**Paper 191-2008**

# Get Your Hands Dirty Cleaning Your Data with SAS® Data Quality Server

Faron Kincheloe, Baylor University, Waco, TX

## ABSTRACT

The SAS Data Quality Server module brings the power of DataFlux® into the SAS platform. This can be extremely beneficial for those who are already using SAS to manipulate and cleanse data. During this hands-on workshop, attendees will use SAS Data Quality Server to parse the elements of a name, identify duplicate names and addresses in dirty data, estimate gender, and standardize organization names. We will cover the syntax and parameters used for creating matches and standardization schemes. We will experiment with these parameters to show how they affect the data cleansing process. A real-world example will be presented to show the impact data quality has on decision making.

## INTRODUCTION

This workshop has three objectives. The first objective is to demonstrate the application and use of some of the primary functions and procedures available within SAS Data Quality Server. We will use the DQSCHEME procedure to create and apply standardization to organization names. We will demonstrate an alternative method of applying standardization using the DQSchemeApply function and discuss the differences.

The second objective is to identify a number of common anomalies and irregularities that occur in data containing names and addresses. The third objective is to show how the SAS Data Quality Server tools can be used to overcome these data issues. Instead of covering these objectives one at a time, they will all be integrated into each exercise we perform as we go through the workshop.

After we have standardized the organization names, we will work with personal names and addresses from a contact list. We will use the DQPARSE and DQPARSETOKENGET functions to identify the components of a full name and separate them into distinct fields. We will demonstrate the use of the DQGENDER function to determine the gender of our contacts. Finally, we will utilize the DQMATCH function to identify duplicate records that are not identical and might otherwise not be recognized as duplicates.

## THE BASICS

In order to use the functions described in this paper, you must have a license for the Data Quality Server and at least one locale as they are licensed separately from Base SAS®. The 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 Data Quality Server. 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='C:\Program Files\SAS\SAS
9.1\dquality\sasmisc\dqsetup.txt');
```

SAS® Data Quality Server provides procedures and functions that allow you to analyze your data for patterns and redundancies as well as create standardization. If you are using data from the international market, 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 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. This paper describes 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. 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 matching procedure described in this paper provides an efficient method of minimizing duplicate entries when only a name and mailing address are available. The ORGANIZATION definition can be used in lieu of the NAME definition to analyze data containing the names of businesses, churches, or similar entities as shown in the discussion on the DQSCHEME procedure.

## CREATING A WINNING SCHEME

Since Baylor is a private university with a religious background, churches are a fertile ground for recruiting new students. However, church names are entered into the prospect database as free form text. As a result, these data are unusable for reporting purposes without some sort of standardization. For example, a large local church, known to be home to many Baylor students did not appear in the top 25 in a report listing the number of students by church. In fact, this church appeared as number 30, 37 and 57 in the list. Further investigation revealed that 56 students had listed this church 17 different ways in the database. The DQSCHEME procedure was used to create a standardization scheme for church names. A program runs every night using DQSCHEME to apply the scheme to church names entered into the database. This converts the irregular entries into standardized entries that can be used for reporting. After standardization, the church mentioned above turned out to be fourth in the report of number of students by church. The table below is an excerpt from a report that illustrates the various data problems.

| State | City | Church | N |
|---|---|---|---|
| AR | CABOT | FBC | 1 |
| | CONWAY | WOODLAND HEIGHTS BAPTIST | 3 |
| | | WOODLAND HEIGHTS BAPTIST CHURC | 2 |
| | DUTCH MILLS | DUTCH MILLS BABTIST CHURCH | 1 |
| | GREEWOOD | GREENWOOD UNITED METHODIST | 1 |
| | N. LITTLE ROCK | LAKEWOOD METHODIST | 2 |
| | NORTH LITTLE ROCK | LAKEWOOD UNITED METHODIST | 1 |
| CO | BOLDER | APPLEWOOD VALLEY | 1 |
| | COLORADO SPGS | WOODMAN VALLEY CHAPEL | 1 |
| | | WOODMEN VALLEY CHAPEL | 1 |
| | COLORADO SPRINGS | WOODMAN VALLEY | 1 |
| | | WOODMAN VALLEY CHAPEL | 2 |
| | | WOODMEN CHAPEL | 1 |
| | | WOODMEN VALLEY | 1 |
| | | WOODMEN VALLEY CHAPEL | 18 |
| | | WOODMEN VALLY CHAPEL | 1 |
| | COLORADO SPRS | WOODMAN VALLEY CHAPEL | 1 |
| | COLORDO SPRINGS | WOODMEN VALLEY CHAPEL | 1 |
| TX | WACO | 1ST BAPTIST CHURCH WOODWA | 1 |
| | | 1ST BAPTIST OF WOODWAY | 1 |
| | | 1ST BAPTIST WOODWAY | 2 |
| | | 1ST BAPTIST-WOODWAY | 2 |
| | | 1ST WOODWAY - BAPTIST | 1 |
| | | 1ST WOODWAY UNITED METHODIST | 1 |
| | | COLUMBUS AVENUE BABTIST | 1 |
| | WOODWAY | 1ST BAPTIST | 1 |
| | | 1ST BAPTIST CHURCH OF WOODWAY | 2 |
| | | 1ST BAPTIST WOODWAY | 2 |
| | | FBC | 6 |
| | | FBC OF WOODWAY | 18 |
| | | FBC WOODWAY | 14 |
| | | FBC, WOODWAY | 2 |
| | | FELLOWSHIP BIBLE | 1 |
| | | FELLOWSHIP BIBLE CHURCH | 1 |
| | | FIRST BABTIST CHURCH WOODWAY | 3 |
| | | FIRST BABTIST WOODWAY | 3 |
| | | FIRST BAPTIST | 163 |
| | | FIRST BAPTIST  WOODWAY | 2 |
| | | FIRST BAPTIST CHURCH | 35 |
| | | FIRST BAPTIST CHURCH - WOODWAY | 2 |
| | | FIRST BAPTIST CHURCH OF W | 5 |
| | | FIRST BAPTIST CHURCH OF WOODWA | 14 |

| | | | |
|---|---|---|---|
| | | FIRST BAPTIST CHURCH WOOD | 2 |
| | | FIRST BAPTIST CHURCH WOODWAY | 16 |
| | | FIRST BAPTIST OF WOODWAY | 6 |
| | | FIRST BAPTIST WOODAY | 2 |
| | | FIRST BAPTIST WOODWAY | 104 |
| | | FIRST BAPTIST-WOODWAY | 2 |
| | | FIRST METHODIST | 10 |
| | | FIRST UNITED METHODIST | 2 |
| | | FIRST UNITED METHODIST CHURCH | 1 |
| | | FIRST UNITED METHODIST WOODWAY | 2 |
| | | FIRST WOODWAY | 18 |
| | | FIRST WOODWAY BABTIST CHURCH | 3 |
| | | FIRST WOODWAY BAPTIST | 12 |
| | | FIRST WOODWAY BAPTIST CHURCH | 2 |
| | | FIRST WOODWAY UMC | 2 |
| | | FRIST BAPTIST CHURCH | 1 |

As you can see from this example there are a number of misspellings and variations in the city and name fields. There are also various abbreviations used in both fields. There are also several cases where the name is truncated at various lengths. All of these differences can be corrected with a properly constructed scheme. The same church names appear under both cities of Waco and Woodway. The Postal Service recognizes both as acceptable names for the same zip code. This ambiguity cannot be corrected with a scheme and requires additional logic in our cleanup program.

The creation of a scheme to standardize church names is relatively simple from a programming standpoint but involves a significant amount of manual effort. The first step is to analyze the data and create a scheme using the DQSCHEME procedure as illustrated below. The procedure shown actually creates two schemes; one for the church name and the other for cities.

```
options NOLABEL;

proc dqscheme nobfd data=churches;
   create scheme=std_church analysis=as_church90
     matchdef='Organization (Scheme Build)'
     var=orig_church sensitivity=90 locale='ENUSA';
   create scheme=std_city analysis=as_city matchdef='City (Scheme Build)'
     var=orig_city locale='ENUSA';
run;
```

The NOLABEL global option and the nobfd procedure option are specified to ensure that the scheme can be stored as a SAS dataset. If you want to use dfPower Studio software to work with the scheme use the bfd procedure option instead of nobfd. The create statement will create two datasets. One dataset is the scheme itself (std_church) and the other is an analysis dataset (as_church90) showing how the scheme was built. The matchdef= parameter specifies the type of data definition that will be used to standardize the data. In this case we are treating our church names as organizations. The definition for creating schemes is different from the definition that would be used to create match codes and is distinguished with “(Scheme Build)” as part of the definition name. The var= parameter specifies the field that contains the data to be standardized.

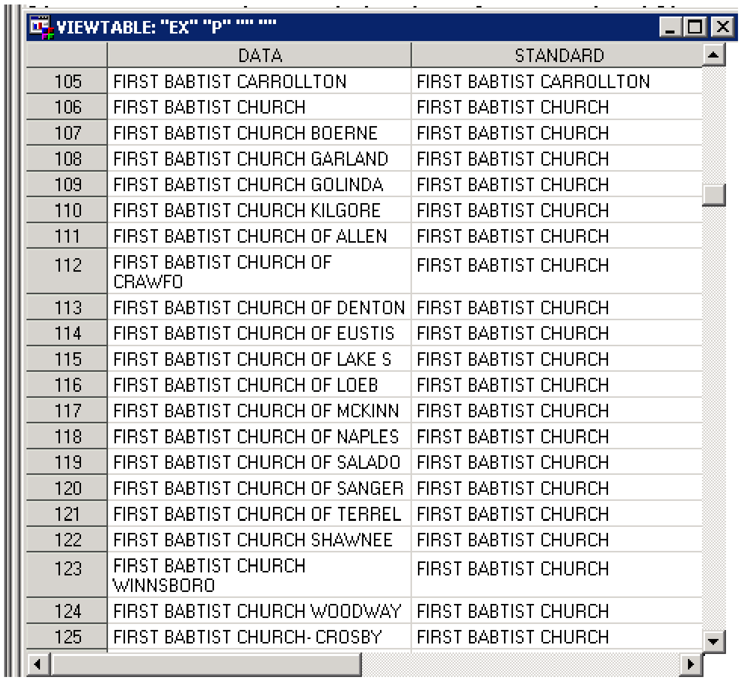
Sensitivity is an optional value. Valid values range from 50 to 95. Since no value is supplied when we create the city scheme, the default value of 85 will be assumed. The sensitivity setting affects how similar the names have to be in order to be included in the same cluster. This will determine the number of manual changes that have to be made to the scheme once it is created. Examples of different sensitivities will be shown later in this paper.

The analysis dataset contains three variables. The first variable shows the number of times that a specific church name occurs in our data. The second variable is the church name field. The third variable is a cluster number to identify other church names that are considered to be the same. If a church name is considered to be unique, it will not be assigned a cluster number and it will not be included in the scheme dataset. The table shown below is from the analysis dataset created with a sensitivity of 90.

| Obs | COUNT | orig_church | CLUSTER |
|---|---|---|---|
| 6 | 1 | FIRST BABTIST ARLINGTON | . |
| 7 | 1 | FIRST BABTIST CHUCH | . |
| 8 | 29 | FIRST BABTIST CHURCH | 1 |
| 9 | 3 | FIRST BABTIST CHURCH OF ALLEN | 1 |
| 10 | 3 | FIRST BABTIST CHURCH WOODWAY | 1 |
| | | | |
| 55 | 166 | FIRST BAPTIST | 3 |
| 56 | 1 | 1ST BAPTIST | 3 |
| 57 | 1 | FIRST BAPTIST-BIRCHWOOD | . |
| 58 | 1 | FIRST BAPTIST-BROWNWOOD | . |
| 59 | 37 | FIRST BAPTIST CHURCH | 4 |
| 60 | 22 | FIRST BAPTIST CHURCH WOODWAY | 4 |
| 61 | 16 | FIRST BAPTIST CHURCH OF WOODWA | 4 |

Several factors come into play when the scheme is being created. Names are treated more like text until the word CHURCH is encountered. The presence of this word also affects how the rest of the name is treated. There is no number in the cluster column until observation 8 indicating that all of these values are considered unique. Notice at observation 55 how FIRST and 1$^{ST}$ were clustered together. Both will be standardized as FIRST BAPTIST in the scheme because that is the value that occurs most frequently in this cluster. However, due to the high sensitivity level, words coming after BAPTIST were considered and the remaining two values were considered to be unique. Such is not the case when CHURCH is part of the name where the remaining words are ignored.

The image below is a view of the scheme data set that was created by this procedure.


VIEWTABLE: "EX" "P" "" ""

| | DATA | STANDARD |
|---|---|---|
| 105 | FIRST BABTIST CARROLLTON | FIRST BABTIST CARROLLTON |
| 106 | FIRST BABTIST CHURCH | FIRST BABTIST CHURCH |
| 107 | FIRST BABTIST CHURCH BOERNE | FIRST BABTIST CHURCH |
| 108 | FIRST BABTIST CHURCH GARLAND | FIRST BABTIST CHURCH |
| 109 | FIRST BABTIST CHURCH GOLINDA | FIRST BABTIST CHURCH |
| 110 | FIRST BABTIST CHURCH KILGORE | FIRST BABTIST CHURCH |
| 111 | FIRST BABTIST CHURCH OF ALLEN | FIRST BABTIST CHURCH |
| 112 | FIRST BABTIST CHURCH OF CRAWFO | FIRST BABTIST CHURCH |
| 113 | FIRST BABTIST CHURCH OF DENTON | FIRST BABTIST CHURCH |
| 114 | FIRST BABTIST CHURCH OF EUSTIS | FIRST BABTIST CHURCH |
| 115 | FIRST BABTIST CHURCH OF LAKE S | FIRST BABTIST CHURCH |
| 116 | FIRST BABTIST CHURCH OF LOEB | FIRST BABTIST CHURCH |
| 117 | FIRST BABTIST CHURCH OF MCKINN | FIRST BABTIST CHURCH |
| 118 | FIRST BABTIST CHURCH OF NAPLES | FIRST BABTIST CHURCH |
| 119 | FIRST BABTIST CHURCH OF SALADO | FIRST BABTIST CHURCH |
| 120 | FIRST BABTIST CHURCH OF SANGER | FIRST BABTIST CHURCH |
| 121 | FIRST BABTIST CHURCH OF TERREL | FIRST BABTIST CHURCH |
| 122 | FIRST BABTIST CHURCH SHAWNEE | FIRST BABTIST CHURCH |
| 123 | FIRST BABTIST CHURCH WINNSBORO | FIRST BABTIST CHURCH |
| 124 | FIRST BABTIST CHURCH WOODWAY | FIRST BABTIST CHURCH |
| 125 | FIRST BABTIST CHURCH- CROSBY | FIRST BABTIST CHURCH |

The DATA column represents the value that is expected in the dirty data. The STANDARD column shows the replacement value that will be supplied when the scheme is applied. Notice the strange label that appears in the title of the data set window. This is metadata that tells SAS about this data set as a scheme. The 'EX' specifies the lookup option of the scheme. This is the default for EXACT meaning that an input value has to exactly match the scheme DATA value in order to be standardized by the scheme. The 'P' value designates the PHRASE mode option for applying the scheme. This means that the entire phrase will be compared to the DATA as opposed to comparing an ELEMENT of the phrase to the DATA in the scheme. If you manually import external data from a source such as Excel into a scheme data set, you must preserve this label or Data Quality Server will not recognize that the scheme exists. At row 106 we can see how cluster 1 from the analysis data set becomes a part of the standardization scheme. In the analysis data set, values are sorted by cluster and number of occurrences. In the scheme dataset, values are sorted alphabetically.

The next example is the analysis data produced by using a sensitivity of 80.

| Obs | COUNT | orig_church | CLUSTER |
|---|---|---|---|
| 3 | 1 | FIRST BABIST CHURCH | . |
| 4 | 16 | FIRST BABTIST | . |
| 5 | 1 | FIRST BABTIST ARLINGTON | 1 |
| 6 | 1 | FIRST BABTIST OF TALLULAH | 1 |
| 7 | 29 | FIRST BABTIST CHURCH | 2 |
| 8 | 3 | FIRST BABTIST CHURCH OF ALLEN | 2 |
| 9 | 3 | FIRST BABTIST CHURCH WOODWAY | 2 |
| 10 | 1 | FIRST BABTIST CHUCH | 2 |
| | | | |
| 55 | 166 | FIRST BAPTIST | 8 |
| 56 | 1 | 1ST BAPTIST | 8 |
| 57 | 1 | FIRST BAPTIST-BIRCHWOOD | 9 |
| 58 | 1 | FIRST BAPTIST-BROWNWOOD | 9 |
| 59 | 37 | FIRST BAPTIST CHURCH | 10 |
| 60 | 22 | FIRST BAPTIST CHURCH WOODWAY | 10 |
| 61 | 16 | FIRST BAPTIST CHURCH OF WOODWA | 10 |
| 62 | 9 | FIRST BAPTIST CH-WOODWAY | 10 |
| 63 | 5 | FIRST BAPTIST CHURCH OF W | 10 |

In the example above, some clustering is being done based on the last words in the church name. In this case FIRST BABTIST OF TALLULAH will be standardized in the scheme as FIRST BAPTIST ARLINGTON. The SCHEME procedure does not attempt to correct for BAPTIST being misspelled as BABTIST.

The next example is produced by using a sensitivity of 70.

| Obs | COUNT | orig_church | CLUSTER |
|---|---|---|---|
| 3 | 1 | FIRST BABIST CHURCH | . |
| 4 | 29 | FIRST BABTIST CHURCH | 1 |
| 5 | 16 | FIRST BABTIST | 1 |
| 6 | 5 | FIRST BABTIST WOODWAY | 1 |
| 7 | 3 | FIRST BABTIST CHURCH OF ALLEN | 1 |
| 8 | 3 | FIRST BABTIST CHURCH WOODWAY | 1 |
| 9 | 1 | FIRST BABTIST ARLINGTON | 1 |
| 10 | 1 | FIRST BABTIST CARROLLTON | 1 |
| | | | |
| 173 | 54 | BRENTWOOD BAPTIST CHURCH | 14 |
| 174 | 30 | BRENTWOOD BAPTIST | 14 |
| 175 | 4 | BRENTWOOD | 14 |
| 176 | 3 | BRENTWOOD OAKS | 14 |
| 177 | 3 | BRENTWOOD UNITED METHODIST | 14 |
| 178 | 3 | BRENTWOOD UNITED METHODIST CHU | 14 |
| 179 | 2 | BRENTWOOD BABTIST CHURCH | 14 |
| 180 | 2 | BRENTWOOD HILLS CHURCH OF CHRI | 14 |
| 181 | 2 | BRENTWOOD METHODIST | 14 |
| 182 | 2 | BRENTWOOD OAKS CH-CHRIST | 14 |

The first one or two words of the name are standardized in a manner similar to that of the other sensitivities. However, words coming after BABTIST are treated almost exactly opposite of the way they are treated with a sensitivity of 90. With the higher sensitivity these words were considered to be unique. Now they are considered to be the same. Notice how the values are ordered according to the count. FIRST BABTIST ARLINGTON will now be standardized as FIRST BABTIST CHURCH using this sensitivity. When we get down to row 173 and beyond we see that this sensitivity is indiscriminately grouping together Baptists, Methodists, and others.

For Baylor, a lower sensitivity is desirable since we have a separate field for the city and do not want to include the city as part of the church name. However, more accuracy is desired so we will try a sensitivity of 75.

| Obs | COUNT | orig_church | CLUSTER |
|---|---|---|---|
| 3 | 1 | FIRST BABIST CHURCH | . |
| 4 | 29 | FIRST BABTIST CHURCH | 1 |
| 5 | 16 | FIRST BABTIST | 1 |
| 6 | 5 | FIRST BABTIST WOODWAY | 1 |
| 7 | 3 | FIRST BABTIST CHURCH OF ALLEN | 1 |
| 8 | 3 | FIRST BABTIST CHURCH WOODWAY | 1 |
| 9 | 1 | FIRST BABTIST ARLINGTON | 1 |
| 10 | 1 | FIRST BABTIST CARROLLTON | 1 |
| | | | |
| 174 | 54 | BRENTWOOD BAPTIST CHURCH | 13 |
| 175 | 30 | BRENTWOOD BAPTIST | 13 |
| 176 | 2 | BRENTWOOD BABTIST CHURCH | 13 |
| 177 | 1 | BRENTWOOD CHURCH OF CHRIST | 14 |
| 178 | 1 | BRENTWOOD CHURCH OF THE NAZARE | 14 |

While this sensitivity is not perfect, it provides an acceptable level of accuracy and a good starting place for our manual adjustments to the scheme. Once the scheme dataset was created, we used the EXPORT procedure to export the data to an Excel file so that it could be easily edited by staff members who did not use SAS. If your organization has established a naming convention, you may want to assign a standard other than the one picked by Data Quality Server. We decided to use the abbreviation FBC. This was easily accomplished by using a find and replace in Excel. Once these changes and other corrections were completed, the Excel file was converted back to a SAS dataset using the IMPORT procedure. If you use this technique to create the final scheme it is important that you specify a label for your imported dataset that is identical to the label created on the original scheme dataset by the DQSCHEME procedure. If you do not restore the label correctly, this metadata will be lost and the dataset may not be recognized when you get ready to apply the scheme to your data.

The final step in the standardization process is to apply the scheme that you have created. The code shown below is an example of how the DQSCHEME procedure is used to apply the scheme to the original data:

```
proc dqscheme nobfd data=dirty.churches out=cleanchurches;
    apply scheme=std_church  var= orig_church;
run;
```

A new dataset will be created called cleanchurches. However, the value of the field orig_church will be replaced according to the standards that we created earlier in our scheme. As a caveat, there are some things occurring in the data that are too similar to be standardized with the scheme alone. One example of this is the cities of Redmond and Edmond. These are easily mistaken in large volumes of data. They would have to be taken out of the scheme and hard coded into the cleanup program based upon state, zip code or other additional information. In order to take into account erroneous variations of these two names, all variations could be standardized as Edmond and later overridden in the program code based on the additional information.

The code below is a sample of how logic is used to override a scheme.

```
/* Use DQSchemeApply function and custom code to override schemes.   */
data cleanchurches;
set dirty.churches;

/*Apply church scheme.  Note: New variable created.*/
std_church = dqSchemeApply(orig_church, 'dirty.bu_std_church', 'nobfd');

/*Apply city scheme.  Note: New variable created.*/
std_city = dqSchemeApply(orig_city, 'dirty.bu_std_city', 'nobfd');

/*Override values in city of WOODWAY */
IF STD_CHURCH='FBC' AND STD_CITY='WOODWAY' AND STATE='TX' THEN DO;
      STD_CHURCH='FBC WOODWAY';
      STD_CITY='WACO';
END;
run;
```

This time the data set cleanchurches is created using a traditional DATA step and the scheme is applied using the DQSchemeApply function instead of the DQScheme procedure. This allows us to do all the work in a single step and eliminates the need for creating an interim data set. Another noteworthy difference is that this creates a new field containing the standardized value instead of overwriting the original value as the DQScheme procedure did. This is desirable because it allows you to preserve your original data for comparison with the standard.

## DUPLICATES THAT AREN'T IDENTICAL

As a private university, Baylor depends almost totally upon tuition and fund raising for its financial existence. Consequently, it is very important for Baylor to maintain a good relationship with prospective students, alumni, donors, and churches. Duplicate records create a number of problems in maintaining good relations. Duplicates often go undetected due to errors in the name or address. No one likes to receive multiple copies of the same piece of mail. This not only makes the University appear unprofessional, it is also costly. The most obvious cost is the expense of printing and mailing the second copy of the communication. There are also indirect costs related to lost opportunity for those who did not receive the communication. It is cost prohibitive to mail to every prospective student.

Therefore, a specific number of targeted prospects receive a communication based on the amount budgeted for a specific campaign. Every duplicate represents a prospect that did not receive the communication. At some point, a number of missed prospects translate into a student who does not enroll which, in turn translates into lost revenue. Another serious problem occurs when duplicate records belong to a person who is applying for admissions. In this case, there is the potential for the application to be entered for one record and important documents such as transcripts and test scores entered into the duplicate record. If this happens, an applicant who completed all the steps for admission would not be considered for acceptance because neither record would appear to be complete in the records system.

The scenario presented in this workshop is a small data set that contains only names and addresses for a group of contacts. These records are representative of the problems commonly seen in the personal data that we process. The entire name is contained in one field. The order of the first and last names is not consistent. Some names contain prefixes and/or suffixes. Nicknames are used and some names are misspelled. Some addresses contain abbreviations while others do not. In this scenario, we need to separate out the various elements of the name. We also need to determine the gender of the person if possible. Finally, we need to identify records that are potentially duplicate records so they can be corrected or consolidated.

We will start by creating the fields that we need for our data requirements and for 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 data set 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.

```
data Dup_Analysis;
  set dirty.dupdata;
  ID=_N_+900000; *Create a unique ID for each record;
```

Since our list contains only a full name field, we will need to use the Data Quality Server parsing functions to create the separate fields as discussed below. If some of the names in your data contain prefixes such as Mrs. or Dr. as ours does, it can be a challenge to identify and separate the components of the name. The DQPARSE function is a powerful feature that can be used to extract the individual elements from a name field. 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.

```
** Use Data Quality Server functions to parse name **;
  parsename=dqparse(full_name, 'name', 'ENUSA');
```

The output shown below demonstrates names delimited by the DQPARSE function.

| Obs | FULL_NAME | ID | parsename |
|---|---|---|---|
| 1 | Smith Thomas | 900001 | /=/Thomas/=//=/Smith/=//=/ |
| 2 | Thomas Smith | 900002 | /=/Thomas/=//=/Smith/=//=/ |
| 15 | Mr. Christopher McDougal | 900015 | Mr./=/Christopher/=//=/McDougal/=//=/ |
| 16 | Christopher Mc Dougal Jr | 900016 | /=/Christopher/=//=/Mc Dougal/=/Jr/=/ |
| 17 | Ryan Dawson | 900017 | /=/Ryan/=//=/Dawson/=//=/ |
| 18 | Dawson, C. Ryan | 900018 | /=/C./=/Ryan/=/Dawson/=//=/ |
| 19 | Brandon Ledbetter | 900019 | /=/Brandon/=//=/Ledbetter/=//=/ |
| 20 | Brandon Ledbtt Er | 900020 | /=/Brandon/=/Ledbtt/=/Er/=//=/ |
| 62 | Dr. Walt Wagner Sr., EdD | 900062 | Dr./=/Walt/=//=/Wagner/=/Sr./=/EdD |

Notice how the function reordered the name in the parsename field. The name definition supplied as the second argument to the function determines the number of elements and their position within the parsename field. The code below is used to assign the values we need for first, middle and last name.

```
** Separate the name into its components **;
  fname=dqparseTokenGet(parsename,'Given Name', 'name', 'ENUSA');
  mname=dqparseTokenGet(parsename,'Middle Name', 'name', 'ENUSA');
  lname=dqparseTokenGet(parsename,'Family Name', 'name', 'ENUSA');
```

Next we will attempt to determine the gender for each person based on the full name using the DQGENDER function.

```
** Create a calculated gender field **;
  gender=dqgender(full_name, 'gender', 'ENUSA');
```

The results of this step are shown below.

| Obs | FULL_NAME | ID | parsename | gender |
|---|---|---|---|---|
| 3 | C. Luck | 900003 | /=/C./=//=/Luck/=//=/ | U |
| 4 | C. Matthew Luck | 900004 | /=/C./=/Matthew/=/Luck/=//=/ | M |
| 7 | Lee Fanning | 900007 | /=/Lee/=//=/Fanning/=//=/ | U |
| 8 | Lee Ellen Fanning | 900008 | /=/Lee/=/Ellen/=/Fanning/=//=/ | F |
| 11 | Eli Beth Eizner | 900011 | /=/Eli/=/Beth/=/Eizner/=//=/ | U |
| 12 | Elizabeth Eizner | 900012 | /=/Elizabeth/=//=/Eizner/=//=/ | F |
| 48 | Jesse Merrimann | 900048 | /=/Jesse/=//=/Merrimann/=//=/ | U |
| 49 | Jesse Herrmann | 900049 | /=/Jesse/=//=/Herrmann/=//=/ | U |

Notice that in rows 4 and 8 the middle name was beneficial in determining the gender but in row 11 creates an ambiguity and returns a value of U for unknown gender. Some names such as Jesse are common for both males and females. Therefore, in this case the unknown gender is also assigned.

One of the common problems in contact names occurs when a person who prefers to go by his middle name supplies one record using his legal name and another using his preferred name. 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 in the code below by concatenating the assumed preferred name with the last name.

```
** 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);
```

In the workshop data only 5 digit zip codes are used. However, in real data many zip codes are in the zip+4 format. The step below would be needed to insure that all zip codes are limited to the first 5 digits. The Data Quality Server includes a match definition for zip code which did not seem necessary to meet our objectives. We do not use city names to test for matches because of the problems with city names that we discussed when creating the standardization scheme. We believe that the zip code is more reliable. However, if you do not have confidence in your zip codes, you could create an additional match code for the city name and use that in conjunction with name and address match codes.

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

The DQMATCH function in Data Quality Server is the workhorse that we will use to identify duplicate records that are not identical. 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_name60=dqmatch(full_name, 'name', 60, 'ENUSA');
```

In this example, MC_name60 is the variable that will contain the match codes returned by the DQMATCH function.

The variable full_name 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 full_name 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. As in the DQSCHEME procedure 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 our matching process, 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.

| | Full_name | MC_name60 | 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 explanations that follow go into more depth and detail than we will have time to cover in the workshop. Therefore, the examples provided come from our original research rather than from the sample data that is provided for the workshop. 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.

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 again 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; *Join any records from the last cleanup with the current list;
   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 ENHANCED MATCHING PROCESS

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 code below shows all of the matches that were created and how they are combined to detect the maximum number of suspected duplicate records.

```
** Match codes for preferred full name **;
  MC_pref=dqmatch(prefname, 'name', 60, 'ENUSA');
** Match codes for full name **;
  MC_name60=dqmatch(full_name, 'name', 60, 'ENUSA');
  MC_name95=dqmatch(full_name, 'name', 95, 'ENUSA');
** Match code for first name **;
  MC_fname60=dqmatch(fname, 'name', 60, 'ENUSA');
** Match codes for street address **;
  MC_addr60=dqMatch(street,'ADDRESS', 60 ,'ENUSA');
  MC_addr50=dqMatch(street,'ADDRESS', 50 ,'ENUSA');
** Text Match codes for street address **;
  MC_addr_txt70=dqMatch(street,'TEXT', 70 ,'ENUSA');
  MC_addr_txt50=dqMatch(street,'TEXT', 50 ,'ENUSA');
run;
```

For the sake of brevity, the first part of the data step is not shown. The code above would follow the code described earlier to parse the name, supply a unique ID and standardize the zip code. This example assumes that the name of the output data set is MC_Contacts. The code below shows how a number of data sets are created using various combinations of match codes that target the various problems defined earlier in the paper.

```
/* Group 1 - Box addresses with full name and zip for more distinction*/
proc sql;
   create table match_pob as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) in ('NMX', 'NMZ')
   group by MC_name60, MC_addr_txt70, ZIP
   having count(*) ge 2
   order by  MC_addr_txt70, zip;
```

```
quit;

/* Group 2 - Box addresses with first name (marriages & distorted last
names)*/
proc sql;
   create table match_pob_fname as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) in ('NMX', 'NMZ')
   group by MC_fname60, MC_addr_txt70, ZIP
   having count(*) ge 2
   order by MC_addr_txt70, zip;
quit;

/* Group 3 - Box addresses with preferred name for people who go by middle
name */
proc sql;
   create table match_pob_pref as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) in ('NMX', 'NMZ')
   group by MC_pref, MC_addr_txt70, ZIP
   having count(*) ge 2
   order by  MC_addr_txt70, zip;
quit;

/* Group 4 - Street addresses with full name (basic match) */
proc sql;
   create table match_addr as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) not in ('NMX', 'NMZ')
   group by MC_name60, MC_addr60
   having count(*) ge 2
   order by  id;
quit;

/* Group 5 - Street addresses with first name (marriages & distorted last
names)*/
proc sql;
   create table match_addr_fname as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) not in ('NMX', 'NMZ')
   group by MC_fname60, MC_addr60, ZIP
   having count(*) ge 2
   order by id;
quit;

/* Group 6 - Sensitive name & zip to catch sloppy street addresses */
proc sql;
   create table match_name as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) not in ('NMX', 'NMZ')
   group by MC_name95, MC_addr50, ZIP
   having count(*) ge 2
   order by id;
quit;
```

```
/* Group 7 - Street addresses with preferred name for people who go by middle
name */
proc sql;
   create table match_pref as
   select * from MC_contacts
   where substr(MC_addr_txt70,1,3) not in ('NMX', 'NMZ')
   group by MC_pref, MC_addr50, ZIP
   having count(*) ge 2
   order by id;
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.

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

```
/* Combine all of the groups together in one data set */
data matches2;
set match_pob match_pob_fname match_addr match_addr_fname match_name
      match_pref match_pob_pref;
run;
```

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.

```
/* Create final list using ID to remove repeated records */
proc sql;
create table cleanup as
select distinct ID, FULL_NAME, STREET, CITY, STATE, ZIP, MC_addr60
from matches2
order by zip, MC_addr60;
quit;
```

## 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 put DQ to work for you.

## REFERENCES

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

## 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