

Potential Duplicate HIV/AIDS Patient Names Across States. A CDC Puzzle

Bikila bi Gwet (Kila Gwet.) Bikila Analytics Inc, Canton, GA

Abstract

New HIV/AIDS patients "declare" their status when they test positive. One patient may do so at different institutions within one state or across states. The Center for Disease Control (CDC) centralizes new HIV/AIDS patient lists and ultimately creates a single list for the entire country. It is seemingly easy to remove duplicates using the SORT procedure with the NODUPKEY option or the SOUNDS LIKE operator of the SQL procedure when joining two tables. There are three problems, however: errors in name spelling are frequent; there are 1,225 ways of selecting 2 states out of 50; a pair-wise approach would not remove duplicates within each list. It makes sense to concatenate all new patient lists into a single file and then attempt to uncover duplicates. This paper № PO18 suggests a simple top-down duplicate name identification process from a single list using mainly lookup arrays and the DATA step. It also introduces the reader to effective process automation.

Introduction and Problem Statement

Compiled say monthly at a central location, Table I.1 represents a sample list of new HIV/AIDS patients from 50 states. Because of privacy and data issues, there is no strong matching key such as the social security number that could be used to remove duplicates using the SORT procedure with the NODUPKEY option. As required by law in at least certain cases, residents of the United States are regularly given the option to opt out of solicitation campaigns. To comply with the demands of regulatory agencies and for business reasons, credit bureaus and other companies keep files with do-not-solicit prospects. A centralized list of this type is cumulative and expands with the addition of new names. One well designed SQL join with the SOUNDS LIKE operator can effectively screen out of a new marketing campaign file most do-not-solicit prospects. A combination of name and address provides a near exact match in this case.

Table I.1. New Nationwide HIV/AIDS Patient List								
Obs	Id Num	L Name	F Name	Gender	Birth Date	City	State	H Phone
1	4302	ADAMS	GERALD	M	8/9/1952	STAMFORD	CT	203/781-1255
2	4983	JONSON	ANTHONY	M	8/16/1962	ANY CITY	NJ	201/586-0808
3	7889	ALIBRANDI	MARIA	F	9/23/1952	BRIDGEPORT	CT	203/675-7715
4	7636	ALHERTANI	ABDULLAH	M	9/23/1951	NEW YORK	NY	212/586-0808
5	3491	JOHNSON	ANTHONY	M	8/16/1962		GA	
6	1554	SANDERSON	NATHAN	M	2/19/1963	BRIDGEPORT	CT	203/675-1715
7	1580	JOHNSON	L	F	5/17/1976		MD	
8	6631	ALVAREZ	MERCEDES	F	9/23/1951	NEW YORK	NY	718/383-1549
9	9575	JOHNSEN	JOHN	M	9/23/1952		NY	212/781-1255
10	6016	BAREFOOT	JOSEPH	M	8/25/1952	PRINCETON	NJ	201/812-5665
11	4073	JOHNSON	TONY	F	8/16/1962		MT	
12	2053	BAUCOM	WALTER	M	9/23/1952	NEW YORK	NY	212/586-8060
13	5969	BANADYGA	JUSTIN	M	12/4/1969	STAMFORD	CT	203/781-1777
14	8423	JOHNSEN	JACK	M	9/23/1952	NEWARK	NJ	201/383-1549
15	2355	BLALOCK	RALPH	M	8/24/1979	NEW YORK	NY	718/384-2849
16	8811	BALLETTI	MARIE	F	5/23/1978	NEW YORK	NY	718/384-8816
17	2790	JONSON	TONY	M	8/16/1962		SC	
18	3237	SANDERS	RAYMOND	M	10/2/1954	NEW YORK	NY	212/588-6615
19	2891	JOHNSON	LESLIE	F	5/17/1976	LINCOLN	NJ	201/732-8787
20	3771	SANDERSON	EDITH	F	6/8/1956	STAMFORD	CT	203/781-1333
21	5042	ALVAREZ	CARLOS	M	9/23/1952	PATERSON	NJ	201/732-8787
22	7837	SANDERSON	E	F	6/8/1956		CO	
23	3146	JOHNSON	LESLEY	F	5/17/1976		DC	

In the absence of a central repository, the above process cannot remove duplicate HIV/AIDS patients from new lists. The possibility of having up to 50 lists each time turns a complex issue into a puzzle. This example uses a combination of identical last name sound, gender and birth date as criterion for a near exact match.

Four out of twenty-three new patients in the sample most likely appear at least twice as suggests the final report in Table I.2. This report is convenient in that for each ID number (variable *IdNum*) the observation holds the original information provided by the state. Next, the report lists all duplicate ID numbers (variable *DupIdNum*), the corresponding data of which are part of another record.

Table I.2. Final Report							
Obs	Id Num	Dup Id Num	L Name	F Name	Gender	Birth Date	State
1	1580	2891 3146	JOHNSON	L	F	5/17/1976	MD
2	2891	1580 3146	JOHNSON	LESLIE	F	5/17/1976	NJ
3	3146	1580 2891	JOHNSON	LESLY	F	5/17/1976	DC
4	8423	9575	JOHNSON	JACK	M	9/23/1952	NJ
5	9575	8423	JOHNSON	JOHN	M	9/23/1952	NY
6	2790	3491 4983	JONSON	TONY	M	8/16/1962	SC
7	3491	2790 4983	JOHNSON	ANTHONY	M	8/16/1962	GA
8	4983	2790 3491	JONSON	ANTHONY	M	8/16/1962	NJ
9	3771	7837	SANDERSON	EDITH	F	6/8/1956	CT
10	7837	3771	SANDERSON	E	F	6/8/1956	CO

1. Reading the Data

```
title; options nodate nonumber ps=32700 formdlim=' '; *set up the SAS session;
libname cdc 'C:\Bikila Analytics\2006 SESUG\Poster'; *assign libref;
```

```
*Routine 1.1: Read sample data;
data cdc.SampleData0;
  label IdNum='Id Num' LName='L Name' FName='F Name'
        Gender='Gender' BirthDate='Birth Date'
        City='City' State='State' HPhone='H Phone';
  infile cards dsd missover firstobs=2;
  input LName:$30. FName:$30. Gender:$1. BirthDate:mmddyy.
        City:$30. State:$2. HPhone:$15.;
  IdNum=put(round(ranuni(129834)*9999),z4.);
  format BirthDate mmddyy10.;
  cards;
```

```
LName,FName,Gender,BirthDate,City,State,HPhone
ADAMS,GERALD,M,08/09/1952,STAMFORD,CT,203/781-1255
JONSON,ANTHONY,M,08/16/1962,ANY CITY,NJ,201/586-0808
ALIBRANDI,MARIA,F,09/23/1952,BRIDGEPORT,CT,203/675-7715
ALHERTANI,ABDULLAH,M,09/23/1951,NEW YORK,NY,212/586-0808
JOHNSON,ANTHONY,M,08/16/1962,,GA,
SANDERSON,NATHAN,M,02/19/1963,BRIDGEPORT,CT,203/675-1715
JOHNSON,L,F,05/17/1976,,MD,
ALVAREZ,MERCEDES,F,09/23/1951,NEW YORK,NY,718/383-1549
JOHNSON,JOHN,M,09/23/1952,,NY,212/781-1255
BAREFOOT,JOSEPH,M,08/25/1952,PRINCETON,NJ,201/812-5665
JOHNSON,TONY,F,08/16/1962,,MT,
BAUCOM,WALTER,M,09/23/1952,NEW YORK,NY,212/586-8060
BANADYGA,JUSTIN,M,12/04/1969,STAMFORD,CT,203/781-1777
JOHNSON,JACK,M,09/23/1952,NEWARK,NJ,201/383-1549
BLALOCK,RALPH,M,08/24/1979,NEW YORK,NY,718/384-2849
BALLETTI,MARIE,F,05/23/1978,NEW YORK,NY,718/384-8816
JONSON,TONY,M,08/16/1962,,SC,
SANDERS,RAYMOND,M,10/02/1954,NEW YORK,NY,212/588-6615
JOHNSON,LESLIE,F,05/17/1976,LINCOLN,NJ,201/732-8787
SANDERSON,EDITH,F,06/08/1956,STAMFORD,CT,203/781-1333
ALVAREZ,CARLOS,M,09/23/1952,PATERSON,NJ,201/732-8787
SANDERSON,E,F,06/08/1956,,CO,
JOHNSON,LESLY,F,05/17/1976,,DC,
;
```

```
*Routine 1.1 (continued);
proc print data=cdc.SampleData0 label;
  title 'Table I.1. New Nationwide
        HIV/AIDS Patient List';
run;
title;
```

SAS® Programmer's Note

Routine 1.1 reads sample data in-stream; it lists data values after the CARDS statement. DATALINES is the modern but longer version of CARDS. This is also the second keyword of the INFILE statement generally used when reading data from a text or some other external file. This statement appears in-stream for the sole purpose of using some of its powerful options, namely DSD, MISSEVER and FIRSTOBS=.

By default (if the DLM= option does not suggest otherwise), the DSD option assumes comma to be the delimiter. It enables SAS language to adequately interpret missing values at the beginning and in the middle of records (those beginning with a comma or having two or more consecutive commas.) The MISSEVER option handles missing values at the end of the records. Option FIRSTOBS= designates the number of the first record to consider.

The data herein analyzed are completely fictive; any resemblance with the information regarding real people is pure coincidence.

2. Duplicate Name Identification Process

Output 2.1 evidences the presence of potential duplicate last names through identical values of variable *SoundCode*. Bolded in the output is sound code value J525, which suggests up to eight potential duplicate names for a single patient. From the original sample data, Section 2.1 presents a process that isolates these and other similar observations effectively.

```
*Routine 2.1: Create last-name sound codes;
data cdc.SampleData;
  label IdNum='Id Num' SoundCode='Sound Code';
  length SoundCode $ 30;
  set cdc.SampleData0;
  SoundCode=soundex(LName);
run;

*Routine 2.1 (continued);
proc print data=cdc.SampleData label;
  title 'Output 2.1. Data Set SampleData';
run;
title;
```

2.1. Duplicate List by Last-Name Sound Code

The list in Output 2.1 still has all distinct and all potential duplicate patient names. This section aims at reducing it to only potential duplicate names by last-name sound code. With respect to variable *SoundCode*, BY-group processing in Routine 2.2 produces two automatic variables *first.SoundCode* and *last.SoundCode* that may not be part of the output data set. The code consequently assigns their values to variables *First* and *Last* respectively. Referred to as Booleans, these variables may only take values 1 or 0. The first occurrence of a particular value sets *first.SoundCode* = 1; this variable equals 0 for all other observations with the same value for variable *SoundCode*. The last occurrence of a particular value sets *last.SoundCode* = 1. Automatic variable *last.SoundCode* equals 0 for all other observations with the same value for variable *SoundCode*.

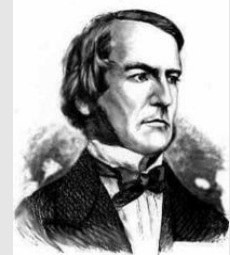
Output 2.1. Data Set SampleData

Obs	Id Num	Sound Code	L Name	F Name	Gender	Birth Date	City	State	H Phone
1	4302	A352	ADAMS	GERALD	M	08/09/1952	STAMFORD	CT	203/781-1255
2	4983	J525	JONSON	ANTHONY	M	08/16/1962	ANY CITY	NJ	201/586-0808
3	7889	A41653	ALIBRANDI	MARIA	F	09/23/1952	BRIDGEPORT	CT	203/675-7715
4	7636	A4635	ALHERTANI	ABDULLAH	M	09/23/1951	NEW YORK	NY	212/586-0808
5	3491	J525	JOHNSON	ANTHONY	M	08/16/1962		GA	
6	1554	S53625	SANDERSON	NATHAN	M	02/19/1963	BRIDGEPORT	CT	203/675-1715
7	1580	J525	JOHNSON	L	F	05/17/1976		MD	
8	6631	A4162	ALVAREZ	MERCEDES	F	09/23/1951	NEW YORK	NY	718/383-1549
9	9575	J525	JOHNSON	JOHN	M	09/23/1952		NY	212/781-1255
10	6016	B613	BAREFOOT	JOSEPH	M	08/25/1952	PRINCETON	NJ	201/812-5665
11	4073	J525	JOHNSON	TONY	F	08/16/1962		MT	
12	2053	B25	BAUCOM	WALTER	M	09/23/1952	NEW YORK	NY	212/586-8060
13	5969	B532	BANADYGA	JUSTIN	M	12/04/1969	STAMFORD	CT	203/781-1777
14	8423	J525	JOHNSON	JACK	M	09/23/1952	NEWARK	NJ	201/383-1549
15	2355	B442	BLALOCK	RALPH	M	08/24/1979	NEW YORK	NY	718/384-2849
16	8811	B43	BALLETTI	MARIE	F	05/23/1978	NEW YORK	NY	718/384-8816
17	2790	J525	JONSON	TONY	M	08/16/1962		SC	
18	3237	S5362	SANDERS	RAYMOND	M	10/02/1954	NEW YORK	NY	212/588-6615
19	2891	J525	JOHNSON	LESLIE	F	05/17/1976	LINCOLN	NJ	201/732-8787
20	3771	S53625	SANDERSON	EDITH	F	06/08/1956	STAMFORD	CT	203/781-1333
21	5042	A4162	ALVAREZ	CARLOS	M	09/23/1952	PATERSON	NJ	201/732-8787
22	7837	S53625	SANDERSON	E	F	06/08/1956		CO	
23	3146	J525	JOHNSON	LESLY	F	05/17/1976		DC	

From Output 2.2 it is obvious that both variables *First* and *Last* equal 1 only for observations with distinct last name sounds. Without George Boole's overflowing genius, it would have been a real headache trying to isolate duplicate names directly through variable *SoundCode* because different sounds produce (unlimited) different codes. With Boolean algebra, it becomes as simple as 1 or 0, yes or no, black or white, and so on, no matter the number of distinct sounds. Routine 2.3 removes all observations with distinct last-name sounds.

```
*Routine 2.2. Identify distinct last-name sounds;
proc sort data=cdc.SampleData;
  by SoundCode;
run;
data FirstLast(keep=First Last SoundCode LName);
  length First Last 8;
  set cdc.SampleData;
  by SoundCode;
  First=first.SoundCode; Last=last.SoundCode;
run;
proc print data=FirstLast label;
  title 'Output 2.2. Data Set FirstLast';
run; title;
```

If you used a computer today, then you owe that to George Boole, mostly self-taught creator of binary mathematics, for computers, the Internet and so many other modern technologies can only use binary arithmetics.



Bikila bi Gwet
Canton, GA, April 22, 2006

[George Boole](#)
(1815-1864)

SAS® Programmer's Note

Effective use of Boolean (binary) variables and expressions creates simple and powerful SAS code. Routine 2.3 removes distinct sound codes using statement "if not(first.SoundCode & last.SoundCode);". It is the same as:

```
if not(first.SoundCode=1 & last.SoundCode=1);.
```

Manipulating logical expressions is easily confusing however, and the corresponding errors produce no messages in the log window. Even when the user is familiar with elementary mathematical logic, a good rule of thumb comes in handy: think through and keep things simple and very precise. We want to **get rid of** all observations **where First = 1 and Last = 1** (Output 2.2); instead "**where First and Last = 1**" is good English but imprecise at best, and no code at all! Try the following **wrong** statement and see the result: `if not first.SoundCode & not last.SoundCode;`

```
*Routine 2.3. Isolate & count sound groups;
data cdc.DuplicateSoundCodes;
  set cdc.SampleData(drop=City HPhone);
  by SoundCode;
  if not(first.SoundCode & last.SoundCode);
  if first.SoundCode then Group+1; else;
run;
```

Output 2.2. Data Set FirstLast				
Obs	First	Last	Sound Code	L Name
1	1	1	A352	ADAMS
2	1	0	A4162	ALVAREZ
3	0	1	A4162	ALVAREZ
4	1	1	A41653	ALIBRANDI
5	1	1	A4635	ALHERTANI
6	1	1	B25	BAUCOM
7	1	1	B43	BALLETTI
8	1	1	B442	BLALOCK
9	1	1	B532	BANADYGA
10	1	1	B613	BAREFOOT
11	1	0	J525	JOHNSON
12	0	0	J525	JONSON
13	0	0	J525	JOHNSON
14	0	0	J525	JOHNSON
15	0	0	J525	JOHNSON
16	0	0	J525	JOHNSON
17	0	0	J525	JONSON
18	0	0	J525	JOHNSEN
19	0	1	J525	JOHNSEN
20	1	1	S5362	SANDERS
21	1	0	S53625	SANDERSON
22	0	0	S53625	SANDERSON
23	0	1	S53625	SANDERSON

2.2. Duplicate List by Sound Group, Gender and Birth Date

Because all three variables *SoundGroup*, *Gender* and *BirthDate* concurrently define the final criterion for duplicate names, observations in the data set must be in that order before numbering sound group elements through variable *Dim* (for dimension. See Routine 2.4.) The dimension of potential duplicates will prove critical in the final name identification process.

```
*Routine 2.4. Number sound group elements;
proc sort data=cdc.DuplicateSoundCodes;
  by SoundCode Gender BirthDate;
run;

data cdc.DuplicateSoundCodes;
  label IdNum='Id Num' SoundCode='Sound Code'
        Group='Group' Dim='Dim';
  set cdc.DuplicateSoundCodes;
  by SoundCode;
  if first.SoundCode then Dim=0; else; Dim+1;
run;
```

```
*Routine 2.4 (continued);
proc print data=cdc.DuplicateSoundCodes label;
  title 'Output 2.3. Data Set DuplicateSoundCodes';
run;
title;
```

A common **mistake** consists of creating secondary by-group variable ranks (herein called dimensions) by using an exclusive conditional process (with no comma after "else"):

```
if first.SoundCode then Dim=0; else Dim+1;
```

Output 2.3. Data Set DuplicateSoundCodes

Obs	Id Num	Sound Code	Group	Dim	L Name	F Name	Gender	Birth Date	State
1	6631	A4162	1	1	ALVAREZ	MERCEDES	F	09/23/1951	NY
2	5042	A4162	1	2	ALVAREZ	CARLOS	M	09/23/1952	NJ
3	4073	J525	2	1	JOHNSON	TONY	F	08/16/1962	MT
4	1580	J525	2	2	JOHNSON	L	F	05/17/1976	MD
5	2891	J525	2	3	JOHNSON	LESLIE	F	05/17/1976	NJ
6	3146	J525	2	4	JOHNSON	LESLY	F	05/17/1976	DC
7	8423	J525	2	5	JOHNSEN	JACK	M	09/23/1952	NJ
8	9575	J525	2	6	JOHNSEN	JOHN	M	09/23/1952	NY
9	2790	J525	2	7	JONSON	TONY	M	08/16/1962	SC
10	3491	J525	2	8	JOHNSON	ANTHONY	M	08/16/1962	GA
11	4983	J525	2	9	JONSON	ANTHONY	M	08/16/1962	NJ
12	3771	S53625	3	1	SANDERSON	EDITH	F	06/08/1956	CT
13	7837	S53625	3	2	SANDERSON	E	F	06/08/1956	CO
14	1554	S53625	3	3	SANDERSON	NATHAN	M	02/19/1963	CT

Section 2.2 merely developed a list that includes all potential duplicate HIV/AIDS patients by last name sound, gender and birth date. It did not clearly identify them. Only the simplicity of the data and the smallness of the sample size enable to "see" these duplicates. This luxury would vanish in real-life situations or even by increasing the size of the above sample just a few folds. However, the information contained in Output 2.3 is critical. The file encompasses a numbered list of duplicate name sound groups (1 – 3 in this example) and numbers the elements of each group (1 – 2, 1 – 9 and 1 – 3 in the above sample.)

2.3. Duplicate Identification: A Forward/Backward Look-Up Approach

The last names in each of three groups of potential duplicates have the same sounds and might correspond to the same individuals. In each group, the simultaneous identity of gender and birth date will meet the criterion for potential duplicate patients. All such individuals must be set aside and reported to their respective states for further investigation. Routine 2.5 is the cornerstone of the entire process. It produces Output 2.4.

```
*Routine 2.5. Tag matching records with Boolean variable Match;
1.  /*let Group=1; %let star=; %debug;
2.    %let Group=2; %let star=; %debug;
3.    %let Group=3; %let star=; %debug;*/
4.  %let star=; %let Group=;
5.  data cdc.DuplicateSexBirth
6.    (keep=Match IdNum DupIdNum Group Dim LName FName Gender BirthDate State);
7.    label Match='Match' IdNum='Id Num' DupIdNum='Dup Id Num';
8.    *look up arrays (retained across DATA step iterations);
9.    array sex{3,9} $ 1 _temporary_; *Gender lookup array (3 groups & up to 9 dimensions);
10.   array dob{3,9} _temporary_; *BirthDate lookup array (3 groups & up to 9 dimensions);
11.   array yes{3,9} _temporary_; *Match lookup array by Gender & BirthDate within each of 3 groups;
12.   array idn{3,9} $ 4 _temporary_; *individual IdNum lookup array;
13.   array ids{3,9} $ 32 _temporary_; *duplicate IdNums/assumes up to 8 IdNums with 4 bytes each;
14.   if _n_=1 then do until(last); *identify matching Gender & BirthDate in each Group;
15.     set cdc.DuplicateSoundCodes end=last;
16.     by SoundCode Dim;
17.     &star.where Group=&Group; *debug;
18.     *load lookup arrays;
19.     sex{Group,Dim}=Gender; dob{Group,Dim}=BirthDate; idn{Group,Dim}=IdNum;
20.     &star.put 'DEBUG NOTE 1: ' Group= Dim= sex{Group,Dim}= dob{Group,Dim}=;
21.     if last.SoundCode then do; *tag duplicate records by Gender & BirthDate;
22.       do t=1 to Dim-1; *treatment elements (Dim>1 for all obs);
23.         do c=t+1 to Dim; *control elements;
24.           ok={sex{Group,t}=sex{Group,c} & dob{Group,t}=dob{Group,c}}; *1=match, 0=no match;
25.           yes{Group,t}=max(yes{Group,t},ok); *MAX function ensures previous match holds;
26.           *mandatory pairwise match provided last element matched (ok=1);
27.           if yes{Group,t} then yes{Group,c}=ok*yes{Group,t}; *ok=0 seals right-hand side;
28.           else yes{Group,c}=yes{Group,t}; *mandatory pair-wise non-match;
29.           &star.put 'DEBUG NOTE 2: ' t= c= sex{Group,t}= sex{Group,c}= dob{Group,t}=
30.             dob{Group,c}= yes{Group,t}= ok= yes{Group,c}=;
31.           if ok then do; *load variable DupIdNum (duplicate IdNum list);
32.             ids{Group,t}=trim(ids{Group,t})||' '||trim(idn{Group,c}); *treatment element;
33.             ids{Group,c}=trim(ids{Group,c})||' '||trim(idn{Group,t}); *goes both ways;
34.           end; *if ok;
35.         end; *do c;
36.       end; *do t;
37.     end; *if last.SoundCode; else;
38.   end; *if _n_;
39.   set cdc.DuplicateSoundCodes; *actual processing begins;
40.   &star.where Group=&Group; *debug;
41.   Match=yes{Group,Dim}; DupIdNum=ids{Group,Dim}; *assign/unload values collected during lookup;
42. run;
proc print data=cdc.DuplicateSexBirth label;
  title 'Output 2.4. Data Set cdc.DuplicateSexBirth';
run;
title;
```

SAS® Analyst's Notes (Routine 2.5)

The apparent complexity of the code in Routine 2.5 – as is often the case in data analysis – does not come from the techniques used: SAS Institute's base SAS programming certification adequately addresses them all. The difficulty resides in the need for the analyst to **remember** a large number of details and techniques and to **apply them where and when** the business case demands, **in the right sequence**. If they exist, the author is not yet aware of analytics training programs that specifically help develop this skill set, which is totally different from mere computer programming.

Non SAS macro programmers must be aware of the fact that lines 1 – 4 and symbols &star. (lines 17, 20, 29 and 40) and &Group (lines 17 and 40) assign values to macro variables or call the latter. The code works perfectly as is and after removing lines 1 through 4, 17, 20, 29 and 40 altogether. To process only the second duplicate group (Group = 2), comment out lines 1, 3 and 4 by first removing symbols /* (from line 1) and */ (from line 3) and putting symbols /* at the beginning of each of lines 1, 3 and 4, and */ at the end of each of the same lines.

Lines 9, 10 and 12 define lookup arrays sex{3,9}, dob{3,9} and idn{3,9} destined to load and hold the values of variables *Gender*, *BirthDate* and *IdNum* as the pointer goes from top to bottom along source file *DuplicateSoundCodes*. Lines 11 and 13 define lookup arrays yes{3,9} and ids{3,9} that will save the values of future output variables *Match* and *DuplIdNum*. All these arrays are two-dimensional. The array dimensions 3 and 9 correspond to the largest number of distinct values of variables *Group* and *Dim*. Lookup arrays end with keyword *_temporary_*, which means that they do not directly create variables in the output data set. SAS software retains the values of lookup arrays across DATA step iterations unlike those of other assignment variables (created by the programmer by putting them in front of an equal sign.) These variables are set to missing each time the DATA step iterates. Variables (from left to right) define regular arrays; data values along observations (from top to bottom) define lookup arrays.

During the first DATA step iteration (when automatic variable *_n_* = 1), the code from line 15 to line 37 executes for every observation in the source file, loading needed variable values and performing necessary calculations. Line 19 loads the three arrays with values of the corresponding variables from the source file on a magnetic tape (hard drive) into the random access memory (RAM) of the computer. Two nested DO loops (**lines 22 – 36**) execute only after the system has finished loading the information concerning an entire distinct *SoundCode* represented by a group number, say Group = 2. There are two tasks at hand for each of three groups:

1. Identify and tag (using Booleans) each and every ID number whose gender **and** birth date match those of any other ID number in the same group.
2. In case of a match, save duplicate ID numbers for reporting.

Consider the third sound code (**Group = 3**) in source file *DuplicateSoundCodes* (Output 2.3.) In the outer loop, the index *t* (to designate the treatment ID number) goes from 1 to 3. The inner loop index *c* (to designate control ID numbers) goes from 2 to 3. For each value of outer index *t*, the inner index *c* goes through all its values (loops.)

Output 2.3.1. Data Set DuplicateSoundCodes (Group=3)					
Obs	Id Num	Dim	L Name	Gender	Birth Date
12	3771	1	SANDERSON	F	6/8/1956
13	7837	2	SANDERSON	F	6/8/1956
14	1554	3	SANDERSON	M	2/19/1963

When t = 1 (Dim = 1, Gender = F and Birth Date = 06/08/1956) and **c = 2** (Dim = 2, Gender = F and Birth Date = 06/08/1956) there is a gender and birth day match. In this case, Boolean expressions `sex{Group,t}=sex{Group,c}` and `dob{Group,t}=dob{Group,c}` are both true. Their conjunction is equally true and therefore produces another Boolean expression with value 1, which is temporarily assigned to variable *ok* as shown on line 24 of Routine 2.5. At this point, this tag (*ok* = 1) may be assigned to the treatment ID number 3771 and saved in the corresponding array as: `yes{Group,t}=ok;`, in fact `yes{3,1}=1;` (Assignment 1.) Because matching occurs in pairs, the control ID number 7837 earns the same tag, which must be saved in the corresponding array as: `yes{Group,c}=yes{Group,t};`, making `yes{3,2}=1` (Assignment 2.)

```
24. | ok=(sex{Group,t}=sex{Group,c} & dob{Group,t}=dob{Group,c}); *1=match, 0=no match;
25. | yes{Group,t}=max(yes{Group,t},ok); *MAX function ensures previous match holds;
26. | *mandatory pairwise match provided last element matched (ok=1);
27. | if yes{Group,t} then yes{Group,c}=ok*yes{Group,t}; *ok=0 seals right-hand side;
28. | else yes{Group,c}=yes{Group,t}; *mandatory pair-wise non-match;
```

When $c = 3$ (Dim = 3, Gender = M and Birth Date = 2/19/1963), there is no match and $ok = 0$ (line 24.) Assignment 1 above now turns array $yes\{Group, t\}$ to 0 although it was a match to a previous ID number. The use of the MAX function forces this array to keep its previous positive value as line 25 shows. Now that array $yes\{Group, t\}$ is kept equal to 1, Assignment 2 above would have awarded 1 to the undeserving ID number 1554 through array $yes\{Group, c\}$. Line 27 avoids that by multiplying $yes\{Group, t\}$ in the right-hand side of the equality by ok , where $ok = 0$.

When $t = 2$ (Dim = 2, Gender = F and Birth Date = 06/08/1956), we know that ID number 7837 is already a match to ID number 3771 through Assignment 2 above, now $yes\{Group, t\} = yes\{3, 2\} = 1$. The only possibility for the inner loop is $c = 3$, which is no match.

When there is a match ($ok = 1$), lines 32 & 33 concatenate duplicate ID numbers in array ids for the treatment and control ID numbers. The entire process continues for groups of any size until all lookup arrays load appropriately and the last observation of the source data set is processed (still during the first iteration of the DATA step.) Then the SET statement in line 39 reads each observation to get variables' values already existing in the source file and line 41 assigns array values $yes\{Group, Dim\}$ and $ids\{Group, Dim\}$ to output variables *Match* and *DupIdNum* respectively. Output 2.4 shows the resulting data set.

```

31. |         if ok then do; *load variable DupIdNum (duplicate IdNum list);
32. |             ids{Group,t}=trim(ids{Group,t})||' '||trim(idn{Group,c}); *treatment element;
33. |             ids{Group,c}=trim(ids{Group,c})||' '||trim(idn{Group,t}); *control element;
34. |         end; *if ok;
    |         . . . . .
39. |     set cdc.DuplicateSoundCodes; *actual processing begins;
40. |     &star.where Group=&Group; *debug;
41. |     Match=yes{Group,Dim}; DupIdNum=ids{Group,Dim}; *assign/unload values collected during lookup;

```

Output 2.4. Data Set cdc.DuplicateSexBirth										
Obs	Match	Id Num	Dup Id Num	Group	Dim	L Name	F Name	Gender	Birth Date	State
1	0	6631		1	1	ALVAREZ	MERCEDES	F	9/23/1951	NY
2	0	5042		1	2	ALVAREZ	CARLOS	M	9/23/1952	NJ
3	0	4073		2	1	JOHNSON	TONY	F	8/16/1962	MT
4	1	1580	2891 3146	2	2	JOHNSON	L	F	5/17/1976	MD
5	1	2891	1580 3146	2	3	JOHNSON	LESLIE	F	5/17/1976	NJ
6	1	3146	1580 2891	2	4	JOHNSON	LESLY	F	5/17/1976	DC
7	1	8423	9575	2	5	JOHNSON	JACK	M	9/23/1952	NJ
8	1	9575	8423	2	6	JOHNSON	JOHN	M	9/23/1952	NY
9	1	2790	3491 4983	2	7	JONSON	TONY	M	8/16/1962	SC
10	1	3491	2790 4983	2	8	JOHNSON	ANTHONY	M	8/16/1962	GA
11	1	4983	2790 3491	2	9	JONSON	ANTHONY	M	8/16/1962	NJ
12	1	3771	7837	3	1	SANDERSON	EDITH	F	6/8/1956	CT
13	1	7837	3771	3	2	SANDERSON	E	F	6/8/1956	CO
14	0	1554		3	3	SANDERSON	NATHAN	M	2/19/1963	CT

2.4. Final Report

Retaining non-missing duplicate ID numbers subsets the above data set to create the final report presented in Table I.2 in the introduction. The code in Routine 2.6 uses the sub-setting IF statement with variable *DupIdNum*. It could also have used variable *Match* as:

```
if Match=1;.
```

```

*Routine 2.6. Final report;
data cdc.PotentialDuplicateNames;
    set cdc.DuplicateSexBirth(drop=Match Group Dim);
    if DupIdNum^=' ';
run;

```

3. Elements of Process Automation

Even well established processes tend to be tedious and time consuming in real-life business settings. This is due to the fact that most projects are either ad-hoc or periodic. Analysts need to reacquaint themselves with processes they face just a few times annually. Moreover, validation of complex tasks slows down work further. Manual processing of data is particularly prone to errors. Finally, typical business units update source files at least once monthly. This implies hardcode inadequacy. Effective automation processes make code flexible.

```
9.      array sex{3,9} $ 1 _temporary_; *Gender lookup array (3 groups & up to 9 dimensions);
10.     array dob{3,9} _temporary_; *BirthDate lookup array (3 groups & up to 9 dimensions);
11.     array yes{3,9} _temporary_; *Match lookup array by Gender & BirthDate within each of 3 groups;
12.     array idn{3,9} $ 4 _temporary_; *individual IdNum lookup array;
13.     array ids{3,9} $ 32 _temporary_; *duplicate IdNums/assumes up to 8 IdNums with 4 bytes each;
```

The identification and reporting on potential duplicate HIV/AIDS patient names in the current example should require prior knowledge of only a few things. They are the name of the source file and the names of variables representing last name, gender, birth date, ID number and state. Using data set *DuplicateSoundCodes* as source file, this section suggests ways to automate Routine 2.5 that presents the most hardcode challenges. The macro must automatically find appropriate numeric values for the array definitions in lines 9 – 13.

Programmers communicate with computers through code, and code rarely comprises data values. Rather, it uses variable and data attributes that are part of the descriptor portion, not the data portion of SAS® files. Efficient and flexible programming generates just-in-time code before compilation. By creating attribute sets, SQL dictionary tables make attributes available to the DATA step, the most powerful unit of SAS software.

Output 3.1. Data Set Columns					
Num	Name	Type	Len	Format	Label
1	IdNum	CHAR	4		Id Num
2	SoundCode	CHAR	30		Sound Code
3	Group	NUM	8		Group
4	Dim	NUM	8		Dim
5	LName	CHAR	30		L Name
6	FName	CHAR	30		F Name
7	Gender	CHAR	1		Gender
8	BirthDate	NUM	8	MMDDYY10.	Birth Date
9	State	CHAR	2		State

```
*Routine 3.1. Create attribute set;
proc sql; *create attribute set from input file;
create table Columns as
select varnum as Num 'Num', name as Name 'Name',
       upcase(type) as Type 'Type',
       length as Len 'Len',
       format as Format 'Format',
       case when label^= ' ' then label
            else name end as Label 'Label'
from dictionary.columns
where libname='CDC' and
       memname='DUPLICATESOUNDCODES';
quit;
proc print data=Columns noobs label;
title "Output 3.1. Data Set Columns";
run;
title;
```

This section shows how dictionary tables and the SAS macro facility establish a dynamic link between descriptor and data portions of SAS files. Attribute set *Columns* in Output 3.1 shows that variables *Gender* and *IdNum* are of character type with lengths 1 and 4 bytes respectively (Len = 1 and Len = 4.) This information contributes to making lines 9, 12 and 13 flexible (see Routine 3.3.)

```
*Routine 3.2. Array dimensions from input file;
data _null_;
set cdc.DuplicateSoundCodes end=last;
retain dims;
dims=max(dims,Dim); *maximum dimension;
if last then do;
call symput('grps',trim(left(Group)));
call symput('dims',trim(left(dims)));
end; *if last; else;
run;
%put grps=&grps dims=&dims;
grps=3 dims=9
```

```
*Routine 3.3. Character array lengths;
%let maxdup=15; %*expected maximum # duplicates;
data _null_;
set Columns;
if upcase(Name) in ('IDNUM','GENDER') then
call symput(trim(Name)||'Len',trim(left(Len)));
else;
if upcase(Name)='IDNUM' then
call symput('DupIdNumLen',trim(left(&maxdup*Len)));
else;
run;
%put maxdup=&maxdup IdNumLen=&IdNumLen
GenderLen=&GenderLen DupIdNumLen=&DupIdNumLen;
maxdup=15 IdNumLen=4 GenderLen=1 DupIdNumLen=60
```

Output 2.3 helps understand the logic of the above routine. Below are array definitions, which would decisively contribute to making Routine 2.5 fully flexible.

```
*Routine 3.4. Flexible array definitions;
9.   array sex{&grps,&dims} $ &GenderLen _temporary_; *Gender lookup array;
10.  array dob{&grps,&dims} _temporary_; *BirthDate lookup array;
11.  array yes{&grps,&dims} _temporary_; *Match lookup array by Gender & BirthDate;
12.  array idn{&grps,&dims} $ &IdNumLen _temporary_; *individual IdNum lookup
    array;
13.  array ids{&grps,&dims} $ &DupIdNumLen _temporary_; *duplicate IdNums;
```

Conclusion

Strong matching variables are sometimes inexistent in real-life files for privacy or other reasons. In this case, a combination of two or more variables may provide a near exact match. When removing duplicates, the SORT procedure with the NODUPKEY option offers no solution in the absence of a strong identifier. The SOUNDS LIKE operator of the SQL procedure works well when a master file such as a do-not-solicit list joins a transaction data set such as a campaign file. In the case of new HIV/AIDS patients, there is no central repository. The SOUNDEX function produces sound codes for variables such as last name within a single list. Look-up arrays offer a convenient way to identify and isolate potential duplicates, especially when using additional variables such as gender and birth date. Non-trivial programming is made flexible by automation through SAS macro language. Finding a way to dynamically link data and descriptor portions of SAS files is a critical component of process automation. Through the creation of attribute sets, SQL dictionary tables offer a remarkable solution to this important challenge, particularly when combined with the SAS macro facility.

References

- [1] SAS Institute Inc. 2004, *SAS® 9.1 Language Reference: Dictionary, Volumes 1, 2 and 3*. Cary, NC: SAS Institute Inc. Pages 826 – 827.
- [2] SAS Institute Inc. 2004, *Base SAS® 9.1 Procedure Guide*. Cary, NC: SAS Institute Inc. Pages 1150 – 1151.

Acknowledgements

I am thankful to my former SAS and analytics student Daniel Wambe for bringing the "CDC puzzle" to my attention. I will forever be indebted to my son Nginyan Gwet, my daughter Lama Ngo Gwet and my wife Marie Ngo Mbelek for enduring my long work hours and for providing unconditional support.

Contact Information

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

Bikila bi Gwet (Kila Gwet) alias Andre Gwet
Bikila Analytics Inc
309 N Brooke Dr
Canton GA 30114-9403
Work Phone: (770) 345-3769 / Cell Phone: (770) 598-0088 / Fax: (770) 704-1878
Email: kila.gwet@b-analytics.com / bbgwet@yahoo.com
Web: <http://www.b-analytics.com>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. *B-Analytics™* is a trademark of Bikila Analytics Inc. Other brand and product names are trademarks of their respective companies.