Issue Date: FoxTalk May 1999

Matchmaker, Mitchmoker... Is That a Match? Or De-mystifying De-duplication

Andrew Coates
a.coates@civilsolutions.com

Finding duplicate entries in your data or matching records from different tables is a problem that has plagued database designers since databases began (and even earlier). In this article, Andrew demonstrates some of the techniques you can use to match records.

You know the story -- two of the departments in the company have each maintained records about their customers, and now the IT manager has decided that the databases need to be merged. The only problem is, how do you decide whether a record from the first database matches one from the other? In this article, I'll explore some options for matching these records (or for detecting duplicates in a data set, which is essentially the same thing). I'll discuss some concepts about what you're trying to achieve, talk about what you can match, and delve briefly into name and address standardization. Next, I'll talk about automating the matching process and a couple of algorithms you can use for matching the sounds, not the spelling, of words. By the end of this article, you should at least be able to plan a matching strategy that fits your needs best.

Matching/de-duping

Matching and de-duplication are essentially the same process -- that of finding records that refer to the same entity, either in two (or more) separate tables, which I call matching, or in one table, which I call de-duplication. If you're working with a single table, then you've probably at least got the advantage of having the information you're examining in the same format for each side of the comparison. When you're matching, you often need to do some pre-processing to get the information from the first table into the same format as the information in the second table. On occasion, it's also useful to do some standardization before attempting to de-duplicate.

Name and address standardization

How do you do this pre-processing? Well, to make sure you're comparing apples to apples, you need to carry out some form of standardization. To standardize names, you might need to break apart names that are stored as one string into their component parts. For example, the single field ContactName containing "Mr Andrew C Coates BE" can be broken down into the following:

ContactTitle	"Mr"
ContactFirstname	"Andrew"
ContactInitials	"C"
ContactSurname	"Coates"
ContactPostNom	"BE"

Doing this will allow you to much more easily find a match with "A.C. Coates," which you'd standardize as follows:

ContactTitle	""
ContactFirstname	"A"
ContactInitials	"C"
ContactSurname	"Coates"
ContactPostNom	""

Break data down into the smallest units you have. For example, split names into their components, and addresses into number, street name, city, state, and postal code.

Use standard abbreviations for common components. For example, the USPS has a comprehensive list of street types ("Street," "Lane," and so forth), common variations ("St," "Str," "Ln," "Lne," and so on), and their preferred abbreviation. Use either the standard full title or the standard abbreviation.

Time spent on standardization is rarely wasted. Extract a set of data from your target to develop your standardization algorithm. Extract another (completely separate) set of data from your target to test the algorithm after you're satisfied with the algorithm. Having this second set will often reveal exceptions or expose inaccuracies that you might not have considered. It's important to have this verification set of data as well as a development set.

For anything but the most trivial data sets, standardization is a computationally intensive task. Be prepared for the standardization processing to take a significant amount of time. Standardization routines I've developed have turned into 100+ case statements and have taken in the order of one quarter of a second per record to process. While this isn't a lot if you're

only processing a thousand or so records (although there's still time to make a cup of coffee), 250,000 records will take more than 17 hours.

As a final standardization remark, I should note that there are commercially available standardization packages, as well as mailing houses that will take your data and return it appropriately compartmentalized. You might consider using these services if you have a single job and can't justify developing your own routine or if you have so much data that you just don't have the resources to handle it.

What to match

Once you've got your data in a standard format, you need to decide what constitutes a match. Is the A. Coates living in Sydney the same as the Anthony Coats from Maroubra in New South Wales? The answer to that question is "possibly." What the probability is of a match is something you need to determine.

If you have a match on a unique identifying number like the U.S. social security number, then the probability of a match is quite high. If you don't have the "smoking gun," then you need to use other means to make the case for a match or otherwise. Possible fields for matching are:

- Address (but don't forget that families often live in the same house and have pretty similar names)
- Date of birth/age
- Name
- Geographical location
- Company name
- Phone numbers

Phonetic matching

One common matching option is to compare the phonetic values of strings such as people's names or street names. There are several algorithms available that assign a value to a string based on how it sounds. Using these techniques, you'll be able to find duplicates that might not be spelled exactly the same way, such as "Smith" and "Smythe." The algorithms I've used to a greater or lesser extent are SOUNDEX and NYSIIS. Each of these has its pros and cons. FoxPro also includes an algorithm called DIFFERENCE() that compares two strings. Another algorithm I found while researching this article is called Metaphone. I've never used it in anger, and I've not been able to find the source for it, but it appears to produce a code that's similar, but not identical, to the NYSIIS algorithm.

Phonetic algorithms basically work by suppressing the vowel information (because it's unreliable) and giving the same code to letters or groups of letters that sound the same (for example, "PH" sounds like "F," so you give them both the same code). You can use the code generated to find matching names in a table; for example, the following will display a browse window with all records with surnames including SMITH, SMYTHE, SCHMIDT, SMYTH, and SCHMIT:

```
BROWSE FOR SOUNDEX(surname) = SOUNDEX("SMITH")
```

SOUNDEX

SOUNDEX is a phonetic coding algorithm that ignores many of the unreliable components of names, but by doing so reports more matches. The rules for coding a name are (from Newcombe):

1. The first letter of the name is used in its un-coded form to serve as the prefix character of the code. (The rest of the code is numerical.)

2. Thereafter, W and H are ignored entirely.

3. A, E, I, O, U, and Y aren't assigned a code number, but they do serve as "separators" (see Step 5).

4. Other letters of the name are converted to a numerical equivalent:

1	B, P, F, V
3	D, T
4	L
5	M, N
6	R
2	All other consonants (C, G, J, K, Q, S, X, Z)

5. There are two exceptions: a) letters that follow prefix letters that would, if coded, have the same numerical code, are ignored in all cases unless a "separator" (see Step 3) precedes them; and b) the second letter of any pair of consonants having the same code number is likewise ignored (unless there's a "separator" between them in the name).

6. The final SOUNDEX code consists of the prefix letter plus three numerical characters. Longer codes are truncated to this length, and shorter codes are extended to it by adding zeros.

Examples of names with the same SOUNDEX code are shown in **Table 1**.

Table 1. Names with the same SOUNDEX code.

Name	Code
ANDERSON, ANDERSEN	A 536
BERGMANS, BRIGHAM	B 625
BIRK, BERQUE, BIRCK	B 620
FISHER, FISCHER	F 260
LAVOIE, LEVOY	L 100
LLWELLYN	L 450

SOUNDEX has an immediate attraction for FoxPro (and SQL Server) developers. It's implemented as a native language function. Issuing the following from the command window will display the code S530 -- no further programming is necessary:

```
? SOUNDEX('SCHMIDT')
```

Compare this with the monstrosity that's NYSIIS.PRG (see the accompanying **Download file**). Having the function built-in also has significant speed advantages.

NYSIIS

NYSIIS differs from SOUNDEX in that it retains information about the position of vowels in the encoded word by converting all vowels to the letter A. It also returns a purely alpha code (no numeric components). NYSIIS isn't part of the native FoxPro command set, nor does it seem to have been implemented by any third-party utility developers. I've coded the algorithm in the FoxPro procedure shown in and included in the accompanying **Download file**, but the high-level pseudo-code for this algorithm is shown in **Listing 1**.

Listing 1. High-level pseudo-code for the NYSIIS algorithm.

```
* Program....: NYSIIS.PRG
* Version....: 1.0
* Author.....: Andrew Coates
* Date.......: March 1, 1999
* Notice.....:
* Compiler...: Visual FoxPro 6 for Windows
* Abstract...: NYSIIS phonetic encoding algorithm
* Taken from Newcombe 1988 pp182-183
* rule number and lettering as per Newcombe
* Changes....:
*!* 1. Change the first letter(s) of the name
*!* 2. Change the last letter(s) of the name
*!* 3. First character of the NYSIIS code is the
*!*    first character of the name
*!* 4. Set the pointer to the second letter
*!*    of the name
*!* 5. Change the current letter(s) of the name
*!* 6. Add a letter to the code
*!* 7. Change the last character of the NYSIIS code
*!* 8. Change the first character of the NYSIIS code
```

NYSIIS has a disadvantage in our rapidly shrinking, multicultural world of being fairly Anglo in its phonetic coding. If the names you're matching have non-Anglo origins, it would probably be better to use a different algorithm -- for example, SOUNDEX.

Phonetic matching example

To demonstrate the use of phonetic matching, I've used the customer table in the testdata database in the sample data that comes with VFP. The code is shown in **Listing 2**.

Listing 2. Phonetic matching example.

```
* Program....: PHONETICS.PRG
* Version....: 1.0
* Author.....: Andrew Coates
* Date.......: March 1, 1999
* Notice.....:
* Compiler...: Visual FoxPro 6
* Abstract...: Extracts surnames from
*  the testdata!customer table and does
*  phonetic comparisons.
*  NB - assumes that NYSIIS.PRG is in the path
* Changes....:
clos data all
open data (home(2) + 'data\testdata.dbc')
* break the contacts' names apart
select cust_id, ;
```

```
  padr(substr(contact, at(' ',contact) + 1), 30) ;
  as Surname ;
  from customer ;
  into cursor names

* get the codes for each contact
select *, ;
  nysiis(surname) as NYSIIS, ;
  soundex(surname) as SNDX ;
  from names ;
  into cursor codes
* get a list of all the nysiis codes that appear
* more than once
select nysiis, count(*) as tot ;
  from codes ;
  group by nysiis ;
  having tot > 1 ;
  into cursor multinysiis

* get a list of all the soundex codes that appear
* more than once
select sndx, count(*) as tot ;
  from codes ;
  group by sndx ;
  having tot > 1 ;
  into cursor multisndx

* generate a list of customers with phonetically
* matching surnames
select names.surname, codes.sndx ;
  from names ;
  inner join codes on ;
    names.cust_id  = codes.cust_id ;
  inner join multisndx on ;
    codes.sndx = multisndx.sndx ;
  order by codes.sndx ;
  into cursor sndx
select names.surname, codes.nysiis ;
  from names ;
  inner join codes on ;
    names.cust_id  = codes.cust_id ;
  inner join multinysiis on ;
    codes.nysiis = multinysiis.nysiis ;
  order by codes.nysiis ;
  into cursor nysiis
```

First, I extract the surnames from the customer name field using the assumption that the surname starts immediately after the first space character in the name field. Note that this assumption isn't valid for all cases -- like the names "José Pedro Freyre" and "Isabel de Castro" -- but I've ignored that problem here.

Next, I calculate the NYSIIS and SOUNDEX codes for the surnames and store them with the customer ID. Then I find all of the codes that appear more than once (potential matches), and finally I generate a cursor with the surname and code for each potential match. The results for SOUNDEX and NYSIIS are shown in **Table 2** and **Table 3**, respectively.

Table 2. Potential SOUNDEX matches from the customer table.

Surname	SOUNDEX
Ashworth	A263
Accorti	A263
Berglund	B624
Bergulfsen	B624
Crowther	C636
Cartrain	C636
Moreno	M650
Moroni	M650
Pereira	P660
Perrier	P660
Wilson	W425

Wang	W520
Wong	W520

Table 3. Potential NYSIIS matches from the customer table.

Surname	NYSIIS
Moreno	MARAN
Moroni	MARAN
Pereira	PARAR
Perrier	PARAR
Wilson	WALSAN
Wang	WANG
Wong	WANG

By comparing Tables 2 and 3, you'll notice that SOUNDEX seems to match more names than does NYSIIS. The additional matches generated by SOUNDEX in this case don't seem to be good matches, but that's not always the case. You need to assess your data (perhaps by pulling a sample of 100 or so matches and calculating the hit rate).

Automating matching

It's possible to assign a value based on a match on some or all of the fields in your table and then use the "matching rank" to automatically determine whether records match. You could set up a system like the one shown in **Table 4**.

Table 4. Sample "matching rank" system.

Field	Match	Rank
Address	All fields identical	100
	Street name SOUNDEX match and Suburb and State match	80
	Suburb and State match	60
	Address 1 within 50 km of address 2	30
	Country doesn't match	-20
	Any other address configuration	0
Name	All Fields identical	100
	Surname identical, first letter of first name matches	50
	Surname SOUNDEX matches, first name matches	40
	No match on any field	-100
Date of Birth	Identical	80
	Any 1 component (day, month, or year) +/-1	30
	Within 18 months	10
	Difference between 5 and 10 years	-30
	Difference > 10 years	-80
	Any date of birth configuration	0
Gender	Matches	0
	No Match	-100

Using this system, you could calculate a matching rank for each record compared with each other record. You could set a threshold value -- say, 200 -- above which you're sure that you've got a match, and another -- say, 100 -- below which you're sure you haven't (the actual numbers you use will depend on your data). The question is what to do with the middle range. Generally, they have to be reviewed by a human.

Another problem you might come across with this system is multiple possible matches. You need to decide how to handle these. Perhaps present the top n possible matches, or perhaps present all of them. You could decide to present any match with a rank of at least 50 percent of the highest possible match. What happens if you have a definite match and a possible match? In this case, there's a possibility that there's a duplicate in the table you're matching. It's worth de-duplicating before you match to try to reduce this problem as much as possible.

You can write a program to automate the matching process. **Listing 3** shows some pseudo-code for such a program.

Listing 3. Pseudo-code for automatic matching process.

```
Open Tables
For each record in table1
  For each record in table2
    Get matching rank for this record combination
    Do case
    Case matching rank < lower threshold
      No match, just go to next record
    Case matching rank > upper threshold
      Definite match - write IDs to matched record
      table
    Otherwise
      Possible match - write IDs and rank to possible
      match table
    End case
  End for && table2
End for && table1
Deal with the possibility of multiple possible matches
```

Avoiding duplicates during data entry

Prevention is better than cure, and if you have control of the data capture phase of your operation, you can apply the matching algorithms suggested previously to the data as it's being keyed. Tell the user if you think they're entering duplicate data, and you'll eliminate the need for costly and time-consuming de-duplication later.

Conclusion

This month, I've probably presented more questions than answers, but that's often the way when describing a fuzzy operation such as matching. I hope that I've been able to point out some of the things you can do to find matching records and avoid duplication in your data sets. Matching is an art, and one you need to practice to perfect. Criteria for determining matches change depending on the data set, and you need to tweak your matching processes accordingly.

Next month, I'll deviate a little from the Data Bus concept and introduce a "cool tool" you can use for sending messages between applications or instances of the same application across a TCP/IP network. I'll show you how to build a simple license manager and a chat server.

References

I got the SOUNDEX and NYSIIS algorithms and several other concepts from the *Handbook of Record Linkage*, Howard B. Newcombe, Oxford University Press, 1988.