

Data Management And Manipulation: Examples For Normalized Databases and Spreadsheets

Marlene Goormastic and Shelly Sapp, Cleveland Clinic Foundation, Cleveland, OH

Abstract

Advances in the ability to transfer data from a variety of computer database and spreadsheet packages into SAS® datasets has made data management and manipulation an increasing challenge for the SAS programmer. Conversion of EXCEL and Lotus files, for example, often leads to datasets with non-numeric and poorly defined variable fields. Several functions including SCAN, SUBSTR, and TRIM will be presented for manipulation of character defined data. The use of SAS/ACCESS® for relational databases such as ORACLE® and Rdb® has added a new level of complexity to programming. These relational databases require the creation of a single analysis dataset using multiple tables, which are often normalized. The building of consolidated datasets using the basic application of procedures such as TRANSPOSE and SQL along with the RETAIN, MERGE, and KEEP functions will be demonstrated. This talk will be of interest to all SAS programmers, beginner and seasoned, who work with less than perfect data.

Character Variables

The conversion of spreadsheets to SAS system files using DBMS COPY or other conversion software often results in variables that are not properly defined for analysis. Numeric fields are defined as character, and dates are defined as text strings. Several SAS functions are extremely useful when dealing with these problems.

Variables that are really numeric but defined as character can be converted by adding zero or by using an input statement.

EXAMPLE 1: CONVERTING A CHARACTER VARIABLE TO NUMERIC

```
age=c_age + 0; or age=input(c_age, 3.);
```

A text field which is really a date variable with imbedded slash (/) can be handled using the SCAN function. SCAN searches a character/text variable until it encounters a delimiter such as a slash, comma or blank space.

EXAMPLE 2: CONVERTING A CHARACTER DATE WITH BACK SLASHES INTO A NUMERIC SAS DATE

```
day1=SCAN(datevar, 1);
```

Puts the text before the 1st delimiter into DAY1 variable.

```
mth1=SCAN(datevar, 2);
```

Puts the text before the 2nd delimiter into MTH1 variable.

```
yr1=SCAN(datevar, 3);
```

Puts the text before the 3rd delimiter into YR1 variable.

```
newdate=MDY(mth1,day1,yr1);
```

Creates date variable called newdate.

Or combining all three scans:

```
newdate=
```

```
mdy(scan(datevar,1),scan(datevar,2),scan(datevar,3));
```

When converting data from disk, leading or trailing blanks will frequently become part of a character string. This makes programming more difficult and listing of the data extremely lengthy. The following procedures can help to correct this problem.

TRIM(varname) - eliminates trailing blanks
LEFT(varname) - left justifies variable
eliminating leading blanks

TRIM(LEFT(varname)) - takes care of both leading and trailing blanks
 SUBSTR(varname,n1,n2) - creates new character variable starting in the n1 position for n2 characters

By combining the INPUT and SUBSTR functions, a SAS date variable can be created from a character date variable.

EXAMPLE 3: COMBINING INPUT AND SUBSTR

```
chardate = '19900101'
numdate=input(substr(chardate,3,6),ymmdd6.);
```

The following examples are useful for maximizing the information in a report or listing.

EXAMPLE 4: SHORTENING A TEXT FIELD

```
newsex=SUBSTR(sex,1,1);
Male becomes M and Female becomes F
```

The concatenate function (||) allows character variables or strings to be linked together.

EXAMPLE 5: CREATING NAME FROM FIRST AND LAST NAME

```
NAME=
TRIM(LEFT(lname))||', '||TRIM(LEFT(fname));
```

lname	fname		name
Johnson	Robert	becomes	Johnson, Robert

Numeric variables can be converted to character using the PUT statement. This allows a numeric variable to be concatenated with a character variable. If, for instance, you have a patient status variable (pt_stat) with the responses 'ALIVE' or 'DEAD' and a numeric status date (stat_dt), then a single character variable can be created containing both pieces of information.

EXAMPLE 6: CREATING TEXT STRING WITH STATUS AND DATE

```
newstat=SUBSTR(pt_stat,1,1)||put(stat_dt,mmddyy8.);
```

pt_stat	stat_dt		newstat
ALIVE	10/05/94	becomes	A10/05/94

Accessing Data From Relational Databases

Data from several relational database packages can be accessed by SAS programs through one of two methods. The first is the creation of SAS accesses and views through the use of SAS/ACCESS, after which either the views can be used in a data step or be referenced using PROC SQL. This method is well documented in the SAS/ACCESS manual. The other method is the use of PROC SQL to directly connect to the database. The advantages of directly accessing the database are 1) the selection criteria for the dataset are documented in the program (unlike window created views) and 2) the process of going through the cumbersome SAS/ACCESS windows to create the table accesses and views is avoided. Since this method is less well known, an example is presented below.

All SQL functions such as joining multiple tables and case expression can be incorporated as usual. Database variable names longer than eight characters will get truncated.

EXAMPLE 7: USING PROC SQL TO DIRECTLY ACCESS AN ORACLE DATABASE ON A UNIX SYSTEM

```
LIBNAME SAVE 'unix account and folder name';
PROC SQL NOPRINT;
CONNECT TO ORACLE AS dbname
(USER=username PASS=password PATH='path');
CREATE table tablename as
SELECT * or list of SAS variable names
      separated by commas
FROM CONNECTION TO dbname
(SELECT * or list of ORACLE variables
      separated by commas
```

```

FROM tablename
WHERE selection criteria);
%PUT &SQLXRC &SQLXMSG; (optional)
(This provides the return codes from the relational
database-useful for debugging).
DISCONNECT FROM ORACLE; (optional)
QUIT; (optional)

```

Merging Many To Many Records

Combining data from two datasets is straight forward as long as the records in each data file have the same primary keys (ie. fields which uniquely identify each row). This is a "one to one" merge. It is also straight forward when only one of the files has multiple occurrences of the primary key(s) ("one to many" merge). The difficulty arises when the merge variable(s), usually all or a subset of the primary keys, does not uniquely identify records in either dataset ("many to many" merge). Combining data from normalized tables is one situation where this could occur. Due to normalization, these tables have several primary keys. In a "many to many" merge, only a subset of these keys are utilized as the merging variable(s) to combine the tables to create a single dataset.

Two ways to accomplish a "many to many" merge are using PROC SQL or merging after a PROC TRANSPOSE. Merging two tables (or datasets) by fields which are not unique in at least one table will usually not result in the desired dataset. The data step MERGE joins one for one with any remaining observations being merged with the last record of the shorter file. On the other hand, PROC SQL will merge in such a manner as to provide all possible combinations. An illustration is provided in the next column.

```

Dataset 1
Larry 1
Larry 2
Larry 3

```

```

Dataset 2
Larry A
Larry B

```

```

Merged by Name
Larry 1 A
Larry 2 B
Larry 3 B

```

```

Proc SQL
Larry 1 A
Larry 1 B
Larry 2 A
Larry 2 B
Larry 3 A
Larry 3 B

```

As another example, patients may have multiple procedure records containing their patient ID, date of procedure and type of procedure (DATASET=PROC). They might also have multiple catheterization visit records with their patient ID, date of catheterization, and right coronary artery (RCA) stenosis (DATASET=CATH). Each procedure may have multiple associated morbidity records containing their patient ID, date of procedure, and morbidity type (DATASET=MORB). The datasets given below will be used in the remaining examples.

CATH			PROC		
ID	CATHDATE	RCA	ID	PROCDATE	TYPE
1	01/01/90	80	1	01/02/90	1
1	01/01/91	60	1	01/02/91	3
1	01/01/92	30	2	02/02/90	1
2	02/01/90	70	2	02/02/91	6
2	06/01/90	45	3	03/02/90	1
2	02/01/91	55	3	03/02/90	2
3	03/01/90	75			

MORB		
ID	PROCDATE	MORBID
1	01/02/90	1
2	02/02/91	2
3	03/02/90	3
3	03/02/90	4

One might want to get a listing of all patients' procedure dates, morbidities associated with the procedures and whether the patient had a procedure type of 1, coronary bypass graft (CABG). Both programming approaches, transposing the data then merging or PROC SQL, can be used to obtain this listing.

With the first approach, the procedure and morbidity datasets are transposed creating records with unique rows per id and procedure date. Then the transposed records are merged together by id and procedure date and dichotomous variables are created using arrays.

EXAMPLE 8 : PROC TRANSPOSE/MERGE

```
proc transpose data=morb out=tranmorb prefix=mrbr;
  var morbid;
  by id procdate;

proc print data=tranmorb;
  title 'Proc Transpose of the Morbidity Dataset';
  format procdate mmdyy8.;

proc transpose data=proc out=tranproc prefix=typ;
  var type;
  by id procdate;

proc print data=tranproc;
  title 'Proc Transpose of the Procedure Dataset';
  format procdate mmdyy8.;

data ex8;
  merge tranproc(in=in1) tranmorb(in=in2);
  by id procdate;
  if in1;
  array m(2) mrbr1-mrb2;
  do i=1 to 2;
    if m[i]=1 then bleed=1;
    if m[i]=2 then mi=1;
    if m[i]=3 then sepsis=1;
    if m[i]=4 then death=1;
  end;
  array z(4) bleed mi sepsis death;
  do j=1 to 4;
    if z[j]=. then z[j]=0;
  end;
  if typ1=1 or typ2=1 then cabg='Yes';
  else cabg='No';
  keep id procdate bleed mi sepsis death cabg;
  format procdate mmdyy8.;

proc print data=ex8;
  title 'Final Listing Using Transpose and Merge';
  run;
```

EXAMPLE 8: OUTPUT

Proc Transpose of the Morbidity Dataset

OBS	ID	PROCDATE	_NAME_	MRB1	MRB2
1	1	01/02/90	MORBID	1	.
2	2	02/02/91	MORBID	2	.
3	3	03/02/90	MORBID	3	4

Proc Transpose of the Procedure Dataset

OBS	ID	PROCDATE	_NAME_	TYP1	TYP2
1	1	01/02/90	TYPE	1	.
2	1	01/02/91	TYPE	3	.
3	2	02/02/90	TYPE	1	.
4	2	02/02/91	TYPE	6	.
5	3	03/02/90	TYPE	1	2

Final Listing Using Transpose and Merge

OBS	ID	PROCDATE	BLEED	MI	SEPSIS	DEATH	CABG
1	1	01/02/90	1	0	0	0	Yes
2	1	01/02/91	0	0	0	0	No
3	2	02/02/90	0	0	0	0	Yes
4	2	02/02/91	0	1	0	0	No
5	3	03/02/90	0	0	1	1	Yes

In the second approach, the morbidity and procedure datasets are joined through Proc SQL. This produces a dataset with all possible combinations of morbidities and procedure types for each patient ID and procedure date. Then, the RETAIN option is used to maintain the current value of the dichotomous variables, initialized to zero using the "if first" statement. Next, if the patient had a morbidity or a CABG, then the associated dichotomous variable is changed to one. Finally, the last record per id and procedure date is outputted with the "if last" statement.

EXAMPLE 9: PROC SQL

```
proc SQL;
  create table proc morb as
  select proc.*, morb.morbid
  from proc left join morb
  on proc.id=morb.id;

proc sort data=proc morb;
  by id procdate;
```

```

proc print data=proc morb;
  title 'Proc SQL of the Morbidity and Procedure
        Datasets';
  format procdate mmddyy8.;

data ex9;
  set proc morb;
  by id procdate;
  length cabg $3.;
  retain bleed mi sepsis death cabg;
  if first.procdate then do;
    bleed=0; mi=0; sepsis=0; death=0; cabg='No';
  end;
  if morbid=1 then bleed=1;
  if morbid=2 then mi=1;
  if morbid=3 then sepsis=1;
  if morbid=4 then death=1;
  if type=1 then cabg='Yes';
  if last.procdate then output;
  keep id procdate bleed mi sepsis death cabg;
  format procdate mmddyy8.;

proc print data=ex9;
  title 'Final Listing Using Proc SQL';
run;

```

EXAMPLE 9: OUTPUT

Proc SQL of the Morbidity and Procedure Datasets

OBS	ID	PROCDATE	TYPE	MORBID
1	1	01/02/90	1	1
2	1	01/02/91	3	.
3	2	02/02/90	1	.
4	2	02/02/91	6	2
5	3	03/02/90	2	4
6	3	03/02/90	1	4
7	3	03/02/90	2	3
8	3	03/02/90	1	3

Final Listing Using Proc SQL

OBS	ID	PROCDATE	BLEED	MI	SEPSIS	DEATH	CABG
1	1	01/02/90	1	0	0	0	Yes
2	1	01/02/91	0	0	0	0	No
3	2	02/02/90	0	0	0	0	Yes
4	2	02/02/91	0	1	0	0	No
5	3	03/02/90	0	0	1	1	Yes

One may also want to select the most recent catheterization information prior to a patient's procedure. First, PROC SQL is used to get the maximum catheterization date before each procedure date.

EXAMPLE 10: PROC SQL: MAX FUNCTION

```

proc SQL;
  create table maxcath as
  select id, procdate, max(cathdate) as cath_dt
  from
    (select * from proc as p
     left join cath as c
       on p.id=c.id and procdate>=cathdate)
  group by id, procdate;

proc print data=maxcath;
  title 'Maximum Catheterization Date Prior to the
        Procedure Date Using Proc SQL';
  format procdate cath_dt mmddyy8.;
run;

```

EXAMPLE 10: OUTPUT

Maximum Catheterization Date Prior to the Procedure Date Using Proc SQL;

OBS	ID	PROCDATE	CATH_DT
1	1	01/02/90	01/01/90
2	1	01/02/91	01/01/91
3	2	02/02/90	02/01/90
4	2	02/02/91	02/01/91
5	3	03/02/90	03/01/90

PROC SQL can then be used to perform a three-way merge to obtain the additional information (RCA stenosis) in the most recent catheterization along with the type of procedure.

EXAMPLE 11: PROC SQL: THREE-WAY MERGE

```

proc SQL;
  create table proccath as
  select mp.*, rca
  from
    (select * from proc as p
     left join maxcath as mc
       on p.id=mc.id and
         p.procdate=mc.procdate) as mp
  left join cath as c
    on c.id=mp.id and c.cathdate=mp.cath_dt;

proc print data=proccath;
  title 'Listing of Procedure Type and RCA
        Stenosis';
  format procdate cath_dt mmddyy8.;
run;

```

EXAMPLE 11: OUTPUT

Listing of Procedure Type and RCA Stenosis

OBS	ID	PROCDATE	TYPE	CATH_DT	RCA
1	1	01/02/90	1	01/01/90	80
2	1	01/02/91	3	01/01/91	60
3	2	02/02/90	1	02/01/90	70
4	2	02/02/91	6	02/01/91	55
5	3	03/02/90	1	03/01/90	75
6	3	03/02/90	2	03/01/90	75

In general, PROC SQL requires less programming and is more intuitive. When working with large databases PROC SQL is generally much faster than a comparable data step statements. However, it requires more temporary space when performing "many to many" merges since every possible combination is created.

Summary

Database management problems can arise from either poorly defined variables or from complex database structures. Often the first problem is a result of data transferred from spreadsheets created by an investigator. The second problem can occur from the normalization of tables in a relational database. Several SAS functions and procedures exist which are helpful when working with either of these problems. These functions and procedures allow for more efficient and intuitive data management and manipulation.

Contact Information

Marlene Goormastic, MPH
Transplant Center
Cleveland Clinic Foundation
9500 Euclid Ave.
Cleveland, OH 44195

e-mail: mgoormas@bio.ri.ccf.org

Shelly Sapp, MS
Department of Biostatistics and
Epidemiology
Cleveland Clinic Foundation
9500 Euclid Ave.
Cleveland, OH 44195

e-mail: ssapp@bio.ri.ccf.org

SAS and SAS/ACCESS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

ORACLE and Rdb are registered trademarks or trademarks of ORACLE Corporations.