

# SAS Temporary Arrays

Steve Croft

The background is a solid teal color. At the bottom right corner, there is a stylized silhouette of a mountain range in a slightly darker shade of teal.

# Benefits

- ◆ Multiple passes through datasets without outputting data between steps
- ◆ Complex many-to-many matches

**By storing more than 1  
observation at the same time**

# Example 1: Iterative calculations

- ◆ A Rim-weighting programme written to select the TV Ratings panel.
- ◆ Used Iterative proportional fitting to calculate a probability of selection for each possible surveyed household.
- ◆ MS Access database using several multi-table queries took about **2 hours** run.
- ◆ Successive proc summary steps and merges took **50 minutes**.
- ◆ The temporary array technique took **2 minutes**.

# Example 2: Complex matches

- ◆ Currently, we're working on a procedure to match nearly 5 individual million vehicle records to a smaller dataset of manufacturer's specifications.
- ◆ The match is very complex requiring a match based on :
  - a VIN number with 17 characters in total and a possible match with one of the first 9-13 characters AND/OR
  - the best set of matches between Make, Model, Year of Manufacture, Series, # doors, engine size etc.
- ◆ We might achieve this using hundreds of successive merges and then we'd have to worry about many-to-many matches.
- ◆ This can all be done in one **single Data Step**.

# What is an array?

- ◆ Definition: “An arrangement of **memory elements** in one or more planes”.
- ◆ What makes them fast is that they are stored in memory.
- ◆ Memory access is 100,000 times faster than accessing the hard disk.

# Normal Arrays in SAS

- ◆ Used in a data step to manipulate data within each observation.
- ◆ Active for only the current observation.
- ◆ Repetitive tasks within the one observation are made easier.

# Normal Arrays in SAS

- ◆ DO Loops are used to cycle through the elements of an array.

```
data ABC;  
array A{5} VAR1-VAR5;  
do i=1 to 5;  
A{i} = A{i} + 2;  
end;  
run;
```

# The Data Step

- ◆ At the beginning of Data Step the Program Data Vector is created.
- ◆ Data is loaded into this one observation at a time.

Data set: ABC

Obs	A	B	C
1	81	34.2	XYS
2	23	59.8	KJE
3	24	27.3	QWE
4	24	16.1	WER
5	4	8.0	OIU

Program Data Vector

A	B	C
81	34.2	XYS

# The Data Step

- ◆ At the beginning of Data Step the Program Data Vector is created.
- ◆ Data is loaded into this one observation at a time.

Data set: ABC

Obs	A	B	C
1	81	34.2	XYS
2	23	59.8	KJE
3	24	27.3	QWE
4	24	16.1	WER
5	4	8.0	OIU

Program Data Vector

A	B	C
23	59.8	KJE



# The Data Step

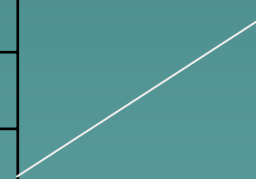
- ◆ At the beginning of Data Step the Program Data Vector is created.
- ◆ Data is loaded into this one observation at a time.

Data set: ABC

Obs	A	B	C
1	81	34.2	XYS
2	23	59.8	KJE
3	24	27.3	QWE
4	24	16.1	WER
5	4	8.0	OIU

Program Data Vector

A	B	C
24	27.3	QWE



# The Data Step (cont'd)

- ◆ After each observation is processed with the statements in the data step, we can reference previous observations using the Lag or Retain functions but cannot change these values in these observations.

Obs	A	B	C
1	81	34.2	XYS
2	23	59.8	KJE
3	24	27.3	QWE
4	24	16.1	WER
5	4	8.0	OIU

A	B	C	C1	C2
24	27.3	QWE	KJE	XYS

C1 = lag1(C);  
C2 = lag2(C);

# Or can we?

- ◆ Consider the following code:

```
data ABC2;  
array AB{5,2} _temporary_;  
do i=1 to 5;  
set ABC;  
AB{i,1} = A;  
AB{i,2} = B;  
end;  
run;
```

A 5 by 2 temporary array is established

Do Loop cycles through all of the data set's observations  
NB. SET statement is nested in DO-LOOP

Variables A and B are loaded into the array

At the END statement we have the entire dataset contained in an array

# Array in memory

- ◆ At the close of the do loop I have a 5 by 2 array called AB which contains the whole ABC dataset:

81	34.2
23	59.8
24	27.3
24	16.1
4	8.0

- ◆ Now I can refer to AND change any cell of the array (or if you like) any variable in any observation in the dataset.

$$AB\{3,2\} = 27.3$$

# What the Online Help says...

Note: Do not use the `_TEMPORARY_` option if you plan to use the SET routine to read values from a SAS table directly into array elements. You must use the GETVARN or GETVARC function to read values from a SAS table into the elements of a temporary array.

So what  
makes the  
array  
Temporary?



# What makes an array temporary?

- ◆ The keyword `_temporary_` instructs SAS not to clear an array when an observation is loaded into the Program Data Vector (as with normal arrays).
- ◆ Perhaps `_Semi Permanent_` might have been a better syntax.
- ◆ OR `_Slightly longer than normal_`

# Multiple Passes

- ◆ Now we can make multiple passes through the array.
- ◆ We can aggregate the data and use this information within the one data step.

# Multiple Passes

```
data ABC2(keep=A B C);  
array AB{5,3} _temporary_;  
do i=1 to 5;  
set ABC(drop=c);  
AB{i,1} = A;  
AB{i,2} = B;  
end;  
totalB=0;  
do i=1 to 5;  
totalB=totalB+AB{i,2};  
end;  
meanB=totalB/5;  
do i=1 to 5;  
AB{i,3}=AB{i,2}/meanB;  
end;  
run;
```

A 5 by 3 temporary array is established

Data is Loaded

Pass 1: Calculate the sum of Variable B

Pass 2: Calculate the ratio of Variable B to its mean

# The Output Data Set

- ◆ At this point, my dataset (ABC2) looks like this:

Obs	I	A	B	totalB	meanB
1	6	4	8	145.4	29.08

- ◆ Variables A and B contain the data from the last observation in the original set statement. Everything else is still in memory.

# Multiple Passes

```
data ABC2(keep=A B C);
array AB{5,3} _temporary_;
do i=1 to 5;
set ABC(drop=c);
AB{i,1} = A;
AB{i,2} = B;
end;
totalB=0;
do i=1 to 5;
totalB=totalB+AB{i,2};
end;
meanB=totalB/5;
do i=1 to 5;
AB{i,3}=AB{i,2}/meanB;
end;
do i=1 to 5;
A = AB{i,1};
B = AB{i,2};
C = AB{i,3};
output;
end;
run;
```

Output the array to  
the new dataset  
ABC2 with the  
calculated ratio C

# The Output Data Set

- ◆ After the output step, my new dataset (ABC2) looks like this:

Obs	A	B	C
1	81	34.2	1.17608
2	23	59.8	2.05640
3	24	27.3	0.93879
4	24	16.1	0.55365
5	4	8.0	0.27510

# Multiple Passes

- ◆ The same result might be achieved with a proc summary step to calculate the means followed by a merge to calculate the ratio.
- ◆ This is a very simple example, but consider iterative calculations requiring hundreds of proc summary and merge steps all done through one data step.

# HELPFUL TIPS

- ◆ Always document the contents of the array in the SAS code - especially with complex arrays and multiple passes.
- ◆ Indent the DO and END statements to ensure they match.
- ◆ Write the input and the output steps first and check that what has gone into the "Black Box" is what you've put in – Remember it all happens in memory.

# HELPFUL TIPS (cont'd)

- ◆ When loading numeric and character variables you will need 2 arrays - an array must contain numeric or character variables only.
- ◆ Use the syntax  
array X{5,1} \$ \_temporary\_;  
to denote character arrays

# Many-to-Many merges

first

A	B
1	34.2
2	59.8
2	27.3
2	16.1
3	8
3	95.8

second

A	C
1	Blue
1	Green
2	Brown
3	Amber
3	Red
3	Brown
3	Yellow

```
data third;  
merge first second;  
by a;  
run;
```

NOTE: MERGE statement has more than one data set with repeats of BY values.

# Many-to-Many merges

A	B
1	34.2
2	59.8
2	27.3
2	16.1
3	8
3	95.8

+

A	C
1	Blue
1	Green
2	Brown
3	Amber
3	Red
3	Brown
3	Yellow

yields

A	B	C
1	34.2	Blue
1	34.2	Green
2	59.8	Brown
2	27.3	Brown
2	16.1	Brown
<b>3</b>	<b>8</b>	<b>Amber</b>
<b>3</b>	<b>95.8</b>	<b>Red</b>
<b>3</b>	<b>95.8</b>	<b>Brown</b>
<b>3</b>	<b>95.8</b>	<b>Yellow</b>

# Many-to-many merges

- ◆ The merge can be handled using a temporary array as a lookup table.
- ◆ First, we count the number of observations in each dataset and store this in a macro variable.
- ◆ This construct is used to avoid hard-coding the do loops.

```
data _null_;  
  if 0 then set first nobs=count;  
  call symput('fcount',trim(left(count)));  
run;
```

# Many-to-many merges

```
data third(keep=a b c);  
array f{&fcount.,2} _temporary_;  
do i=1 to &fcount.;  
set first;  
f{i,1} = A;  
f{i,2} = B;  
end;  
do i=1 to &scount.;  
set second;  
do j=1 to &fcount.;  
if a=f{j,1} then  
do;  
b=f{j,2};  
output;  
end;  
end;  
end;  
run;
```

The “first” dataset is loaded into a temporary array

The “second” dataset is loaded into the PDV

Using the first dataset as a lookup table we output possible matches for the variable A

**FCOUNT=6**  
**SCOUNT=7**

# The result

A	B
1	34.2
2	59.8
2	27.3
2	16.1
3	8
3	95.8

+

yields

A	C
1	Blue
1	Green
2	Brown
3	Amber
3	Red
3	Brown
3	Yellow

A	B	C
1	34.2	Blue
1	34.2	Green
2	59.8	Brown
2	27.3	Brown
2	16.1	Brown
3	8	Amber
3	95.8	Amber
3	8	Red
3	95.8	Red
3	8	Brown
3	95.8	Brown
3	8	Yellow
3	95.8	Yellow

# Many-to-many merges

- ◆ As with most procedures in SAS, this can be done another way – a SQL join.
- ◆ However, the temporary array allows us a lot better control over which observations we want to keep in our final matched dataset.
- ◆ Moreover, we don't have to merge only two datasets – imagination and available memory are the only limitations.

# Summary

- ◆ Temporary arrays, whilst complex to use and not well documented provide a very powerful technique for manipulating data.
- ◆ Access any observation anywhere in your dataset.
- ◆ Many-to-many merges can be achieved with much more control than other techniques.
- ◆ Lookups from many datasets can be done within the one data step.
- ◆ Many iterations that use constant input and output, can be done in one not-so-easy step.

Any Questions not about tunnels  
and older drivers?