIONS
11	E00011	POOLE	2594	CARY	FLIGHT OPERATIONS
12	E00012	LEWIS	2207	CARY	SALES & MARKETING
13	E00013	DBAIBO	1002	BOSTON	HUMAN RESOURCES & FACILITIES
14	E00014	KEARNEY	2075	CARY	FLIGHT OPERATIONS
15	E00015	BROWN	1263	CARY	AIRPORT OPERATIONS
16	E00017	SIMPSON	2821	CARY	HUMAN RESOURCES & FACILITIES
17	E00018	CROSS	1459	CARY	HUMAN RESOURCES & FACILITIES
18	E00019	DANZIN	1005	BRUSSELS	SALES & MARKETING
19	E00020	JOHNSON	1256	CARY	HUMAN RESOURCES & FACILITIES
20	E00021	BAKER JR.	1001	HOUSTON	SALES & MARKETING

Partial **work.dups** Data Set

Work.Dups Data Set

Obs	EmpID	LastName	Phone	Location	Division
1	E00019	DANZIN	1012	CARY	AIRLINE OPERATIONS
2	E00059	BAUWENS	1001	BRUSSELS	SALES & MARKETING
3	E00068	PENDERGRASS	1060	SYDNEY	HUMAN RESOURCES & FACILITIES

Using the EQUALS | NOEQUALS Option

The EQUALS | NOEQUALS option specifies the order of the observations in the output data set.

- For observations with identical BY-variable values, EQUALS maintains the relative order of the observations within the input data set in the output data set.
- NOEQUALS does not necessarily preserve this order in the output data set.

13

EQUALS is the default.

Additionally, there is a new SAS global option, SORTEQUALS | NOSORTEQUALS, that enables you to globally disengage the stable sorting logic (EQUALS) that is on by default in the SORT procedure. SORTEQUALS is the shipped default to maintain backward compatibility, but NOSORTEQUALS is recommended.

EQUALS Option versus NOEQUALS Option

```
proc sort data = ia.allemps out = allemps
          nodupkey equals; ❶
   by EmpID;
run;
```

```
proc sort data = ia.allemps out = allemps
          nodupkey noequals; ❷
   by EmpID;
run;
```

When you use the NODUPKEY option to remove observations in the output data set, the choice of EQUALS or NOEQUALS can affect which observations are removed.

14 c06s2d3

① EQUALS maintains the relative order of the observations within the input data set in the output data set.

② NOEQUALS does not necessarily preserve this order in the output data set.

Using the EQUALS | NOEQUALS Option

Using NOEQUALS can save CPU time and memory.

However, with multi-threaded sort the following results might occur:

- Using the NOEQUALS option might result in the order of observations within BY groups being different in each run.
- Using the EQUALS option might reduce I/O performance because partitioned data sets will be processed as if they were non-partitioned data sets.

15

Exercises

1. Creating Data Sets with the SORT procedure

The data set **`ia.retirees`** is a list of recent retirees from International Airlines and contains duplicate observations. Create two data sets, one named **`retirees`** that contains unique rows of data for each employee ID number and the other named **`duprets`** containing the duplicate observations.

`Retirees` data set:

```
                          Retirees Data Set - After Duplicates Removed

Obs Division                         HireDate  LastName

  1 FINANCE & IT                    28DEC1945  LIMING
  2 AIRPORT OPERATIONS              03MAY1943  NOSCHKA
  3 FLIGHT OPERATIONS               02NOV1947  WALKER
  4 HUMAN RESOURCES & FACILITIES    16DEC1941  MILLER
  5 HUMAN RESOURCES & FACILITIES    21JUN1946  COOKE
  6 FINANCE & IT                    29APR1940  STROTHER
  7 FLIGHT OPERATIONS               06DEC1944  SHARMA
  8 FINANCE & IT                    22MAY1940  JAYAWICKRAMA
  9 SALES & MARKETING               26APR1950  SEDELL
 10 FLIGHT OPERATIONS               13DEC1945  ERICKSON
 11 FLIGHT OPERATIONS               03OCT1945  LEGEROS
 12 SALES & MARKETING               04JUN1944  BAYLOR JR.
 13 FINANCE & IT                    23MAY1943  MORRIS
 14 AIRPORT OPERATIONS              17AUG1946  PELLET

Obs FirstName                           EmpCountry

  1 RHONDA D.                           USA
  2 IRIS                                GERMANY
  3 CHARLES H.                          USA
  4 RAYMA M.                            USA
  5 HARALD                              GERMANY
  6 ROGER                               USA
  7 STEVEN                              UNITED KINGDOM
  8 LEWIS                               USA
  9 SANDRA                              USA
 10 KECIA H.                            USA
 11 SELBY                               USA
 12 JULIE R.                            USA
 13 MARK J.                             USA
 14 ISABELLE                            FRANCE
                                              Job
Obs EmpLocation         Phone       EmpID     Code

  1 CARY                    2215    E00369    FINACT
  2 FRANKFURT               1128    E00566    FLSCHD
  3 CARY                    3070    E00919    MECH01
  4 DALLAS                  1061    E01394    FACMNT
  5 FRANKFURT               1023    E01854    RESCLK
  6 CARY                    2910    E01976    ITCLK
  7 LONDON                  1131    E02044    PILOT1
  8 CARY                    2011    E02225    FINMGR
  9 SAN FRANCISCO           1009    E02663    MKTCLK
 10 CARY                    1156    E03083    FLTAT1
 11 CARY                    2186    E03292    FLTAT3
 12 DALLAS                  1004    E03486    SALCLK
 13 CARY                    2411    E03693    FINMGR
 14 PARIS                   1063    E04182    GRCSUP
```

(Continued on the next page.)

```
                     Retirees Data Set - After Duplicates Removed

Obs Division                         HireDate  LastName

 15 AIRPORT OPERATIONS              17NOV1941  FABIAN
 16 FLIGHT OPERATIONS               16FEB1947  HUMMEL

Obs FirstName                           EmpCountry

 15 GUENTER                             GERMANY
 16 THOMAS                              GERMANY

                                              Job
Obs EmpLocation         Phone       EmpID     Code

 15 FRANKFURT               1036    E04395    CHKCLK
 16 FRANKFURT               1071    E04614    MECH01
```

Duprets data set:

```
                                   Duprets Data Set

Obs Division                         HireDate  LastName

 1  FINANCE & IT                    28DEC1945  LIMING
 2  FLIGHT OPERATIONS               13DEC1945  ERICKSON
 3  FLIGHT OPERATIONS               03OCT1945  LEGEROS

Obs FirstName                           EmpCountry

 1  RHONDA D.                           USA
 2  KECIA H.                            USA
 3  SELBY                               USA

                                              Job
Obs EmpLocation         Phone       EmpID     Code

 1  CARY                    2215    E00369    FINACT
 2  CARY                    1156    E03083    FLTAT1
 3  CARY                    2186    E03292    FLTAT3
```

6.3 Sorting Resources

Objectives

- Define threading.
- Understand the workspace and library space required to sort a SAS data file.
- Estimate sort workspace.
- Allocate sort workspace.

18

Threading

In SAS®9, the SORT procedure is multi-threaded.

A *thread* is defined as the following:

- a single path of execution
- a basic unit of program execution in a thread-enabled operating environment

19

Multi-Threaded Processing

- *Multi-threaded processing* is a type of parallel processing introduced in SAS®9.
- *Parallel processing* means that multiple units of work are available to be scheduled for concurrent execution by the operating system.
- This technology takes advantage of hardware called *symmetric multiprocessing machines* (SMPs) that has multiple central processing units (CPUs).

20

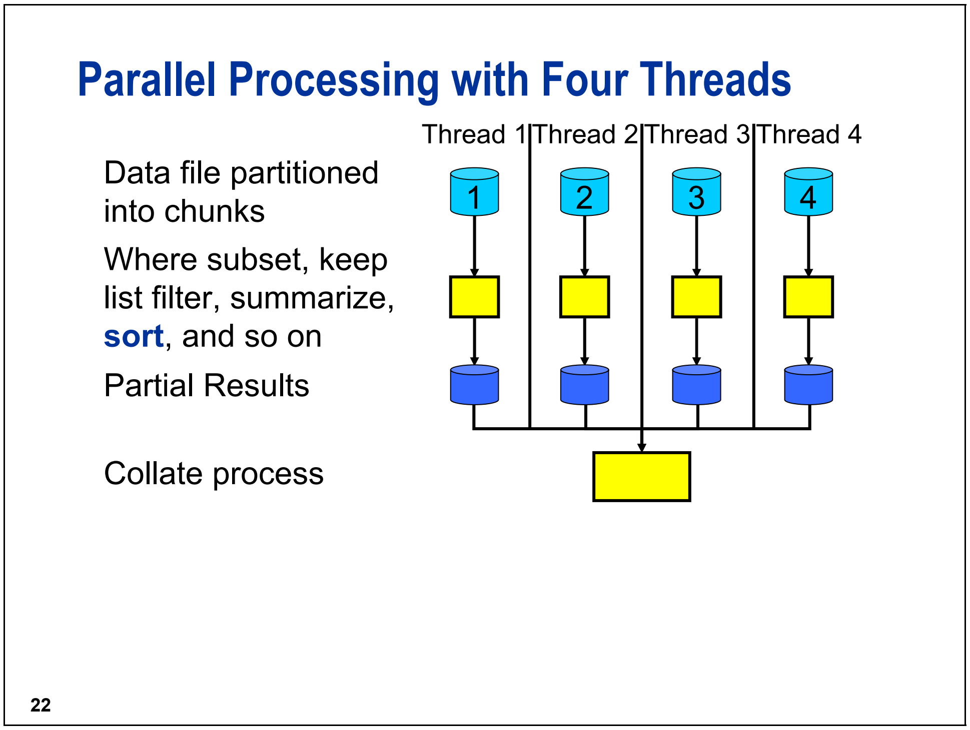
Multi-Threaded Processing

A *symmetric multiprocessing environment* possesses the following features:

- has multiple CPUs that share the same memory and a thread-enabled operating system
- can spawn and process multiple threads simultaneously using multiple CPUs
- enables the application to coordinate threads from the same process to share data very efficiently

21

- In an SMP computer environment, one instance of an operating system runs on several CPUs. Applications that run under this operating system can also run on several or all existing CPUs. All processes (operating system and applications) share the same memory and the same I/O resources.
- SMP systems are referred to as *shared everything* systems.
- One advantage of the SMP architecture is the ability to distribute the computational load dynamically over the existing CPUs and thus achieve equal loading of the CPUs.
- SMP systems can be arranged in multiple *clusters* to achieve even more scalability that often extends into 10 terabytes or more of data capacity and processing support.



In this example, four processing threads are created:

- Thread 1 starts reading and processing the first chunk of data.
- Thread 2 takes the second chunk of data.
- Thread 3 takes the third chunk of data.
- Thread 4 takes the fourth chunk of data.

The chunks of data are approximately equal in size and the size is generally the total number of observations in the data set size divided by the current value of the CPUCOUNT SAS system option. For example, if the total data set has 1,073,741,824 observations and the value of CPUCOUNT is 4, each thread has a chunk of data that is approximately 268,435,456 observations in size.

Multi-Threaded Processing

Processes suitable for threading are the following:

- sorting
- grouping
- summarizing

Threading can be enabled or disabled for the following Base SAS procedures:

- MEANS/SUMMARY
- REPORT
- SORT (excludes TAGSORT option)
- SQL (GROUP BY and ORDER BY)
- TABULATE

23

When you benchmark using the threaded procedures, use the Real Time statistic rather than the CPU time statistic. The back-end collating process to re-create the single data set might result in an increase in total CPU time, while reducing wall-clock time (time from submission of code for execution to return of results).

Threaded Procedures in Base SAS

Threaded processing can be controlled via the SAS system option THREADS | NOTHREADS.

```
OPTIONS THREADS | NOTHREADS;
```

The default is THREADS.

24

Threaded Procedures in Base SAS

- The THREADS | NOTHREADS option also can be specified in the PROC statement, which enables or disables multi-threaded processing of the input data set.
- When the option is specified in the PROC statement, the SAS system option THREADS | NOTHREADS is overridden.

Example:

```
PROC SORT DATA = SAS-data-set THREADS | NOTHREADS;
```

25

Threaded Procedures in Base SAS

The number of CPUs to use for processing can be controlled with the CPUCOUNT system option.

```
OPTIONS CPUCOUNT = 1-1024 | ACTUAL;
```

1-1024
: is the number of CPUs that SAS will assume are available for use by threaded-enabled applications.

ACTUAL
: is the number of CPUs that SAS detects are available for a specific session.

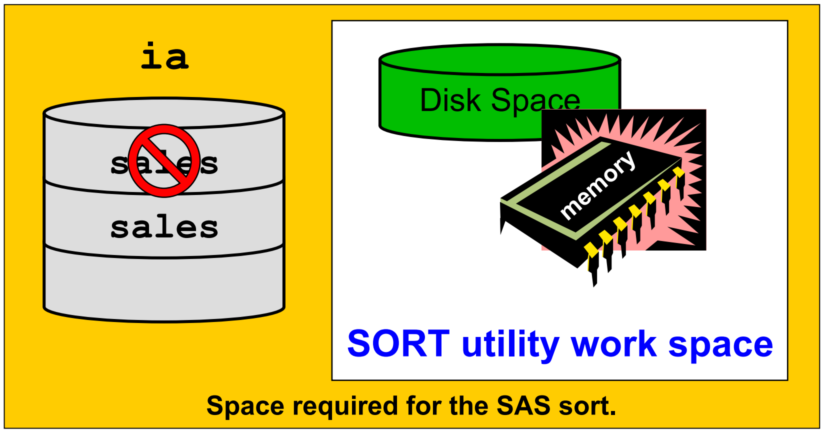
The default is ACTUAL.

26

✎ The SAS Administrator might have limited the number of CPUs that are available for SAS processing, so the value **ACTUAL** might be less than the total number of CPUs in the machine that SAS is using.

Sort Space Requirements

```
proc sort data = ia.sales force;
   by FltDate FlightID;
run;
```



27 ...

Sort Space Requirements

The amount of space that the SAS sort needs depends on the following conditions:

- whether the sorting can be accomplished with threading
- the length of the observations
- the number of variables in the BY statement
- the length of the variables in the BY statement
- the operating environment in which the PROC SORT executes
- the library to which the sorted data is written

28

Sort Space Requirements

By default, the space requirements of the SAS sort include the space for two copies of the original data set and the utility work space that can be split between disk and memory.

A quick rule of thumb method for **estimating** the space requirements for sorting with the SAS sort would be four times the size of the SAS data set being sorted.

This provides a "ballpark estimate" that is greatly influenced by the factors listed previously.

29

Estimating Sort Workspace (Self-Study)

The formula below calculates the **estimated** amount of space needed by a single-threaded PROC SORT.

bytes required = ((4 * obslen) + (2 * keylen)) * numobs

The formula below calculates the **estimated** amount of space needed by a multi-threaded PROC SORT.

bytes required = 3 * (obslen * numobs)

Use the CONTENTS or DATASETS procedure to gather the required information.

30

obslen length of the observation

keylen length of the BY variables when concatenated to form a single value

numobs number of observations in the data set

This space calculation assumes that the SAS® 9 sort can take place in memory, without using utility swap files.

The space calculation for the SAS Release 8.2 sort is as follows:

bytes required = (keylen + obslen) * numobs*N

where N = 4 (Windows and z/OS) or N = 5 (UNIX).

Estimating Sort Workspace

You want to sort the **ia.sales** data set by **FltDate** and **FlightID**. Before you submit the SORT procedure, submit this program:

```
proc contents data = ia.sales;
run;
```

c06s3d1

31

```
                          The CONTENTS Procedure

Data Set Name         IA.SALES                        Observations         329264
Member Type           DATA                            Variables            21
Engine                V9                              Indexes              2
Created      Monday, March 28, 2005 04:33:52 PM       Observation Length   168
```

```
               -----Engine/Host Dependent Information-----

Data Set Page Size           16384
Number of Data Set Pages     3396
```

```
         -----Alphabetic List of Variables and Attributes-----

 #  Variable    Type Len Pos Format  Informat Label
----------------------------------------------------------------------------
 1  FlightID    Char   7                      Flight Number
 6  FltDate     Num    8     DATE9.           Scheduled Date of Flight
```

32

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Estimating Single-Threaded Sort Workspace

Substitute the values into the equation to calculate the workspace required to sort the data using a single thread:

bytes required = ((4 * obslen) + (2 * keylen)) * numobs

((4 * 168) + (2 * (8+7)) * 329264 = 231,143,328 bytes

33

Estimating Multi-Threaded Sort Workspace

Substitute the values into the equation to calculate the workspace required to sort the data using multiple threads:

```
bytes required = 3 * ( obslen * numobs )
```

```
3 * (168  *  329264) = 165,949,056 bytes
```

34

In multi-threaded environments, if you use the OVERWRITE option in the PROC SORT statement, you need space equal to the data set size. The OVERWRITE option enables the input data set to be deleted before the replacement output data set is populated with observations. The OVERWRITE option is supported by the SAS sort and SAS multi-threaded sort only. The option has no effect if you use a host sort or the TAGSORT option.

Use the OVERWRITE option only with a data set that is backed up or with a data set that you can reconstruct. Because the input data set is deleted, data will be lost if a failure occurs while the output data set is being written.

Allocating Sort Workspace

If the actual required workspace is **less** than or equal to the value specified in the SORTSIZE= system or procedure option, then the entire sort can occur in memory, which reduces processing time.

If the actual required workspace is **greater** than the value specified in the SORTSIZE= option, then utility files on disk are required, which increases processing time.

The multi-threaded SAS 9.1.*x* sort fails to complete a sort if the value of SORTSIZE is too small.

35

Using the SORTSIZE= Option

Use the SORTSIZE= option to do the following:

- specify the amount of memory that is available to the SORT procedure
- improve the sort performance by restricting the swapping of memory that is controlled by the operating system

General form of the SORTSIZE= option:

SORTSIZE=*n* | *n*K | *n*M | *n*G | MIN | MAX | *hex*X | SIZE;

36

The SORTSIZE= Option to Increase Efficiency

If the SORT procedure requires more workspace than specified in SORTSIZE=, it performs the following tasks:

- creates a temporary utility file in the SAS Work directory or mainframe temporary area
- requests memory up to the value specified by SORTSIZE=
- writes partially sorted data to the utility file
- repeats the process until all the data is sorted
- combines the data in the utility files to create the final data set

37

The SORTSIZE= Option to Increase Efficiency

- The SORT procedure's algorithm can swap data more efficiently than the operating environment can because the procedure knows what data is needed and what is not.
- For optimal performance, set the SORTSIZE= option to a value less than the available physical memory, and allow programs and the operating environment to stay in memory.

38

Using the SORTSIZE= Option

You should investigate how resources are affected if you change the SORTSIZE= option.

```
proc sort data = ia.sales force
          sortsize = max;
   by FlightID FltDate;
run;
```

39

c06s3d2

6.4 Choosing the Right Sort Routine (Self-Study)

Objectives

- Understand the processing differences between host and portable sort utilities.
- Learn how to specify a particular sort utility.

41

What Is the Portable Sort?

The portable sort utility has the following characteristics:

- is supplied by SAS for all operating environments
- executes in memory up to the limit imposed by the SORTSIZE= option
- minimizes its use of external storage

42

What Is a Host Sort Utility?

- Third-party sort packages
- Available on Windows platforms in SAS Release 8.2 and later
- Available in UNIX and z/OS

Ask your system administrator if a host sort utility is available at your site.

43

Host Sort Utilities

Platform	Host Sort Utilities
z/OS	Dfsort * Syncsort
Unix	Syncsort *
Windows	Syncsort *

* Default

44

SAS System Options for Selecting a Host Sort

Use these options to select a host sort:

- SORTPGM=
- SORTCUTP=
- SORTNAME=

45

SORTPGM= System Option

The SORTPGM= system option specifies which sort utility to use.

General form of the SORTPGM= system option:

OPTIONS SORTPGM = *utility* | BEST | HOST | SAS;

46

utility names the host sort utility.

BEST specifies that SAS chooses the sort utility. This is the default in z/OS.

HOST specifies that the host sort utility is always used.

SAS specifies that the SAS sort utility is always used. This is the default on UNIX and Windows.

SORTPGM = BEST

- enables SAS to pass an ORDER BY clause to the DBMS when the SAS data set is accessed via a SAS/ACCESS engine.
- sorts data according to the DBMS sort rules, then the host sort rules, and then the SAS sort rules. (Sorting uses the first available and pertinent sorting algorithm in this list.) This is the default.

SORTCUTP= System Option

The SORTCUTP= system option specifies the size limit (in bytes) of a SAS data set. If the data set size is larger than the specified size, the host sort utility is used instead of the portable sort utility.

General form of the SORTCUTP= system option:

OPTIONS SORTCUTP=*n* | *n*K | *n*M | *n*G | MAX | MIN | *hex*X;

47

Under UNIX and Windows, SORTCUT= is an alias for SORTCUTP=.

Default SORTCUTP= System Option Values

Platform	Default SORTCUTP= Values
z/OS	4M *
UNIX	0 **
Windows	0 **

* SAS sort is used until this value is reached.

** SAS sort is always used.

48

SORTNAME= System Option

The SORTNAME= system option specifies the host sort utility to be invoked if SORTPGM=BEST | HOST.

General form of the SORTNAME= system option:

OPTIONS SORTNAME = *host-sort-utility-name*;

49

The SORTNAME= option is only required if you have more than one host sort installed at your site on your platform.

6.5 Alternatives to Sorting

Objectives

- Use indexes to return the data in sorted order.
- Use indexes to combine data horizontally.
- Use a format to group data for BY-group processing.
- Use a CLASS statement.

51

Using an Index for BY-Group Processing

- BY-group processing with an index eliminates the need to sort data.
- Having multiple indexes enables sequencing data by different variables without having to repeat the sort procedure.
- Indexes are updated when observations are modified or are added to a SAS data set, and thus eliminate the need to re-sort.

```
options msglevel = i;
proc print data = ia.sales(obs = 25);
   by Origin;
   var FlightID FltDate Cap1st CapBus CapEcon;
   title 'Using Indexes to Avoid a Sort';
run;
```

52 c06s5d1

Using an index for BY-group processing with Scalable Performance Data Engine data is discussed in a later chapter.

The data set **`ia.sales`** used for demonstrations and exercises contains fewer observations than the data set **`ia.sales`** used for the course notes.

Using an Index for BY-Group Processing

However, BY-group processing with an index has the following limitations:

- less efficient than sequentially reading a sorted data set
- storage space requirement for the index
- extra memory requirement to use the index

Partial Log

```
49  options msglevel = i;
450  proc print data=ia.sales(obs = 25);
451     by Origin;
INFO: Index Origin selected for BY clause processing.
NOTE: An index was selected to execute the BY statement.
      The observations will be returned in index order rather than in physical order.
      The selected index is for the variable(s):
 Origin
452     var FlightID FltDate Cap1st CapBus CapEcon;
453     title 'Using Indexes to Avoid a Sort';
454  run;

NOTE: There were 25 observations read from the data set IA.SALES.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

53 c06s5d1

Using an Index for BY-Group Processing

Partial Output

```
                         Using Indexes to Avoid a Sort

     ------------------- Start Point=AKL ------------------
                                      Flight
 Obs       ID         FltDate         Cap1st      CapBus      CapEcon

  447    IA10800    01JAN2004           12           .          138
  448    IA10801    01JAN2004           12           .          138
  449    IA10802    01JAN2004           12           .          138
  450    IA10803    01JAN2004           12           .          138
  451    IA10804    01JAN2004           12           .          138
  452    IA10805    01JAN2004           12           .          138
  898    IA10800    02JAN2004           12           .          138
  899    IA10801    02JAN2004           12           .          138
  900    IA10802    02JAN2004           12           .          138
  901    IA10803    02JAN2004           12           .          138
  902    IA10804    02JAN2004           12           .          138
  903    IA10805    02JAN2004           12           .          138
 1350    IA10800    03JAN2004           12           .          138
 1351    IA10801    03JAN2004           12           .          138
 1352    IA10802    03JAN2004           12           .          138
```

54

c06s5d1

BY Statement Does Not Use an Index

A BY statement does not use an index if the following conditions are present:

- The BY statement includes the DESCENDING or NOTSORTED option.
- SAS is aware that the data file is physically stored in sorted order on the BY variables.

55

Using Indexes

You can use the SET/SET statements with the KEY= option to avoid sorting a large data set when you merge a large SAS data set with a smaller data set that can be indexed.

1. The first SET statement names the data set that has the key values that will be used to retrieve observations from the second data set.
2. Specify the KEY= option in the second SET statement to use an index to retrieve observations.

General form of the KEY= option:

SET *SAS-data-file-name* KEY = *index-name*;

56

Use of the SET/SET statements with the KEY= option is also a good technique for merging a small driver data set with a larger indexed data set when only the matches are required to be returned.

Using Indexes

The SAS data set **ia.distances** contains the distance for each airline route.

Partial Data Set

RouteID	Distance
0000108	298
0000070	231
0000034	3480
0000032	2018
0000066	762
0000074	1130
0000024	480
0000096	893
0000036	442
.	.
.	.
0000103	147
0000102	4581
0000072	388
0000107	298
0000106	1446

57

Using Indexes

The data set **ia.sales** is not sorted by **RouteID**. There are two indexes on the data set, **Origin** and **DteFlt**. Neither of them can be used in the merge, and you do not want to sort the large data set.

Partial Data Set

Flight ID	RouteID	Origin	Dest	DestType	FltDate . . .
IA10700	0000107	WLG	AKL	International	01JAN2004 . . .
IA10701	0000107	WLG	AKL	International	01JAN2004 . . .
IA10702	0000107	WLG	AKL	International	01JAN2004 . . .
IA10703	0000107	WLG	AKL	International	01JAN2004 . . .
IA10704	0000107	WLG	AKL	International	01JAN2004 . . .
IA10705	0000107	WLG	AKL	International	01JAN2004 . . .
IA06900	0000069	LHR	AMS	International	01JAN2004 . . .
IA06901	0000069	LHR	AMS	International	01JAN2004 . . .
IA06902	0000069	LHR	AMS	International	01JAN2004 . . .

58

✎ The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Using Indexes

```
proc datasets lib = ia;
   modify distances;
   index create RouteID;
run;
quit;

data routes;           ❶         ❷             ❸
   set ia.sales;
   set ia.distances key = RouteID/unique;
run;
```

59

c06s5d2

① **ia.sales** is read sequentially.

② **ia.distances** is read by direct access using the index on **RouteID**.

③ The UNIQUE option causes a KEY= search to always begin at the top of the index.

Without the UNIQUE Option

When the UNIQUE option is not specified, the following events occur:

- Each change in the value of the KEY= variable(s) causes the SET statement to begin searching at the top of the index.
- Repeated values of the KEY= variable(s) cause the SET statement to retrieve successive observations that have duplicate values of the KEY= variables.
- If more consecutive duplicate KEY= values are specified than exist in the data set that is being read, _ERROR_ is set to 1 and _IORC_ is not equal to 0.

61

Without the UNIQUE Option

In this example, the UNIQUE option is needed because the data set **`ia.sales`** has duplicate **`RouteID`** values.

Without the UNIQUE option, the output is correct. However, the following conditions exist:

- The value of _IORC_ ne 0.
- The value of _ERROR_ = 1.
- The log contains data error messages.

62

Without the UNIQUE Option

Partial Log **Without** the UNIQUE Option

```
45  data routes_bad;
846    set ia.sales;
847    set ia.distances key = RouteID;
848 run;

FlightID=IA10701 RouteID=0000107 Origin=WLG Dest=AKL DestType=International
FltDate=01JAN2004
Cap1st=12 CapBus=. CapEcon=138 CapPassTotal=150 CapCargo=36900 Num1st=12
NumBus=. NumEcon=136
NumPassTotal=148 Rev1st=$1,524.00 RevBus=. RevEcon=$5,712.00
CargoRev=$1,460.00 RevTotal=$8,696
CargoWeight=7300 Distance=298 _ERROR_=1 _IORC_=1230015 _N_=2
FlightID=IA10702 RouteID=0000107 Origin=WLG Dest=AKL DestType=International
FltDate=01JAN2004
Cap1st=12 CapBus=. CapEcon=138 CapPassTotal=150 CapCargo=36900 Num1st=10
NumBus=. NumEcon=112
NumPassTotal=122 Rev1st=$1,270.00 RevBus=. RevEcon=$4,704.00
CargoRev=$2,500.00 RevTotal=$8,474
CargoWeight=12500 Distance=298 _ERROR_=1 _IORC_=1230015 _N_=3
. . . . . . . Lines omitted . . . . . . .
ERROR: Limit set by ERRORS= option reached. Further errors of this type
will not be printed.
 . . . . . . . Lines omitted . . . . . . .
```

63

Without the UNIQUE Option

Partial Output

Using the KEY= to Merge Data Sets

Obs	Flight ID	RouteID	FltDate	Origin	Dest	Distance
1	IA10700	0000107	01JAN2004	WLG	AKL	298
2	IA10701	0000107	01JAN2004	WLG	AKL	298
3	IA10702	0000107	01JAN2004	WLG	AKL	298
4	IA10703	0000107	01JAN2004	WLG	AKL	298
5	IA10704	0000107	01JAN2004	WLG	AKL	298
6	IA10705	0000107	01JAN2004	WLG	AKL	298
7	IA06900	0000069	01JAN2004	LHR	AMS	231
8	IA06901	0000069	01JAN2004	LHR	AMS	231
9	IA06902	0000069	01JAN2004	LHR	AMS	231
10	IA06903	0000069	01JAN2004	LHR	AMS	231

64

Using the NOTSORTED Option

The data set **`ia.lhr`** contains passenger count data for flights leaving from London's Heathrow Airport on January 1, 2005. The data set is sorted by destination, but not by city. However, the data is **grouped** by city.

Destination	City
BRU	Brussels
CDG	Paris
GLA	Glasgow
GVA	Geneva

Sorted by Destination

Grouped by City

65

Using the NOTSORTED Option

```
proc print data = ia.lhr;
   by City notsorted;
run;
```

The NOTSORTED option turns off sequence checking. If your data is **not** grouped, it can produce a very large amount of output.

66

c06s5d3

The data set **ia.lhr** is not sorted or grouped by **FlightID**.

c06s3d3a

```
title 'Printing ia.lhr by FlightID';
proc print data = ia.lhr;
   by FlightID notsorted;
run;
```

Partial Output

```
                         Printing ia.lhr by FlightID

     ----------------------- Flight Number=IA06900 ------------------------

                                                        Num
                                          Num    Num    Pass
       Obs   Dest     FltDate   Num1st   Bus   Econ   Total      City

         1   AMS    01JAN2005     13       .    102     115    Amsterdam


     ----------------------- Flight Number=IA06901 ------------------------

                                                        Num
                                          Num    Num    Pass
       Obs   Dest     FltDate   Num1st   Bus   Econ   Total      City

         2   AMS    01JAN2005     13       .    105     118    Amsterdam


     ----------------------- Flight Number=IA06902 ------------------------

                                                        Num
                                          Num    Num    Pass
       Obs   Dest     FltDate   Num1st   Bus   Econ   Total      City

         3   AMS    01JAN2005     12       .     95     107    Amsterdam


     ----------------------- Flight Number=IA06903 ------------------------

                                                        Num
                                          Num    Num    Pass
       Obs   Dest     FltDate   Num1st   Bus   Econ   Total      City

         4   AMS    01JAN2005     14       .    119     133    Amsterdam


     ----------------------- Flight Number=IA06904 ------------------------

                                                        Num
                                          Num    Num    Pass
       Obs   Dest     FltDate   Num1st   Bus   Econ   Total      City

         5   AMS    01JAN2005     14       .    103     117    Amsterdam


     ----------------------- Flight Number=IA06905 ------------------------

                                                        Num
                                          Num    Num    Pass
       Obs   Dest     FltDate   Num1st   Bus   Econ   Total      City

         6   AMS    01JAN2005     12       .    100     112    Amsterdam
```

Using the NOTSORTED Option

Partial Output

-------------------------City=Amsterdam--------------------------

Obs	Flight ID	Dest	FltDate	Num1st	Num Bus	Num Econ	Num Pass Total
1	IA06900	AMS	01JAN2005	13	.	102	115
2	IA06901	AMS	01JAN2005	13	.	105	118
3	IA06902	AMS	01JAN2005	12	.	95	107
4	IA06903	AMS	01JAN2005	14	.	119	133
5	IA06904	AMS	01JAN2005	14	.	103	117
6	IA06905	AMS	01JAN2005	12	.	100	112

---------------------City=Brussels (Bruxelles)-------------------

Obs	Flight ID	Dest	FltDate	Num1st	Num Bus	Num Econ	Num Pass Total
7	IA04900	BRU	01JAN2005	13	.	102	115
8	IA04901	BRU	01JAN2005	12	.	114	126
9	IA04902	BRU	01JAN2005	12	.	115	127
10	IA04903	BRU	01JAN2005	13	.	101	114
11	IA04904	BRU	01JAN2005	13	.	103	116
12	IA04905	BRU	01JAN2005	12	.	105	117

continued...

67

Using the NOTSORTED Option

-------------------------City=Paris-------------------------

Obs	Flight ID	Dest	FltDate	Num1st	Num Bus	Num Econ	Num Pass Total
13	IA04300	CDG	01JAN2005	13	.	116	129
14	IA04301	CDG	01JAN2005	14	.	98	112
15	IA04302	CDG	01JAN2005	11	.	122	133
16	IA04303	CDG	01JAN2005	11	.	112	123
17	IA04304	CDG	01JAN2005	12	.	95	107
18	IA04305	CDG	01JAN2005	13	.	119	132

-------------------------City=Frankfurt-------------------------

Obs	Flight ID	Dest	FltDate	Num1st	Num Bus	Num Econ	Num Pass Total
19	IA04700	FRA	01JAN2005	12	.	108	120
20	IA04701	FRA	01JAN2005	11	.	103	114
21	IA04702	FRA	01JAN2005	12	.	109	121
22	IA04703	FRA	01JAN2005	14	.	120	134

68

NOTSORTED Option for the BY Statement

The NOTSORTED option for the BY statement works best when observations with the same BY value are stored together, but are not necessarily sorted in alphabetical or numeric order.

General form of the NOTSORTED option:

BY *variable-name* **NOTSORTED**;

69

Tips for Using the NOTSORTED Option

The NOTSORTED option has the following features:

- can appear anywhere in the BY statement
- is useful if you have data that falls into other logical groupings, such as chronological order or categories
- can be used with **First.*variable*** and/or **Last.*variable***
- cannot be used with the MERGE and UPDATE statements

70

The BYSORTED SAS system option has the following characteristics:

- specifies that observations in a data set or data sets are sorted in alphabetic or numeric order
- should be used if the data set is ordered by the BY variable

```
OPTIONS BYSORTED;
```

If observations with the same BY value are grouped together but are not necessarily sorted in alphabetic or numeric order, use the NOBYSORTED option.

```
OPTIONS NOBYSORTED;
```

The default is BYSORTED.

When the NOBYSORTED option is specified, you do not have to specify NOTSORTED in every BY statement to access the data set(s).

GROUPFORMAT Option for the BY Statement

Create a summary report that provides the total cargo revenue for each quarter in 2000. The data for the report is in the SAS data set **`ia.revhistory`**.

`ia.revhistory` (First Eight Observations)

Obs	SaleMon	Month No	Year	First Class	Business	Economy	Cargo
1	JAN2000	1	2000	79065931	54229602	275767675	264931122
2	JAN2001	1	2001	80822951	55434704	281895846	270818480
3	JAN2002	1	2002	83458482	57242357	291088102	279649517
4	JAN2003	1	2003	85215503	58447460	297216272	285536876
5	JAN2004	1	2004	83667651	57385822	278553207	280350393
6	JAN2005	1	2005	87851034	60255113	306408528	294367913
7	FEB2000	2	2000	71641733	49018523	248842095	240029928
8	FEB2001	2	2001	73233772	50107824	254371920	245363926

71

Using the GROUPFORMAT Option

```
proc format;
   value $qtrfmt
         'JAN2000', 'FEB2000', 'MAR2000' = '1'
         'APR2000', 'MAY2000', 'JUN2000' = '2'
         'JUL2000', 'AUG2000', 'SEP2000' = '3'
         'OCT2000', 'NOV2000', 'DEC2000' = '4';
run;

data sum(keep = TotalCargo SaleMon
         rename = (SaleMon = qtr));
   set ia.revhistory;
   format SaleMon $qtrfmt.;
   by groupformat SaleMon notsorted;   ❶
   where year = 2000;
   if first.SaleMon then TotalCargo = 0;
   TotalCargo + Cargo;
   if last.SaleMon;
run;
```

72 c06s5d4 ...

① The GROUPFORMAT option enables the BY statement to use the $QTRFMT format to create FIRST.SALEMON and LAST.SALEMON. The NOTSORTED option is used because the data is grouped by **`SaleMon`** but not sorted by **`SaleMon`**.

Using the GROUPFORMAT Option

```
Using the GROUPFORMAT Option

qtr               TotalCargo

 1           $770,915,528.00
 2           $778,976,417.00
 3           $788,588,795.00
 4           $779,322,475.00
```

73

GROUPFORMAT Option for the BY Statement

The GROUPFORMAT option uses the formatted values, not the stored values, of the variable when you reference **first.*variable*** and **last.*variable*** in a DATA step.

General form of the GROUPFORMAT option:

BY GROUPFORMAT *variable-name* <NOTSORTED>;

74

First.*variable* and **last.*variable*** are temporary automatic variables in the PDV that identify the first and last observations in each BY-group.

Tips for Using the GROUPFORMAT Option

The GROUPFORMAT option has the following features:

- is available only in the DATA step
- is useful when you define formats for grouped data
- enables the DATA step to process the same groups of data as a summary procedure or PROC REPORT

75

Restrictions on the GROUPFORMAT Option

When the GROUPFORMAT option is used, the data set must meet one of the following conditions:

- be sorted by the GROUPFORMAT variable

or

- be grouped by the formatted values of the GROUPFORMAT variable

76

Advantages of NOTSORTED and GROUPFORMAT

The NOTSORTED and GROUPFORMAT options have the following advantages:

- can be used to create ordered/grouped reports without sorting the data
- cause the DATA step to process formatted BY values in the same way that SAS procedures do
- frequently eliminate the need for another step

77

Disadvantages of NOTSORTED and GROUPFORMAT

- The NOTSORTED option cannot be used with the MERGE or UPDATE statements.
- The NOTSORTED option can generate an enormous amount of output if the data is not grouped.
- The GROUPFORMAT option can only be used in the DATA step.

78

Using the CLASS Statement

Instead of using a BY statement to group data, you can use the CLASS statement to specify the variables whose values define the subgroup combinations for an analysis by a SAS procedure.

What are the differences between using a BY statement and using a CLASS statement in a procedure?

- The BY statement requires that you previously sorted the data by the BY variables or have an index based on the BY variables.
- The CLASS statement does not have either requirement.
- Report layouts differ.

79

PROC MEANS with a BY Statement

```
proc sort data = ia.sales(keep = FlightID
                               Rev1st--CargoRev)
          out = sales;
   by FlightID;
run;

proc means data = sales sum;
   by FlightID;
   var Rev1st--CargoRev;
run;
```

80 c06s5d5 ...

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

PROC MEANS with a BY Statement

Partial Report

```
------------------------------ Flight Number=IA00100 ---------------------------

                            The MEANS Procedure

          Variable    Label                                         Sum
          ----------------------------------------------------------------
          Rev1st      Revenue from First Class Passengers     14428800.00
          RevBus      Revenue from Business Passengers        21006480.00
          RevEcon     Revenue from Economy Passengers         55384362.00
          CargoRev    Revenue from Cargo                      81998560.00
          ----------------------------------------------------------------

------------------------------ Flight Number=IA00101 ---------------------------

          Variable    Label                                         Sum
          ----------------------------------------------------------------
          Rev1st      Revenue from First Class Passengers     14700800.00
          RevBus      Revenue from Business Passengers        21047900.00
          RevEcon     Revenue from Economy Passengers         55332324.00
          CargoRev    Revenue from Cargo                      81944660.00
          ----------------------------------------------------------------
```

81

PROC MEANS with a CLASS Statement

```
proc means data = ia.sales sum;
   class FlightID;
   var Rev1st--CargoRev;
run;
```

82 c06s5d6 ...

PROC MEANS with a CLASS Statement

Partial Report

```
                              The MEANS Procedure

Flight
Number     N Obs    Variable    Label                                    Sum
--------------------------------------------------------------------------------
IA00100      728    Rev1st      Revenue from First Class Passengers      14428800.00
                    RevBus      Revenue from Business Passengers         21006480.00
                    RevEcon     Revenue from Economy Passengers          55384362.00
                    CargoRev    Revenue from Cargo                       81998560.00

IA00101      728    Rev1st      Revenue from First Class Passengers      14700800.00
                    RevBus      Revenue from Business Passengers         21047900.00
                    RevEcon     Revenue from Economy Passengers          55332324.00
                    CargoRev    Revenue from Cargo                       81944660.00
```

83

Using the CLASS Statement to Group Data Values

General form of the CLASS statement:

CLASS *variable(s)* </ *options*>;

You can use the CLASS statement with the following Base SAS procedures:

- MEANS
- TABULATE
- SUMMARY
- UNIVARIATE

84

Reference Information

Selected options for the CLASS statement are as follows:

- ORDER = INTERNAL | FORMATTED | DATA | FREQ
  specifies the order in which to group the levels of the class variables in the output, where the following conditions can occur:

INTERNAL	orders values by ascending **unformatted** values. The INTERNAL order yields the same order as the SORT procedure. The order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically. INTERNAL is the default order. The term UNFORMATTED is an alias for INTERNAL.
DATA	orders values according to their order in the input data set.
FORMATTED	orders values by the ascending **formatted** values. This order depends on your operating environment.
FREQ	orders values by descending frequency count.

- DESCENDING specifies to sort the class variable values in descending order.
- MISSING considers missing values as valid class variable levels. Special missing values that represent numeric values (the letters A through Z and the underscore(_) character) are each considered as a separate value.
- GROUPINTERNAL specifies not to apply formats to the class variables when the MEANS, SUMMARY, or TABULATE procedures group the values to create combinations of class variables.

Using the SORTEDBY= Option

If the input data set is in sorted order, you can specify the order by using the SORTEDBY= output data set option.

The SORTEDBY= option has the following attributes:

- sets the sort flag on the data set
- defines the sort flag as an asserted data order
- requires that SAS check the order of the data as it processes it

General form of the SORTEDBY option:

> *data-set-name*(SORTEDBY = *by-clause* | _NULL_)

85

by-clause indicates the data order. You can specify variables and options as you can in a BY statement.

NULL removes any existing sort information.

Using the SORTEDBY= Option

Create a SAS data set from an external file containing invoice information. The external file is in sorted order by invoice number.

```
data invoices (sortedby = InvoiceID);
   infile extdata;
   input @1 InvoiceID $char4.
         @5 Supplier $char15.
         @30 Itemno $char4.
         @34 Amount comma8. ;
run;

proc contents data = invoices;
run;
```

c06s5d7

86

...

Using the SORTEDBY= Option

Partial Log

```
Data Set Name          WORK.INVOICES                          Observations          9
Member Type            DATA                                   Variables             4
Engine                 V9                                     Indexes               0
Created                Monday, June 06, 2005 01:48:38 PM     Observation Length    32
Last Modified          Monday, June 06, 2005 01:48:38 PM     Deleted Observations  0
Protection                                                    Compressed            NO
Data Set Type                                                 Sorted                YES
Label
Data Representation    WINDOWS_32
Encoding               wlatin1  Western (Windows)

      <lines removed>

                              Sort Information

                            Sortedby       InvoiceID
                            Validated      NO
                            Character Set  ANSI
```

87

Using the SORTEDBY= Option

Attempt to sort the data.

```
proc sort data = invoices;
   by InvoiceID;
run;
```

Log

```
77   proc sort data = invoices;
78      by InvoiceID;
79   run;

NOTE: Input data set is already sorted, no sorting done.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

88 c06s5d7

If a CONTENTS procedure is run after the PROC SORT, the Validated flag is still set to NO.

Partial Log

```
                    Sort Information

             Sortedby        InvoiceID
             Validated       NO
             Character Set   ANSI
```

To set the Validated flag to YES, use the FORCE option in the PROC SORT statement.

```
proc sort data = invoices force;
   by InvoiceID;
run;

proc contents data = invoices;
run;
```

Partial Log

```
                    Sort Information

             Sortedby        InvoiceID
             Validated       YES
             Character Set   ANSI
```

Exercises

2. Using the MEANS Procedure

The data set **`ia.crew`** is sorted by **`JobCode`** but not by **`JobCat`**. Use the MEANS procedure to calculate the total salary for each **`JobCat`** with the following conditions:

a. using a CLASS statement

b. using the BY statement without sorting the data

Desired Output for **a**.

```
                    Using the CLASS Statement

                       The MEANS Procedure

                  Analysis Variable : Salary

                                     N
         JobCat                    Obs              Sum
         ----------------------------------------------
         Flight Attendant           32        991000.00

         Navigator                   8        556000.00

         Pilots                     17       1520000.00

         Senior Flight Attendant    12        531000.00
         ----------------------------------------------
```

Desired Output for **b**.

```
                        Using the BY Statement

       ------------------- JobCat=Flight Attendant --------------------

                         The MEANS Procedure

                     Analysis Variable: Salary

                                     Sum
                            ____________
                               991000.00
                            ____________

       ---------------- JobCat=Senior Flight Attendant ----------------

                     Analysis Variable: Salary

                                     Sum
                            ____________
                               531000.00
                            ____________

       ----------------------- JobCat=Navigator -----------------------

                     Analysis Variable: Salary

                                     Sum
                            ____________
                               556000.00
                            ____________
       ------------------------ JobCat=Pilots -------------------------

                     Analysis Variable: Salary

                                      Sum
                             ____________
                               1520000.00
                             ____________
```

3. Creating a Sorted Data Set

Open the program, c06ex3Start, which contains the following INFILE and INPUT statements:

```
infile 'operate.dat';  *PC/UNIX;
*'.prog3.rawdata(operate)';  *z/OS;
input HireDate : date9. LastName : $32.
         FirstName : $32. EmpCountry : $25.
         EmpLocation : $16.
         EmpID $ JobCode $;
```

Create a SAS data set named **`oper`** from the comma-separated raw data file, **`operate`**, that is sorted by **`JobCode`**. Print the data in sorted order without presorting the operations data.

Partial List of the Raw Data File **operate**

```
28DEC1986,KRISCHOCK,JENNIFER ANNE,AUSTRALIA,SYDNEY,E02912,BAGCLK
11JUN1991,LAHANE,SEAN,AUSTRALIA,SYDNEY,E00253,BAGCLK
19AUG1986,LAI,CRAIG NEIL,AUSTRALIA,SYDNEY,E02197,BAGCLK
19MAY1993,LAWS,MERIAN,AUSTRALIA,SYDNEY,E02314,BAGCLK
20JUL1980,LINDSAY,ROBERT,AUSTRALIA,SYDNEY,E03113,BAGCLK
23APR1987,LONG,CARMEN,AUSTRALIA,SYDNEY,E03179,BAGCLK
02AUG1989,LOWRIE,KERRIE,AUSTRALIA,SYDNEY,E03421,BAGCLK
```

Partial Output from the PRINT Procedure (page 3 of output if the **OPTIONS PS=60 LS=120;** statement is submitted)

```
------------------------------------ JobCode=BAGCLK ------------------------------------
                                        (continued)

                Hire                                         Emp
        Obs     Date    LastName        FirstName          Country    EmpLocation        EmpID

        106     8082    HOWELL          MARY B.              USA      DALLAS             E01297
        107     7668    HUBBARD         VELERIE              USA      DALLAS             E00649
        108    11528    HURT            SUSAN L.             USA      CARY               E03548
        109     9886    JACKSON         YIQUN                USA      DALLAS             E03415
        110    10710    JACOBSON        SANDRA L.            USA      CARY               E04667
        111     9528    JENSEN          OREN                 USA      DALLAS             E02872
        112    12464    JONES           MARY B.              USA      CARY               E02739
        113     9435    JONES           MARY C.              USA      CARY               E01527
        114     9390    JONES           MICHAEL E.           USA      CARY               E00062
        115     7683    JONES JR.       THOMAS J.            USA      DALLAS             E04640
        116     7521    JORDAN          THOMAS F.            USA      DALLAS             E04071
        117     7459    KERR            BRADFORD E           USA      DALLAS             E01481
        118    10360    KLEIN           SUSANNE G.           USA      DALLAS             E01263
        119    11876    KOELLING        JAMES M.             USA      DALLAS             E00932
        120     8538    LIN             BARBARA J.           USA      NEW YORK           E03405
        121     9171    LORENCE         MATTHEW G.           USA      NEW YORK           E03839
        122     8486    LUMSDEN         TIMOTHY J.           USA      CARY               E00971
        123    12461    LUTZ            CATHRYN J.           USA      CARY               E04514
        124    12701    MACCORMICK      DAVID C.             USA      CARY               E01455
        125    12054    MARSHALL        MARY S.              USA      NEW YORK           E02991
        126    12269    MATZ            JACQUELYN            USA      NEW YORK           E03395
        127     8851    MCCUE           JIN-WHAN             USA      NEW YORK           E03724
        128     8056    MCLEAN          MICHAEL J.           USA      NEW YORK           E04655
        129     8754    MULLIGAN        STEPHEN A.           USA      NEW YORK           E00260
        130     9927    NICHOLSON       MARGARET F.          USA      NEW YORK           E03311
        131     9725    ONG             MICHELLE A.          USA      NEW YORK           E00916
        132     8948    REDPATH         CHERYL L.            USA      CARY               E01745
        133    12745    RODGERS         CONNIE S.            USA      SAN FRANCISCO      E00957
        134     8746    ROGERS          JASON W.             USA      CARY               E01048
        135     7625    SMITH           LINDA                USA      CARY               E01382
        136    11738    SMITH           MARTIN P.            USA      CARY               E00201
        137    12165    VAUGHAN         SHELLY               USA      CARY               E00371
        138     8990    ZEID            DOUGLAS H.           USA      SAN FRANCISCO      E02213
        139    12086    ZHANG           VIRGIL S.            USA      CARY               E04713
```

(Continued on the next page.)

```
---------------------------------- JobCode=BAGSUP --------------------------------------

              Hire                                                      Emp
      Obs     Date    LastName       FirstName     EmpCountry        Location      EmpID

      140     9382    JONES          MICHAEL A.    AUSTRALIA         SYDNEY        E00368
      141     7318    HUGHES         MONICA S.     CANADA            TORONTO       E03523
      142     7887    TANG           AARON         CHINA             HONG KONG     E02786
      143     7788    KEJSER         NIKOLAJ       DENMARK           COPENHAGEN    E00642
      144    12254    LAFOSSE        LOUIS         FRANCE            PARIS         E02892
      145     7719    FENERTY        WERNER        GERMANY           FRANKFURT     E03513
      146    10920    FRITZ          HORST         GERMANY           FRANKFURT     E01591
      147     7347    GOLFIERI       MARGHERITA    ITALY             ROME          E03553
      148     8449    NAGASAWA       KATSUMI       JAPAN             TOKYO         E03139
      149     8933    POPP           MATTHIAS      SWITZERLAND       GENEVA        E01099
      150    10464    FITZPATRICK    MICHAEL       UNITED KINGDOM    LONDON        E01219
```

6.6 Solutions to Exercises

1. Creating Data Sets with the SORT Procedure

The data set **ia.retirees** is a list of recent retirees from International Airlines and contains duplicate observations. Create two data sets, one named **retirees** that contains unique rows of data for each employee ID number and the other named **duprets** containing the duplicate observations.

```
proc sort data = ia.retirees out = retirees
          dupout = duprets nodupkey;
   by EmpID;
run;

/*  alternative solution  */
proc sort data = ia.retirees out = retirees;
   by EmpID;
run;

data retirees duprets;
   set retirees;
   by EmpID;
   if first.EmpID then output retirees;
   else output duprets;
run;
```

2. Using the MEANS Procedure

The data set **ia.crew** is sorted by **JobCode** but not by **JobCat**. Use the MEANS procedure to calculate the total salary for each **JobCat** with the following conditions:

a. using a CLASS statement

```
proc means data=ia.crew sum;
   class JobCat;
   var Salary;
   title 'Using the CLASS Statement';
run;
```

b. using the BY statement without sorting the data.

```
proc means data = ia.crew sum;
   by JobCat notsorted;
   var Salary;
   title 'Using the BY Statement';
run;
```

3. Creating a Sorted Data Set

Open the program, c06ex3Start, which contains the following INPUT and INFILE statements.

```
infile 'operate.dat';  *PC/UNIX;
*'.prog3.rawdata(operate)';  *z/OS;
input HireDate : date9. LastName : $32.
         FirstName : $32. EmpCountry : $25.
         EmpLocation : $16.
         EmpID $ JobCode $;
```

Create a SAS data set named **oper** from the comma-separated raw data file, **operate**, that is sorted by **JobCode**. Print the data in sorted order without presorting the operations data.

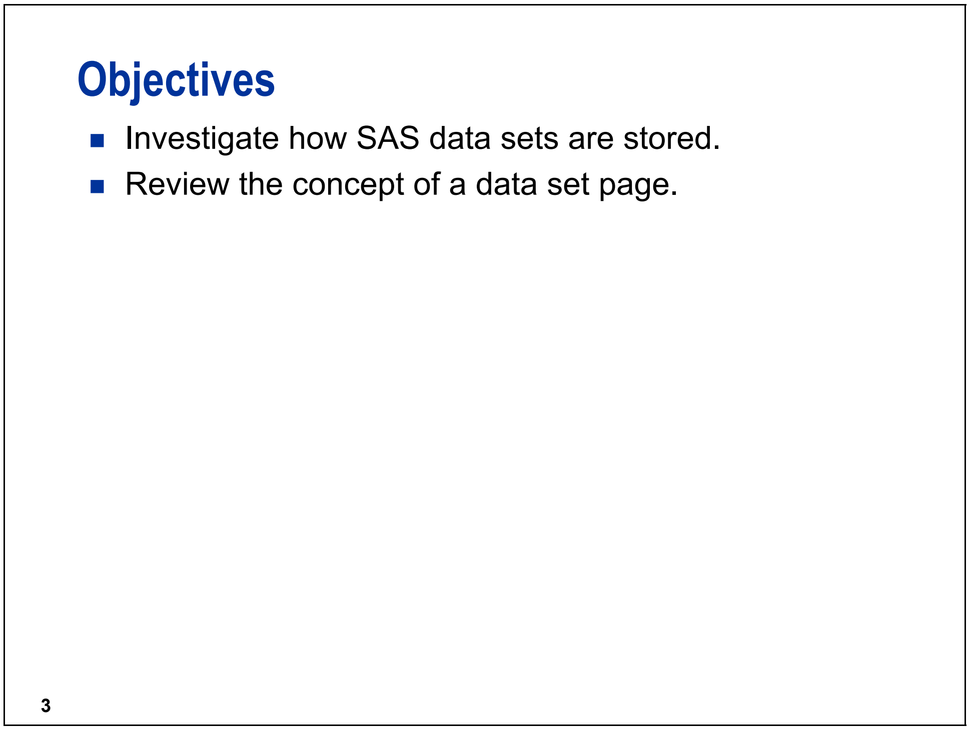
Partial List of the Raw Data File **operate**

```
28DEC1986,KRISCHOCK,JENNIFER ANNE,AUSTRALIA,SYDNEY,E02912,BAGCLK
11JUN1991,LAHANE,SEAN,AUSTRALIA,SYDNEY,E00253,BAGCLK
19AUG1986,LAI,CRAIG NEIL,AUSTRALIA,SYDNEY,E02197,BAGCLK
19MAY1993,LAWS,MERIAN,AUSTRALIA,SYDNEY,E02314,BAGCLK
20JUL1980,LINDSAY,ROBERT,AUSTRALIA,SYDNEY,E03113,BAGCLK
23APR1987,LONG,CARMEN,AUSTRALIA,SYDNEY,E03179,BAGCLK
02AUG1989,LOWRIE,KERRIE,AUSTRALIA,SYDNEY,E03421,BAGCLK
```

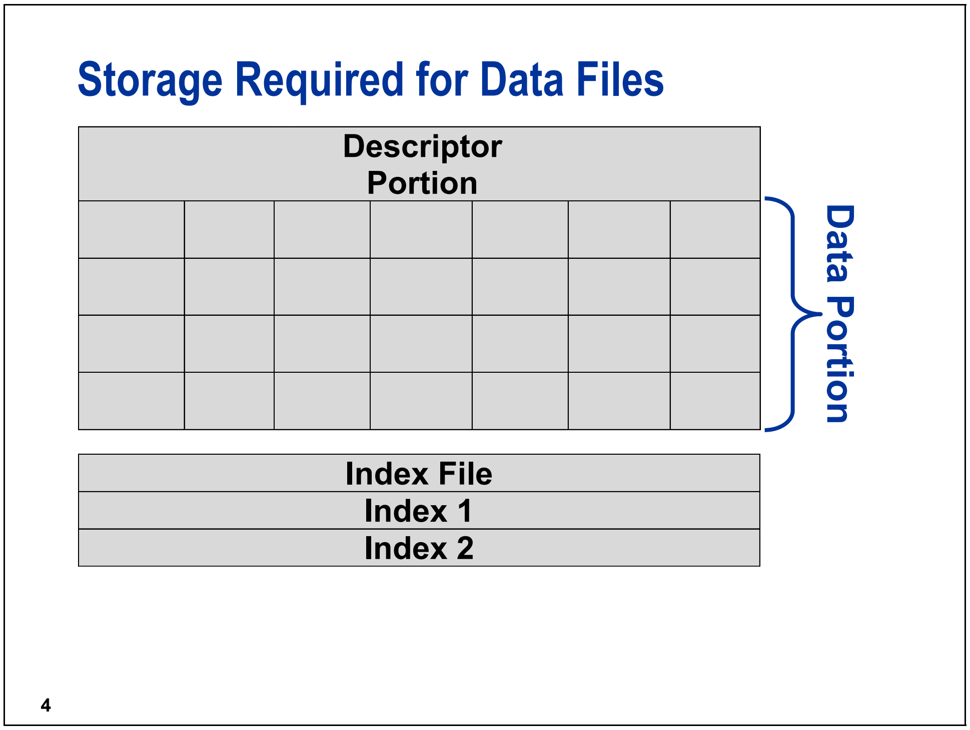
```
data oper (sortedby = JobCode);
   infile 'operate.dat' dsd;          * PC/UNIX;
   *infile '.prog3.rawdata(operate)';  * z/OS;
   input HireDate : date9. LastName : $32.
         FirstName : $32. EmpCountry : $25.
         EmpLocation : $16.
         EmpID $ JobCode $;
run;
proc print data = oper;
   by JobCode;
run;
```

Chapter 7 Controlling Data Storage Space

7.1 Introduction



This chapter addresses Base SAS data sets only. Scalable Performance Data Engine data is addressed in a later chapter.



The total amount of storage required for a SAS data file is the sum of the space required for the following:

- the descriptor portion
- the observation length multiplied by the number of observations
- any associated indexes
- any operating-system-specific storage overhead

Review of the Data Set Page

A data set page

- is the unit of data transfer between the SAS storage device and main memory
- includes the bytes used by the descriptor portion, the data values, and any overhead
- is fixed in size when the data set is created.

5

The total number of bytes occupied by a data set equals the data page size times the number of pages plus the index page size times the number of pages.

Determining Page Size with PROC CONTENTS

```
proc contents data = ia.sales;
run;
```

Partial Output

```
 Engine/Host Dependent Information

Data Set Page Size          16384
Number of Data Set Pages    3396
First Data Page             1
Max Obs per Page            97
Obs in First Data Page      78
Index File Page Size        4096
Number of Index File Pages  2552
Number of Data Set Repairs  0
File Name                   C:\workshop\w
Release Created             9.0101M3
Host Created                XP_PRO
```

ia.sales contains 55,640,064 bytes of data in the data portion and 10,452,992 bytes for the index file. The total number of bytes is 66,093,056.

c07s1d1

6

The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

7.2 Reducing the Length of Numeric Variables

Objectives

- Describe how SAS stores numeric values.
- Determine how to safely reduce the space required to store numeric values in SAS data sets.

8

Characteristics of Numeric Variables

Numeric variables

- store multiple digits per byte
- take eight bytes of storage per variable, by default
- can be reduced in size
- always have a length of eight bytes in the PDV
- are stored as floating-point numbers in real-binary representation
- use a minimum of one byte to store the sign and exponent of the value (depending on the operating environment) and use the remaining bytes to store the mantissa of the value.

9

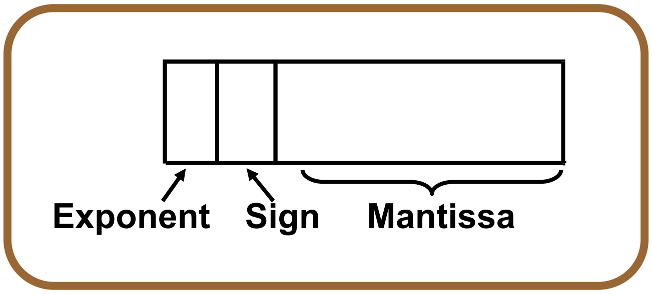
Default Length of Numeric Variables

The number 35298 can also be written as follows:

```
+0.35298*(10**5)
```

Sign Mantissa Base Exponent

SAS stores numeric variables in floating point form:



10

SAS stores numeric values in native floating point representation. On UNIX, Linux, Windows, and Open VMS/Alpha platforms, this form is "IEEE format" as defined in ISO standard IEC 60559. On z/OS, SAS stores numeric values in IBM mainframe floating-point representation.

Summary of Floating-Point Numbers Stored in Eight Bytes

Representation	Base	Exponent Bits	Maximum Mantissa Bits
IBM mainframe	16	7	56
IEEE	2	11	52

Assigning the Length of Numeric Variables

- You can use a LENGTH statement to assign a length from two to eight bytes to numeric variables.
- The minimum length of numeric variables depends on the operating environment.

Example:

```
data reducedsales;
   length Cap1st CapBus CapEcon 3
          CapCargo Num1st NumBus
          NumEcon CargoWeight FltDate 4
          Rev1st RevBus
          RevEcon CargoRev 5;

   <more SAS code>
run;
```

11 c07s2d1

To decrease the length of all numeric variables, you can use the DEFAULT= option in the LENGTH statement:

```
data reducedsales;
   length default = 4;
  ... more SAS code ...
run;
```

Assigning the Length of Numeric Variables

Size of **ia.sales** (without index)	Size of **reducedsales**	% Difference
55,640,064 bytes	37,134,336 bytes	33%

12

Comparing Data Sets

```
proc compare data = ia.sales
             compare = reducedsales;
run;
```

Partial Output

```
                    Observation Summary

             Observation      Base  Compare

             First Obs           1        1
             Last  Obs      329264   329264

Number of Observations in Common: 329264.
Total Number of Observations Read from ia.sales: 329264.
Total Number of Observations Read from work.reducedsales: 329264.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 329264.

NOTE: No unequal values were found. All values compared are exactly equal.
```

13 c07s2d2

Possible Storage Lengths for Integer Values

Windows and UNIX

Length (bytes)	Largest Integer Represented Exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,992

14

Possible Storage Lengths for Integer Values

z/OS

Length (bytes)	Largest Integer Represented Exactly
2	256
3	65,536
4	16,777,216
5	4,294,967,296
6	1,099,511,627,776
7	281,474,946,710,656
8	72,057,594,037,927,936

15

Exceeding the number of integer digits recommended above or reducing the stored size of non-integer data can result in a loss of precision due to the truncation of nonzero bytes. It is **not** recommended.

Assigning the Length of Numeric Variables

The use of a numeric length less than 8 bytes

- reduces the number of bytes available for the mantissa, and thus reduces the precision of the largest number that can be accurately stored
- does not affect how numbers are stored in the PDV; numbers are always eight bytes in length in the PDV
- causes the number to be truncated to the specified length when the value is written to the SAS data set
- causes the number to be expanded to eight bytes in the PDV when the data set is read by padding the mantissa with binary zeros.

16

Reading Reduced-Length Numeric Variables

Reading reduced-length numeric variables

- requires less I/O
- uses additional CPU
- can be dangerous for high precision values, including non-integer and large integer values.

17

Dangers of Reduced-Length Numeric Variables

It is **not** recommended that you change the length of non-integer numeric variables.

```
data test;
   length x 4;
   x = 1/10;
   y = 1/10;
run;

data _null_;
   set test;
   put x=;
   put y=;
run;
```

18 c07s2d3

Dangers of Reduced-Length Numeric Variables

Partial Log

```
81   data test;
82      length x 4;
83      x = 1/10;
84      y = 1/10;
85   run;
NOTE: The data set WORK.TEST has 1 observations and 2 variables.

86
87   data _null_;
88      set test;
89      put x=;
90      put y=;
91   run;

x=0.0999999642
y=0.1
NOTE: There were 1 observations read from the data set WORK.TEST.
```

19

Just as a decimal number system cannot store the fraction 1/3 exactly in a finite number of digits, a binary number system (or multiple thereof, such as octal or hexadecimal) cannot store the fraction 1/10 exactly in any finite number of digits.

Dangers of Reduced-Length Numeric Variables

It is **not** recommended that you change the length of integer numeric variables inappropriately or that you change the length of large integer numeric variables.

```
data test;
   length x 3;
   x = 8193;
run;

data _null_;
   set test;
   put x=;
run;
```

20 c07s2d4

This example illustrates the dangers of inappropriately reducing integer values.

Dangers of Reduced-Length Numeric Variables

Partial Log

```
192
193  data _null_;
194     set test;
195     put x=;
196  run;

x=8192
NOTE: There were 1 observations read from the
data set WORK.TEST.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

21

7.3 Compressing Data Files

Objectives

- Define the structure of a compressed SAS data file.
- Create a compressed SAS data file.
- Examine the advantages and disadvantages of compression.

23

SAS data files, but not views, can be stored in compressed form.

Uncompressed SAS Data File Structure

Features of uncompressed SAS data files:

- Each variable occupies the same number of bytes in every observation.
- Each observation occupies the same number of bytes as every other observation.
- Character values are padded with blanks.
- Numeric values are padded with binary zeros.
- The descriptor portion of the data set is stored at the end of the first data set page.

continued...

24

Uncompressed SAS Data File Structure

- There is a 16-byte overhead at the beginning of each page.
- There is a 1-bit per observation overhead rounded up to the nearest byte.
- New observations are added at the end of the file. If a new page is needed for a new observation, a whole data set page is added.
- Deleted observation space is never reused, unless the entire data file is rebuilt.

25

In uncompressed SAS data files, each observation is a fixed-length record.

Compressed SAS Data File Structure

Features of compressed SAS data files:

- Each observation is a single string of bytes. Variable types and boundaries are ignored.
- Each observation can have a different length.
- Consecutive repeating characters and numbers are collapsed into fewer bytes.
- If an updated observation is larger than its original size, it is stored on either the same data set page or a different page with a pointer to the original page.
- The descriptor portion of the data set is stored at the end of the first data set page.

continued...

26

Compressed SAS Data File Structure

- There is a 28-byte overhead at the beginning of each page.
- There is a 12-byte-per-observation overhead on 32-bit systems.
- There is a 24-byte-per-observation overhead on 64-bit systems.
- Deleted observation space can be reused if the REUSE=YES data set or system option was turned on when the SAS data file was compressed.

27

Compressing a file reduces the number of bytes required to represent each observation. In a compressed file, each observation is a variable-length record.

Compressing SAS Files

There are two different algorithms that can be used to compress files:

- the RLE (Run Length Encoding) compression algorithm (compress = YES | CHAR)
- the RDC (Ross Data Compression) algorithm (COMPRESS = BINARY)

The optimal algorithm depends on the characteristics of your data.

28

Creating an Uncompressed Data File

```
data sales;
   infile 'Sales.dat';
   input @1 FlightID $7.      @8 RouteID $7.
         @15 Origin $3.       @18 Dest $3.
         @21 DestType $13.    @34 FltDate date9.
         @43 Cap1st 3.        @46 CapBus 3.
         @49 CapEcon 3.       @52 CapPassTotal 3.
         @55 CapCargo 6.      @61 Num1st 3.
         @64 NumBus 3.        @67 NumEcon 3.
         @70 NumPassTotal 3.  @73 Rev1st 7.
         @80 RevBus 7.        @87 RevEcon 7.
         @94 CargoRev 8.      @102 RevTotal 10.
         @112 CargoWeight 5.;
run;
```

29 c07s3d1

Creating a Compressed Data File

```
data saleschar(compress = char);
   infile 'Sales.dat';
   input @1 FlightID $7.      @8 RouteID $7.
         @15 Origin $3.       @18 Dest $3.
         @21 DestType $13.    @34 FltDate date9.
         @43 Cap1st 3.        @46 CapBus 3.
         @49 CapEcon 3.       @52 CapPassTotal 3.
         @55 CapCargo 6.      @61 Num1st 3.
         @64 NumBus 3.        @67 NumEcon 3.
         @70 NumPassTotal 3.  @73 Rev1st 7.
         @80 RevBus 7.        @87 RevEcon 7.
         @94 CargoRev 8.      @102 RevTotal 10.
         @112 CargoWeight 5.;
run;
```

30 c07s3d2

The external file **sales** used for demonstrations and exercises contains fewer records than the external file **sales** used for the course notes.

Partial Log

```
NOTE: The data set WORK.SALESCHAR has 329264 observations and 21
variables.
NOTE: Compressing data set WORK.SALESCHAR decreased size by 28.14
percent.
      Compressed is 4930 pages; un-compressed would require 6861 pages.
NOTE: DATA statement used (Total process time):
      real time           17.36 seconds
      cpu time            3.25 seconds
```

31

Creating a Compressed Data File

```
data salesbin(compress = binary);
   infile 'Sales.dat';
   input @1 FlightID $7.      @8 RouteID $7.
         @15 Origin $3.       @18 Dest $3.
         @21 DestType $13.    @34 FltDate date9.
         @43 Cap1st 3.        @46 CapBus 3.
         @49 CapEcon 3.       @52 CapPassTotal 3.
         @55 CapCargo 6.      @61 Num1st 3.
         @64 NumBus 3.        @67 NumEcon 3.
         @70 NumPassTotal 3.  @73 Rev1st 7.
         @80 RevBus 7.        @87 RevEcon 7.
         @94 CargoRev 8.      @102 RevTotal 10.
         @112 CargoWeight 5.;
run;
```

32 c07s3d3

Partial Log

```
NOTE: The data set WORK.SALESBIN has 329264 observations and 21
variables.
NOTE: Compressing data set WORK.SALESBIN decreased size by 31.51
percent.
      Compressed is 4699 pages; un-compressed would require 6861 pages.
NOTE: DATA statement used (Total process time):
      real time           7.04 seconds
      cpu time            3.62 seconds
```

33

Summary of Compression Results

Data Set	Algorithm Used	Number of Bytes	Decreased size
sales	none	55,623,680	
saleschar	CHAR	40,386,560	28.14%
salesbin	BINARY	38,494,208	31.51%

34

Creating a Compressed Data File

To create a compressed data file, use the COMPRESS= output data set option or system option.

General forms of the COMPRESS= options:

SAS-data-set(COMPRESS = NO | YES | CHAR | BINARY)

OPTIONS COMPRESS = NO | YES | CHAR | BINARY;

35

COMPRESS = Values	Action
NO	does not compress the data file (default).
CHAR | YES	uses the Run Length Encoding (RLE) compression algorithm, which compresses repeating consecutive bytes, such as trailing blanks or repeated zeros.
BINARY	uses Ross Data Compression (RDC), which combines run length encoding and sliding window compression.

Not all engines support compression.

The COMPRESS= data set option overrides the COMPRESS= system option.

The COMPRESS= options interact with two other system or data set options, POINTOBS= and REUSE=. See "COMPRESS= Data Set Option" in the dictionary of SAS language elements in *SAS Language Reference: Dictionary* in the Base SAS documentation for additional information on these interactions.

Comparing Compression Methods

COMPRESS = YES | CHAR

- is effective with character data that contains repeated characters (such as blanks)

COMPRESS = BINARY

- takes significantly more CPU time to uncompress than COMPRESS=YES | CHAR
- is more efficient with observations greater than a thousand bytes in length
- can be very effective with numeric data
- can be effective with character data that contains patterns, rather than simple repetitions

36

How SAS Compresses Data

A data file has these variables:

Name	Type	Length
`LastName`	Character	20
`FirstName`	Character	15

In uncompressed form, all observations use 35 bytes for these two variables.

LastName FirstName

01 ... 20 ...

A	D	A	M	S								B	I	L	L						

37

COMPRESS = CHAR | YES

In run length encoding compressed form, the **LastName** and **FirstName** values for this observation use only 13 bytes.

LastName							FirstName					
01							08					13
@	A	D	A	M	S	#	@	B	I	L	L	#

38

COMPRESS = BINARY

Ross Data Compression uses both run-length encoding and sliding window compression.

A data set has these variables:

Name	Type	Length
Answer1	Numeric	8
. . .		
Answer200	Numeric	8

In uncompressed form, the data file resembles this:

Obs	answer1	answer2	answer3	answer4	answer5		answer200
1	1	2	1	2	1	. . .	2
2	1	1	1	1	1	. . .	1
3	2	2	2	2	2	. . .	2

39 ...

COMPRESS = BINARY

In Ross data compressed form, the first observation in the data file resembles the form below:

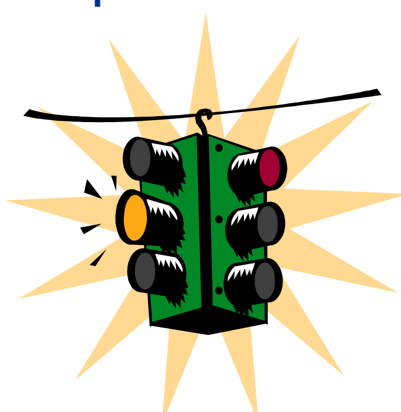
01								09
@	+ 1	1	#	@	+ 1	2	#	%

- The @ indicates how many uncompressed characters follow.
- The # indicates the number of binary zeros repeated at this point in the observation.
- The % indicates how many times these values are repeated.

40

+ 1	Indicates the sign and exponent.

Compression Guidelines



Some data sets do not compress well or at all.

41

Compression Dependencies

Because there is higher overhead for each observation, a data file can occupy more space in compressed form than in uncompressed form if the file has the following:

- few repeated characters
- small physical size
- few missing values
- short text strings

42

Compression Guidelines

```
data capacity(compress = yes);
   set ia.capacity;
run;
```

Partial Log

```
1175  data capacity(compress = yes);
1176     set ia.capacity;
1177  run;

NOTE: There were 108 observations read from the data set IA.CAPACITY.
NOTE: The data set WORK.CAPACITY has 108 observations and 7 variables.
NOTE: Compressing data set WORK.CAPACITY increased size by 50.00 percent.
      Compressed is 3 pages; un-compressed would require 2 pages.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

43 c07s3d4

Compression Dependencies

When you use the COMPRESS= data set option or the COMPRESS= system option, SAS knows the following:

- size of the overhead introduced by compression
- maximum size of an observation

If the maximum size of the observation is not larger than the overhead introduced by compression, SAS disables compression, creates an uncompressed data set, and issues a note stating that the file was not compressed.

44

This feature is available in SAS Release 8.2 and later.

Compression Dependencies

```
data test(compress = yes);
   x = 1;
run;

NOTE: Compression was disabled for data set
WORK.TEST because compression overhead would increase
the size of the data set.
NOTE: The data set WORK.TEST has 1 observations and
1 variables.
NOTE: DATA statement used:
      real time           0.51 seconds
      cpu time            0.10 seconds
```

45 c07s3d5

Compression Trade-Offs

Uncompressed	Compressed
Usually requires more disk storage.	Usually requires less disk storage.
Requires less CPU time to prepare observation for I/O.	Requires more CPU time to prepare observation for I/O.
Uses more I/O operations.	Uses fewer I/O operations.

The savings in I/O operations greatly outweighs the increase in CPU time.

continued...

46

Compression Trade-Offs

Uncompressed	Compressed
An updated observation fits in its original location.	An updated observation might be moved from its original location.
Deleted observation space is never reused.	Deleted observation space can be reused.
New observations are always inserted at the end of the data file.	When REUSE=YES, new observations might not be inserted at the end of the data file.

47

Exercises

1. Creating Reduced-Length Numeric Variables and Compressed SAS Data Files

Use the program, c07ex1start, as a starting program for the following:

a. Submit the program and record the number of pages and the page size for the data set **sales**.

b. Edit the program to decrease the length of the numeric variables **Cap1st**, **CapBus**, and **CapEcon** to 3; **CapCargo**, **Num1st**, **NumBus**, **NumEcon**, **NumPassTotal**, **CapPassTotal**, **CargoWeight** and **FltDate** to 4; and **Rev1st**, **RevBus**, **RevEcon**, **RevCargo** and **RevTotal** to 5.

Change the name of the output data set to **salesnum**. Resubmit it, and record the number of pages and the page size for the data set **salesnum**.

c. Edit the original c07ex1start program to create a compressed data set using COMPRESS=CHAR. Change the name of the output data set to **saleschar**. Be sure not to use the reduced length numeric program to create **saleschar**. Submit the program, and record the number of pages and the page size for the data set **saleschar**.

d. Edit the program to create a compressed data set using COMPRESS=BINARY. Change the name of the output data set to **salesbin**. Resubmit it, and record the number of pages and the page size for the data set **salesbin**.

2. Comparing CPU Time

Submit the program, c07ex2start, and compare the user CPU time for reading **sales**, **salesnum**, **saleschar**, and **salesbin**.

The external file s**ales** used for demos and exercises contains fewer records than the external file **sales** used for the course notes.

7.4 Creating a DATA Step View

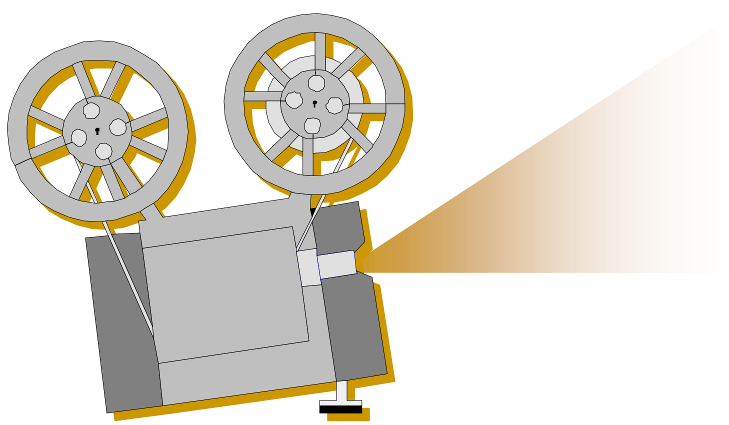
Objectives

- Investigate types of SAS data sets.
- Create and use DATA step views.
- Determine the advantages of DATA step views.
- Examine guidelines for using DATA step views.

50

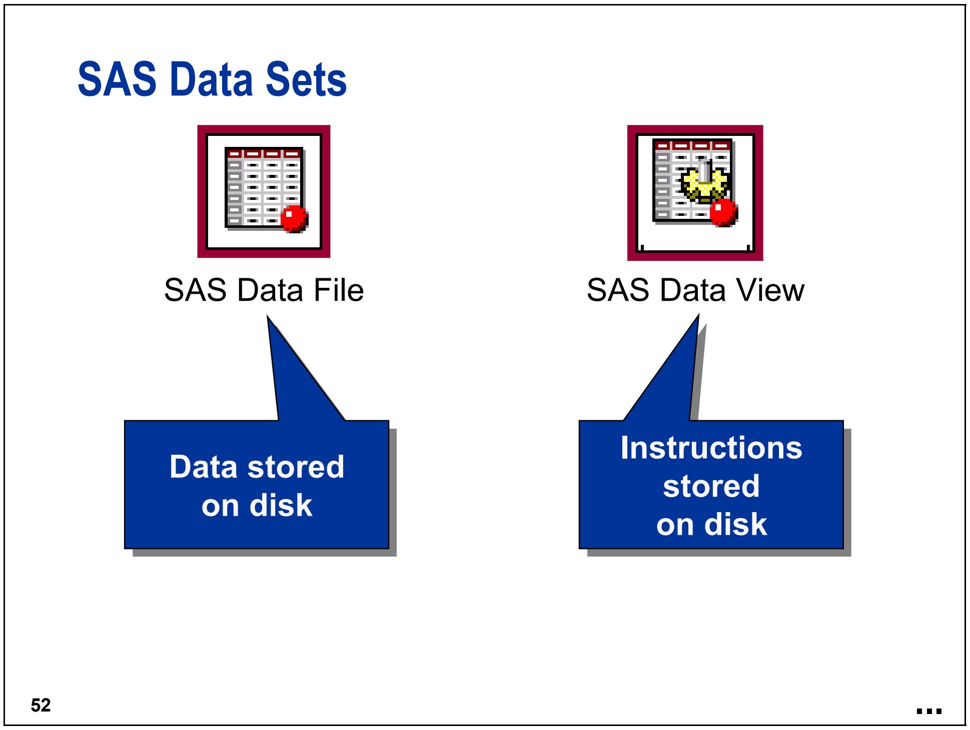
Creating a DATA Step View

Instead of creating a SAS data file that contains three months of raw data, as discussed in a previous chapter, you can create a DATA step view.

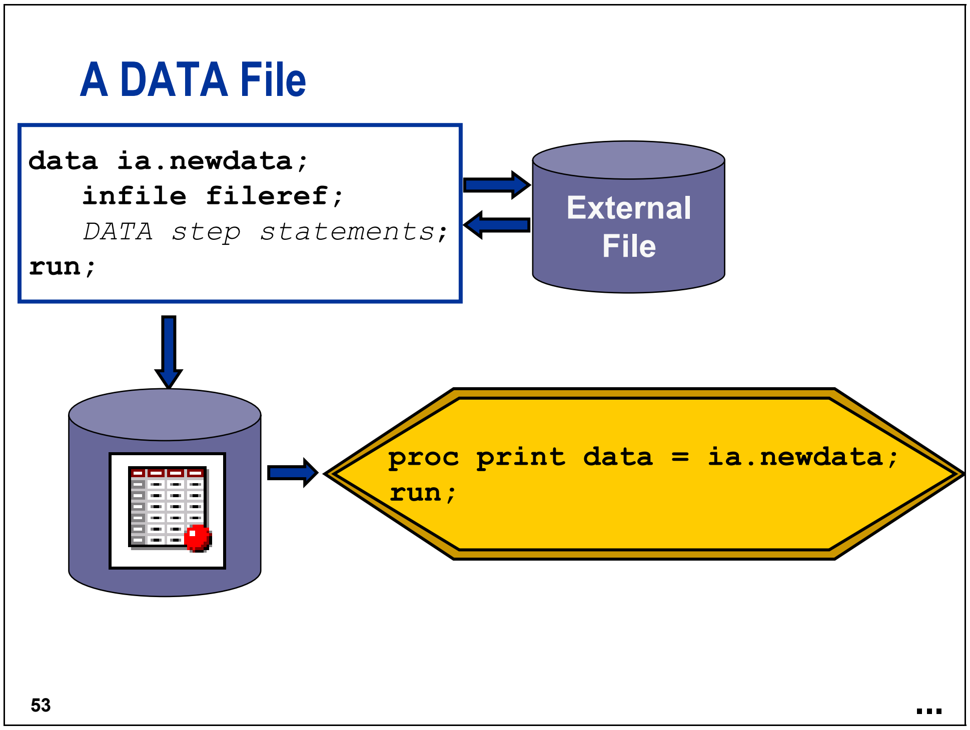


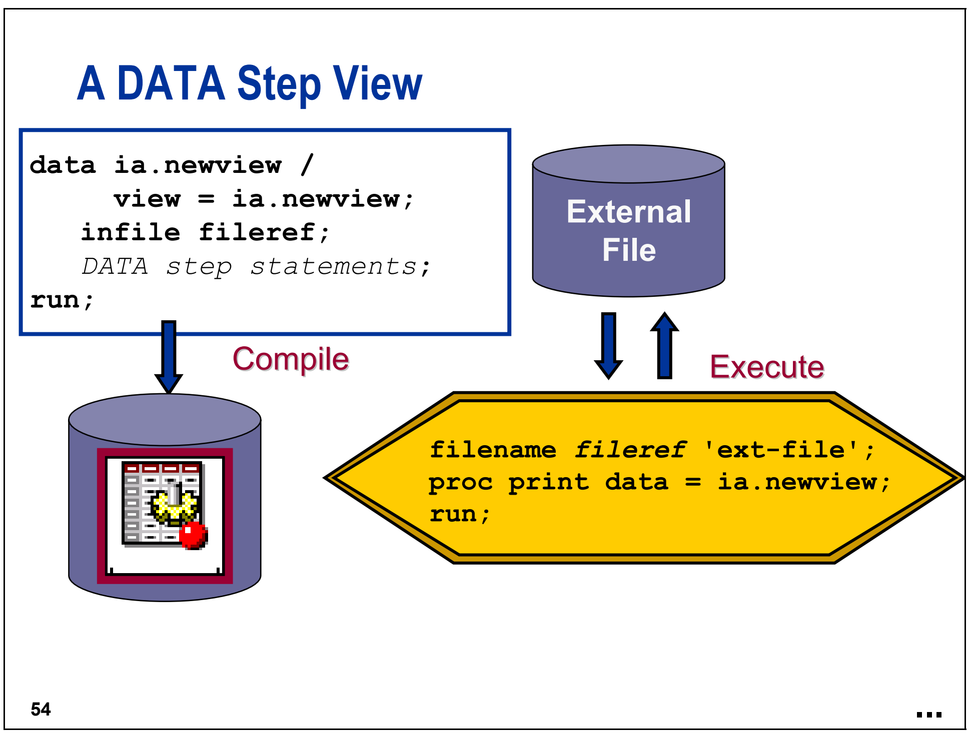
51

The FILENAME statement and the FILEVAR option for the INFILE statement were discussed in an earlier chapter.



A DATA file…	A DATA step view…
is a SAS file with a member type of DATA.	is a SAS file with a member type of VIEW.
enables read or write capabilities.	is read-only.
contains data and a descriptor portion that are stored on disk.	contains no data.
	contains a partially compiled DATA step.





The name of a DATA view must be different from the name of an existing DATA file in the same SAS library.

Creating a DATA Step View

c07s4d1

```
data ia.firstq / view = ia.firstq;
   infile Q1;
   input Flight $ Origin $ Dest $ Date : date9.
         RevCargo : comma15.2;
run;
```

Log

```
  data ia.firstq / view=ia.firstq;
     infile Q1;
     input Flight $ Origin $ Dest $ Date : date9.
           RevCargo : comma15.2;
  run;

NOTE: DATA STEP view saved on file IA.FIRSTQ.
NOTE: A stored DATA STEP view cannot run under a different operating system.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.01 seconds
```

```
filename Q1 ('month1.dat' 'month2.dat' 'month3.dat');

proc print data = ia.firstq;
   title 'ia.firstq DATA Step View';
   format Date date9.;
run;
```

Partial Output

ia.firstq DATA Step View

Obs	Flight	Origin	Dest	Date	Rev Cargo
1	IA10200	SYD	HKG	01JAN2000	191187
2	IA10201	SYD	HKG	01JAN2000	169653
3	IA10300	SYD	CBR	01JAN2000	850
4	IA10301	SYD	CBR	01JAN2000	970
5	IA10302	SYD	CBR	01JAN2000	1030
6	IA10303	SYD	CBR	01JAN2000	1410
7	IA10304	SYD	CBR	01JAN2000	870
8	IA10305	SYD	CBR	01JAN2000	730
9	IA10400	CBR	SYD	01JAN2000	1390
10	IA10401	CBR	SYD	01JAN2000	750

Log

```
  filename Q1 ('month1.dat' 'month2.dat' 'month3.dat');

  proc print data=ia.firstq;
     title 'ia.firstq DATA Step View';
     format Date date9.;
  run;

NOTE: The infile Q1 is:
      File Name=c:\workshop\winsas\prog3\month1.dat,

      File List=('c:\workshop\winsas\prog3\month1.dat'
      'c:\workshop\winsas\prog3\month2.dat'
      'c:\workshop\winsas\prog3\month3.dat'),
      RECFM=V,LRECL=256

NOTE: The infile Q1 is:
      File Name=c:\workshop\winsas\prog3\month2.dat,

      File List=('c:\workshop\winsas\prog3\month1.dat'
      'c:\workshop\winsas\prog3\month2.dat'
      'c:\workshop\winsas\prog3\month3.dat'),
      RECFM=V,LRECL=256

NOTE: The infile Q1 is:
      File Name=c:\workshop\winsas\prog3\month3.dat,

      File List=('c:\workshop\winsas\prog3\month1.dat'
      'c:\workshop\winsas\prog3\month2.dat'
      'c:\workshop\winsas\prog3\month3.dat'),
      RECFM=V,LRECL=256

NOTE: 2299 records were read from the infile Q1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2090 records were read from the infile Q1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2297 records were read from the infile Q1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: View IA.FIRSTQ.VIEW used:
      real time           0.15 seconds
      cpu time            0.16 seconds

NOTE: There were 6686 observations read from the data set IA.FIRSTQ.
NOTE: PROCEDURE PRINT used:
      real time           0.15 seconds
      cpu time            0.16 seconds
```

c07s4d2

```
/* The following program appends data from 3 months.
   The data selected is dependent on today's date. */

data ia.movingq / view = ia.movingq;
   drop MonNum MidMon LastMon I today;
   today = today();
   MonNum = month(today);
   MidMon = month(intnx('month',today,-1));
   LastMon = month(intnx('month',today,-2));
   do I = MonNum, MidMon, LastMon;
      NextFile = "month"||put(i,2.)||".dat";
      NextFile = compress(NextFile,' ');
      do until (LastObs);
         infile in filevar = NextFile end = LastObs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

Log

```
/* The following program appends data from 3 months.
    The data selected is dependent on today's date. */

 data ia.movingq / view=ia.movingq;
    drop MonNum MidMon LastMon I today;
    today = today();
    MonNum = month(today);
    MidMon = month(intnx('month',today,-1));
    LastMon = month(intnx('month',today,-2));
    do I = MonNum, MidMon, LastMon;
       NextFile = "month"||put(i,2.)||".dat";
       NextFile = compress(NextFile,' ');
       do until (LastObs);
          infile in filevar = NextFile end = LastObs;
          input Flight $ Origin $ Dest $ Date : date9.
                RevCargo : comma15.2;
          output;
       end;
    end;
    stop;
 run;

NOTE: DATA STEP view saved on file IA.MOVINGQ.
NOTE: A stored DATA STEP view cannot run under a different operating system.
NOTE: DATA statement used:
      real time           0.07 seconds
      cpu time            0.00 seconds
```

```
data view = ia.movingq;
   describe;
run;
```

Log

```
  data view = ia.movingq;
     describe;
  run;

NOTE: DATA step view IA.MOVINGQ is defined as:

data ia.movingq / view = ia.movingq;
   drop MonNum MidMon LastMon I today;
   today = today();
   MonNum = month(today);
   MidMon = month(intnx('month',today,-1));
   LastMon = month(intnx('month',today,-2));
   do I = MonNum, MidMon, LastMon;
      NextFile = "month"||put(i,2.)||".dat";
      NextFile = compress(NextFile,' ');
      do until (LastObs);
         infile in filevar = NextFile end = LastObs;
         input Flight $ Origin $ Dest $ Date : date9. RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

```
options date;

proc print data = ia.movingq;
   title 'ia.movingq DATA Step View';
   var Flight Origin Date Dest RevCargo;
   format Date date9.;
run;
```

Partial Output

```
                                    ia.movingq DATA Step View
                                                           12:41 Wednesday, February 4, 2004

                                                                     Rev
               Obs     Flight     Origin          Date     Dest     Cargo

                 1     IA10200     SYD       01FEB2000     HKG     177801
                 2     IA10201     SYD       01FEB2000     HKG     174891
                 3     IA10300     SYD       01FEB2000     CBR       1070
                 4     IA10301     SYD       01FEB2000     CBR       1310
                 5     IA10302     SYD       01FEB2000     CBR        850
                 6     IA10303     SYD       01FEB2000     CBR       1030
                 7     IA10304     SYD       01FEB2000     CBR        910
                 8     IA10305     SYD       01FEB2000     CBR       1270
                 9     IA10400     CBR       01FEB2000     SYD       1310
                10     IA10401     CBR       01FEB2000     SYD       1110
```

Log

```
 options date;

 proc print data = ia.movingq;
    title 'ia.movingq DATA Step View';
    var Flight Origin Date Dest RevCargo;
    format Date date9.;
 run;

NOTE: The infile IN is:
      File Name=c:\workshop\winsas\prog3\month2.dat,
      RECFM=V,LRECL=256

NOTE: The infile IN is:
      File Name=c:\workshop\winsas\prog3\month1.dat,
      RECFM=V,LRECL=256

NOTE: The infile IN is:
      File Name=c:\workshop\winsas\prog3\month12.dat,
      RECFM=V,LRECL=256

NOTE: 2090 records were read from the infile IN.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2299 records were read from the infile IN.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 2190 records were read from the infile IN.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: View IA.MOVINGQ.VIEW used:
      real time           0.83 seconds
      cpu time            0.23 seconds

NOTE: There were 6579 observations read from the data set IA.MOVINGQ.
NOTE: PROCEDURE PRINT used:
      real time           0.83 seconds
      cpu time            0.23 seconds
```

DATA Statement with VIEW= Option Syntax

General form of the DATA statement with VIEW= option:

DATA *data-set-name(s)* / VIEW = *view-name*;
    INFILE *fileref*;
    INPUT *variable(s);*
RUN;

VIEW = *view-name*

view-name specifies a name that the DATA step uses to store the partially compiled DATA step. The *view-name* must match one of the data set names.

56

You can also create SAS data files in the DATA step that creates the view; but you can only create one view per DATA step.

The DESCRIBE Statement

You can use the DESCRIBE statement to retrieve program source code from a DATA step view. SAS writes the source statements to the SAS log.

General form of the DESCRIBE statement:

DATA VIEW = *view-name*;
    DESCRIBE;
RUN;

57

Advantages of DATA Step Views

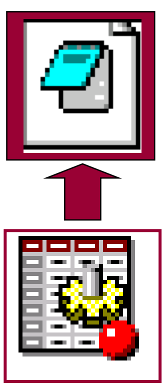
You can use DATA step views to do the following:

- combine data from multiple sources
- hide complex code from users
- access the most current data in changing files
- avoid storing a SAS copy of a large data file
- avoid creating intermediate copies of data

58

Guidelines for Creating and Using Views

If data is used many times in one program, it is more efficient to create and reference a SAS data file than to create and reference a view.



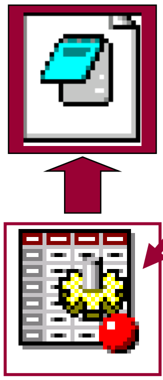
```
proc print data = ia.sview;
run;

proc freq data = ia.sview;
   tables JobCode;
run;

proc means data = ia.sview;
run;
```

59

...

Guidelines for Creating and Using Views

If data is used many times in one program, it is more efficient to create and reference a SAS data file than to create and reference a view.

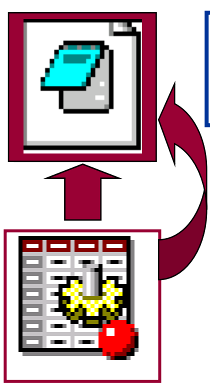


```
data staff;
   set ia.sview;
run;

proc print data = staff;
run;

proc freq data = staff;
   tables JobCode;
run;

proc means data = staff;
run;
```

63 ...

Guidelines for Creating and Using Views

Expect a degradation in performance when you use a SAS data view with a procedure that requires multiple passes through the data.



```
proc print data = ia.sview uniform;
run;
```

64 ...

The PRINT procedure with the UNIFORM option, the CLASS statement in the MEANS/SUMMARY, TABULATE, and UNIVARIATE procedures, and many SAS/STAT procedures require multiple passes through the data.

Guidelines for Creating and Using Views

Avoid creating views on files whose structures often change.

file1

```
FLTATEN1 23456
```

file2

```
23456 FLTATEN1
```

file3

```
LEVELI FLTATEN1
```

```
filename rawdata 'file1';
proc print data = ia.sview;
run;
filename rawdata 'file2'
proc freq data = ia.sview;
   tables JobCode;
run;
filename rawdata 'file3'
proc means data = ia.sview;
run;
```

65 ...

Reference Information

Creating a VIEW and a FILE

Only one view can be created in a DATA step.

In addition to the view name, you can specify other data set names in the DATA statement. The data sets are not created until the view is processed.

c07ref1

```
data ia.movingq work.movingq / view = ia.movingq;
   drop MonNum MidMon LastMon I today;
   today = today();
   MonNum = month(today);
   MidMon = month(intnx('month',today,-1));
   LastMon = month(intnx('month',today,-2));
   do I = MonNum, MidMon, LastMon;
      NextFile = "month"||put(i,2.)||".dat"; * Windows/UNIX;
        *Nextfile = ".prog3.rawdata(month"!!put(i,2.)!!")"; /* z/OS */
      NextFile = compress(NextFile,' ');
      do until (LastObs);
         infile in filevar = NextFile end = LastObs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;

proc print data = ia.movingq;
   title 'ia.movingq DATA Step View';
   title2 'triggers creation of work.movingq data set';
   var Flight Origin Date Dest RevCargo;
   format Date date9.;
run;
```

Partial Log

```
  proc print data = ia.movingq;
     title 'ia.movingq DATA Step View';
     title2 'triggers creation of work.movingq data set';
     var Flight Origin Date Dest RevCargo;
     format Date date9.;
  run;

NOTE: The data set WORK.MOVINGQ has 6684 observations and 5 variables.
NOTE: There were 6684 observations read from the data set IA.MOVINGQ.
NOTE: PROCEDURE PRINT used:
      real time           0.30 seconds
      cpu time            0.25 seconds
```

Using Macro Variables

Because SAS macro variables are resolved during compilation, any macro variables used in a DATA step view are resolved when the view is created.

You can use the SYMGET function to postpone macro resolution until the view is executed.

c07ref2

```
data ia.movingq / view = ia.movingq;
   drop MonNum MidMon LastMon I today;
   today = today();
   MonNum = month(today);
   MidMon = month(intnx('month',today,-1));
   LastMon = month(intnx('month',today,-2));
   do I = MonNum, MidMon, LastMon;
      NextFile = "month"!!put(i,2.)!!".dat";* Windows/UNIX;
   *Nextfile = ".prog3.rawdata(month"!!put(i,2.)!!")"; /* z/OS */
      NextFile = compress(NextFile,' ');
         do until (LastObs);
            infile in filevar = NextFile end = LastObs;
            input Flight $ Origin $ Dest $ Date : date9.
                  RevCargo : comma15.2;
            if Dest = symget('ThisDest') then output;
         end;
   end;
   stop;
run;
```

Use the %LET statement to provide a value for the macro variable **ThisDest**.

```
%let ThisDest = MCI;
proc print data = ia.movingq;
   title "Flight to &ThisDest";
   var Flight Origin Date Dest RevCargo;
   format Date date9.;
run;
```

Partial Output

```
                              Flights to MCI

                                                             Rev
              Obs    Flight     Origin         Date    Dest    Cargo

                1    IA03904     RDU      01JAN2000    MCI      4161
                2    IA03904     RDU      04JAN2000    MCI      7125
                3    IA03903     RDU      05JAN2000    MCI      7239
                4    IA03900     RDU      16JAN2000    MCI      4275
                5    IA03903     RDU      18JAN2000    MCI      7581
                6    IA03900     RDU      20JAN2000    MCI      5073
                7    IA03904     RDU      20JAN2000    MCI      5871
```

Exercises

3. Creating a DATA Step View

Use the program, c07ex3start as a starting program. Write one DATA step to create both a view and a file.

HINT: Investigate the **Reference Information** on **Creating a VIEW and a FILE**.

a. Name the DATA step view **`laircraft`**. The view should contain the aircraft where the **`CapTotal`** value is over 200.

b. Name the data file **`saircraft`**. The file should contain the aircraft where the **`CapTotal`** value is less than or equal to 200.

4. Printing the DATA Step File Unsuccessfully

Attempt to print the **`saircraft`** data.

5. Printing the DATA Step View

Print the **`laircraft`** data.

6. Printing the DATA Step File Successfully

Print the **`saircraft`** data.

7. Investigating the Results

Answer the following questions:

a. Why was the first attempt to print **`saircraft`** unsuccessful?

___

b. Why was the second attempt to print **`saircraft`** successful?

___

7.5 Solutions to Exercises

1. Creating Reduced-Length Numeric Variables and Compressed SAS Data Files

Use the program, c07ex1start, as a starting program for the following:

a. Submit the program and record the number of pages and the page size for the data set **sales**.

```
data sales;
   infile 'sales.dat' missover;     /* Windows and UNIX */
*  infile '.prog3.rawdata(sales)';  /* Mainframe        */
   input @1 FlightID $7.       @8 RouteID $7.
         @15 Origin $3.        @18 Dest $3.
         @21 DestType $13.     @34 FltDate date9.
         @43 Cap1st 3.         @46 CapBus 3.
         @49 CapEcon 3.        @52 CapPassTotal 3.
         @55 CapCargo 6.       @62 Num1st 3.
         @64 NumBus 3.         @67 NumEcon 3.
         @70 NumPassTotal 3.   @73 Rev1st  7.
         @80 RevBus 7.         @87 RevEcon 7.
         @94 RevCargo 7.       @102 RevTotal 10.
         @112 CargoWeight 5.;
run;

proc contents data = sales;
run;
```

b. Edit the program to the length of the numeric variables **Cap1st**, **CapBus**, **CapEcon** to 3; **CapCargo**, **Num1st**, **NumBus**, **NumEcon**, **NumPassTotal**, **CapPassTotal**, **CargoWeight** and **FltDate** to 4; and **Rev1st**, **RevBus**, **RevEcon**, **RevCargo** and **RevTotal** to 5.

Change the name of the output data set to **salesnum**. Resubmit it, and record the number of pages and the page size for the data set **salesnum**.

```
data salesnum;
   length Cap1st CapBus CapEcon 3
          CapCargo Num1st NumBus NumEcon NumPassTotal
          CapPassTotal CargoWeight FltDate 4
          Rev1st RevBus RevEcon RevCargo RevTotal 5;

   infile 'sales.dat' missover;     /* Windows and UNIX */
*  infile '.prog3.rawdata(sales)';  /* Mainframe        */

   input @1 FlightID $7.       @8 RouteID $7.
         @15 Origin $3.        @18 Dest $3.
         @21 DestType $13.     @34 FltDate date9.
         @43 Cap1st 3.         @46 CapBus 3.
         @49 CapEcon 3.        @52 CapPassTotal 3.
         @55 CapCargo 6.       @62 Num1st 3.
         @64 NumBus 3.         @67 NumEcon 3.
         @70 NumPassTotal 3.   @73 Rev1st  7.
         @80 RevBus 7.         @87 RevEcon 7.
         @94 RevCargo 7.       @102 RevTotal 10.
         @112 CargoWeight 5.;
run;

proc contents data = salesnum;
run;
```

c. Edit the original c07ex1start program to create a compressed data set using COMPRESS=CHAR. Change the name of the output data set to **saleschar**. Be sure not to use the reduced length numeric program to create **saleschar**. Submit the program, and record the number of pages and the page size for the data set **saleschar**.

```
data saleschar (compress = char);
   infile 'sales.dat' missover;     /* Windows and UNIX */
*  infile '.prog3.rawdata(sales)';  /* Mainframe        */
   input @1 FlightID $7.       @8 RouteID $7.
         @15 Origin $3.        @18 Dest $3.
         @21 DestType $13.     @34 FltDate date9.
         @43 Cap1st 3.         @46 CapBus 3.
         @49 CapEcon 3.        @52 CapPassTotal 3.
         @55 CapCargo 6.       @62 Num1st 3.
         @64 NumBus 3.         @67 NumEcon 3.
         @70 NumPassTotal 3.   @73 Rev1st  7.
         @80 RevBus 7.         @87 RevEcon 7.
         @94 RevCargo 7.       @102 RevTotal 10.
         @112 CargoWeight 5.;
run;

proc contents data = saleschar;
run;
```

d. Edit the program to create a compressed data set using COMPRESS=BINARY. Change the name of the output data set to **`salesbin`**. Resubmit it, and record the number of pages and page size for the data set **`salesbin`**.

```
data salesbin (compress = binary);
   infile 'sales.dat' missover;     /* Windows and UNIX */
*  infile '.prog3.rawdata(sales)';  /* Mainframe        */
   input @1 FlightID $7.       @8 RouteID $7.
         @15 Origin $3.        @18 Dest $3.
         @21 DestType $13.     @34 FltDate date9.
         @43 Cap1st 3.         @46 CapBus 3.
         @49 CapEcon 3.        @52 CapPassTotal 3.
         @55 CapCargo 6.       @62 Num1st 3.
         @64 NumBus 3.         @67 NumEcon 3.
         @70 NumPassTotal 3.   @73 Rev1st  7.
         @80 RevBus 7.          @87 RevEcon 7.
         @94 RevCargo 7.       @102 RevTotal 10.
         @112 CargoWeight 5.;
run;

proc contents data = salesbin;
run;
```

2. Comparing CPU Time

Submit the program, c07ex2start, and compare the user CPU time for reading **`sales`**, **`salesnum`**, **`saleschar`**, and **`salesbin`**.

SAS Log

```
318  options fullstimer;
319
320  data _null_;
321    set sales;
322  run;

NOTE: There were 329264 observations read from the data set WORK.SALES.
NOTE: DATA statement used (Total process time):
      real time           0.11 seconds
      user cpu time       0.07 seconds
      system cpu time     0.04 seconds
      Memory                             153k

323
324  data _null_;
325    set salesnum;
326  run;

NOTE: There were 329264 observations read from the data set WORK.SALESNUM.
NOTE: DATA statement used (Total process time):
      real time           0.09 seconds
      user cpu time       0.06 seconds
      system cpu time     0.04 seconds
      Memory                             147k
```

(Continued on the next page.)

```
327
328  data _null_;
329    set saleschar;
330  run;

NOTE: There were 329264 observations read from the data set WORK.SALESCHAR.
NOTE: DATA statement used (Total process time):
      real time           0.50 seconds
      user cpu time       0.40 seconds
      system cpu time     0.04 seconds
      Memory                             153k

331
332  data _null_;
333    set salesbin;
334  run;

NOTE: There were 329264 observations read from the data set WORK.SALESBIN.
NOTE: DATA statement used (Total process time):
      real time           0.64 seconds
      user cpu time       0.60 seconds
      system cpu time     0.02 seconds
      Memory                             153k
```

3. Creating a DATA Step View

Use the program, c07ex3start as a starting program. Write one DATA step to create both a view and a file.

HINT: Investigate the **Reference Information** on **Creating a VIEW and a FILE**.

a. Name the DATA step view **`laircraft`**. The view should contain the aircraft where the **`CapTotal`** value is over 200.

b. Name the data file **`saircraft`**. The file should contain the aircraft where the **`CapTotal`** value is less than or equal to 200.

```
data laircraft saircraft / view = laircraft;
   infile air;
   input ModelType $15. Model $8. AircraftID $6.
         CapFirst 4. CapBusiness 4. CapEconomy 4.
         CapTotal 5. CapCargo 6. Range 6.
         InServiceDate Date9. LastMaintDate Date9.
         CruiseSpeed 6.;
   if CapTotal > 200 then output laircraft;
   else output saircraft;
run;
```

4. Printing the DATA Step File Unsuccessfully

Attempt to print the **`saircraft`** data.

```
filename air 'aircraft.dat'; *Windows/UNIX;
* filename air '.prog3.rawdata(aircraft)';  *z/OS;

proc print data = saircraft;
run;
```

5. Printing the DATA Step View

Print the **laircraft** data.

```
filename air 'aircraft.dat';  *Windows/UNIX;
* filename air '.prog3.rawdata(aircraft)';  *z/OS;

proc print data = laircraft;
run;
```

6. Printing the DATA Step File Successfully

Print the **saircraft** data.

```
filename air 'aircraft.dat';  *Windows/UNIX;
* filename air '.prog3.rawdata(aircraft)';  *z/OS;

proc print data = saircraft;
run;
```

7. Investigating the Results

Answer the following questions:

a. Why was the first attempt to print **saircraft** unsuccessful?

The file **saircraft** is not created until the view is accessed.

b. Why was the second attempt to print **saircraft** unsuccessful?

Printing **laircraft** automatically executed the compiled code for **laircraft**. Therefore, the **saircraft** file was created.

Chapter 8 Utilizing Best Practices to Improve Efficiency

8.1 Introduction

Objectives

- Review best practice techniques.

3

What Are Best Practices

Best practices reduce usage of five critical resources to improve system performance:

- CPU
- I/O
- disk space
- memory
- network traffic

Reducing one resource often increases another.

4

Techniques for Conserving CPU

- Execute only necessary statements.
- Eliminate unnecessary passes of the data.
- Read and write only the data that you require.
- Do not reduce the length of numeric variables.
- Do not compress SAS data sets.

5

Techniques for Reducing I/O Operations

- Process only the necessary variables and observations.
- Reduce the number of times that data is processed.
- Reduce the number of data accesses using the appropriate BUFSIZE= and BUFNO= options for the way that the data is accessed.
- Create a SAS data set, if you process the same data (other than SAS data) repeatedly. SAS can process SAS data sets more efficiently than it can process raw data files.
- Create indexes on variables used for WHERE processing.

6

Because the CPU performs all the processing that is needed to perform an I/O operation, an option or technique that reduces the number of I/O operations can also reduce CPU usage.

Techniques for Reducing Disk Space

- Process only the necessary variables.
- Create reduced length numerics.
- Compress SAS data files.

7

Reducing Memory Usage

- Use KEEP= and DROP= so that only relevant variables consume memory during processing.
- Use small data set page sizes. This can also reduce I/O for data sets that are accessed in a sparse random pattern and can minimize wasted disk space for small SAS data files.
- Use a small value for BUFNO= when the data is accessed randomly instead of sequentially.
- Create a small copy of a large data file with only the observations and variables that are used by subsequent reporting or analysis steps.

The techniques that reduce CPU and I/O can increase memory usage.
Benchmark carefully to balance the need to conserve memory with the need to reduce CPU and I/O.

8

Techniques to Reduce Network Traffic

- Manipulate the data as close to the source of the data as possible.
- Transfer subsets of data or summarized data.

9

Utilizing Best Practices

This chapter presents best practices not discussed in previous chapters.

- Execute only necessary statements.
- Eliminate unnecessary passes of the data.
- Read and write only the data that you require.
- Utilize networking efficiently.

> The data set **`ia.sales`** used for demonstrations and exercises contains fewer observations than the data set **`ia.sales`** used for the course notes.

10

8.2 Executing Only Necessary Statements

Objectives

Use the most efficient technique to perform the following tasks:

- Subset your data by using the subsetting IF statement.
- Use IF-THEN/ELSE or SELECT statements to create new variables.

12

Execute Only Necessary Statements

You minimize the CPU time that SAS uses when you execute the minimum number of statements in the most efficient order.

Techniques for executing only the statements that you require include the following:

- subsetting your data as soon as logically possible
- processing your data conditionally by using the most appropriate syntax for your data

13

Subsetting IF Statement at Bottom of Step

Create a new SAS data set from **ia.sales**. The new SAS data set should contain four new variables and only those flights filled to less than 80% capacity.

```
data totals;
   set ia.sales;
   PercentCap =
      sum(Num1st,NumEcon,NumBus)/CapPassTotal;
   NumNonEconomy = sum(Num1st,NumBus);
   CargoKG = CargoWeight*0.454;
   Month = month(FltDate);
   if PercentCap < 0.8;
run;
```

14 c08s2d1a

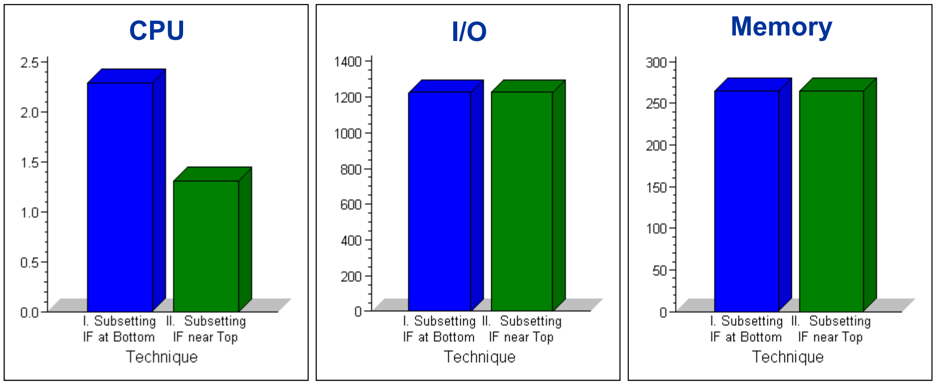
Subsetting IF Statement as High as Possible

```
data totals;
   set ia.sales;
   PercentCap =
      sum(Num1st,NumEcon,NumBus)/CapPassTotal;
   if PercentCap < 0.8;
   NumNonEconomy = sum(Num1st,NumBus);
   CargoKG = CargoWeight*0.454;
   Month = month(FltDate);
run;
```

15 c08s2d1b

Comparing Techniques

Technique	CPU	I/O	Memory
I. Subsetting IF at Bottom	2.3	1226.0	265.0
II. Subsetting IF near Top	1.3	1226.0	265.0
Percent Difference	42.8	0.0	0.0



16

All of the benchmarks were run on HP-UX 11 (64-bit) in SAS 9.1.3 SP2.

Using Conditional Logic

You can use *conditional logic* to alter the way that SAS processes specific observations.

IF-THEN/ELSE statement	executes a SAS statement for observations that meet specific conditions.
SELECT statement	executes one of several statements or groups of statements.

17

Using Parallel IF Statements

For the data in **ia.sales**, create a variable named **Month**, based on the existing variable **FltDate**.

```
data month;
   set ia.sales;
   if month(FltDate) = 1 then Month = 'Jan';
   if month(FltDate) = 2 then Month = 'Feb';
   if month(FltDate) = 3 then Month = 'Mar';
   if month(FltDate) = 4 then Month = 'Apr';
   if month(FltDate) = 5 then Month = 'May';
   if month(FltDate) = 6 then Month = 'Jun';
   if month(FltDate) = 7 then Month = 'Jul';
   if month(FltDate) = 8 then Month = 'Aug';
   if month(FltDate) = 9 then Month = 'Sep';
   if month(FltDate) = 10 then Month = 'Oct';
   if month(FltDate) = 11 then Month = 'Nov';
   if month(FltDate) = 12 then Month = 'Dec';
run;
```

18 c08s2d2a

Using ELSE-IF Statements

```
data month;
   set ia.sales;
   if month(FltDate) = 1 then Month = 'Jan';
   else if month(FltDate) = 2 then Month = 'Feb';
   else if month(FltDate) = 3 then Month = 'Mar';
   else if month(FltDate) = 4 then Month = 'Apr';
   else if month(FltDate) = 5 then Month = 'May';
   else if month(FltDate) = 6 then Month = 'Jun';
   else if month(FltDate) = 7 then Month = 'Jul';
   else if month(FltDate) = 8 then Month = 'Aug';
   else if month(FltDate) = 9 then Month = 'Sep';
   else if month(FltDate) = 10 then Month = 'Oct';
   else if month(FltDate) = 11 then Month = 'Nov';
   else if month(FltDate) = 12 then Month = 'Dec';
run;
```

19 c08s2d2b

Using the Function Only Once

```
data month(drop=mon);
   set ia.sales;
   mon = month(FltDate);
   if mon = 1 then Month = 'Jan';
   else if mon = 2 then Month = 'Feb';
   else if mon = 3 then Month = 'Mar';
   else if mon = 4 then Month = 'Apr';
   else if mon = 5 then Month = 'May';
   else if mon = 6 then Month = 'Jun';
   else if mon = 7 then Month = 'Jul';
   else if mon = 8 then Month = 'Aug';
   else if mon = 9 then Month = 'Sep';
   else if mon = 10 then Month = 'Oct';
   else if mon = 11 then Month = 'Nov';
   else if mon = 12 then Month = 'Dec';
run;
```

20 c08s2d2c

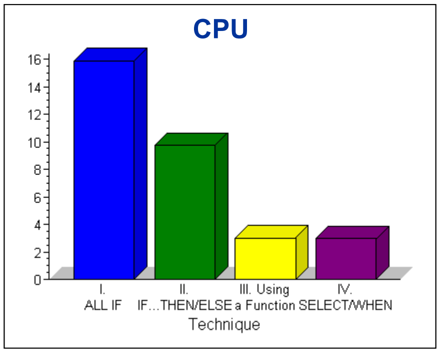
Using a SELECT Block

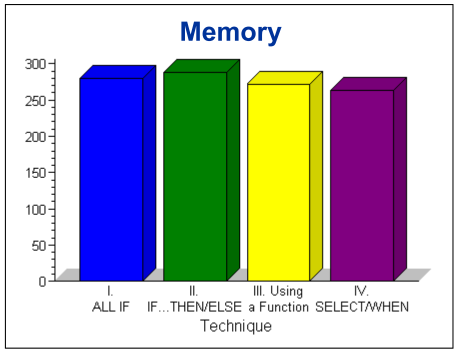
```
data month;
   set ia.sales;
   select(month(FltDate));
      when(1) Month = 'Jan';  when(2) Month = 'Feb';
      when(3) Month = 'Mar';  when(4) Month = 'Apr';
      when(5) Month = 'May';  when(6) Month = 'Jun';
      when(7) Month = 'Jul';  when(8) Month = 'Aug';
      when(9) Month = 'Sep';  when(10) Month = 'Oct';
      when(11) Month = 'Nov'; when(12) Month = 'Dec';
      otherwise;
   end;
run;
```

21 c08s2d2d

Comparing Techniques

Technique	CPU	I/O	Memory
I. ALL IF Statements	15.9	6797.0	280.0
II. ELSE-IF Statements	9.7	6797.0	288.0
III. Using a Function Once	3.0	6797.0	272.0
IV. SELECT/WHEN Block	3.0	6795.0	263.0





22

The I/O for each technique is the same.

Guidelines for Writing Efficient IF/THEN Logic

- Use IF-THEN/ELSE statements when the following circumstances exist:
  - There are few conditions to check.
  - The data values are not uniformly distributed.
  - The values are character or discrete numeric data.
  - There are bounded ranges of data (for example, 1<x<2) .
- For mutually exclusive conditions, use the ELSE-IF statement rather than an IF statement for all conditions except the first.
- Check the most frequently occurring condition first.
- When you execute multiple statements based on a condition, put the statements into a DO group.

23

To determine the distribution of your data values, use the following:

- FREQ procedure to examine the distribution of the data values
- GCHART or GPLOT procedure to display the distribution graphically
- UNIVARIATE procedure to examine distribution statistics and display the information graphically

Guideline for Using a SELECT Statement

Use a SELECT statement when you have a long series of mutually exclusive conditions.

24

SELECT statements perform slightly better for a large selection of uniformly distributed numeric values.

8.3 Eliminating Unnecessary Passes through the Data

Objectives

Use the most efficient technique to accomplish the following tasks:

- Create multiple subsets.
- Create a sorted subset.
- Modify variable attributes.

26

Eliminate Unnecessary Passes of the Data

Avoid reading or writing data more than necessary in order to minimize I/O operations.

Techniques include the following:

- creating multiple output data sets from one pass of the input data, rather than processing the input data each time that you create an output data set
- creating sorted subsets with the SORT procedure

27

Multiple DATA Steps

Create six subsets from **ia.sales**, one for each destination on the East Coast.

```
data rdu;
   set ia.sales;
   if Dest = 'RDU';
run;
data bos;
   set ia.sales;
   if Dest = 'BOS';
run;
```

continued...

28

c08s3d1a

Multiple DATA Steps

```
data iad;
   set ia.sales;
   if Dest = 'IAD';
run;
data jfk;
   set ia.sales;
   if Dest = 'JFK';
run;
data mia;
   set ia.sales;
   if Dest = 'MIA';
run;
data pwm;
   set ia.sales;
   if Dest = 'PWM';
run;
```

29

c08s3d1a

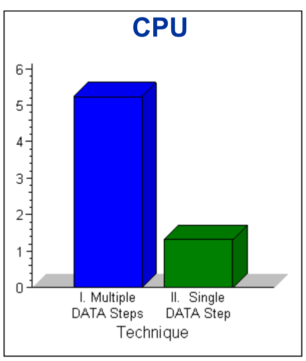
Single DATA Step

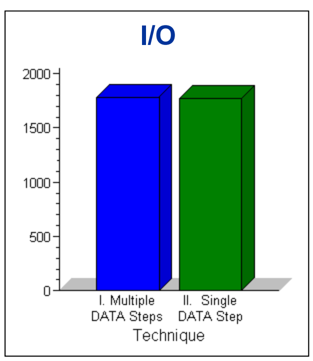
```
data rdu bos iad jfk mia pwm;
   set ia.sales;
   if Dest = 'RDU' then output rdu;
   else if Dest = 'BOS' then output bos;
   else if Dest = 'IAD' then output iad;
   else if Dest = 'JFK' then output jfk;
   else if Dest = 'MIA' then output mia;
   else if Dest = 'PWM' then output pwm;
run;
```

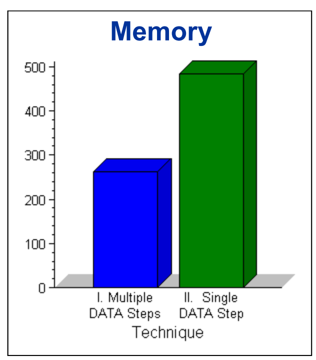
30 c08s3d1b

Comparing Techniques

Technique	CPU	I/O	Memory
I. Multiple DATA Steps	5.2	1781.0	262.0
II. Single DATA Step	1.3	1774.0	483.0
Percent Difference	74.8	0.4	-84.4







31

The memory increases for the single DATA step because multiple data sets are open in memory for output.

DATA Step / PROC SORT Step

Create a sorted subset of **ia.sales** that contains the flights to the East Coast.

```
data east;
   set ia.sales;
   where Dest in
        ('RDU','BOS','IAD','JFK','MIA','PWM');
run;
proc sort data = east;
   by Dest;
run;
```

32 c08s3d2a

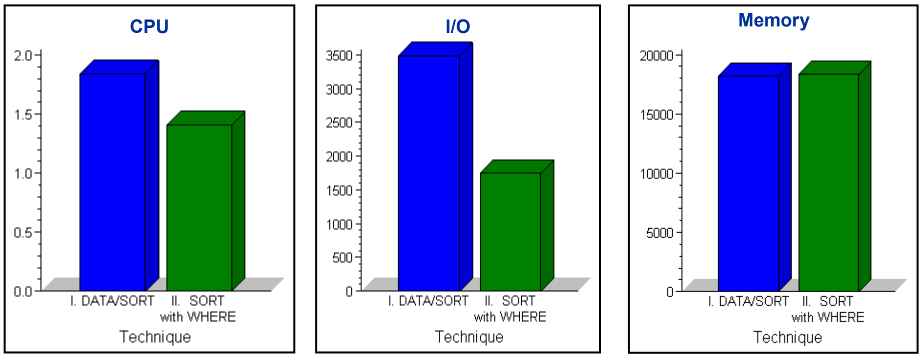
PROC SORT Step

```
proc sort data = ia.sales out = east;
   by Dest;
   where Dest in
       ('RDU','BOS','IAD','JFK','MIA','PWM');
run;
```

33 c08s3d2b

Comparing Techniques

Technique	CPU	I/O	Memory
I. DATA/SORT	1.8	3490.0	18199
II. SORT with WHERE	1.4	1745.0	18355
Percent Difference	23.4	50.0	-0.9



34

Business Task

Change the variable attributes in **`ia.salesc`** to be consistent with those in **`ia.sales`**.

	Var Name	Var Format
`ia.sales`	**`FlightID`**	$7.
	`FltDate`	DATE9.
`ia.salesc`	**`FlightIDNumber`**	$7.
	`FltDate`	MMDDYYP10.

35

DATA Step / PROC DATASETS

```
data ia.salesc;
   set ia.salesc;
   rename FlightIDNumber = FlightID;
   format FltDate date9.;
run;
```

c08s3d3a

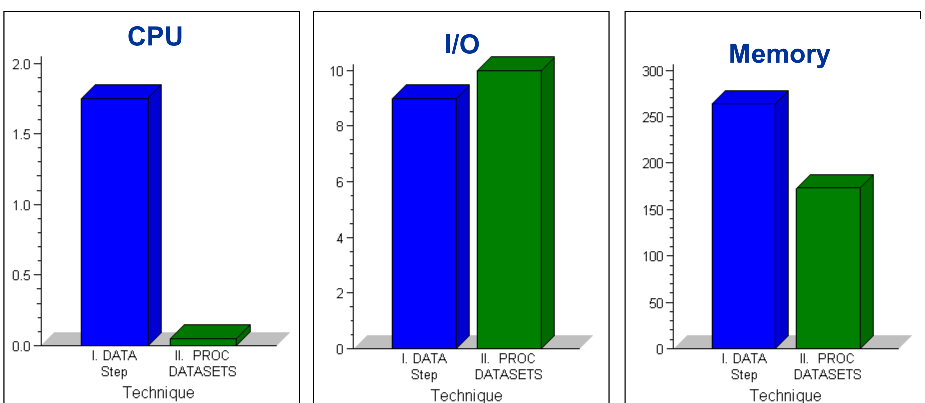
```
proc datasets library=ia nolist;
   modify salesc;
      rename FlightIDNumber=FlightID;
      format FltDate date9.;
quit;
```

36

c08s3d3b

Comparing Techniques

Technique	CPU	IO	Memory
I. DATA Step	1.8	9.0	264.0
II. PROC DATASETS	0.1	10.0	173.0
Percent Difference	97.1	-11.1	34.5



37

8.4 Reading and Writing Only Essential Data

Objectives

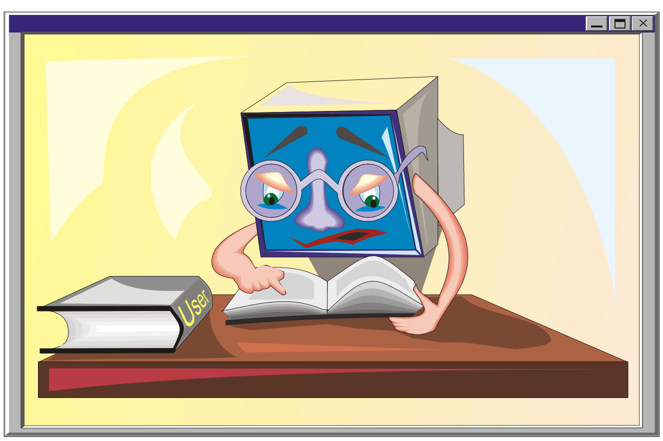
Use the most efficient technique to select the following;

- observations
- variables

39

Read and Write Data Selectively

If you process fewer variables and observations,
CPU and/or I/O operations can be affected significantly.



40

Selecting Observations

WHERE **Dest** = "BWI"

Destination	Flight Number	Route Number
BWI	SE00007	0000206
ATL	SE0003	0000202
GSP	SE0001	0000200
BWI	SE0006	0000206

41 ...

Selecting Observations

IF **Dest** = "BWI"

Destination	Flight Number	Route Number
BWI	SE00007	0000206
ATL	SE0003	0000202
GSP	SE0001	0000200
BWI	SE0006	0000206

42 ...

Subsetting IF versus WHERE

Create a subset of the sales data that contains data for West Coast destinations.

```
data west;
   set ia.sales;
   if Dest in ('LAX','SEA','SFO');
run;
```

c08s4d1a

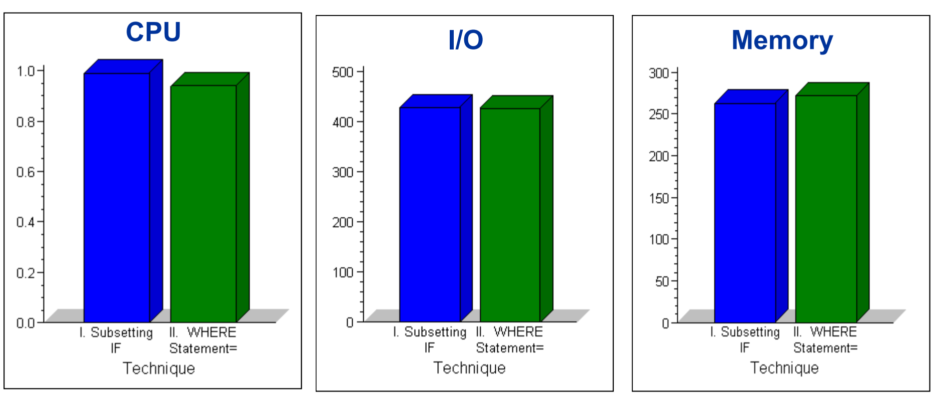
```
data west;
   set ia.sales;
   where Dest in ('LAX','SEA','SFO');
run;
```

43

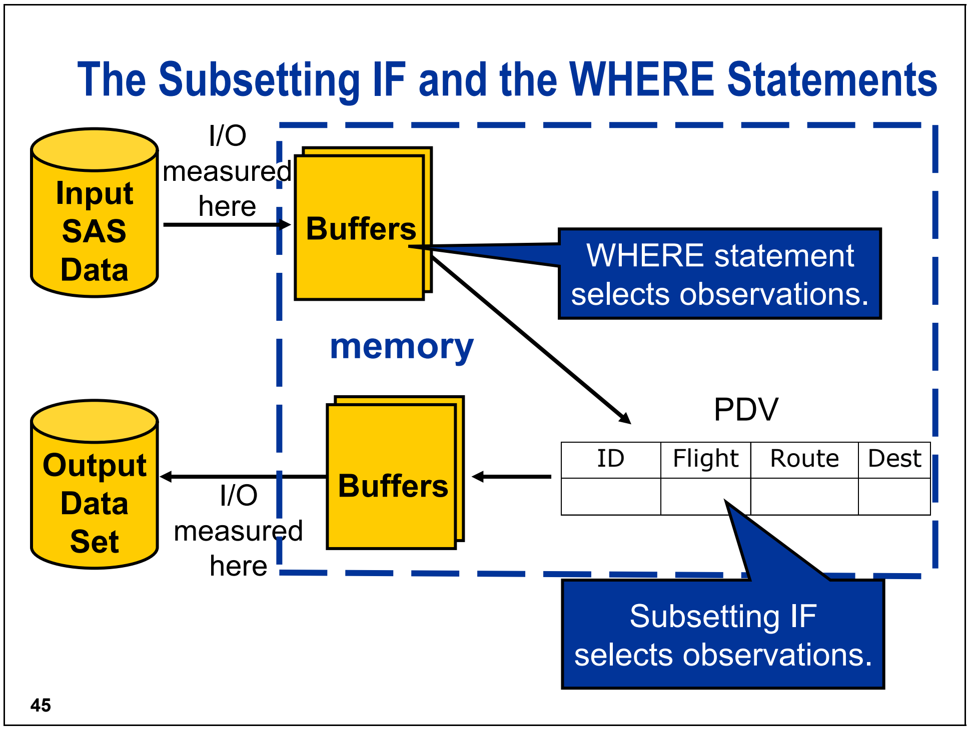
c08s4d1b

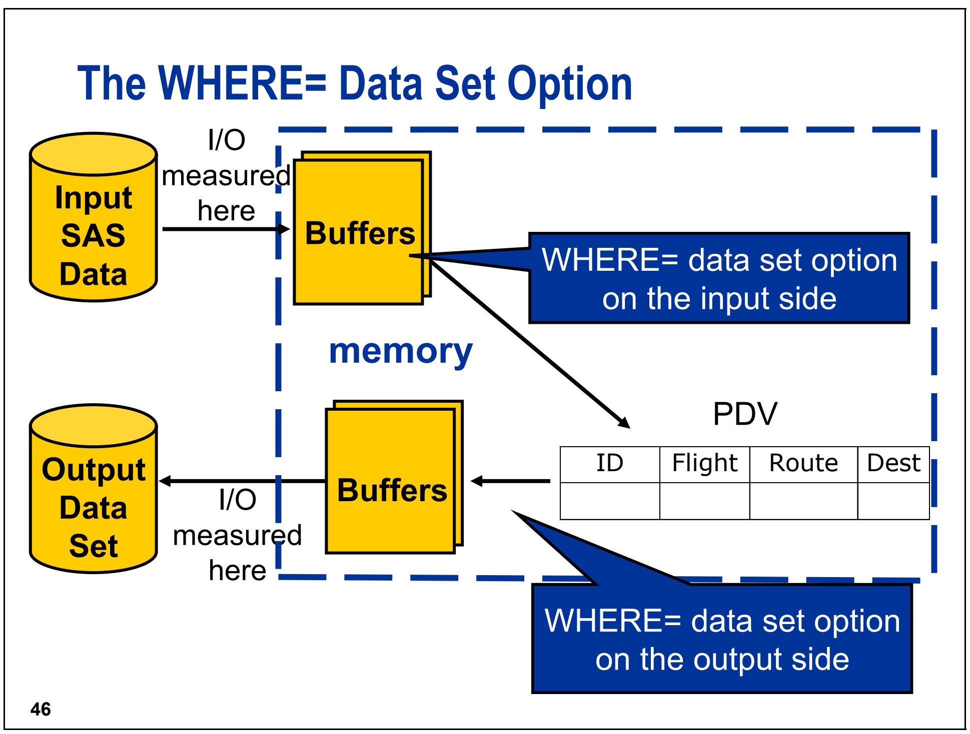
Comparing Techniques

Technique	CPU	I/O	Memory
I. Subsetting IF	1.0	429.0	263.0
II. WHERE Statement	0.9	427.0	272.0
Percent Difference	5.1	0.5	-3.4



44





Input operations are not affected by the subsetting IF, the WHERE statement, or the WHERE= data set options.

Reference Information

The WHERE and subsetting IF statement are not equivalent. While both statements test a condition to determine whether SAS should process an observation, there are differences:

- The WHERE statement selects observations **before** they are brought into the PDV. The subsetting IF statement works on observations **after** they are read into the PDV.
- The WHERE statement can produce a different data set than the subsetting IF when a BY statement accompanies a SET, MERGE, or UPDATE statement.
- When you use the subsetting IF statement with the MERGE statement, SAS selects observations **after** the current observations are combined. When you use the WHERE statement, SAS applies the selection criteria to each input data set **before** it combines observations.
- The WHERE statement can select observations only from SAS data sets. The subsetting IF statement selects observations from SAS data sets, those created with an INPUT statement, or where the selection criteria is based on computed variables.
- The WHERE statement cannot be executed conditionally as part of an IF statement, but the subsetting IF statement can.

If you use the WHERE= data set option and the WHERE statement in the same DATA step, SAS ignores the WHERE statement for data sets with the WHERE= data set option. There is no significant efficiency difference between a WHERE statement and a WHERE= data set option on an input data set.

Subsetting an External File

Create a subset of data that contains only the flights to the West Coast. The data is in an external file that contains information about all flights.

47

Reading All Variables and Subsetting

```
data west;
   infile rawdata ;
   input FlightID $7.  RouteID $7.
         Origin $3.  Dest $3.
         DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
   if Dest in ('LAX','SEA','SFO');
run;
```

48

c08s4d2a

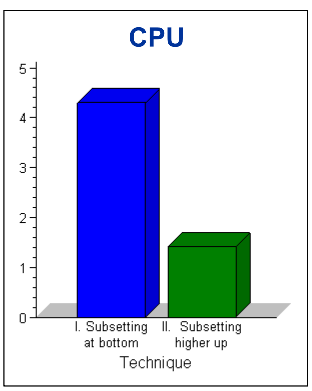
Reading Selected Variable(s) and Subsetting

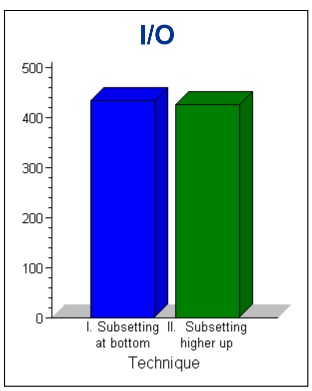
```
data west;
   infile rawdata ;
   input @18 Dest $3. @;
   if Dest in ('LAX','SEA','SFO');
   input @1 FlightID $7.  RouteID $7.
         Origin $3.
         @21 DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
run;
```

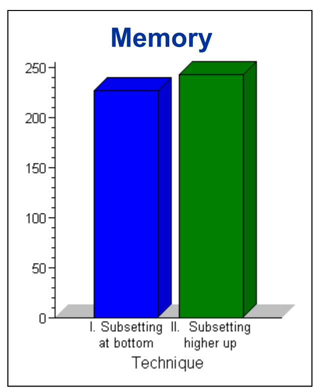
49 c08s4d2b

Comparing Techniques

Technique	CPU	I/O	Memory
I. Subsetting at bottom	4.3	433.0	227.0
II. Subsetting higher up	1.4	425.0	243.0
Percent Difference	67.2	1.8	-7.0

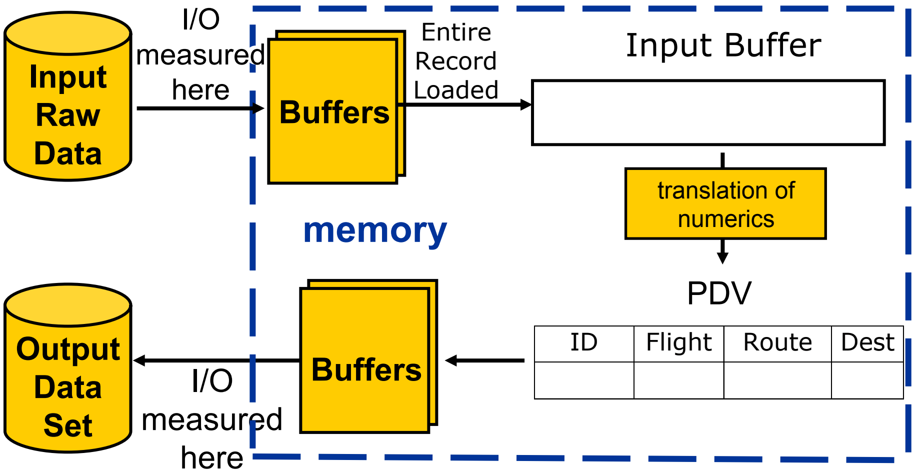






50

Reading External Files

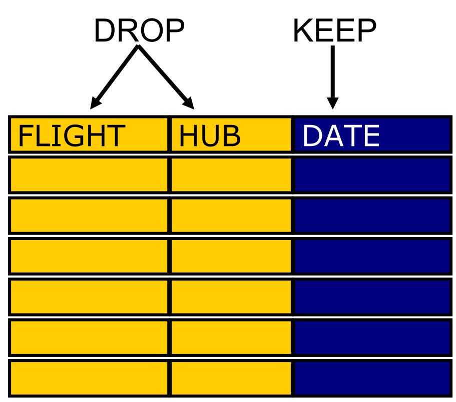


51

Subsetting Variables

To subset variables, you can use the following:

- DROP and KEEP statements
- DROP= and KEEP= data set options



52

Reading and Writing All Variables

Create a report that contains the average and median of the total number of passengers on the flights for each destination in **ia.sales** that has 21 variables.

```
data totals;
   set ia.sales;
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

53 c08s4d3a

Reading All Variables/Writing Two Variables

```
data totals(keep = Dest NonEconPass);
   set ia.sales;
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

54 c08s4d3b

Reading Three Variables

```
data totals;
   set ia.sales(keep = Dest Num1st
                NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

55 c08s4d3c

Reading Three Variables/Writing Two Variables

```
data totals(keep = Dest NonEconPass);
   set ia.sales(keep = Dest Num1st
                NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

56 c08s4d3d

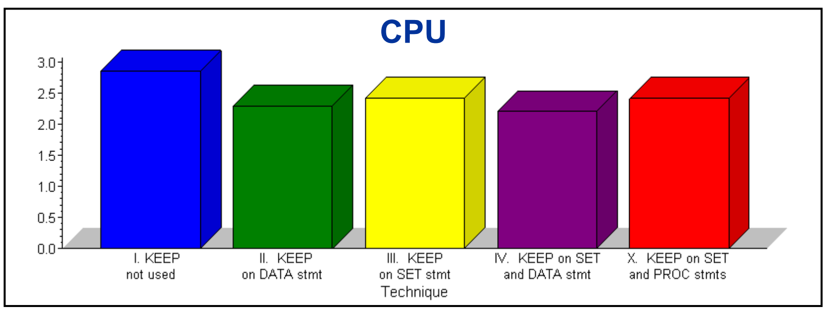
Reading Three Variables/Reading Two Variables

```
data totals;
   set ia.sales(keep = Dest Num1st
                       NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals
                (keep = Dest NonEconPass)
                 mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

57 c08s4d3e

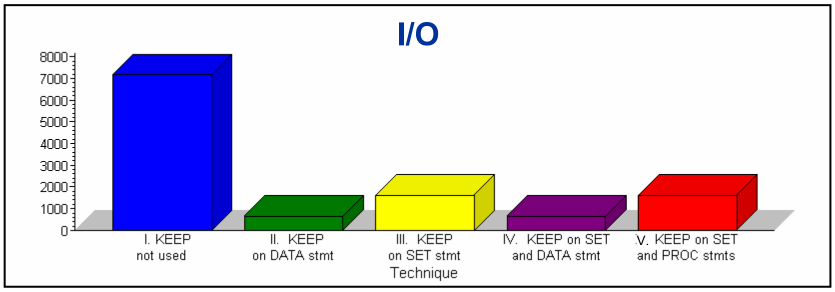
Comparing Techniques

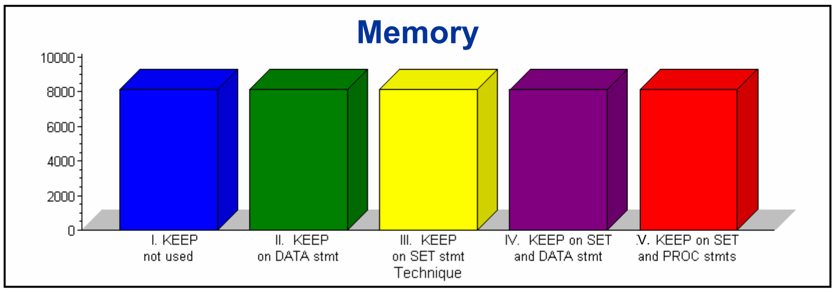
Technique	CPU	I/O	Memory
I. KEEP not used	2.9	7177	8140
II. KEEP on DATA statement	2.3	656	8138
III. KEEP on SET statement	2.4	1625	8138
IV. KEEP on SET and DATA statements	2.2	662	8138
V. KEEP on SET and PROC statements	2.4	1625	8139



58

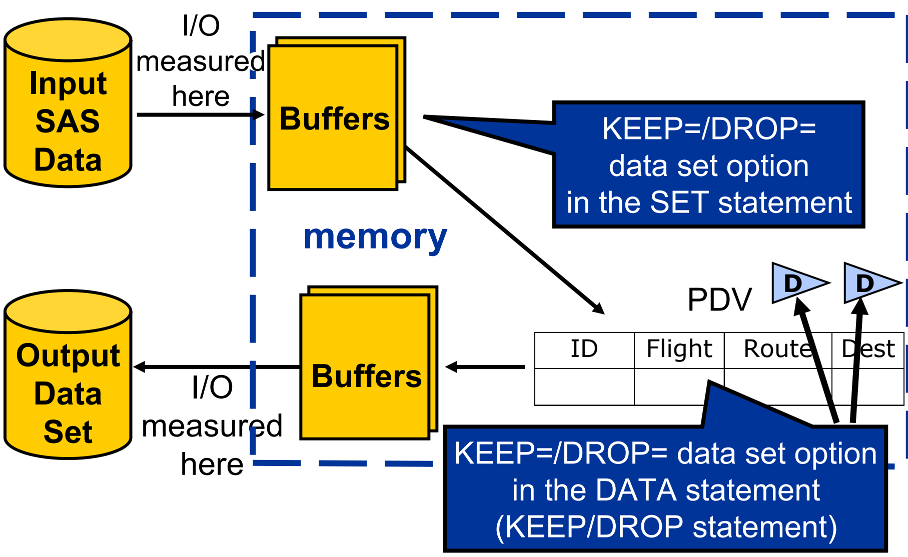
Comparing Techniques





59

Using the KEEP=/DROP= Options



60

Reading All Fields

```
data sales(keep = FlightID Num1st
                  NumBus NumEcon NumPassTotal);
   infile rawdata ;
   input FlightID $7.  RouteID $7.
         Origin $3.  Dest $3.
         DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
run;
```

61 c09s4d4a

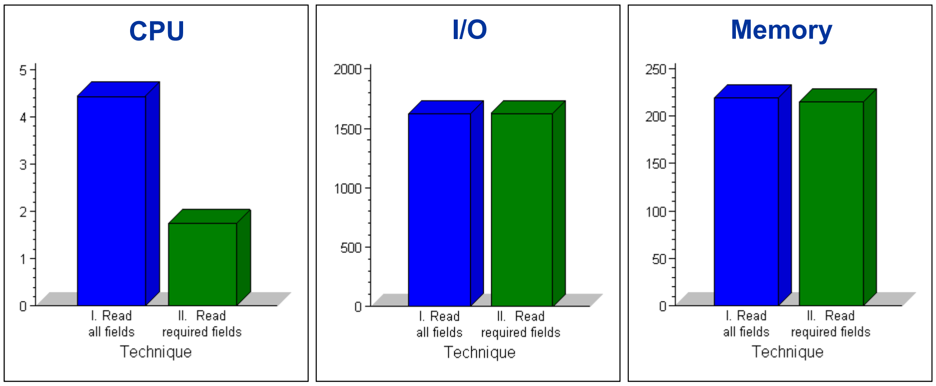
Reading Required Fields

```
data sales;
   infile rawdata ;
   input FlightID $7. @85 Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. ;
run;
```

62 c09s4d4b

Comparing Techniques

Technique	CPU	I/O	Memory
I. Read all fields	4.4	1627.0	219.0
II. Read required fields	1.7	1625.0	215.0
Percent Difference	60.7	0.1	1.8



63

Conclusions

If the variable is already in a SAS data set, you can use the following to minimize the volume of data processed:

- WHERE statements in DATA and PROC steps
- KEEP and DROP statements in the DATA step
- WHERE=, KEEP=, and DROP= data set options in DATA and PROC steps

If the data is not in a SAS data set or the variable is a calculated variable, you can use the following to minimize the volume of data processed:

- subsetting IF statements
- selective INPUT statements

64

8.5 Networking Efficiency Considerations (Self-Study)

Objectives

Examine available efficiency techniques to do the following tasks:

- access database data
- perform remote SAS processing

66

Accessing Database Data

When you access database (DBMS) data, the performance of your SAS job can be influenced by the following:

- technique chosen to access the data
- number of columns and rows returned
- ordering of the rows
- choice of SAS procedures or DATA steps

67

Choosing a DBMS Access Technique

Access your DBMS data with the following primary techniques:

- SAS/ACCESS LIBNAME engine
- SQL Pass-Through Facility

68

The SAS/ACCESS LIBNAME engine writes native DBMS SQL statements from your SAS statements and sends them to the DBMS for processing.

The SQL Pass-Through Facility enables you to write native DBMS SQL statements from within the SQL procedure and pass them directly to the DBMS for processing.

LIBNAME Engine Advantages

DATA and PROC step features:

- You can take advantage of threaded reads.
- The WHERE clause can be passed to DBMS.
- Sort requests can be passed to DBMS.
- Transparent access to DBMS data occurs.
- DATA and PROC step syntax is unchanged.
- Knowledge of DBMS-specific SQL is unnecessary.
- Data retrieval results can be saved as a SAS table or a view.

69

LIBNAME Engine Advantages

Additional SQL procedure features:

- Joins can be passed to DBMS.
- GROUP BY criteria can be passed to DBMS.
- Aggregate functions are passed to DBMS.

70

The list of aggregate functions that are passed varies by database. See the documentation for the SAS/ACCESS Interface to your database for a list of aggregate functions that are passed to your database for processing.

Using SASTRACE and SASTRACELOC

Behind the scenes, when SAS sees that the code references a DBMS table, SAS sends an SQL query directly to the DBMS.

To display this query in the log, you can use the SASTRACE= and the SASTRACELOC= options.

> The SASTRACE= and SASTRACELOC= system options are typically turned on for debugging and off for production jobs.

71

Using SASTRACE and SASTRACELOC

General form of the SASTRACE= option:

```
SASTRACE=',,,d'
```

General form of the SASTRACELOC= option:

```
SASTRACELOC = stdout | SASLOG
```

Example:

```
options sastrace= ',,,d' sastraceloc = saslog;
```

STDOUT is the file reference that can be assigned at invocation for the standard output files.

72

',,,d' specifies that all SQL statements sent to the DBMS are sent to the log. These statements include the following:

- SELECT
- DELETE
- CREATE
- SYSTEM CATALOG
- DROP
- COMMIT
- INSERT
- ROLLBACK
- UPDATE

There are four possible positional arguments to SASTRACE. The commas in the value for the SASTRACE option are placeholders for other debugging options. For other values, please see the SAS documentation.

Threaded Reads

A threaded read retrieves the result set from the database on multiple connections between SAS and the DBMS.

Threaded reads are accomplished by doing the following:

- using the LIBNAME engine
- establishing a read connection between the DBMS and each SAS thread
- partitioning the result set across the connections
- passing the rows to SAS simultaneously (in parallel) across the connections

73

Most, but not all, SAS/ACCESS interfaces support threaded reads in SAS 9.1.

Scope of Threaded Reads

SAS steps, named *threaded applications*, are automatically eligible for a threaded read.

- Base SAS procedures
  - MEANS, REPORT, SORT, SQL, SUMMARY, TABULATE
- SAS/STAT procedures
  - GLM, LOESS, REG, ROBUSTREG
- SAS/SHARE procedure
  - SERVER (with the experimental THREADEDTCP option)
- SAS Enterprise Miner procedures
  - DMINE, DMREG

74

Performance Impact of Threaded Reads

Optimal performance of threaded reads requires the following:

- SAS running on a fast uniprocessor or a multiprocessor machine
- the database running on a high-end symmetric multiprocessor (SMP) machine
- partitioned database table(s)
- similar size partitions
- large DBMS result set

75

Reading Columns

Techniques for limiting the number of columns returned from the DBMS include the following:

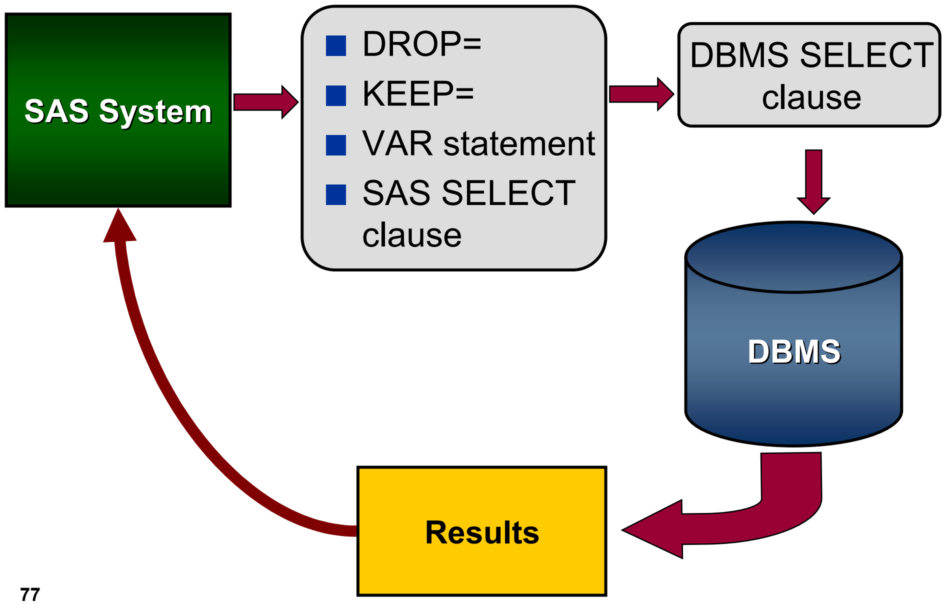
- DROP= SAS data set option
- KEEP= SAS data set option
- VAR statement in the PRINT procedure
- SELECT clause in the SQL procedure

Examples:

```
data temp;
   set mylib.table(keep = name age state);
run;
proc sql;
   select name, age, state
      from mylib.table;
quit;
```

76

Reading Columns



77

Subsetting Using WHERE Criteria

Subset the rows returned from a query to potentially reduce the following:

- processing time
- network traffic
- memory requirements

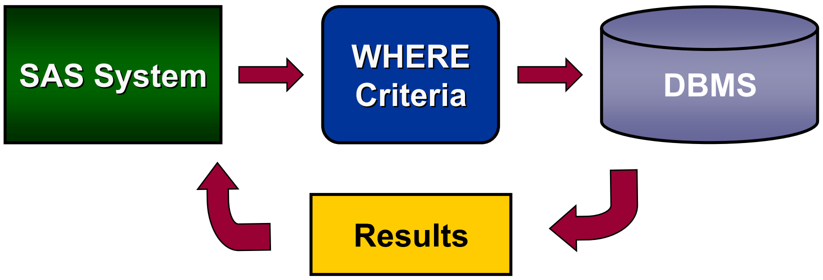
Examples:

```
data temp;
   set mylib.table;
   where state in ('NC', 'SC');
run;
proc sql;
   select *
      from mylib.table
         where state in ('NC', 'SC');
quit;
```

78

Subsetting Using WHERE Criteria

If the SAS/ACCESS engine can do so, the WHERE criteria is passed directly to the database to gain efficiency in processing.



79

Splitting the WHERE Criteria

If the WHERE clause or statement contains SAS enhancements not known to the database, the following events occur:

- The WHERE clause or statement is split up, which enables the DBMS to process as much of the WHERE criteria as possible.
- Rows that satisfy those criteria are sent back to SAS, and then checked to see if they meet the remaining WHERE clause or statement conditions.

80

SAS enhancements include functions or operators that are not a part of the native database SQL. The SASTRACE= system option can help you determine what is passed to the database to process.

Sorting the Rows Returned

If sorting is required, you can perform it by doing the following:

- Using a BY statement in a DATA or PROC step forces the DBMS to sort the data in the order specified by the BY variable(s) before returning the results to SAS.
- If you use an ORDER BY clause in PROC SQL, the ORDER BY clause is passed to the DBMS.

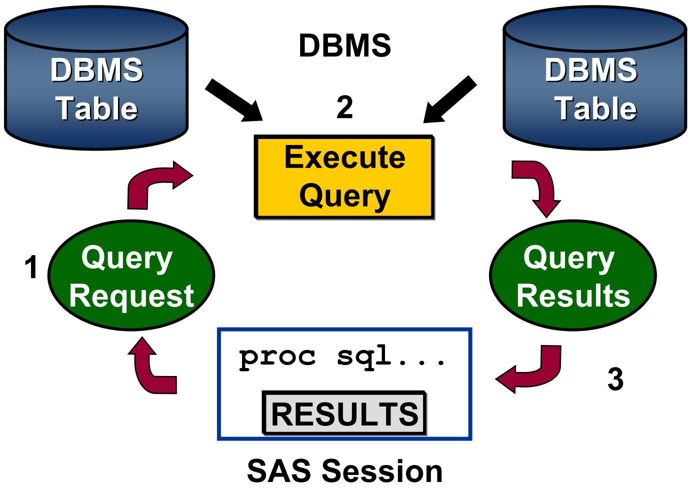
```
data temp;
   set mylib.table;
   by state;
run;

proc sql;
   select * from mylib.table
      order by state;
quit;
```

81

Be aware that SAS sorts null values low; most DBMSs sort null values high.

If you specify a BY statement in a DATA or PROC step that references a DBMS data source, it is recommended for performance reasons that you associate the BY variable (in a DATA or PROC step) with an indexed DBMS column. If you reference DBMS data in a SAS program and the program includes a BY statement for a variable that corresponds to a column in the DBMS table, the SAS/ACCESS LIBNAME engine automatically generates an ORDER BY clause for that variable. The ORDER BY clause causes the DBMS to sort the data before the DATA or PROC step uses the data in a SAS program. If the DBMS table is very large, this sorting can adversely affect your performance. Use a BY variable that is based on an indexed DBMS column in order to reduce this negative impact.

SQL Procedure Pass-Through Facility



82

SQL Pass-Through Advantages

- DBMS can optimize all table joins.
- Results of a query can be saved as a SAS data file.
- A SAS SQL view can contain a pass-through query.

83

SQL Pass-Through Example

```
proc sql;
   connect to DBMS (DBMS-specific connection
                    options);
   select *
      from connection to DBMS
         (select flightnumber, flightdate,
                 dayofweek, delay
             from DBMS-table-name
            where substr(destination, 1, 1)
              = 'C');
   disconnect from DBMS;
quit;
```

84

The Embedded LIBNAME Statement

An alternative to coding the LIBNAME statement or using the SQL Pass-Through Facility when you create a PROC SQL view is the embedded LIBNAME statement. The embedded LIBNAME statement has these characteristics:

- is defined in a USING clause within the PROC SQL view
- is assigned when the view begins to execute
- can contain connection information
- uses the LIBNAME engine to access the DBMS
- can store label, format, and alias information
- is de-assigned when the view completes executing

85

The Embedded LIBNAME Statement

Example:

```
proc sql;
   create view sasuser.joinview as
   select m.FlightNumber, m.FlightDate,
          Deplaned, DayOfWeek, Delay
      from oralib.marchflights as m,
           oralib.flightdelays as f
      where m.flightnumber = f.flightnumber
           and m.flightdate = f.flightdate
           and delay > 0
      using libname oralib engine
            engine-connection-options;
   select * from sasuser.joinview;
quit;
```

86

SAS/ACCESS Summary

The SAS/ACCESS LIBNAME engine enables transparent access to your DBMS tables. As much code as possible is passed behind the scenes by SAS to the DBMS for processing in order to optimize performance.

The SQL Pass-Through Facility enables the programmer to control the native DBMS SQL queries that are passed to the database to execute.

87

Distributed Processing

Distributed processing can be defined as any one of the following:

- one process (a client or local host) requesting services or data from another process (a server or remote host) executing on a different machine
- the distribution of computing resources to enable utilization of data files, hardware resources, and software resources between different computers
- the division of applications into tasks to be performed on the most appropriate machine, thereby maximizing all computing resources

88

Parallel Processing

Parallel processing is the dividing of an application into subunits of work that can be executed simultaneously.

This parallel processing can occur on the same machine or different machines.

The purposes of parallel processing (also known as multiprocessing or asynchronous processing) are to do the following:

- execute independent tasks in parallel (SAS Version 8)
- execute select dependent tasks in parallel (SAS®9)
- take advantage of multiple processors on a *symmetric multiprocessing* (*SMP*) single machine

continued...

89

Parallel Processing

- take advantage of each processor on a network of machines
- complete a job in less total **elapsed** time than it would take to execute the same job serially
- increase usage of underutilized CPUs
  - exploit current investment
  - prevent further monetary outlay for hardware

90

Grid Computing

A *computing grid* is a collection of multiple computers that solve one application problem.

The concept of grid computing is to tap into the unused processor cycles of computers hooked up to a network to solve problems that require a massive amount of processing power and deal with vast amounts of data.

The idea of grid computing is that any device or computer could hook into a network and make use of the collective unused power of every device on the network or grid.

continued...

91

Grid Computing

The goal is to use the processing cycles of all computers in a network for solving problems too intensive for any stand-alone machine.

Grid computing is not a new concept, but one that has gained renewed interest recently for at least two reasons:

- IT budgets were cut, and grid computing offers a less expensive alternative to purchasing new, larger server platforms.
- Computing problems in several industries involve processing large volumes of data and/or performing repetitive computations to the extent that the workload requirements exceed existing server platform capabilities.

92

Distributed Processing Solutions

A distributed processing solution is implemented when an application requires a service from another computer or itself.

Services include the following:

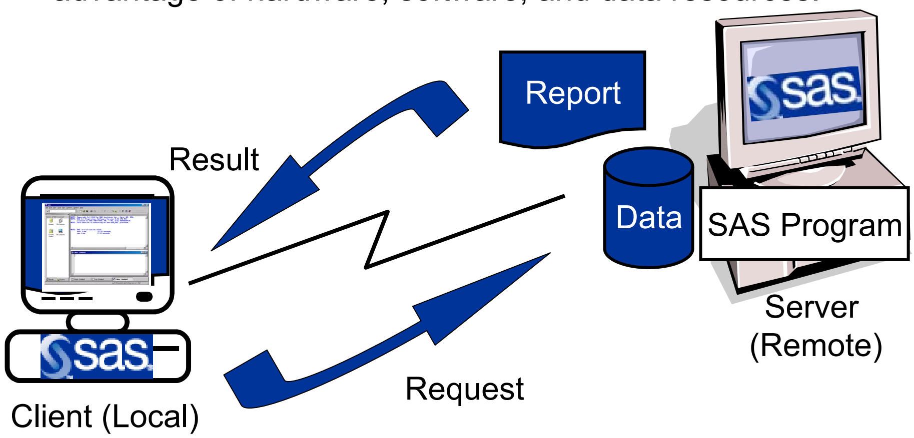
- compute services
- data transfer services
- remote library services (RLS)

93

Distributed processing using SAS software requires a license for SAS/CONNECT, SAS/SHARE, or SAS Integration Technologies.

Compute Services

Compute services enable you to move any or all segments of an application to other processors to take advantage of hardware, software, and data resources.



94 ...

Compute Services Benefits

Compute services are useful when the following conditions exist:

- Processing remote data files that have these attributes:
  - are too large to transfer
  - are frequently updated
  - must remain on the remote platform for security reasons
- The remote machine has necessary hardware or software resources that the local machine does not have.
- A remote CPU is underutilized.

95

Compute Services Considerations

Compute services are less appropriate when these circumstances occur:

- Data files are small.
- A remote CPU is near 100% utilization.
- The remote computer's I/O subsystem is heavily loaded.
- The remote computer has little memory available.

96

Requirements for Compute Services

To use compute services, you need to do the following:

- have SAS/CONNECT on both machines
- sign on to the remote machine to begin a remote SAS session
- submit an RSUBMIT block

97

Using Compute Services

Before you use compute services, a connection to the remote machine must be established. You can do either of the following:

- Sign on directly with a SIGNON statement.
- Use the AUTOSIGNON=YES option to specify to sign on when compute services needs to start a task on the remote machine.

98

Using Compute Services

The AUTOSIGNON option enables the local SAS session to automatically invoke a new SAS session when a request is made.

General form of the AUTOSIGNON option:

OPTIONS AUTOSIGNON = NO|YES;

The default is NO.

Example:

```
options autosignon = yes;
```

99

Using Compute Services

After a connection to a remote machine is established, you can send code to execute on that machine by enclosing the code in an RSUBMIT block.

General form of the RSUBMIT block:

> **RSUBMIT** <*remote-machine-name*>;
>   *code to be processed on the remote machine*
> **ENDRSUBMIT;**

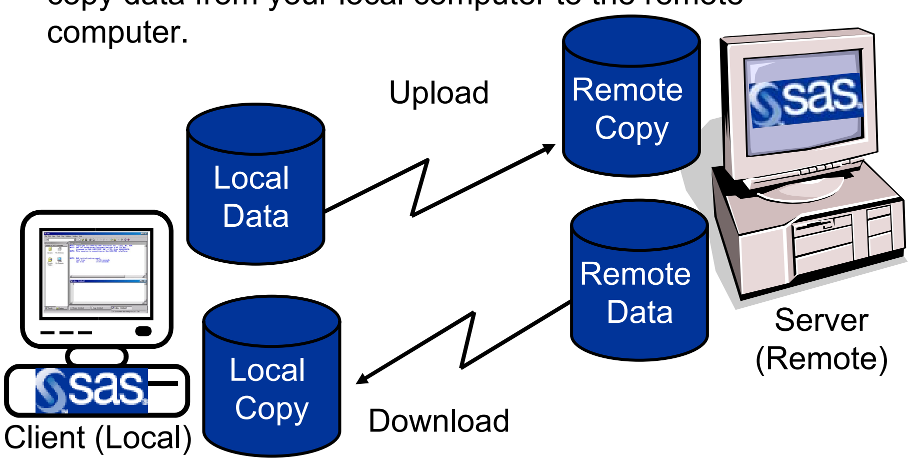
Example:

```
local SAS session
rsubmit bcom1;
  SAS code to run on remote machine
endrsubmit;
```

100

Data Transfer Services

Using data transfer services, you can transfer a copy of a remote data file to your local computer for processing, or copy data from your local computer to the remote computer.



101 ...

✎ You can transfer SAS files, flat files, and extracts of DBMS tables.

The UPLOAD and DOWNLOAD Procedures

To perform data transfer, use the UPLOAD and DOWNLOAD procedures. The UPLOAD and DOWNLOAD procedures enable you to do the following:

- transfer an entire SAS library or selected members of a SAS library in a single step
- transfer an entire SAS catalog or selected entries in a catalog in a single step
- transfer external files

continued...

102

The UPLOAD and DOWNLOAD Procedures

- enable WHERE processing to subset the data before the transfer occurs
- enable data set options (for example, DROP= or KEEP=) when transferring individual SAS data sets
- replicate certain data set attributes, including indexes and constraints

103

UPLOAD and DOWNLOAD Procedure Benefits

Benefits of using the UPLOAD and DOWNLOAD procedures over other data transfer applications are as follows:

- control over variables and observations transferred
- transparent translation of SAS files across operating system types (for example, EBCDIC to ASCII)
- transparent translation of SAS files across differing releases of SAS

104

Transferring a SAS Data Library

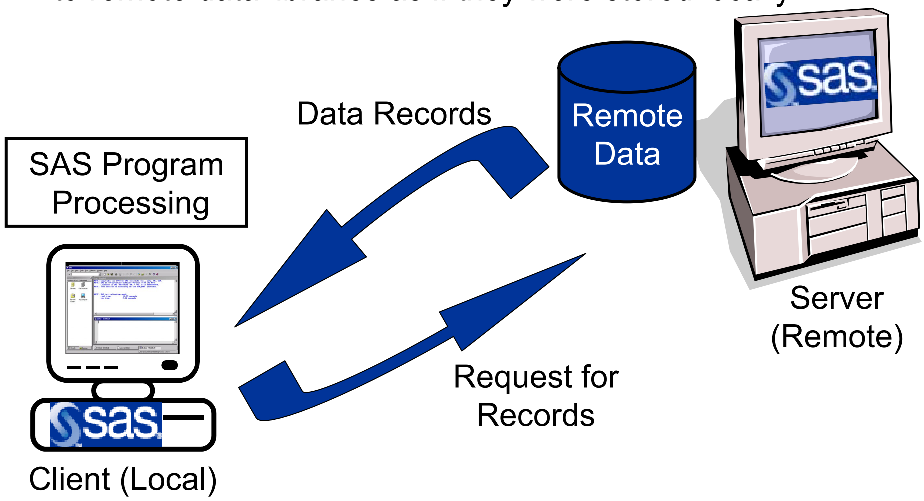
Example: Transfer the entire SAS data library on the remote machine to the local machine.

```
libname orionwin 'directory-on-Windows';
rsubmit bcom1;
libname orionunx 'directory-on-UNIX';
proc download inlib = orionunx
              outlib = orionwin;
run;
endrsubmit;
```

105

Remote Library Services

Remote library services (RLS) provide transparent access to remote data libraries as if they were stored locally.



106 ...

✎ Remote data can be SAS files or external database tables or views.

Benefits of RLS

- A single copy of the data can be maintained while processing is performed on the local machine.
- The data appears to be local.
- RLS enables updates to remote data as a result of local processing.
- RLS permits a user interface to reside on the local system while the data is on a remote system.

107

Considerations for RLS

- Multiple passes of the data require the same data to go across the network multiple times. Examples include the following:
  - statistical procedures
  - multiple PROC steps on the same data
- Network traffic might significantly increase.

108

Requirements for RLS

To use RLS, you need to do one of the following:

- to have SAS/CONNECT on both machines or SAS/CONNECT on the local machine and SAS/SHARE on the remote machine
- to sign on to the remote machine to begin a remote SAS session, if SAS/CONNECT is used on the remote machine
- to issue a LIBNAME statement in your local session with the SERVER= option

109

SERVER= Option

General form of the SERVER= option in the LIBNAME statement:

LIBNAME *libref* '*SAS-data-library*' | SLIBREF=*server-libref*
SERVER=*remote-host*;

Examples:

Access a library stored on your user ID on UNIX:

```
libname rmtunx '/orion/sasdata' server = sdcunx;
```

Access the Work library on z/OS:

```
libname rmtwork slibref = work server = sdcmvs;
```

110

libref	is a libref defined to your local session referencing a remote SAS library.
SAS-data library	is the physical location of the remote SAS library.
server-libref	is an existing libref in the server's session, for example, **Work**.
remote-host	is the same name previously specified with OPTIONS REMOTE=*id* or the value of *server-ID* on the SIGNON statement.

Decisions, Decisions, Decisions

When deciding which strategy is most appropriate for your application, you must determine the following:

- computing needs of your application
- computing capacity and load of each computer
- charge-backs for use of mainframe or UNIX time and data storage
- amount of data to be processed
- load on your network
- output needs
  - printers
  - tape drives
  - GUI display

continued...

111

Decisions, Decisions, Decisions

- appropriateness of the data location
  - the frequency of data updates
  - available disk space
  - the increased speed of the application if the data is on the same computer
  - problems related to storing multiple copies of the data

112

Chapter 9 Using the Scalable Performance Data Engine (Self-Study)

9.1 Introduction to the Scalable Performance Data Engine

Objectives

- Define the Scalable Performance Data Engine (SPDE).
- Discuss symmetric multiprocessing (SMP) machines.
- Compare SPDE tables with Base SAS tables.

3

What Is the Scalable Performance Data Engine?

The Scalable Performance Data (SPD) Engine can be defined as follows:

- is a high-speed alternative to the Base SAS engine for processing very large data sets
- can take advantage of the following:
  - SMP machines
  - multiple I/O channels

4

The SPD Engine is part of Base SAS software and runs on UNIX, Windows, z/OS (zFS file system only), and OpenVMS Alpha (on ODS-5 file systems only).

An SMP machine is a **S**ymmetric **M**ulti**P**rocessor machine, which has more than one CPU and a thread-enabled operating system.

Advantages of the SPD Engine

The SPD Engine provides the following:

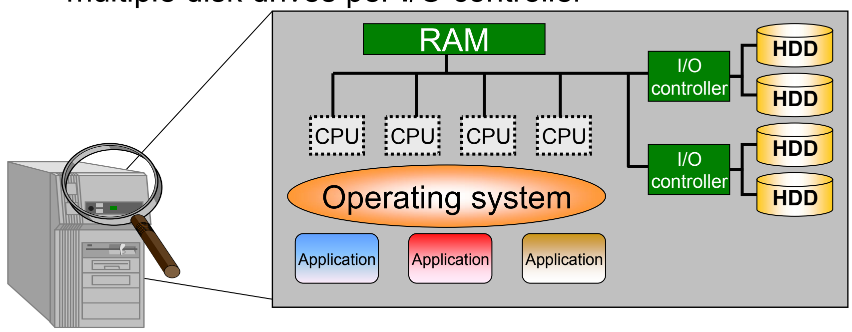
- optimization for the storage and sequential access of large and very large data sets (millions of rows, many gigabytes of data)
- scalability on symmetric multiprocessor (SMP) machines
- parallel WHERE selections
- parallel loads
- parallel index creation
- parallel I/O data delivery to applications
- implicit sorting on BY statements

5

Using an SMP Machine

A symmetric multiprocessing (SMP) machine can be described as follows:

- has multiple central processing units (CPUs) and an operating system that supports threads
- is usually configured with multiple I/O controllers and multiple disk drives per I/O controller



6

The SPD Engine running on an SMP machine provides the capability to read and deliver much more data to an application in a given elapsed time. When the SPD Engine reads a data file, it launches one or more threads for each CPU. These threads read data in parallel from multiple disk drives, driven by one or more controllers.

The exact number of CPUs on an SMP machine varies by manufacturer and model. The operating system of the machine is also specialized; it must be capable of scheduling code segments so that they execute in parallel. If the operating system kernel is threaded, performance is further enhanced because it prevents contention between the executing threads. While threads run on the SMP machine, managed by a threaded operating system, the available CPUs work together. The synergy between the CPUs and threads enables the software to scale processing performance.

Although it is not necessary to utilize an SMP machine for SPD Engine data files, it is highly recommended to achieve maximum performance.

SPD Engine Data Organization

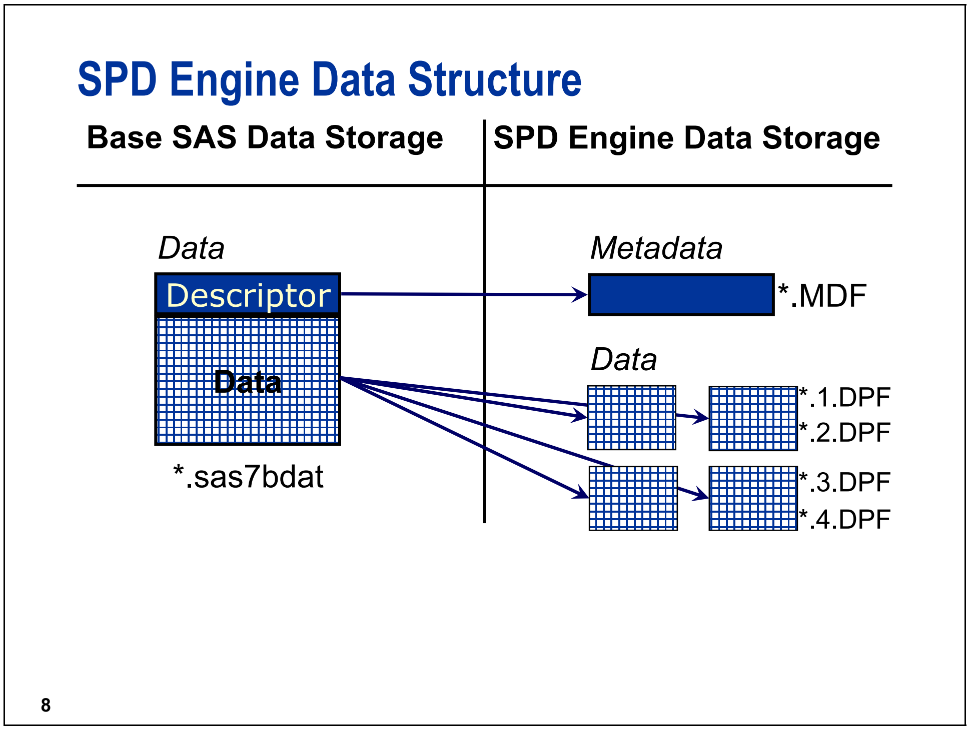
The SPD Engine creates separate component files for the following:

- data
- data descriptor
- two index files, if the data set is indexed

The advantage of the separate component files is the speed of data retrieval.

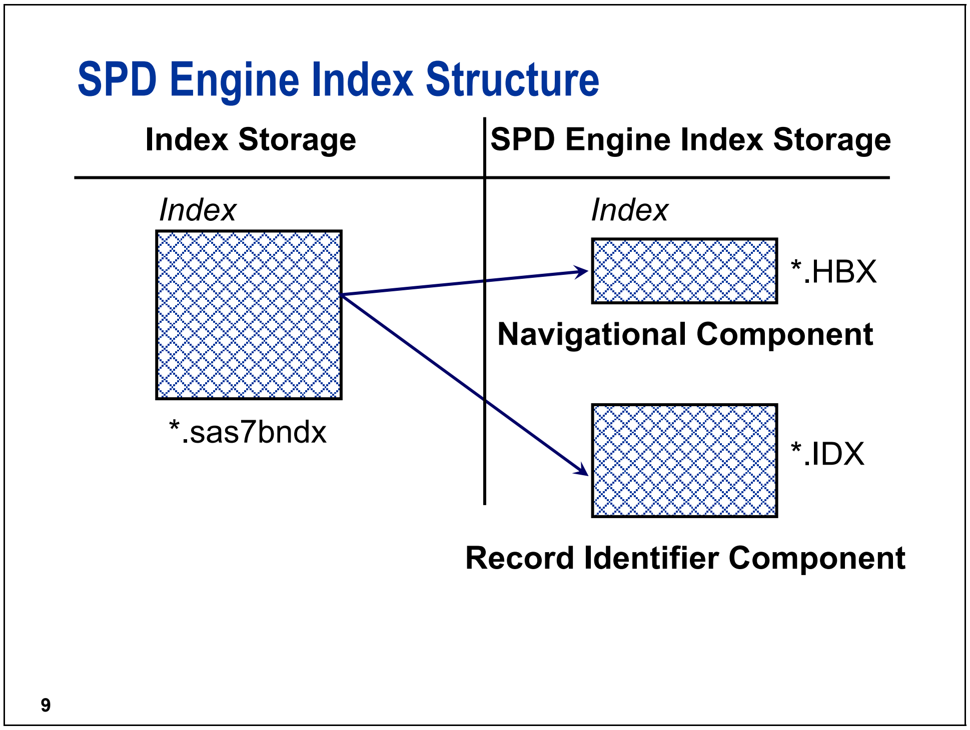
7

Each of these components can comprise one or more physical files so that the components can span volumes, but are referenced as one logical file.



- When a SAS data file is copied from a base engine library to SPD Engine data storage, the file is split into a metadata file (*.mdf) and at least one data file (*.dpf). Because of the particular way data is stored with SPD Engine, several data files (*.1.dpf, *.2.dpf) might also be generated, which splits the data file into several file segments.
- On UNIX file systems, you can use standard commands, such as **`ls`**, to see these files. On Windows platforms, you can use Windows Explorer to see these files.

✎ It is not recommended that you move SPD Engine data files using operating system commands because of disk file segmentation.



SPD Engine creates a separate index file for each index. For example, if five indexes are defined, the SAS base engine stores them all in one index file. There would be at least ten files in SPD Engine data storage, and each would contain the values of the appropriate index variable(s).

The navigational component file (.HBX) has each unique value for an index and the data partitions in which that value can be found. The record identifier component file (.IDX) has pointers to each row in the table containing the value of the index variable(s).

Storing Data with SPD Engine

The SPD Engine usually uses four different areas to store the various components that make up an SPD Engine data set:

- metadata area
- data area
- index area
- work area

For information on disk set-up requirements, consult the Appendix to the SPDE Reference.

10

9.2 Creating SPD Engine Tables

Objectives

- Discuss the LIBNAME statement and the LIBNAME options.
- Create SPDE tables.
- Create SPDE indexes.

12

Using the SPDE LIBNAME Engine

UNIX

```
libname mylib spde '/disk/data';
```

Windows

```
libname mylib spde 'c:\workshop\winsas\prog3\meta';
```

For speed of data retrieval, it is highly recommended that metadata, data, and index files be stored in separate, unique locations.

13

In this example, the index and data components are stored in the same location.

Using the SPDE LIBNAME Engine

General form of the SPDE LIBNAME engine:

LIBNAME libref SPDE '*full-primary-path*' <options> ;

full-primary-path

- is the fully qualified pathname of the primary path for the SPD Engine library
- must be recognized by the operating environment
- must be unique for each library
- is where the metadata is stored

14

The metadata for the library **must** start in the primary path. It can continue in secondary paths using the **METADATA=** option.

DATAPATH= LIBNAME Statement Option

General form of the DATAPATH= LIBNAME statement option:

DATAPATH = ('*path1*' '*path2*'... '*pathn*')

UNIX

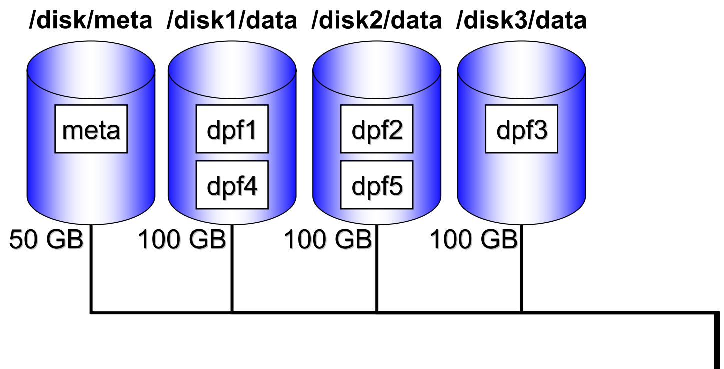
```
libname mylib spde '/disk/meta'
        datapath = ('/disk1/data'
                    '/disk2/data'
                    '/disk3/data');
```

Windows

```
libname mylib spde 'c:\workshop\winsas\prog3\meta'
        datapath = ('c:\workshop\winsas\prog3\data1'
                    'c:\workshop\winsas\prog3\data2'
                    'c:\workshop\winsas\prog3\data3');
```

15

Data Set File Storage



The SPD Engine creates as many partitions as are needed to store all the data. The partitions are created in the paths specified using the DATAPATH= option in a round-robin fashion.

16 ...

INDEXPATH= LIBNAME Statement Option

UNIX

```
libname mylib spde '/disk/meta'
        datapath = ('/disk1/data'
                    '/disk2/data'
                    '/disk3/data')
        indexpath = ('/disk4/index'
                     '/disk5/index');
```

Windows

```
libname mylib spde 'c:\workshop\winsas\prog3\meta'
        datapath = ('c:\workshop\winsas\prog3\data1'
                    'c:\workshop\winsas\prog3\data2'
                    'c:\workshop\winsas\prog3\data3')
        indexpath = ('c:\workshop\winsas\prog3\index1'
                     'c:\workshop\winsas\prog3\index2');
```

17

For UNIX:

- The metadata is stored in **'/disk/meta'**.
- The data is stored in **'/disk1/data'**, **'/disk2/data'**, and **'/disk3/data'**.
- The index is stored in **'/disk4/index'** and **'/disk5/index'**.

For Windows:

- The metadata is stored in **'c:\workshop\winsas\prog3\meta'**.
- The data is stored in **'c:\workshop\winsas\prog3\data1'**, **'c:\workshop\winsas\prog3\data2'**, and **'c:\workshop\winsas\prog3\data3'**.
- The index is stored in **'c:\workshop\winsas\prog3\index1'** and **'c:\workshop\winsas\prog3\index2'**.

INDEXPATH= LIBNAME Statement Option

General form of the INDEXPATH= LIBNAME statement option:

```
INDEXPATH = ('path1' 'path2'... 'pathn')
```

The SPD Engine creates the two index component files (HBX and IBX) in the location specified. When there is not enough space, the index component files overflow into the second specified file path.

18

Creating SPD Engine Tables

Base SAS engine data sets must be converted to the SPD Engine format in order for the SPD Engine to access them.

You can convert the Base SAS engine data sets easily using the following:

- COPY procedure
- APPEND procedure

19

Using the COPY Procedure

UNIX

```
libname ia '.';

libname mylib spde '/disk/meta'
        datapath = ('/disk1/data'
                    '/disk2/data'
                    '/disk3/data')
        indexpath = ('/disk4/index'
                     '/disk5/index');

proc copy in = ia out = mylib;
   select sales international revenue;
run;
```

20

c09s2d1_unix

✎ The data sets **ia.sales**, **ia.international**, and **ia.revenue** are used as examples. They are too small to partition well. The data set **ia.sales** used for demonstrations and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

Using the COPY Procedure

Windows

```
libname ia '.';

libname mylib spde 'c:\workshop\winsas\prog3\meta'
        datapath = ('c:\workshop\winsas\prog3\data1'
                    'c:\workshop\winsas\prog3\data2'
                    'c:\workshop\winsas\prog3\data3')
        indexpath = ('c:\workshop\winsas\prog3\index1'
                     'c:\workshop\winsas\prog3\index2');

proc copy in = ia out = mylib;
   select sales international revenue;
run;
```

21

c09s2d1_win

The data sets **ia.sales**, **ia.international**, and **ia.revenue** are used as examples. They are too small to partition well. The data set **ia.sales** used for demos and exercises contains fewer observations than the data set **ia.sales** used for the course notes.

SPD Engine Component Files

Data Set Component Files (UNIX)

sales.dpf._disk_meta.0.1.spds9	data file partition #1
sales.dpf._disk_meta.1.1.spds9	data file partition #2
sales.dpf._disk_meta.2.1.spds9	data file partition #3
sales.dpf._disk_meta.3.1.spds9	data file partition #4
sales.hbxorigin._disk_meta.0.1.spds9	global index for variable **Origin**
sales.idxorigin._disk_meta.0.1.spds9	segmented index for variable **Origin**

All the data and index files are tied back to the location of the metadata files by the 3rd segment of the component file name.

22

SPD Engine Component Files

Data Set Component Files (Windows)

`sales.dpf.c_workshop_winsas_prog3_meta.0.1.spds9`	data file partition #1
`sales.dpf.c_workshop_winsas_prog3_meta.1.1.spds9`	data file partition #2
`sales.dpf.c_workshop_winsas_prog3_meta.2.1.spds9`	data file partition #3
`sales.dpf.c_workshop_winsas_prog3_meta.3.1.spds9`	data file partition #4
`sales.hbxorigin.c_workshop_winsas_prog3_meta.0.1.spds9`	global index for variable `Origin`
`sales.idxorigin.c_workshop_winsas_prog3_meta.0.1.spds9`	segmented index for variable `Origin`

All the data and index files are tied back to the location of the metadata files by the 3rd segment of the component file name.

23

When you create an SPD Engine data set, many component files can result. SPD Engine component files are stored with the following naming conventions:

Metadata files	***filename*.mdf.0.*p#*.*v#*.spds9**
Data files	***filename*.dpf.*fuid*.*p#*.*v#*.spds9**
Index files	***filename*.idx*suffix*.*fuid*.*p#*.*v#*.spds9**
	***filename*.hbx*suffix*.*fuid*.*p#*.*v#*.spds9**

where

filename	is a valid SAS file name.
mdf	identifies the metadata component file.
dpf	identifies the partitioned data component files.
p#	is the partition number.
v#	is the version number.
fuid	is the unique file ID, which is set to the primary (metadata) path.
idx*suffix*	identifies the segmented view of an index, where *suffix* is the name of the index.
hbx*suffix*	identifies the global view of an index, where *suffix* is the name of the index.
spds9	denotes a SAS®9 SPD Engine component file.

Only the ***filename*** portion of the data component names and the ***suffix*** portion of the index component names are user-controllable. SPDE uses these names and the metadata path, partition number, and version number to build the individual file names.

Controlling the Partition Size

- The data partition size should be chosen in a way so that three or four partitions of each data set reside in each data path.
- The number of partitions per data path should not exceed ten.

> Too many partitions cause too many physical files to be opened when the data set is opened. This has a negative impact on operating system resources and on other applications that execute at the same time.

24

Using the PARTSIZE= Data Set Option

You can control the partition size by using the PARTSIZE= data set option. The PARTSIZE= data set option does the following:

- specifies the largest size (in megabytes) that the data component partitions must be
- is fixed when an SPD Engine data set is created
- applies only to the data component files

25

Using the PARTSIZE= Data Set Option

General form for the PARTSIZE= data set option:

PARTSIZE = *n*

Example:

```
libname mylib spde '/disk/meta'
        datapath = ('/disk1/data'
                    '/disk2/data'
                    '/disk3/data');

data mylib.data (partsize = 512);
   data-step syntax;
run;
```

26 c09s2d2

n is the size of the partition in megabytes. The default is 128. The maximum value is 2047.

Using the PARTSIZE= Data Set Option

To determine an adequate partition size for a new SPD Engine data set, you should be aware of the following:

- the types of applications that will run against the data
- how much data you have
- how many CPUs will be available to the applications
- which disks are available for storing the partitions
- the relationship of these disks to the CPUs

27

See the SPD Engine documentation for additional information on setting an adequate value for the PARTSIZE= data set option.

Creating SPD Engine Indexes

You can create indexes on your SPD Engine data in parallel (asynchronously). To enable asynchronous index creation, use the ASYNCINDEX= data set option.

Use this option with the following:

- the DATA step INDEX= option
- the PROC DATASETS INDEX CREATE statement
- on the PROC APPEND statement when you create an SPD Engine data set from a Base SAS engine data set that has an index

Each method enables all of the declared indexes to be populated from a single scan of the data set.

28

Using the ASYNCINDEX= Data Set Option

General form of the ASYNCINDEX= data set option:

ASYNCINDEX = NO | YES

```
proc append base = mylib.sales(asyncindex = yes)
            data = ia.sales;
run;
```

29 c09s2d3

The SPD Engine spawns a single thread for each index created, and then processes the threads simultaneously. Although creating indexes in parallel is much faster than creating one index at a time, the default for this option is NO because parallel creation requires additional utility work space and additional memory, which might not be available. If the index creation fails due to insufficient resources, set the system option to MEMSIZE=0 or increase the size of the utility file space using the SPDEUTILLOC= system option.

See the SPDE documentation in the SAS OnlineDoc for information about the SPDEUTILLOC= system option.

Creating Indexes Asynchronously

The DATASETS procedure has the flexibility to enable batched parallel index creation by using multiple MODIFY groups. Instead of creating all of the indexes at once, you can create the indexes in groups.

```
proc datasets lib = mylib;
   modify International(asyncindex = yes);
      index create FltDate=(FlightID FltDate);
      index create Origin;
   run;
   modify Revenue(asyncindex = yes);
      index create Origin Dest;
   run;
quit;
```

30

c09s2d4

9.3 Using the SPD Engine Efficiently

Objectives

- Investigate the efficiencies of the SPD Engine.

32

Efficiently Processing Data

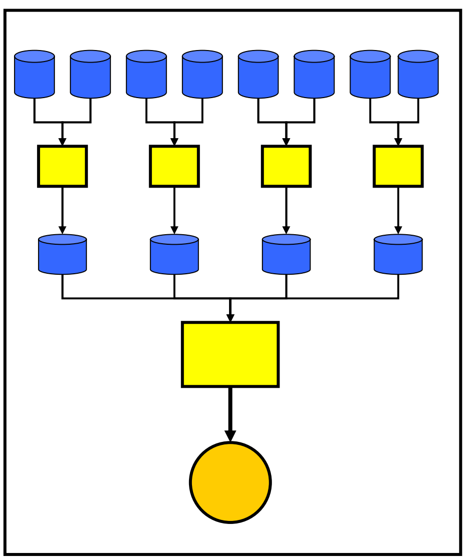
Partition Data Files

Where List Filter

Partial Sort Files

Sort Collation Process

Final Result



33

Using BY-Group Processing

When sort order is relevant, eliminating the SORT procedure and using the BY statement in the PROC step eliminates extra data transfer.

```
proc print data = mylib.sales;
   by RouteID;
   where Dest = 'ANC';
   var FlightID FltDate Dest;
run;
```

When you use a BY statement, the SPD Engine automatically sorts the data without affecting the permanent data set or producing a new output data set.

34

c09s3d1

Using BY-Group Processing

SPD Engine performs the following tasks:

- attempts to use an index for BY-Group processing
- looks for an index that has variables in the order specified by the BY statement
- reads the keys in order from the index and return the rows based on the index

35

Using BY-Group Processing

If the data is in random order and no suitable index exists, SPD engine uses a parallel table scan sort that keeps the rows intact with the sort key.

The time required to access the data in sorted order through a parallel table scan can be more than the time to sort the rows with the SORT procedure.

36

You can suppress the use of indexes for BY-group processing by using the SPDSNIDX=YES macro variable or the NOINDEX = YES data set option.

All SPD Engine macro variables values of NO|YES must be typed in **uppercase**.

Using BY-Group Processing

If several DATA or PROC steps are going to process the same data set using the same BY statement, precede those steps with a PROC SORT that includes WHERE= and/or KEEP= data set options to accomplish the following:

- do the sort once
- minimize the size of the sorted data
- consume fewer resources

The SPD Engine's automatic sorting is good when only a single pass through the data is expected.

37

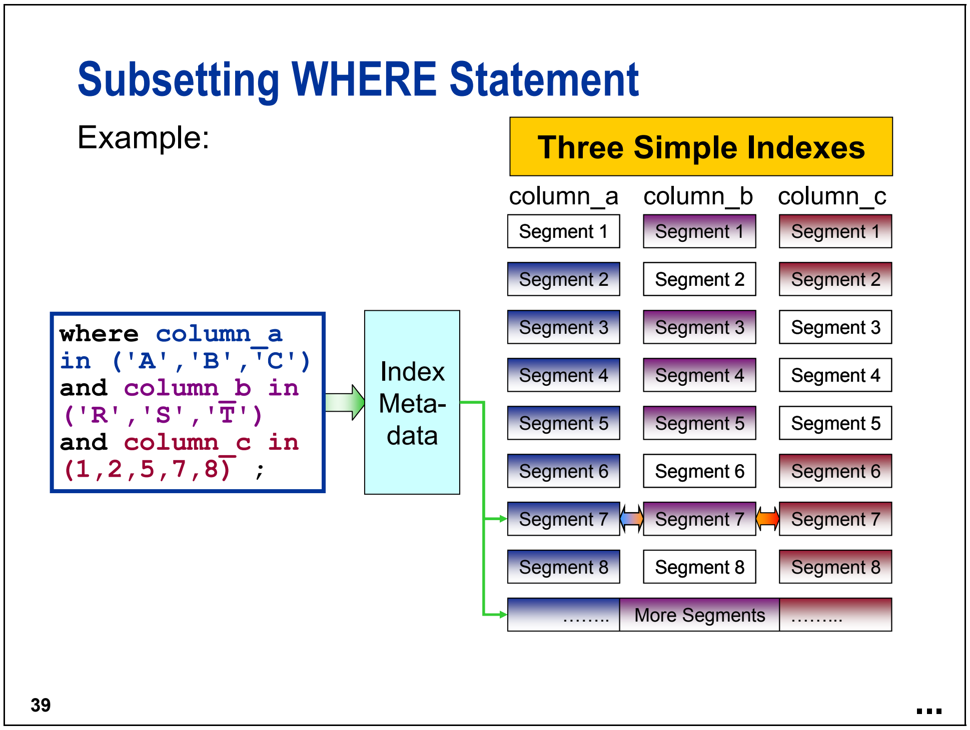
Using a WHERE Statement

- The SPD Engine automatically determines the optimal process to use to evaluate observations for qualifying criteria specified in a WHERE statement.
- WHERE statement efficiency depends on such factors as whether the variables in the expression are indexed.

38

These efficiencies apply to both WHERE statements and WHERE= data set options.

The WHERE evaluation planner included in the SPD Engine chooses the best method to use to evaluate WHERE expressions that use indexes.



The SPD Engine can return some query results without reading the data. An example of such a query is shown below:

```
proc sql;
   select origin, count(*)
      from mylib.sales
         group by origin;
quit;
```

The SPD Engine checks the HBX index component to locate the distinct values of origin. It then goes to the IDX index component to count the rows for each value of origin. The actual **mylib.sales** data set never has to be opened; only the index files for the **mylib.sales** data set are opened.

The Base SAS Engine would need to read the entire **mylib.sales** data set to find the count for each value of origin.

9.4 SPD Engine LIBNAME Statement Options List

Reference Information

BYSORT= specifies for the SPD Engine to perform an automatic implicit sort when it encounters a BY statement.

DATAPATH= specifies a list of paths in which to store data partitions (.dpf) for an SPD Engine data set.

ENDOBS= specifies the end observation number in a user-defined range of observations to be processed.

INDEXPATH= specifies a path or list of paths in which to store the two index component files (.hbx and .idx) associated with an SPD Engine data set.

METAPATH= specifies a list of overflow paths to store metadata (.mdf) component files for an SPD Engine data set.

PARTSIZE= specifies, when an SPD Engine data set is created, the size (in megabytes) that the data component partitions must be. This is a fixed-length size. This specification applies only to partitions in the data component files.

STARTOBS= specifies the starting observation number in a user-defined range of observations to be processed.

TEMP= specifies to store the library in a temporary subdirectory of the primary directory.

Chapter 10 Additional Topics (Self-Study)

10.1 Modifying SAS Data Sets in Place

Objectives

- Use the MODIFY statement in a DATA step to update a data set in place.
- Use a transaction data set to make modifications to a SAS data set.
- Use the KEY= option with the MODIFY statement to make modifications to a SAS data set.

3

Business Task

International Airlines decided to give passengers more leg room, so they want to decrease the number of seats for business and economy classes.

First Class	Capacity Business	Capacity Economy
14	27	154

```
data ia.capacity;
   set ia.capacity;
  *or modify ia.capacity;
   CapEcon = int(CapEcon * .95);
   CapBusiness = int(CapBusiness * .90);
run;
```

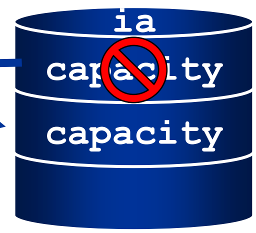
c10s1d1

4 ...

Using the SET Statement

```
data ia.capacity;
   set ia.capacity;
   CapEcon = int(CapEcon * .95);
   CapBusiness = int(CapBusiness * .90);
run;
```

Implied Output



The SET statement requires enough space in the data library for two copies of the data set. When the DATA step is complete, the original copy of the data is removed from the data library.

5 ...

Updating a Data Set in Place

If every observation in a SAS data set requires the same modification, you can specify the modification using an assignment statement.

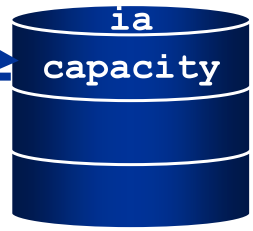
DATA *SAS-data-set*;
    MODIFY *SAS-data-set*;
    existing-variable = *expression*;
RUN;

6

✎ The name of the data set on the DATA and MODIFY statements must match.

Using the MODIFY Statement

```
data ia.capacity;
   modify ia.capacity;
   CapEcon = int(CapEcon * .95);
   CapBusiness = int(CapBusiness * .90);
run;
```



Implied Replace

Additional storage space is not required with the MODIFY statement.

7 ...

The name of the data set on the DATA and MODIFY statements must match.

Using the MODIFY Statement

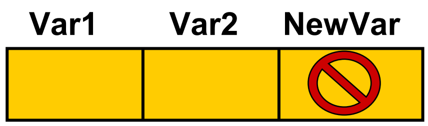
Using the MODIFY statement, you can modify the following:

- every observation in a data set
- observations using a transaction data set and a BY statement
- observations located using an index

8

How MODIFY Affects DATA Step Processing

During compilation, new variables can be added to the PDV, but are not written to the SAS data set.



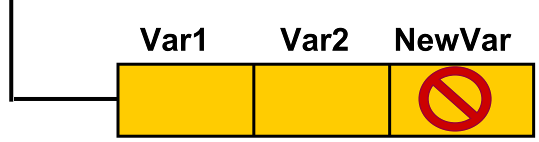
continued...

9

How MODIFY Affects DATA Step Processing

- When a MODIFY statement is used in a DATA step without an OUTPUT, REPLACE, or REMOVE statement, an implied REPLACE statement is executed at the bottom of the DATA step loop.
- This is different from the SET statement that, in the absence of an explicit OUTPUT statement, executes an implied OUTPUT statement at the bottom of the DATA step loop.

IA00100	0000001	RDU	LHR
IA00201	0000002	LHR	RDU
IA00300	0000003	RDU	FRA
IA00400	0000004	FRA	RDU



10

Updating a Data Set in Place

Reduce the number of economy and business seats in the data set **ia.capacity**.

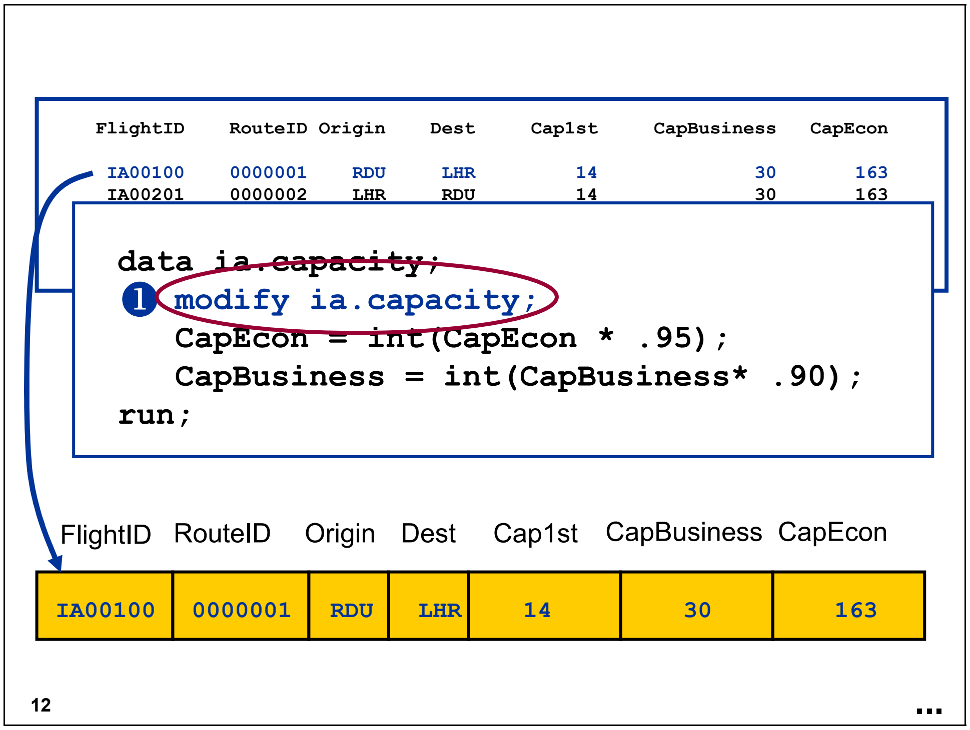
```
data ia.capacity;
   modify ia.capacity;
   CapEcon = int(CapEcon * .95);
   CapBusiness = int(CapBusiness * .90);
run;
```

11

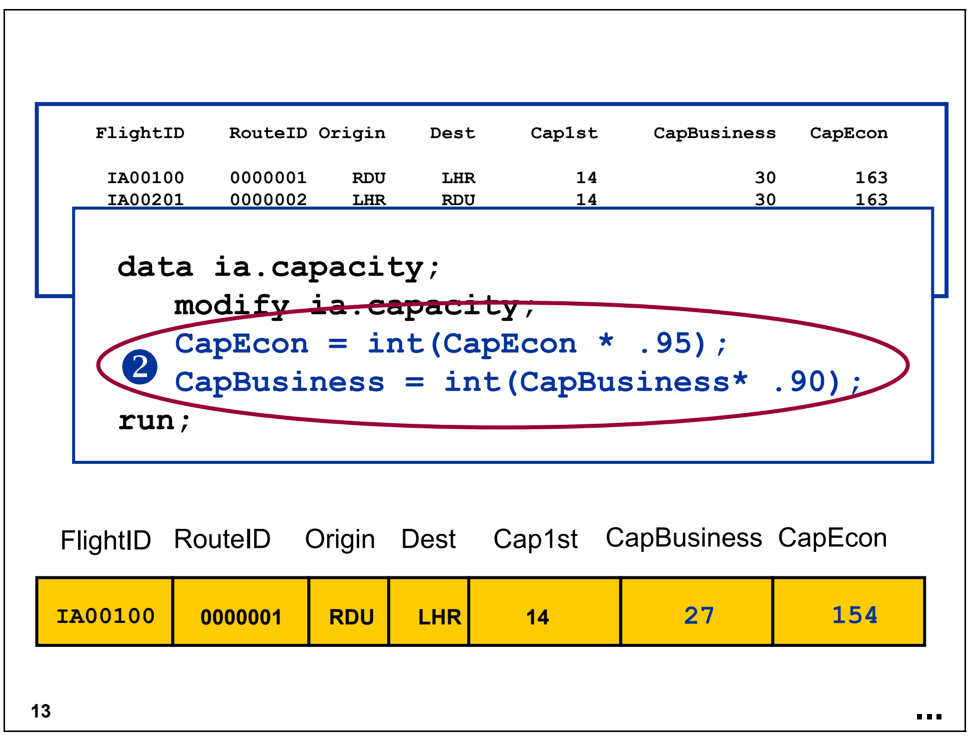
c10s1d1

If the system terminates abnormally while a DATA step that is using the MODIFY statement is processing, you can lose data and possibly damage your master data set. You can recover from the failure by doing the one of the following:

- restoring the master file from a backup and restarting the step
- keeping an audit file and using this file to determine which master observations were updated
- creating generations of SAS data sets



① Reads an observation.



② Updates the PDV using an assignment statement.

FlightID	RouteID	Origin	Dest	Cap1st	CapBusiness	CapEcon
IA00100	0000001	RDU	LHR	14	27	154
IA00201	0000002	LHR	RDU	14	30	163

```
data ia.capacity;
   modify ia.capacity;
   CapEcon = int(CapEcon * .95);
   CapBusiness = int(CapBusiness* .90);
run;
```

③ Implied Replace

FlightID	RouteID	Origin	Dest	Cap1st	CapBusiness	CapEcon
IA00100	0000001	RDU	LHR	14	27	154

14 ...

③ Rewrites the updated observation (same location).

Using the MODIFY Statement

You can use the MODIFY statement to modify observations by applying changes from a transaction data set.

Some of the route ID numbers changed. The changes are stored in a SAS data set.

ia.newrtnum

FlightID	RouteID	Origin	Dest
IA00500	0000035	RDU	JFK
IA02000	0000080	BOS	RDU
IA03500	0000045	RDU	BNA
IA05000	0000120	BRU	LHR
IA06700	0000067	LHR	PRG

15

Using a Transaction Data Set to Update

You need to apply these changes to the data set, **`ia.capacity`**.

`ia.capacity`

Obs	Flight ID	RouteID	Origin	Dest	Cap1st	Cap Business	CapEcon
1	IA00100	0000001	RDU	LHR	14	27	154
2	IA00201	0000002	LHR	RDU	14	27	154
3	IA00300	0000003	RDU	FRA	14	27	154
4	IA00400	0000004	FRA	RDU	14	27	154
5	IA00500	0000005	RDU	JFK	16	.	238
6	IA00600	0000006	JFK	RDU	16	.	238

16

Using a Transaction Data Set to Update

The MODIFY statement is used with a BY statement to apply updates to a master data set from a transaction data set.

```
data ia.capacity;
   modify ia.capacity
          ia.newrtnum;
   by FlightID;
run;
```

DATA *SAS-data-set*;
    MODIFY *SAS-data-set*
        transaction data set;
    BY *key-variable*;
RUN;

17

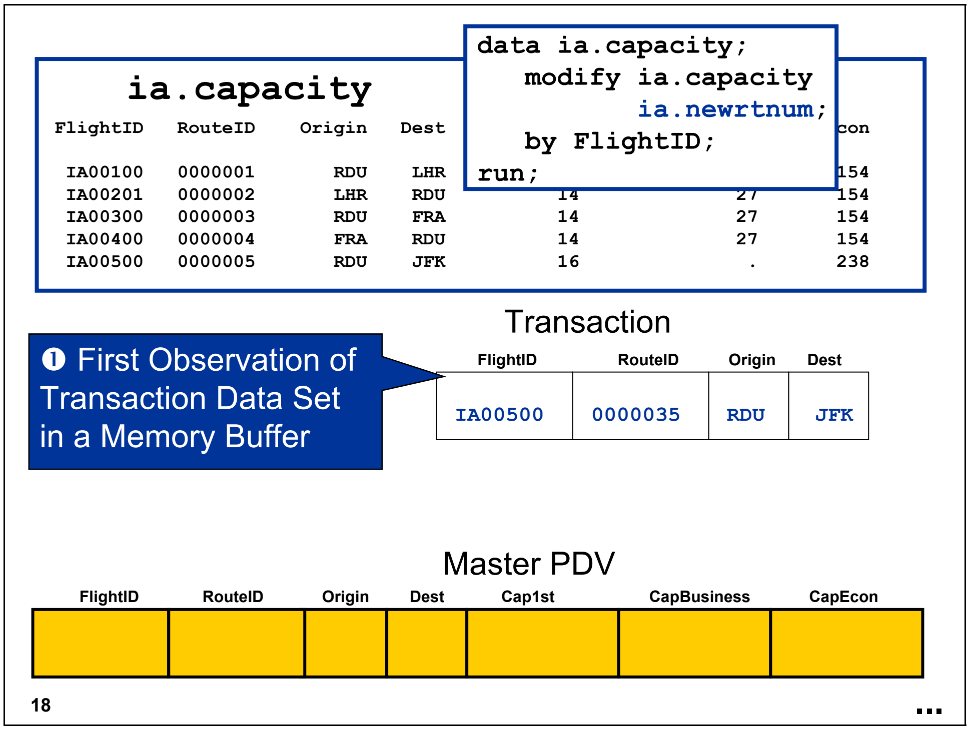
c10s1d2

When you use the MODIFY statement to update a data set, the following conditions might occur:

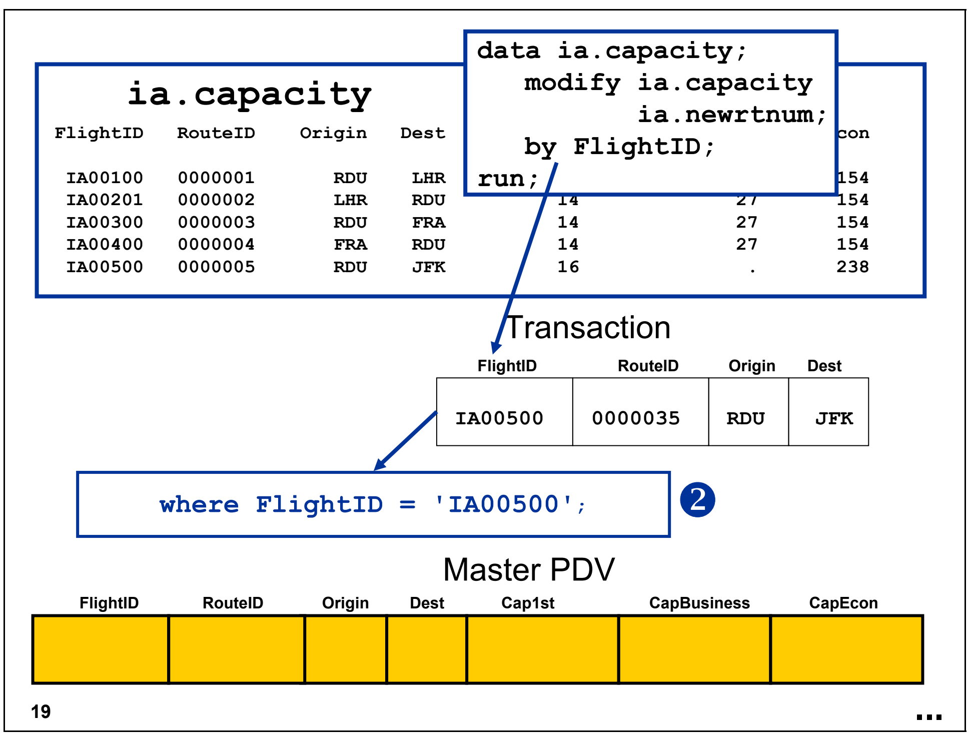
- If a variable has a missing value in the transaction data set, the corresponding master value is not changed by default.
- If duplicate values of the BY variable exist in the master data set, only the first observation of the group is updated.
- If multiple transactions exist for one master observation, all transactions are applied in order.

The MODIFY statement locates the matching observation in the master data set by using dynamic WHERE processing.

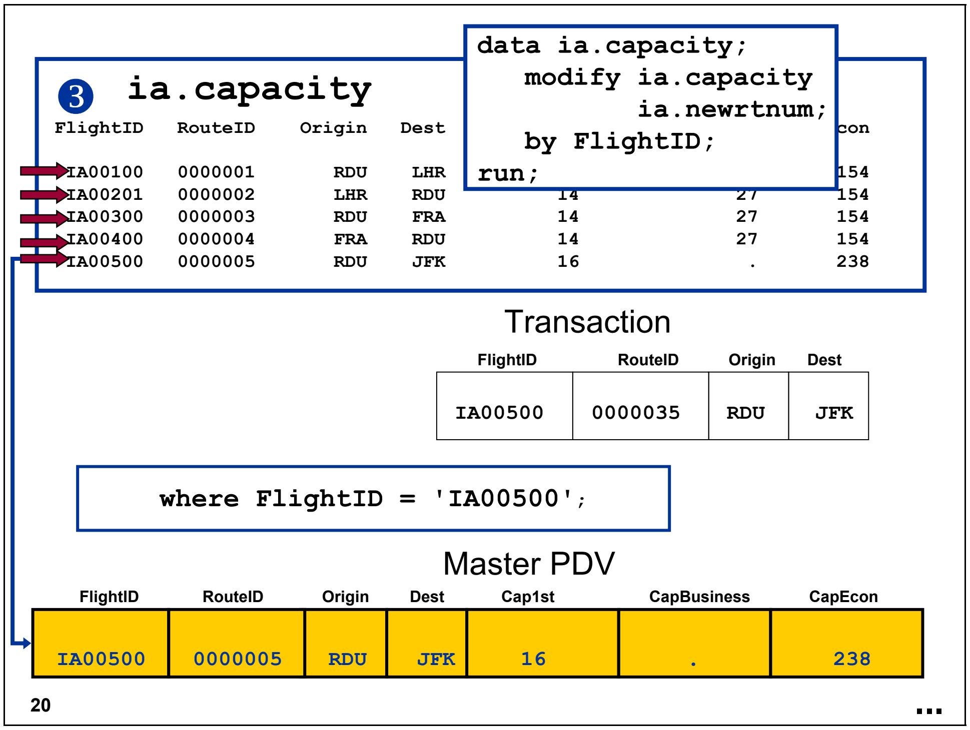
Neither data set requires sorting.



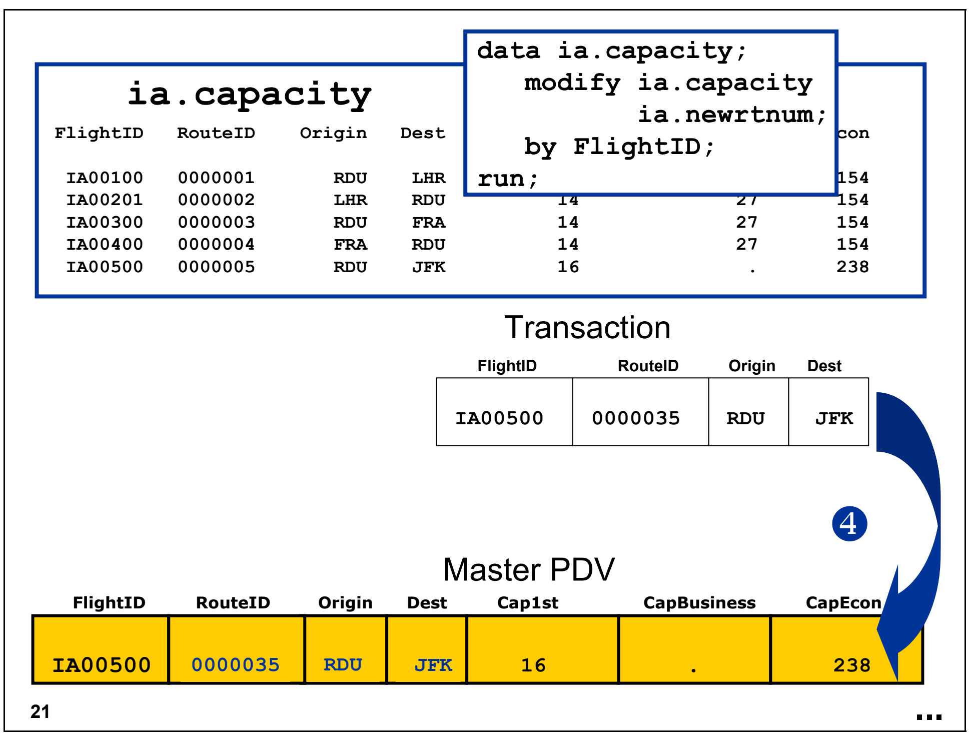
① Reads the transaction observation into a memory buffer.



② Builds a dynamic WHERE statement.



③ Applies a dynamic WHERE statement to the master data set. Reads an observation from the master data set into the PDV.



④ Overlays common variables in the PDV.

```
data ia.capacity;
   modify ia.capacity
          ia.newrtnum;
   by FlightID;
run;
```

ia.capacity

FlightID	RouteID	Origin	Dest			con
IA00100	0000001	RDU	LHR			154
IA00201	0000002	LHR	RDU	14	27	154
IA00300	0000003	RDU	FRA	14	27	154
IA00400	0000004	FRA	RDU	14	27	154
IA00500	0000035	RDU	JFK	16	.	238

❺

Transaction

FlightID	RouteID	Origin	Dest
IA00500	0000035	RDU	JFK

Master PDV

FlightID	RouteID	Origin	Dest	Cap1st	CapBusiness	CapEcon
IA00500	0000035	RDU	JFK	16	.	238

22 ...

⑤ Rewrites the observation back to the master data set in the same location.

Partial Output

```
proc print data = ia.capacity(obs = 5);
   title 'Using a Transaction Data Set for Modifications';
run;
```

Using a Transaction Data Set for Modifications

Obs	FlightID	RouteID	Origin	Dest	Cap1st	CapBusiness	CapEcon
1	IA00100	0000001	RDU	LHR	14	27	154
2	IA00201	0000002	LHR	RDU	14	27	154
3	IA00300	0000003	RDU	FRA	14	27	154
4	IA00400	0000004	FRA	RDU	14	27	154
5	IA00500	0000035	RDU	JFK	16	.	238

23

c10s1d2

Business Task

The cargo figures for 1999 are stored in **`ia.cargo99`**, which has a composite index named **`FlghtDte`** consisting of **`FlightID`** and **`Date`**.

`ia.cargo99`

Flight ID	RouteID	Origin	Dest	CapCargo	Date	Cargo Wgt	CargoRev
IA00100	0000001	RDU	LHR	82400	01JAN1999	45600	$111,720.00
IA00100	0000001	RDU	LHR	82400	01AUG1999	44600	$109,270.00
IA00100	0000001	RDU	LHR	82400	20AUG1999	44600	$109,270.00
IA00100	0000001	RDU	LHR	82400	02SEP1999	47400	$116,130.00
IA00100	0000001	RDU	LHR	82400	29DEC1999	44200	$108,290.00
IA00101	0000001	RDU	LHR	82400	01JAN1999	48000	$117,600.00
IA00101	0000001	RDU	LHR	82400	18MAR1999	45400	$111,230.00

24

Business Task

An accountant discovered that some of the figures are incorrect. You must modify the cargo data to correct the figures. The correct cargo numbers are stored in **`ia.newcgnum`**.

`ia.newcgnum`

Flight ID	RouteID	Origin	Dest	Cap Cargo	Date	Cargo Wgt	CargoRev
IA00101	0000001	RDU	LHR	82400	01JAN1999	.	121879.9
IA01400	0000014	IAD	RDU	35055	07JUL1999	14190	2322.0
IA01503	0000015	RDU	SEA	73530	27AUG1999	35860	58288.8
IA01700	0000017	SEA	SFO	35055	20MAR1999	.	3973.2
IA01704	0000017	SEA	SFO	35055	01MAY1999	11770	5521.2

25

Updating Selected Observations

When you have an indexed data set, you can use the following:

- a SET statement to read a transaction data set
- the MODIFY statement with the KEY= option to locate the observations for updating

26

Updating Selected Observations

```
data ia.cargo99;
   set ia.newcgnum (rename =
                   (CapCargo = newCapCargo
                    CargoWgt = newCargoWgt
                    CargoRev = newCargoRev));
   modify ia.cargo99 key = FlghtDte;
   CapCargo = newCapCargo;
   CargoWgt = newCargoWgt;
   CargoRev = newCargoRev;
run;
```

27

c10s1d3

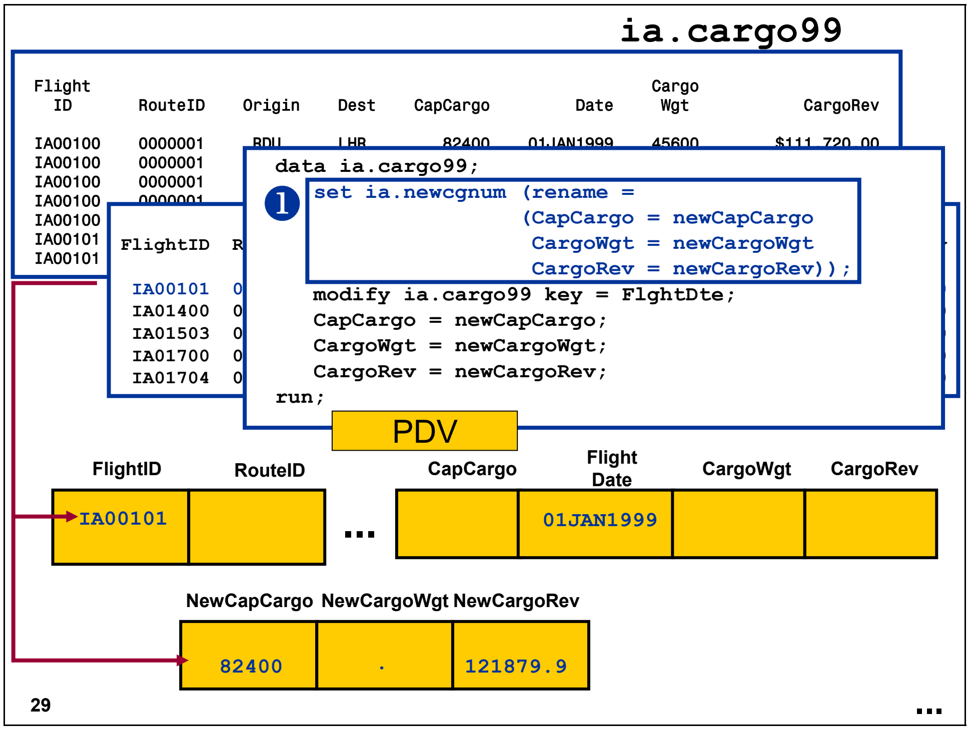
Updating Selected Observations

```
DATA SAS-data-set;
     SET transaction data set;
     MODIFY SAS-data-set
              KEY = key-variable;
     old-variable = new-variable;
RUN;
```

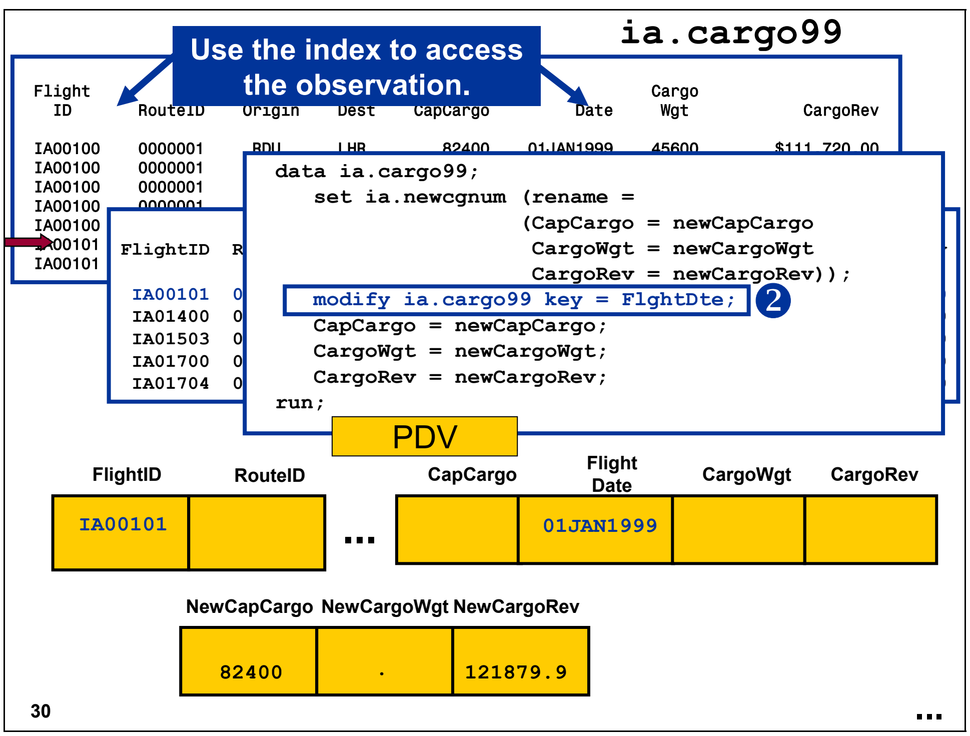
28

When you use an index with the MODIFY statement, these situations occur:

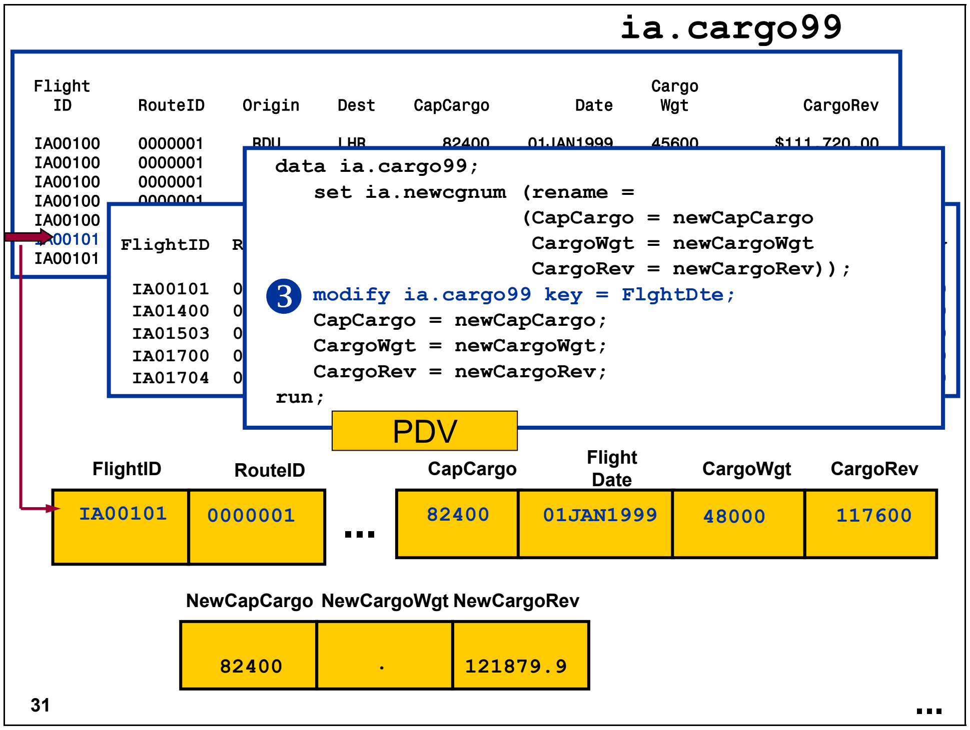
- The index named in the KEY= option can be a simple or composite index.
- You must explicitly specify the update you want to occur. No automatic overlay of nonmissing transaction values occurs as it does with the MODIFY/BY method.
- The data set you are updating must have an index on the key variable. (Data views or sequential libraries, for example, cannot be processed.)
- Each transaction must have a matching observation in the master data set. If you have multiple transactions for one master observation, only the first transaction is applied. The others generate runtime errors and terminate the DATA step (unless you use the UNIQUE option, which is discussed in this section).



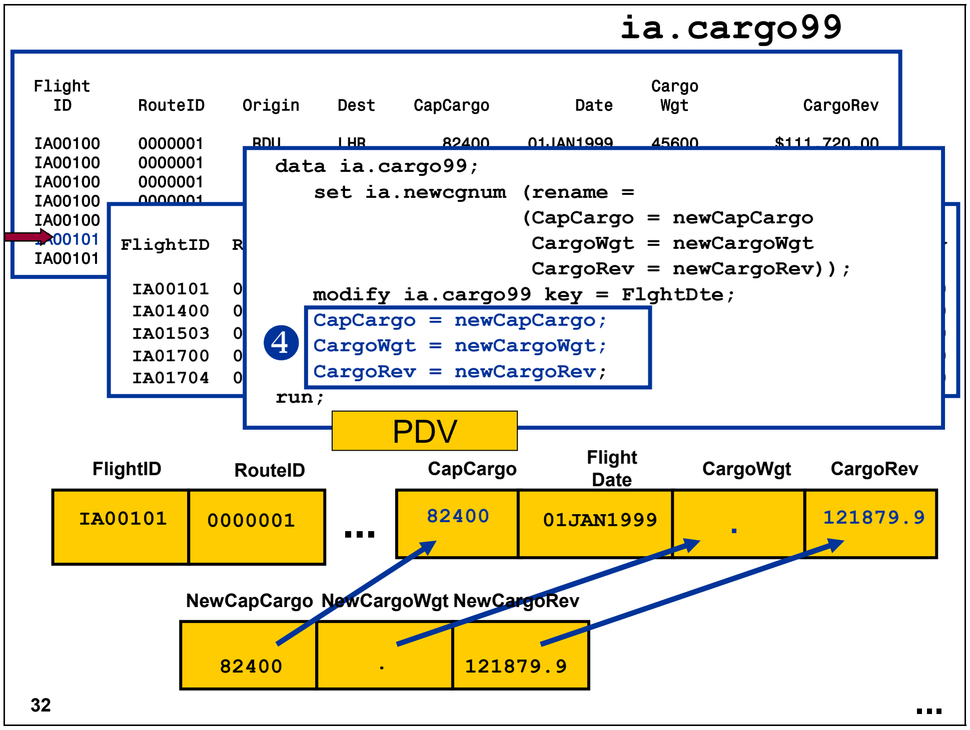
① The SET statement reads an observation from the transaction data set into the PDV.



② The KEY= option uses the **FlghtDte** index to locate an observation in the master data set.

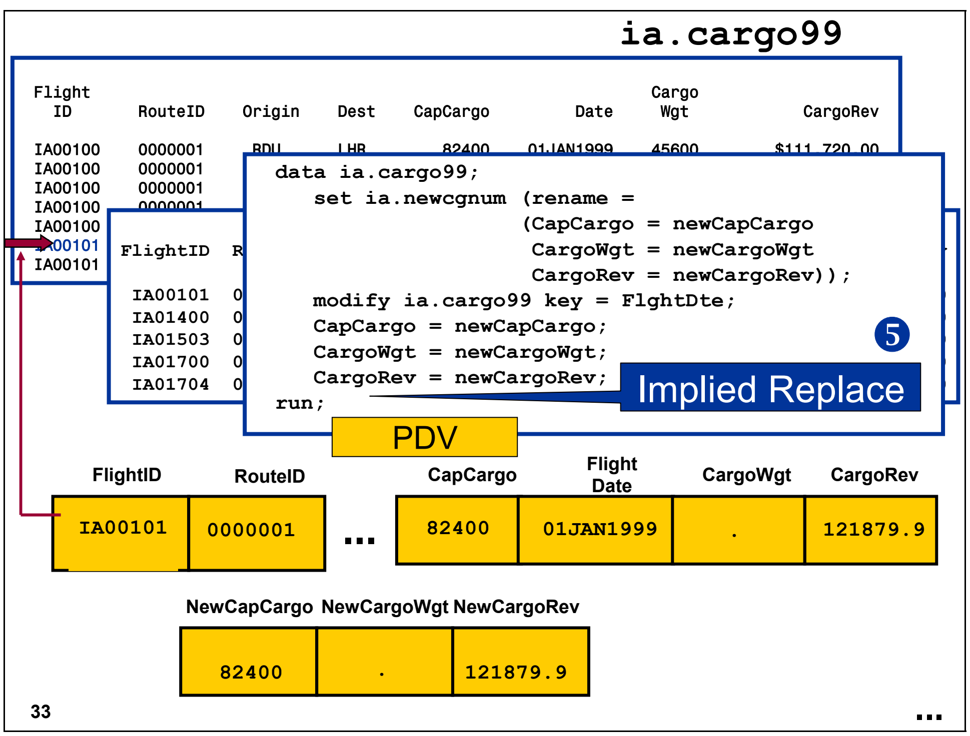


③ The MODIFY statement reads the observation in the master data set using the index and writes values to the PDV.



④ Assignment statements update **CapCargo**, **CargoWgt**, and **CargoRev**.

✎ Because **CargoWgt** was assigned a missing value using an assignment statement, the missing value replaces the original data in the master data set.



⑤ The updated observation is written back to the master data set.

Exercises

1. Setting Up the Files for Exercises

Copy the **ia.empdata** SAS data set into the Work library using PROC COPY:

```
proc copy in = ia out = work;
   select empdata;
run;
```

This is a backup copy of the data in case your program must be submitted multiple times as you test and debug.

2. Modifying All Observations in a SAS Data Set

Give all the employees in the **empdata** SAS data set a 5% salary increase using the MODIFY statement. Print the data before and after the increase.

Partial Output

```
                              Original Data


                                                      Last
    Obs    Division                        HireDate    Name       FirstName

      1    FLIGHT OPERATIONS              11MAR1992    MILLS      DOROTHY E
      2    FINANCE & IT                   19DEC1983    BOWER      EILEEN A.
      3    HUMAN RESOURCES & FACILITIES   12MAR1985    READING    TONY R.
      4    HUMAN RESOURCES & FACILITIES   16OCT1989    JUDD       CAROL A.
      5    AIRPORT OPERATIONS             19DEC1981    WONSILD    HANNA

                                                      Job
    Obs    Country    Location      Phone    EmpID    Code          Salary

      1    USA        CARY          2380     E00001   FLTAT3       $25,000
      2    USA        CARY          1214     E00002   FINCLK       $27,000
      3    USA        CARY          1428     E00003   VICEPR      $120,000
      4    USA        CARY          2061     E00004   FACMNT       $42,000
      5    DENMARK    COPENHAGEN    1086     E00005   GRCREW       $19,000
```

Partial Output

Modified Data

Obs	Division	HireDate	Last Name	FirstName
1	FLIGHT OPERATIONS	11MAR1992	MILLS	DOROTHY E
2	FINANCE & IT	19DEC1983	BOWER	EILEEN A.
3	HUMAN RESOURCES & FACILITIES	12MAR1985	READING	TONY R.
4	HUMAN RESOURCES & FACILITIES	16OCT1989	JUDD	CAROL A.
5	AIRPORT OPERATIONS	19DEC1981	WONSILD	HANNA

Obs	Country	Location	Phone	EmpID	Job Code	Salary
1	USA	CARY	2380	E00001	FLTAT3	$26,250
2	USA	CARY	1214	E00002	FINCLK	$28,350
3	USA	CARY	1428	E00003	VICEPR	$126,000
4	USA	CARY	2061	E00004	FACMNT	$44,100
5	DENMARK	COPENHAGEN	1086	E00005	GRCREW	$19,950

3. Modifying a SAS Data Set with Values in a Transaction Data Set

Use the transaction data set **`ia.empdatu`** to modify the **`empdata`** SAS data set by the employee ID. Do not use an index. Print the **`EmpID`**, **`Phone`**, **`JobCode`**, **`Division`**, and **`Salary`** variables before and after the updates to verify the changes.

Partial Output

Modified Data

Obs	EmpID	Phone	Job Code	Division	Salary
11	E00011	2594	FLTAT3	FLIGHT OPERATIONS	$28,350
12	E00012	2207	MKTCLK	SALES & MARKETING	$34,650
13	E00013	1002	RECEPT	HUMAN RESOURCES & FACILITIES	$23,100
14	E00014	2075	MECH03	FLIGHT OPERATIONS	$20,950
15	E00015	1263	GRCSUP	AIRPORT OPERATIONS	$43,050
16	E00017	2821	RESCLK	HUMAN RESOURCES & FACILITIES	$37,800
17	E00018	1459	FACMNT	HUMAN RESOURCES & FACILITIES	$34,650
18	E00019	1005	SALCLK	SALES & MARKETING	$30,450
19	E00020	1256	FACCLK	HUMAN RESOURCES & FACILITIES	$22,050
20	E00021	1001	ITMGR	FINANCE & IT	$46,150
21	E00022	1255	FACCLK	HUMAN RESOURCES & FACILITIES	$28,350
22	E00023	1172	FLTAT2	FLIGHT OPERATIONS	$32,550
23	E00024	1395	FLTAT3	FLIGHT OPERATIONS	$22,050
24	E00025	1248	BAGCLK	AIRPORT OPERATIONS	$24,150
25	E00026	1516	ITSUPT	FINANCE & IT	$25,200
26	E00027	1215	FINACT	FINANCE & IT	$32,550
27	E00028	0001	ITCLK	FINANCE & IT	$40,900
28	E00029	1325	FLSCHD	AIRPORT OPERATIONS	$17,850

4. Modifying a SAS Data Set Using a Transaction Data Set and an Index

Use the transaction data set **`ia.empdatu2`** to modify the **`empdata`** SAS data set by the employee ID number. Use the index on the **`empdata`** SAS data set. Modify the variables **`LastName`**, **`Location`**, and **`Salary`**. Print the data set before and after the changes.

Partial Output

Modified Data

Obs	EmpID	LastName	Location	Salary
1	E00001	MILLS	CARY	$26,250
2	E00002	SMITH	CARY	$29,350
3	E00003	READING	CARY	$126,000
4	E00004	JUDD	CARY	$44,100
5	E00005	WONSILD	COPENHAGEN	$22,950
6	E00006	ANDERSON	CARY	$32,550
7	E00007	MASSENGILL	CARY	$30,450
8	E00008	BADINE	TORONTO	$89,250
9	E00009	DEMENT	CHICAGO	$36,700
10	E00010	FOSKEY	CARY	$30,450

Reference Information

Missing Values

The MODIFY statement with a BY statement enables you to specify how missing values in the transaction data set are handled by using the UPDATEMODE= option in the MODIFY statement.

MODIFY *SAS-data-set1 SAS-data-set2*
<UPDATEMODE=
MISSINGCHECK |
NOMISSINGCHECK>;
BY *by-expression*;

The default is MISSINGCHECK. When MISSINGCHECK is in effect, SAS checks for missing data in the transaction data set and does not replace the data in the master data set with missing values unless they are special missing values.

NOMISSINGCHECK does not check for missing values in the transaction data set and enables missing values in the transaction data set to replace the values in the master data set. Special missing values in the transaction data set still replace values in the master data set.

Example:

```
modify sasdata.payroll sasdata.update1
         updatemode = nomissingcheck;
```

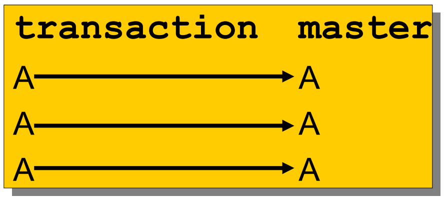
Duplicate Values

If there are duplicates in either MASTER or TRANSACTION:

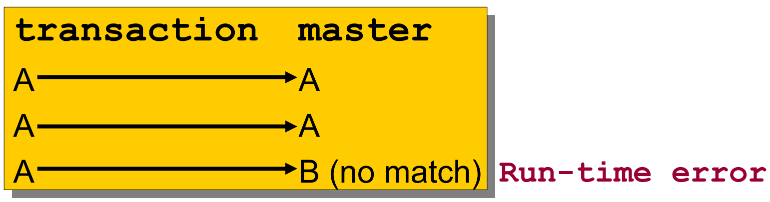
```
data master;
   set transaction;
   modify master key = id;
   x = y;
run;
```

Duplicate Key Values

Example 1: Contiguous duplications in **`transaction`**



Example 2: Contiguous duplications in **`transaction`**



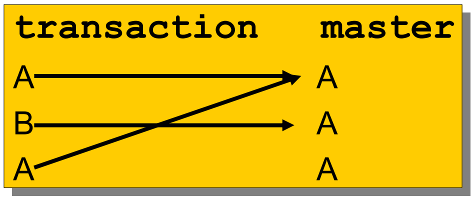
35

EXAMPLE 1: If there are contiguous duplications in **`transaction`**, each of which has a match in **`master`**, then SAS performs a one-to-one update.

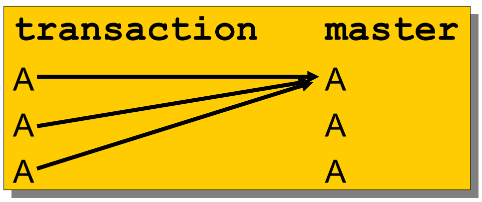
EXAMPLE 2: If there are contiguous duplications in **`transaction`**, some of which do not have a match in **`master`**, then SAS performs a one-to-one update until it finds a non-match. At that time, SAS encounters a run-time error.

Duplicate Key Values

Example 3: Noncontiguous duplications in **`transaction`**



Example 4: Contiguous duplications in **`transaction`** with the UNIQUE option



36

You can specify the UNIQUE argument with the KEY= option in the MODIFY statement to perform the following tasks:

- apply multiple transactions to one master observation
- identify that each observation in the master data set contains a unique value of the index variable(s)

For example:

```
data master;
   set transaction;
   modify master key = id/unique;
   x = y;
run;
```

EXAMPLE 3: If there are noncontiguous duplications in **`transaction`**, then SAS updates the first observation in **`master`**. This is the same action as if the UNIQUE option were used.

EXAMPLE 4: If there are contiguous duplications in **`transaction`** and the UNIQUE option is used, then SAS updates the first observation in **`master`**.

Controlling the Update Process

You can further control processing.

REPLACE specifies that the current observation is rewritten to the master data set. An implied REPLACE statement is added to the end of the DATA step by default if a REPLACE, OUTPUT, or REMOVE statement is not specified.

REMOVE specifies that the current observation is deleted from the master data set.

OUTPUT specifies that the current observation is written to the end of the master data set.

If you use an OUTPUT statement in conjunction with a REMOVE or REPLACE statement, be sure the OUTPUT statement is executed after any REMOVE or REPLACE statements to ensure the integrity of the index position.

If the SAS data set **transaction** has a variable named **code** having values of `'yes'`, `'no'`, and `'new'`, you can submit the following program to accomplish the following:

- delete the rows for the **code** value of `'no'`
- update the rows with the **code** value of `'yes'`
- append the rows for the **code** value of `'new'`

```
data master;
   set transaction;
   modify master key = id;
   a = b;
   if code = 'no' then remove;
   else if code = 'yes' then replace;
   else if code = 'new' then output;
run;
```

If you do not have a variable that indicates how to process the data, you can use the automatic variable **_IORC_**, which is assigned a value after a MODIFY statement KEY= option is executed, indicating abnormal I/O conditions.

An **_IORC_ = 0** indicates that the MODIFY statement was successful, and that the observation was located in the data set.

For example:

```
data master;
   set transaction;
   modify master key = id;
   a = b;
   if _IORC_ = 0 then replace;
   else do;
      output;
      _ERROR_ = 0; /* prevents PDV being printed */
                   /* when there is no match.    */
   end;
run;
```

Monitoring I/O Error Conditions

You can use the automatic variable **_IORC_** with the %SYSRC autocall macro to test for specific I/O error conditions that are created when you use the KEY= option in the MODIFY or SET statements or use the BY statement with the MODIFY statement.

General form for using %SYSRC with _IORC_:

IF _IORC_ = %SYSRC(*mnemonic*) **THEN**…

MNEMONIC	MEANING
_DSENMR	The observation in the transaction data set does not exist in the master data set. Used with the MODIFY statement with a BY statement.
_DSEMTR	Multiple transaction data set observations do not exist in the master data set. Used with the MODIFY statement with a BY statement.
_DSENOM	No matching observation. Used with the KEY= option.
_SOK	The observation was located. _SOK has a value of 0.

To test for error conditions, use the mnemonics above.

The %SYSRC macro is in the AUTOCALL library. You must have the MACRO system option in effect to use this macro. You can view the source code for the %SYSRC macro in sas/core/sasmacro.

For example:

```
data master;
   set transaction;
   modify master key = id;
   select (_IORC_);
      when (%sysrc(_sok)) do;
         a = b;
         replace;
      end;
      when (%sysrc(_dsenom)) do;
         output;
         _ERROR_ = 0;
      end;
      otherwise;
   end;
run;
```

10.2 Creating Generation Data Sets

Objectives

- Introduce the terminology for generation data sets.
- Create generations of a SAS data set.
- Process generations of a SAS data set.

38

Using Generation Data Sets

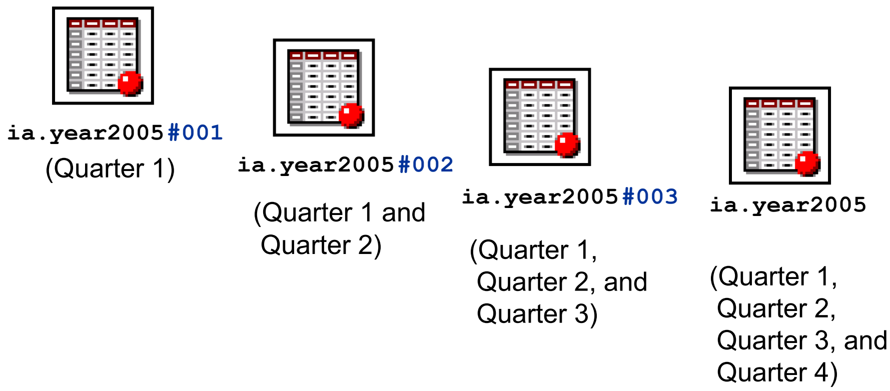
Quarterly, detail cargo revenue values are stored in a SAS data set. At the end of each quarter, the quarterly data set is appended to a year-to-date data set.

Quarter 1 **Quarter 2** **Quarter 3** **Quarter 4**

39

Creating Generation Data Sets

As you append data onto the data set **`ia.year2005`**, the generations of the data are kept.



40 ...

Generation data sets are historical versions of SAS data files.

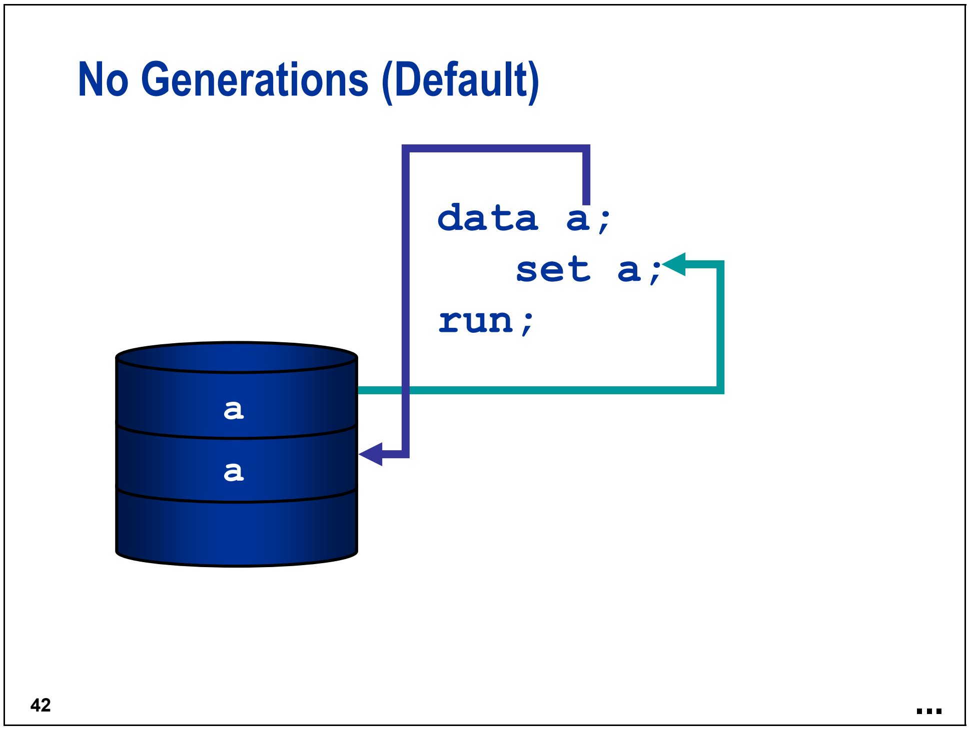
Uses of Generation Data Sets

You can use generation data sets to do the following:

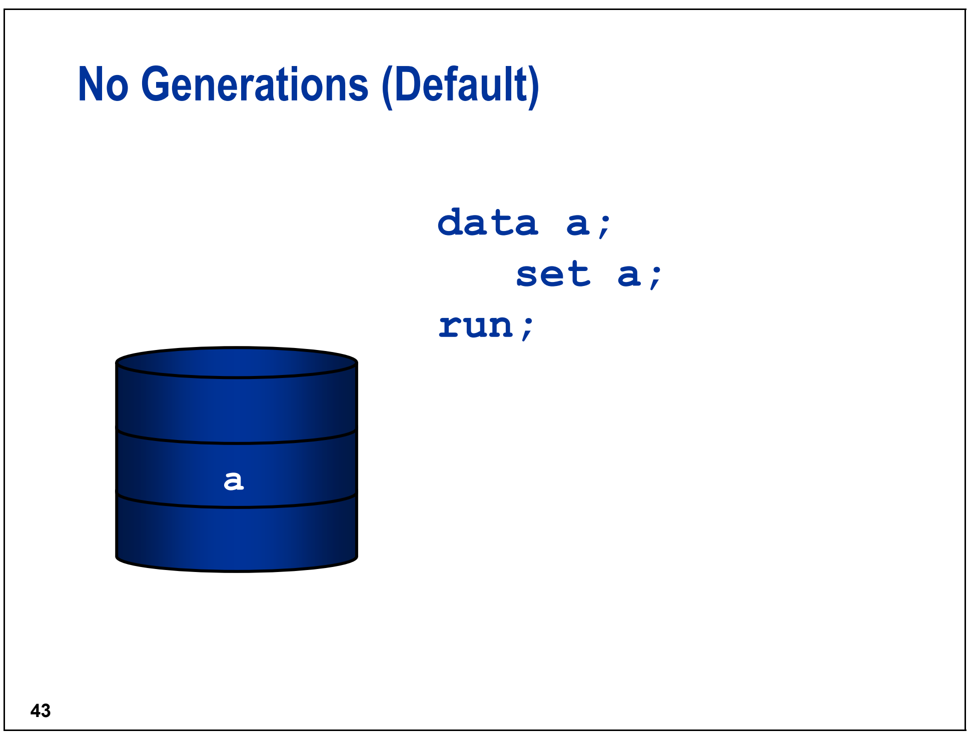
- have multiple copies of either SAS data files or SAS data views
- archive data without having to age the data manually

41

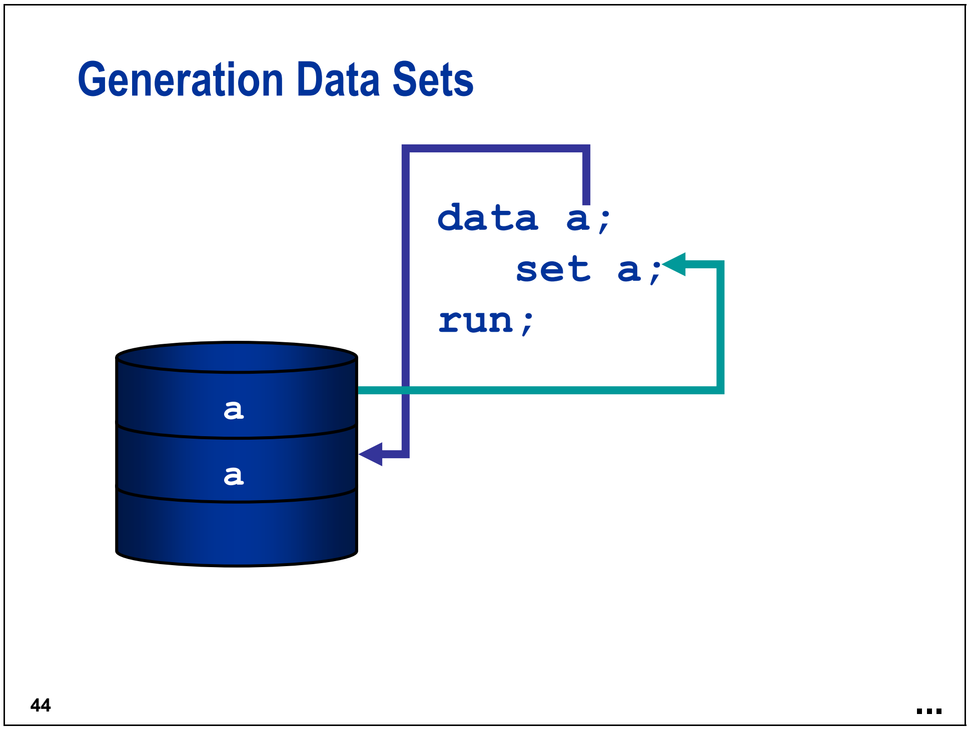
The SAS Scalable Performance Data Engine and OpenVMS do not support generation data sets.



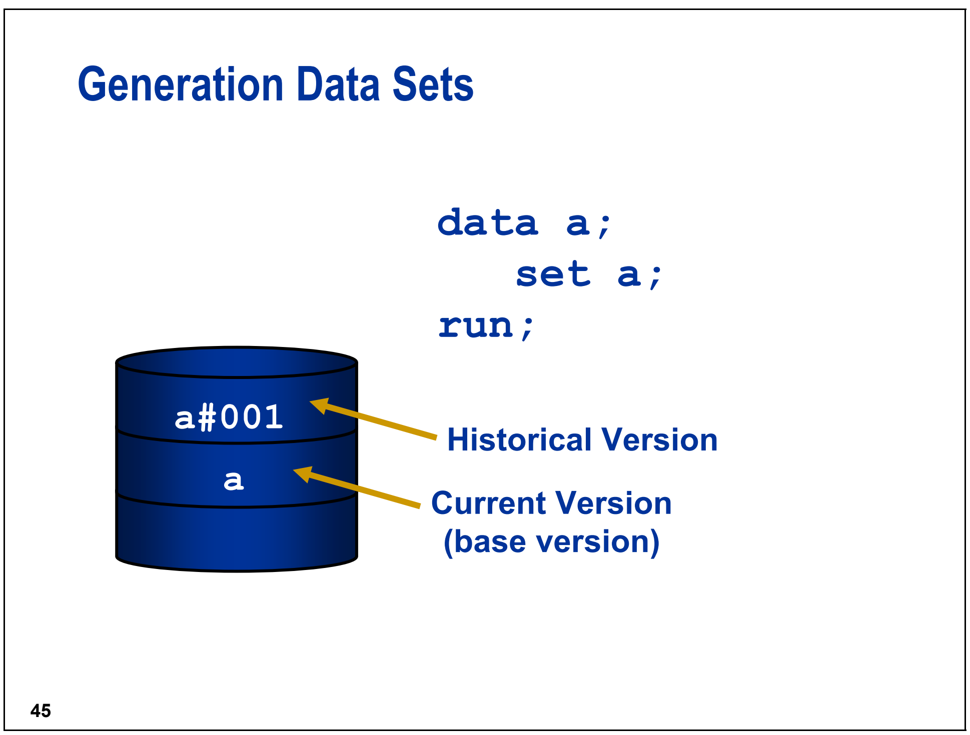
By default, as the SAS data set **a** is replaced, there are two copies of **a** in the SAS data library.



When the DATA step completes execution, SAS removes the original copy of the data set **a** from the data library.



By default, as the SAS data set **a** is replaced, there are two copies of **a** in the SAS data library.



When the DATA step completes execution, SAS keeps the original copy of the SAS data set **a** in the data library and renames it.

✎ New versions are created only when a data set is replaced; not when it is modified in place.

Terms to Know

Generation group

the group of files that represents a series of replacement data sets. The generation group consists of the base version and a set of historical versions of a file.

Version

any one of the files in a generation group

Base version

the most recently created version of a file

continued...

46

Terms to Know

Historical versions

all the versions of a file in the generation group except the base version

Youngest version

the version that is chronologically closest to the base version

Oldest version

the oldest version in a generation group

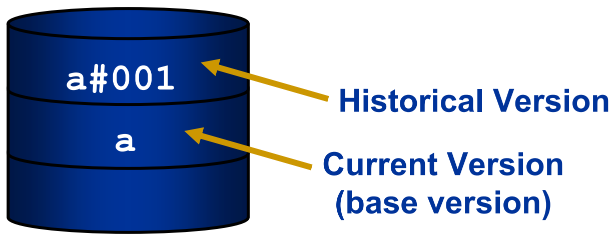
47

When the number of created generations exceeds the value of the GENMAX= option, the oldest versions *age off*. When this happens, the oldest version is not the first version that was created.

Names for Generation Data Sets

When generations are in effect, SAS filenames are limited to 28 characters.

The last four characters are reserved for the version numbers.



48 ...

This internal version number is **not** used in programs.

Documentation of Generation Data Sets

The Explorer window displays the base name followed by all of the historical names.

The CONTENTS and DATASETS procedures include generation information.

49

The **`dictionary.tables`** file does not include information about generation data sets.

Data Set Option to Create Generations

GENMAX=

an output data set option that establishes how many generations to keep.

A GENMAX value

=0 No historical versions are kept. (This is the default.)

>0 how many versions of the file will be kept. For example, GENMAX=2 keeps the base version and one historical version.

50

Example

Create a SAS data set with a maximum of four versions.

```
proc datasets lib = ia  nolist;
   modify year2005 (genmax = 4);
run;
quit;
```

c10s2d1

51

The GENMAX= option can be specified in the same way as a regular data set option.

```
data ia.year2005(genmax = 4);
   data-step-syntax
run;
```

Generation Data Sets: Time 1

Data Set Name	Absolute Generation Number	Relative Generation Number
ia.year2005	1	0

52

...

Creating New Generations

To create new generations, use the following:

- a DATA step with a SET, MERGE, or UPDATE statement
- PROC SORT
- PROC SQL with a CREATE TABLE statement

53

These are all **replacement** techniques; not updating techniques.

Generation Data Sets: Time 2

Replace the data set.

```
data ia.year2005;
   set ia.year2005
       ia.quarter2;
run;
```

54

c10s2d1

Generation Data Sets: Time 2

Data Set Name	Absolute Generation Number	Relative Generation Number
ia.year2005	2	0
ia.year2005 #001	1	-1

55

...

The original data set is renamed as **ia.year2005#001**. The relative generation number is reassigned as –1.

The absolute generation number is a permanent attribute of the data set, stored in the descriptor portion.

Generation Data Sets: Time 3

Replace the data set.

```
data ia.year2005;
   set ia.year2005
       ia.quarter3;
run;
```

c10s2d1

56

Generation Data Sets: Time 3

Data Set Name	Absolute Generation Number	Relative Generation Number
ia.Year2005	3	0
ia.Year2005 #002	2	-1
ia.Year2005#001	1	-2

57 ...

The second version of **`ia.year2005`** is renamed as **`ia.year2005#002`** and is assigned a new relative generation number of –1.

The first version of **`ia.year2005`**, named **`ia.year2005#001`**, is reassigned a relative generation number of –2.

Generation Data Sets: Time 4

Replace the data set.

```
data ia.year2005;
   set ia.year2005
       ia.quarter4;
run;
```

58 c10s2d1

Generation Data Sets: Time 4

Data Set Name	Absolute Generation Number	Relative Generation Number
`ia.year2005`	4	0
`ia.year2005#003`	3	-1
`ia.year2005#002`	2	-2
`ia.year2005#001`	1	-3

59 ...

The third copy of **`ia.year2005`** [**`ia.year2005#003`**] is assigned a relative generation number of –1.

The second copy of **`ia.year2005`** [**`ia.year2005#002`**] is assigned a relative generation number of –2.

The first copy of **`ia.year2005`** [**`ia.year2005#001`**] is reassigned a relative generation number of –3.

Generation Data Sets: Time 5

Sort the data set.

```
proc sort data = ia.year2005;
   by Date;
run;
```

60

c10s2d1

Generation Data Sets: Time 5

Data Set Name	Absolute Generation Number	Relative Generation Number
ia.year2005	5	0
ia.year2005#004	4	-1
ia.year2005#003	3	-2
ia.year2005#002	2	-3
~~ia.year2005#001~~	~~1~~	~~-3~~

Deleted

61

...

The fourth copy of **ia.year2005** [**ia.year2005#004**] is assigned a relative generation number of –1.

The third copy of **ia.year2005** [**ia.year2005#003**] is assigned a relative generation number of –2.

The second copy of **ia.year2005** [**ia.year2005#002**] is assigned a relative generation number of –3.

The first version of **ia.year2005** [**ia.year2005#001**] is deleted.

Generation Data Sets

c10s2d2

```
proc datasets library = ia  nolist;
   title 'All data sets in the ia library';
   contents data = _all_  nods;
   title 'Contents of the Current Version of ia.year2005';
   contents data = year2005;
run;
quit;
```

The NODS option suppresses printing the contents of individual files when you specify _ALL_ in the DATA= option. The CONTENTS statement prints only the SAS data library directory.

Partial Output

```
                    Contents of the Current Version of ia.year2005

                                The DATASETS Procedure

                           Gen  Member       File
                 #  Name       Num  Type         Size  Last Modified

                53  YEAR2005        DATA        25600  19Jan04:17:43:12
                54  YEAR2005     2  DATA        17408  19Jan04:17:43:12
                55  YEAR2005     3  DATA        25600  19Jan04:17:43:12
                56  YEAR2005     4  DATA        25600  19Jan04:17:43:12
```

Partial Output

```
                        Contents of the Current Version of ia.year2005

                                    The DATASETS Procedure

Data Set Name        IA.YEAR2005                      Observations          364
Member Type          DATA                             Variables             7
Engine               V9                               Indexes               0
Created              Monday, January 19,              Observation Length    56
                     2004 05:43:12 PM
Last Modified        Monday, January 19,              Deleted Observations  0
                     2004 05:43:12 PM
Protection                                            Compressed            NO
Data Set Type                                         Sorted                YES
Max Generations      4
Next Generation Num  5
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)

                              Engine/Host Dependent Information

     Data Set Page Size          8192
     Number of Data Set Pages    3
     First Data Page             1
     Max Obs per Page            145
     Obs in First Data Page      113
     Number of Data Set Repairs  0
     File Name                   c:\workshop\winsas\prog3\year2005.sas7bdat
     Release Created             9.0101M0
     Host Created                WIN_PRO

                         Alphabetic List of Variables and Attributes

             #    Variable    Type    Len    Format       Informat

             1    CrgoRev1    Num       8    DOLLAR12.    COMMA12.
             2    CrgoRev2    Num       8    DOLLAR12.    COMMA12.
             3    CrgoRev3    Num       8    DOLLAR12.    COMMA12.
             4    CrgoRev4    Num       8    DOLLAR12.    COMMA12.
             5    CrgoRev5    Num       8    DOLLAR12.    COMMA12.
             6    CrgoRev6    Num       8    DOLLAR12.    COMMA12.
             7    Date        Num       8    DATE9.

                                     Sort Information

                               Sortedby       Date
                               Validated      YES
                               Character Set  ANSI
```

Processing Generation Data Sets

GENNUM =

an input/update data set option that identifies which generation to open

A GENNUM value

>0	absolute reference to a historical version by its generation number
<0	relative reference to historical versions
=0	current version

63

GENNUM= Option

For example,

GENNUM = -1	refers to the youngest version.
GENNUM = 0	refers to the current version.
GENNUM = 1	refers to the first version created.

As new generations are created, the absolute generation number increases sequentially.

As older generations are deleted, the absolute generation numbers are retired.

64

Processing Generation Data Sets

Examples

Print the current version:

```
proc print data = ia.year2005;
run;
```

Print the youngest version:

```
proc print data = ia.year2005(gennum = 4);
run;
```

or

```
proc print data = ia.year2005(gennum = -1);
run;
```

65

Printing Generation Data Sets

c10s2d3

Example 1 – Absolute Reference

```
proc print data = ia.year2005(gennum = 4 obs = 5);
   title 'The Youngest Generation of ia.year2005';
run;
```

Output

```
                         The Youngest Generation of ia.year2005

       Obs        CrgoRev1        CrgoRev2        CrgoRev3        CrgoRev4

         1      $3,280,638        $561,692      $2,128,545      $1,817,984
         2      $3,275,164        $534,184      $1,878,010      $1,860,242
         3      $3,258,884        $552,088      $2,123,491      $1,840,034
         4      $3,330,580        $552,294      $2,357,934      $1,812,278
         5      $3,301,534        $564,340      $2,145,639      $1,819,898

       Obs        CrgoRev5        CrgoRev6          Date

         1        $223,134               .     01JAN2005
         2        $214,236        $969,241     02JAN2005
         3        $213,864        $942,459     03JAN2005
         4        $226,276        $958,295     04JAN2005
         5        $227,258        $982,329     05JAN2005
```

Example 2 – Relative Reference

```
proc print data = ia.year2005(gennum = -1 obs = 5);
   title 'The Youngest Generation of ia.year2005';
run;
```

Output

```
                         The Youngest Generation of ia.year2005

       Obs        CrgoRev1        CrgoRev2        CrgoRev3        CrgoRev4

         1      $3,280,638        $561,692      $2,128,545      $1,817,984
         2      $3,275,164        $534,184      $1,878,010      $1,860,242
         3      $3,258,884        $552,088      $2,123,491      $1,840,034
         4      $3,330,580        $552,294      $2,357,934      $1,812,278
         5      $3,301,534        $564,340      $2,145,639      $1,819,898

       Obs        CrgoRev5        CrgoRev6          Date

         1        $223,134               .     01JAN2005
         2        $214,236        $969,241     02JAN2005
         3        $213,864        $942,459     03JAN2005
         4        $226,276        $958,295     04JAN2005
         5        $227,258        $982,329     05JAN2005
```

Reference Information

Maintenance of Generation Data Sets

You can do the following:

- browse or update an historical version
- transfer generations with PROC COPY
- use PROC DATASETS to perform these tasks:
  - delete all or some of the generations
  - rename an entire generation or any member of the group to a new base name
  - increase or decrease the GENMAX value

You cannot do the following:

- retain the version number when renaming a member
- open an historical version for output

Examples:

To change the number of historical versions (all the generations) created:

```
proc datasets library = ia;
   modify sales (genmax = 10);
run;
```

To rename historical versions (all the generations):

```
proc datasets library = ia;
   change sales = sales2005;
run;
```

To rename only the second historical data set:

```
proc datasets library = ia;
   change sales2005(gennum = 2) = sales2005Q2;
run;
```

To delete one historical version: (This might leave a hole in the generation group.)

```
proc datasets library = ia;
   delete sales2005(gennum = -1);
run;
```

To delete all of the historical versions:

```
proc datasets library = ia;
   delete sales2005(gennum = HIST);
run;
```

✎ HIST is a keyword for the GENNUM= option in the PROC DATASETS DELETE statement that refers to all generations (excludes the base name).

To delete all of the SAS data sets in a generation group:

```
proc datasets library = ia;
   delete sales2005(gennum = ALL);
run;
```

✎ ALL is a keyword for the GENNUM= option in the PROC DATASETS DELETE statement that refers to the base name and all generations.

Exercises

5. Creating Generation Data Sets

Modify the data set **ia.jobhstry** by adding a maximum of three generations.

a. Use the **ia.y200061** and **ia.y200062** data sets to concatenate to **ia.jobhstry** and test your program.

b. Use PROC DATASETS to look at the generation information for **ia.jobhstry**.

Partial Output

```
                                 Directory

              Libref         IA
              Engine         V9
              Physical Name  c:\workshop\winsas\prog3
              File Name      c:\workshop\winsas\prog3

                       Gen  Member      File
        #  Name        Num  Type        Size  Last Modified

        1  ACITIES          DATA       13312  26Nov03:11:34:24
        2  AIRPORTS         DATA     1364992  21Feb01:17:17:52
        3  ALLEMPS          DATA       41984  24Oct01:14:23:44
        4  APORTS           VIEW        5120  21Jan04:13:07:33
        5  CAP2000          DATA      123904  06Apr01:09:54:20
        6  CAPACITY         DATA        9216  27Mar01:12:58:06
        7  CAPINFO          DATA       13312  19Jan04:17:43:20
           CAPINFO          INDEX       9216  19Jan04:17:43:20
        8  CARGO99          DATA      132096  02Nov01:12:17:54
           CARGO99          INDEX     119808  02Nov01:12:17:54
        9  CARGOREV         DATA       37888  26Nov03:10:28:54
       10  COMPETE          DATA        5120  19Sep01:14:14:24
       11  CONTRIB          DATA        9216  09Mar01:12:48:00
       12  CTARGETS         DATA       13312  20Sep01:12:17:12
       13  DNUNDER          DATA       33792  12Mar01:21:38:18
       14  ECONTRIB         DATA        5120  16Mar01:10:48:10
       15  EMPDATA          DATA      115712  19Jan04:17:43:42
           EMPDATA          INDEX      17408  19Jan04:17:43:42
       16  EMPDATU          DATA       17408  17Oct01:12:36:52
       17  EMPDATU2         DATA       17408  12Apr01:18:11:02
       18  EXERFMTS         CATALOG    21504  03Jan02:10:29:06
       19  EXPENSES         DATA       50176  21Feb01:15:27:42
       20  FIRSTQ           VIEW        5120  21Jan04:12:26:37
       21  FLIGHTS          DATA        5120  14Sep01:14:22:48
       22  FLIGHTS2         DATA        5120  26Sep01:13:48:30
       23  FORMATS          CATALOG    21504  21Jan04:13:07:34
       24  JCODEDAT         DATA        9216  07Mar01:09:49:42
       25  JOBHSTRY         DATA        5120  19Jan04:17:43:39
       26  JOBHSTRY      1  DATA        5120  19Jan04:17:43:39
       27  JOBHSTRY      2  DATA        5120  19Jan04:17:43:39
```

Output

```
                              The DATASETS Procedure

Data Set Name        IA.JOBHSTRY                      Observations          40
Member Type          DATA                             Variables             4
Engine               V9                               Indexes               0
Created              Monday, January 19,              Observation Length    45
                     2004 05:43:39 PM
Last Modified        Monday, January 19,              Deleted Observations  0
                     2004 05:43:39 PM
Protection                                            Compressed            NO
Data Set Type                                         Sorted                NO
Max Generations      3
Next Generation Num  3
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)

                           Engine/Host Dependent Information

     Data Set Page Size          4096
     Number of Data Set Pages    1
     First Data Page             1
     Max Obs per Page            90
     Obs in First Data Page      40
     Number of Data Set Repairs  0
     File Name                   c:\workshop\winsas\prog3\jobhstry.sas7bdat
     Release Created             9.0101M0
     Host Created                WIN_PRO

                         Alphabetic List of Variables and Attributes

                          #    Variable    Type    Len

                          2    Job1        Char      6
                          3    Job2        Char      6
                          4    Job3        Char      8
                          1    LastName    Char     25
```

10.3 Creating Integrity Constraints

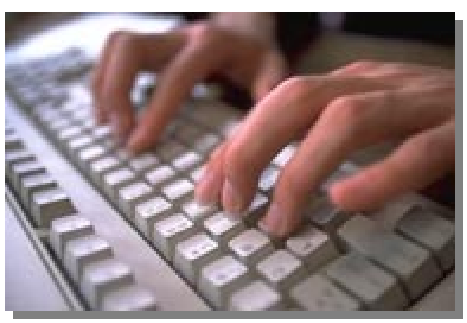
Objectives

- Define integrity constraints.
- Determine the available types of integrity constraints.
- Describe the benefits of integrity constraints.
- Create integrity constraints.

69

Business Task

The data set **`ia.capinfo`** is updated frequently and data errors are prevalent.



70

Integrity Constraints

You can create integrity constraints on the data to accomplish the following:

- preserve the consistency and correctness of data
- validate data when inserting or updating the values of a column for which integrity constraints are defined

71

Integrity constraints are rules that SAS data set modifications must follow to guarantee the validity of data. Integrity constraints apply **only** when data values are modified in place; **not** when the table is replaced.

Techniques for modifying data in place include the following:

- Viewtable window
- FSVIEW window
- FSEDIT window
- DATA step with the MODIFY statement
- PROC SQL with the INSERT INTO, DELETE FROM, or UPDATE statements or the SET statement
- PROC APPEND

Two Categories of Integrity Constraints

General constraints

enable you to restrict the data values accepted for a column.

Referential constraints

enable you to link the data values for a column in one table to the values of columns in another table.

72

Five Integrity Constraints

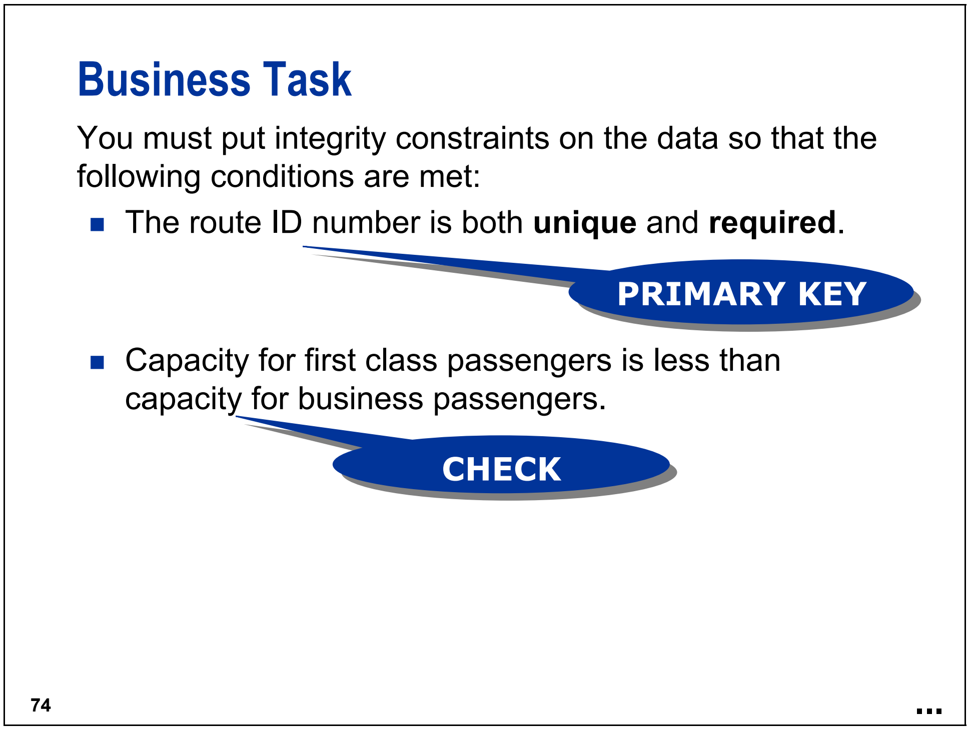
General	Referential
NOT NULL	FOREIGN KEY
CHECK	
UNIQUE	
PRIMARY KEY	

73

You can create integrity constraints for tables containing no rows, one row, or many rows.

NOT NULL	guarantees that corresponding columns have non-missing values in each row.
CHECK	insures that a specific set or range of values is the only value in a column. It can also check the validity of a value in one column based on another value in another column within the same row.
UNIQUE	enforces uniqueness for the value of a column. DISTINCT is an alias for UNIQUE.
PRIMARY KEY	uniquely defines a row within a table. There can be at most one primary key based on one column or a set of columns. The primary key includes the NOT NULL and UNIQUE attributes.
FOREIGN KEY	links one or more rows in a table to a specific row in another table by matching a column or set of columns in one table with the primary key in another table. This parent/child relationship limits modifications made to both primary and foreign keys. The only acceptable values for a foreign key are values of the primary key or missing values.

✎ If the table contains data, all data values are checked to determine whether they satisfy the constraint before the constraint is added.



For the UNIQUE constraint and the PRIMARY KEY constraint, SAS builds unique indexes on the column(s) involved if an appropriate index does not already exist. Any index created by an integrity constraint can be used for other purposes, such as WHERE processing or the KEY= option in a SET statement.

Such an index cannot be removed through ordinary index deletion methods, because it is owned by the constraint.

CHECK Constraint

First Class Capacity must be less than Business Capacity.

Constraint:
Cap1st <
CapBusiness or
CapBusiness = .;

Edit Cap1st for these selected rows.

RouteID	Cap1st	CapBusiness
0000001	18	30
0000005	15	.
0000029	38	30
0000077	19	56

ia.capinfo

75 ...

Methods for Creating Integrity Constraints

- PROC SQL
- PROC DATASETS
- SCL (SAS Component Language) ICCREATE function

76

PROC SQL can assign constraints in the CREATE TABLE and ALTER TABLE statements.

PROC DATASETS can only assign constraints to an existing table.

Creating Integrity Constraints

c10s3d1

```
/* Execute one PROC only. They do the same thing. */

proc datasets lib = ia nolist;
   modify capinfo;
       ic create PKIDInfo = Primary Key (RouteId)
          message = 'You must supply a Route ID Number';
       ic create Class1 = check
                         (where = (Cap1st < CapBusiness or
                                   CapBusiness = .))
message = 'First Class Capacity must be less than Business Capacity';
   contents data = capinfo;
run;
quit;
```

PROC DATASETS uses a WHERE= data set option for the CHECK constraint.

Output

```
                                The DATASETS Procedure

Data Set Name        IA.CAPINFO                        Observations          108
Member Type          DATA                              Variables             7
Engine               V9                                Indexes               1
Created              Thursday, August 03,              Integrity Constraints 2
                     2000 11:37:38 AM
Last Modified        Wed, Jan 21, 2004 05:15:19 PM     Observation Length    48
Protection                                             Deleted Observations  0
Data Set Type                                          Compressed            NO
Label                                                  Sorted                NO
Data Representation  WINDOWS_32
Encoding             Default

                           Engine/Host Dependent Information

     Data Set Page Size          4096
     Number of Data Set Pages    3
     First Data Page             1
     Max Obs per Page            84
     Obs in First Data Page      44
     Index File Page Size        4096
     Number of Index File Pages  2
     Number of Data Set Repairs  0
     File Name                   c:\workshop\winsas\prog3\capinfo.sas7bdat
     Release Created             8.0101M0
     Host Created                WIN_NT
```

(Continued on the next page.)

```
                         Alphabetic List of Variables and Attributes

# Variable   Type Len Format Informat Label

5 Cap1st     Num    8 8.     8.       Aircraft Capacity - First Class Passengers
6 Cap        Num    8 8.     8.       Aircraft Capacity - Business Class Passengers
7 CapEcon    Num    8 8.     8.       Aircraft Capacity - Economy Class Passengers
4 Dest       Char   3                 Dest
1 FlightID   Char   7                 Flight Number
3 Origin     Char   3                 Start Point
2 RouteID    Char   7                 Route Number

                          Alphabetic List of Integrity Constraints

    Integrity                          Where
 #  Constraint  Type         Variables  Clause

 1  Class1      Check                   (Cap1st<CapBusiness) or (CapBusiness=.)
 2  PKIDInfo    Primary Key  RouteID

         User
       #  Message

       1  First Class Capacity must be less than Business Capacity
       2  You must supply a Route ID Number

                         Alphabetic List of Indexes and Attributes

                                                     # of
                                  Unique    Owned    Unique
                  #    Index      Option    by IC    Values

                  1    RouteID    YES       YES         108
```

```
proc sql;
   alter table ia.capinfo
       add constraint PKIDInfo Primary Key (RouteID)
   message = 'You must supply a Route ID Number'
       add constraint Class1 check
                                   (Cap1st < CapBusiness or
                                    CapBusiness = .)
   message = 'First Class Capacity must be less than
              Business Capacity';
   describe table constraints ia.capinfo;
quit;
```

PROC SQL uses a WHERE clause for a CHECK constraint.

Log

```
53   proc sql;
54      alter table capinfo
55          add constraint PKIDInfo Primary Key (RouteID)
66      message = 'You must supply a Route ID Number'
57          add constraint Class1 check
58                                      (Cap1st < CapBusiness or
59                                       CapBusiness = .)
70      message = 'First Class Capacity must be less than Business
70 ! Capacity';
NOTE: Table WORK.CAPINFO has been modified, with 7 columns.
60      describe table constraints capinfo;
NOTE: SQL table WORK.CAPINFO ( bufsize=4096 ) has the following
      integrity constraint(s):


                        -----Alphabetic List of Integrity Constraints-----

                       Integrity                       Where
                     # Constraint Type        Variables Clause
                     _________________________________________________________________
                     1 Class1     Check                 (Cap1st<CapBusiness)
                                                        or (CapBusiness=.)
                     2 PKIDInfo   Primary Key RouteID

                        -----Alphabetic List of Integrity Constraints-----

                       User
                     # Message
                     ____________________________________________________________
                     1 First Class Capacity must be less than Business Capacity
                     2 You must supply a Route ID Number
61   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.55 seconds
      cpu time            0.07 seconds
```

Integrity Constraints and PROC DATASETS

```
PROC DATASETS LIB = libref;
     MODIFY member;
          INTEGRITY CONSTRAINT CREATE
               constraint-name = constraint
               MESSAGE = 'New Error Message';
          INTEGRITY CONSTRAINT DELETE
               constraint-name;
```

78

You can abbreviate INTEGRITY CONSTRAINT as IC.

For additional information about maintaining integrity constraints using PROC DATASETS, see the IC CREATE, IC DELETE, and IC REACTIVATE statements of PROC DATASETS in the Procedures chapter of the Base SAS Procedures Guide in the Base SAS documentation.

PROC SQL and Integrity Constraints

```
PROC SQL;
     ALTER TABLE table-name
     <constraint-clause-1>,…
     <constraint-clause-n>;
```

79

PROC SQL and Integrity Constraints

```
PROC SQL;
     CREATE TABLE table-name
          (column-definition  <column-attribute>,
        <CONSTRAINT constraint-name
                             constraint>);
```

80

See the SAS documentation for additional information about maintaining integrity constraints using PROC SQL.

Documenting Integrity Constraints

General form of the PROC SQL with the DESCRIBE statement:

```
PROC SQL;
     DESCRIBE TABLE CONSTRAINTS table-name;
```

General form of the PROC CONTENTS statement:

```
PROC CONTENTS DATA=libref.dataname;
RUN;
```

81

The DESCRIBE statement in PROC SQL prints the report in the Log window.

Business Task

The data set **ia.cap2000** contains information about every flight in 2000.

You need to ensure that an added route ID number is valid and that it is one of the route ID numbers in the data set **ia.capinfo**.

0000001

0000045

000077

0000145

82

Primary Keys and Foreign Keys

ia.capinfo

(parent table)

Route ID Number
0000001
0000045
0000077
0000112

RouteIDNumber is Primary Key.

Linked

ia.cap2000

(child table)

Route IDNumber	Weight Of Cargo	Revenue Cargo
0000001	45600	111720
0000045	14500	3190
0000077	67500	128250
0000112	55700	181582

RouteIDNumber is Foreign Key.

83

Primary Keys and Foreign Keys

When you use the primary keys and foreign keys, specify the following:

- the primary key on a parent table
- the foreign key on the child tables and identify these items:
  - the name of the parent table
  - what happens when you add data to the child table
  - what happens when you delete data from the parent table

84

Primary Keys and Foreign Keys

If you update or delete an observation in the parent table, you must specify the action you want to take.

UPDATE/ DELETE	Action
RESTRICT	prevents the data values of the primary key variables from being updated or deleted if there is a matching value in one of the foreign key data file's corresponding foreign key variables.
SET NULL	enables the data values of the primary key variables to be updated or deleted, but matching data values in the foreign key data files are changed to null (missing) values.
CASCADE	enables the data values in the primary key variables to be updated, and additionally updates matching data values in the foreign key data files to the same value.

85

ON UPDATE RESTRICT and ON DELETE RESTRICT are the defaults for foreign keys.

Referential constraints are defined in the child tables.

The requirements for establishing a referential relationship are as follows:

- The primary key and foreign key must reference the same number of variables, and the variables must be in the same order.
- The variables must be of the same type (character or numeric) and length.
- If the foreign key is added to a data file that already contains data, the data values in the foreign key data file must match existing values in the primary key data file or be null.

The foreign key data file can exist in the same SAS library as the referenced primary key data file (intra-libref) or in different SAS libraries (inter-libref). However, if the library that contains the foreign key data file is temporary, then the library containing the primary key data file must be temporary as well. In addition, referential integrity constraints cannot be assigned to data files in concatenated libraries.

There is no limit to the number of foreign keys that can reference a primary key. However, additional foreign keys can adversely impact the performance of update and delete operations.

Creating Integrity Constraints

c10s3d2

1. Create the foreign key constraint on the child table.

```
proc sql;
   alter table ia.cap2000
      add Constraint FKRoute Foreign Key (RouteID)
                    references ia.capinfo
                    on update restrict
                    on delete restrict;
quit;
```

2. Add an invalid observation.

```
proc sql;
   insert into ia.cap2000
      set FlightID = 'IA00101',
          RouteID = '0000145',
          Origin = 'RDU',
          Dest = 'LHR',
          Cap1st = 15,
          CapBusiness = 29,
          CapEcon = 200;
quit;
```

Log

```
proc sql;
    insert into ia.cap2000
        set FlightID = 'IA00101',
            RouteID = '0000145',
            Origin = 'RDU',
            Dest = 'LHR',
            Cap1st = 15,
            CapBusiness = 29,
            CapEcon = 200;
ERROR: Observation was not added/updated because a matching primary key value
       was not found for foreign key FKRoute.
NOTE: Deleting the successful inserts before error noted above to restore table
      to a consistent state.
 quit;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used:
      real time           0.33 seconds
      cpu time            0.02 seconds
```

Adding a Row to the Child Table

ia.capinfo
(parent table)

Route ID Number
0000001
0000045
0000077
0000112

ia.cap2000
(child table)

Route IDNumber	Weight Of Cargo	Revenue Cargo
0000001	45600	111720
0000045	14500	3190
0000077	67500	128250
0000112	55700	181582

87 ...

Adding a Row to the Child Table

ia.capinfo
(parent table)

Route ID Number
0000001
0000045
0000077
0000112

?

ia.cap2000
(child table)

Route IDNumber	Weight Of Cargo	Revenue Cargo
0000001	45600	111720
0000045	14500	3190
0000077	67500	128250
0000112	55700	181582
0000145	23987	176000

88 ...

You want to add the route number **0000145** to the child table, **ia.cap2000**. The parent table, **ia.capinfo**, is checked to see if route number **0000145** exists.

Adding a Row to the Child Table

ia.capinfo
(parent table)

Route ID Number
0000001
0000045
0000077
0000112

ia.cap2000
(child table)

Route IDNumber	Weight Of Cargo	Revenue Cargo
0000001	45600	111720
0000045	14500	3190
0000077	67500	128250
0000112	55700	181582

89 ...

If route number **0000145** does not exist in **ia.capinfo**, **0000145** is not added to the data set **ia.cap2000**.

Adding a Row to the Child Table

ia.capinfo
(parent table)

Route ID Number
0000001
0000045
0000077
0000112
0000145

ia.cap2000
(child table)

Route IDNumber	Weight Of Cargo	Revenue Cargo
0000001	45600	111720
0000045	14500	3190
0000077	67500	128250
0000112	55700	181582

90 ...

In order to add **0000145** to the data set **ia.cap2000**, the value **0000145** must first be added to **ia.capinfo.**

Adding a Row to the Child Table

ia.capinfo
(parent table)

Route ID Number
0000001
0000045
0000077
0000112
0000145

ia.cap2000
(child table)

Route IDNumber	Weight Of Cargo	Revenue Cargo
0000001	45600	111720
0000045	14500	3190
0000077	67500	128250
0000112	55700	181582
0000145	**23987**	**176000**

91

After **0000145** is added to **ia.capinfo**, **0000145** can be added to **ia.cap2000**.

Reference Information

To drop a constraint, use the DROP CONSTRAINT clause of the ALTER TABLE statement in PROC SQL or the IC DELETE statement in PROC DATASETS.

c10ref1

```
proc sql;
   alter table ia.cap2000
      drop constraint FKRoute;
   alter table ia.capinfo
      drop constraint PKIDInfo
      drop constraint Class1;
quit;

proc datasets lib = ia;
   modify cap2000;
      ic delete FKRoute;
   modify capinfo;
      ic delete PKIDInfo Class1;
run;
quit;
```

Exercises

6. Creating Integrity Constraints

Create integrity constraints with PROC DATASETS for **`ia.empdata`**.

- Place a primary key on the variable **`EmpID`** and add a custom message.
- Do not allow missing values for the **`LastName`** variable and add a custom message.
- Use PROC FSEDIT or Viewtable to test the constraints.

(Hint for Viewtable: Select **Edit Mode** on the View pull-down menu.)

7. Creating a Foreign Key

Create a foreign key on the data set **`ia.pilots`** on the variable **`EmpID`** using PROC SQL. The parent table is **`ia.empdata`**.

- Restrict the update and deletion of the **`EmpID`** value.
- Test the constraints by trying to add the employee number E01724 to the **`ia.pilots`** data set using the PROC SQL INSERT statement.

10.4 Creating and Using Audit Trails

Objectives

- Determine what an audit trail file is.
- Examine the columns in an audit trail file.
- Initiate an audit trail file.
- Add values to the audit trail file.
- Report on an audit trail file.
- Manage an audit trail file.

94

Business Task

You must monitor the updates for the data set **`ia.capinfo`**.

Creating an audit trail file enables you to document the following:

Who?

What?

When?

95

Audit Trail

The audit trail is an optional SAS file that logs modifications to a SAS table.

For each addition, deletion, and update to the data, the audit file stores information about the following:

- who made the modification
- what was modified
- when the modification was made

96

The MODIFY statement is one method with which to modify a SAS table. When a MODIFY statement is used, integrity constraints are checked and edits are recorded in an audit trail.

The Audit Trail File

The *audit trail file* is defined as follows:

- a SAS file with the same name as the data file it is monitoring, but with a member type of AUDIT
- created by PROC DATASETS
- read-only
- read by any SAS procedure that accepts the TYPE= data set option

97

- The audit trail file must reside in the same SAS data library as the data file associated with it.
- A SAS table can have, at most, one audit file.
- Procedures such as PRINT, TABULATE, and FREQ can read audit trail files using the TYPE= data set option.

Audit Trail File Variables

The audit trail file can contain three types of columns:

- data file variables
  - copies of the columns in the audited SAS data file
- _AT*_ variables
  - store information about the data modifications
- USER_VAR variables
  - user-defined special columns that enable you to enter information into the audit file

98

For the _AT*_ variables, the asterisk is replaced by a specific string, such as DATETIME.

USER_VAR variables are optional. They supplement the information automatically recorded in the _AT*_ variables.

_AT*_ Variables

_AT*_ Variable	Description
ATDATETIME	Date and time of a modification
ATUSERID	Log-on user ID associated with a modification
ATOBSNO	Observation number affected by the modification unless REUSE=YES
ATRETURNCODE	Event return code
ATMESSAGE	SAS log message at the time of the modification
ATOPCODE	Code describing the type of operation

99

By default, SAS logs all _ATOPTCODE_ codes. You can change this behavior when you initiate an audit trail.

ATOPTCODE Values

Code	Event
DA	Added data record image
DD	Deleted data record image
DR	Before-update record image
DW	After-update record image
EA	Observation add failed
ED	Observation delete failed
EU	Observation update failed

100

An image can be one of the following:

- an edited data value
- an added row
- a deleted row

User Variables

User variables have the following characteristics:

- defined as part of the audit trail specification
- displayed when the associated data file is opened for update
- edited as you would edit data values
- written to the audit trail as each row is saved
- not available when the associated data file is opened for browsing

101

Creating and Viewing an Audit Trail

c10s4d1

```
proc datasets library = ia nolist;
   audit cap2000;
   initiate;
   user_var who $20 label = 'Who made the change'
            why $20 label = 'Why the change was made';
run;
quit;

proc sql;
   insert into ia.cap2000
      set FlightID = 'IA00040',
          RouteID  = '0000100',
          Origin = 'CDG',
          Dest = 'LHR',
          Cap1st = 12,
          CapBusiness = 20,
          CapEcon = 120,
          Date = '03JUN2000'd,
          who = 'Administrator',
          why = 'New Flight';
quit;

proc print data = ia.cap2000 (type = audit);
   title 'Audit Trail for ia.cap2000';
run;

/*  To terminate the audit trail  */

proc datasets library = ia nolist;
   audit cap2000;
   terminate;
run;
quit;
```

- The TERMINATE statement deletes the audit file. Do not delete the audit file using operating system methods because this can damage the SAS data file.
- To stop auditing without deleting the audit file, use the SUSPEND statement.
- To resume auditing after a suspension, use the RESUME statement.

Output

```
                                    Audit Trail for ia.cap2000

           Flight                                      Cap
       Obs   ID      RouteID  Origin  Dest    Cap1st  Business    CapEcon

        1  IA00040  0000100    CDG    LHR        12        20        120

       Obs      Date       who           why             _ATDATETIME_

        1  03JUN2000  Administrator  New Flight   19JAN2004:16:55:39

       Obs _ATOBSNO_ _ATRETURNCODE_ _ATUSERID_ _ATOPCODE_ _ATMESSAGE_

        1     2001          .          saswjr       DA
```

Initiating an Audit Trail

```
proc datasets lib = ia;
   audit cap2000;
   initiate;
   user_var who $20 label = 'Who made the change'
            why $20 label = 'Why the change was made';
   run;
quit;
```

103 c10s4d1

Initiating an Audit Trail with PROC DATASETS

```
PROC DATASETS LIB = libname;
      AUDIT SAS-file <SAS-password>;
      INITIATE;
         <LOG <BEFORE_IMAGE = YES|NO>
                 <DATA_IMAGE = YES|NO>
                 <ERROR_IMAGE = YES|NO>>;
         <USER_VAR specification-1
                         <specification-n>>;
      RUN;
QUIT;
```

104

libname — is the library where the table to be audited resides.

SAS-file — states the name of the table to be audited.

SAS-password — provides the SAS data file password, if one exists.

INITIATE — creates the audit file.

LOG — specifies the images (events) to be logged on the audit file. If you omit the LOG statement, all images are recorded.

BEFORE_IMAGE=YES|NO
: controls storage of before-update record images (for example, the 'DR' operation).

DATA_IMAGE=YES|NO
: controls storage of after-update record images (for example, other operations starting with 'D').

ERROR_IMAGE=YES|NO
: controls storage of unsuccessful update record images (for example, operations starting with 'E').

The audit file uses the SAS password that is assigned to the parent data file; therefore, it is recommended that you alter the password for the parent data file. Use the ALTER= data set option to assign an *alter-password* to a SAS file or to access a read-, write-, or alter-protected SAS file. If another password is used or no password is used, then the audit file is still created, but is not protected.

PROC DATASETS USER_VAR Statement

USER_VAR *variable-name* <*$*><*length*>
<**LABEL** = '*variable-label*'>
<*variable-name-n …*>**;**

105

USER_VAR variables are unique in SAS in that they are stored in one file (for example, the audit file) and opened for update in another (for example, the data file).

When the data file is opened for update, the USER_VAR variables appear, and you can edit them as though they were part of the data file.

Controlling the Audit Trail

After you initiate the audit trail, use PROC DATASETS to do the following:

- suspend logging
- resume logging
- terminate (delete) the audit file

106

Suspending and Resuming Audit Trails

To suspend an audit:

```
proc datasets lib = ia;
   audit cap2000;
      suspend;
   run;
quit;
```

To resume an audit:

```
proc datasets lib = ia;
   audit cap2000;
      resume;
   run;
quit;
```

107

c10s4d2

Terminating an Audit Trail

To terminate and delete an audit trail:

```
proc datasets lib = ia;
   audit cap2000;
      terminate;
   run;
quit;
```

108

c10s4d2

Exercises

8. Creating an Audit Trail

Create an audit trail for the data set **`ia.pilots`**.

- Add user variables to track who edited the data set and why it was edited.
- Use PROC FSEDIT to give a pilot a salary increase. Be sure to include who edited the data set and give a reason for the increase.
- Use PROC PRINT to look at the audit trail.
- Terminate the audit trail.

10.5 Working with Perl Regular Expressions

Objectives

- Describe Perl regular expressions and metacharacters.
- Use pattern matching to validate data.
- Use pattern matching to replace text.

111

Perl regular expressions are new in SAS®9.

Perl Regular Expressions

Perl has the following features:

- is an open source language useful for scripting, reporting, manipulating text, and other general-purpose programming
- uses character patterns for search and replacement
- is documented at www.perldoc.com

112

Perl stands for Practical Extraction and Reporting Language.

Perl Regular Expressions

A Perl regular expression specifies a character pattern to be searched (matched) or replaced (substituted).

Examples:

m/boat/

match the substring *boat.*

s/boat/ship/

substitute the string *ship* for the substring *boat.*

113

The m (*match*) directive is optional.

Perl Regular Expressions

A set of *metacharacters* is used to specify the following:

- wildcard characters
- special characters
- number of matches
- capture buffers

Forward slashes (/) are required to enclose a regular expression.

114

Perl Regular Expressions

Selected Perl metacharacters (symbols):

Symbol	Meaning	Examples	
\	escape character	\d	1 digit
.	any character	a-z, A-Z, 0-9, $%&-+:	
\w	any word character	a-z, A-Z, _, 0-9	
\d	any digit	0-9	
\s	white-space character	space, tab, carriage return	
[ ]	set of characters	[abc]	a, b, or c
*	match 0 or more times	\d*	0 or more digits

continued...

115

Perl Regular Expressions

Selected Perl metacharacters (symbols):

Symbol	Meaning	Examples	
+	match 1 or more times	\d+	1 or more digits
?	match 0 or 1 times	\d?	0 or 1digit
{*n*}	match *n* times	\d{2}	two digits
( )	capture buffer	(\d{2})	store two-digit match in a capture buffer
^	start match at 1st character	/^	start match in position 1
$	end match at last character	$/	end match at last character

116

SAS PRX Functions

Selected SAS PRX functions:

- PRXPARSE parses (compiles) a Perl regular expression and returns an identifier.
- PRXMATCH searches for a substring and returns the position when found.
- PRXCHANGE replaces a substring with another string.

117

The PRXPARSE Function

The PRXPARSE function compiles a Perl regular expression for use in a search or replace operation.

PRXPARSE(*Perl-regular-expression*)

Examples:

```
re=prxparse('m/boat/');

re=prxparse('s/boat/ship/');
```

The variable **re** is written to the output data set.

118

The argument to the PRXPARSE function is a character value or character expression.

The PRXPARSE function returns a numeric identifier representing the parsed expression. This identifier can be used with the following:

- the PRXMATCH function — to search for a pattern match and return the position at which the pattern is found
- the PRXCHANGE function — to perform a pattern-matching replacement
- the PRXPOSN function — to return the value in a capture buffer
- the PRXNEXT routine — to find the next occurrence of the search pattern
- the PRXPAREN function — to return the last capture buffer

If *Perl regular expression* is a constant or if it uses the `/o` option, the Perl regular expression is compiled only once. Successive calls to PRXPARSE do not cause a recompile, but return the identifier that was already compiled. This behavior simplifies the code because you do not need to use an initialization block (**IF _N_ =1**) to initialize Perl regular expressions.

The PRXMATCH Function

The PRXMATCH function uses a Perl regular expression to search for a **pattern** and returns the **starting** position at which the pattern is found.

If the pattern is not found, 0 is returned.

PRXMATCH(*Perl-regular-expression, source*)

Perl-regular-expression

specifies for which a character pattern to search.

source

specifies the string to be searched.

119

The PRXMATCH Function

Find all the names that do not have a valid Social Security number pattern of *ddd-dd-dddd*.

`ia.Staff` (Partial Listing)

```
Name                        SSN

O'REILY, MARY             897-37-4135
PYLES, JANE               42-8321-982
HOFFMAN, VALERIE          171-32-8038
DAWN, JENNIFER
VAN HUSEN, JEFF           801-5A-3640
SIM-SMITH, ANGELA         219-68-2436abc
TIMMONS, DAVID            hello219-68-1098
BENJAMIN, CATHERINE       236-73-7392
```

120

The PRXMATCH Function

```
data Invalidssn;
   retain re;
   set ia.Staff;
   if _n_ = 1 then
      re = prxparse('/\d{3}-\d{2}-\d{4}/');
   if prxmatch(re, ssn) = 0;
run;
proc print data=Invalidssn;
   title 'Invalid Social Security Numbers';
   var Name SSN;
run;
```

121

c10s5d1

Equivalent code:

```
where ssn like '___-__-____' and verify(ssn,'0123456789') = 0;
```

The LIKE operator would select `364-9A-7412` as a valid SSN because it cannot distinguish letters from digits. The VERIFY function validates that the characters were digits.

The roles of the items in the regular expression:

`/`	Start regular expression.
`\d{3}`	Match three digits
`-`	followed by a dash
`\d{2}`	followed by two digits
`-`	followed by a dash
`\d{4}`	followed by four digits.
`/`	End the regular expression.

The PRXMATCH Function

Output

```
          Invalid Social Security Numbers

      Obs     Name                         SSN

       1      PYLES, JANE              42-8321-982
       2      DAWN, JENNIFER
       3      VAN HUSEN, JEFF          801-5A-3640
```

What happened
to Angela and David?

129 c10s5d1

The PRXMATCH Function

The number pattern is *ddd-dd-dddd*.

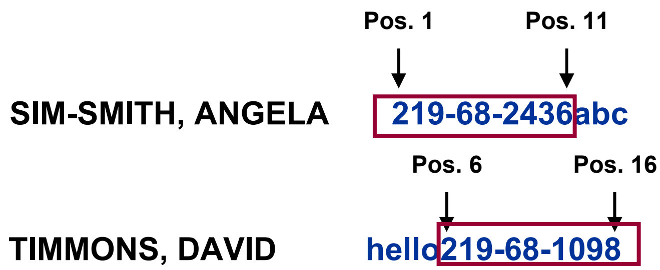
ia.Staff (Partial Listing)

```
     Name                        SSN

     SIM-SMITH, ANGELA        219-68-2436abc
     TIMMONS, DAVID           hello219-68-1098
```

130

The PRXMATCH Function

The PRXMATCH function performs a sliding window search. For character strings longer than the 11 specified characters, invalid strings could be considered a match.



The PRXMATCH Function

Adding the caret (^) and the dollar sign ($) to the PRXPARSE function will start in position 1 for 11:

```
data Invalidssn;
   set ia.Staff;
   re = prxparse('/^\d{3}-\d{2}-\d{4}$/');
   if prxmatch(re, trim(ssn)) = 0;
run;
proc print data=Invalidssn;
   title 'Invalid Social Security Numbers';
   var Name SSN;
run;
```

132 c10s5d2

Be sure to trim the blanks from the end of the SSN variable. In Perl expressions, blanks have significance.

If the Perl regular expression is a constant or if it uses the /o option, then the Perl regular expression is compiled once and each use of PRXMATCH reuses the compiled expression.

If the Perl regular expression is **not** a constant and if it does **not** use the /o option, then the Perl regular expression is recompiled for each call to PRXMATCH.

The compile-once behavior occurs when you use PRXMATCH in a DATA step, in a WHERE clause, or in PROC SQL. For all other uses, the Perl regular expression is recompiled for each call to PRXMATCH.

The PRXMATCH Function

Output

```
          Invalid Social Security Numbers

Obs     Name                        SSN

 1      PYLES, JANE                 42-8321-982
 2      DAWN, JENNIFER
 3      VAN HUSEN, JEFF             801-5A-3640
 4      SIM-SMITH, ANGELA           219-68-2436abc
 5      TIMMONS, DAVID              hello219-68-1098
```

c10s5d2

133

The PRXMATCH Function

The PRXPARSE function is not required to compile the regular expression. A regular expression can be used in the PRXMATCH function.

```
data Invalidssn;
   set ia.Staff;
   if
prxmatch('/^\d{3}-\d{2}-\d{4}$/',trim(ssn))=0;
run;
proc print data = Invalidssn;
   title 'Invalid Social Security Numbers';
   var Name SSN;
run;
```

c10s5d3

134

The PRXCHANGE Function

The PRXCHANGE function uses a Perl regular expression to perform a pattern-match **replacement**.

PRXCHANGE(*Perl-regular-expression, times, source*)

Perl-regular-expression
: specifies a pattern to search for and a string to replace with.

times
: specifies number of times to perform the replacement.

source
: specifies the string to be searched.

135

Use the value -1 for the *times* argument to replace all occurrences.

The PRXCHANGE Function

Create a variable **NewName** with Firstname Lastname.

`ia.Staff` (Partial Listing)

```
Name

O'REILY, MARY
PYLES, JANE
HOFFMAN, VALERIE
DAWN, JENNIFER
VAN HUSEN, JEFF
SIM-SMITH, ANGELA
TIMMONS, DAVID
BENJAMIN, CATHERINE
```

136

The PRXCHANGE Function

```
data Namechange;
   set ia.Staff;
   re = prxparse('s/([^,]+), (\w+(\s+\w+)?)/$2 $1/');
   NewName = prxchange(re,1,Name);
run;
proc print data=Namechange;
   title 'Rearranged Names';
   var Name NewName;
run;
```

137 c10s5d4

The roles of the items in the regular expression:

`s`	Perform a substitution.
`/`	Start regular expression.
`(`	Start capture buffer #1 to store the last name.

`[^,]+`
`[ ]`
`^`
`,`
`+`
	Match one or more non-comma characters.
	Specify a set of characters.
	NOT.
	Match a comma.
	One or more times.

`)`	End capture buffer #1.
`,`	Match a comma.
	Match a space.
`(`	Start capture buffer #2 to store the first name.
`\w+`	Match a word character one or more times.

(Continued on the next page.)

(	Start capture buffer #3 to store an optional middle name.
\s+	Match a white space (space, tab, carriage return) one or more times.
\w+	Match a word character one or more times.
)	End capture buffer #3. It is part of capture buffer #2.
?	Match zero or one time. (Person may not have a middle name.)
)	End capture buffer #2; holds first name and middle name.
/	End regular expression and start replacement text.
$2	Insert capture buffer #2, which contains first name and middle name.
	Insert a space.
$1	Insert capture buffer #1, which contains the last name.
/	End replacement text.

Equivalent code:

```
data Namechange;
   set ia.Staff;
   First=scan(name, 2, ' ,');
   Middle=scan(name, 3, ' ,');
   Last = scan(name,1, ' ,');
   if middle ne ' '
   then NewName=trim(first) || ' ' ||
                trim(middle) || ' ' || last;
   else NewName=trim(first) || ' ' || last;
run;
```

The PRXCHANGE Function

Partial Output

```
                  Firstname Lastname

Obs     Name                        NewName

  1     O'REILY, MARY               MARY O'REILY
  2     PYLES, JANE                 JANE PYLES
  3     HOFFMAN, VALERIE            VALERIE HOFFMAN
  4     DAWN, JENNIFER              JENNIFER DAWN
  5     VAN HUSEN, JEFF             JEFF VAN HUSEN
  6     SIM-SMITH, ANGELA           ANGELA SIM-SMITH
  7     TIMMONS, DAVID              DAVID TIMMONS
  8     BENJAMIN, CATHERINE         CATHERINE BENJAMIN
  9     WINDSOR, STEPHEN            STEPHEN WINDSOR
 10     RICHARDSON, LARRY           LARRY RICHARDSON
```

160

c10s5d4

The PRXCHANGE Function

The PRXPARSE function is not required to compile the regular expression. The regular expression can be used in the PRXCHANGE function.

```
data Namechange;
   set ia.Staff;
   NewName = prxchange('s/([^,]+), (\w+(\s+\w+)?)/$2 $1/',
                       1,Name);
run;
proc print data = Namechange;
   title 'Rearranged Names';
   var Name NewName;
run;
```

161

c10s5d5

Exercises

9. Using Perl Expressions

Create a report showing all employees in the **ia.staff** data set with invalid telephone numbers. Valid numbers are of the form *ddd-ddd-dddd*.

Partial Listing

ia.staff

Obs	Name	PhoneNumber
1	O'REILY, MARY	203-781-1255
2	PYLES, JANE	203-675-7715
3	HOFFMAN, VALERIE	212-586-0808
4	DAWN, JENNIFER	718-383-1549
5	VAN HUSEN, JEFF	201-732-8787
6	SIM-SMITH, ANGELA	201-812-5665
7	TIMMONS, DAVID	586-806
8	BENJAMIN, CATHERINE	203-781-1777
9	WINDSOR, STEPHEN	718-384-2849
10	RICHARDSON, LARRY	718-384-8816
11	BELLUM, SARAH	203-675-3434
12	GARCIA, TRACY	212-587-1247
13	MONTGOMERY, ADAM	212-587-3622
14	GEORGE, CLARA	203-781-1212
15	SABATINI, ANTHONY	203-781-0019

Use the PRINT procedure with a WHERE statement to create the report.

Output

Employees with Invalid Phone Numbers

Obs	Name	Phone Number
7	TIMMONS, DAVID	586-806

10.6 Solutions to Exercises

1. Setting Up the Files for Exercises

Copy the **ia.empdata** SAS data set into the Work library using PROC COPY:

```
proc copy in = ia out = work;
select empdata;
run;
```

✐ This is a backup copy of the data in case your program must be submitted multiple times as you test and debug.

2. Modifying All Observations in a SAS Data Set

Give all the employees in the **empdata** SAS data set a 5% salary increase using the MODIFY statement. Print the data set before and after the increase.

```
proc print data = empdata (obs = 5);
   title 'Original Data';
run;

data empdata;
   modify empdata;
   salary = salary * 1.05;
run;

proc print data = empdata (obs = 5);
   title 'Modified Data';
run;
```

3. Modifying a SAS Data Set with Values in a Transaction Data Set

Use the transaction data set **ia.empdatu** to modify the **empdata** SAS data set by the employee ID number. Do not use an index. Print the **EmpID**, **Phone**, **JobCode**, **Division**, and **Salary** variables before and after the updates to verify the changes.

```
proc print data = empdata;
   var EmpID Phone JobCode Division Salary;
   title 'Original Data';
run;

data empdata;
   modify empdata ia.empdatu;
   by EmpID;
run;

proc print data = empdata;
   var EmpID Phone JobCode Division Salary;
   title 'Modified Data';
run;
```

4. Modifying a SAS Data Set Using a Transaction Data Set and an Index

Use the transaction data set **ia.empdatu2** to modify the **empdata** SAS data set by the employee ID number. Use the index on the **empdata** SAS data set. Modify the variables **LastName**, **Location**, and **Salary**. Print the data set before and after the changes.

```
proc print data = empdata;
   var EmpID LastName Location Salary;
   title 'Original Data';
run;

data empdata;
   set ia.empdatu2 (rename = (LastName = NewLastName
                              Location = NewLocation
                              Salary = NewSalary));
   modify empdata key = EmpID;
   LastName = NewLastName;
   Location = NewLocation;
   Salary = NewSalary;
run;

proc print data = empdata;
   var EmpID LastName Location Salary;
   title 'Modified Data';
run;
```

5. Creating Generation Data Sets

Modify the data set **ia.jobhstry** by adding a maximum of three generations.

a. Use the **ia.y200061** and **ia.y200062** data sets to concatenate to **ia.jobhstry** and test your program.

b. Use PROC DATASETS to look at the generation information for **ia.jobhstry**.

```
proc datasets lib = ia  nolist;
   modify jobhstry (genmax = 3);
run;
quit;

data ia.jobhstry;
   set ia.jobhstry ia.y200061;
run;

data ia.jobhstry;
   set ia.jobhstry ia.y200062;
run;

proc datasets library = ia  nolist;
     contents data = _all_  nods;
     contents data = Jobhstry;
run;
quit;
```

6. Creating Integrity Constraints

Create integrity constraints with PROC DATASETS for **ia.empdata**.

- Place a primary key on the variable **EmpID** and add a custom message.
- Do not allow missing values for the **LastName** variable and add a custom message.
- Use PROC FSEDIT to test the constraints.

```
proc datasets lib = ia nolist;
   modify empdata;
   ic create PKEmpID = Primary Key (EmpID)
      message = 'You must supply an employee ID number';
   ic create LName = Not Null (LastName)
   message = 'You must supply a last name for the employee';
   contents data = empdata;
run;
quit;

proc fsedit data = ia.empdata;
run;
```

7. Creating a Foreign Key

Create a foreign key on the data set **ia.pilots** on the variable **EmpID** using PROC SQL. The parent table is **ia.empdata**.

- Restrict the update and deletion of the **EmpID** value.
- Test the constraints by trying to add the employee number E01724 to the **ia.pilots** data set using the PROC SQL INSERT statement.

```
proc sql;
   alter table ia.pilots
      add constraint FKEmpID Foreign Key (EmpID)
         references ia.empdata
            on update restrict
            on delete restrict;
   describe table constraints ia.pilots;
quit;

proc sql;
   insert into ia.pilots
      set EmpID = 'E01724';
quit;
```

Log

```
434  proc sql;
435     insert into IA.Pilots
436        set EmpID = 'E01724';
ERROR: Observation was not added/updated because a matching primary key value
       was not found for foreign key FKEmpID.
NOTE: Deleting the successful inserts before error noted above to restore table
      to a consistent state.
437  quit;
NOTE: The SAS System stopped processing this step because of errors.
```

8. Creating an Audit Trail

Create an audit trail for the data set **ia.pilots**.

- Add user variables to track who edited the data set and why it was edited.
- Use PROC FSEDIT to give a pilot a salary increase. Be sure to include who edited the data set and give a reason for the increase.
- Use PROC PRINT to look at the audit trail.
- Terminate the audit trail.

```
proc datasets library = ia nolist;
   audit pilots;
   initiate;
   user_var who $20 label = 'Who made the change'
            why $20 label = 'Why the change was made';
run;
quit;

proc fsedit data = ia.pilots;
run;

proc print data = ia.pilots(type = audit);
   title 'Audit Trail for ia.pilots';
run;

proc datasets library = ia nolist;
   audit pilots;
   terminate;
run;
quit;
```

9. Using Perl Expressions

Create a report showing all employees in the **ia.Staff** data set with invalid telephone numbers. Valid numbers are of the form *ddd-ddd-dddd*.

Partial Listing

```
                                ia.Staff

          Obs    Name                       PhoneNumber

            1    O'REILY, MARY              203-781-1255
            2    PYLES, JANE                203-675-7715
            3    HOFFMAN, VALERIE           212-586-0808
            4    DAWN, JENNIFER             718-383-1549
            5    VAN HUSEN, JEFF            201-732-8787
            6    SIM-SMITH, ANGELA          201-812-5665
            7    TIMMONS, DAVID             586-806
            8    BENJAMIN, CATHERINE        203-781-1777
            9    WINDSOR, STEPHEN           718-384-2849
           10    RICHARDSON, LARRY          718-384-8816
           11    BELLUM, SARAH              203-675-3434
           12    GARCIA, TRACY              212-587-1247
           13    MONTGOMERY, ADAM           212-587-3622
           14    GEORGE, CLARA              203-781-1212
           15    SABATINI, ANTHONY          203-781-0019
```

Use the PRINT procedure with a WHERE statement to create the report.

Output

```
                Employees with Invalid Phone Numbers

                                                 Phone
                       Obs          Name         Number

                         7    TIMMONS, DAVID    586-806
```

```
proc print data=ia.Staff;
   where prxmatch('/\d{3}-\d{3}-\d{4}/', PhoneNumber) = 0;
   var Name PhoneNumber;
   title "Employees with Invalid Phone Numbers";
run;
```

Appendix A Index