SAS GLOBAL FORUM 2020 #SASGF

Paper 5145 – 2020

Urge to MERGE? Maybe You Should UPDATE Instead

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT:
The DATA step's UPDATE statement is similar to the MERGE, but it has some helpful built-in logic of which many users of SAS may not be familiar. In most cases, this built in logic can yield much simpler DATA steps. This paper sheds light on some of these build-in features and takes a step-by-step approach to showing you how to take advantage of this power that is already there.

INTRODUCTION:
Several years ago, I was at a client's site to do some SAS training. One morning as I was about to enter the classroom, I was met by some of the company's SAS 'gurus'. They were trying to write a program to do some UPDATING, but they were writing a DATA step with a MERGE statement to do the job. They could not get the program to work properly and were asking for my help. It was an impressive looking DATA step; all four pages looked like it could work, but it was not doing what they wanted. My first reaction was this program could be written with an UPDATE statement instead of a MERGE statement and it would be less complex. They could have written a four statement program to accomplish their task instead of a four page program. The UPDATE statement has a lot of logic built into it. My client did not know this and was trying to write a DATA step using the MERGE statement and re-create the logic that is already available in the UPDATE statement.

The UPDATE statement performs a special type of merge. It's function is to update a master file, in the form of a SAS dataset, by applying transactions (observations from another SAS dataset). The UPDATE statement is used to do the following:

- change data values for variables in the master SAS dataset
- add observations to the master SAS dataset.

The typical syntax for the UPDATE statement is:

```
UPDATE  master-file    transaction-file ;
BY  id-variable ;
```

The rules for using the UPDATE statement are:

- only two datasets can appear on the UPDATE statement
- the MASTER file must be listed first
- a BY statement containing the ID-Variable must be used
- both datasets must be sorted by the BY variable
- the MASTER file must have only one observation per unique value of the BY variable.

The following examples will use these two simple datasets from the insurance field. The **Policy_Holder** dataset is the master file while the **New_Information** dataset serves as the transaction file.

Policy_Holder				
P_Num	LOB	Gender	Age	Phone
10010	Auto	Male	37	828-4001
10021	Home	Female	32	833-4514
10033	Life	Male	27	811-1949
11010	Auto	Female	44	927-1108
20004	Life	Female	41	727-2508

New_Information				
P_Num	LOB	Gender	Age	Phone
10010			73	
10021	Life			
10033		Female	28	
11010				927-1801
20044	Home	Male	27	864-2224

Figure 1.

The first example is to update the master file (Policy_Holder) with the transaction file (New_Information) which has new information on the policy holders. A simple four statement DATA Step will get the job done.

```
data Updated_1;
     update Policy_Holder New_Information;
     by P_Num;
run;

proc print data=Updated_1;
     title 'Updated Records for First Quarter';
run;
```

Figure 2.

Using the ODS TEXT destination (the Output window), the PROC PRINT report looks like this...

Updated Records for First Quarter

Obs	P_Num	LOB	Gender	Age	Phone
1	10010	Auto	Male	73	828-4001
2	10021	Life	Female	32	833-4514
3	10033	Life	Female	28	811-1949
4	11010	Auto	Female	44	927-1801
5	20004	Life	Female	41	727-2508
6	20044	Home	Male	27	864-2224

Figure 3.

Notice that the missing values from the transaction data set do NOT overwrite the non-missing values from the master file**.**

What if someone wants to have their information updated by NOT displaying their phone number or other information? In other words, what if we WANT missing values in the transaction file to overwrite non - missing information in the master file? There is a special MISSING statement that is used in the DATA step to identify a character that represents a special kind of missing value.

The next task is to overwrite non-missing values in the master-file with missing values from the transaction file. Notice the MISSING statement.

```
data Updated_1;
     update Policy_Holder New_Information;
     by P_Num;
     missing _;
run;

proc print data=Updated_1;
     title 'Updated Records for First Quarter Part 2';
run;
```

Figure 4.

In this example, the MISSING statement identifies the '_' as the special missing value that is in the transaction file and is used to overwrite non-missing values from the master file. The output for this DATA step is on the next page.

Updated Records for First Quarter Part 2

Obs	P_Num	LOB	Gender	Age	Phone
1	10010	Auto	Male	73	
2	10021	Life	Female	.	833-4514
3	10033	Life	Female	28	
4	11010	Auto		.	927-1801
5	20004	Life	Female	41	727-2508
6	20044	Home	Male	27	864-2224

Figure 5.

The output shown here illustrates that missing values in the transaction file did, in fact, overwrite non-missing values in the master file.

The next example illustrates updating a bank account and illustrates another major advantage of using the UPDATE statement. This example uses the following two datasets. In this example, Account is the master dataset while Transaction contains observations that will update the master file.

Account

Acct_Id	Name	Balance	Phone
S10010	Mason	1037	828-4001
S10021	Scott	325	833-4514
S10033	King	2775	811-1949
S11010	Ferrell	1445	927-1108
S20004	Baker	410	727-2508

Transaction

Acct_Id	Type	Amount
S10010	Debit	127.11
S10021	Debit	55.05
S10021	Credit	92.50
S11010	Debit	927.01
S11010	Credit	500.00

Figure 6.

Notice the single observation for acct_id of S11010 in the master file and duplicate rows for that same acct_id in the transaction file. What we need to do is apply both rows in the transaction file to the single row in the master file that matches the value. Also, in this example, we need to deploy the logic of adding

the amount to the balance if the transaction type is 'Credit', and subtracting the amount from the balance if the transaction type is 'Debit'. Here is the DATA step to accomplish this task.

```
proc sort data=transaction;
     by Acct_Id;
run;
data New_Master (keep = Acct_Id  Balance);
     update Account  Transaction;
     by Acct_Id;
     if type = 'Credit' then balance + amount;
     else if type = 'Debit' then balance + - amount;
 run;
proc print data=New_Master;
     title 'New_Master Data Set';
run;
```

Figure 7.

Notice the two SUM statements embedded in the IF-THEN_ELSE statements. The first SUM statement is used in the traditional sense, it is adding amount to balance. The second SUM statement is used to subtract amount from balance.

RENAMING VARIABLES WHILE UPDATING:

A master file (HEALTH) contains health information on it's members. There is a transaction file (FITNESS) that contains new values for WEIGHT. The challenge is that in BOTH datasets the variable that contains the individuals weight is called the same thing. By default, the values of WEIGHT in the transaction file will over write the values of WEIGHT in the MASTER file; which is usually what you want. However, in this situation, we want to end up with two weight variables, one to reflect the OLD weight value, and one to reflect the NEW weight values.

Health - Master

Obs	ID	NAME	WEIGHT
1	1114	Stacy	125
2	1441	Sue	145
3	1750	Jack	189
4	1994	Martin	165
5	2304	James	170

Fitness - Transaction

Obs	ID	NAME	WEIGHT
1	1114	Stacy	119
2	1994	Martin	174
3	2304	James	170

Figure 8.

What is needed to allow us to do this is the ability to rename variables so that there is NO data over write in the Program Data Vector. What we need to do is to map the WEIGHT values from the master file to a variable named ORIG_WEIGHT so that the WEIGHT values from the transaction file go into a different column, thus preventing data over write. See Figure 9.

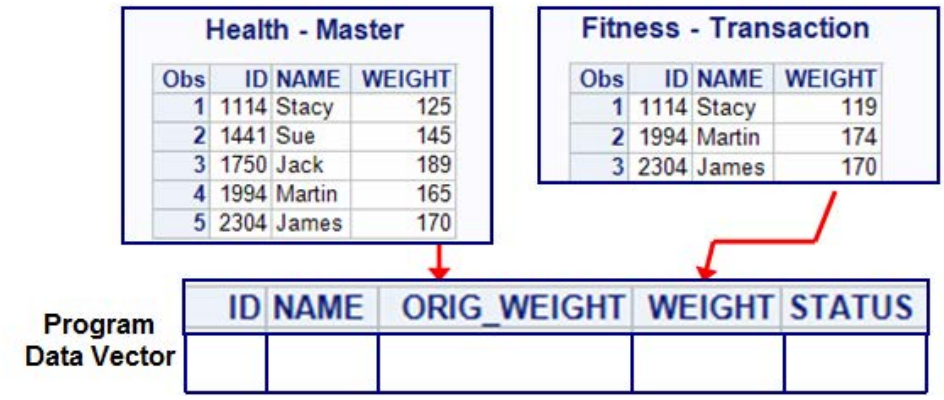

Figure 9.

Is there a way to do this in the DATA step while we are UPDATING at the same time? Of course the answer is Yes. In writing a program to accomplish the above task, the first thing we need to do is to sort both datasets by ID, as shown by the Proc SORT steps here.

```
proc sort data=health;
   by id;
run;
proc sort data=fitness;
   by id;
run;
```

Figure 10.

The next step, is to write the following DATA step to do the UPDATING and renaming:

```
data New_Health;
   update health(rename=(weight=ORIG_WEIGHT) )
          fitness(drop=name );
   by ID;
   if weight<orig_weight  then STATUS='Loss   ';
      else if weight>orig_weight  then status='Gain';
      else status='Same';
   if weight = . then status='Missing';
run;
```

Figure 11.

Notice the RENAME= option on the UPDATE statement. This option allows the values of WEIGHT from the HEALTH dataset to be written to the ORIG_WEIGHT "slot" on the program data vector, thus preventing data from being overwritten. Notice the results below.

Obs	ID	NAME	ORIG_WEIGHT	WEIGHT	STATUS
1	1114	Stacy	125	119	Loss
2	1441	Sue	145	.	Missing
3	1750	Jack	189	.	Missing
4	1994	Martin	165	174	Gain
5	2304	James	170	170	Same

Figure 12.

COMPARING UPDATE WITH MERGE:

The following two datasets are used to show the comparison between the UPDATE statement and the MERGE statement. Notice:

- the missing values in the New Information dataset.
- two rows for P_Num 11010 in the New Information and only one row for that P_Num in the master file (Policy Holders).

Policy Holders

P_Num	LOB	Gender	Age	Phone
10010	Auto	Male	37	828-4001
10021	Home	Female	32	833-4514
10033	Life	Male	27	811-1949
11010	Auto	Female	44	927-1108
20004	Life	Female	41	727-2508

New Information

P_Num	LOB	Gender	Age	Phone
10010			73	
10021	Life		.	
10033	Life	Female	28	
11010			.	927-1801
11010			45	
20044	Home	Male	27	864-2224

Figure 8.

Write two identical DATA steps to process these two datasets, except use an UPDATE statement in one and a MERGE statement in the other.

```
data Updated_2;
     update Policy_Holder New_Information;
     by P_Num;
run;
data Merge_2;
     merge Policy_Holder New_Information;
     by P_Num;
run;

proc print data=Updated_2; title 'Update Results';
run;
proc print data=Merge_2; title 'Merge Results';
run;
```

Figure 9.

The output is shown on the next page.

Update Results

Obs	P_Num	LOB	Gender	Age	Phone
1	10010	Auto	Male	73	828-4001
2	10021	Life	Female	32	833-4514
3	10033	Life	Female	28	811-1949
4	11010	Auto	Female	45	927-1801
5	20004	Life	Female	41	727-2508
6	20044	Home	Male	27	864-2224

Figure 10a.

Merge Results

Obs	P_Num	LOB	Gender	Age	Phone
1	10010			73	
2	10021	Life		.	
3	10033	Life	Female	28	
4	11010			.	927-1801
5	11010			45	
6	20004	Life	Female	41	727-2508
7	20044	Home	Male	27	864-2224

Figure 10b.

In the dataset generated by the UPDATE statement, missing values in the transaction dataset did not overwrite non-missing values in the master dataset. And, both observations for P_Num 11010 in the transaction dataset are applied to the single row for P_Num 11010 in the master dataset.

CONCLUSION

There are some powerful features written into the UPDATE statement that make it very useful in 'updating' SAS datasets. This paper has presented some of the major benefits of the UPDATE statement as well as compare and contrast it to the MERGE statement.

More examples may be shown during the presentation of this paper.

The author can be reached at:
Ben Cochran
The Bedford Group
3224 Bedford Avenue
Raleigh, NC 27607
(919) 741-0370
bentcochran@gmail.com