临床研究SAS高级编程
—— SAS Dataset Creation

Contents

- SAS data step
- Access to PC files
- Access to Oracle
- Access to SQL

SAS Data Step

- Contents
  - Creating SAS data sets from raw data
  - Creating and managing variables

Creating SAS Data Sets from Raw Data

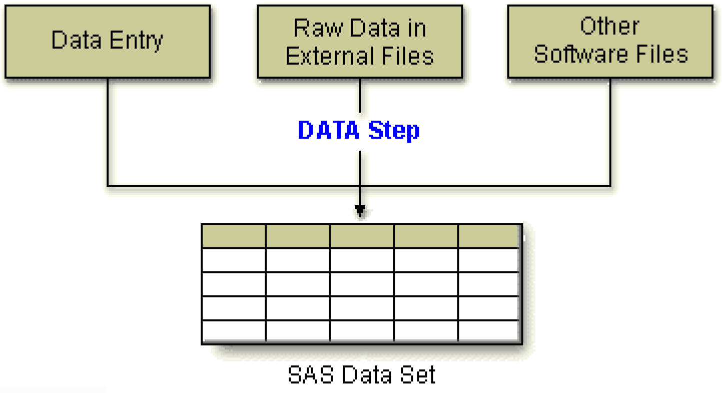
- Contents

- Overview
- Raw data files
- Steps to create a SAS data set
- Referencing a SAS library
- Referencing a raw data file
- Writing a DATA step program
- Submitting the DATA step program
- Creating and modifying variables
- Subsetting data
- Reading instream data
- Steps to create a raw data file

Overview

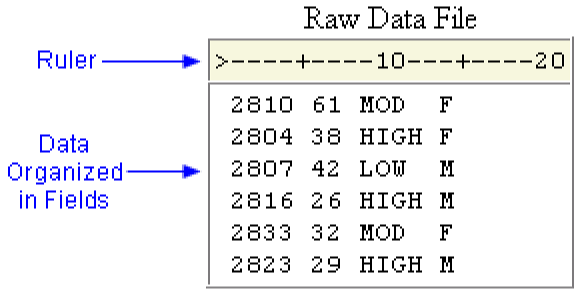
Introduction

- In order to create reports with SAS procedures, your data must be in the form of a SAS data set. If your data is not stored in the form of a SAS data set, then you need to create a SAS data set by entering data, by reading raw data, or by accessing external files (files that were created by other software)
- This shows you how to design and write a DATA step program to create a SAS data set from **raw data** that is stored in an external file. It also shows you how to read data from a SAS data set and write observations out to a raw data file.



Raw Data Files

A raw data file is an external text file whose records contain data values that are organized in fields. Raw data files are non-proprietary and can be read by a variety of software programs.



```
>----+----10---+----20
 2810 61 MOD  F
 2804 38 HIGH F
 2807 42 LOW  M
 2816 26 HIGH M
 2833 32 MOD  F
 2823 29 HIGH M
```

Steps to Create a SAS Data Set

To read the raw data file, the DATA step must provide the following instructions to SAS:

- the location or name of the external text file
- a name for the new SAS data set
- a reference that identifies the external file
- a description of the data values to be read.

To do this...	**Use this SAS statement...**
Reference SAS data library	LIBNAME statement
Reference external file	FILENAME statement
Name SAS data set	DATA statement
Identify external file	INFILE statement
Describe data	INPUT statement
Execute DATA step	RUN statement
List the data	PROC PRINT statement
Execute final program step	RUN statement

Referencing a SAS Library

Using a LIBNAME statement

- As you begin to write the program, remember that you use a LIBNAME statement to reference the permanent SAS library in which the data set will be stored.

To do this...	Use this SAS statement...	Example
Reference a SAS library	**LIBNAME statement**	libname libref 'SAS-data- library';

- For example, the LIBNAME statement below assigns the libref **Taxes** to the SAS library **C:\Users\Acct\Qtr1\Report** in the Windows environment.

```
libname taxes 'c:\users\acct\qtr1\report';
```

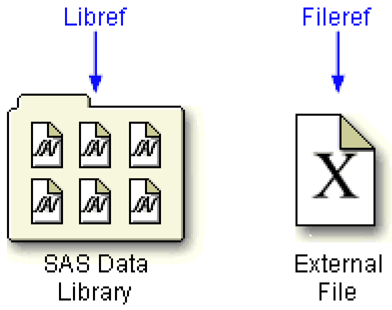
Referencing a Raw Data File (1)

Using a FILENAME statement (1)

- Before you can read your raw data, you must point to the location of the external file that contains the data. You use the FILENAME statement to point to this location.

To do this...	Use this SAS statement...	Example
Reference an external file	**FILENAME statement**	filename tests 'c:\users\tmill.dat';

- Filerefs perform the same function as librefs: they temporarily point to a storage location for data. However, librefs reference SAS data libraries, whereas filerefs reference external files.



Referencing a Raw Data File (2)

Using a FILENAME statement (2)

- General form, FILENAME statement:

  FILENAME ***fileref 'filename';***

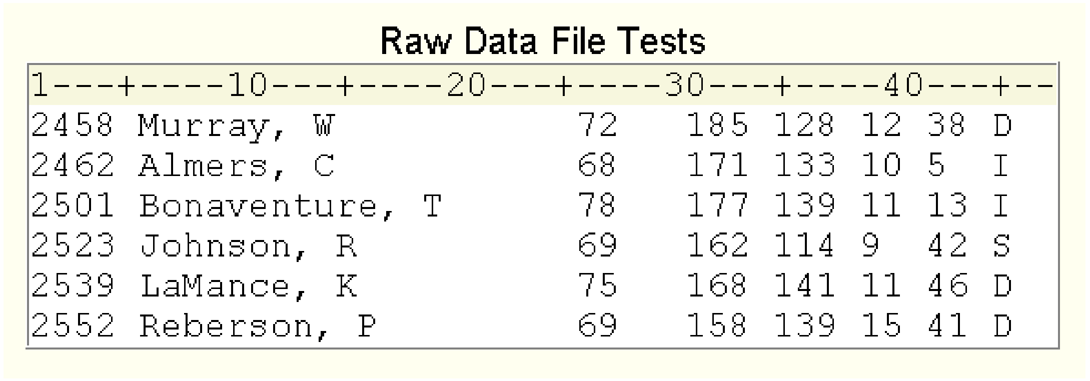
  where

  - *fileref* is a name that you associate with an external file. The name must be I to 8 characters long, begin with a letter or underscore, and contain only letters, numbers, or underscores.
  - *filename* is the fully qualified name or location of the file.

Referencing a Raw Data File (3)

- Defining a fully qualified filename
  - The following FILENAME statement temporarily associates the fileref **Tests** with the external file that contains the data from the exercise stress tests. The complete filename is specified as **C:\Users\Tmill.dat** in the Windows environment.

filename tests 'c:\users\tmill.dat';

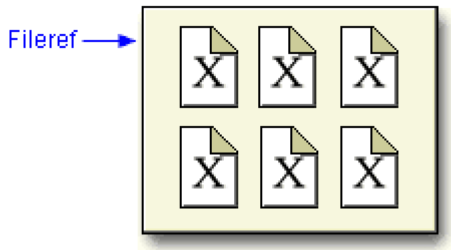

Raw Data File Tests

```
1---+----10---+----20---+----30---+----40---+--
2458 Murray, W              72    185 128 12 38 D
2462 Almers, C              68    171 133 10 5  I
2501 Bonaventure, T         78    177 139 11 13 I
2523 Johnson, R             69    162 114 9  42 S
2539 LaMance, K             75    168 141 11 46 D
2552 Reberson, P            69    158 139 15 41 D
```

Referencing a Raw Data File (4)

Defining an aggregate storage location

- You can also use a FILENAME statement to associate a fileref with an aggregate storage location, such as a directory that contains multiple external files.



- This FILENAME statement temporarily associates the fileref **Finance** with the aggregate storage directory **C:\Users\Personal\Finances**:

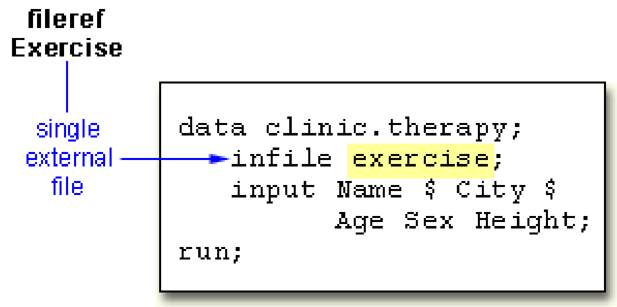
filename finance 'c:\users\personal\finances';

  - Note: Both the LIBNAME and FILENAME statements are global. In other words, they remain in effect until you change them, cancel them, or end your SAS session.

Referencing a Raw Data File (5)

Referencing a fully qualified filename

- When you associate a fileref with an individual external file, you specify the fileref in subsequent SAS statements and commands.

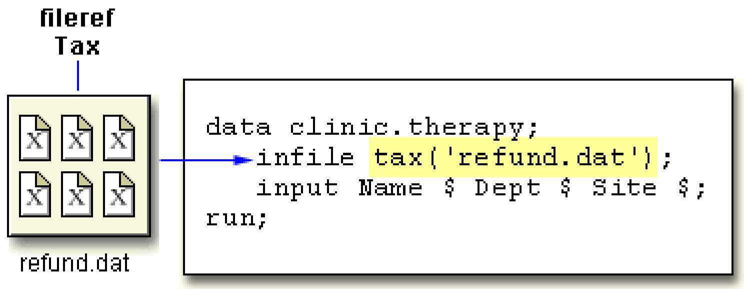


```
data clinic.therapy;
   infile exercise;
   input Name $ City $
         Age Sex Height;
run;
```

Referencing a Raw Data File (6)

Referencing a file in an aggregate storage location

- To reference an external file with a fileref that points to an aggregate storage location, you specify the fileref followed by the individual filename in parentheses:



```
data clinic.therapy;
   infile tax('refund.dat');
   input Name $ Dept $ Site $;
run;
```

- Note: In the Windows operating environment, you can omit the filename extension but you will need to add quotation marks when referencing the external file, as in

infile tax(‘refund’);

Writing a DATA Step Program (1)

Naming the data set

- The DATA statement indicates the beginning of the DATA step and names the SAS data set to be created.

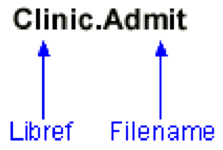
To do this...	Use this SAS statement...	Example
Name a SAS data set	**DATA statement**	data Sasuser.Stress;

- General form, basic DATA statement:

DATA SAS-data-set-1 <...SAS-data-set-n>;

where *SAS-data-set* is the name (*libref.filename*) of the data set to be created.

- Remember that the SAS data set name is a two-level name. For example, the two-level name **Clinic.Admit** specifies that the data set **Admit** is stored in the permanent SAS library to which the libref **Clinic** has been assigned.



Writing a DATA Step Program (2)

Specifying the raw data file (1)

- When reading raw data, use the INFILE statement to indicate which file the data is in.

To do this...	Use this SAS statement...	Example
Identify an external file	**INFILE statement**	infile tests obs=10;

- General form, INFILE statement:

INFILE *file-specification <options>;*

where

- *file-specification* can take the form *fileref* to name a previously defined file reference or '*filename*' to point to the actual name and location of the file
- *options* describes the input file's characteristics and specifies how it is to be read with the INFILE statement.

Writing a DATA Step Program (3)

Specifying the raw data file (2)

- To read the raw data file to which the fileref **Tests** has been assigned, you write the following INFILE statement:

```
infile tests;
```

  - Note: Instead of using a FILENAME statement, you can choose to identify the raw data file by specifying the entire filename and location in the INFILE statement.
    - For example, the following statement points directly to the **C:\Irs\Personal\Refund.dat** file:

```
infile 'c:\irs\personal\refund.dat';
```

Writing a DATA Step Program (4)

- Quiz
  - Which statement identifies the name of a raw data file to be read with the fileref Products and specifies that the DATA step read only records 1–15?
    - a. infile products obs 15;
    - b. infile products obs=15;
    - c. input products obs=15;
    - d. input products 1-15;
  - Correct answer: b
    - You use an INFILE statement to specify the raw data file to be read. You can specify a fileref or an actual filename (in quotation marks). The OBS= option in the INFILE statement enables you to process only records 1 through *n.*

Writing a DATA Step Program (5)

- Column input
  - In this section, you'll be working with column input, the most common input style. Column input specifies actual column locations for values. However, column input is appropriate only in certain situations. When you use column input, your data **must** be
    - standard character or numeric values
    - in fixed fields.

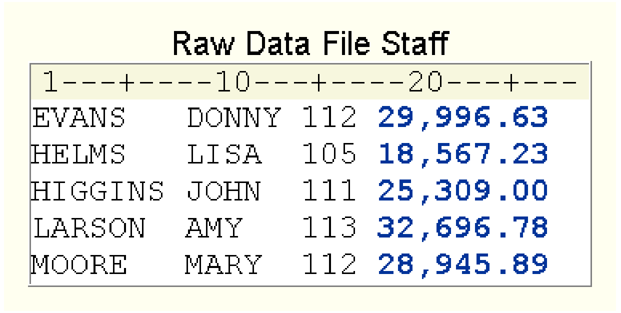
Writing a DATA Step Program (6)

- Standard and nonstandard numeric data (1)
  - Standard numeric data values can contain only
    - numbers
    - decimal points
    - numbers in scientific or E-notation (*2.3E4*, for example)
    - plus or minus signs.
  - Nonstandard numeric data includes
    - values that contain special characters, such as percent signs (%), dollar signs ($), and commas (,)
    - date and time values
    - data in fraction, integer binary, real binary, and hexadecimal forms.
  - The external file that is referenced by the fileref **Staff**. The fields contain values for each employee’s last name, first name, job title, and annual salary.

Writing a DATA Step Program (7)

Standard and nonstandard numeric data (2)

- Notice that the values for Salary contain commas. The values for Salary are considered to be nonstandard numeric values. You cannot use column input to read these values.



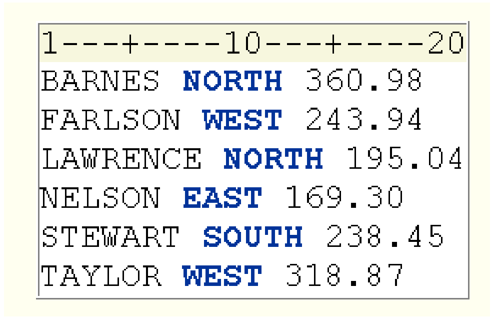
Raw Data File Staff

```
1---+----10---+----20---+---
EVANS    DONNY 112 29,996.63
HELMS    LISA  105 18,567.23
HIGGINS  JOHN  111 25,309.00
LARSON   AMY   113 32,696.78
MOORE    MARY  112 28,945.89
```

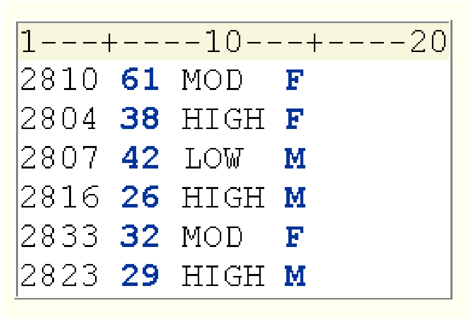
Writing a DATA Step Program (8)

Fixed-field data

- Raw data can be organized in several different ways.
- This external file contains data that is free-format, meaning data that is not arranged in columns. Notice that the values for a particular field do not begin and end in the same columns. You cannot use column input to read this file.
- This external file contains data that is arranged in columns or fixed fields. You can specify a beginning and ending column for each field. Let's look at how column input can be used to read this data.



```
1---+----10---+----20
BARNES NORTH 360.98
FARLSON WEST 243.94
LAWRENCE NORTH 195.04
NELSON EAST 169.30
STEWART SOUTH 238.45
TAYLOR WEST 318.87
```



```
1---+----10---+----20
2810 61 MOD  F
2804 38 HIGH F
2807 42 LOW  M
2816 26 HIGH M
2833 32 MOD  F
2823 29 HIGH M
```

Writing a DATA Step Program (9)

Describing the data (1)

- The INPUT statement describes the fields of raw data to be read and placed into the SAS data set.

To do this...	Use this SAS statement...	Example
Describe data	**INPUT statement**	input ID 1-4 Name $ 6-25 ...;
Execute the DATA step	**RUN statement**	run;

- General form, INPUT statement using column input:

INPUT *variable ;***<$>** *startcol-endcol . . .*

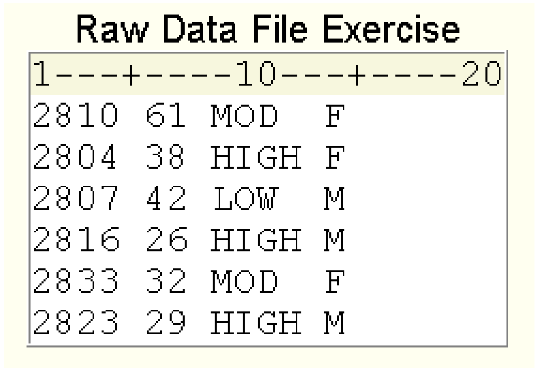
where

- *variable* is the SAS name that you assign to the field
- the dollar sign ($) identifies the variable type as character (if the variable is numeric, then nothing appears here)
- *startcol* represents the starting column for this variable
- *endcol* represents the ending column for this variable.

Writing a DATA Step Program (10)

Describing the data (2)

- Look at the small data file shown below. For each field of raw data that you want to read into your SAS data set, you must specify the following information in the INPUT statement:
  - a valid SAS variable name
  - a type (character or numeric)
  - a range (starting column and ending column).



Raw Data File Exercise

```
1---+----10---+----20
2810 61 MOD  F
2804 38 HIGH F
2807 42 LOW  M
2816 26 HIGH M
2833 32 MOD  F
2823 29 HIGH M
```

Writing a DATA Step Program (11)

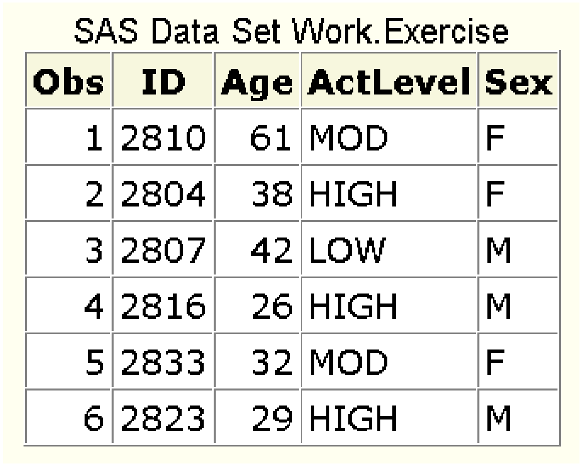
Describing the data (3)

- The INPUT statement below assigns the character variable ID to the data in columns 1-4, the numeric variable Age to the data in columns 6-7, the character variable ActLevel to the data in columns 9-12, and the character variable Sex to the data in column 14.

```
filename exer 'c:\users\exer.dat';
data exercise;
     infile exer;
     input ID $ 1-4 Age 6-7 ActLevel $ 9-12 Sex $ 14;
run;
```

Writing a DATA Step Program (12)

Describing the data (4)



SAS Data Set Work.Exercise

Obs	ID	Age	ActLevel	Sex
1	2810	61	MOD	F
2	2804	38	HIGH	F
3	2807	42	LOW	M
4	2816	26	HIGH	M
5	2833	32	MOD	F
6	2823	29	HIGH	M

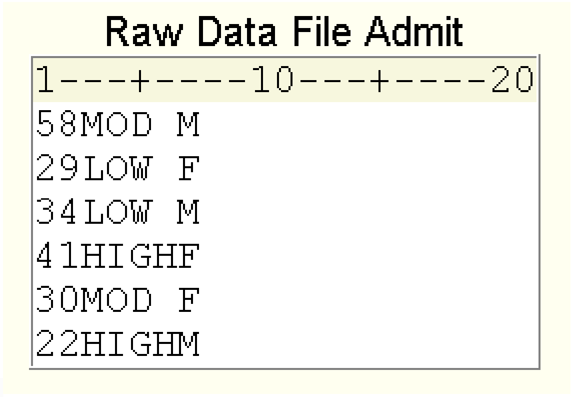
- When you use column input, you can
  - read any or all fields from the raw data file
  - read the fields in any order
  - specify only the starting column for values that occupy only one column.

```
input ActLevel $ 9-12 Sex $ 14 Age 6-7;
```

Writing a DATA Step Program (13)

Specifying variable names (1)

- Each variable has a name that conforms to SAS naming conventions. Variable names
  - must be 1 to 32 characters in length
  - must begin with a letter (A-Z) or an underscore (_)
  - can continue with any combination of numbers, letters, or underscores.
- Let’s look at an INPUT statement that uses column input to read the three data fields in the raw data file below.


Raw Data File Admit

```
1---+----10---+----20
58MOD M
29LOW F
34LOW M
41HIGHF
30MOD F
22HIGHM
```

Writing a DATA Step Program (14)

Specifying variable names (2)

- The values for the variable that you are naming Age are located in columns l-2. Because Age is a numeric variable, you do not specify a dollar sign ($) after the variable name.

```
input Age 1–2
```

- The values for the variable ActLevel are located in columns 3-6. You specify a $ to indicate that ActLevel is a character variable.

```
input Age 1-2 ActLevel $ 3–6
```

- The values for the character variable Sex are located in column 7. Notice that you specify only a single column.

```
input Age 1–2 ActLevel $ 3–6 Sex $ 7;
```

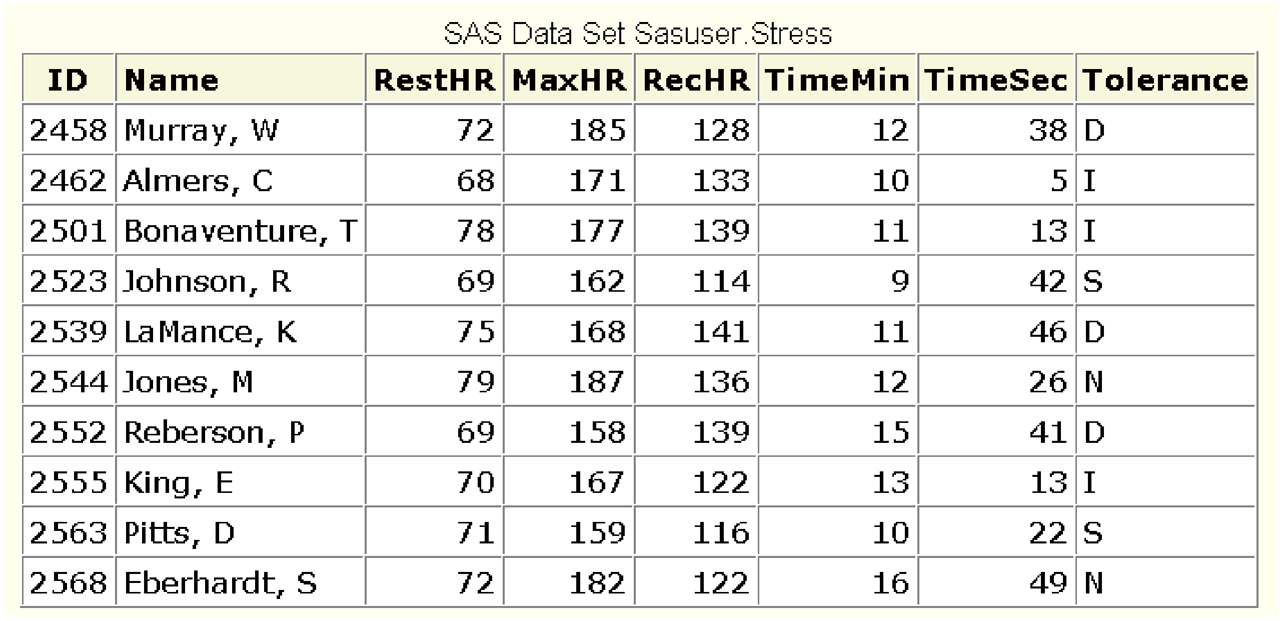
Submitting the DATA Step Program (1)

- Verifying the data (1)
  - To verify your data, it is a good idea to use the **OBS= option** in the INFILE statement. Adding OBS=*n* to the INFILE statement enables you to process only records 1 through *n*, so you can verify that the correct fields are being read before reading the entire data file.
  - The program below reads the first ten records in the raw data file referenced by the fileref **Tests**. The data is stored in a permanent SAS data set, named **Sasuser.Stress**. Don't forget a RUN statement, which tells SAS to execute the previous SAS statements.

```
data Sasuser.stress;
     infile tests obs=10;
     input ID 1-4        Name $ 6-25    RestHR 27-29
           MaxHR 31-33   RecHR 35-37    TimeMin 39-40
           TimeSec 42-43 Tolerance $ 45;
run;
```

Submitting the DATA Step Program (2)

- Verifying the data (2)



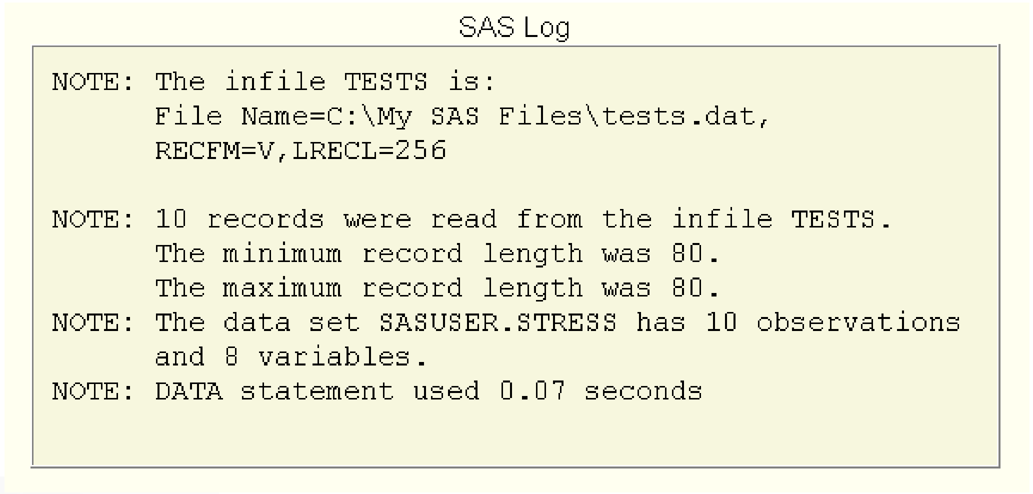
SAS Data Set Sasuser.Stress

ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance
2458	Murray, W	72	185	128	12	38	D
2462	Almers, C	68	171	133	10	5	I
2501	Bonaventure, T	78	177	139	11	13	I
2523	Johnson, R	69	162	114	9	42	S
2539	LaMance, K	75	168	141	11	46	D
2544	Jones, M	79	187	136	12	26	N
2552	Reberson, P	69	158	139	15	41	D
2555	King, E	70	167	122	13	13	I
2563	Pitts, D	71	159	116	10	22	S
2568	Eberhardt, S	72	182	122	16	49	N

Submitting the DATA Step Program (3)

Checking DATA step processing

- After submitting the previous program, messages in the log verify that the raw data file was read correctly. The notes in the log indicate that
  - 10 records were read from the raw data file
  - the SAS data set **Sasuser.Stress** was created with 10 observations and 8 variables.


SAS Log

```
NOTE: The infile TESTS is:
      File Name=C:\My SAS Files\tests.dat,
      RECFM=V,LRECL=256

NOTE: 10 records were read from the infile TESTS.
      The minimum record length was 80.
      The maximum record length was 80.
NOTE: The data set SASUSER.STRESS has 10 observations
      and 8 variables.
NOTE: DATA statement used 0.07 seconds
```

Submitting the DATA Step Program (4)

- Listing the data set
  - The messages in the log seem to indicate that the DATA step program correctly accessed the raw data file. But it is a good idea to look at the ten observations in the new data set before reading the entire raw data file. You can submit a PROC PRINT step to view the data.

To do this...	**Use this SAS statement...**	**Example**
List the data	**PROC PRINT statement**	proc print data=Sasuser.Stress;
Execute the final program step	**RUN statement**	run;

  - The following PROC PRINT step lists the **Sasuser.Stress** data set.

```
proc print data=Sasuser.Stress;
run;
```

    - The PROC PRINT output indicates that the variables in the **Sasuser.Stress** data set were read correctly for the first 10 records.

Submitting the DATA Step Program (5)

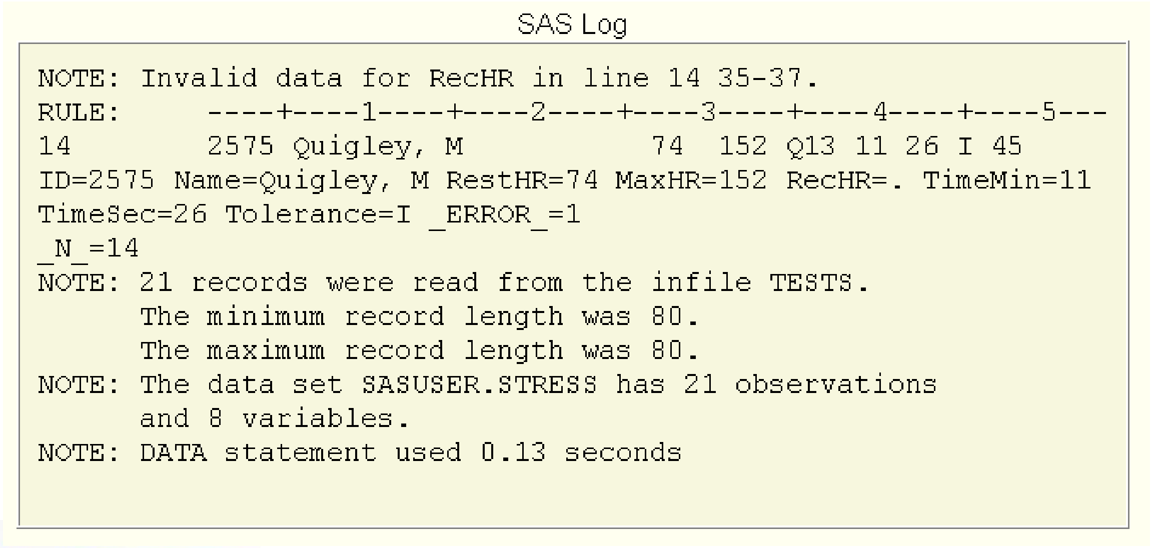
- Reading the entire raw data file
  - Now that you’ve checked the log and verified your data, you can modify the DATA step to read the entire raw data file. To do so, remove the OBS= option from the INFILE statement and re- submit the program.

```
data Sasuser.Stress;
   infile tests;
   input ID 1-4         Name $ 6-25    RestHR 27-29
         MaxHR 31-33    RecHR 35-37    TimeMin 39-40
         TimeSec 42-43 Tolerance $ 45;
run;
```

Submitting the DATA Step Program (6)

Invalid data (1)

- When you submit the revised DATA step and check the log, you see a note indicating that invalid data appears for the variable RecHR in line 14 of the raw data file, columns 35-37.
- This note is followed by a column ruler and the actual data line that contains the invalid value for RecHR.


SAS Log

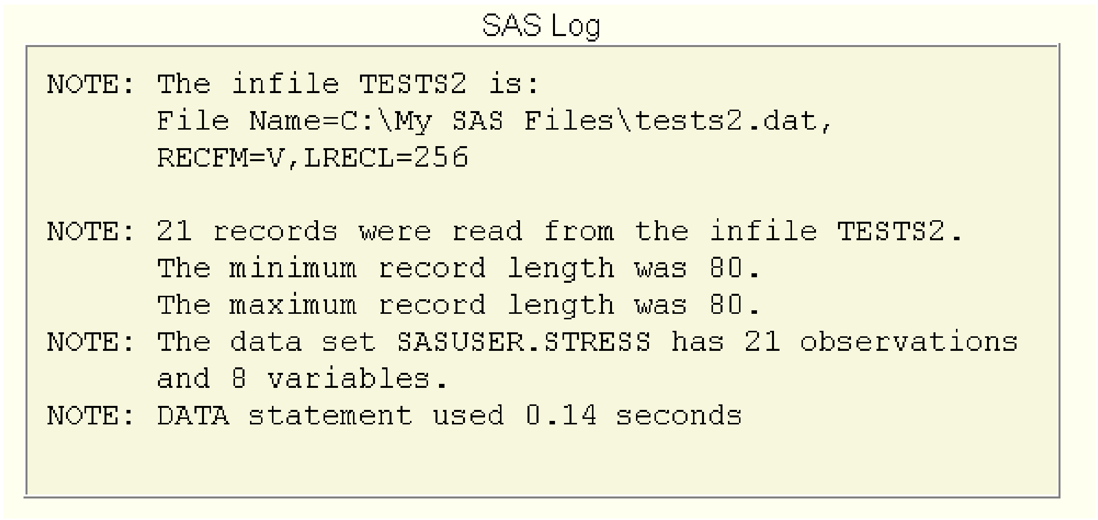
```
NOTE: Invalid data for RecHR in line 14 35-37.
RULE:     ----+----1----+----2----+----3----+----4----+----5---
14        2575 Quigley, M           74  152 Q13 11 26 I 45
ID=2575 Name=Quigley, M RestHR=74 MaxHR=152 RecHR=. TimeMin=11
TimeSec=26 Tolerance=I _ERROR_=1
_N_=14
NOTE: 21 records were read from the infile TESTS.
      The minimum record length was 80.
      The maximum record length was 80.
NOTE: The data set SASUSER.STRESS has 21 observations
      and 8 variables.
NOTE: DATA statement used 0.13 seconds
```

Submitting the DATA Step Program (7)

- Invalid data (2)
  - The value *Q13* is a data-entry error. It was entered incorrectly for the variable RecHR.
  - RecHR is a numeric variable, but *Q13* is not a valid number. So RecHR is assigned a missing value, as indicated in the log. Because RecHR is numeric, the missing value is represented with a period.
  - Notice, though, that the DATA step does not fail as a result of the invalid data but continues to execute. Unlike syntax errors, invalid data errors **do not** cause SAS to stop processing a program.
  - Assuming that you have a way to edit the file and can justify a correction, you can correct the invalid value and rerun the DATA step. If you did this, the log would then show that the data set Sasuser.Stress was created with 21 observations, 8 variables, and no messages about invalid data.

Submitting the DATA Step Program (8)

Invalid data (3)


SAS Log

```
NOTE: The infile TESTS2 is:
      File Name=C:\My SAS Files\tests2.dat,
      RECFM=V,LRECL=256

NOTE: 21 records were read from the infile TESTS2.
      The minimum record length was 80.
      The maximum record length was 80.
NOTE: The data set SASUSER.STRESS has 21 observations
      and 8 variables.
NOTE: DATA statement used 0.14 seconds
```

► After correcting the raw data file, you can list the data again to verify that it is correct.

```
proc print data=Sasuser.Stress;
run;
```

Submitting the DATA Step Program (9)

- Invalid data (4)
  - Whenever you use the DATA step to read raw data, remember the steps that you followed in this chapter, which help ensure that you don’t waste resources when accessing data:
    - Write the DATA step using the OBS= option in the INFILE statement.
    - Submit the DATA step.
    - Check the log for messages.
    - View the resulting data set.
    - Remove the OBS= option and re-submit the DATA step.
    - Check the log again.
    - View the resulting data set again.

Submitting the DATA Step Program (10)

- Quiz
  - Which statement correctly reads the fields in the following order: StockNumber, Price, Item, Finish, Style?

Field Name	Start Column	End Column	Data Type
StockNumber	1	3	character
Finish	5	9	character
Style	11	18	character
Item	20	24	character
Price	27	32	numeric

```
1---+----10---+----20---+----30---+
310 oak   pedestal table  329.99
311 maple pedestal table  369.99
312 brass floor    lamp    79.99
313 glass table    lamp    59.99
313 oak   rocking  chair  153.99
```

  - *a.* input StockNumber $ 1-3 Finish $ 5-9 Style $ 11-18 Item $ 20-24 Price 27-32;
  - *b.* input StockNumber $ 1-3 Price 27-32 Item $ 20-24 Finish $ 5-9 Style $ 11-18;
  - *c.* input $ StockNumber 1-3 Price 27-32 $ Item 20-24 $ Finish 5-9 $ Style 11-18;
  - *d.* input StockNumber $ 1-3 Price $ 27-32 Item $ 20-24 Finish $ 5-9 Style $ 11-18;

- Correct answer: b
  - You can use column input to read fields in any order. You must specify the variable name to be created, identify character values with a $, and name the correct starting column and ending column for each field.

Creating and Modifying Variables (1)

Overview

- To modify existing values or to create new variables, you can use an **assignment statement** in any DATA step.
- General form, assignment statement:

  variable=expression;

  where
  - *variable* names a new or existing variable
  - *expression* is any valid SAS expression.
    - Note: The assignment statement is one of the few SAS statements that doesn't begin with a keyword.
- For example, here is an assignment statement that assigns the character value Toby Witherspoon to the variable Name:

```
Name='Toby Witherspoon';
```

Creating and Modifying Variables (2)

- SAS expressions
  - You use SAS expressions in assignment statements and many other SAS programming statements to
    - transform variables
    - create new variables
    - conditionally process variables
    - calculate new values
    - assign new values.
  - An expression is a sequence of operands and operators that form a set of instructions. The instructions are performed to produce a new value:
    - **Operands** are variable names or constants. They can be numeric, character, or both.
    - **Operators** are special-character operators, grouping parentheses, or functions.

Creating and Modifying Variables (3)

Using operators in SAS expressions (1)

- To perform a calculation, you use **arithmetic operators**. The table below lists arithmetic operators.

Operator	Action	Example	Priority
-	negative prefix	negative=-x;	I
**	exponentiation	raise=x**y;	I
*	multiplication	mult=x*y;	II
/	division	divide=x/y;	II
+	addition	sum=x+y;	III
-	subtraction	diff=x-y;	III

- When you use more than one arithmetic operator in an expression,
  - operations of priority I are performed before operations of priority II, and so on
  - consecutive operations that have the same priority are performed
    - from right to left within priority I
    - from left to right within priority II and III
  - you can use parentheses to control the order of operations.
  - Warning: When a value that is used with an arithmetic operator is missing, the result of the expression is **missing**. The assignment statement assigns a **missing value** to a variable if the result of the expression is missing.

Creating and Modifying Variables (4)

Using operators in SAS expressions (2)

- You use the following **comparison operators** to express a condition.

Operator	Meaning	Example
= or **eq**	equal to	name='Jones, C.'
^= or **ne**	not equal to	temp ne 212
> or **gt**	greater than	income>20000
< or **lt**	less than	partno lt "BG05"
>= or **ge**	greater than or equal to	id>='1543'
<= or **le**	less than or equal to	pulse le 85

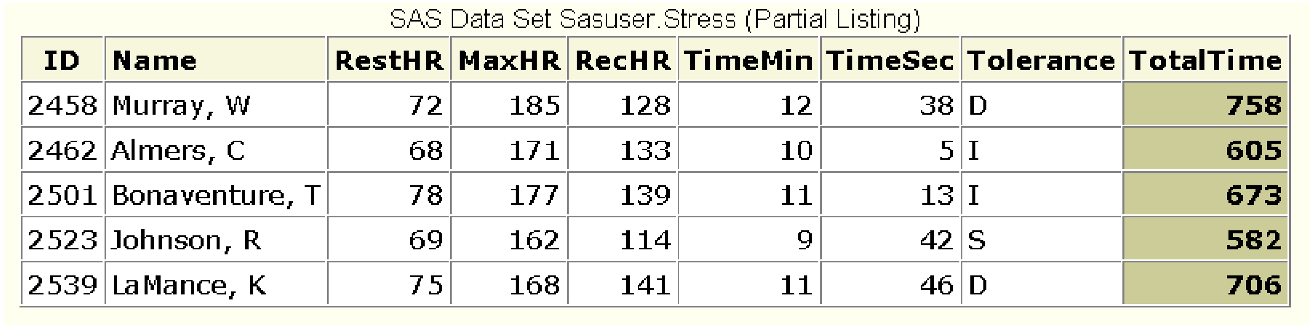
- To link a sequence of expressions into compound expressions, you use **logical operators**, including the following.

Operator	Meaning
AND or **&**	and, both. If both expressions are true, then the compound expression is true.
OR or **|**	or, either. If either expression is true, then the compound expression is true.

Creating and Modifying Variables (5)

- More examples of assignment statements (1)
  - ➤ The assignment statement in the DATA step below creates a new variable, TotalTime, by multiplying the values of TimeMin by 60 and then adding the values of TimeSec.

```
data Sasuser.Stress;
     infile tests;
     input ID 1-4         Name $ 6-25 RestHr 27-29
           MaxHR 31-33    RecHR 35-37 TimeMin 39-40
           TimeSec 42-43 Tolerance $ 45;
     TotalTime=(timemin*60)+timesec;
run;
```



SAS Data Set Sasuser.Stress (Partial Listing)

ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime
2458	Murray, W	72	185	128	12	38	D	758
2462	Almers, C	68	171	133	10	5	I	605
2501	Bonaventure, T	78	177	139	11	13	I	673
2523	Johnson, R	69	162	114	9	42	S	582
2539	LaMance, K	75	168	141	11	46	D	706

Creating and Modifying Variables (6)

More examples of assignment statements (2)

- The expression can also contain the variable name that is on the left side of the equal sign, as the following assignment statement shows. This statement re-defines the values of the variable RestHR as 10 percent higher.

```
data Sasuser.Stress;
     infile tests;
     input ID 1-4          Name $ 6-25     RestHr 27-29
           MaxHR 31-33     RecHR 35-37     TimeMin 39-40
           TimeSec 42-43   Tolerance $ 45
     resthr=resthr+(resthr*.10);
run;
```

Creating and Modifying Variables (7)

Date constants (1)

- You can assign date values to variables in assignment statements by using **date constants**. To represent a constant in SAS date form, specify the date as *'ddmmmyy'* or *'ddmmmyyyy'*, followed by a D.
- General form, date constant:

  'ddmmm<yy>yy' D or *"ddmmm<yy>yy"* D

  where
  - *dd* is a one- or two-digit value for the day
  - *mmm* is a three-letter abbreviation for the month (JAN, FEB, and so on)
  - *yy* or *yyyy* is a two- or four-digit value for the year, respectively.
  - Note: Be sure to enclose the date in quotation marks.

Creating and Modifying Variables (8)

Date constants (2)

- Example
  - In the following program, the second assignment statement assigns a date value to the variable TestDate.

```
data Sasuser.Stress;
     infile tests;
     input ID 1-4           Name $ 6-25       RestHr 27-29
           MaxHR 31-33      RecHR 35-37       TimeMin 39-40
           TimeSec 42-43    Tolerance $ 45;
     TotalTime=(timemin*60)+timesec;
     TestDate='01jan2000'd;
run;
```

- Note: You can also use SAS **time constants** and SAS **datetime constants** in assignment statements.

```
Time='9:25't;
DateTime='18jan2005:9:27:05'dt;
```

Subsetting Data (1)

- As you read your data, you can subset it by processing only those observations that meet a specified condition. To do this, you can use a **subsetting IF statement** in any DATA step.

- The subsetting IF statement causes the DATA step to continue processing only those raw data records or observations that meet the condition of the expression specified in the IF statement. The resulting SAS data set or data sets contain a subset of the original external file or SAS data set.

- General form, subsetting IF statement:

  IF *expression;*

  where *expression* is any valid SAS expression.

  - If the expression is **true**, the DATA step continues to process that record or observation.
  - If the expression is **false**, no further statements are processed for that record or observation, and control returns to the top of the DATA step.

Subsetting Data (2)

- For example, the subsetting IF statement below selects only observations whose values for Tolerance are *D*. The IF statement is positioned in the DATA step so that other statements do not need to process unwanted observations.

```
data Sasuser.Stress;
     infile tests;
     input ID 1-4          Name $ 6-25     RestHr 27-29
           MaxHR 31-33     RecHR 35-37     TimeMin 39-40
           TimeSec 42-43   Tolerance $ 45;
     if tolerance='D';
     TotalTime=(timemin*60)+timesec;
run;
```

- Because Tolerance is a character variable, the value *D* must be enclosed in quotation marks, and it must be the same case as in the data set.

Reading Instream Data (1)

- Throughout this section, our program has contained an INFILE statement that identifies an **external file** to read.

```
data Sasuser.Stress;
     infile tests;
     input ID 1-4          Name $ 6-25      RestHr 27-29
           MaxHR 31-33     RecHR 35-37      TimeMin 39-40
           TimeSec 42-43   Tolerance $ 45;
     if tolerance='D';
     TotalTime=(timemin*60)+timesec;
run;
```

- However, you can also read **instream data lines** that you enter directly in your SAS program, rather than data that is stored in an external file. Reading instream data is extremely helpful if you want to create data and test your programming statements on a few observations that you can specify according to your needs.

Reading Instream Data (2)

- To read instream data, you use
  - a **DATALINES statement** as the last statement in the DATA step (except for the RUN statement) and immediately preceding the data lines
  - a **null statement** (a single semicolon) to indicate the end of the input data.

```
data Sasuser.Stress;
   input ID 1-4          Name $ 6-25     RestHr 27-29
         MaxHR 31-33     RecHR 35-37     TimeMin 39-40
         TimeSec 42-43   Tolerance $ 45;
   datalines;
.
data lines go here
.
;
```

- General form, DATALINES statement:

DATALINES;

  - Note: You can use only one DATALINES statement in a DATA step. Use separate
  - DATA steps to enter multiple sets of data.
  - Note: You can also use **CARDS**; as the last statement in a DATA step (except for the RUN statement) and immediately preceding the data lines. The CARDS statement is an alias for the DATALINES statement.
  - Note: If your data contains semicolons, use the DATALINES4 statement plus a null statement that consists of four semicolons (;;;;) to indicate the end of the input data.

Reading Instream Data (3)

Example

- To read the data for the treadmill stress tests as instream data, you can submit the following program:

```
data Sasuser.Stress;
     input ID 1-4          Name $ 6-25      RestHr 27-29
           MaxHR 31-33     RecHR 35-37      TimeMin 39-40
           TimeSec 42-43   Tolerance $ 45;
     if tolerance='D';
     TotalTime=(timemin*60)+timesec;
     datalines;
2458 Murray, W             72  185 128 12 38 D
2462 Almers, C                68  171 133 10  5 I
2501 Bonaventure, T        78  177 139 11 13 I
….
;
```

- Warning: Notice that you do not need a RUN statement following the null statement (the semicolon after the data lines). The null statement functions as a step boundary when the DATALINES statement is used, so the DATA step is executed as soon as SAS encounters it. If you do place a RUN statement after the null statement, any statements between the null statement and the RUN statement are **not** executed as part of the DATA step.

Steps to Create a Raw Data File (1)

Using the _NULL_ keyword

- Because the goal of your SAS program is to create a raw data file and not a SAS data set, it is inefficient to list a data set name in the DATA statement. Instead, use the keyword **_NULL_**, which enables you to use the DATA step without actually creating a SAS data set. A SET statement specifies the SAS data set that you want to read from.

```
data _null_;
   set Sasuser.Stress;
```

- The next step is to specify the output file.

Steps to Create a Raw Data File (2)

Specifying the raw data file (1)

- You use the FILE and PUT statements to write the observations from a SAS data set to a raw data file, just as you used the INFILE and INPUT statements to create a SAS data set. These two sets of statements work almost identically.
- When writing observations to a raw data file, use the FILE statement to specify the output file.
  - General form, FILE statement:

    FILE *file-specification <host-options list>*;
  - where *file-specification* can take the form *fileref* to name a previously defined file reference or '*filename*' to point to the actual name and location of the file.

Steps to Create a Raw Data File (3)

Specifying the raw data file (2)

- For example, if you want to read the **Sasuser.Stress** data set to a raw data file that is referenced by the fileref **Newdat**, you would begin your program with the following SAS statements.

```
data _null_;
   set Sasuser.Stress;
   file newdat;
```

- Instead of identifying the raw data file with a SAS fileref, you can choose to specify the entire filename and location in the FILE statement. For example, the following FILE statement points directly to the **C:\Clinic\Patients\Stress.dat** file.

```
data _null_;
   set Sasuser.Stress;
   file 'c:\clinic\patients\stress.dat';
```

Steps to Create a Raw Data File (4)

Describing the data (1)

- Whereas the FILE statement specifies the output file, the PUT statement describes the lines to write to the raw data file.
  - General form, PUT statement using column output:

    PUT *variable startcol-endcol . . .;*

    where
    - *variable* is the name of the variable whose value is written
    - *startcol* indicates where in the line to begin writing the value
    - *endcol* indicates where in the line to end the value.

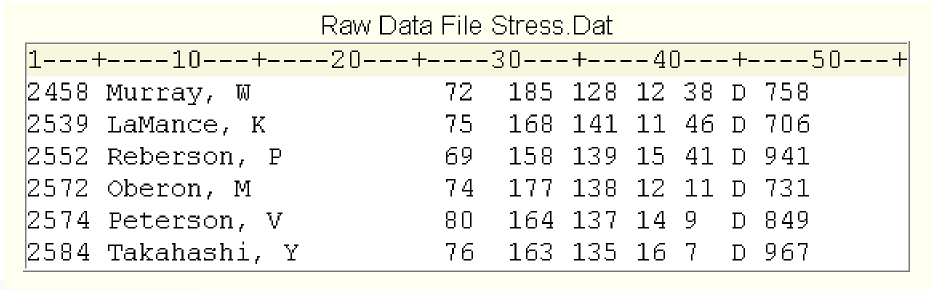
Steps to Create a Raw Data File (5)

Describing the data (2)

➤ In general, the PUT statement mirrors the capabilities of the INPUT statement. In this case you are working with column output. Therefore, you need to specify the variable name, starting column, and ending column for each field that you want to create. Because you are creating raw data, you don't need to follow character variable names with a dollar sign ($).

```
data _null_;
   set Sasuser.Stress;
   file 'c:\clinic\patients\stress.dat';
       put id 1-4          name 6-25     resthr 27-29
           maxhr 31-33     rechr 35-37   timemin 39-40
           timesec 42-43   tolerance 45  totaltime 47-49;
run;
```

➤ The resulting raw data file would look like this:


Raw Data File Stress.Dat

```
1---+----10---+----20---+----30---+----40---+----50---+
2458 Murray, W                72  185 128 12 38 D 758
2539 LaMance, K               75  168 141 11 46 D 706
2552 Reberson, P              69  158 139 15 41 D 941
2572 Oberon, M                74  177 138 12 11 D 731
2574 Peterson, V              80  164 137 14 9  D 849
2584 Takahashi, Y             76  163 135 16 7  D 967
```

Creating and Managing Variables

Contents

- Introduction
- Accumulating totals
- Initializing accumulator variables
- Assign values conditionally
- Specifying lengths for variables
- Subsetting data
- Assigning permanent labels and formats
- Assigning values conditionally using SELECT Groups
- Group statements using DO groups

Introduction (1)

Objective

- You've learned how to create a SAS data set from raw data that is stored in an external file. You've also learned how to subset observations and how to assign values to variables.
- This lesson shows you additional techniques for creating and managing variables. In this lesson, you learn how to create sum variables, assign variable values conditionally, select variables, and assign permanent labels and formats to variables.

Introduction (2)

Obs	ID	Name	RestHR	MaxHR	RecHR	Tolerance	TotalTime	Cumulative Total Seconds (+5,400)	TestLength
1	2458	Murray, W	72	185	128	D	758	6,158	Normal
2	2539	LaMance, K	75	168	141	D	706	6,864	Short
3	2572	Oberon, M	74	177	138	D	731	7,595	Short
4	2574	Peterson, V	80	164	137	D	849	8,444	Long
5	2584	Takahashi, Y	76	163	135	D	967	9,411	Long

Accumulating Totals (1)

- It is often useful to create a variable that accumulates the values of another variable.
- Suppose you want to create the data set Clinic.Stress and to add a new variable, SumSec, to accumulate the total number of elapsed seconds in treadmill stress tests.

SAS Data Set Clinic.Stress (Partial Listing)

ID	Name	RestHr	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime
2458	Murray, W	72	185	128	12	38	D	758
2462	Almers, C	68	171	133	10	5	I	605
2501	Bonaventure, T	78	177	139	11	13	I	673
2523	Johnson, R	69	162	114	9	42	S	582
2539	LaMance, K	75	168	141	11	46	D	706

Accumulating Totals (2)

Example

- To find the total number of elapsed seconds in treadmill stress tests, you need a variable (in this example, SumSec) whose value begins at *0* and increases by the amount of the total seconds in each observation. To calculate the total number of elapsed seconds in treadmill stress tests, you use the Sum statement shown below.

```
data clinic.stress;
     infile tests;
     input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
           RecHR 35-37 TimeMin 39-40 TimeSec 42-43
           Tolerance $ 45;
     TotalTime=(timemin*60)+timesec;
     SumSec+totaltime;
run;
```

Accumulating Totals (3)

- The value of the variable on the left side of the plus sign (here, SumSec) begins at *0* and increases by the value of TotalTime with each observation.

SumSec	=	TotalTime	+	previous total
0				
758	=	758	+	0
1363	=	605	+	758
2036	=	673	+	1363
2618	=	582	+	2036
3324	=	706	+	2618

Initializing Accumulator Variables (1)

- In a previous example, the accumulator variable SumSec was initialized to *0* by default before the first observation was read. But what if you want to initialize SumSec to a different number, such as the total seconds from previous treadmill stress tests?
- You can use the **RETAIN statement** to assign an initial value other than the default value of *0* to a variable whose value is assigned by a Sum statement.
- The RETAIN statement
  - assigns an initial value to a retained variable
  - prevents variables from being initialized each time the DATA step executes.

Initializing Accumulator Variables (2)

- General form, simple RETAIN statement for initializing accumulator variables:

RETAIN *variable initial-value;*

- where
  - *variable* is a variable whose values you want to retain
  - *initial-value* specifies an initial value (numeric or character) for the preceding variable.
- Note **The RETAIN** statement
  - is a compile-time only statement that creates variables if they do not already exist
  - initializes the retained variable to missing before the first execution of the DATA step if you do not supply an initial value
  - has no effect on variables that are read with SET, MERGE, or UPDATE statements. (The SET and MERGE statements are discussed in later chapters.)

Initializing Accumulator Variables (3)

- Example
  - Suppose you want to add 5400 seconds (the accumulated total seconds from a previous treadmill stress test) to the variable SumSec in the **Clinic.Stress** data set when you create the data set. To initialize SumSec with the value *5400*, you use the RETAIN statement shown below:

```
data clinic.stress;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
run;
```

Initializing Accumulator Variables (4)

- Now the value of SumSec begins at *5400* and increases by the value of TotalTime with each observation.

SumSec	=	TotalTime	+	previous total
5400				
6158	=	758	+	0
6763	=	605	+	6158
7436	=	673	+	6763
8018	=	582	+	7436
8724	=	706	+	8018

Assign Values Conditionally (1)

- In the previous section, you created the variable SumSec by using a Sum statement to add total seconds from a treadmill stress test. This time, let's create a variable that categorizes the length of time that a subject spends on the treadmill during a stress test. This new variable, TestLength, will be based on the value of the existing variable TotalTime. The value of TestLength will be assigned conditionally.

Assign Values Conditionally (2)

If TotalTime is . . .	**then TestLength is . . .**
greater than 800	*Long*
750 - 800	*Normal*
less than 750	*Short*

To perform an action conditionally, use an **IF-THEN** statement. The IF-THEN statement executes a SAS statement when the condition in the IF clause is true.

Assign Values Conditionally (3)

- General form, **IF-THEN** statement:

  IF *expression* **THEN** *statement;*

  - where
    - *expression* is any valid SAS expression
    - *statement* is any executable SAS statement.

Assign Values Conditionally (4)

- Example
  - To assign the value *Long* to the variable TestLength when the value of TotalTime is greater than 800, add the following IF-THEN statement to your DATA step:

```
【 data clinic.stress;
     infile tests;
     input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
           RecHR 35-37 TimeMin 39-40 TimeSec 42-43
           Tolerance $ 45;
     TotalTime=(timemin*60)+timesec;
     retain SumSec 5400;
     sumsec+totaltime;
     if totaltime>800 then TestLength='Long';
   run;
```

  - SAS executes the assignment statement only when the condition (TotalTime>800) is true. If the condition is false, then the value of TestLength will be missing.

Assign Values Conditionally (5)

Comparison and logical operators (1)

Operator	Comparison Operation
= or **eq**	equal to
^= or **ne**	not equal to
> or **gt**	greater than
< or **lt**	less than
>= or **ge**	greater than or equal to
<= or **le**	less than or equal to
in	equal to one of a list

Operator	Logical Operation
&	and
|	or
^ or **~**	not

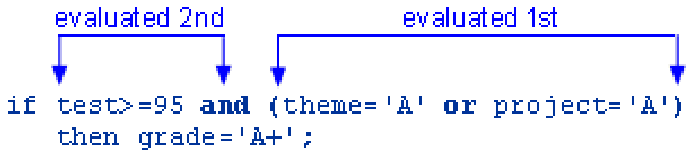
Assign Values Conditionally (6)

- Comparison and logical operators (2)
  - Example
    - Comparison
      - `if test<85 and time<=20 then Status='RETEST';`
      - `if region in ('NE','NW','SW') then Rate=fee-25;`
      - `if target gt 300 or sales ge 50000 then Bonus=salary*.05;`
    - Logical
      - `if status='OK' and type=3 then Count+1;`
      - `if (age^=agecheck | time^=3) & error=1 then Test=1;`
      - `if not (loghours<7500) then Schedule='Quarterly';`
      - `if region not in ('NE','SE') then Bonus=200;`
      - `if status='OK' and type=3 then Count+1; if status='S' or cond='E' then Control='Stop';`
      - `if not(loghours<7500) then Schedule='Quarterly'; if region not in ('NE','SE') then Bonus=200;`

      - Note: the last two example shows that character values must be specified in the same case in which they appear in the data set and must be enclosed in quotation marks.

Assign Values Conditionally (7)

- Logical comparisons that are enclosed in parentheses are evaluated as true or false before they are compared to other expressions. In the example below, the OR comparison in parentheses is evaluated before the first expression and the AND operator are evaluated.



```
if test>=95 and (theme='A' or project='A')
   then grade='A+';
```

Assign Values Conditionally (8)

- In SAS, **any numeric value other than 0 or missing is true, and a value of 0 or missing is false.** Therefore, a numeric variable or expression can stand alone in a condition. If its value is a number other than 0 or missing, the condition is true; if its value is 0 or missing, the condition is false.

0 = False

. = False

1 = True

Assign Values Conditionally (9)

- As a result, you need to **be careful when using the OR operator** with a series of comparisons. Remember that only one comparison in a series of OR comparisons must be true to make a condition true, and any nonzero, nonmissing constant is always evaluated as true. Therefore, the following subsetting IF statement is always true:

```
if x=1 or 2;
```

Assign Values Conditionally (10)

SAS first evaluates **x=1**, and the result can be either true or false; however, since the 2 is evaluated as nonzero and nonmissing (true), the entire expression is true. In this statement, however, the condition is not necessarily true because either comparison can evaluate as true or false:

```
if x=1 or x=2;
```

Assign Values Conditionally (11)

- Providing an alternative action
  - Now suppose you want to assign a value to TestLength based on the other possible values of TotalTime. One way to do this is to add IF-THEN statements for the other two conditions, as shown below.

```
if totaltime>800 then TestLength='Long';
if 750<=totaltime<=800 then TestLength='Normal';
if totaltime<750 then TestLength='Short';
```

  - However, when the DATA step executes, each IF statement is evaluated in order, even if the first condition is true. This wastes system resources and slows the processing of your program.

Assign Values Conditionally (12)

- Providing an alternative action
  - Instead of using a series of IF-THEN statements, you can use the **ELSE** statement to specify an alternative action to be performed when the condition in an IF-THEN statement is false. As shown below, you can write multiple ELSE statements to specify a series of mutually exclusive conditions

```
if totaltime>800 then TestLength='Long';
else if 750<=totaltime<=800 then TestLength='Normal';
else if totaltime<750 then TestLength='Short';
```

Assign Values Conditionally (13)

General form, ELSE statement (1)

ELSE *statement*;

- where *statement* is any executable SAS statement, including another IF-THEN statement.
- So, to assign a value to TestLength when the condition in your IF-THEN statement is false, you can add the ELSE statement to your DATA step, as shown below:

```
data clinic.stress;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
    if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
```

Assign Values Conditionally (14)

General form, ELSE statement (2)

- Using ELSE statements with IF-THEN statements can save resources:
  - Using IF-THEN statements without the ELSE statement causes SAS to evaluate all IF-THEN statements.
  - Using IF-THEN statements with the ELSE statement causes SAS to execute IF-THEN statements until it encounters the first true statement. Subsequent IF-THEN statements are not evaluated.

- For greater efficiency, construct your IF-THEN/ELSE statements with conditions of decreasing probability.

Assign Values Conditionally (15)

General form, ELSE statement (3)

➤ Note Remember that you can use PUT statements to test your conditional logic.

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
   if totaltime>800 then TestLength='Long';
   else if 750<=totaltime<=800 then TestLength='Normal';
   else put 'NOTE: Check this Length: ' totaltime=;
run;
```

Specifying Lengths for Variables (1)

- Previously, you added IF-THEN and ELSE statements to a DATA step in order to create the variable TestLength. Values for TestLength were assigned conditionally, based on the value for TotalTime.

```
data clinic.stress;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
    if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
 run;
```

Specifying Lengths for Variables (2)

- But look what happens when you submit this program.
- During compilation, when creating a new character variable in an assignment statement, SAS allocates as many bytes of storage space as there are characters in the first value that it encounters for that variable. In this case, the first value for TestLength occurs in the IF-THEN statement, which specifies a four-character value (*Long*). So TestLength is assigned a length of 4, and any longer values (*Normal* and *Short*) are truncated**.**

Specifying Lengths for Variables (3)

Variable TestLength
(Partial Listing)

TestLength
Norm
Shor
Shor
Shor
Norm
Shor
Long
...

The example above assigns a character constant as the value of the new variable. The table that follows lists more examples of the default type and length that SAS assigns when the type and length of a variable are not explicitly set.

Specifying Lengths for Variables (4)

Expression	Example	Resulting Type of X	Resulting Length of X	Explanation

Character variable	**length a $ 4;**
	x=a;
		Character variable	4	Length of source variable

Character literal (character constant)	**x='ABC';**
	
	x='ABCDE';
		Character variable	3	Length of first literal (constant) encountered

Concatenation of variables	**length a $ 4**
	
	b $ 6
	c $ 2;
	x=a||b||c;
		Character variable	12	Sum of the lengths of all variables

Concatenation of variables and literal	**length a $ 4;**
	x=a||'CAT';
	x=a||'CATNIP';
		Character variable	7	Sum of the lengths of variables and literals (constants) encountered in first assignment statement

Numeric variable	**length a 4;**
	x=a;
		Numeric variable	8	Default numeric length (8 bytes unless otherwise specified)
				Note: In general, it is not recommended that you change the default length of numeric variables, as this as can affect numeric precision. See the SAS documentation for more information.

Specifying Lengths for Variables (5)

- General form, LENGTH statement
  - **LENGTH** *variable(s)* <$> *length;*
    - Where
      - *variable(s)* names the variable(s) to be assigned a length
      - $ is specified if the variable is a character variable
      - *length* is an integer that specifies the length of the variable.
    - Example

```
length Type $ 8;
length Address1 Address2 Address3 $ 200;
length FirstName $ 12 LastName $ 16;
```

    - Within your program, you include a LENGTH statement to assign a length to accommodate the longest value of the variable TestLength. The longest value is *Normal*, which has six characters. Because TestLength is a character variable, you must follow the variable name with a dollar sign ($).

Specifying Lengths for Variables (6)

```
data clinic.stress;
    infile tests;
    input  ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
    length TestLength $ 6;
    if totaltime>800 then testlength='Long';
    else if 750<=totaltime<=800 then testlength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
```

Variable TestLength (Partial Listing)

TestLength
Norm
Shor
Shor
Shor
Norm
Shor
Long
…

Note: Make sure the LENGTH statement appears before any other reference to the variable in the DATA step. If the variable has been created by another statement, then a later use of the LENGTH statement will not change its size.

Now that you have added the LENGTH statement to your program, the values of TestLength are no longer truncated.

Subsetting Data (1)

Deleting unwanted observations

- So far in this chapter, you've learned to use IF-THEN statements to execute assignment statements conditionally. But you can specify any executable SAS statement in an IF-THEN statement. For example, you can use an **IF-THEN statement** with a **DELETE statement** to determine which observations to omit from the data set that SAS is creating as it reads raw data.
  - The IF-THEN statement executes a SAS statement when the condition in the IF clause is true.
  - The DELETE statement stops processing the current observation.

Subsetting Data (2)

- General form, **DELETE** statement

  DELETE;

- To conditionally execute a DELETE statement, you submit a statement in the following general form:
  - **IF** *expression* **THEN DELETE;**
    - If the *expression* is
      - true**,** the **DELETE** statement executes, and control returns to the top of the DATA step (the observation is deleted).
      - false**,** the **DELETE** statement does not execute, and processing continues with the next statement in the DATA step.

Subsetting Data (3)

- Example
  - ► The IF-THEN and DELETE statements below omit any observations whose values for RestHR are lower than 70.

```
data clinic.stress;
  infile tests;
  input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
        RecHR 35-37 TimeMin 39-40 TimeSec 42-43
        Tolerance $ 45;
  if resthr<70 then delete;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $ 6;
  if totaltime>800 then testlength='Long';
  else if 750<=totaltime<=800 then testlength='Normal';
  else if totaltime<750 then TestLength='Short';
run;
```

Subsetting Data (4)

Selecting Variables with the DROP= and KEEP= Data Set Options

- Sometimes you might need to read and process fields that you don't want to keep in your data set. In this case, you can use the **DROP= and KEEP= data set options** to specify the variables that you want to drop or keep.
- Use the KEEP= option instead of the DROP= option if more variables are dropped than kept. You specify data set options in parentheses after a SAS data set name.

Subsetting Data (5)

- General form, **DROP= and KEEP=** data set options:

  (**DROP=***variable(s)*)

  (**KEEP=***variable(s)*)

  - where
    - the DROP= or KEEP= option, in parentheses, follows the name of the data set that contains the variables to be dropped or kept
    - *variable(s)* identifies the variables to drop or keep.

Subsetting Data (6)

Example

► Suppose you are interested in keeping only the new variable TotalTime and not the original variables TimeMin and TimeSec. You can drop TimeMin and TimeSec when you create the **Stress** data set

```
data clinic.stress(drop=timemin timesec);
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
    if tolerance='D';
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
    length TestLength $ 6;
    if totaltime>800 then testlength='Long';
    else if 750<=totaltime<=800 then testlength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
```

Subsetting Data (7)

Another way to exclude variables from your data set is to use the **DROP statement** or the **KEEP statement**. Like the DROP= and KEEP= data set options, these statements drop or keep variables. However, the DROP statement differs from the DROP= data set option in the following ways:

- You cannot use the DROP statement in SAS procedure steps.
- The DROP statement applies to all output data sets that are named in the DATA statement.
- To exclude variables from some data sets but not from others, place the appropriate DROP= data set option next to each data set name that is specified in the DATA statement.

The KEEP statement is similar to the DROP statement, except that the KEEP statement specifies a list of variables to write to output data sets. Use the KEEP statement instead of the DROP statement if the number of variables to keep is significantly smaller than the number to drop.

Subsetting Data (8)

- General form, **DROP and KEEP** statements:

  DROP *variable(s);*

  KEEP *variable(s);*

  - where *variable(s)* identifies the variables to drop or keep.

Subsetting Data (9)

- Example

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   if tolerance='D';
   drop timemin timesec;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
   length TestLength $ 6;
   if totaltime>800 then testlength='Long';
   else if 750<=totaltime<=800 then testlength='Normal';
   else if totaltime<750 then TestLength='Short';
run;
```

Assigning Permanent Labels and Formats (1)

- At this point, you've read and manipulated your raw data to obtain the observations, variables, and variable values that you want. Your final task in this chapter is to permanently assign **labels** and **formats** to variables.

Assigning Permanent Labels and Formats (2)

- Example

```
data clinic.stress;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
    if resthr<70 then delete;
    if tolerance='D';
    drop timemin timesec;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
    length TestLength $ 6;
    if totaltime>800 then testlength='Long';
    else if 750<=totaltime<=800 then testlength='Normal';
    else if totaltime<750 then TestLength='Short';
    label sumsec='Cumulative Total Seconds (+5,400)';
    format sumsec comma6.;
run;
```

Assigning Values Conditionally Using SELECT Groups (1)

- Earlier in this chapter, you learned to assign values conditionally by using IF-THEN/ELSE statements. You can also use **SELECT groups** in DATA steps to perform conditional processing.
- A SELECT group contains these statements:

This statement...	**Performs this action...**
SELECT	begins a SELECT group.
WHEN	identifies SAS statements that are executed when a particular condition is true.
OTHERWISE (optional)	specifies a statement to be executed if no WHEN condition is met.
END	ends a SELECT group.

Assigning Values Conditionally Using SELECT Groups (2)

- You can decide whether to use IF-THEN/ELSE statements or SELECT groups based on the following criteria.
- When you have a **long series** of mutually exclusive conditions and the comparison is numeric, using a SELECT group is slightly more efficient than using a series of IF-THEN or IF-THEN/ELSE statements because CPU time is reduced. SELECT groups also make the program easier to read and debug.
- For programs with **few conditions**, use IF-THEN/ELSE statements.

Assigning Values Conditionally Using SELECT Groups (3)

- General form, SELECT group:
  - **SELECT** <(*select-expression*)>;
    - **WHEN-1** (*when-expression-1 <..., when-expression-n>*) *statement;*
    - **WHEN-*n*** (*when-expression-1 <..., when-expression-n>*) *statement;*
    - <**OTHERWISE** *statement;*>

    END;

    where
    - **SELECT** begins a SELECT group.
    - the optional *select-expression* specifies any SAS expression that evaluates to a single value.
    - **WHEN** identifies SAS statements that are executed when a particular condition is true.
    - *when-expression* specifies any SAS expression, including a compound expression. You
    - must specify at least one *when-expression*.
    - *statement* is any executable SAS statement. You must specify the *statement* argument.
    - the optional **OTHERWISE** statement specifies a statement to be executed if no WHEN condition is met.
    - **END** ends a SELECT group.

Assigning Values Conditionally Using SELECT Groups (4)

- Example: select group in a data step

```
data emps(keep=salary group);
   set sasuser.payrollmaster;
   length Group $ 20;
   select(jobcode);
        when ("FA1") group="Flight Attendant I";
        when ("FA2") group="Flight Attendant II";
        when ("FA3") group="Flight Attendant III";
        when ("ME1") group="Mechanic I";
        when ("ME2") group="Mechanic II";
        when ("ME3") group="Mechanic III";
        when ("NA1") group="Navigator I";
        when ("NA2") group="Navigator II";
        when ("NA3") group="Navigator III";
        when ("PT1") group="Pilot I";
        when ("PT2") group="Pilot II";
        when ("PT3") group="Pilot III";
        when ("TA1","TA2","TA3") group="Ticket Agents";
        otherwise group="Other";
   end;
run;
```

Assigning Values Conditionally Using SELECT Groups (5)

- Specifying SELECT statements with expressions
  - As you saw in the general form for SELECT groups, you can optionally specify a *selectexpression* in the SELECT statement. The way SAS evaluates a *when-expression* depends on whether you specify a *select-expression*.
  - If you **do** specify a *select-expression* in the SELECT statement, SAS compares the value of the *select-expression* with the value of each *when-expression*. That is, SAS evaluates the *selectexpression* and *when-expression,* compares the two for equality, and returns a value of true or false.
    - If the comparison is **true,** SAS executes the *statement* in the WHEN statement.
    - If the comparison is **false,** SAS proceeds either to the next *when-expression* in the current WHEN statement, or to the next WHEN statement if no more expressions are present. If no WHEN statements remain, execution proceeds to the OTHERWISE statement, if one is present.

Assigning Values Conditionally Using SELECT Groups (6)

- Specifying SELECT statements with expressions
  - **Warning** If the result of all SELECT-WHEN comparisons is false and no OTHERWISE statement is present, SAS issues an **error message** and **stops executing** the DATA step.
  - In the following SELECT group, SAS determines the value of toy and compares it to values in each WHEN statement in turn. If a WHEN statement is true compared to the toy value, then SAS assigns the related price and continues processing the rest of the DATA step. If none of the comparisons is true, then SAS executes the OTHERWISE statement and writes a debugging message to the SAS log.

```
select (toy);
    when ("Bear") price=35.00;
    when ("Violin") price=139.00;
    when ("Top","Whistle","Duck") price=7.99;
    otherwise put "Check unknown toy: " toy=;
end;
```

Assigning Values Conditionally Using SELECT Groups (7)

- Specifying SELECT Statements without Expressions
- If you don't specify a *select-expression*, SAS evaluates each *when-expression* to produce a result of true or false.
  - If the result is **true**, SAS executes the *statement* in the WHEN statement.
  - If the result is **false**, SAS proceeds either to the next *when-expression* in the current
    - WHEN statement, or to the next WHEN statement if no more expressions are present, or to the OTHERWISE statement if one is present. (That is, SAS performs the action that is indicated in the first true WHEN statement.)

Assigning Values Conditionally Using SELECT Groups (8)

- Specifying SELECT statements without expressions
  - If more than one WHEN statement has a true *when-expression*, **only the first** WHEN statement is used; once a *when-expression* is true, no other *when-expressions* are evaluated.
    - **Warning** If the result of all *when-expressions* is false and no OTHERWISE statement is present, SAS issues an **error message.**
  - In the example below, the SELECT statement does not specify a *select-expression*. The WHEN statements are evaluated in order, and only one is used. For example, if the value of toy is *Bear* and the value of month is *FEB*, only the second WHEN statement is used, even though the condition in the third WHEN statement is also met. In this case, the variable price is assigned the value *25.00*.

```
select;
   when (toy="Bear" and month in ('OCT', 'NOV', 'DEC'))
   price=45.00;
   when (toy="Bear" and month in ('JAN', 'FEB'))
   price=25.00;
   when (toy="Bear") price=35.00;
   otherwise;
end;
```

Group Statements Using DO Groups (1)

- So far in this chapter, you've seen examples of conditional processing (IF-THEN/ELSE statements and SELECT groups) that execute only a single SAS statement when a condition is true. However, you can also execute a group of statements as a unit by using **DO groups**.
- To construct a DO group, you use the DO and END statements along with other SAS statements.

Group Statements Using DO Groups (2)

- General form, simple DO group

```
DO;
      SAS statements
END;
```

- where
  - the **DO** statement begins DO-group processing
  - *SAS statements* between the DO and END statements are called a DO group and execute as a unit
  - the **END** statement terminates DO-group processing.
  - Note: You can nest DO statements within DO groups.

Group Statements Using DO Groups (3)

- Example

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
   length TestLength $ 6 Message $ 20;
   if totaltime>800 then
       do;
         testlength='Long';
         message='Run blood panel';
       end;
   else if 750<=totaltime<=800 then testlength='Normal';
   else if totaltime<750 then TestLength='Short';
run;
```

Group Statements Using DO Groups (4)

- Indenting and nesting DO groups
  - You can nest DO groups to any level, just like you nest IF-THEN/ELSE statements. (The memory capabilities of your system might limit the number of nested DO statements that you can use. For details, see the SAS documentation about how many levels of nested DO statements your system's memory can support.)

- The following is an example of nested DO groups:

```
do;
   statements;
   do;
      statements;
      do;
         statements;
      end;
   end;
end;
```

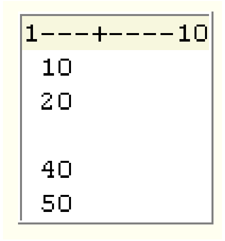
Group Statements Using DO Groups (5)

- There are three other forms of the DO statement
  - The **iterative DO statement** executes statements between DO and END statements repetitively based on the value of an index variable. The iterative DO statement can contain a WHILE or UNTIL clause.
  - The **DO UNTIL statement** executes statements in a DO loop repetitively until a condition is true, checking the condition after each iteration of the DO loop.
  - The **DO WHILE statement** executes statements in a DO loop repetitively while a condition is true, checking the condition before each iteration of the DO loop.

Quiz

Now consider the revised program below. What is the value of Count after the third observation is read?

```
data work.newnums;
   infile numbers;
   input Tens 2-3;
   retain Count 100;
   count+tens;
run;
```



```
1---+----10
 10
 20

 40
 50
```

- a.missing   b.0   c.100   d.130

- Correct answer: d
  - The RETAIN statement assigns an initial value of 100 to the variable Count, so the value of Count in the third observation would be 100+10+20+0, or 130.

Quiz

What is the length of the variable Type, as created in the DATA step below?

```
data finance.newloan;
   set finance.records;
   TotLoan+payment;
   if code='1' then Type='Fixed';
   else Type='Variable';
   length type $ 10;
run;
```

- a.5 b.8 c.10 d. it depends on the first value of Type

- Correct answer: a
  - The length of a new variable is determined by the first reference in the DATA step, not by data values. In this case, the length of Type is determined by the value *Fixed*. The LENGTH statement is in the wrong place; it must be read **before** any other reference to the variable in the DATA step. The LENGTH statement cannot change the length of an existing variable.

Access to PC Files

Contents

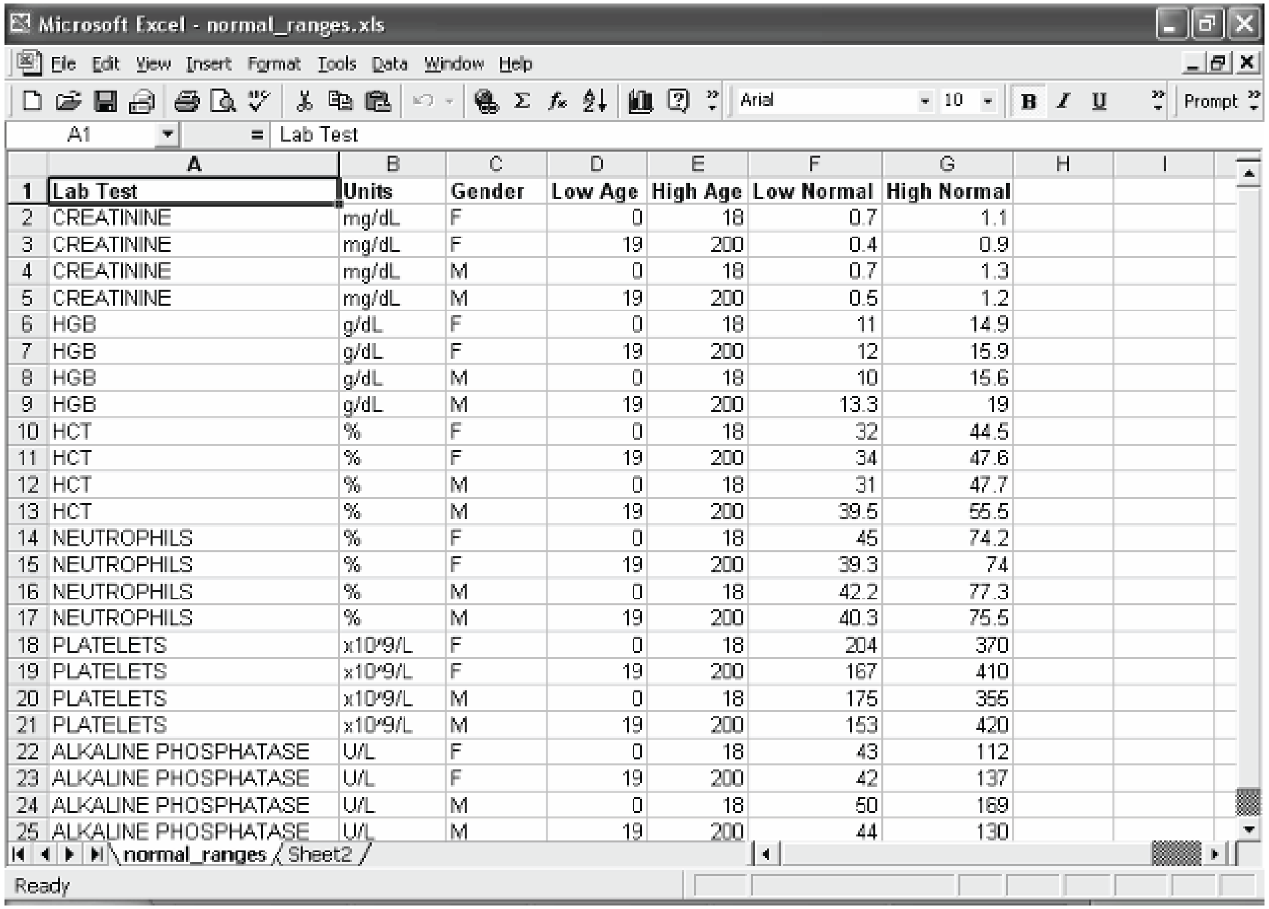
- Introduction
- LIBNAME statement
- Import Wizard and PROC IMPORT
- SAS/ACCESS SQL Pass-Through Facility

Introduction (1)

- SAS/ACCESS for PC files enables you to read data from PC files, to use that data in SAS reports or applications, and to use SAS data sets to create PC files in various formats.
- Because Microsoft Office is so widely used, it is sometimes necessary for you to import data directly from Microsoft Excel or Microsoft Access. Here in this section, we will take PC files of Microsoft Excel or Microsoft Access as example.
- SAS provides several ways to read Microsoft Excel and Access files. Some commonly used SAS tools include:
  - the LIBNAME statement
  - the Import Wizard/PROC IMPORT
  - the SQL Pass-Through Facility
  - SAS Enterprise Guide.

Introduction (2)

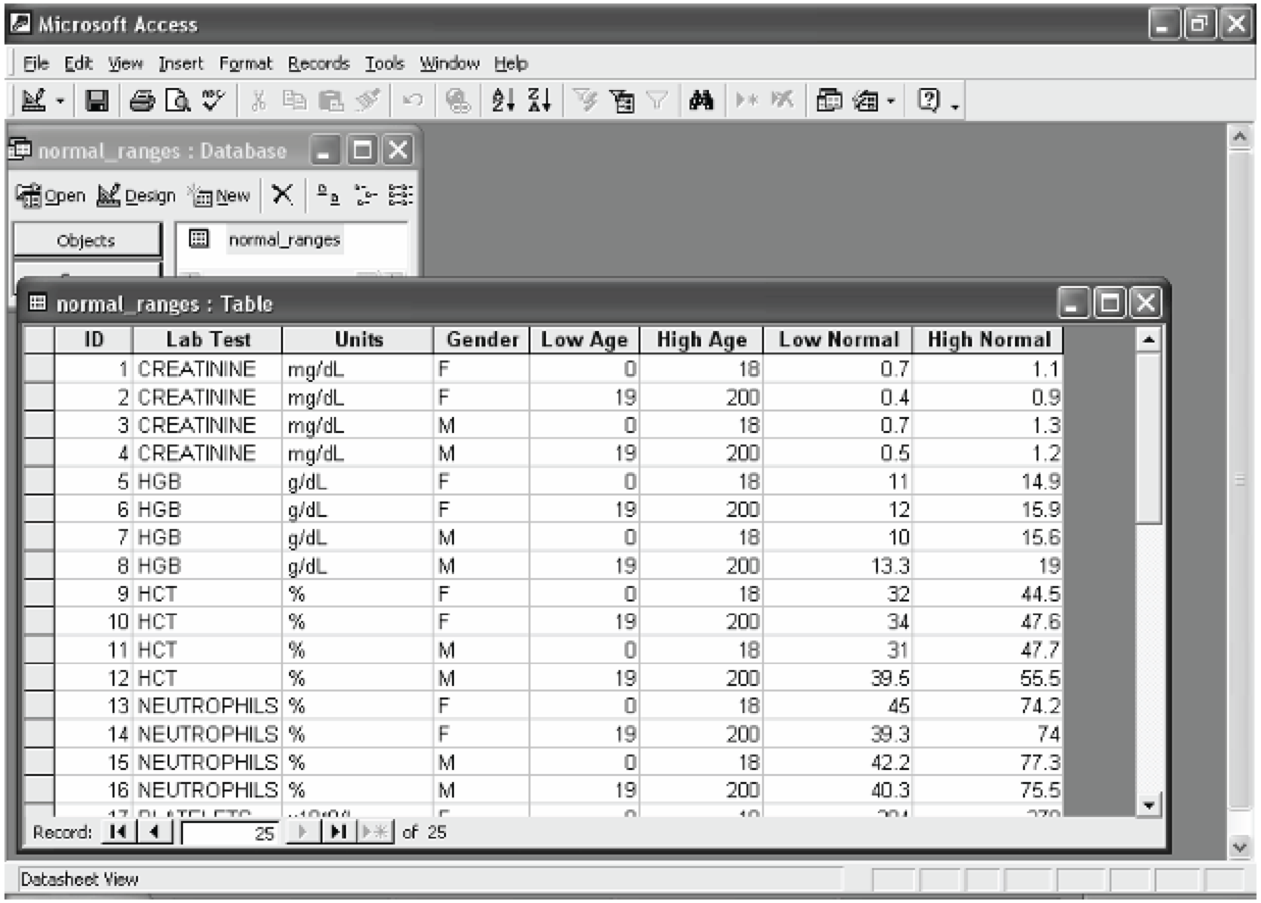
- In Microsoft Excel, the lab normal data file might look like the following:


Microsoft Excel - normal_ranges.xls
File Edit View Insert Format Tools Data Window Help
Arial 10 Prompt
A1 = Lab Test
normal_ranges Sheet2
Ready

	A	B	C	D	E	F	G
1	Lab Test	Units	Gender	Low Age	High Age	Low Normal	High Normal
2	CREATININE	mg/dL	F	0	18	0.7	1.1
3	CREATININE	mg/dL	F	19	200	0.4	0.9
4	CREATININE	mg/dL	M	0	18	0.7	1.3
5	CREATININE	mg/dL	M	19	200	0.5	1.2
6	HGB	g/dL	F	0	18	11	14.9
7	HGB	g/dL	F	19	200	12	15.9
8	HGB	g/dL	M	0	18	10	15.6
9	HGB	g/dL	M	19	200	13.3	19
10	HCT	%	F	0	18	32	44.5
11	HCT	%	F	19	200	34	47.6
12	HCT	%	M	0	18	31	47.7
13	HCT	%	M	19	200	39.5	55.5
14	NEUTROPHILS	%	F	0	18	45	74.2
15	NEUTROPHILS	%	F	19	200	39.3	74
16	NEUTROPHILS	%	M	0	18	42.2	77.3
17	NEUTROPHILS	%	M	19	200	40.3	75.5
18	PLATELETS	x10^9/L	F	0	18	204	370
19	PLATELETS	x10^9/L	F	19	200	167	410
20	PLATELETS	x10^9/L	M	0	18	175	355
21	PLATELETS	x10^9/L	M	19	200	153	420
22	ALKALINE PHOSPHATASE	U/L	F	0	18	43	112
23	ALKALINE PHOSPHATASE	U/L	F	19	200	42	137
24	ALKALINE PHOSPHATASE	U/L	M	0	18	50	169
25	ALKALINE PHOSPHATASE	U/L	M	19	200	44	130

Introduction (3)

- In Microsoft Access the lab normal data might look like this:



ID	Lab Test	Units	Gender	Low Age	High Age	Low Normal	High Normal
1	CREATININE	mg/dL	F	0	18	0.7	1.1
2	CREATININE	mg/dL	F	19	200	0.4	0.9
3	CREATININE	mg/dL	M	0	18	0.7	1.3
4	CREATININE	mg/dL	M	19	200	0.5	1.2
5	HGB	g/dL	F	0	18	11	14.9
6	HGB	g/dL	F	19	200	12	15.9
7	HGB	g/dL	M	0	18	10	15.6
8	HGB	g/dL	M	19	200	13.3	19
9	HCT	%	F	0	18	32	44.5
10	HCT	%	F	19	200	34	47.6
11	HCT	%	M	0	18	31	47.7
12	HCT	%	M	19	200	39.5	55.5
13	NEUTROPHILS	%	F	0	18	45	74.2
14	NEUTROPHILS	%	F	19	200	39.3	74
15	NEUTROPHILS	%	M	0	18	42.2	77.3
16	NEUTROPHILS	%	M	19	200	40.3	75.5
17	[illegible]	[illegible]	F	0	[illegible]	[illegible]	[illegible]

LIBNAME Statement (1)

Beginning with SAS 9.1, the LIBNAME statement can be used to simply map to an Excel or Access database. For example, the following SAS code reads in and then prints the lab normal file normal_ranges.xls.

```
libname xlsfile EXCEL "C:\normal_ranges.xls";
proc contents
   data = xlsfile._all_;
run;
proc print
   data = XLSFILE.'normal_ranges$'n;
run;
```

- Note that the "EXCEL" engine specification is optional, because SAS would read the ".xls" extension in the physical filename and assume it indicates a Microsoft Excel file.
- Also note that the "xlsfile" libref refers to the entire Excel workbook. In the subsequent PROC PRINT, the "normal_ranges" must be specified so SAS will know which Excel worksheet to read.
- The data set/worksheet name in the PROC PRINT looks odd because of the existence of a special "$" character, which is normally not allowed as part of a data set name.

LIBNAME Statement (2)

The normals_ranges.mdb Microsoft Access file could be read in with the following similar SAS code.

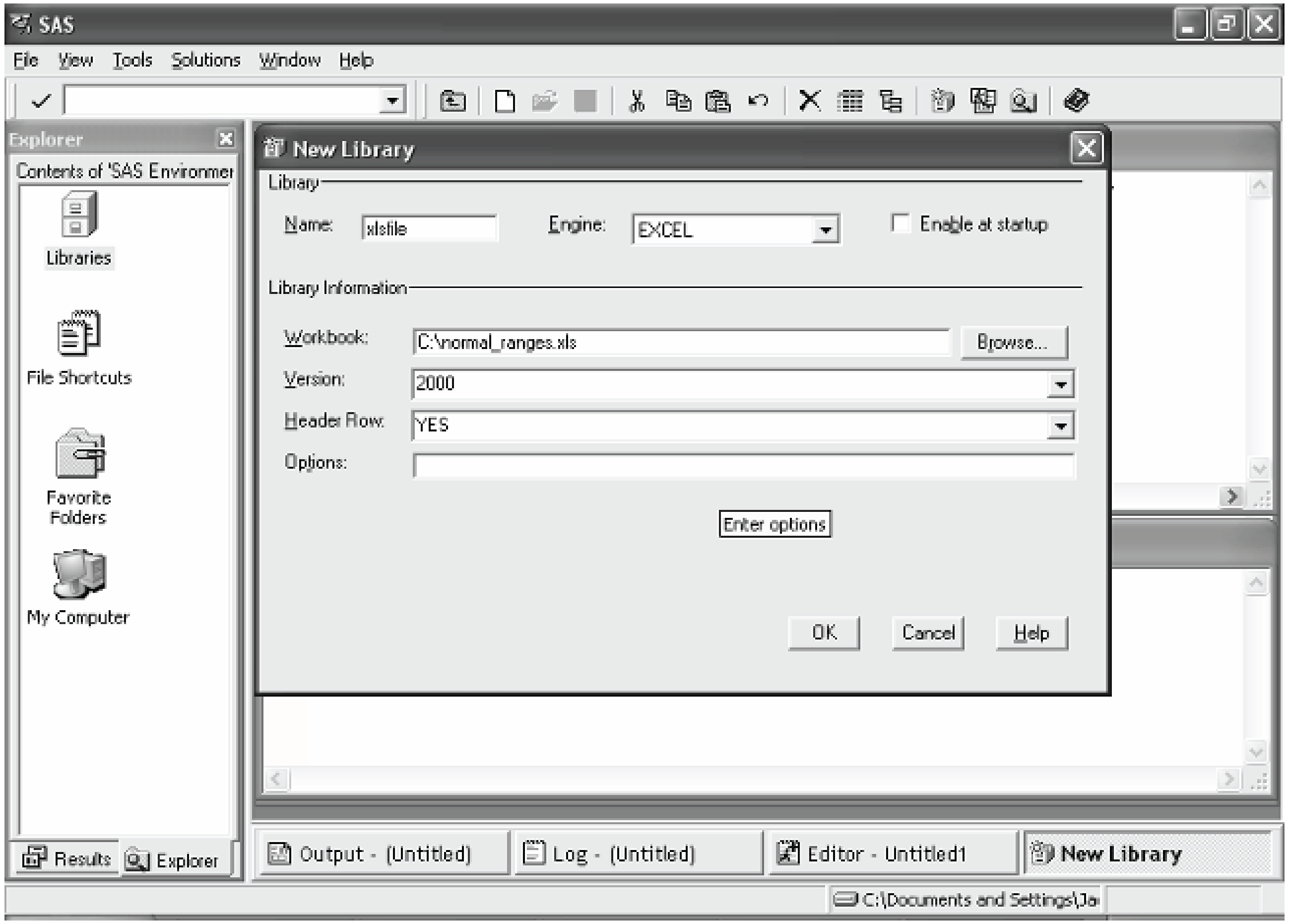
```
libname accfile ACCESS "C:\normal_ranges.mdb";
proc contents
   data = accfile._all_;
run;
proc print
   data = accfile.normal_ranges;
run;
```

- Again, the “ACCESS” specification as a LIBNAME engine is optional, as the libref would default to Microsoft Access because “.mdb” is in the physical filename.
- Note that the ACCESS LIBNAME engine seems by default to import all text fields as 255 characters in length.
- Also note that all dates that come from Microsoft Access via the ACCESS engine are represented in SAS as SAS datetime fields. This is because Access has only datetime fields compared with SAS, which has date, time, and datetime variables.

LIBNAME Statement (3)

- SAS ACCESS and EXCEL librefs can be specified interactively by right-clicking on the Libraries icon in the SAS Explorer window and completing the parameters in the New Library window. You can define a libref called xlsfile that points to normal_ranges like following picture.

LIBNAME Statement (4)



Library

Name: xlsfile

Engine: EXCEL

Enable at startup

Library Information

Workbook: C:\normal_ranges.xls

Version: 2000

Header Row: YES

Options:

Be aware that the LIBNAME statement approach allows for both reading and writing to and from Microsoft Office files, which means the contents of the Microsoft Office files can be changed by SAS.

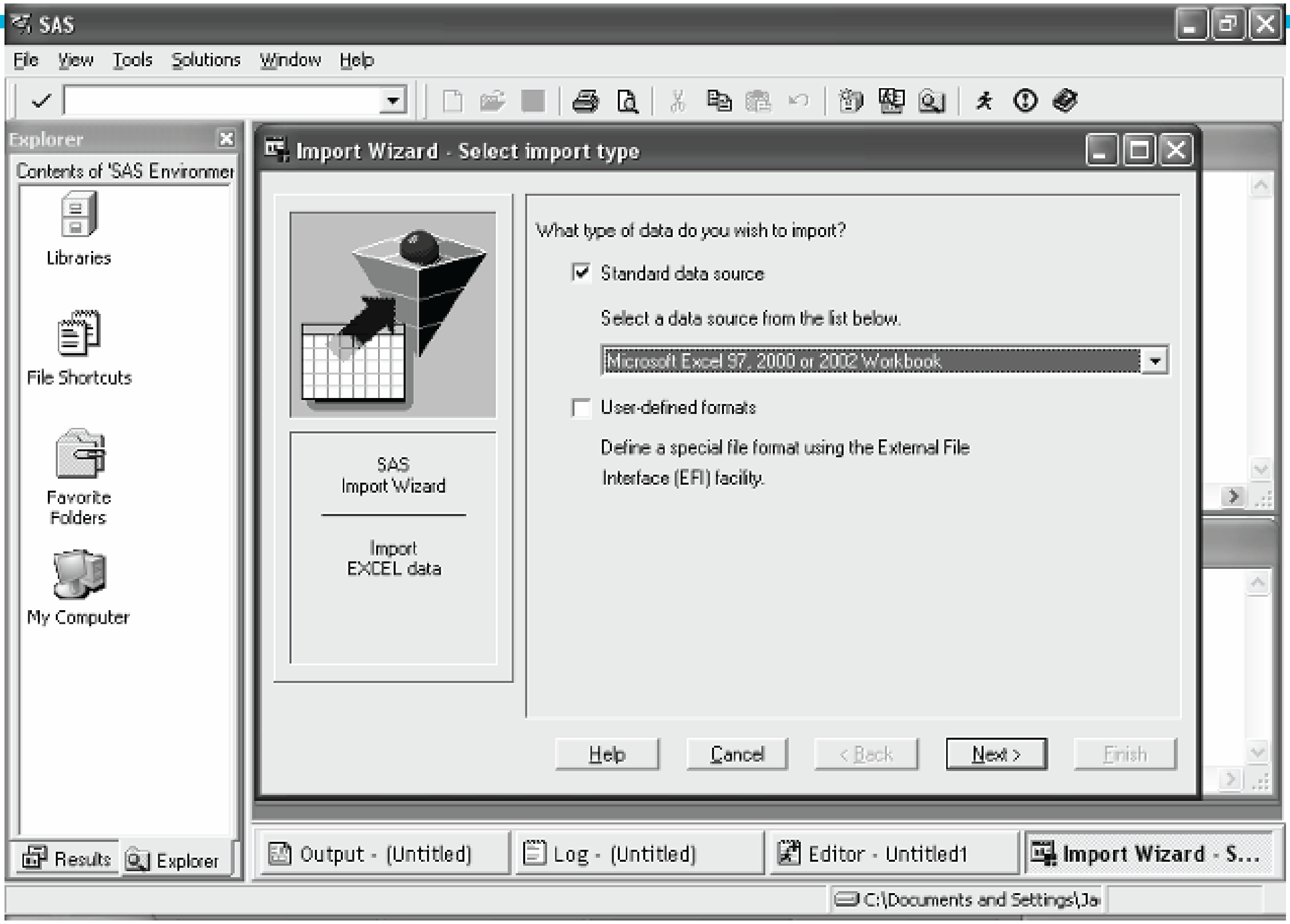
北京生物统计与数据管理联合会
Beijing Biometric Association

Import Wizard and PROC IMPORT (1)

- The interactive SAS Import Wizard provides an easy way to import the contents of Microsoft Excel and Access files into SAS. Here again, the Import Wizard is essentially a graphical user interface that builds the PROC IMPORT code for you.

- Begin in the interactive SAS windowing environment by selecting “File” from the toolbar and then “Import Data…” from the drop-down menu. A window like the following will appear, where you can select Microsoft Excel as a standard data source.

Import Wizard and PROC IMPORT (2)



What type of data do you wish to import?

[✓] Standard data source

Select a data source from the list below.

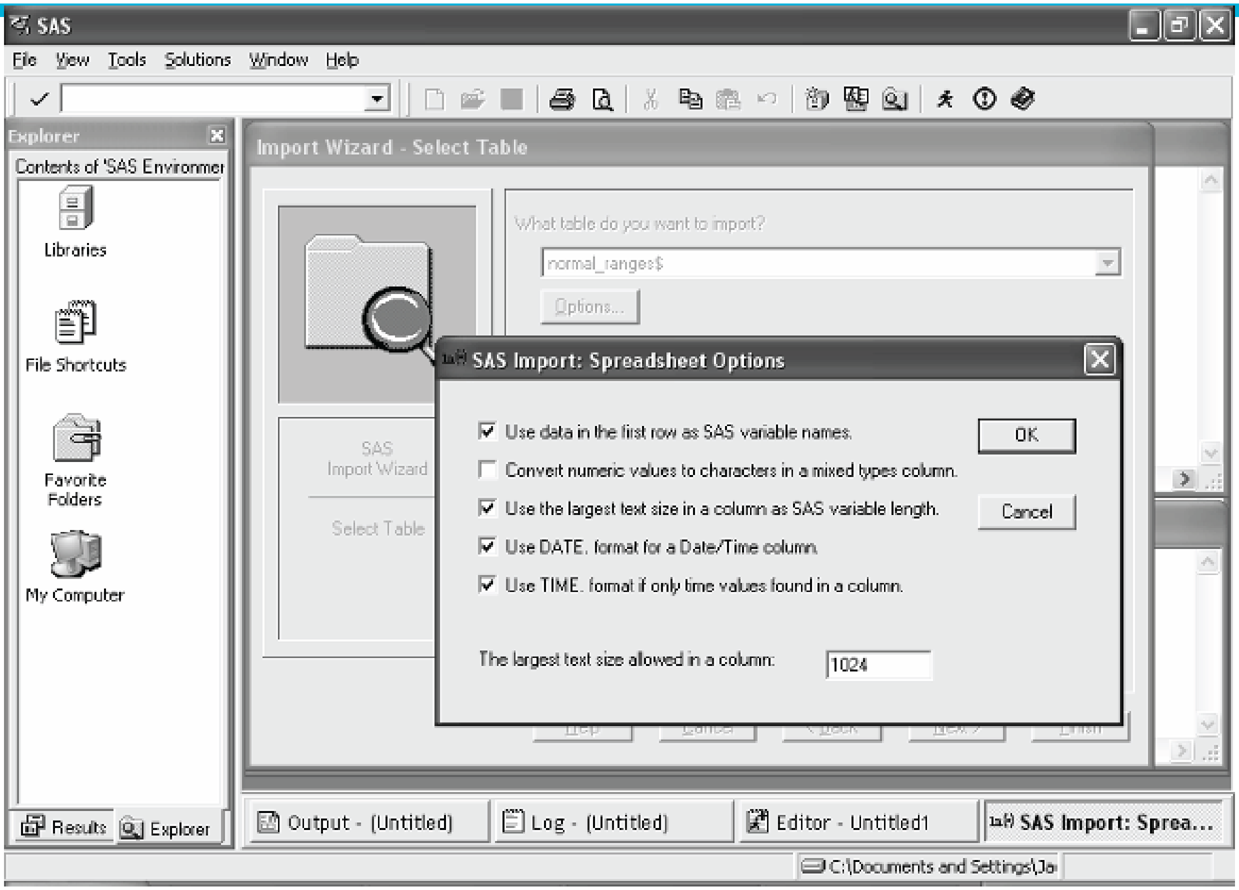
Microsoft Excel 97, 2000 or 2002 Workbook

[ ] User-defined formats

Define a special file format using the External File Interface (EFI) facility.

Click "Next," and a file browser window will open that allows for the drill-down and selection of the Microsoft Excel file of interest. Once the file is selected, a Select Table window will open. This window allows you to pick which worksheet in the Excel file you want to turn into a SAS data set. Click the "Options" button to see the new options available with SAS 9.1 and PROC IMPORT.

Import Wizard and PROC IMPORT (3)



SAS Import: Spreadsheet Options

- [✓] Use data in the first row as SAS variable names.
- [ ] Convert numeric values to characters in a mixed types column.
- [✓] Use the largest text size in a column as SAS variable length.
- [✓] Use DATE. format for a Date/Time column.
- [✓] Use TIME. format if only time values found in a column.

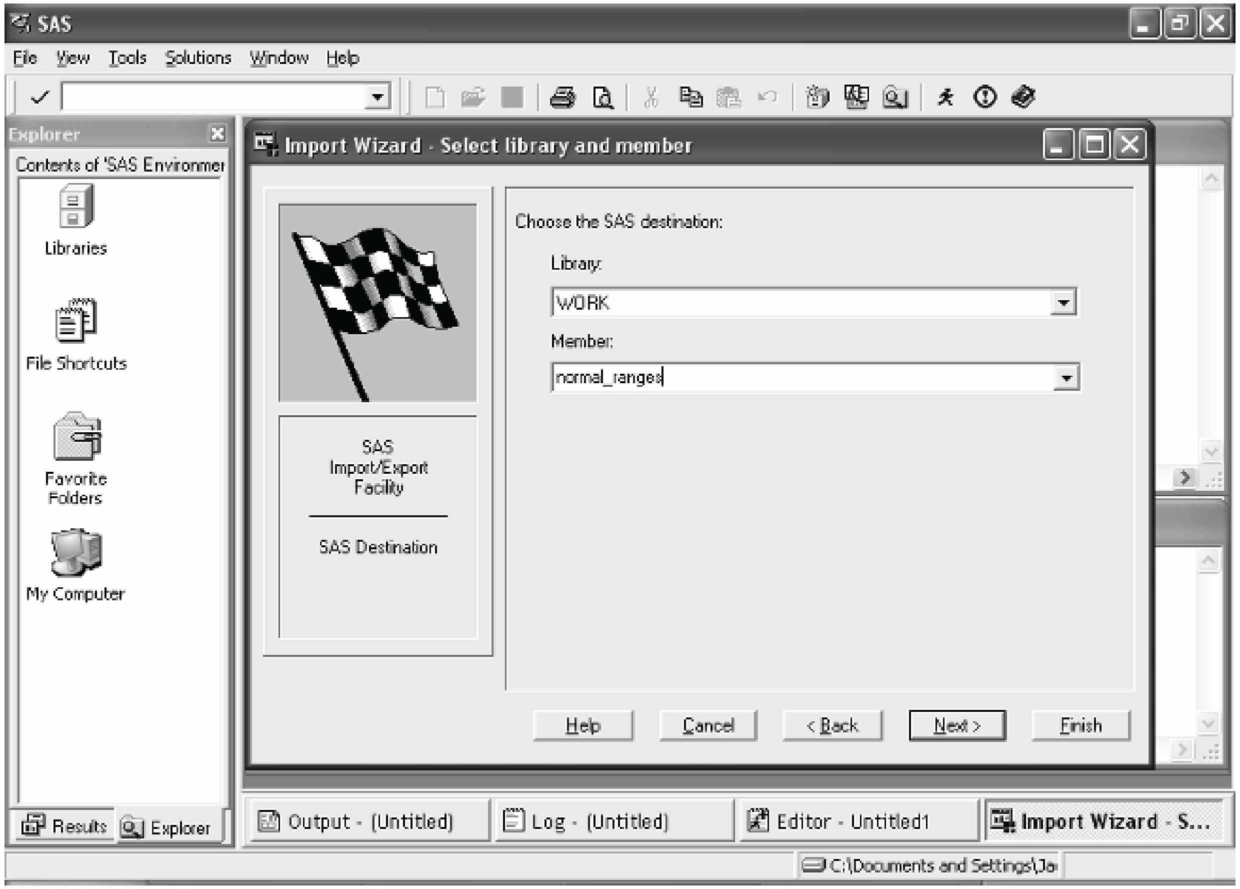
The largest text size allowed in a column: 1024

OK

Cancel

Note that the default options were chosen in the preceding window. We will look at those further when we explore the subsequent PROC IMPORT code. Now, click “OK” in the Spreadsheet Options window and then click “Next” in the Select Table window. The Select Library and Member window opens, which allows for the selection of a SAS library and data set name as follows.

Import Wizard and PROC IMPORT (4)



Choose the SAS destination:

Library: WORK

Member: normal_ranges

Click "Next" and SAS will prompt you to see if you want to save the PROC IMPORT code generated by the Import Wizard. Click "Finish" to complete the file import.

Import Wizard and PROC IMPORT (5)

Here is the PROC IMPORT code generated by SAS from this run.

```
PROC IMPORT OUT= WORK.normal_ranges
            DATAFILE= "C:\normal_ranges.xls"
            DBMS=EXCEL REPLACE;
   SHEET="normal_ranges$";
   GETNAMES=YES;
   MIXED=NO;
   SCANTEXT=YES;
   USEDATE=YES;
   SCANTIME=YES;
RUN;
```

- If the options on PROC IMPORT do not produce what is desired, they can be changed and resubmitted, or the code can be saved to edit and run in batch mode later.

Import Wizard and PROC IMPORT (6)

Here are the new options available with SAS 9.1 along with my recommended settings.

Option	Purpose
DBSASLABEL	By default, the SAS label for an imported variable is set to the column name. Setting DBSASLABEL=NONE places null values into the SAS labels.
MIXED	Converts numeric values to character values if a column displays numeric and character text cells. Note that the default here is NO. Keep in mind that SAS scans only the first eight rows of the Excel column to determine whether the column is numeric or character. If SAS picks character and there are numeric cells later, then those will be set to blank. For this reason consider setting MIXED=YES.
SCANTEXT	When set to YES, this option tells SAS to scan the entire column to determine the width of the column. Always leave this set to YES.
SCANTIME	When set to YES, this option applies a SAS time format to a field if it appears to contain only time entries.
TEXTSIZE	This option hardcodes the maximum width of a character variable. It overrides SCANTEXT=YES.
USEDATE	When set to YES, this option formats SAS datetime fields with a date format. If you prefer to use datetime formats with datetime fields, set USEDATE=NO.

► The Import Wizard process for Microsoft Access files works like the one for Excel files and produces similar PROC IMPORT code. Keep in mind that text fields get a default length of 255 characters when PROC IMPORT is used with Microsoft Access files.

SAS/ACCESS SQL Pass-Through Facility (1)

The SAS/ACCESS SQL Pass-Through Facility is another way for SAS to dynamically establish a connection to Microsoft Excel or Access files. You can connect to the Microsoft Excel file normal_ranges.xls by using the following SAS code.

```
**** OBTAIN AVAILABLE WORKSHEET NAMES FROM EXCEL FILE;
proc sql;
   connect to excel (path = "C:\normal_ranges.xls");
   select table_name from connection to excel(jet::tables);
quit;
**** GO GET NORMAL_RANGES WORKSHEET FROM EXCEL FILE;
proc sql;
   connect to EXCEL (path = "C:\normal_ranges.xls" header = yes
                     mixed = yes version = 2000 );
   create table normal_ranges as
      select * from connection to excel
      (select * from [normal_ranges$]);
   disconnect from excel;
quit;
```

► Study the preceding SAS code. Notice how the first SQL step uses a special Microsoft Jet Engine query to obtain the names of the worksheet in normal_ranges.xls. Also note that the SQL step that fetches the normal_ranges worksheet from normal_ranges.xls does so by placing the worksheet in braces in the inner SELECT statement.

SAS/ACCESS SQL Pass-Through Facility (2)

The following SAS code uses the SQL Pass-Through Facility to connect to the Microsoft Access file normal_ranges.mdb.

```
*** OBTAIN AVAILABLE TABLE NAMES FROM ACCESS FILE;
proc sql;
   connect to access (path = "C:\normal_ranges.mdb");
   select table_name from connection to access(jet::tables);
quit;
**** GO GET NORMAL_RANGES WORKSHEET FROM ACCESS FILE;
proc sql;
   connect to access (path="C:\normal_ranges.mdb");
   create table normal_ranges as
      select * from connection to access
      (select * from normal_ranges);
   disconnect from access;
quit;
```

Note that the SQL Pass-Through Facility to Microsoft Excel and Access files does default to 255 characters in length for character fields.

Access to Oracle

Contents

- Introduction
- SAS/ACCESS SQL Pass-Through Facility
- SAS/ACCESS LIBNAME statement

Introduction

- Importing relational databases and clinical data management systems
  - Most **clinical data management systems** used for clinical trials today store their data in relational database software such as Oracle or Microsoft SQL Server.
  - A **relational database** is composed of a set of rectangular data matrices called “tables” that relate or associate with one another by certain key fields.
  - The language most often used to work with relational databases is *structured query language* (*SQL*). The SAS/ACCESS SQL Pass-Through Facility and the SAS/ACCESS LIBNAME engine are the two methods that SAS provides for extracting data from relational databases.

SAS/ACCESS SQL Pass-Through Facility (1)

The SAS/ACCESS SQL Pass-Through Facility has long been one of the only ways of getting data out of a relational database. It is still a flexible means of obtaining relational database data, as it allows for using SAS SQL as a means of filtering or modifying data on the way into SAS.

```
proc sql;
   connect to oracle as oracle_tables
          (user = USERID orapw = PASSWORD path = "INSTANCE");

   create table AE as
      select * from connection to oracle_tables
      (select * from AE_ORACLE_TABLE );

   disconnect from oracle_tables;
quit;
```

SAS/ACCESS SQL Pass-Through Facility (2)

- This code simply extracts the data from the table named “AE_ORACLE_TABLE” within Oracle and places it in its entirety in a SAS work library data set called AE.

- The USER, ORAPW, and PATH parameters are specific to the Oracle database settings at a particular site, so you would need to consult an Oracle database administrator to get the proper values.

- The good thing about the SQL Pass-Through Facility is that it lets you create a more desirable SAS data set with some slight modifications.

SAS/ACCESS SQL Pass-Through Facility (3)

```
proc sql;
   connect to oracle as oracle_tables
     (user = USERID orapw = PASSWORD path ="INSTANCE");

   create table library.AE as
      select * from connection to oracle_tables
      (select subject, verbatim, ae_date, pt_text
          from AE_ORACLE_TABLE
          where query_clean=”YES”);
   disconnect from oracle_tables;
quit;
```

► Notice how the highlighted changes allow for a permanent SAS data set to be created containing only the variables desired and only the records that have all data queries resolved by data management.

SAS/ACCESS LIBNAME statement (1)

- The Oracle specific syntax for the LIBNAME statement is:

  LIBNAME *libref* **oracle** *<connection-options> <LIBNAME - options>*

  - *libref* is any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.
  - oracle is the SAS/ACCESS engine name for the interface to Oracle.
  - *connection – options* provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS.
  - *LIBNAME – options* define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine locking or naming behavior.

SAS/ACCESS LIBNAME statement (2)

- Beginning with SAS 7, a new SAS/ACCESS LIBNAME statement interface was available for accessing data in relational databases.

- For example, the previous example of the SQL Pass-Through Facility can be distilled to the following LIBNAME statement and associated DATA step.

```
libname oratabs oracle user=USERNAME
        orapw = PASSWORD path = "@INSTANCE" schema = TRIALNAME;
data adverse;
   set oratabs.AE_ORACLE_TABLE;
       where query_clean = “YES”;
       keep subject verbatim ae_date pt_text;
run;
```

SAS/ACCESS LIBNAME statement (3)

- In this program the “oratabs” libref allows all of the tables found in that Oracle data instance to be treated like SAS data sets. This is a simple and fast way of accessing relational databases, and it requires no knowledge of SQL to implement.

- Although the preceding examples import Oracle data, SAS/ACCESS can be used to access quite a number of relational databases, including Oracle, Microsoft SQL Server.

Access to SQL

Contents

- SAS/ACCESS SQL Pass-Through Facility
- SAS/ACCESS LIBNAME statement

SAS/ACCESS SQL Pass-Through Facility

The following example sends Microsoft SQL Server 6.5 an SQL query for processing. The results from the query serve as a virtual table for the PROC SQL FROM clause. In this example, MYDB is the connection alias.

```
proc sql;
   connect to SQLSVR as mydb
      (datasrc="SQL Server" use=testuser password=testpass);
   select * from connection to mydb
      (select CUSTOMER, NAME, COUNTRY
          from CUSTOMERS
          where COUNTRY <> 'USA');
quit;
```

SAS/ACCESS LIBNAME statement (1)

- The Microsoft SQL Server specific syntax for the LIBNAME statement is:

  LIBNAME *libref* **sqlsvr** <*connection-options*> <*LIBNAME - options*>

  - *Libref* is any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.
  - Sqlsvr is the SAS/ACCESS engine name for the interface to Microsoft SQL Server.
  - *connection – options* provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS.
  - *LIBNAME – options* define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine locking or naming behavior.

SAS/ACCESS LIBNAME statement (2)

- Microsoft SQL server LIBNAME statement examples
  - In following example, USER= and PASSEORD= are connection options.

```
Libname mydblib sqlsvr user=testuser password=testpass;
```

  - In the following example, the libref MYDBLIB connects to a Microsoft SQL Server database using the NOPROMPT= option.

```
libname mydblib sqlsvr
   noprompt="uid=testuser;
   pwd=testpass;
   dsn=sqlservr;"
   stringdates=yes;

proc print data=mydblib.customers;
   where state='CA';
run;
```