The Building Blocks of SAS® Datasets – S-M-U
(Set, Merge, and Update)

Andrew T. Kuligowski
FCCI Insurance Group
What is S-M-U?
What is S-M-U?

Shmoo?
What is S-M-U?

Southern Methodist University?
What is S-M-U?

Saskatchewan and Manitoba Universities?
What is S-M-U?

Saskatchewan and Manitoba Ukrainians?
So, what is S-M-U?
A representation of the building blocks to process a SAS Dataset through a DATA Step?

- **SET** statement
- **MERGE** statement
- **UPDATE** statement
SAS datasets are stored in **SAS Data Libraries**.

The standard specification for a SAS dataset contains 2 levels:

```
SAS-Data-Library.SAS-dataset-name
```

If a SAS dataset reference contains only one level, it is implied that the SAS Data Library is the **WORK** library.
SAS Data Libraries must be denoted by a **Libname** (also known as a **Libref**). This is a “nickname” for your Data Library.
How to define a Libref:

1) Let the operating system do it:
   
   ```
   //LREF1 DD DSN=mvs.data.set,DISP=SHR
   ```

2) Let SAS do it as a global statement:
   ```
   LIBNAME libref <engine> 'external file' <options>;
   ```

3) Let SAS do it as a function:
   ```
   X = LIBNAME(libref, 'external file' <,engine> <,options>);
   ```
Special LIBNAME keywords:

To deactivate a Libref:

```
LIBNAME libref CLEAR;
```

To display all active Librefs:

```
LIBNAME libref LIST;
```
The **SET** statement, according to the manual, “... reads observations from one or more existing SAS datasets.”
The simplest example:

```
DATA newdata;
  SET olddata;
RUN;
```

This routine creates an exact replica of a SAS dataset called *olddata* and stores it in a SAS dataset called *newdata*. 
You can combine two or more SAS datasets into one:

```sas
DATA newdata;
   SET olddata1 olddata2;
RUN;
```

The new dataset contains the contents of `olddata1`, followed by `olddata2`.

This is commonly referred to as *concatenation*. 
You can combine two or more SAS datasets:

DATA newdata;
   SET olddata1 olddata2;
RUN;

NOTE: You could also use PROC APPEND to do this. The I/O will be more efficient. However, you can only process two datasets at a time, and you cannot perform any additional processing against the data.
You can combine two or more SAS datasets: Variables that are common to multiple datasets must have consistent definitions.

**IMPORTANT SAFETY TIP!**

**DIFFERENT LENGTHS:**
SAS will use the first definition it encounters, which can result in padding or truncation of values. Fix this with a LENGTH statement prior to the SET statement.
You can combine two or more SAS datasets: Variables that are common to multiple datasets *must* have consistent definitions.

**CHARACTER vs. NUMERIC:**
- **ERROR:** Variable `<varname>` has been defined as both character and numeric.
- `_ERROR_ = 1`
- `RC = 8`
You can do conditional processing for each input dataset:

```sas
DATA newdata;
    SET student(IN=in_stdnt)
        teacher(IN=in_teach);
    IF in_teach THEN salary = salary + bonus;
RUN;
```

IN= specifies a temporary variable that is set to 1 if the corresponding dataset is the source of the current obs. Otherwise, it is set to 0.
You can *interleave* two or more SAS datasets:

```
DATA newdata;
   SET olddatal olddata2;
   BY keyvar;
RUN;
```

The new dataset contains the contents of *olddata1* and *olddata2*, sorted by *keyvar*. **NOTE:** *olddatal* and *olddata2* must each be sorted by *keyvar* prior to executing this DATA step.
You can create a *subset* of your original dataset:

```plaintext
DATA newdata;
  SET olddata1;
  IF keyvar < 10;
RUN;
```

-or-

```plaintext
-DATA newdata;
  SET olddata1;
  WHERE keyvar < 10;
RUN; (continued) ...
```
You can create a subset of your original dataset:

- or -

DATA newdata;
  SET olddata1
    (WHERE=(keyvar < 10));
RUN;

Any of these three methods can be used for subsetting.
You can determine when you have reached the end of your input data:

```sas
DATA newdata;
  SET olddata1 END=lastrec;
  count + 1;
  ttlval = ttlval + value;
  IF lastrec THEN
    avg = ttlval / count;
RUN;
```

Temporary variable `lastrec` is initialized to 0. It is set to 1 when the SET statement processes its last observation.
All of the examples to this point have sequentially processed the input data.

It is also possible to read a SAS dataset using direct access (also known as random access).
Reading a SAS dataset using Direct Access:

```
DATA newdata;
  DO rec=2 TO maxrec BY 2;
    SET olddata1 POINT=rec
       NOBS=maxrec;
    OUTPUT;
  END;
  STOP;
RUN;
```

Let’s take a closer look at the arguments in this example.
DATA newdata;
  DO rec=2 TO maxrec BY 2;
    SET olddata1 POINT=rec ① NOBS=maxrec; ②
    OUTPUT;
  END;
  STOP;
RUN;

① Reads the observation number corresponding to the temporary variable rec.

② Determines the total number of observations in olddata1, and stores that value in temporary variable maxrec.
DATA newdata;
   DO rec=2 TO maxrec BY 2;
      SET olddata1 POINT=rec ①
         NOBS=maxrec; ②
      OUTPUT; ③
   END;
   STOP; ④
RUN;

WHY?

① An OUTPUT statement is necessary in this example.
② A STOP statement is necessary whenever you use the POINT= option.
Let us look at some assorted examples that really exercise the SET statement …
Determining and using NOBS without executing SET statement.

DATA _NULL_
   IF cnt > 100000 THEN
       PUT 'Big Dataset!';
   STOP;
   SET dataset NOBS=cnt;
RUN;

The STOP statement stops further execution – before SET statement executes. BUT, value of variable \texttt{cnt} is determined.
Initializing constants at the beginning of a job:

```plaintext
DATA newdata;
  IF _N_ = 1 THEN DO;
    RETAIN k1-k5;
    SET konstant;
  END;
  SET fulldata;
/* more statements follow */
```

Note that *konstant* is only read once - at the beginning of the DATA step.

The RETAIN statement holds the variables values for subsequent obs.
Using multiple SET statements against the same dataset:

```sas
DATA newdata;
  SET family;
  IF peoplect > 1 THEN
    DO I = 2 TO peoplect;
      SET family;
    END;
/* code snipped here */
```
Using the contents of one dataset to process another:

```plaintext
DATA newdata;
   SET parent;
   IF married = 'Y' THEN
      SET spouse;
   IF childct > 1 THEN
      DO I = 1 TO childcnt;
         SET children;
      END;
/* code snipped here */
```

Can you find the flaw in this logic?
A single SET statement combines two or more SAS datasets vertically.
This is true even when a BY statement is used to interleave records.

<table>
<thead>
<tr>
<th>A-1</th>
<th>B-1</th>
<th>B-2</th>
<th>A-2</th>
<th>A-3</th>
<th>A-4</th>
<th>B-3</th>
<th>A-5</th>
<th>B-4</th>
<th>B-5</th>
</tr>
</thead>
</table>
Multiple SET statements can combine SAS datasets horizontally. This is known as One-to-One Reading.
WARNING: Multiple SET statements can produce undesirable results!

For one thing, the coding can be cumbersome. For another, processing stops once the smallest dataset has reached the End of File.

There is an easier way …
The **MERGE** statement, according to the manual, “... joins corresponding observations from two or more SAS datasets into single observations in a new SAS dataset.”
The simplest example – *One-to-One Merging*

```plaintext
DATA newdata;
  MERGE olddata1 olddata2;
RUN;
```

This routine takes the 1st record in `olddata1` and the first record in `olddata2`, and joins them together into a single record in `newdata`. This is repeated for the 2nd records in each dataset, the 3rd records, etc.
One-to-One Merging is generally safer than One-to-One Reading - but not safe enough in most instances.

For example, the files must be pre-arranged to be in the correct order and to have a one-to-one match on each record.

There is a safer way ...
**Set-Merge-Update**

**MERGE statement**

*Match-Merging* combines observations based on a common variable(s):

```
DATA newdata;
  MERGE olddata1 olddata2;
  BY keyvar(s);
RUN;
```
Match-Merging continued …

DATA newdata;
    MERGE olddata1 olddata2;
    BY keyvar(s) ;
RUN;

Both datasets must be sorted by keyvar(s). The MERGE statement will properly handle one-to-one, one-to-many, and many-to-one matches.
Set-Merge-Update
MERGE statement

Match-Merging continued …

Data1
A
B
C
C

Data2
A
B
B
C

1-1 ("A"), 1-many ("B"), and many-1 merges ("C") within the same datasets.
Set-Merge-Update
MERGE statement

**Match-Merging**
continued …

<table>
<thead>
<tr>
<th>Data1</th>
<th>Data2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

1-1 (“A”), 1-many (“B”), and many-1 merges (“C”) within the same datasets.
**Match-Merging** cannot be used to address many to many merges.
Example: Using MERGE to perform conditional processing.

DATA newdata;
  MERGE olddata1(IN=in_old1)
       olddata2(IN=in_old2);
  IF in_old1 & in_old2  THEN both_cnt + 1;
  IF in_old1 & ¬in_old2 THEN old_only + 1;
  IF ¬in_old1 & in_old2 THEN new_only + 1;
  IF ¬in_old1 & ¬in_old2 THEN no_cnt + 1;
Example: Using MERGE to perform conditional processing.

DATA newdata;
  MERGE olddata1(IN=in_old1)
    olddata2(IN=in_old2);
  IF in_old1 & in_old2  THEN both_cnt + 1;
  IF in_old1 & ~in_old2 THEN old_only + 1;
  IF ~in_old1 & in_old2 THEN new_only + 1;
  IF ~in_old1 & ~in_old2 THEN no_cnt + 1;

Why is this statement unnecessary?
Useful dataset options to use with the SET and MERGE statements …

\( dsname(IN=variable) \)

specifies a temporary True/False variable.

\( dsname(WHERE=(condition)) \)

limits the incoming records to only those meeting the specified condition.

(These two options were mentioned earlier today.)
Useful dataset options to use with the SET and MERGE statements …

\texttt{dsname(/FIRSTOBS=number)}

starts processing at the specified observation.

\texttt{dsname(/OBS=number)}

stops processing at the specified observation.
Useful dataset options to use with the SET and MERGE statements …

\[ \text{dsname(RENAME=(oldname1=newname1))} \]
changes the name of a variable.

This is especially useful when combining datasets that have variables with the same name.
The **UPDATE** statement is similar to the MERGE statement. However, according to the manual, “... the UPDATE statement performs the special function of updating master file information by applying transactions ...”
Differences between the UPDATE and MERGE statements:

UPDATE can only process *two* SAS datasets at a time - *Master* and *Transaction*. MERGE can process *three or more* SAS datasets.
Differences between the UPDATE and MERGE statements:

The BY statement is *required* with UPDATE, but is *optional* with MERGE. (However, it is *very* common to employ a BY statement with MERGE.)
Differences between the UPDATE and MERGE statements:

**UPDATE** can only process 1 record per unique BY group value in the Master dataset.

**UPDATE** can process multiple records in the Transaction dataset. However, they will be overlaying the same Master record - they may be canceling themselves out.
Depicted graphically:

<table>
<thead>
<tr>
<th>Master</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>
**Set-Merge-Update**

**UPDATE statement**

Depicted graphically:

- **Master**
- **Transaction**

**UPDATE**

```
A A
B B
C C
```
UPDATE is easy to code:

DATA newdata;
  UPDATE olddata1 olddata2;
  BY keyvar(s);
RUN;

The only difference in basic syntax between MERGE and UPDATE is the actual word “UPDATE”.
The true utility of the UPDATE statement is best demonstrated by examining the data - BEFORE and AFTER executing UPDATE:
**Set-Merge-Update UPDATE statement**

**UPDATE example - data:**

<table>
<thead>
<tr>
<th>OBS</th>
<th>KEY1</th>
<th>UPDT1</th>
<th>UPDT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AB</td>
<td>1001</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>BC</td>
<td>1002</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>MB</td>
<td>1003</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>NS</td>
<td>1004</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SAS Dataset: XACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBS</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
**Set-Merge-Update**

**UPDATE statement**

**UPDATE example - data:**

SAS Dataset: MASTER
OBS KEY1 UPDT1 UPDT2
1 AB 1001 2

SAS Dataset: XACTION
OBS KEY1 UPDT1 UPDT2 NEW3
1 AB 1111 52 A

SAS Dataset: UPDATIT
OBS KEY1 UPDT1 UPDT2 NEW3
1 AB 1111 52 A

changed  added
changed
UPDATE example - data:

SAS Dataset: MASTER
OBS KEY1 UPDT1 UPDT2
  2  BC  1002  4

SAS Dataset: XACTION
OBS KEY1 UPDT1 UPDT2 NEW3
(No corresponding record)

SAS Dataset: UPDATIT
OBS KEY1 UPDT1 UPDT2 NEW3
  2  BC  1002  4

Record unchanged
### UPDATE example - data:

SAS Dataset: MASTER
OBS KEY1 UPDT1 UPDT2
3 MB 1003 8

SAS Dataset: XACTION
OBS KEY1 UPDT1 UPDT2 NEW3
2 MB 1133 54
3 MB 2133 . C

SAS Dataset: UPDATIT
OBS KEY1 UPDT1 UPDT2 NEW3
3 MB 1133 54

---

**Updated with changes:**
- OBS 3, KEY1 MB 1133 UPDT1 54 UPDT2
- OBS 3, KEY1 MB 2133 missing
- OBS 3, KEY1 MB 1133 changed

---

**Missing:**
- OBS 3, KEY1 MB 2133
UPDATE example - data:

SAS Dataset: MASTER
OBS KEY1 UPDT1 UPDT2
3   MB   1003   8

SAS Dataset: XACTION
OBS KEY1 UPDT1 UPDT2 NEW3
2   MB   1133   54
3   MB   2133   .   C

SAS Dataset: UPDATIT
OBS KEY1 UPDT1 UPDT2 NEW3
3   MB   2133   54   C

changed  changed
Left alone
**Set-Merge-Update**

**UPDATE statement**

**UPDATE example - data:**

SAS Dataset: MASTER
OBS KEY1 UPDT1 UPDT2
4 NS 1004 16

SAS Dataset: XACTION
OBS KEY1 UPDT1 UPDT2 NEW3
4 NS . . D

SAS Dataset: UPDATIT
OBS KEY1 UPDT1 UPDT2 NEW3
4 NS 1004 16 D

---

Left alone added
Left alone
UPDATE example - data:

SAS Dataset: MASTER
OBS KEY1 UPDT1 UPDT2
(No corresponding record)

SAS Dataset: XACTION
OBS KEY1 UPDT1 UPDT2 NEW3
5 SK 4555 65 E

SAS Dataset: UPDATIT
OBS KEY1 UPDT1 UPDT2 NEW3
5 SK 4555 65 E

Record added
Set-Merge-Update
UPDATE statement

UPDATE example - data:

SAS Dataset: MASTER
OBS  KEY1  UPDT1  UPDT2
1   AB    1001   2
2   BC    1002   4
3   MB    1003   8
4   NS    1004   16

SAS Dataset: UPDATIT
OBS  KEY1  UPDT1  UPDT2  NEW3
1   AB    1111   52   A
2   BC    1002   4
3   MB    2133   54   C
4   NS    1004   16   D
5   SK    4555   65   E

XACTION
DT2  NEW3
52   A
54
.
.
.
64

4   NS    .   .   D
5   SK    4555   65   E
Question: What if you want to update your Master dataset with missing values?

Answer: Use the `MISSING` statement to define special missing values.
The **MODIFY** statement, introduced back in Version 6.07, “…extends the capabilities of the DATA step, enabling you to manipulate a SAS data set in place without creating an additional copy,” according to the 6.07 Changes & Enhancements Report.
The **MODIFY** statement, when used with a BY statement, is like an UPDATE statement

... except neither the Master dataset nor the Transaction dataset needs to be sorted or indexed.  (*1)
The **MODIFY** statement, when used with a BY statement, is like an UPDATE statement... except both the Master and Transaction datasets can have duplicate BY values. (*2)
The **MODIFY** statement, when used with a BY statement, is like an UPDATE statement

... except it cannot add variables to the dataset.
The **MODIFY** statement, when used with a **BY** statement, is like a **SET MERGE-UPDATE** statement.
(*1) Failure to sort your datasets prior to a MODIFY/BY statement combination could result in excessive processing overhead.
(*2) The processing rules for duplicate BY values depend on whether those values are consecutive.
“Damage to the SAS data set can occur if the system terminates abnormally during a DATA step containing the MODIFY statement.”
The MODIFY statement can also be used via Sequential (Random) access, similar to a SET statement with the POINT= option.
You could go to the 6.07 Changes and Enhancements report – if you can find one! *(Check in my attic!)*

Better yet, go to the Version 9 online documentation for more information and examples using the MODIFY statement.
PROC SQL

SELECT / FROM similar to SET

SELECT / FROM / JOIN similar to MERGE
The presentation was a *brief* introduction to SET, MERGE, and UPDATE.

In-depth details on options available for each command can be found in the manual(s).
Hands-on experimentation will increase your comprehension.

Even your mistakes can provide a much better education for you than anything I presented – provided you learn from them.
For further information / assistance, contact the author:

KuligowskiConference@gmail.com

Akuligowski@fcci-group.com
My lawyer