STAT 541

Chapter 18: Modifying SAS Data Sets and Tracking Changes

©Spring 2012 Imelda Go, John Grego, Jennifer Lasecki and the University of South Carolina

Modifying Data without Replacing the Data Set

- This can be done using the MODIFY statement in a DATA step.
- Using the MODIFY statement allows replacement, deletion, or appending observations in an existing data set without creating an additional copy of the data.

Modifying Data without Replacing the Data Set (continued)

- The process involves:

1. Using the MODIFY statement to update all observations in a data set
2. Using a transaction data set to make modifications to a data set
3. Using an index to locate observations to modify in a data set

- When modifying data, safeguarding data and tracking the changes can be a concern.

1. Placing integrity constraints on variables in a data set
2. Initiating and managing audit trail file
3. Creating and processing generation data sets

Using the MODIFY Statement

- When a DATA step creates a data set named in a MERGE, UPDATE, or SET statement, SAS creates a second copy of the input data set. Once execution is complete, the original data set is deleted and is replaced by the new data set. The set of variables can change in this case.
- When using a MODIFY statement, SAS DOES NOT create a second copy but updates the original data set. No variables can be added or deleted. The set of variables does not change.

Using the MODIFY Statement (continued)

- There is an implied REPLACE statement at the bottom of the DATA step instead of an OUTPUT statement.
- The MODIFY statement can update:

1. Every observation in a data set
2. Observations using a transaction data set and a BY statement
3. Observations located using an index

Using the MODIFY Statement (continued)

- Data can be lost if the DATA step using a MODIFY statement abnormally terminates. This can damage the master data set.
- Failure recovery includes:

1. Restoring the master file from a backup and restarting the step, or
2. Keeping an audit trail file and using it to determine which master observations have been updated

Modifying All Observations in a SAS Data Set

- Use an assignment statement to the existing variable by specifying the modification through the expression.

DATA *SAS-data-set*;
MODIFY *SAS-data-set*;
existing-variable = expression;
run;

Modifying All Observations in a SAS Data Set (continued)

```
data transactions;
   modify transactions;
   expenses=int(1.07*expenses);
   income=int(income/2);
run;
```

The INT function returns the integer portion of the result.

Modifying Observations Using a Transaction Data Set

- A master data set can be modified with the values of a transaction data set by using the MODIFY statement with a BY statement to apply updates by matching observations.

```
DATA SAS-data-set;
   MODIFY SAS-data-set transaction-data-set;
   BY key-variable;
run;
```

Modifying Observations Using a Transaction Data Set (continued)

- The master data set must be followed by the transaction data set.
- Dynamic WHERE processing is used when determining BY statement matches. Neither data set needs to be sorted according to the variable in the BY statement.
- Having the master data set sorted or indexed and the transaction data sorted requires fewer resources.

Modifying Observations Using a Transaction Data Set (continued)

```
data transactions;
   modify transactions newactions;
   by customerID;
run;
```

Handling Duplicate Values When Using MODIFY and BY Statements

- WHERE processing starts at the top of the master data set and finds the first match and updates it.
- If duplicate values of the BY variable are in the master data set, only the first observation in the group of duplicate values is updated because WHERE processing begins at the top of the data set and updates the first match.
- If duplicate values of the BY variable are in the transaction data set, the duplicate values overwrite each other so that the last value in the group of duplicate transactions is the result in the master data set.

Handling Duplicate Values When Using MODIFY and BY Statements (continued)

- Avoid overwriting duplicate values by writing an accumulation statement so that all observations in the transaction data set are added to the master observations.
- If duplicate values exist in both the master and transaction data sets, you can use PROC SQL to apply the duplicate values in the transaction data set to the duplicate values in the master data set in a one-to-one correspondence.

Handling Missing Values in the Transaction Data Set

- If there are missing values in the transaction data set, SAS does not replace the data in the master data set with missing values unless they are special missing values.
- Use the UPDATEMODE= option in the MODIFY statement to specify how missing values in the transaction data set are handled.

SYNTAX

MODIFY *master-data-set transaction-data-set*

UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK

MISSINGCHECK (default) prevents missing values in the transaction data set from replacing values in the master data set unless they are special missing values, while NOMISSINGCHECK allows it but special missing values still replace the values in the master data set.

Modifying Observations Located by an Index

- You can use a BY statement to access values you want to update in a master data set by matching. When you have an indexed data set, you can use the index to directly access the values you want to update. The steps are:

1. Use a MODIFY statement with the KEY= option to name an indexed variable to locate the observations for updating.
2. Use another data source (typically a SAS data set named on a SET statement or an external file read by an INPUT statement) to provide a like-named variable whose values are supplied to the index.

Modifying Observations Located by an Index (continued)

MODIFY *master-data-set* KEY=*index-name*;

Index-name is the name of the simple or composite index that you are using to locate observations.

The KEY= option requires that:

1. you explicitly specify the update. No automatic overlay of non-missing values in the transaction data set occurs as it does with the MODIFY/BY method.
2. each observation in the transaction data set must have a matching observation in the master data set. If there are multiple observations in the transaction data set per one master observation, only the first observation in the transaction data set is applied. Other observations generate run time errors and terminate the DATA step (unless the UNIQUE option is used).

Modifying Observations Located by an Index (continued)

```
proc datasets;
   modify olddata;
      index create id / unique;
data olddata;
  set newdata
(rename =(oldvalue1=newvalue1
                oldvalue2=newvalue2));
  modify olddata key=ID;
    oldvalue1=newvalue1;
    oldvalue2=newvalue2;
run;
```

OldData Before Modification

ID	Oldvalue1	Oldvalue2
1	100	300
2	200	400

NewData

ID	Oldvalue1	Oldvalue2
1	1001	3003
2	2002	4004

OldData After Modification

ID	Oldvalue1	Oldvalue2
1	1001	3003
2	2002	4004

Modifying Observations Located by an Index: Handling Duplicate Values

- If there are duplicates in the master data set, only the first occurrence is updated.
- Duplicate index values in the transaction data set might cause problems.
- If there are *nonconsecutive duplicates* in the transaction data set, the first observation in the master data set is updated with the last duplicate transaction value.
- An error results if there are *consecutive duplicates* in the transaction data set where some do not have a match in the master data set.

Modifying Observations Located by an Index: Handling Duplicate Values (continued)

MODIFY *SAS-data-set* KEY=*index-name/UNIQUE*;

- UNIQUE option applies multiple transactions to one master observation by returning to the top of the index when looking for a match for values in the transaction data set.

Controlling the Update Process

- When a DATA Step contains a MODIFY statement, SAS will process the data a certain way. If the OUTPUT, REPLACE, and REMOVE statements are not present, there is actually an *implied* REPLACE statement at the end of the DATA Step. When this happens, SAS writes the current observation to its original place in the SAS data set.
- To override the default, explicitly use the OUTPUT, REPLACE, or REMOVE statements.
- If any one of these three are used, you must explicitly program each action that needs to be taken. These three statements can be used together as long as the sequence is logical.

Controlling the Update Process (continuation)

- OUTPUT; specifies that the current observation be written to the end of the data set
- REPLACE; specifies that the current observation be rewritten to the same location in the data set
- REMOVE; specifies that the current observation be deleted from the master data set
- If the OUTPUT statement is used with the REPLACE or REMOVE statement, the OUTPUT statement should be executed *after* any REPLACE or REMOVE statement to ensure the integrity of the index position.

Controlling the Update Process (continuation)

```
data master;
  set transaction;
  modify master key = id;
  a = b;
  if code= 'no' then remove;
    else if code= 'yes' then replace;
    else if code= 'new' then output;
run;
```

- Delete rows with REMOVE.
- Update rows with REPLACE.
- Append rows with OUTPUT.

Controlling the Update Process- Monitoring I/O Conditions

- Using _IORC_ with %SYSRC
  - %SYSRC is a macro that allows you to check the value of _IORC_ (created when using MODIFY) for specific Input/Output conditions/errors
  - Some “errors” aren’t really errors, and the value _ERROR_ can be reset to 0 to allow execution to continue
  - Mnemonics
    - _DSENMR (No match in master data set for observation—used with BY)
    - _DSEMTR (Multiple unmatched observations in master data set—used with BY)
    - _DSENOM (No match in master data set for observation—used with KEY)
    - _SOK (match found)

Integrity Constraints

- We studied CHECK, NOT NULL, UNIQUE and PRIMARY KEY in SQL in Chapter 5
- Integrity constraints can also be set up using PROC DATASETS

```
proc datasets nolist;
modify lab2012;
ic create check_USC=check(where=(USC in ('N'
  'Y' 'y' 'n'))message="Incorrect code for
  USC");
quit;
```

- Integrity constraints can be removed with IC DELETE

Audit Trails

- Audit trails can be used to track changes to a data set made in
  - Viewtable
  - MODIFY in the DATA step
  - UPDATE, INSERT, DELETE in PROC SQL
- Changes are stored in an audit file
- SAS commands (CREATE TABLE, PROC SORT, DATA step) can delete the audit trail

Audit Trail Example

```
proc datasets nolist;
audit nonprof;
initiate;
quit;
data nonprof;
modify nonprof;
years=year(today())-since;
run;
```

Audit Trail Example

- SAS creates the audit file WORK.nonprof.audit
- You can view the audit file with PROC CONTENTS

```
proc contents
  data=nonprof(type=audit);
run;
```

Audit File Variables

- In addition to data set variables, the audit file contains metadata
  - _ATOPCODE_ (Type of operation)
  - _ATDATETIME_ (Date and time)
  - _ATOBSNO_ (Affected observation #'s)
  - _ATUSERID_

Audit File Variables

- You can use the LOG statement to limit data that appears in the audit file, typically by _ATOPCODE_ class
- User variables can be added to the audit file as well
  - Once created, they can be updated and then saved in the audit trail file
- The audit trail can be suspended, resumed and ended

Generation Data Sets

- Multiple version (generations) of data sets can be saved each time a data set is replaced
- The generation number is typically small
- The naming is a little counterintuitive

Generation Data Sets

```
proc datasets nolist;
modify nonprof (genmax=4);
quit;
data nonprof; set nonprof;
years=year(today())-since;
run;
proc sort data=nonprof;
by memberid; run;
```

Generation Data Sets

- The current data set is still `nonprof`
- The data set created by SET is `nonprof#002`
- The original data set is `nonprof#001`

Generation Data Sets

You can use gennum to refer to particular data sets:

```
proc print data=nonprof
  (gennum=2); run;
proc print data=nonprof
  (gennum=0); run;
proc sgplot
  data=nonprof(gennum=-2);
histogram years; run;
```

Generation Data Sets

- Generations can be deleted or assigned new names in PROC DATASETS
- HIST (all historical versions) and ALL keywords can be used with DELETE