

EFFICIENT COMPARISON OF NONPRECISE CHARACTER DATA

S. David Riba, JADE Tech, Inc.

ABSTRACT

There are many occasions when character values must be compared on non-precise, or "fuzzy", criteria. Traditionally, SAS programming techniques have relied on the SUBSTR function to extract and compare portions of the character values. Unfortunately, this has led to some brute-force programming techniques when it has been necessary to compare character data on approximate or nonprecise criteria. This article examines several other tools in the SAS System which can be useful when comparing "fuzzy" character data.

INTRODUCTION

A "fuzzy" character comparison is what might be needed when you are uncertain of the spelling or contents of character variables in your data. Have you ever considered how many different ways there are to (mis)spell some names? "Fuzzy" character searches allow your SAS program to find matches even when there are misspellings or typographical errors in the data.

The typical way most SAS programs deal with character data is with the SUBSTR function:

```
if (SUBSTR(charvar,1,x) eq '--') then ...
```

This is acceptable for an exact match. However, most character data is never that clean. There are typographical errors, upper- and lowercase sensitivity problems, extra or missing spaces, and so on. How do you process the exceptions? Multiple nested IF statements? There has to be a better way!

In fact, there are several ways to deal with the situation in SAS software.

FUNCTIONS AND OPERATORS FOR CHARACTER MATCHING

First, eliminate the obvious types of problems. How many times have you written code that looks like

```
if ((charvar eq 'Y') or (charvar eq 'y')) ...
```

In SAS, uppercase characters and lowercase characters compare as different character values.

There are two SAS functions that can be useful to assure the accurate comparison of character data regardless of the case of the characters. The UPCASE function converts ALL character data to uppercase before making a comparison. The LOWCASE function converts ALL character data to lowercase before making a comparison. Thus, you can code the above as either

```
if (UPCASE(charvar) eq 'Y') ...  
or  
if (LOWCASE(charvar) eq 'y') ...
```

It is important to note that both the UPCASE function and the LOWCASE function do not actually change the contents of the variable, but they merely do a comparison based on the upper- or lowercase values of the data. All examples in this article use the UPCASE function when equalizing for case, but either UPCASE or LOWCASE could have been used throughout.

The SUBSTR function can now compare like values. Because functions can be nested, the above example could be restated as

```
if (UPCASE(SUBSTR(charvar,1,x))  
    eq '-----') then ....
```

However, this still does not resolve the problem of nonprecise or "fuzzy" character comparisons. Have you ever seen this comparison operator in SAS?

= :

Probably not, although this operator has been around in the SAS System for a long time. The colon after the equal sign changes the operator from an equality operator to an operator that compares all values that start with a given value. For example, contrast the following

```
if (charvar = 'P') ...
if (charvar =: 'P') ...
```

The first example will find a match for every observation where the variable CHARVAR is equal to the character P. The second example will find a match for every observation where the variable CHARVAR begins with the character P. With a simple change to the equality operator, the IF test will now succeed for a much broader range of values without the necessity of coding and evaluating the results of a SUBSTR function.

Depending on your data, you can now do searches like

```
if (UPCASE(lastname) =: 'SM') then ...
```

to find all records that begin with the letters SM (Smith, Smythe, Smithfield, Smedley, and so on). There is another advantage to this particular construction over using the SUBSTR function to locate matches on the beginning of character strings. With the SUBSTR function, you must code both the beginning range (a 1 to start at the beginning) and the ending range of the string to extract. With the =: operator, all that is necessary is to enter as much of the string as is necessary to define the unique match. For example,

```
if (UPCASE(lastname) =: 'SMIT') then ...
```

will find all records that begin with the letters SMIT (Smith, Smithfield, and so on), regardless of the case of the character data.

Consider two other SAS functions that are useful in determining character matches. The INDEX and INDEXC functions can indicate if a string of characters is present in a target variable. Both functions return the position number of the match in the target variable. A zero indicates that the search argument is not present in the target variable. This is very useful for testing purposes, such as the following:

```
if (INDEX(charvar,'SM') gt 0) then ...
```

If the character string 'SM' is not found in the target variable, then the result is a zero, causing the IF condition to evaluate as false. Otherwise, the result is a number indicating the position of the first occurrence of 'SM' in the target variable.

The INDEXC function allows multiple arguments and will identify the first occurrence of any of the characters in any of the arguments, but otherwise functions similarly to the INDEX function.

```
if (INDEXC(charvar,'SM',';()=. ') gt 0)
then ...
```

A positive number is returned If the target CHARVAR contains either the letter 'S', the letter 'M', or any punctuation found in the second expression. If the target variable CHARVAR does not contain any matches to either of the expressions, then a zero is returned from the INDEXC function.

Another member of the INDEXx family of character functions is INDEXW, which was introduced in Release 6.07 and documented in Technical Report P-222. The INDEXW function performs exactly the same as the INDEX function, with one significant exception. Like the INDEX function, INDEXW returns the position number of the first match of a character string in a target variable. The difference between INDEX and INDEXW is that INDEXW only returns a match of the target occurs on a word boundary. For example:

```
if (INDEXW(charvar,'SM') gt 0) then ...
```

would only return a non-zero result if the character string 'SM' started at the beginning of a word boundary. Like INDEX, the INDEXW function is case sensitive. Therefore, to identify the position of a character that begins a word boundary with the letters 'SM', regardless of case:

```
if (INDEXW(UPCASE(charvar),'SM') gt 0)
then ...
```

WHERE STATEMENT OPERATORS

The WHERE statement was added to the SAS System in Version 6. The WHERE statement can be used in both PROC steps and DATA steps and can contain SAS functions as part of the statement. Thus, any of the functions and operators discussed above can be used with a WHERE statement in either a DATA or PROC step to extract or compare character information.

In addition, the WHERE statement has three operators that can be used for performing character comparisons. These are the CONTAINS, LIKE, and Sounds-like operators.

The first operator is CONTAINS or ?. This operator tests if a target variable includes a particular character string.

For example

```
WHERE (word ? 'LAM' ) ;
```

will find all values of the variable WORD that contain the upper case letters "LAM". In this example,

```
WHERE (UPCASE(word) ? 'LAM' ) ;
```

will match all values of the variable WORD that contain either an upper-, lower-, or mixed case "LAM" including such strings as bLAME, LAMinate, and bedLAM, regardless of the case of the target text.

The LIKE operator can be used for 'wild card' searches. If you have used the * or ? operators in DOS on a personal computer, you are already familiar with 'wild card' searches. With the LIKE operator, character variables can be matched against a pattern of characters and wild cards.

There are two types of pattern matching that are possible with the LIKE operator. A percent sign (%) will replace any number of characters and is equivalent to the DOS *. An underscore (_) will replace only a single character, and is equivalent to the DOS ?.

For example

```
WHERE (name LIKE 'S%' ) ;  
WHERE (name LIKE 'SM_TH' ) ;
```

In the first example, all values of the variable NAME that start with the upper case letter 'S' will be included. In the second example, only those values of the variable NAME that start with the upper case letters 'SM', end with the upper case letters 'TH' , and have a single letter between them will be included. As in the example for CONTAINS, use either the UPCASE or LOWCASE functions to compare text regardless of the case of the values in the character variable.

The final WHERE operator that is useful for selecting character values is the Sounds-like (=*) operator. This operator selects observations based on their phonetic values. It is particularly useful to compare character values phonetically when the likelihood of typographical errors is great. The Sounds-like operator uses the soundex algorithm for phonetic matching. Consider the many different ways to (mis)type the name RALEIGH. A typical mistake might be transposition of characters, such as the IE combination or the GH combination, resulting in RALIEGH or even RALIEHG. Another mistake might be phoneticized spelling, resulting in RALEGH or ROLEIGH. The Sounds-like operator compares character data based on phonetic matches for a character string. For example

```
WHERE (city =* 'RALEIGH' ) ;
```

The Sounds-like operator will find all case matches for the comparison string including Raleigh, RALIEGH, RALIEHG, ROLEHG, RLIEG, or even RLG. All of these values have the same phonetic equivalent.

PHONETIC MATCHING

This brings us to the concept of soundex. The soundex idea has been around for many years, but is relatively new to SAS, having been introduced in Release 6.07. The SAS SOUNDEX function is documented in Technical Report P-222.

The soundex algorithms were first patented by Margaret K. Odell in 1918 and Robert C. Russell in 1922. They are based on an underlying principle of English and other Indo-European languages.

```
Fr scr nd svn yrs ....
```

```
-brhm Lncln
```

The above introduction to the Gettysburg Address is moderately understandable even without the vowels or all the consonants. It is a technique that has long been used by linguists and speed-writers. That is, most English words can reasonably be represented by their consonants alone. It is this technique that forms the basis of soundex phonetic comparisons. All words can be reduced to a phonetic equivalent character string, which can then be compared.

The steps used by soundex to derive the phonetic equivalent of a character string are:

1. Retain the first letter of the character string
2. Discard the letters A E H I O U W Y
3. Assign a numeric value to the following consonants:
 1. B F P V
 2. C G J K Q S X Z
 3. D T
 4. L
 5. M N
 6. R
4. Discard all duplicate classification values if they are adjacent (that is DT will result in a single value of 3, NN will result in a single value of 5).

In addition, the original algorithms stated that the maximum length of the resulting character equivalency be no more than four characters, which the designers felt would represent the sound of a given word and any other words whose pronunciation was fairly similar. Also, words that had a soundex value with less than four characters were to be padded to a length of four for comparison. SAS does not follow either of these particular requirements. The results of the SAS soundex function and the Sounds-like operator (=*) in the WHERE statement are non-truncated and non-padded character strings.

The SOUNDEX function returns the phonetic value of a character string as a character variable based on the above soundex algorithm. Unlike other character assignments in SAS, the length of a new character variable that stores the results of the SOUNDEX function is not defined based on the function results. Instead, it is assigned a length identical to the argument for the SOUNDEX function. For example

```
value = SOUNDEX('RALEIGH') ;
```

The variable VALUE has a character value of 'R42' and a length of 7, the same length as the characters 'RALEIGH'. Subsequent assignments are treated by SAS the same as any other character assignments. A shorter value is stored without charge, while a longer value is truncated to the defined length. To avoid any unexpected

results, it is always prudent to use the LENGTH statement to specify the length of character variables that will contain the results of a soundex evaluation.

In the case of the city of Raleigh, the value returned is R42. Each of the phonetic (mis) spellings above also will return a value of R42, allowing for a comparison based on phonetic equivalents. Some other examples of commonly misspelled character names:

```
cityval = SOUNDEX('Philadelphia') ;  
stateval = SOUNDEX('Pennsylvania') ;
```

The value for CITYVAL is P4341 and the value of stateval is P52415. The results are the same whether the target character strings are defined as upper or lower case.

The SOUNDEX function allows selection of information based on their phonetic equivalents. It will often pick up additional values that are similar to the target string, but will rarely miss values that it should find. As a hashing algorithm, the original concept of a four character evaluation string produces at most 8,918 unique pronunciation groups.

The major drawback to using SOUNDEX is that the first letter must be an exact match. In the case of PHILADELPHIA, for example, SOUNDEX would not have found any records spelled as FILADELFIA. However, since SOUNDEX returns a character string it would be easy to compare based on all characters except the first character. The SOUNDEX value of PHILADELPHIA is P4341, while FILADELFIA has a value of F4341. This drawback to soundex can be eliminated by using either the SUBSTR function or the LIKE operator with the _ wildcard in a WHERE statement. To compare on the text string PHILADELPHIA:

```
WHERE (SOUNDEX(cityname) LIKE '_4341') ;
```

OR

```
WHERE (SUBSTR(SOUNDEX(cityname),2) =:  
      '4341') ;
```

Since the above examples are part of a WHERE statement, they can be placed in either DATA or PROC steps. Thus, if these statements were part of a PROC PRINT or a PROC REPORT, for

example, the resulting report would consist only of those records that phonetically compared to the city of Philadelphia regardless of the spelling of the city name.

CONCLUSION

Depending on the data involved and the types of comparisons that are needed, the SAS System contains a variety of options besides the SUBSTR function which are particularly useful for non-precise or 'fuzzy' character comparisons. With an understanding of these functions as well as several of the useful operators available with the WHERE statement, nonprecise character data can be evaluated more effectively in both DATA and PROC steps.

AUTHOR CONTACT:

S. David Riba
JADE Tech, Inc.
P. O. Box 4517
Clearwater, FL 34618
(727) 726-6099
dave@jadetek.com

AUTHOR BIO:

S. David Riba is CEO of JADE Tech, Inc., a SAS Institute Quality Partner who specializes entirely in applications development and training in the SAS® System.

Dave is the founder and President of the Florida Gulf Coast SAS Users Group. He chartered and served as Co-Chair of the first SouthEast SAS Users Group conference, SESUG '93, and serves on the Executive Board of the SouthEast SAS Users Group. He has been a SAS user for 14 years, and has been actively involved in both SUGI and the Regional SAS Users Group community since then. He has presented papers on SAS-related topics at SUGI, SESUG, NESUG, MWSUG, and SCSUG. His major areas of interest are efficient programming techniques and applications development using the SAS® System, particularly using Screen Control Language with SAS/AF® and FRAME technology.

