Update to the Rescue

By Robert Virgile

Update to the Rescue

Bob Virgile has been teaching and consulting in SAS for over 35 years. He formerly composed the problem-solving contests for regional and international SAS conferences, wrote and presented numerous SAS papers, and wrote three books for SAS Institute. He is the only person in the history of NESUG to be barred from the SAS Bowl competition due to superior SAS knowledge.

Update Basics …

```
data combined;
  update master changes;
  by id;
run;
```

Update combines exactly two SAS data sets.

Both data sets must be sorted.

There can be mismatches.

The first data set contains no more than one observation per ID.

The second data set can contain many observations per ID.

Names of variables and data sets are up to you.

Update Basics …

```
data combined;
  update master changes;
  by id;
run;
```

The final COMBINED data set will contain just one observation per ID.

For each ID:

- Begin with the data values found in MASTER.
- For each observation in CHANGES, use nonmissing values only (replacing any previous values).
- Output a single observation per ID, once all the changes have been applied.

Update Basics …

MASTER

ID	Height	Weight
Alice	.	130
Bob	72	180
Carol	66	125

CHANGES

ID	Height	Weight
Alice	68	.
Alice	.	132
Carol	.	120

Current values for

ID	Height	Weight
Alice	.	130
Alice	68	130
Alice	68	132
Bob	72	180
Carol	66	125
Carol	66	120

Output data set COMBINED

ID	Height	Weight
Alice	68	132
Bob	72	180
Carol	66	120

Update Basics …

That’s nice if you need it.

But who needs it?

Let’s consider some situations that require a bit of creativity.

Case 1: Replace Missings Only

The scenario:

- Two data sets contain just a single observation per ID.
- There might be some mismatches.
- The CHANGES data set should replace only the missing values in MASTER. Nonmissing values in MASTER should remain untouched.

Case 1: Replace Missings Only

MASTER

ID	Height	Weight
Alice	.	130
Bob	72	180
Carol	66	125

CHANGES

ID	Height	Weight
Alice	68	.
Bob	.	132
Carol	.	120

The intent is to replace only missing values in the MASTER data set.

Output data set COMBINED

ID	Height	Weight
Alice	68	130
Bob	72	180
Carol	66	125

Case 1: Replace missings only

MERGE fails. It replaces all values, not just the nonmissing values.

```
data combined;
  merge master changes;
  by ID;
run;
```

ID	Height	Weight
Alice	68	.
Bob	.	132
Carol	.	120

UPDATE fails. It can replace nonmissing values, not just missing values.

```
data combined;
  update master changes;
  by ID;
run;
```

ID	Height	Weight
Alice	68	130
Bob	72	132
Carol	66	120

Case 1: Replace missings only

Programming around the problem is cumbersome:

```
data combined;
  merge master changes (rename=(height=height2 weight=weight2));
  by ID;
  if height = . then height = height2;
  if weight = . then weight = weight2;
  drop height2 weight2;
run;
```

What if there were 100 variables instead of two?

Case 1: Replace missings only

What would work?

Switch the order of the data sets within the UPDATE statement.

```
data combined;
  update changes master;
  by ID;
run;
```

CHANGES

ID	Height	Weight
Alice	68	.
Bob	.	132
Carol	.	120

MASTER

ID	Height	Weight
Alice	.	130
Bob	72	180
Carol	66	125

Case 2: Collapsing Rows

A single data set contains data spread across multiple observations:

ID	Height	Weight
Alice	68	.
Alice	.	132
Bob	.	180
Bob	70	.
Carol	62	.
Carol	.	140

A successful “collapse”:

ID	Height	Weight
Alice	68	132
Bob	70	180
Carol	62	140

Case 2: Collapsing Rows

Incoming data:

ID	Height	Weight
Alice	68	.
Alice	.	132
Bob	.	180
Bob	70	.
Carol	62	.
Carol	.	140

```
data collapsed;
 set my_data;
 by ID;
 if first.ID then do;
   replace_height = height;
   replace_weight = weight;
 end;
 else do;
   if height > . then replace_height = height;
   if weight > . then replace_weight = weight;
 end;
 retain replace_height replace_weight;
 if last.ID;
 drop height weight;
 rename replace_height = height
 replace_weight = weight;
run;
```

Case 2: Collapsing Rows

Keep in mind:

- The program is complex enough with only a few variables. What if there were 20 variables spread out in similar fashion?

Features that resemble UPDATE:

- Process nonmissing values and ignore missing values
- Generate just one observation per ID

Strange, and nothing like UPDATE:

- There's only one data set.

Case 2: Collapsing Rows

Is the solution simple?

```
data collapsed;
  update my_data (obs=0) my_data;
  by ID;
run;
```

Incoming data:

ID	Height	Weight
Alice	68	.
Alice	.	132
Bob	.	180
Bob	70	.
Carol	62	.
Carol	.	140

You don't even need to know the names of the variables! For each ID:

- Take nothing from the first data set, but at least there is a first data set.
- One at a time, take observations for an ID from the second data set. Use the non-missing values.
- When all observations for that ID have been processed, output the result.

Case 3: Combining Multiple Sources

Three data sets (GOOD, BETTER, and BEST) all contain the same set of variables.

The intent:

- Generate a single observation per ID.
- Use all the nonmissing values in BEST.
- For data values that are missing from BEST, use nonmissing values from BETTER.
- For data values that are still missing, use nonmissing values from GOOD.

Case 3: Combining Multiple Sources

The incoming data sets all contain ID, HEIGHT, and WEIGHT:

GOOD		
Amy	60	140
Bob	64	155
Dan	70	198
Eve	60	140
Eve	.	135
Joe	72	220
Lou	63	150

BETTER		
Amy	61	135
Bob	65	160
Dan	68	.
Eve	61	135
Irv	61	135
Joe	71	.

BEST		
Amy	.	150
Bob	.	150
Eve	.	142
Irv	.	150
Joe	73	.
Lou	.	160
Lou	.	165

The intended result: use the values in red, for each ID.

Case 3: Combining Multiple Sources

The set-up (after sorting each data set by ID):

```
data all3;
  set good better best;
  by ID;
run;
```

The order is important, putting the most “valuable” observations at the end.

Then following the same logic as in Case 2:

```
data final;
  update all3 (obs=0) all3;
  by ID;
run;
```

Case 4: LOCF

LOCF is a method of replacing missing values.

“Last Observation Carried Forward”

Before

ID	RecNo	Amount
Amy	1	1234
Amy	2	.
Amy	3	2468
Bob	1	.
Bob	2	3456
Bob	3	.
Bob	4	4567
Bob	5	.

After

ID	RecNo	Amount
Amy	1	1234
Amy	2	1234
Amy	3	2468
Bob	1	.
Bob	2	3456
Bob	3	3456
Bob	4	4567
Bob	5	4567

Case 4: LOCF

Without UPDATE:

```
data after;
  set before;
  by ID;
  if first.ID then replacement = amount;
  else if amount > . then replacement = amount;
  else amount = replacement;
  retain replacement;
  drop replacement;
 run;
```

Before

ID	RecNo	Amount
Amy	1	1234
Amy	2	.
Amy	3	2468
Bob	1	.
Bob	2	3456
Bob	3	.
Bob	4	4567
Bob	5	.

Case 4: LOCF

This program has advantages and disadvantages:

```
data after;
  update before (obs=0) before;
  by ID;
run;
```

UPDATE ignores missing values, and holds onto the last-encountered non-missing value.

But UPDATE outputs just one observation per ID.

ID	RecNo	Amount
Amy	1	1234
Amy	2	.
Amy	3	2468
Bob	1	.
Bob	2	3456
Bob	3	.
Bob	4	4567
Bob	5	.

ID	RecNo	Amount
Amy	3	2468
Bob	5	4567

Case 4: LOCF

So what do we do?

```
data after;
  update before (obs=0) before;
  by ID;
  output;
run;
```

ID	RecNo	Amount
Amy	1	1234
Amy	2	.
Amy	3	2468
Bob	1	.
Bob	2	3456
Bob	3	.
Bob	4	4567
Bob	5	.

ID	RecNo	Amount
Amy	1	1234
Amy	2	1234
Amy	3	2468
Bob	1	.
Bob	2	3456
Bob	3	3456
Bob	4	4567
Bob	5	4567