Paper 106-2007

Squeaky Clean Data with SAS® Data Quality Server

Faron Kincheloe, Baylor University, Waco, TX

ABSTRACT

Anyone who uses names and addresses in his or her business knows that duplicates can be a costly headache. Identity theft legislation has essentially taken away the Social Security number as a unique identifier making duplicates even harder to identify. The DQMATCH function in the SAS® Data Quality Server module is a powerful tool for identifying potential matches so the data can be cleaned. However, as with any matching tool, there is a trade off between false positives on one side and overlooking true matches on the other. This paper covers the fundamentals of using the DQMATCH function to clean names and addresses. Additionally, it presents some programming techniques that can be used to optimize the matching process. These techniques enable you to win the match game by detecting the maximum number of true matches while minimizing the number of false matches.

INTRODUCTION

The SAS® Data Quality Server module is a successor to the Data Quality Cleanse module that was introduced in SAS® Version 8. Both versions provide procedures and functions that allow you to analyze your data for patterns and redundancies as well as create standardization. The SAS® Data Quality Server with Version 9.1 provides additional power and flexibility. The matching techniques described in this paper are relevant to both versions. However, the function names and syntax were totally changed in Version 9.1.

The SAS® Data Quality Server provides for the usage of a number of locales. Locales accommodate differences in region specific formats. For example, addresses in the United States are formatted differently from addresses in Great Britain. When the SAS® Data Quality Server attempts to parse addresses for matching, it is important for it to know which format to expect. For the purposes of this paper, all of the research and documentation was done using the U.S. English (ENUSA) locale.

Within the ENUSA locale are various match definitions that may be used to analyze a number of types of organizational data. Among those are ACCOUNT NUMBER, ADDRESS, CITY, DATE, E-MAIL, NAME, ORGANIZATION, PHONE, STATE, TEXT, and ZIP. The scope of this paper is limited to the use of ADDRESS, NAME and TEXT definitions for the purpose of identifying duplicates in a contact database containing the names and addresses of people. The ORGANIZATION definition could be used in lieu of the NAME definition to analyze data containing the names of businesses, churches, or similar entities. Additional data elements such as social security number, birth date, and phone number, when available, can be extremely useful in identifying and resolving duplicate entries. However, the aim of this paper is to provide an efficient method of minimizing duplicate entries when only a name and mailing address are available.

THE BASICS

In order to use the DQMATCH function described in this paper, you must have a license for the SAS® Data Quality Server and at least one locale as they are licensed separately from Base SAS®. The SAS® Data Quality Server is often discussed in the context of Data Warehousing or Data Quality Solutions. However, you do not have to have a data warehouse to enjoy the benefits of the tools provided by SAS® Data Quality Server. SAS® Data Quality Server is available on at least the Windows, UNIX, and z/OS platforms. All examples in this document were taken from a Windows installation.

To ensure that the proper locales are loaded into memory, a line similar to the one shown below should be used at the beginning of each data cleansing program (The path value following DQSETUPLOC may be different depending upon your installation of SAS®.):

```
%DQLOAD(DQLOCALE=(ENUSA), DQSETUPLOC='E:\Program Files\SAS\SAS
9.1\dquality\sasmisc\dqsetup.txt');
```

The DQMATCH function may be invoked from a DATA step, PROC SQL, or SCL. The function returns a match code that is an encoded representation of the character variable being analyzed. The length of the match code can vary from 1 to 255 depending on the match definition that is being used. Equivalent match codes can be grouped together to identify potential duplicates within the data. The example below is taken from a DATA step:

```
  MC_name=dqmatch(fullname, 'name', 60, 'ENUSA');
```

In this example, MC_name is the variable that will contain the match codes returned by the DQMATCH function.

The variable fullname contains the names of the people in our dataset.

The ‘name’ argument specifies the definition that the function is to use to analyze our data. We want it to treat fullname as a name as opposed to text or an address.

The number 60 represents the desired sensitivity and controls the amount of information that will be provided by the match code. This is an optional value. Valid values range from 50 to 95. If no value is supplied, the default value of 85 will be assumed. The higher the sensitivity number, the more similar names would have to be to return the same match code. In this example, a sensitivity value below 55 would require only the last name to match in order to return identical match codes. We have found that a sensitivity of 60 or 70 works well for the results we want to achieve with our data. There is a trade off between false matches and missed matches when determining the sensitivity level. We have decided to tolerate a higher number of false matches in order to increase the detection of true duplicates.

It does not appear that match results change incrementally with the changes in sensitivity numbers. For the list of names used to develop this paper, there was no change in the number of matches using a sensitivity of 60 or 65. However, there was a significant change between sensitivity levels of 65 and 70. There was only a very small change when the sensitivity level jumped from 70 to 85.

The locale (‘ENUSA’) is also an optional argument. If your data contains addresses from different parts of the world, you might want to produce separate match codes for various locales.

The table below shows the name values that were passed to the DQMATCH function and the match codes that were returned using the function call statement shown above. The column labeled MC_name95 shows what the match codes would be if the sensitivity were increased to 95.

	fullname	MC_name	MC_name95
1	Thomas Weldon Johnson	CB4~B$$$$$$$$$$$	CB4B$$$$$~B$$$$
2	Thomas Johnson	CB4~B$$$$$$$$$$$	CB4B$$$$$~B$$$$
3	Tom Johnson	CB4~B$$$$$$$$$$$	CB4B$$$$$~B$$$$
4	Johnson, Thomas Weldon	CB4~B$$$$$$$$$$$	CB4B$$$$$~B$$$$
5	Johnson, Thomas Wendell	CB4~B$$$$$$$$$$$	CB4B$$$$$~B$$$$
6	Johnson, Theodore	CB4~8$$$$$$$$$$$	CB4B$$$$$~87$$$

Notice that only the person is row 6 is perceived to have a different name as indicated by a different match code. The DQMATCH function is able to correctly parse the components of the name regardless of the format in which the names are presented as long as standard conventions are used. “Tom” is recognized as a shortened form or nickname for “Thomas”. Therefore, row 3 returns the same match code as the others. Even though DQMATCH is able to properly parse the components of a name, the middle name appears to be ignored in the matching process. Row 2 has no middle name and rows 3 and 4 have two different middle names, yet all three receive the same match code. Even the highest sensitivity level does not distinguish among missing and different middle names.

The following paragraphs will demonstrate and explain the SAS® programming code used to produce a basic list of matching names and addresses. The data for this program is in a dataset called CONTACTS which contains name, street address, city, state and zip code. The DATA step shown below creates a new dataset that contains the original data along with match codes for name and street.

```
data MC_contacts;
  set contacts;
** Match code for name **;
  MC_name=dqmatch(name, 'name', 60, 'ENUSA');
** Match code for street address **;
  MC_addr=dqMatch(street,'ADDRESS', 60 ,'ENUSA');
run;
```

The next step is to eliminate clearly unique records leaving a dataset containing suspected duplicates. The PROC SQL statement below groups records together using the match codes for name and address. Any group with a count of 2 or more is retained as a potential duplicate. The contents of this dataset can then be printed as a guide for cleaning up the original data.

```
proc sql;
   create table matches1 as
   select * from MC_contacts
   group by MC_name, MC_addr
   having count(*) ge 2
   order by MC_name;
quit;
```

The basic technique described above identified 977 suspected duplicates from a mailing list of 62,000 prospective Baylor students. This mailing list was taken directly from Baylor's student information system where the data had been entered or imported using our best practices. No other screening had been used to validate addresses or cleanse the data. About 4% of the suspected duplicates appeared to be false matches. Further investigation revealed that fewer than 70% of the actual duplicates had been identified.

THE CHALLENGES

There are a number of naturally occurring anomalies that make it difficult for the DQMATCH function to accurately detect all of the duplicates. Many of these challenges can be overcome by modifying the parameters of DQMATCH and making multiple passes through the data. The following paragraphs contain descriptions of these challenges and how they were overcome using an enhanced detection process.

Last Names: Even when entered correctly, there is a wide variation in the spellings of last names. This limits the amount of sensitivity that can be built into match functionality for last names. Part of the matching process involves removing some of the vowels so a misspelling involving a vowel does not usually affect the matching process. However, when a consonant is wrong two separate match codes will be created. For example, the letter D is often mistaken for the letters O or P. When this happens, the two different spellings create two different match codes and the names are not considered as duplicates.

Examples		
	name	**MC_name**
1	John Deulschlaeger	8W4CB$$$$$$$$$$$
2	John Oehlschlaeger	#W4CB$$$$$$$$$$$
3	Josh Gelger	FWFC4$$$$$$$$$$$
4	Joshua Geiger	FFYC4$$$$$$$$$$$

This problem is resolved by creating a match code for the first name and grouping this with the address match code and the zip code. Most of the additional duplicates from the enhanced process were detected by overcoming the last name challenge.

Marriage/Divorce: In some instances a marriage or divorce has taken place between the time of the first contact and a subsequent contact. Several instances of this were observed where a woman took her husband's family name or returned to using her maiden name but did not change her mailing address. Often, confirmation of this could be observed as a hyphenated last name or the former last name in the place of the middle name.

Examples		
	name	**MC_name**
1	Ruth Brown-Dawson	MYLY~$$$$$$$$$$$
2	Ruth Dawson	84BY~$$$$$$$$$$$
3	Mitchell, Lauren Lyn	B~JWY$$$$$$$$$$$
4	Durham, Lauren Mitchell	8YBWY$$$$$$$$$$$

This problem is resolved with the last name process described above where the match is based on first name, address, and zip code.

Middle Names: It is not uncommon for a person's preferred name to be the middle name or some variation of the middle name. Depending upon the source of the contact information, it may contain the full legal name or only the preferred name and last name. "Official" documents such as entrance test scores and transcripts are likely to have the full name whereas a reply card is more likely to have the preferred name. This may cause the redundancy to be overlooked.

Examples		
	name	**MC_name**
1	Greer, Ryan	FYYYB$$$$$$$$$$
2	Greer, William Ryan	FYYLW$$$$$$$$$$
3	Ford, Ben	GY~MB$$$$$$$$$$
4	Ford, Robert Benjamin	GY~YM$$$$$$$$$$

The solution for this is to create an assumed preferred name for each record based on the presence or absence of a middle name. (If an actual preferred name exists in the data, it should be used in lieu of the assumed preferred name described here.) If there is no middle name or only a middle initial, the first name is assumed to be the preferred first name. If the middle name exists, it is assumed to be the preferred name. A preferred full name is created by concatenating the preferred name to the last name. A match code for comparison with other records is then created for the new preferred full name. It is necessary to use the preferred full name when creating the match code because DQMATCH does not do a good job matching name variations such as “Beth” for “Elizabeth” when a single name is passed to the function.

Twins: It is a common practice for parents of twins to give their children similar names. In many cases, these names are so similar that they have the same match code. In the university environment, most of the contacts still live at home with their parents or use that address as their permanent address. For this reason, the addresses for twins also produce identical match codes.

Examples			
	name	**MC_name60**	**MC_name95**
1	Timothy Horton	2Y~~B$$$$$$$$$$	2Y~B$$$$$~B7$$$
2	Thomas Horton	2Y~~B$$$$$$$$$$	2Y~B$$$$$~B$$$$
3	Frank Spaulding	4MWGY$$$$$$$$$$	4MW8BF$$$GYB$$$
4	Francis Spaulding	4MWGY$$$$$$$$$$	4MW8BF$$$GYB$$$

An acceptable solution for this problem has yet to be found. Twins make up the majority of the false matches in both the basic and enhanced matching processes. Sensitivity levels have a minimal effect with this issue. The match codes shown above are produced at sensitivity levels of 60 and 65. At sensitivity levels of 70 and higher, Timothy and Thomas are given different match codes. However, at even the highest sensitivity level Frank and Francis are given identical match codes.

P.O. Boxes: When DQMATCH receives a name or an address, it attempts to parse out the various components before creating the match code. In the case of an address, it attempts to identify elements such as street number, pre-direction, street name, and street type. Since post office box addresses do not follow the typical street address format, DQMATCH does not do a good job creating a match code. P.O. Boxes were a contributor to the number of false matches in the basic matching process. While the number of these false matches was not significant for potential university students, it could be a significant nuisance if you are analyzing business contacts where the majority of addresses are box numbers. Most of these false matches could be eliminated by simply including the zip code in the matching criteria. However, the zip code would not be a viable solution if large numbers of the contacts were from the same city.

Examples			
	STREET	**MC_addr**	**MC_addr_txt**
1	P O Box 1111	$$$$$$ZZ$$$$$$$	NMXZZ$$$$$$$$$$
2	Po Box 11	$$$$$$ZZ$$$$$$$	NMXZZZZ$$$$$$$$
3	# 241pr429	$$$$$$HS$$$$$$$	HSZNYSH-$$$$$$$
4	Po Box 245	$$$$$$HS$$$$$$$	NMXHS5$$$$$$$$$
5	Po Box 115	$$$$$$ZZ$$$$$$$	NMXZZ5$$$$$$$$$
6	P O Box 1176	$$$$$$ZZ$$$$$$$	NMXZZI6$$$$$$$$

The solution to this problem is to create match codes for P.O. Boxes using the text definition instead of the address definition. The examples above show that a P.O. Box can even be confused with another address. This confusion is eliminated by segregating P.O. Box records from other address types when creating the groups of potential matches.

Street Addresses: Even though street addresses follow a general format, there are so many variations that parsing and match creation can be a challenge. This is compounded by mistakes and inconsistencies when addresses are entered. With the improvement and cost reduction of scanning technologies, it appears that more contacts are being entered via optical character recognition. This also contributes to the difficulties of parsing and matching. Missing or mistaken characters are often easy to detect when they turn normal words or names into nonsensical ones. This is not the case for missing or mistaken numbers as an incorrect number looks just as valid as a correct one.

Examples				
	STREET	**MC_addr**	**MC_addr50**	**MC_addr_txt50**
1	5216 Peach Leaf Cv.	5HZN3W$$$$$$$$$	5HN3$$$$$$$$$$$	5HZ6NJ$$$$$$$$$
2	5216 Peachleaf Cove	5HZNJ2$$$$$$$$$	5HNJ$$$$$$$$$$$	5HZ6NJ$$$$$$$$$
3	1106 Evening Sun Lane	ZZ0_VP$$$$$$$$$	ZZ_V$$$$$$$$$$$	ZZ06VP$$$$$$$$$
4	106 Eveing Sun Lane	Z06_VP$$$$$$$$$	Z0_V$$$$$$$$$$$	Z06VPF$$$$$$$$$
5	38W4S8 E. Mary Lane	KDLBYW$$$$$$$$$	KDBY$$$$$$$$$$$	KDLS4D$$$$$$$$$
6	3800458 E Mary Ln	KD0BYW$$$$$$$$$	KDBY$$$$$$$$$$$	KD00S5$$$$$$$$$
7	18827 Forest Bend Creek Way	ZDDGY4$$$$$$$$$	ZDGY$$$$$$$$$$$	ZDDHIZ$$$$$$$$$
8	1880 Forest Bend Crk.	ZDDGY4$$$$$$$$$	ZDGY$$$$$$$$$$$	ZDD0Z4$$$$$$$$$

More program code has been invested in this challenge than any of the others with the lowest payback. However, each method produced some legitimate duplicates that were undetected by another method. Since program code and CPU time are relatively inexpensive, it was decided to retain each method in the interest of detecting even 1 or 2 additional duplicates. The first step is to use the basic match process based on the full name and address with sensitivities of 60. The zip code is not included in this step because a number of duplicate records were found where the two zip codes were different. The second step is to match on a high sensitivity name, a low sensitivity address, and the zip code. Consideration was given to eliminating the address portion of this criterion. However, eliminating the address tripled the percentage of false matches. You might consider dropping the address portion temporarily to search for additional duplicates after the address list has been cleaned. The third step is to use the name, a low sensitivity text code from street, and the zip code. Using the text definition to create a match code for street yielded only a small number of additional matches that were not already detected by other criteria. This third step is not necessary if the address is totally eliminated in the second step described above.

Problems with street address are believed to account for almost all of the true duplicates that cannot be detected. There was one known duplicate in the list that we were never able to detect programmatically using any variation of match codes for the street. The match codes were always different for “106 Eveing Sun Lane” and “1106 Evening Sun Lane”. This is an area where a judgment call has to be made. How many false matches are you willing to tolerate in order to uncover a few more genuine duplicates? There is always a trade off.

Neighbors: Neighbors who have the same or similar first names and live across the street from each other or in the same building contribute to the number of false matches in the enhanced process. These false matches are primarily introduced by the solution to the last name and marriage/divorce challenges. However, the percentage was negligible compared to the number of duplicates that were revealed.

Examples		
	NAME	**STREET**
1	Crystal Godwin	100 Whittington
2	Christopher Perkins	100 Whittington Ave
3	Kirsten David	5406 Red Oak Ln.
4	Christine Young	5407 Red Oak Lane

Like the twins challenge, the neighbors challenge appears to have no equitable solution. If data cleansing is performed on a routine basis, the cleanup list will eventually be comprised almost exclusively of twins, neighbors, and other false matches. These cannot be totally excluded because there is the possibility that a duplicate to one of these records could be introduced into the data at any time. However, there is a work around. Each time the cleanup program is run, save the cleanup list as a permanent dataset that can be retrieved the next time the program is run. Use the old cleanup dataset to create a list of IDs and a repeat flag. After the new cleanup dataset is created, join it to the repeat flag list so that any new records will be missing the repeat flag. This prevents the cleanup personnel from having to repeatedly investigate records that have already been determined to be false matches. The code below demonstrates how to create the repeat flag and merge the datasets together:

```
proc sql;
   create table lasttime as
   select ID, 'Y' as repeat_flag
   from mydata.cleanup
   order by ID;
quit;

** Insert matching code here (This assumes matching code creates a dataset
called cleanup in the work library.) **;

proc sql;
   create table mydata.cleanup as
   select a.*, b.repeat_flag
   from work.cleanup as a
        left join
        work.lasttime as b
   on a.ID=b.ID
   order by zip, MC_addr;
quit;

** Use a PRINT or REPORT procedure to print the contents of the new
mydata.cleanup dataset. **;
```

THE SOLUTION

After incorporating a solution to each challenge into the enhanced matching process, the number of detected duplicates rose from 977 to 1416 while holding the number of false matches below 4%. Although it is difficult to determine precisely, it is believed that over 98% of all the duplicates have now been identified. The following outline is an overview of the steps that make up the enhanced process.

- **I. Prepare the data**
  - **A. Unique ID**
  - **B. Full Name**
  - **C. First Name**
  - **D. Middle Name**
  - **E. Last Name**
  - **F. Preferred Full Name**
  - **G. Street**
  - **H. Zip Code**
  - **I. Match Codes**
    - **1. Full Name 60**
    - **2. Full Name 95**
    - **3. First Name 60**
    - **4. Preferred Full Name 60**
    - **5. Street (address) 60**
    - **6. Street (address) 50**
    - **7. Street (text) 70**
    - **8. Street (text) 50**
- **II. Create groups of matches**
  - **A. P.O. Box addresses**
    - **1. Full Name 60 + Street (text) 70 + Zip Code**
    - **2. First Name 60 + Street (text) 70 + Zip Code**
    - **3. Preferred Full Name 60 + Street (text) 70 + Zip Code**
  - **B. Street addresses**
    - **1. Full Name 60 + Street (address) 60**
    - **2. First Name 60 + Street (address) 60 + Zip Code**
    - **3. Full Name 60 + Street (text) 50 + Zip Code**
    - **4. Full Name 95 + Street (address) 50 + Zip Code**
    - **5. Preferred Full Name 60 + Street (address) 50 + Zip Code**
- **III. Merge groups together**
- **IV. Remove repeated rows and sort by clusters**
- **V. Print list for cleanup**

The actual program code used in the enhanced process along with an explanation of each section follows:

```
data MC_contacts;
  set contacts;
  ID=_N_+900000; *Create a unique ID for each record;
run;
```

The dataset MC_contacts will contain all of the fields necessary to complete the matching process. Each record needs a unique ID number to distinguish repeated records from suspected duplicates when all of the matches are merged back together. The variable _N_ contains the observation number of each record. Here it is added to 900000 to create a six digit ID number. If your data already contains a unique key, the creation of this ID is not necessary.

```
** Create field for full name **;
  namem=  trim(fname)||' '||trim(mname)||' '||trim(lname);
```

If your list has separate fields for each component of the name, you will need to concatenate them together to create a full name field. Otherwise, you will need to use the SAS® Data Quality Server parsing functions to create the separate fields as shown below.

```
** Use Data Quality Server functions to parse name **;
  parsename=dqparse(name, 'name', 'ENUSA');
  fname=dqparseTokenGet(parsename,'Given Name', 'name', 'ENUSA');
  mname=dqparseTokenGet(parsename,'Middle Name', 'name', 'ENUSA');
  lname=dqparseTokenGet(parsename,'Family Name', 'name', 'ENUSA');
```

If some of the names in your data contain prefixes such as Mrs. or Dr., it can be a challenge to identify and separate the components of the name. When DQMATCH assigns a match code it automatically attempts to parse the name. This powerful feature can be tapped by using DQPARSE to extract the individual elements from a name field. SAS® Data Quality Server refers to these elements as tokens. The DQPARSE function creates a delimited text field in which each element of the name is separated by a delimiter. The DQPARSETOKENGET function then returns the requested element or token from the delimited text field.

```
** Create hypothetical preferred name using middle name **;
  if length(mname)=1 or substr(mname,2,1)='.' then prefguess=fname;
  else prefguess=mname;
    prefname=  trim(prefguess)||' '||trim(lname);
```

For test purposes we assume that everyone who supplies a middle name prefers to use that name. If the middle name is missing or only a middle initial is supplied the length will be 1. Some middle initials are punctuated with a period so the substr function checks for that. A hypothetical preferred full name is created by concatenating the assumed preferred name with the last name.

```
** Standardize the zip code field **;
  zip=substr(zip,1,5);
```

While some addresses contain zip+4, most do not. This step insures that all zip codes are limited to the first 5 digits. The SAS® Data Quality Server includes a match definition for zip code which did not seem necessary to meet our objectives.

```
** Match codes for full name **;
  MC_name=dqmatch(name, 'name', 60, 'ENUSA');
  MC_name95=dqmatch(name, 'name', 95, 'ENUSA');
** Match code for first name **;
  MC_fname=dqmatch(fname, 'name', 60, 'ENUSA');
** Match codes for preferred full name **;
  MC_pref=dqmatch(prefname, 'name', 60, 'ENUSA');
** Match codes for street address **;
  MC_addr=dqMatch(street,'ADDRESS', 60 ,'ENUSA');
  MC_addr50=dqMatch(street,'ADDRESS', 50 ,'ENUSA');
** Text Match codes for street address **;
  MC_addr_txt=dqMatch(street,'TEXT', 70 ,'ENUSA');
  MC_addr_txt50=dqMatch(street,'TEXT', 50 ,'ENUSA');
```

The final segment of the DATA step uses DQMATCH to create all of the match codes that will be used to test for duplicate records.

```
proc sql;
   create table match_pob as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) in ('NMX', 'NMZ')
   group by MC_name, MC_addr_txt, ZIP
   having count(*) ge 2 ;
quit;

proc sql;
   create table match_pob_fname as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) in ('NMX', 'NMZ')
   group by MC_fname, MC_addr_txt, ZIP
   having count(*) ge 2 ;
quit;

proc sql;
   create table match_pob_pref as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) in ('NMX', 'NMZ')
   group by MC_pref, MC_addr_txt, ZIP
   having count(*) ge 2 ;
quit;

proc sql;
   create table match_addr as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) not in ('NMX', 'NMZ')
   group by MC_name, MC_addr
   having count(*) ge 2 ;
quit;

proc sql;
   create table match_addr_fname as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) not in ('NMX', 'NMZ')
   group by MC_fname, MC_addr, ZIP
   having count(*) ge 2;
quit;

proc sql;
   create table match_addr_txt as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) not in ('NMX', 'NMZ')
   group by MC_name, MC_addr_txt50, zip
   having count(*) ge 2 ;
quit;

proc sql;
   create table match_name as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) not in ('NMX', 'NMZ')
   group by MC_name95, MC_addr50, ZIP
   having count(*) ge 2 ;
quit;
```

```
proc sql;
   create table match_pref as
   select * from MC_contacts
   where substr(MC_addr_txt,1,3) not in ('NMX', 'NMZ')
   group by MC_pref, MC_addr50, ZIP
   having count(*) ge 2
quit;
```

Each of the SQL procedures above creates a table of suspected matches based on different criteria as indicated by the different variables used in the group by statement. The first three procedures analyze only records with P.O. Box type addresses. The remaining five procedures analyze records with street addresses. It was discovered that the match codes using the text definition begin with “NMX” for all variations of P.O. Box and “NMZ” for POB. Therefore, it is much simpler to subset the data using the match code than to try to include all variations of the literal text in a where clause.

```
data matches2;
   set match_pob match_pob_fname match_pob_pref match_addr
   match_addr_fname match_addr_txt match_name match_pref;
run;
```

This DATA step concatenates all of the tables into one table containing all potential matches.

```
proc sql;
   create table cleanup as
   select distinct ID, NAME, STREET, CITY, STATE, ZIP, MC_addr
   from matches2
   order by zip, MC_addr;
quit;
```

There is significant overlap in the matches detected by each of the SQL procedures. Therefore, rows from the original data may be repeated several times. By including the ID and using the distinct statement, we can insure that each suspected duplicate only appears once in our output. The order in which the records are presented is critical. The output is only useful if the suspected matches are presented in pairs or clusters. The use of so many different match codes to identify the duplicates makes it impossible to get a perfect grouping. You may need to try different sort criteria based on the variability patterns in your data. The objective is to get the matches as close together as possible. Zip and address match code seemed to work well for our data. Another combination that worked fairly well was address match code and first name match code.

CONCLUSION

SAS® Data Quality Server provides some powerful tools to help you identify redundancies in your data. By understanding the behavior of these tools and the patterns within your data, you can create procedures to further enhance the power of these tools and win the match game.

REFERENCES

SAS Institute Inc. 2004. *SAS® 9.1.2 Data Quality Server: Reference.* Cary, NC: SAS Institute Inc.

ACKNOWLEDGEMENTS
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.

UNIX is a registered trademark of The Open Group.

Windows is a registered trademark of Microsoft Corporation.

z/OS is a registered trademark of International Business Machines Corporation.

Special thanks to Lin Zhang, Graduate Statistics student, Baylor University for the numerous hours she spent testing and documenting the processes described in this document.

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

Faron Kincheloe
Baylor University
One Bear Place #97032
Waco, TX 76798
Phone: (254) 710-8835
Email: Faron_Kincheloe@baylor.edu