Paper 143

Generating and Using Name Keys for Fuzzy Matches: Calling a Third Party Dynamic Link Library as a Module in the SAS® System

Richard Carson, Work and Income NZ, Wellington, New Zealand

ABSTRACT

Identifying duplicate customer records or tying together customer records from different sources is done most efficiently if names can be "fuzzy" matched against each other. While SAS implements the SOUNDEX algorithm it also allows use of external DLLs via the CALL statement, so specialist third party routines can be used, such as Search Software America's (SSA) specialist search key technology. In a trial the SSA DLL was successfully used to generate search keys in a SAS database of 160,000 names. The integration was complex and a number of implementation issues are raised in this paper.

INTRODUCTION

BUSINESS CONTEXT

The National Debt Management Unit of Work and Income New Zealand, contrary to its name, manages a national asset which is the money owed to the Government by former income support customers and others who have debts arising out of past social obligations. The total amount of debt owing is over $NZ 700 million, with most of the recent debt currently under collection. However, a sizeable part of the older debt is not currently collectable because we have lost contact with the debtor. Debtor location is a significant issue and we are always looking for tools that will increase our ability to identify and locate non paying debtors.

We also have a problem with cross referencing people's records in some legacy databases with others which conform to current and different business rules for customer identification. While having multiple entries for a customer can be a nuisance for organisations with mailing lists it can be a major embarrassment for collecting debt, especially if the debtor is thanked for clearing the debt only to find subsequently that other debt is still outstanding!

CUSTOMER IDENTIFICATION BY NAME

The problem of managing customer identification is similar to finding a "unique key" for accessing records in a database. In the distant future we may have some way of encoding DNA which will provide an inherent and unarguably unique "key" for everyone (except identical siblings and clones and robots...), but such an arrangement is neither feasible nor acceptable now, and nor are widespread fingerprinting or forehead tattooing. In the United States, a bastion of civil liberties, citizens are expected to have and use a social security number which uniquely identifies them, and this number is used as a general purpose identifier in other contexts such as employment and taxation. This of course just means that the problems of identity and avoidance of duplication have to be dealt with by the Department of Social Security on behalf of other users of the social security number. In New Zealand we have legislated against this approach.

In the absence of a unique identifier we have to rely on a range of identifiers such as name, date of birth and place of birth, but even a combination of these need not be unique. I once knew a John Smith, born in London. It's quite possible he was one of several John Smiths born in London on the same day. Even a slightly less common name like mine can recur, as I found when I went to an internet site to see if I was listed (I wasn't, but several other people named Richard Carson were). Another example: we changed banks because our previous bank consistently confused my wife with another Wellingtonian of the same name.

Often all we have to go on is name and current address but this is certainly not unique. Earlier last year there were three people named Richard Carson living at my home address. So far, the examples that have been given all illustrate the dangers falsely concluding that two different entities are the same; but the reverse problem of failing to match up two different instances of the same identity can cause further problems. Where people and place names are involved comparisons have to be "fuzzy" to allow for spelling variations, misspelling, word order, and truncation or omission. In addition, names are often replaced by dissimilar nicknames which make comparisons even more hazardous.

TOOLS FOR NAME COMPARISONS

SOUNDEX

In SAS and other similar applications the SOUNDEX function is used for fuzzy matching by encoding names to minimise the effect of spelling variations and vowel shifts. The SOUNDEX function was first patented in 1918, and was implemented in SAS from release 6.07. It is documented in SAS Technical Report P-222 *Changes and Enhancements to Base SAS Software*, pages 64 and 65. The function is also used in the LIKE operator in a WHERE statement:

```
       where name like "Smith" ;
```

The SOUNDEX function has stood the test of time for European style names but it is relatively weak with Maori, Pacific Island and some Asian names where vowels are most important components of the name word. For instance

```
       soundex (Tawa)         returns: T
       soundex (Ohaewae)      returns: O
  and  soundex (Waharoa) = soundex (Waiwera)
                         = "W6".
```

(These are all Maori Place names in New Zealand.)

In addition to vowels SOUNDEX strips out all but initial occurrences of W and H - both significant place holders in Maori names - but with the WH combination representing a consonant phonetically close to F. So Kowhai and Kaio are both represented solely by K.

As an example of how culturally dependent phonetic associations can be I came across a Fijian who regarded a name beginning with "Fifi" as being phonetically equivalent to a name beginning with "Sisi".

KEY GENERATION

The returned value of the SOUNDEX function is a "key" that can be used to index and match names for further testing (eg on date of birth). The efficiency of such a key can be judged by the amount to which it narrows down the search for matching names (excludes non matches) while not overlooking any (fails to include matches). While SOUNDEX may be a useful general purpose key it is not very efficient with a large number of New Zealand person and place names.

While investigating this problem I found more efficient keys are available and one supplier, Search Software America (SSA), [http://www.searchsoftware.com], could make their key generation software available as a dynamic link library (DLL) for the MicroSoft Windows operating environment. Software in this format can be accessed by other Windows applications and SAS is no exception, so I was able to trial the key generation and assess it.

SSA KEYS

The SSA approach to key generation differs in several ways from the SOUNDEX function. The most distinct difference was that SSA generates not one but several keys for each full name that it processes. This obviates the need for surname detection and allows names where the surname order in the full name has changed to be successfully matched. It also allows matching when one or more names are truncated to an initial or omitted altogether. A preferred key can be identified for situations when a single key is required.

SSA uses a different algorithm for compressing names than that used by SOUNDEX. Unlike the latter it preserves some vowel information so it is better suited to New Zealand conditions. Better still, the algorithm can be tuned to the characteristics of the customer database. This is done by processing a sample list of names to establish name frequencies. If a person had a combination of fairly common names more information is preserved in the key to distinguish them from other people with similar names. Thus, unlike the SOUNDEX key, the SSA key(s) are not invariant with respect to the input names - the keys can depend on the relative frequency of other names in the database. An implication of this is that the keys may need to be regenerated from time to time as the database of names evolves and as naming fashions change. The algorithms used also allow common and custom nickname replacement, and omission of noise words like titles. They also allow identification of multiple names such as "John and Susan Smith", generating keys for each person.

SSA keys can be used in applications to test the likelihood of any pair of name entries describing the same entity. The use of multiple keys allows for a more sophisticated answer than "yes / no" - all that you get with SOUNDEX - and instead give you in effect a rating of how close the match is. They can also be used for positive and "negative" searches - the latter for the situation where you want to exclude any possibility that a given name is not already on your list. With all these features Search Software America claim a much higher strike rate for matching names than is possible with SOUNDEX and although I have not been able to test this claim I find it easy to believe.

GENERATING SSA KEYS IN SAS

THE SAS MODULE CALLS TO DLLS

To generate SSA keys you need a licence for the relevant SSA software in an operating environment where SAS supports calls to external modules. It seems functionality was originally provided in SAS to allow use of COBOL routines in the MVS environment, but the techniques used have been ported to SAS for OS2 and SAS for Windows (3.11, 95 and NT) to allow access to dynamically linked libraries (DLLs). A different method has to be used for SAS in the UNIX environment, though some of the issues raised in this paper might need to be addressed. The methods are described *in SAS Companion for the Microsoft Windows Environment Version 6 Second Edition (1996)* on pages 115 to 124. (It was not documented in the First Edition). There you will find not one but several of the most draconian warnings issued by SAS about the pitfalls of using these techniques. Live dangerously! (The warnings are fully justified, as I found.)

Any PC with Windows installed has its windows subdirectories stuffed full of DLLs. These are chunks of executable code that are not generally stand alone programs - instead they get linked to some main program as and when required, and they generally contain one or more functions (identified as ROUTINEs by SAS) which can be used by the main program. The calling program has to allocate memory space for parameters that are passed between it and the DLL (identified as a MODULE by SAS). It also has to know the name of the ROUTINE it is calling in the DLL and the number, type and format of the parameters that are being passed either way.

In SAS these attributes are read from a special text file that has to be generated before the DLL can be called. The filename SASCBTBL must be assigned to this file. (The name betrays the origins - SAS Cobol Table - though the current terminology refers to the Attribute Table.) SAS has had to address the situation that the parameters used by the called DLL might not be restricted to a length of 200 (a frequent restriction within SAS itself). It has to have a way of synthesising such a parameter from a collection of SAS character variables in the SAS data step or procedure making the call. There are a number of other issues that have to be addressed, such as whether the DLL has 16 or 32 bit architecture, whether the stack of parameters is ordered according to Pascal or C conventions, and whether parameters are denoted to the DLL by their value or by their address in memory. In Visual Basic (which is also able to call external functions in DLLs) these specifications are handled in the DIM statement that allocates the space for the parameter stack; in SAS they are in an external text file. One of the curious results is that the SASCBTBL filename need not be used in any INFILE or FILE statement subsequently - so long as the name is assigned SAS knows what to do with it.

Once the attribute table is set up and referenced SAS provides a variety of ways of passing data to and from the DLL - in a CALL MODULE statement, MOCULEN and MODULEC functions, and a slew of PEEK functions which can be used following a MODULE statement or function if a parameter is passed by address and you need the value of the result. The statement and functions can be used in subsequent SAS DATA steps, and modified versions of the functions can be used in PROC IML.

COMPILING THE SSA-NAME3 DLL

The software I tested was SSA-NAME3 (version 1.7) with SSA-EXTENSIONS under Windows3.11. As described above the software is tuned to the customer's data using a sample test file and there are some decisions on options that are made before the product can be compiled (it is written in C). The documentation was comprehensive and SearchSoftwareAmerica representative Lyndee Hyde helped me set it up. This stage of the process is not trivial but proceeded without too much drama. Lyndee also supplied a version of some SAS code written by Richard Langston of SAS USA and which had been used with a previous version of the SSA product.

The resultant DLL is unusual in that in addition using parameters to pass data back and forth it also uses parameters to allocate a substantial chunk of workspace - of the order of 10Kb. This subsequently caused problems when I tried to use the optional MODULE Log messages to trace a fault - implemented by passing '*i' as the first parameter in the MODULE statement (page 123). The MODULE Log is very verbose even for a module with a small amount of parameter passing - (It's advisable to set options obs=2 ; !) - with over 10Kb of parameters it was humungous. So much so that on the second call to the module SAS and Windows ran out of memory.

CALLING THE SSA DLL FROM SAS

Once the DLL was compiled the SAS source code originally supplied by Rick Langston was reviewed and modified to bring it

up to date with the changes to parameter and workspace specifications in the version of the SSA software we were using. A test database with 10,000 names and another with over 100,000 names were prepared. After several rounds of "spot the obvious errors" [all mine, sadly] the code ran and lo, the keys came forth! - at least for the first 2000 names in the dataset.

I modified the Rick Langston's code in a couple of ways. My plan was to use the SSA call to create the keys in a file where I could examine ways of using the keys to make fuzzy matches in SAS. First I took on board SSA's recommendation that to use the multiple keys it generated efficiently it was advisable to set up a separate table with one record per key - name - customer identifier, rather than to try and put multiple keys in a single customer record. Second I noted SAS's recommendations (on pages 119 -120) about accessing external DLLs efficiently. The DLL would stay loaded for the duration of each iteration of the DATA step controlling the process, but on return for the next "observation" it would be unloaded and reloaded when the MODULE call was next encountered. As I planned to process in excess of 160,000 names this meant the attribute table would have to be read each time, the DLL loaded into memory and the call processed. Plus I had some concerns at the time about memory leak in the process of continually allocating parameter stacks and loading the DLL. This was also going to add considerable (and unnecessary) overhead to the key generation process. The solution is to have just one iteration of the overall data step and to read the data in a DO loop using the end of file variable to halt the loop. The final version of the SAS source I used is appended.

I now entered a very unhappy time for me and for the support staff at SAS and SSA. The SAS source would run and call the DLL, and generate the keys - until it got to a certain point when everything died. We tweaked the amount of workspace allocated and we reinstalled Windows. No change. Eventually a site call by a SSA technical expert passing through Wellington pinned the problem down - it was in the data. A person with the surname "Bishop" was confusing the DLL because it was ignoring that word as a title, and it was in an endless loop looking for a second name. A patch to the software was made available immediately and it cured the problem. It did illustrate that the difficulty of finding a solution to a multi-supplier problem is some power of the number of suppliers involved - even if, as in this case, the technical support from both suppliers was excellent. Keys were generated for over 160,000 names - on average about three per person. See appendix1 for the SAS source for this stage.

This demonstrated the feasibility of using SAS and SSA together to generate name keys for a customer database. This would make most sense where the name data was already in a SAS dataset such as in a data warehouse, though using the particular methods described in this paper require the data to be available in a Windows (including NT) environment. Against this it should be noted that to obtain maximum benefits from the use of name keys the SSA keys should be generated at the point of data entry so they can be part of the validation process, and SAS is not often used in this context. Instead the call to the SSA software should be generated within the primary customer enrolment and management application.

USING THE GENERATED SSA KEYS

I had originally intended to spend more time on using the SSA keys to identify potential matches, but it took me longer to get the software working properly than I had anticipated. In the event it was relatively easy to use the generated keys to sort the data into groups of likely matches. Because each name could be represented by two or more keys matching pairs tended to recur through the data. This could be avoided by dropping out any group of matches where there was no name with a preferred key. Despite this I came to the conclusion that clustering matching names efficiently was not a trivial exercise and it might be best left to specialised software such as SSA's Data Clustering Engine. (See Appendix 2 for my code, which did work.)

What about the SSA keys and the SAS LIKE operator to search for particular names in a whole database? LIKE is bound to the SOUNDEX function so to use the SSA keys you would have to replace the where statement with something like this (which I have not tested), assuming that SSA keys have already been generated for the database

```
/*  Generate the SSA Key for
    "Smith" in a prior
    data _null_ step including
        call module ('N3SG..., etc) ;
    to generate the SSA keys, and
        call symput ("smithkey", SSA_keys)
        ;
    to define a macro variable containing
    the SSA keys (each enclosed in quotes)
    for "Smith"
    */
    data new ;
        set old ;
        where namekey IN (&smithkey) ;
        . . .
```

Hardly pretty, but it should do the job. Again, there are more elegant ways of doing this kind of search with the software SSA provide.

So is there a role in a SAS shop for using SSA? Possibly, especially if the shop is a SAS data warehouse and the host organisation is already using SSA technology to provide keys for customer names and addresses. In this situation the combination of SAS and SSA could be used to make specialised searches, regenerate keys for the warehouse, and identify redundancies and duplications.

FINDINGS

SAS and SSA can be made to work together, though the DLL generated by SSA tests SAS' module capacity to the limit. I'm sure that if it had to, SSA could clean up their side of the interface by only exposing as parameters the data it is strictly necessary to pass between the applications and by allocating its workspaces some in other way. Similarly SAS's requirement to read a text file containing the attributes table for the DLL each time it loads the module, and its action to unload the module at the end of each iteration of a data step, could if necessary be eliminated. I don't think either of these things will happen soon, because it is only in the interaction of these two pieces of software that the peculiarities of the one become a problem for the other.

Will I continue to use SAS calls to SSA DLLs in my analyses? No, with regret, not until my organisation implements name search keys like those generated by SSA in its primary customer database applications so it can realise the benefits I have demonstrated to be available. If this were to happen it would be easier to justify using the SSA software with SAS in the data warehouse.

TRADEMARKS

REFERENCES

Barron, D.L. and Hemedinger, C., "Accessing Dynamic Link Library Routines with the SAS System for Windows" in *Observations®* Volume 5, Number 2, pages 26-34,

"External DLL Example 1 : Calling Functions Within External DLLs from the DATA Step", in the SAS Sample Library

"External DLL Example 2 : Calling Functions Within External DLLs from a SAS Applications", in the SAS Sample Library

"Accessing External DLLs with SAS 6.1x for Win32s", as document TS 460, SAS Institute

For an alternative approach to fuzzy matching in SAS:

Charles Partridge, "The Fuzzy Feeling SAS Software Provides to the Electronic Matching of Records Without Common Keys", Paper 28 SUGI22 (1997)

ACKNOWLEDGMENTS

I have appreciated the opportunity to work in this area and thank SAS Institute (NZ), particularly Franco Pierantoni, for technical support; and Lyndee Hyde, Martin Gronerth and technical staff at SearchSoftwareAmerica in Australia that I came into contact with, for their support throughout.

CONTACT INFORMATION

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

Richard Carson
Work and Income NZ
PO Box 12 136
Wellington 6004
NEW ZEALAND
Work Phone: +64 4 916 3457
Fax: +64 4 916 3472
Email: richard.carson@winz.govt.nz
richard.c.carson@clear.net.nz

APPENDICES

1. CONTENT OF TEXT FILE REFERENCED AS SASCBTBL FOR THE ATTRIBUTE TABLE.

This text file contains information about the attributes of the parameters passed to and from the DLL. The attributes listed here are specific to the versions of SAS and SSA used in the project. The file defines the module (DLL name), the routine called (Function within DLL), some options, and 12 parameters including workspace. Where parameters exceed 200 characters they are defined in blocks of **args**, with the first block marked **fdstart**. Note the 10,000 bytes allocated to workspace.

This file is not submittted as program code but is referenced with the mandatory filename **SASCBTBL** in SAS code. (The fdstart statement and the comment, prefixed by *, should be on the same physical line.)

```
   routine N3SGNZP minarg=65 maxarg=65
   arch=bit16
   stackorder=l2r
   stackpop=called
   module=N3SGNZ
   ;
   arg 01 input char format=$char8.    fdstart;
          * service;
   arg 02 update char format=$cstr20.  fdstart;
          * extrc;
   arg 03 input char format=$char80.   fdstart;
          * extfunc;
   arg 04 input char format=$cstr55.   fdstart;
          * namein;
   arg 05 update char format=$cstr55.  fdstart;
          * cleanname ;
   arg 06 output char format=$char200. fdstart;
          * word stack;
   arg 07 output char format=$char58. ;
   arg 08 output char format=$char142. fdstart;
          * key stack;
   arg 09 output char format=$char200. fdstart;
          * search table;
   arg 10 output char format=$char200. ;
   arg 11 output char format=$char200. ;
   arg 12 output char format=$char77. ;
   arg 13 output char format=$char20.  fdstart;
          * cats;
   arg 14 output char format=$char200. fdstart;
          * ssawork;
   arg 15 output char format=$char200. ;
   arg 16 output char format=$char200. ;
   arg 17 output char format=$char200. ;
   arg 18 output char format=$char200. ;
   arg 19 output char format=$char200. ;
   arg 20 output char format=$char200. ;
* 40 lines omitted ;
   arg 61 output char format=$char200. ;
   arg 62 output char format=$char200. ;
   arg 63 output char format=$char200. ;
   arg 64 output char format=$char1.
      fdstart;    * parmx;
   arg 65 output char format=$char1.
      fdstart;    * parmx;
```

2. SAS CODE GENERATES SSA KEYS FOR A SAS DATA SET CONTAINING NAMES

```
/*  Program to allocate SSA search keys
    to a file consisting of names and id
    numbers.
    By  Richard Carson
        Senior Analyst
        National Debt Management Unit
        Work and Income NZ

    richard.carson@winz.govt.nz
    richard.c.carson@clear.net.nz

This code is derived from an example
written by
    Rick Langston <sasrdl@unx.sas.com>
    SAS Institute Inc.

It was developed on a Windows 3.11 PC
and the module parameters shown are
specific to the 16 bit platform.
*/

COMMENT
    Assign libname for test data ;

    libname ssatest 'h:\adhoc\data\' ;

COMMENT
    Allocate the file name for the
    attribute table.
    *Must* use this filename - physical
    file name is not critical ;

    filename sascbtbl 'attrtbl.dat';

COMMENT
    Macro variables define which records
    would be processed.  This can be done
    in other ways but here I needed the
    flexibility to set these parameters
    explicitly while testing. ;

%let firstone = 1 ;
%let lastone = 161234 ;
%let finish = %eval(&lastone - &firstone
                    + 1) ;

COMMENT
    The input data is in a SAS data set
    but earlier I used a fixed length
    text file.  Any names which generate
    a SSA warning code are output to the
    excep data set. SWN is the main
    customer identifier number ;

data    ssatest.coded (drop = ext_rc)
        ssatest.excep ;
    length   swn               8
             namein            $ 55
             ssakey            $ 5
             keytyp            $ 2
             prefkey           $ 1
             service           $ 8
             ext_rc            $ 20
             ext_func          $ 80
             namein            $ 55
             cleannam          $ 55
             wrdstk01          $ 200
             wrdstk02          $ 58
             Keystak1          $ 142
             srchtb01-srchtb03 $ 200
             srchtb04          $ 77
                 /* 677 total*/
             cats              $ 20
             ssawrk01-ssawrk50 $ 200
             parmx             $ 1
         ;
    format   ssakey $hex10.
             swn z9.
             namein $48.
         ;
    keep   swn namein ssakey keytyp ext_rc
           prefkey ;
    *   swn is customer identity number
        namein is customer name in full
        ssakey is generated SSA key for name
            [multiple keys per name, one
            record per key]
        keytyp identifies type of key
        ext_rc is the return code generated
            by the module
        prefkey identifies whether the key is
            the preferred key
        ;
*NOTE   Data step modified to only load DLL
        once ;

    do k=1 to &finish ;
       set ssatest.dmsnames ;
       namein = trim (cstfname) || " " ||
                trim (cstlname) ;
       service     = 'NAMESETM';
       ext_rc      = '90000000009000000000';
       ext_func    = '*NOSTAB*' ;
*   Resetting parameter values before the
    module call ;

        call module (
                'N3SGNZP',
                service,
                ext_rc,
                ext_func,
                namein,
                cleannam,
                of wrdstk01-wrdstk02,
                keystak1,
                of srchtb01-srchtb04,
                cats,
                of ssawrk01-ssawrk50,
                parmx,
                parmx
                ) ;
        do i = 1 to input (substr
                   (keystak1,1,2), 2.) ;
            ssakey = substr(keystak1,
                     7*i-4,5) ;
* NB key type is in following 2 bytes ;
            keytyp = substr(keystak1,
                     7*i+1,2) ;
            if ssakey = substr
                      (srchtb01,1,5)
                then prefkey = "0" ;
                else  prefkey = "1" ;
            if substr(ext_rc,1,4) > "0000"
                then output excep ;
                /*else*/ output coded ;
        end ;
    end ;
    stop ;
run;
```

3. SAS SOURCE FOR CLUSTERING NAMES USING SSA KEYS

```
/*  Program to cluster similar names
    based on SSA search keys using the
    file output from the previous
    SAS program in which
```

```
        swn is customer identity number
        namein is customer name in full
        ssakey is generated SSA key for
            name [multiple keys per name,
            one record per key]
        keytyp identifies type of key
        ext_rc is the return code
            generated by the module
        prefkey identifies whether the
            key is the preferred key
    By  Richard Carson
        Senior Analyst
        National Debt Management Unit
        Work and Income NZ
    richard.carson@winz.govt.nz
    richard.c.carson@clear.net.nz
*/

COMMENT
    First sort records in key order
    Note that records containing the same
    person but with an alternate key will
    be separated to be adjacent to records
    of other entities with similar names
    ;

proc sort data=ssatest.coded
          out = coded1 ;
    by ssakey prefkey ;
run ;

COMMENT
    Eliminate records which are on their own
    - no other record with matching keys.
    To cut down the number of clusters
    repeating the same names under different
    keys, only keep the clusters [BY groups]
    containing at least one record with a
    preferred key ;

data coded2 ;
    set coded1 ;
    by ssakey prefkey ;
    if not (first.ssakey and last.ssakey) ;
    retain keepgrp "Y" ;
    if first.ssakey and prefkey > "0" then
            keepgrp = "N" ;
    if keepgrp = "Y" then output ;
    if last.ssakey then
            keepgrp = "Y" ;
    drop keepgrp ;
run ;

COMMENT
    Sort the remaining clusters back into
    original order.  SWN is the customer
    identifier number in the database ;

proc sort data = coded2 ;
    by swn ;
run ;

COMMENT
    Merge the clusters back with the original
    data extract to pick up other identifying
    data.
    Can have several records per identity
    still at this stage. ;

data coded3 ;
    merge   coded2 (in=coded)
            ssatest.dmsnames
                (keep = swn bthdt address1
                        dist
                        /* other data */) ;
    by swn ;
        if coded ;
run ;

COMMENT
    Restore the cluster order but sort by
    additional identifiers ;

proc sort data=coded3 ;
    by ssakey bthdt ;
run ;

COMMENT
    Eliminate matching names with different
    birth dates -
    coded4 contains clusters grouped by name
        and birthdate
    coded5 contains records without matching
        dates of birth or where the date of
        birth is missing for checking on
        other identifiers ;

data coded4 coded5 ;
    set coded3 ;
    by ssakey bthdt ;
    retain qdist ;
        if not (first.bthdt and
                last.bthdt)
            and bthdt > . then
            do ;
                if first.bthdt then
                    qdist = dist ;
                output coded4 ;
            end ;
        else
            do ;
                qdist = " " ;
                output coded5 ;
            end ;
run ;

COMMENT
    Finish elimination of single record
    clusters and group clusters by site
    for further inspection ;

data coded6 ;
    set coded4 ;
        by ssakey bthdt ;
            if first.bthdt and last.bthdt
                then delete ;
run ;

proc sort   data=coded6
            out = ssatest.coded6 ;
    by qdist ssakey ;
run ;
```