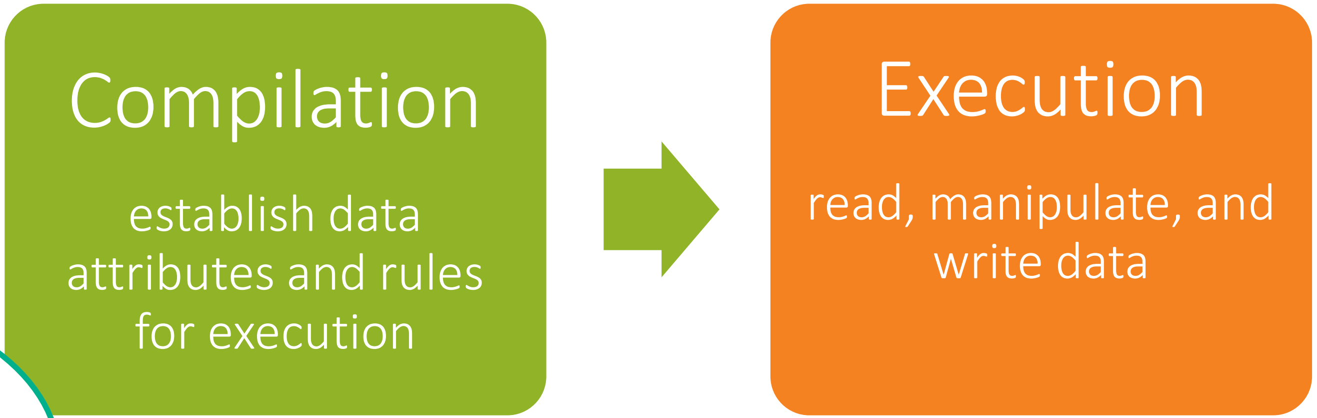
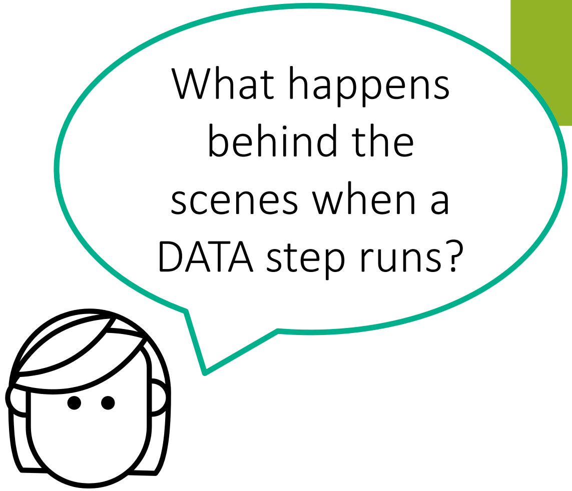
SAS Lesson 05

DATA Step Processing

Understanding DATA Step Processing – Chapter 7

DATA Step Processing


Compilation
establish data attributes and rules for execution
Execution
read, manipulate, and write data



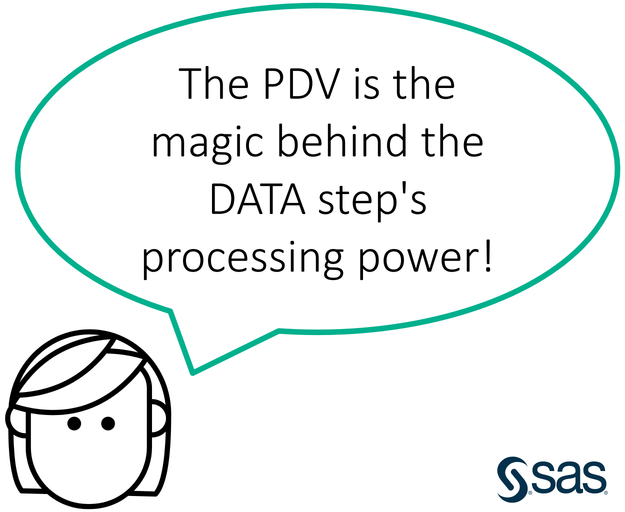
DATA Step Processing: Compilation

Compilation

1) Check for syntax errors.
2) Create the *program data vector (PDV)*, which includes all columns and attributes.
3) Establish the specifications for processing data in the PDV during execution.
4) Create the descriptor portion of the output table.

PDV

Season N 8	Name $ 25	StartDate N 8	Ocean $ 8


The PDV is the magic behind the DATA step's processing power!

DATA Step Processing: Compilation

```
data storm_complete;
    set pg2.storm_summary_small;
    length Ocean $ 8;
    drop EndDate;
    where Name is not missing;
    Basin=upcase(Basin);
    StormLength=EndDate-StartDate;
    if substr(Basin,2,1)="I" then Ocean="Indian";
    else if substr(Basin,2,1)="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

Define the library and a name for the output table.

DATA Step Processing: Compilation

```
data storm_complete;
    set pg2.storm_summary_small;
    length Ocean $ 8;
    drop EndDate;
    where Name is not missing;
    Basin=upcase(Basin);
    StormLength=EndDate-StartDate;
    if substr(Basin,2,1)="I" then Ocean="Indian";
    else if substr(Basin,2,1)="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

Columns are added to the PDV in the order in which they appear in the input table.

PDV

Name $ 15	Basin $ 2	MaxWind N 8	StartDate N 8	EndDate N 8

Attributes are inherited from the input table.

DATA Step Processing: Compilation

```
data storm_complete;
    set pg2.storm_summary_small;
    length Ocean $ 8;
    drop EndDate;
    where Name is not missing;
    Basin=upcase(Basin);
    StormLength=EndDate-StartDate;
    if substr(Basin,2,1)="I" then Ocean="Indian";
    else if substr(Basin,2,1)="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

The remaining columns are added to the PDV in the order in which they appear in the DATA step.

Each column must have at least a name, type, and length.

PDV

Name $ 15	Basin $ 2	MaxWind N 8	StartDate N 8	EndDate N 8	Ocean $ 8	StormLength N 8

DATA Step Processing: Compilation

```
data storm_complete;
    set pg2.storm_summary_small;
    length Ocean $ 8;
    drop EndDate;
    where Name is not missing;
    Basin=upcase(Basin);
    StormLength=EndDate-StartDate;
    if substr(Basin,2,1)="I" then Ocean="Indian";
    else if substr(Basin,2,1)="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

DROP or KEEP statements flag columns that will be excluded from the output table.

PDV

Name $ 15	Basin $ 2	MaxWind N 8	StartDate N 8	EndDate D N 8	Ocean $ 8	StormLength N 8

DATA Step Processing: Compilation

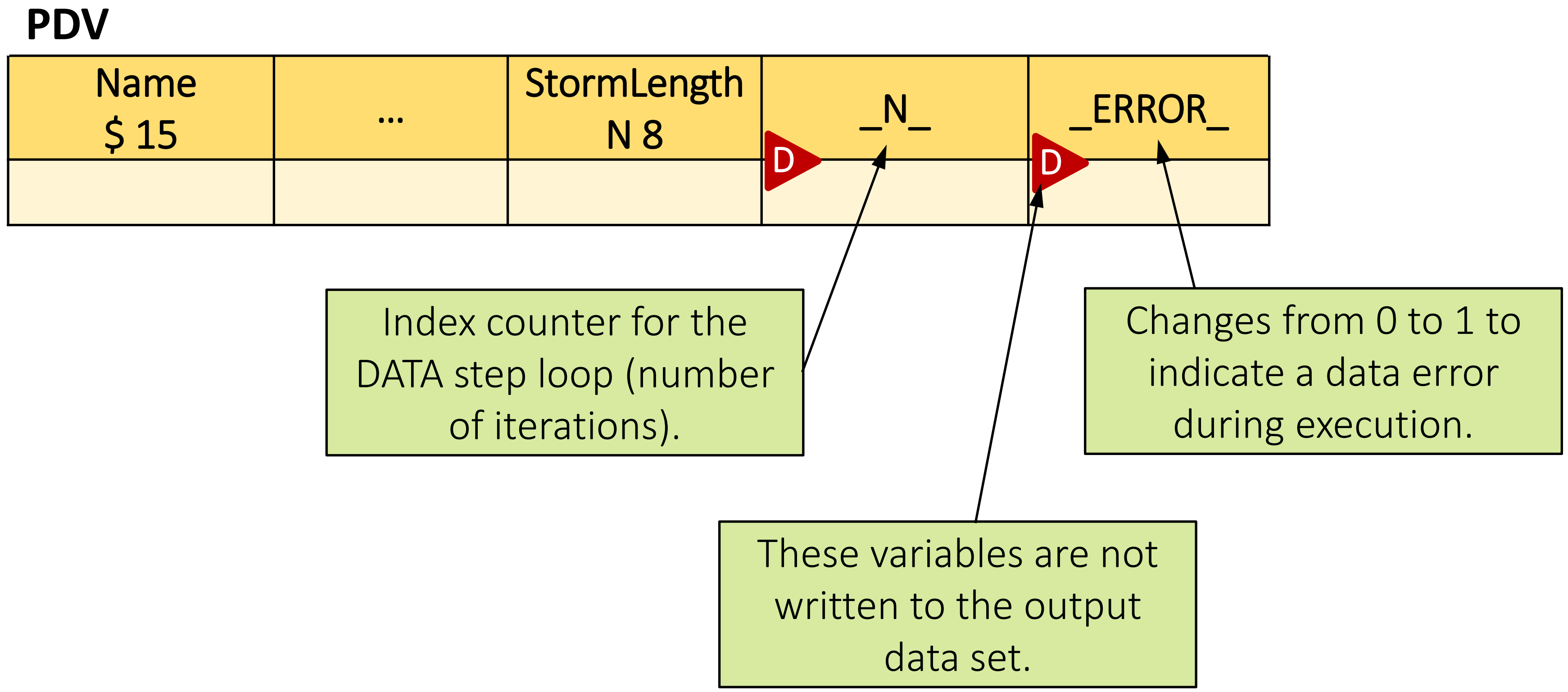
```
data storm_complete;
    set pg2.storm_summary_small;
    length Ocean $ 8;
    drop EndDate;
    where Name is not missing;
    Basin=upcase(Basin);
    StormLength=EndDate-StartDate;
    if substr(Basin,2,1)="I" then Ocean="Indian";
    else if substr(Basin,2,1)="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

The WHERE statement establishes conditions for which rows will be read from the input table into the PDV.

PDV

Name $ 15	Basin $ 2	MaxWind N 8	StartDate N 8	EndDate D N 8	Ocean $ 8	StormLength N 8

DATA Step Processing: Compilation



Index counter for the DATA step loop (number of iterations).

Changes from 0 to 1 to indicate a data error during execution.

These variables are not written to the output data set.

DATA Step Processing: Compilation

```
data storm_complete;
    set pg2.storm_summary_small;
    length Ocean $ 8;
    drop EndDate;
    where Name is not missing;
    Basin=upcase(Basin);
    StormLength=EndDate-StartDate;
    if substr(Basin,2,1)="I" then Ocean="Indian";
    else if substr(Basin,2,1)="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

The descriptor portion is created for the output table.

work.storm_complete

Name	Basin	MaxWind	StartDate	Ocean	StormLength
$ 15	$ 2	N 8	N 8	$ 8	N 8

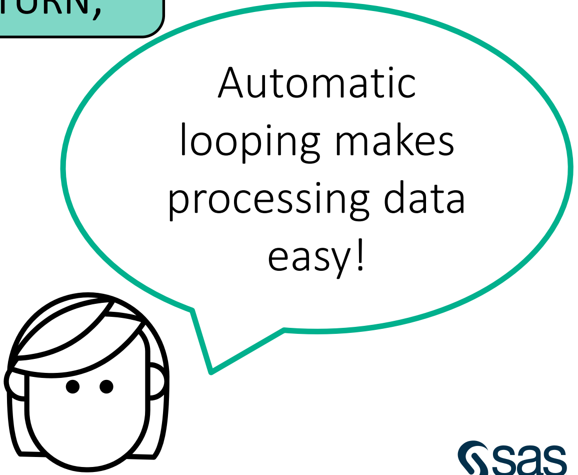
DATA Step Processing: Execution

Execution

1) Initialize the PDV.
2) Read a row from the input table into the PDV.
3) Sequentially process statements and update values in the PDV.
4) At the end of the step, write the contents of the PDV to the output table.
5) Return to the top of the DATA step.

```
data output-table;
     set input-table;
     ...other statements...
run;
```

Implicit OUTPUT;
Implicit RETURN;



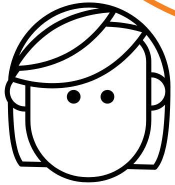
DATA Step Processing: Execution

Iteration 2+

1) Re-Initialize the PDV*.
2) Read a row from the input table into the PDV.
3) Sequentially process statements and update values in the PDV.
4) At the end of the step, write the contents of the PDV to the output table.
5) Return to the top of the DATA step.

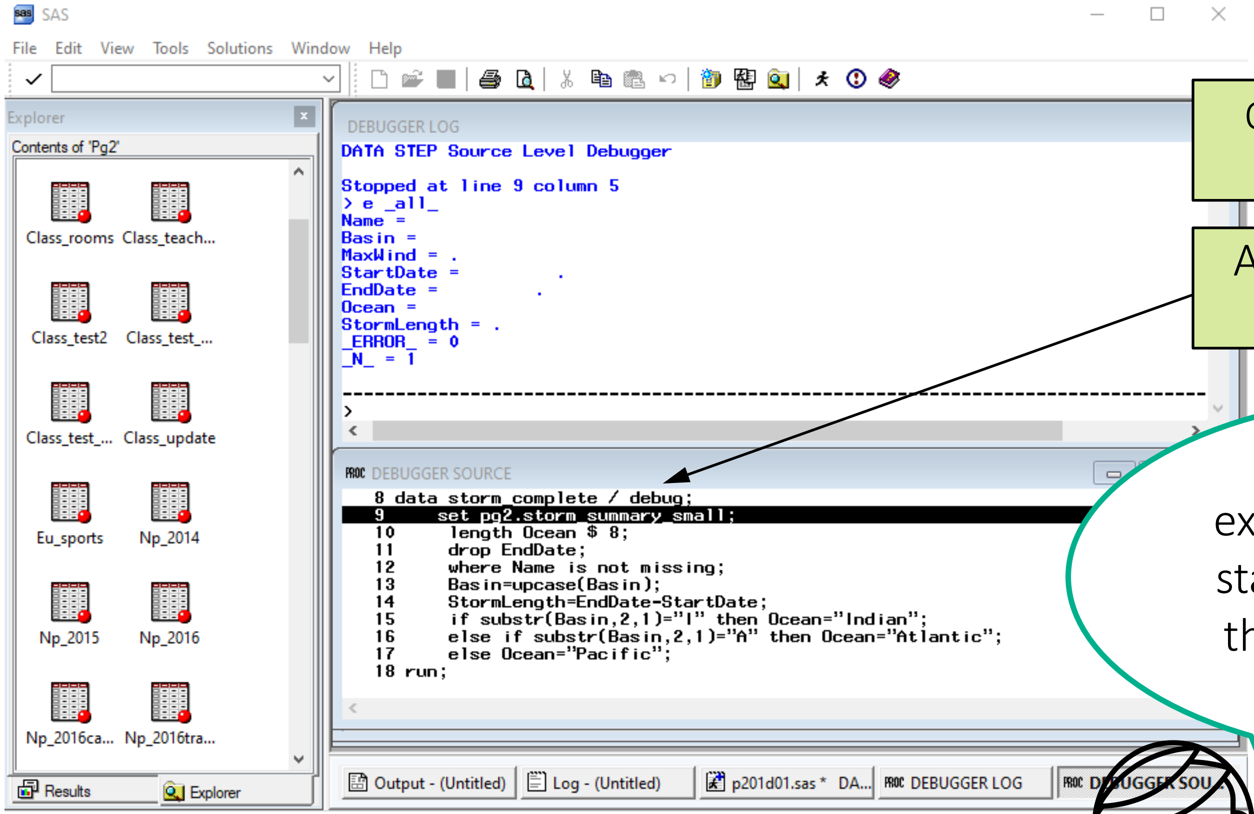
```
data output-table;
     set input-table;
     ...other statements...
run;
```

Implicit OUTPUT;
Implicit RETURN;



*Variables read from data sets are NOT re-initialized!

DATA Step Processing in Action

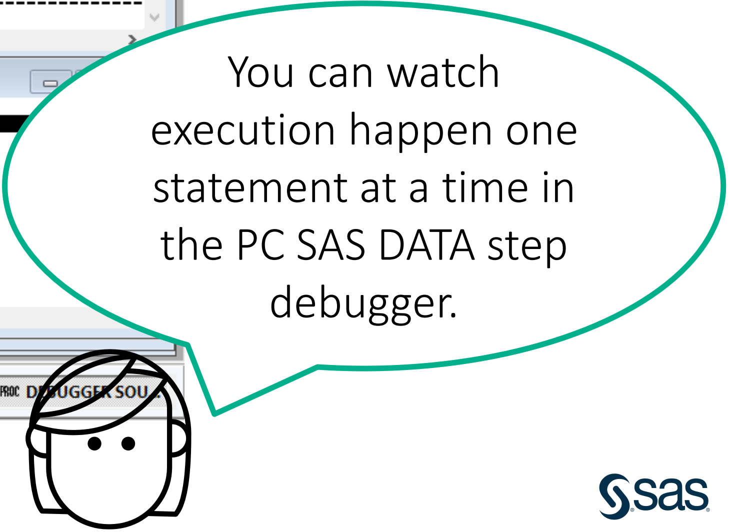


```
DATA STEP Source Level Debugger

Stopped at line 9 column 5
> e _all_
Name =
Basin =
MaxWind = .
StartDate =          .
EndDate =          .
Ocean =
StormLength = .
_ERROR_ = 0
_N_ = 1

------------------------------------------------------------------------------
>
```

```
 8 data storm_complete / debug;
 9     set pg2.storm_summary_small;
10      length Ocean $ 8;
11      drop EndDate;
12      where Name is not missing;
13      Basin=upcase(Basin);
14      StormLength=EndDate-StartDate;
15      if substr(Basin,2,1)="I" then Ocean="Indian";
16      else if substr(Basin,2,1)="A" then Ocean="Atlantic";
17      else Ocean="Pacific";
18 run;
```

Only available in PC SAS or Enterprise Guide.

Add / debug to the end of the DATA step..


You can watch execution happen one statement at a time in the PC SAS DATA step debugger.

DATA Step Processing

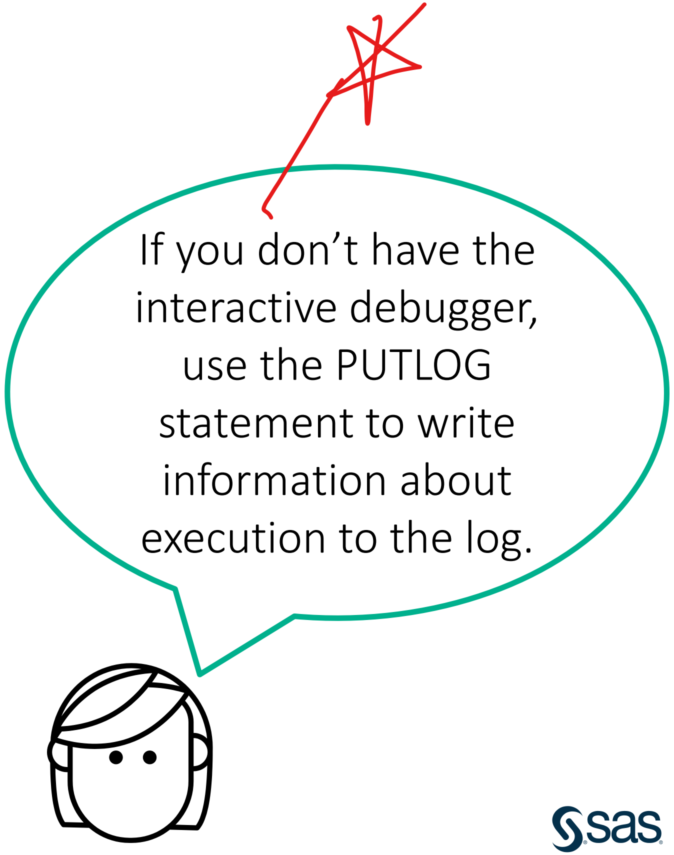
This demonstration illustrates using the DATA step debugger in PC SAS to observe the process of execution.

Viewing Execution in the Log

PUTLOG _ALL_; — writes all columns and values in the PDV to the log

PUTLOG *column*=; — writes selected columns and values in the PDV to the log

PUTLOG "*message*"; — writes a text string to the log



If you don't have the interactive debugger, use the PUTLOG statement to write information about execution to the log.

PUT vs. PUTLOG

Prep Guide Chapter 5

PUT

- Writes to output files if open
- Can be used to control text in output files
- Writes to log only if no file destination is open

PUTLOG

- Only writes to log

Viewing Execution in the Log

```
data storm_complete;
    set pg2.storm_summary_small(obs=2);
    putlog "PDV after SET Statement";
    putlog _all_;
    ...
```

The OBS= data set option limits the observations that are read.

```
PDV after SET Statement
Name=AGATHA Basin=EP MaxWind=115 StartDate=09JUN1980
EndDate=15JUN1980 Ocean=  StormLength=. _ERROR_=0 _N_=1
PDV after SET Statement
Name=ALBINE Basin=SI MaxWind=. StartDate=27NOV1979
EndDate=06DEC1979 Ocean=  StormLength=. _ERROR_=0 _N_=2
```

DATA Step Processing

This demonstration illustrates using log messages with the DATA step to observe the process of execution.

DATA Step Processing

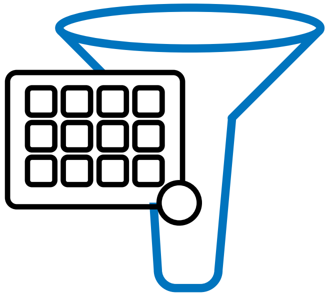
This demonstration illustrates the behavior caused by unbalanced quotation marks.

09-Debug

DATA Step Processing

Filtering Rows

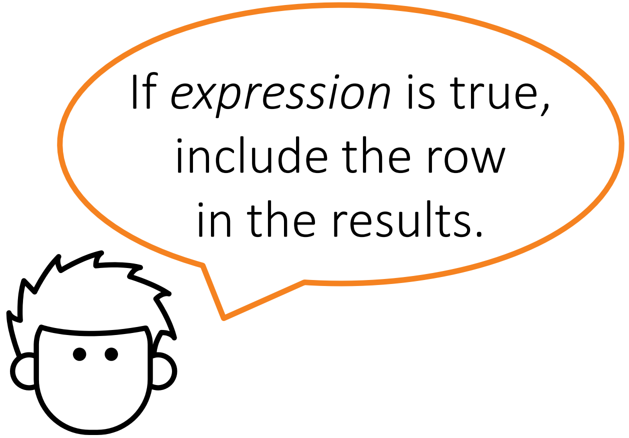
Filtering Rows with the WHERE Statement



```
DATA outset;
     SET inset;
     WHERE expression;
RUN;
```

```
PROC procedure-name . . . ;
     WHERE expression;
RUN;
```

filters rows in the results based on the expression



Using Basic Operators in an Expression

WHERE *expression*;

column **operator** value

= or EQ	< or LT
^= or ~= or NE	>= or GE
> or GT	<= or LE

```
Type = "SUV"
Type EQ "SUV"
MSRP <= 30000
MSRP LE 30000
```

Specifying Values in an Expression

WHERE *expression*;

column | operator | **value**

Character values are case sensitive and must be enclosed in double or single quotation marks.

Numeric values must be standard numeric (that is, no symbols).

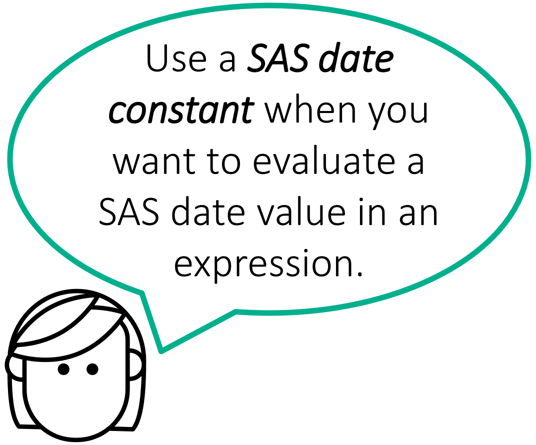
```
Type = "SUV"
Type = 'Wagon'
MSRP <= 30000
```

Specifying Values in an Expression

WHERE *expression*;

column | operator | **value**

"ddmmmyyyy"d



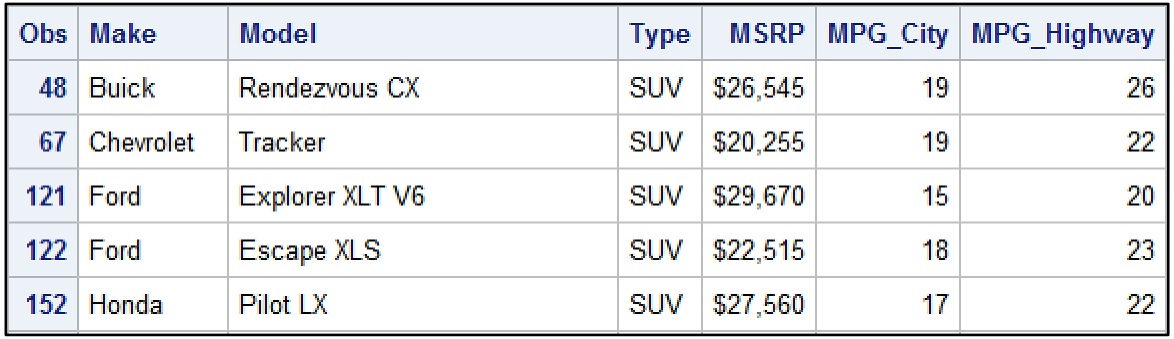
Use a ***SAS date constant*** when you want to evaluate a SAS date value in an expression.

```
where date > "01JAN2015"d;

where date > "1jan15"d;
```

Combining Expressions

```
proc print data=sashelp.cars;
    var Make Model Type MSRP MPG_City MPG_Highway;
    where Type="SUV" and MSRP <= 30000;
run;
```

Expressions can be combined with AND or OR.



Obs	Make	Model	Type	MSRP	MPG_City	MPG_Highway
48	Buick	Rendezvous CX	SUV	$26,545	19	26
67	Chevrolet	Tracker	SUV	$20,255	19	22
121	Ford	Explorer XLT V6	SUV	$29,670	15	20
122	Ford	Escape XLS	SUV	$22,515	18	23
152	Honda	Pilot LX	SUV	$27,560	17	22

Using the IN Operator

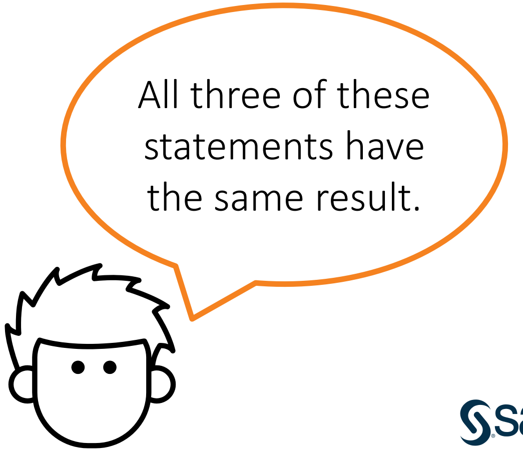
WHERE *col-name* IN (*value-1,…,value-n*);
WHERE *col-name* NOT IN (*value-1,…,value-n*);

Values can be character or numeric.

```
where Type="SUV" or Type="Truck" or Type="Wagon";
```

```
where Type in ("SUV","Truck","Wagon");
```

```
where Type in ("SUV" "Truck" "Wagon");
```



Special WHERE Operators

Special WHERE operators are operators that can only be used in a where-expression*.

Symbol	Mnemonic	Definition
	BETWEEN-AND	an inclusive range
	IS NULL	missing value
	IS MISSING	missing value
?	CONTAINS	a character string
	LIKE	a character pattern

* Do not work on IF statements!

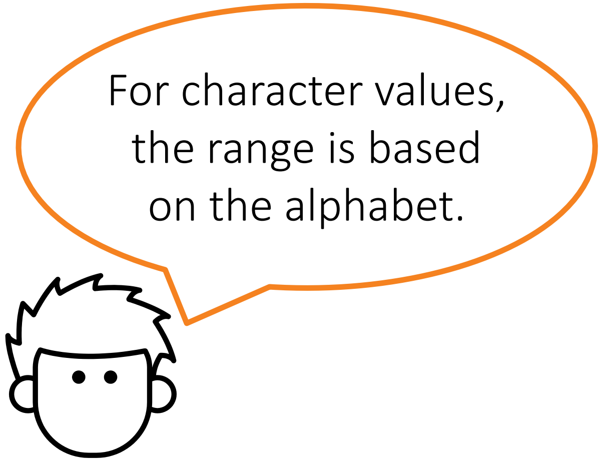
Using Special WHERE Operators

WHERE *col-name* BETWEEN *value-1* AND *value-2*;

```
where age between 20 and 39;
```

```
where 20 <= age <= 39;
```

includes rows with values ***between and including*** the endpoints that you specify

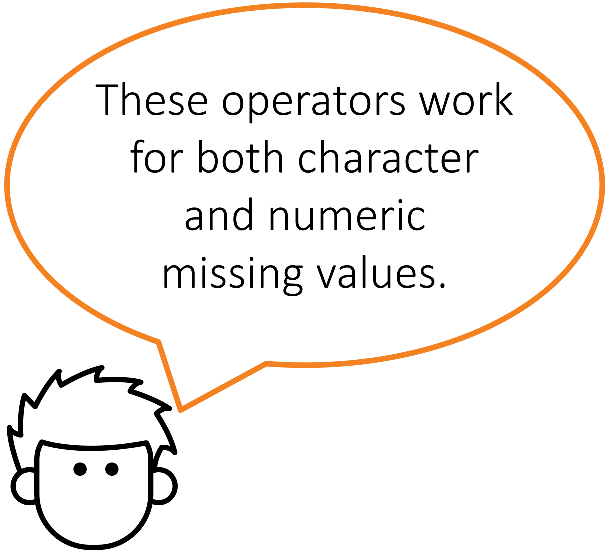


Using Special WHERE Operators

WHERE *col-name* IS NULL;
WHERE *col-name* IS NOT NULL;

WHERE *col-name* IS MISSING;
WHERE *col-name* IS NOT MISSING;

```
where age is missing;

where name is not missing;
```


These operators work for both character and numeric missing values.

Using Special WHERE Operators

WHERE *col-name* CONTAINS "*value*";

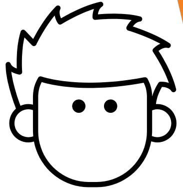
```
where Job_Title contains 'Rep';
```

The **position** of the substring within the variable's values is **not important**.

The operator is **case sensitive** when you make comparisons.

Republican Party
House of Representatives

The *CONTAINS (?) operator* selects observations that include the specified substring.



Using Special WHERE Operators

WHERE *col-name* LIKE "*value*";

```
where City like "New%";
```

New York
New Delhi
Newport
Newcastle
New

wildcard for any number of characters

```
where City like "Sant_%";
```

Santa Clara
Santa Cruz
Santo Domingo
Santo Tomas

wildcard for a single character

Using Special WHERE Operators

How do you search for % and _ in your data?

Beginning with 9.2, an escape character forces literal search for % and _.

Example:

```
where Name like 'A/_C' escape '/';
```

Filtering Rows with Basic Operators

This demonstration illustrates using the WHERE statement and basic operators to subset rows in a procedure.

Lesson Quiz

1. Which statement is false concerning the compilation phase of the DATA step?

a. Initial values are assigned to the columns.
b. The program data vector (PDV) is created.
c. The DATA step is checked for syntax errors.
d. The descriptor portion of the output table is created.

1. Which statement is false concerning the compilation phase of the DATA step?

[✓ a.] Initial values are assigned to the columns.
b. The program data vector (PDV) is created.
c. The DATA step is checked for syntax errors.
d. The descriptor portion of the output table is created.

2. Which statement is not a compile-time-only statement?

a. KEEP

b. LENGTH

c. SET

d. WHERE

2. Which statement is not a compile-time-only statement?

a. KEEP

b. LENGTH

[✓ c.] SET

d. WHERE

3. Which statement is true concerning the execution phase of the DATA step?

a. Data is processed in the program data vector (PDV).
b. An implied OUTPUT occurs at the top of the DATA step.
c. An implied REINITIALIZE occurs at the bottom of the DATA step.
d. Columns read from the input table are set to missing when SAS returns to the top of the DATA step.

3. Which statement is true concerning the execution phase of the DATA step?

[✓ a.] Data is processed in the program data vector (PDV).
b. An implied OUTPUT occurs at the top of the DATA step.
c. An implied REINITIALIZE occurs at the bottom of the DATA step.
d. Columns read from the input table are set to missing when SAS returns to the top of the DATA step.

4. The DATA step debugger in SAS Enterprise Guide can be used with DATA and PROC steps.

a. True

b. False

4. The DATA step debugger in SAS Enterprise Guide can be used with DATA and PROC steps.

a. True

[✓ b.] False

5. Which PUTLOG statements create the following results in the SAS log?

```
Name=Alfred Height=69 Weight=112.5 Ratio=0.61 _ERROR_=0 _N_=1
Ratio=0.61
```

a. **`putlog all; putlog Ratio;`**

b. **`putlog all; putlog Ratio=;`**

c. **`putlog _all_; putlog Ratio;`**

d. **`putlog _all_; putlog Ratio=;`**

5. Which PUTLOG statements create the following results in the SAS log?

```
Name=Alfred Height=69 Weight=112.5 Ratio=0.61 _ERROR_=0 _N_=1
Ratio=0.61
```

a. **`putlog all; putlog Ratio;`**

b. **`putlog all; putlog Ratio=;`**

c. **`putlog _all_; putlog Ratio;`**

[✓ d.] **`putlog _all_; putlog Ratio=;`**

1. Which statement is false concerning the options for the PROC EXPORT statement?

   a. The DATA= option identifies the input SAS table.
   b. The REPLACE option specifies to overwrite an existing file.
   c. The DBMS= option specifies the database identifier for the type of file being created.
   d. The OUT= option specifies the path and file name of the external data file being created.

1. Which statement is false concerning the options for the PROC EXPORT statement?

   a. The DATA= option identifies the input SAS table.
   b. The REPLACE option specifies to overwrite an existing file.
   c. The DBMS= option specifies the database identifier for the type of file being created.
   [✓ d.] The OUT= option specifies the path and file name of the external data file being created.

2. Which PROC EXPORT step contains valid syntax?

a.
```
proc export outfile="c:\temp\cars.txt" tab
            data=sashelp.cars replace; run;
```

b.
```
proc export data=sashelp.cars dbms=csv
            outfile="c:\temp\cars.csv"; run;
```

c.
```
proc export data=sashelp.class; dbms=csv;
            outfile="c:\temp\cars.csv"; run;
```

d.
```
proc export dbms=tab data=sashelp.cars replace=yes
            outfile="c:\temp\cars.txt"; run;
```

2. Which PROC EXPORT step contains valid syntax?

a.
```
proc export outfile="c:\temp\cars.txt" tab
            data=sashelp.cars replace; run;
```

[✓ b.]
```
proc export data=sashelp.cars dbms=csv
            outfile="c:\temp\cars.csv"; run;
```

c.
```
proc export data=sashelp.class; dbms=csv;
            outfile="c:\temp\cars.csv"; run;
```

d.
```
proc export dbms=tab data=sashelp.cars replace=yes
            outfile="c:\temp\cars.txt"; run;
```

3. What does the following program create?

```
libname sales xlsx 'c:\mydata\midyear.xlsx';

data sales.q1_2018;
    set sasdata.qtr1_2018;
run;
data sales.q2_2018;
    set sasdata.qtr2_2018;
run;
```

a. two SAS tables: **sales.q1_2018** and **sales.q2_2018**

b. two Excel workbooks: **sales.q1_2018** and **sales.q2_2018**

c. two worksheets in the Excel workbook: **midyear**: **q1_2018** and **q2_2018**

d. two worksheets in the Excel workbook: **sales**: **q1_2018** and **q2_2018**

3. What does the following program create?

```
libname sales xlsx 'c:\mydata\midyear.xlsx';

data sales.q1_2018;
    set sasdata.qtr1_2018;
run;
data sales.q2_2018;
    set sasdata.qtr2_2018;
run;
```

a. two SAS tables: **sales.q1_2018** and **sales.q2_2018**

b. two Excel workbooks: **sales.q1_2018** and **sales.q2_2018**

[✓ c.] two worksheets in the Excel workbook: **midyear**: **q1_2018** and **q2_2018**

d. two worksheets in the Excel workbook: **sales**: **q1_2018** and **q2_2018**

5. What type of output file does this program create?

```
libname mylib xlsx "s:/workshop/output/test.xlsx";

data class_list;
    set sashelp.class;
run;
```

a. SAS table

b. delimited file

c. Microsoft Excel XLS file

d. Microsoft Excel XLSX file

5. What type of output file does this program create?

```
libname mylib xlsx "s:/workshop/output/test.xlsx";

data class_list;
    set sashelp.class;
run;
```

[✓ a.] SAS table

b. delimited file

c. Microsoft Excel XLS file

d. Microsoft Excel XLSX file

6. Which of these programs creates a Microsoft Excel file?

a.

```
ods excel file="s:/workshop/output/class.xlsx";
proc print data=sashelp.class;
run;
ods excel close;
```

b.

```
libname mylib xlsx "s:/workshop/output/class.xlsx";
data mylib.class_list;
   set sashelp.class;
run;
```

c. both

d. neither

6. Which of these programs creates a Microsoft Excel file?

a.

```
ods excel file="s:/workshop/output/class.xlsx";
proc print data=sashelp.class;
run;
ods excel close;
```

b.

```
libname mylib xlsx "s:/workshop/output/class.xlsx";
data mylib.class_list;
   set sashelp.class;
run;
```

[✓ c.] both

d. neither