Paper CC-05-2011

Who are You? Use of Soundex and Merge to Cross-Walk one Dataset to Another

Misty Johnson, State of Wisconsin Dept of Health Services, Madison, WI, USA

MWSUG 2011 Kansas City, MO

Abstract

Spelling variations of a first name and multiple personal identification numbers (ID's) can be a problem when trying to merge information in two datasets. Some systems accept multiple forms of personal ID within one data field, while others separate out each type of ID, such as Social Security Number (SSN), Medicaid (MA) Identification number, and Recipient Identification Number (Recipient ID). This paper demonstrates the use of SOUNDEX and MERGE to cross-walk information from one reporting system that used one field to hold an SSN or an MA number or a Recipient ID, to a second reporting system that held each of these personal ID's in three separate variables.

A series of merges between the starting dataset and the larger dataset were used to find the possible matches. Each DATA step produced two files: one with possible matches, the other with the remaining, un-matched observations. This process was repeated until all observations had a match. Lastly, the possible matches were verified by the combination of first name, last name and date of birth. The SOUNDEX function was used to match "Jenny" with "Jennifer" for the same last name and same date of birth.

Systems are continually changing and evolving. This paper has applications to other industries in which any identification field is different, or has changed, and must be cross-walked to a second identification system. The SOUNDEX function is a useful tool for verifications of similar character strings between datasets that otherwise would need to be verified visually.

Introduction

Personal Identification numbers in healthcare have quickly changed in recent years. In the past, many health care services were recorded via the participant's nine-digit Social Security Number (SSN). Later, Medicaid services were recorded via the participant's Medicaid (MA) Identification number, which for many was simply the 9-digit SSN number with a zero (0) at the end. In recent years, a new identification number, Recipient ID, has come into use for Medicaid services. Recipient ID is also a 10-digit number but is completely different from the person's SSN or MA ID, thereby not disclosing a person's SSN to minimize identity theft.

Reporting systems that remain in existence through these changes have evolved to accept these new forms of identification. The existing reporting system for the State of Wisconsin Medicaid Waiver services, the Human Services Reporting System (HSRS), adapted to the changes by allowing the entry of either a nine-digit SSN, a 10-digit MA ID or a 10-digit Recipient ID into the "SSN/MA" field. This was a good adaptation, because the ID field was flexible, and any of the three would be accepted.

However, the acute care Medicaid database for the State of Wisconsin, interChange, relied heavily on the new Recipient ID. Older forms of identification, such as SSN and MA number, were retained only in the history file along with the new Recipient ID. Inevitably, vital pieces of information, such as prior Medicaid Waiver service costs, were stored with one type of identification number in the older database, while the new database contained other pieces of information, such as the start date for a new service package, were stored with the new identification number in the new database. This created the need for a cross-walk between the older dataset (HSRS), with a field that could be either of three identification numbers (SSN or MA ID or Recipient ID) to a newer dataset (interChange) that used Recipient ID, with the SSN and MA ID retained only in the history file. The cross-walk was achieved by a series of DATA step merges, logically attempting to merge each individual by MA ID, then Recipient ID, and lastly SSN. Potential matches were verified by a successful match on first name, last name and date of birth. The SOUNDEX function proved very useful to verify matches that had a spelling variation on the first name.

Preparation of Data

Both datasets needed a bit of preparation work before the datasets could be merged. The matching variables needed to be formatted and named the same for ease of matching. I also needed to document the number of observations that sought a match.

Preparation of the Old Dataset

Data in the old dataset, HSRS, had to be prepared to be merged with data in the new dataset, interChange. The observations in need of a match were unduplicated on an unrelated "client episode ID" field (named "srtepdcd") using PROC SORT, shown below:

```
PROC SORT DATA=DIS5 NODUPKEY;
BY SRTEPDCD;
RUN;
```

The "SSN/MA" variable could hold either a nine-digit SSN, a 10-digit MA number or a 10-digit Recipient ID. There was no explanation to what the number was, other than the following assumptions:

- A nine-digit number was most likely the SSN
- A 10-digit number that ended in zero could be the MA number (SSN + 0)
- A 10-digit number that did not end in zero was most likely the Recipient ID

The goal was to link this variable to one of the three variables in the history file of the new dataset. The "SSN/MA" variable was transformed into three new variables, since it was unknown what it really represented. All observations in need of a match had an MA ID and a Recipient ID in either the old or new dataset; however, it was possible that their record in the old dataset had only an SSN on file that was never updated with an MA ID or Recipient ID. For these observations, the best estimate of their MA ID was the SSN concatenated with a zero (0).

The following is how the three "matching" variables from the "SSN/MA" field were defined. All variables created were formatted to match the variables they'd later be matched with:

- If "SSN/MA" length=9, define it as a new variable called social_security_num. Also define a variable called previous_medicaid_id and put "SSN/MA", concatenated with a zero at the end, in this field. This is the only estimate of the MA ID when given only the SSN.
- If SSN/MA length=10, then the value is either a Recipient ID or a Medicaid ID. Define it as a recipient_id and also as a previous_medicaid_id. This ten-digit number may also be the SSN+0, so a SUBSTR function was used to put the first 9-digits into a field called social_security_num.

The old dataset stored the client name as "Last, First, MI" while the new dataset stored client name as three separate variables. A SCAN function was used to separate out the first and last name; each was formatted to match the new dataset. Birth date was put into a new variable called cltbthdt_hsrs to preserve it since the birth date variable, cltbthdt, will be used to match with the new dataset. Important note: when you merge two datasets having variables of the same name, the variables from the second dataset will overwrite the first. This is why the birth date was put into a new variable to preserve it for verification of the match.

The SAS code that prepared the old dataset (named "DIS5") for matching to the new dataset is shown below:

```
*TRANSFORM CLTSSNMA INTO EITHER AN SSN OR RECIP_ID OR OLD MA ID, DEPENDING ON LENGTH.
SPLIT NAME INTO FIRST AND LAST. ;
DATA DIS6;
FORMAT SOCIAL_SECURITY_NUM $9. PREVIOUS_MEDICAID_ID RECIPIENT_ID $12. LAST_NAME $20.
       FIRST_NAME $15. CLTBTHDT_HSRS DATE9.;
SET DIS5;
IF (LENGTH(COMPRESS(CLTSSNMA)))=9 THEN DO;
       SOCIAL_SECURITY_NUM=LEFT(COMPRESS(CLTSSNMA));
       PREVIOUS_MEDICAID_ID=LEFT(COMPRESS(CLTSSNMA)||'0'); END;
ELSE IF (LENGTH(COMPRESS(CLTSSNMA)))=10 THEN DO;
       RECIPIENT_ID=LEFT(COMPRESS(CLTSSNMA));
       PREVIOUS_MEDICAID_ID=LEFT(COMPRESS(CLTSSNMA));
       SOCIAL_SECURITY_NUM=SUBSTR(LEFT(COMPRESS(CLTSSNMA)),1,9); END;
LAST_NAME=SCAN(CLTNAME,1);
FIRST_NAME=SCAN(CLTNAME,2);
CLTBTHDT_HSRS=CLTBTHDT;
RUN;
```

Preparation of the New Dataset

The new dataset, interChange, held the personal identifier in three different variables in the history file as previously noted. The DATEPART function was used to store the client birth date as a new variable called cltbthdt that was formatted to match the matching variable of the same name in the old dataset. To speed up the run time, only the relevant variables in the new dataset were kept. The SAS code that prepared the new dataset for matching is shown below:

```
DATA RECIP_BASE;
```

```
FORMAT CLTBTHDT DATE9.;
SET DSS.RECIP_BASE (KEEP= RECIPIENT_ID UNIVERSAL_ID LAST_NAME FIRST_NAME
      COUNTY_CODE BIRTH_DATE SOCIAL_SECURITY_NUM MEDICAID_CASE_NUM
      PREVIOUS_MEDICAID_ID MASTER_CLIENT_INDEX ADDED_DATE);
CLTBTHDT=DATEPART(BIRTH_DATE);
RUN;
```

Noting the Number of Observations in Need of a Match

This program was used to report participant enrollment dates in a new Medicaid program. Since all observations needed a match in the new dataset, I needed to ensure that all observations found a match. A system function was used to count the number of observations for whom it sought a match; a LET statement in open code to put this number into a macro variable for reference only.

```
*************************************************************;
* NOTE HOW MANY PEOPLE YOU HAVE                             ;
* THIS MACRO VARIABLE WILL SERVE AS A CHECK FOR DUPLICATES  ;
*************************************************************;
%let nobs_ORIGINAL=%sysfunc(attrn(%sysfunc(open(DIS6,i)),nobs));
%PUT NOBS_ORIGINAL=&NOBS_ORIGINAL.;
```

Matching the Old to the New via a Series of MERGEs

Observations were matched between the two datasets on the “SSN/MA” field, which was further defined as three possible identifiers, and the combination of first name, last name and birth date.

Matching of the datasets took place as four different DATA steps, each following a similar format. Both datasets were sorted by the matching variable, such as previous_medicaid_id. Next, a DATA step was used to merge the observations in both datasets on this matching variable. As previously stated, when two datasets having variables of the same name are merged, the variables from the second dataset will overwrite the first. Therefore, I was careful to always put the new dataset last in the MERGE statement, because I had already preserved the variables of the old dataset by re-defining them. These re-defined variables would later be used to confirm the possible matches.

Two different IF-THEN-OUTPUT statements created two resulting datasets, one (called “TryMatch1”) with the observations that found a match, and the other (called “Remainder1”) with the remaining observations that will seek a match in the next DATA step. The SAS code for the first matching variable, previous_medicaid_id, is shown below:

```
*FIRST ATTEMPT:  MATCH ON THE OLD MA # (SSN+0);
PROC SORT DATA=DIS6; BY PREVIOUS_MEDICAID_ID; RUN;
PROC SORT DATA=RECIP_BASE; BY PREVIOUS_MEDICAID_ID; RUN;

DATA TRYMATCH1 REMAINDER1;
* IF IT MATCHES, 2ND DATASET WILL OVERWRITE ANY VARS WITH THE SAME NAME;
MERGE DIS6(IN=MINE) RECIP_BASE(IN=ALL);
BY PREVIOUS_MEDICAID_ID;
IF MINE AND ALL THEN OUTPUT TRYMATCH1;
ELSE IF MINE AND NOT ALL THEN OUTPUT REMAINDER1;
RUN;
```

Once a potential match was found for each observation in the old dataset, it was moved into the output dataset; the input dataset was not the original “old” dataset (named “DIS6”), but only the remainder, such as “Remainder1”. In contrast, the “new” dataset (named “Recip_Base”) was used in its entirety each time; giving an equal chance for all observations to obtain a match.

This code was executed three more times using the matching variables of Recipient ID, SSN and lastly a combination of Last Name, First Name and Birth Date (all three variables had to match). Note that on subsequent runs of this code, which would’ve lent itself well to a macro, the resulting output files were named in a similar fashion with the run count at the end, such as “TryMatch2” and “Remainder2”, etc. The second run of this code is demonstrated below:

```
*SECOND ATTEMPT:  MATCH ON THE NEW RECIPIENT ID;
PROC SORT DATA=REMAINDER1; BY RECIPIENT_ID; RUN;
PROC SORT DATA=RECIP_BASE; BY RECIPIENT_ID; RUN;
```

```
DATA TRYMATCH2 REMAINDER2;
* IF IT MATCHES, 2ND DATASET WILL OVERWRITE ANY VARS WITH THE SAME NAME;
MERGE REMAINDER1(IN=MINE) RECIP_BASE(IN=ALL);
BY RECIPIENT_ID;
IF MINE AND ALL THEN OUTPUT TRYMATCH2;
ELSE IF MINE AND NOT ALL THEN OUTPUT REMAINDER2;
RUN;
```

Verify the Results

The four DATA statements resulted in four tentative matched datasets, named “Trymatch1”, “Trymatch2”, “Trymatch3”, and “Trymatch4”. These were stacked into one dataset and assigned a new variable, called “Match”, with their respective match number, such as “4” for “Trymatch4”. This would later assist in troubleshooting of the tentative matches as well as writing a short narrative for the end user of the report to explain how the observations were matched and the possible limitations thereof. An example would be an observation that was matched on an MA number derived by concatenating the “SSN” with a zero; if this match was later found to be not valid, one would know that the assumed MA number was incorrect and could research further. The DATA step that stacked the output files into one is shown below.

```
DATA TRYMATCH_ALL;
SET TRYMATCH1(IN=ONE) TRYMATCH2(IN=TWO) TRYMATCH3(IN=THREE)
TRYMATCH4(IN=FOUR);
IF ONE THEN MATCH=1;
ELSE IF TWO THEN MATCH=2;
ELSE IF THREE THEN MATCH=3;
ELSE IF FOUR THEN MATCH=4;
RUN;
```

Finding Those with Multiple Matches

Inevitably, there was one observation that found two matches. In theory this shouldn’t have happened, but this participant was listed under two different names for the same SSN in the new dataset. This could also happen if this person had two identifying numbers in the original dataset. Good data practices should always check for duplicates. I checked for duplicates manually by comparing the number of observations in the log and comparing to the number of original observations stored in the macro variable I created, called “nobs_original”. When I stacked the matches and came up with a number of observations one larger than the number stored in “nobs_original”, I knew I needed investigate further.

A FIRST statement within a DATA step was used to write an output dataset of duplicates. Most of the time, one would want to check for first observations on a personal identifying number. However, in this instance, in which two different client names were associated with the same SSN, it was a better choice for me to use FIRST with client name. The SAS code that found these duplicates is shown below:

```
PROC SORT DATA=TRYMATCH_ALL; BY CLTNAME; RUN;
DATA DUPLICATE;
SET TRYMATCH_ALL;
BY CLTNAME;
IF NOT FIRST.CLTNAME THEN OUTPUT;
RUN;
```

This “duplicates” file was examined and remedied by removing the appropriate row from the dataset of matched observations. Once the number of matched observations matched the initial unduplicated count that was put into the macro variable called “nobs_original”, it was time to verify the potential matches.

My preparation of the datasets allowed all of the matches to retain their original name and birth date from each dataset. Therefore, for each observation, there was a name and date of birth as held in the old dataset as well as that held in the new dataset. In theory, these should match and verify that the match was valid. If any of them did not agree, it may or may not invalidate the match. If any of the of information, last name, first name and birth date, did not match, I created three variables, named ln_match (last name match), fn_match (first name match) and bd_match (birth date match), to hold a binary value to indicate this. A value of 1 was assigned if the two variables matched between the two datasets; a value of 0 if they did not match.

The DATA step that began to verify the potential matches wrote three output files:

- "Verify1" held those observations that were a verified match between all three variables (first name, last name and date of birth)
- "NameGame" held those observations with a match between two of the three variables
- "CheckMe" held observations with a match between less than two of the three variables

The "NameGame" dataset will be checked with the SOUNDEX function, while the few in the "CheckMe" dataset will need to be manually verified. The code that broke the matches into these three datasets is shown below:

```
DATA VERIFY1(DROP=LN_MATCH FN_MATCH BD_MATCH SUM_MATCH)
CHECKME NAME_GAME;
SET TRYMATCH_ALL1;
*THE LAST NAME, FIRST NAME AND BIRTH DATE MATCH;
IF COMPRESS(LAST_NAME)=SCAN(CLTNAME,1) AND
   COMPRESS(FIRST_NAME)=SCAN(CLTNAME,2) AND
   DATEPART(BIRTH_DATE)=CLTBTHDT_HSRS
THEN OUTPUT VERIFY1;
*TELL ME WHAT DOESN'T MATCH;
ELSE DO;
IF COMPRESS(LAST_NAME) NE SCAN(CLTNAME,1)  THEN LN_MATCH=0; ELSE LN_MATCH=1;
IF COMPRESS(FIRST_NAME) NE SCAN(CLTNAME,2) THEN FN_MATCH=0; ELSE FN_MATCH=1;
IF DATEPART(BIRTH_DATE) NE CLTBTHDT_HSRS   THEN BD_MATCH=0; ELSE BD_MATCH=1;
SUM_MATCH=SUM(LN_MATCH,FN_MATCH,BD_MATCH);
*CHECK THESE BY HAND;
IF SUM_MATCH LT 2 THEN OUTPUT CHECKME;
*THESE ARE THE JENNYs AND JENNIFERs;
ELSE OUTPUT NAME_GAME;
END;
RUN;
```

Use of SOUNDEX to match "Jenny" to "Jennifer"

Those that had only a difference on first and/or last name, as determined by the binary variables created in the DATA step above, were further researched by the use of the SOUNDEX function. The SOUNDEX function returns a code for an input character string according to an algorithm that was originally developed by Margaret K. Odell and Robert C. Russel (SOUNDEX Function).

Table 1: Examples of SOUNDEX values for simple spelling variations of a first name

Input String	SOUNDEX
Jen	J5
Jenny	J5
Jenni	J5
Steven	S315
Stephan	S315
Bill	B4
Billy	B4

Table 1 demonstrates that two different spellings of a first name, such as "Jen" and "Jenny", have the same SOUNDEX value and could be verified by comparing not the name itself, but the SOUNDEX value. Most name verifications involved a variation on the spelling of only the first name, such as "Jen Smith" and "Jenny Smith". These could be verified as a match when the birth date matched and the SOUNDEX of both first and last names matched. The SAS code for checking the SOUNDEX of the first and last name is shown below. Also shown below is the next case that needed to be examined-when the SOUNDEX did not match. At this point, I needed to find the SOUNDEX value of the first name in the new dataset ("oracle_sound_first"), and the last name in the new dataset ("oracle_sound_last"), as well as the SOUNDEX value for the first and last name from the old dataset, which were "hsrs_sound_first" and "hsrs_sound_last" respectively. These values will be compared and sorted in the next DATA step.

```
*DEAL WITH THE "JENs" AND "JENNYs";
DATA OK NAME_GAME1;
SET NAME_GAME;
WHERE BD_MATCH NE 0;
IF SOUNDEX(FIRST_NAME)=SOUNDEX(SCAN(CLTNAME,2)) AND
SOUNDEX(LAST_NAME)=SOUNDEX(SCAN(CLTNAME,1)) THEN OUTPUT OK;
```

```
ELSE DO; *EXAMINE THESE IN THE NEXT DATA STEP;
      ORACLE_SOUND_FIRST=SOUNDEX(FIRST_NAME);
      ORACLE_SOUND_LAST=SOUNDEX(LAST_NAME);
      HSRS_SOUND_FIRST=SOUNDEX(SCAN(CLTNAME,2));
      HSRS_SOUND_LAST=SOUNDEX(SCAN(CLTNAME,1));
      OUTPUT NAME_GAME1;
END;
RUN;
```

After the easy matches were verified and removed to a verified dataset, the next task involved matching "Jenny Smith" to "Jennifer Smith". The SOUNDEX value of Jenny is not the same as Jennifer, but the first two characters of the SOUNDEX value are the same. Examples of SOUNDEX values for larger spelling variations of the first name are shown in Table 2 below.

Table 2: Examples of SOUNDEX values for larger variations of a first name

Input String	SOUNDEX
Jenny	J5
Jennifer	J516
Steve	S31
Steven	S315
William	W45
Bill	B4
Billy	B4
Brian	B65
Richard	R263
Dick	D2
Dicky	D2
Dan	D5
Thomas	T52
Tom	T5
Tommy	T5

Table 2 demonstrates that for most spelling variations, the first two characters of the SOUNDEX value are the same. For some name variations, such as "William" and "Bill", only the second character will be the same. Further, one can see that "Bill" and "Billy" have the same SOUNDEX value, but it is different that the SOUNDEX value for a similar name starting with the same letter, such as "Brian". Variations that involved looking at only the second character of the SOUNDEX value was not encountered in this project but is noted for reference for the reader.

The SUBSTR function was used to examine only the first two characters of the resultant SOUNDEX value on the first name. If the first two characters of the SOUNDEX matched, and all the characters of the SOUNDEX of the last name matched, the observation was verified as a match. The SAS code below demonstrates this.

```
DATA NAME_GAME2 OK1;
SET NAME_GAME1;
IF ORACLE_SOUND_LAST=HSRS_SOUND_LAST AND
      SUBSTR(ORACLE_SOUND_FIRST,1,2)=SUBSTR(HSRS_SOUND_FIRST,1,2)
      THEN OUTPUT OK1;
ELSE OUTPUT NAME_GAME2;
RUN;
```

Inevitably, there were a few matches that needed to be manually examined and researched to verify the potential match. However, use of the SOUNDEX function kept these observations to a minimum.

Once all potential matches were verified, the newer client identifier "Recipient ID" was obtained for all. This "Recipient ID" of the history file of the newer dataset could then be linked to yet another dataset containing the vital pieces of information originally sought. Finally, the report could be produced, which matched data elements from the old system, such as name and program end date, to data elements in the new system, such as enrollment date in the new program.

Conclusions

Data systems change over time creating variables in new formats. Many times, not all data elements are present in both the old and new data systems and the data must be merged to obtain needed information. Some times, a

variable can hold more than one piece of information, and assumptions can be used to employ this variable as a matching field.

A series of DATA steps with a MERGE statement can be used as a “sifting and winnowing” way of matching pieces of information together. These potential matches can be verified by checking key pieces of information from the potential matches, such as the original first name, original last name and original birth date from each dataset.

Variations on the first name can be tested by using the SOUNDEX function instead of manually examining pairs of matches. If the SOUNDEX value for two character strings, such as first names, are the same or similar, it is a good probability that the names are the same and the match is valid.

This method of using a series of data step merges to cross-walk one dataset to another and verify the matches with SOUNDEX has applications to many other industries. Any system change, resulting in a change to user ID, client ID or vendor ID are possible applications.

References

SOUNDEX Function. 2004. SAS Help and Documentation. SAS Institute Inc., Cary, NC, USA.

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Misty Johnson
State of Wisconsin, Department of Health Services
1 W Wilson St, Rm 750
Madison, WI 53701-7851
Phone: 608-267-9561
Fax: 608-264-9874
E-mail: misty.johnson@wi.gov