

Fuzzy Key Linkage

Robust Data Mining Methods for Real Databases

Sigurd W. Hermansen, Westat

Abstract

Results of data mining depend heavily on the quality of linkage keys within a search dataset and within its database target. Linkage failures due to errors or variations in linkage keys have few symptoms, and can hide or distort what data have to tell us. More robust methods have promise as remedies, but require careful planning and understanding of specialized technologies. A tour of fuzzy linkage issues and robust linkage methods precedes a review of the results of a recent linkage project. Sample SAS programs include tools and tips ranging from SOUNDEX() and SPEDIS() functions to hash indexing macroprograms.

Introduction

Relational Database Management Systems (RDBMS's) have evolved into a multi-billion dollar industry. In no small part the industry has succeeded because RDBMS's protect the integrity and quality of data. Large organizations have committed huge sums of money and many person hours to enterprise RDBMS's. But while typical RDBMS's effectively repel any attempt to insert duplicate key values in data tables and subvert database integrity, they remain remarkably vulnerable to other types of errors. Most obvious of all, linkage of an insert or update transaction to a database fails whenever the search key in the transaction fails to match bit-by-bit the target key in the database. If a transaction key contains the person ID US Social Security Number (SSN) of 105431002, for instance, instead of the correct 105431802, it will fail to link to the corresponding record for the same person. Correct linkages of tables in an RDBMS depend entirely on the accuracy of columns of data used as key values. Errors in the face values of keys, whatever the sources, not only lead to linkage errors, but also persist. Once admitted to a database, errors in keys seldom thereafter appear on the radar screen of a system administrator.

Do errors in primary and foreign keys actually occur in real databases? Pierce (1997) cites a number of reports indicating that in the early 1990's a near majority or better of US business executives recognized data quality problems in their companies. Arellano and Weber (1998) assert that the patient record duplication rate in single medical facilities falls in the 3%-10% range. Many who have assessed the accuracy of the US SSN as a personal identifier in federated databases, including the author, peg its

accuracy at somewhere between 93% and 97%.

These estimates suggest a 5% $\pm$ 2% rate of error in attempts to link transactions or events to a master database.

Failures of keys to link properly have more impact where analysts are mining data for a few nuggets of information in a mountain of data, or where access to critical data requires a series of successful key links. In both situations, errors in keys propagate. Consider how a 1% key linkage failure rate propagates over a series of key links required for a basic summation query [SAS PROC SQL syntax],

```
SELECT DISTINCT Person_ID,
      SUM(amount) as OUTCOME
FROM Events GROUP BY Person_ID;
```

Key linkage failures may hide the skew of the true distribution. Even small rates of errors produce bias and outliers, such as the summary of amounts per group by a count of related events (GT10), as shown below.

ID Group		GT10		amount
true	in DB	true	in DB	
10111	10111	T	T	300
10111	10111	T	T	100
13111	12111	T	F	200
10111	10111	T	T	100
12111	12111	F	F	100
13111	13111	T	T	100
10111	18111	T	F	400

Result of summation query:

		OUTCOME	
GT10	true	computed	
T	1,200	600	
F	100	700	

Errors can affect both the counts of related events and the amounts being summed. Chains of keys that link data in subsidiary tables to master records, typical in SQL views, prove even more vulnerable to errors. Transposing digits in a short integer key likely converts one key into another key value already used to link a different set of data, as in

Table T1		Table T2	
ID	status	ID2	ID
21	Negative	43	21
12	Positive	34	21*

TUTORIALS

Table 3

ID3	ID2	subject
76	43	patientXYZ
67	34	patientRST

*in truth, T2.ID=12

VIEW V1:

```
SELECT T23.subject, T1.status FROM
T1
  INNER JOIN
  (SELECT T2.ID, T3.subject AS subject
   FROM T2 INNER JOIN T3
     ON T2.ID2=T3.ID2) AS T23
 ON T1.ID=T23.ID;
```

In this case, a transposition in T3.ID links PatientRST to "Negative" and not to the correct value of "Positive", yet does not trigger a referential integrity constraint. The RDBMS validation scheme fails and the error remains unnoticed. Key linkage errors such as these undermine the efforts of data miners to extract interesting and important information from data warehouses and distributed databases. Many database administrators may have good reason to believe that critical identifying keys in their databases have much lower error rates. Others, whose databases support applications such as direct marketing, might view a 5% linkage failure rate as perfectly acceptable. All others need to consider more robust linkage methods. Knowledge is power, but bad information quickly pollutes a knowledge base.

Alternative Linkage Keys

Robust linkage methods take advantage of alternative key patterns. An alternative key may work when linkage on a primary key pattern fails. If linkage on a 10-digit integer key fails, for instance, an alternative key consisting of standardized names and a date of birth could have a reasonable chance of being a correct match. So would other alternatives, such as a partial sequence of digits in a numeric identifier combined with either a standardized last name or a month and year of birth. Others, such as a match on first name and zip-code, would not.

Alternative linkage keys have to meet at least a couple of basic requirements. First and foremost, a key has to have a fairly high degree of discriminatory power. A weak identifier often finds too many matches that contain too little information to rule them out, much less verify them. Second, the alternative key has to have a good chance of linking correctly when the primary key fails to link. Two alternative linkage keys with independent 5% error rates, for example, have an expected joint failure rate of 0.25% or only 1/20th the rate of either taken alone. For independent 1% error rates, the combined rate

falls to 1/100th of the rate of either taken alone.

Fuzzy key linkage gains much of its power by providing alternatives that we would not need in a world of perfect information, yet, in the real world prove necessary to prevent costly linkage failures.

Because linkage failures present no obvious symptoms in a typical database system, the information that these failure hide often surprises clients. As data miners' close cousins, statisticians, know all too well, it takes a lot more evidence and effort to build a good case for a finding that goes against the grain of conventional wisdom, but it scores a lot more points. To compete effectively with an established database administration group, a data miner needs to offer alternatives to routine methods.

Nonetheless, any scheme that involves alternative linkage keys inevitably creates problems for database programmers and, by extension, database clients. The latter group includes not only persons who depend on enterprise RDBMS's for information, but also clients of networks, of Internet search engines, and of wireless services. These groups are growing rapidly and becoming increasingly dependent on fuzzy key linkage for information. Who among us has not found it frustrating to search through results of a Web search and still not find a Web page that should be there? A Boolean alternative ($x \text{ OR } y$) search may find a few more relevant pages, but it often buries them in an ocean of irrelevant pages. In Silicon Valley speak, the sounds of terms associated with robust database searches, "disjunctive query" (Claussen et al, 1996), "iceberg query" (Fang et al, 1998), "curse of dimensionality" (Beyer et al, 1999), "semi-structured data" (McHugh et al, 1977), forewarn us of the computational burden of alternative key linkage.

Of course a decision to integrate alternative linkage keys into database access does not settle the issue. A data miner must also choose the right degree of fuzziness in key linkage. Suppose a data miner uses a search key to locate instances of something that occurs at a rate of approximately one-percent in a database. If the data miner selects an alternative key that matches in error to 1% of the same database, the specificity of key linkage cannot exceed 50% on average. For each true match selected, fuzzy linkage would select on average one false match.

Fuzzy Linkage Methods

Fuzzy key linkage has at least one thing in common with drug therapy. A few "active ingredients" (AI) help alleviate the problem of errors in linkage keys, but each has side-effects that have to be managed carefully with "buffering agents" (BA). The active

ingredients in fuzzy linkage increase dramatically the time and resources needed to compare two sets of records and determine which records to link. The buffering agents do everything possible to make up for losses of efficiency and bring the linkage process sufficiently up to speed to make it feasible.

The order in which different active ingredients get used in the linkage process proves critical. Initial stages of linkage have to strip irrelevant data from key values and filter data as they are being read into buffer caches under operating system control, and do so before the linkage program moves them into working memory or disk space.

Reduced Structure Databases and Data Shaping (AI)

As the scale of databases and the dimensions of alternative keys increase, the idea of loading all key values into a single database, much less contiguous memory, becomes increasingly an academic fantasy. A more realistic linkage model leaves very large data objects in place, outside the key linkage application, and lets the linkage program select only key values and relevant data for processing within the application.

Alternative keys usually represent a dimension of the real world, such as place, interval of time, and other context cues, plus event outcomes, attributes, or other facts that in some sense belong to an entity. In distributed or federated databases, alternative key values retain their meaning while integer key values removed from the context of a RDBMS lose their meaning. An integer makes a good employee ID in an enterprise database, but a poor ID for a person in a database that spans enterprises.

The so-called Star Schema for data ware-housing makes it easier to develop a logical view of data that includes alternative and overlapping information. Earlier articles, especially Hermansen (2000), present ideas for implementing alternative keys, disentangling data from file systems, and restructuring databases into forms that better support alternative logical views.

Real database case study(1): *A database contains over 10 million records of blood donations. The number of donation records per donor varies from one to several hundred. Multiple observations of donor demographics show surprising variations within sets of records for individual donors. Related donation and donor tables allow full capture of observed responses by donors as well as most likely attributes based on modes of multiple responses. The accuracy of the donor database improves over time*

as true responses have a better chance of repeating than random errors.

Compression, Piping, Filtering, and Parallel Processing (BA)

No way around it: alternative linkage keys crowd whatever bandwidth a network and OS have to offer. Even bit-mapped composite key indexes become unwieldy. Multiple columns containing names, addresses, date/times, category labels, and capsule descriptions replace neat, continuous sequences of nine-digit ID's.

Harry X Lime 1923256Vienna Austria
replaces
105342118

Practical remedies include

- 1) *compression of data streams on a database server and decompression in a pipe that an linkage program reads:*

State-of-the-art mainframes implement data compression and piping transparently. Smaller machines leave it up to the programmer to compress data files and set up pipes to decompress them in stream. In the SAS System (Unix in this case) the FILENAME statement,

```
FILENAME zipPipe PIPE 'gzcat <file  
path(s) with .zip* extension>';
```

reads zipped files through an INPUT process. The programmer can enter a list of file names in a specific order, or specify a regular expression that yields a list. (The last asterisk in *.zip* may only prove necessary when files have hidden version numbers.) In either case the source data files remain in place while data stream into the linkage program. When reading a very large set of records with a SAS program, a pipe often works faster than inputting data directly from an intermediate SAS dataset;

- 2) *filtering data while cached in memory buffers, before they move to the working storage that a linkage program allocates:*

In the SAS System, an INPUT statement in a DATA STEP VIEW and a PROC SQL statement referencing the DATA STEP VIEW in a FROM clause caches data in memory where a SQL WHERE clause acts as a filter. Only data that meet initial conditions pass through the filter and enter a SAS WORK dataset;

TUTORIALS

- 3) *extracting minimal subsets of data from database servers using views:*

As a rule a database server does a more efficient job than an application program of handling basic operations on its data tables. A SQL SELECT statement or equivalent in a stored view has primary access to indexes, integrity constraints, and other metadata of the database object, and it executes in an environment tuned to allow quick access.

- 4) *running data extraction programs in parallel on multiple processors, or even on multiple database servers:*

Some database systems allow more than one thread of a key linkage program to execute in parallel on different processors. The MP CONNECT procedure under the SAS/CONNECT® product, for example, lets the programmer RSUBMIT different sections of a program to different processors on a SAS server, or to different database servers, where they can execute in parallel. Doninger (2001) and Bentley (2000) describe the benefits of parallel processing with MP CONNECT and include examples. Parallel execution of views on different database servers, for example, makes good use of this new feature of SAS Version 8.

Real database case study (2): *A database programmer reported recently on SAS-L that subsetting data into a SAS dataset via a DB2 view cut CPU time to 11% of that required to read the full database and then subset it. It also reduced elapsed time by a factor of six (see SAS-L Archives, subject: RE: SQL summarization question, 12/14/2000).*

Data Blurring, Condensing, and Degrees of Similarity (AI)

A linkage key, at least after encoding for storage in a digital computer, amounts to nothing more than a pattern of bits. To attain a higher degree of specificity in key linkage, one must either add information (more bits) or simplify the pattern (using some form of metadata template for valid patterns). To attain a higher degree of sensitivity of key linkage, one must either suppress information (mask bits) or simplify the pattern. Greater specificity means fewer false links among keys; greater sensitivity means fewer failures to find true links. Suppressing bits in a key pattern prior to comparing two keys obviously risks trading more false links for fewer failures to find true links. Confining a sequence of bits (a field in a record) to a limited domain has some chance of increasing sensitivity of linkage by reducing

meaningless variations in keys related to the same entity. Fuzzy key linkage attempts to achieve better sensitivity of linkage with the least loss of specificity.

Although the term "fuzzy", as in "fuzzy math", suggests a vague or inconsistent method of comparison, fuzzy key linkage actually provides more precise and consistent results of comparisons. Fuzzy key linkage resolves comparisons of vague and inconsistent identifiers, and does so in a way that makes better use of information in data than bit-by-bit comparisons of keys. It simply takes a lot more time and effort to eliminate alternatives.

Real database case study(3): *The article by Cheryl Doninger cited above appeared first under the name "Cheryl Garner". Under the SAS Web page, "Technical Documents: SAS Technical Support Documents--TS 600 to TS699", a search on the criterion "multiprocessing AND Garner" produced nothing, but a search on the alternative "multiprocessing AND Cheryl" located the correct document.*

Operators and functions specifically developed for fuzzy key linkage make it easier to compare certain forms of alternatives. Each of three general types has a special purpose.

"Blurring" and "condensing" functions transform instances in a domain of values into a domain that has a smaller number of distinct values. Blurring maps similar values to one value; it reduces incidental variation in a key. Condensing reduces the remapped values to a set (distinct values). The SOUNDSEX() function or operator (=), for instance, condenses a set of surname strings to a relatively small number of distinct values:

SURNAME SOUNDSEX

Neill	N4
Neal	N4
Neil	N4
Niell	N4
Neall	N4
Nil	N4
Nel	N4
Nill	N4
Nell	N4
Nilson	N425
Nelson	N425
O'Neil	O54
O'Neal	O54
Oneill	O54

Blurring and condensing facilitate indexing of keys. An index on a blurred and condensed key occupies less bandwidth and memory, and it clusters similar

key values. A SOUNDEX() transform of any of the nine similar surnames beginning with an "N" and ending with an "L" (above) will match to an index containing "N4".

A "degree of similarity" operator or function compares two key values and produces a numeric value within an upper and lower bound. The upper bound indicates identical keys; the other bound indicates no similarities. Combined with a decision rule, usually based on an explicit, contextual, or default threshold value, the fuzzy operator or function reduces to a Boolean. It aggregates the results of comparisons of alternative keys into a numeric score, accepts as true links those with scores that exceed a threshold, and rejects the others. As an example, the SAS SPEDIS() or "spelling distance" function calculates a cost of rearranging one string to form another, where each basic operation used to rearrange the string has a cost associated with it. A CASE clause in SAS SQL implements SPEDIS() in a way that sets a neutral value of 0.4 should either of two US SSN strings turn up missing, and a value in the range of zero to one if the comparison goes forward.

```
case when t1.&SSN1="" or
t2.&SSN2=""
then 0.4
else max((1-length(t1.&SSN1)*
pedis(t1.&SSN1,t2.&SSN2)/200)),0.1)
end as SSNcost
```

A programmer can use the calculated variable SSNcost in a Boolean "OR" expression, as in

```
WHERE (calculated SSNcost > 0.5
AND t1.surname=t2.surname)
• OR (calculated SSNcost > 0.8
AND t1.firstname=t2.firstname),
```

to implement linkage on alternative key patterns, or combine it with another degree of similarity, to achieve the same goal.

So-called "regular expressions" and other pattern-matching operators and functions (such as the SAS INDEX() function) normally point to a location in a string or file, or return a "not found" value. This feature in particular facilitates checks for alternative patterns in strings, numbers, or other semi-structured elements in databases. Rhoads (1997) demonstrates practical uses of pattern-matching functions. These include tests for a match on a template. Extensions, such as a series of searches for the location of a phone

Data Standardization, Cleansing, and Summaries (BA)

Specialized programs for "mailing list hygiene", standardizing codes, parsing text into fields, and other database maintenance tasks are beginning to appear in greater numbers and variety each year. Patridge (1988) and the www.sconsig.com Web site offer both free and commercial database standardization, cleansing, and summarization programs. Pre-processing can obviously reduce the risk of fuzzy linkage failures and false matches. Partially for that reason, database administrators are paying more attention to data quality metadata, including audit trails. Best practice combines data quality control with robust key linkage methods. To help fill in that niche, SAS® has recently purchased Dataflux and its Blue Fusion and dPower Match standardization and fuzzy linkage products.

A number of data warehouse developers are rediscovering that old warhorse, the SAS PROC FREQ, and even more sophisticated stuff such as stratified sampling, linear regression, and cluster analysis. Linkage quality really comes down to keeping expected costs of errors within limits.

Real database case study(4): *In a database of >5M blood donation records linked by a non-informative donor ID to 1.5M donors, we grouped by donor and identified unusual sequences of screening test results. We separated out borderline cases, verified the results of database searches, estimated 0.05% (95% CI 0-1.5%) frank technical errors in data management, testing systems, and process controls (Busch et al, 2000), and recommended process enhancements in the blood collection industry to help detect and prevent false-negative results.*

Blocking and Screening on Synthetic, Disjunctive Keys (AI)

RDBMS performance depends heavily on indexes of search keys that clients use to link database records to transactions. As volumes of data approach the limits of a database, platform, and network, indexes take on an increasingly important, and constraining, role. Indexes bound a search on that index to a small block of key values. A *deus ex machina* database tuner can conjure up indexes to optimize transactions, but in very large databases searches on alternative keys mire in quicksand.

"Blocking", an old trick in record linkage methodology, implements a series of searches on partial primary key candidates. A clever blocking

scheme might have an initial search on surname and date of birth, followed by search on date of birth and postal code, and finally a search on surname and postal code. Later stages consider only new candidates for links, not those confirmed as correct links in a prior stage. A good blocking strategy implements alternatives (surname AND DOB) OR (DOB and PostCode) OR (surname and PostCode) so that errors or variations in any one field do not cause linkage failures. The fact that each block consists of a conjunctive (AND) query means that an index can keep the computational burden of an indexed search within bounds.

Blocking has one major disadvantage. It takes multiple passes through a set of data to complete the screening process. When searching databases with many millions of rows in a single table, a single-pass solution makes better sense.

A better screening strategy defines a "synthetic key" for each block and creates an index for each. Each of the synthetic keys represents an alternative linkage key or fragments of an alternative key. Table 1 provides a picture of definitions of eight keys (columns) synthesized from eleven fragments or transforms of alternative keys. Whether character or numeric, key patterns reduce to strings of bits. By design each synthetic key has sufficient discriminatory power to bind only to similar key patterns, but relatively narrow bandwidth. For instance, it takes just seventeen bytes to represent a first initial of a first name, a soundex transform of a surname, and a Julian DOB. On anything larger than a low-end PC, a number of such indexes will fit

Table 1: Synthetic Linkage Keys

	k ₁	k ₂	k ₃	k ₄	k ₅	k ₆	k ₇	k ₈
SSN				✓				
SSN5					✓			
SSN4						✓		
LN	✓	✓			✓	✓		
SLN			✓				✓	✓
FN	✓	✓						✓
FN1			✓		✓	✓	✓	
MI								
DOB			✓		✓	✓		
DOB*	✓						✓	

Glossary:

k_i: synthetic search keys
 SSN5: digits 1-5 of SSN in decreasing order;
 SSN4: digits 6-9 of SSN in decreasing order;
 LN: last name;
 SLN: yield of Soundex(LN);
 FN: first name;
 FN1: first letter of first name;
 MI: middle initial (not used in screening);
 DOB: date of birth;
 DOB*: date of birth +/- (U/L) 32 days;
 MDX: day and month of birth;
 Sex: (not used in screening)

into addressable memory. It then becomes technically feasible to

- transform and synthesize k alternative keys or fragments of keys from a moderately large (say, 100K rows) search dataset and build multiple indexes;
- load all of the indexes into memory and hold them there;
- scan a huge dataset one row at a time, transform and synthesize alternative keys for that row, and check each synthetic key against its corresponding index;
- select from the huge dataset only those rows linked to any one or more of the indexes.

The indexes implement a disjunctive "query flock" (Tsur, 1998) that screens for possible links during one pass through a huge dataset. Rows of data that fail to match any of the multiple key patterns pass through the screen. Those that match at least one of the patterns get set aside for further attention.

Multiple, Concurrent Key or Hash Indexes and Rescreening (BA)

Clearly a flock of indexes has to be compact to load into memory and easy and quick to search. The balanced B-Tree indexes used in RDBMS's conform to the first constraint. In early attempts to implement screening on multiple indexes, we read data from files and used them to write SAS FORMATS. If the SAS expression PUT(key,ndx_i) yielded a "+", the key value matched the ith index. This effective implementation in SAS of a B-Tree index, called "Big Formats" on SAS-L, worked surprisingly well. Nonetheless, subsequent postings by Paul Dorfman on SAS-L proved and demonstrated that hash indexes implemented in SAS worked much quicker. Testing of hash indexes against big formats, data step merges, and SQL query optimizations, by Ian Whitlock and others, established the equivalence or superiority of

Dorfman's hash indexes, and the dramatic improvements they bring to linkage of very large volumes of data.

Though ideal choices in theory, hash indexes prove cryptic and difficult to specify. Almost all of the SAS programmers who had a penchant for refining Knuth's sorting and searching algorithms have by now found jobs in Palo Alto or Seattle, and are writing C++ object classes and Java applets. Fortunately, Dorfman and a few others have remained loyal to the craft. To make it easier for the rest of us, Dorfman has written SAS macro-programs %hsize, %hload, and %hsearch:

```
%macro hsize (data=,hid=,load=.5);
  %global z&hid;
  data _null_;
    p = ceil(nobs / &load);
    do until (j = u + 1);
      p ++ 1;
      u = ceil(sqrt(p));
      do j=2 to u;
        if mod(p,j) = 0 then leave;
      end;
    end;
    call
      symput("z&hid",compress(put(p,best.)
    ));
    put "info: size computed for hash
      table &hid is " p +(-1) ".";
  put;
  stop;
  set &data nobs=nobs;
run;
%mend hsize;

%macro hload (data=,hid=,key=,pib1=6);
  %global t&hid p&hid;
  %local dsid nvars found varfound vname
    varnum vnum;
  %local keytyp keylen xpib1;
  %let dsid = %sysfunc(open(&data,i));
  %let nvars = %sysfunc(attrn(&dsid,nvars));
  %let varfound = 0;
  %do varnum=1 %to nvars;
    %let vname=
      %sysfunc(varname(&dsid,&varnum));
    %if %upcase(&vname) = %upcase(&key)
      %then %do;
      %let varfound = 1;
      %let vnum = &varnum;
    %end;
  %end;
  %if &varfound = 0 %then %do;
    do;
      put "error: key=&key variable not
        found in dataset &data..";
      abort;
    end;
    %let rc = %sysfunc(close(&dsid));
    %goto mexit;
  %end;
  %let keytyp =
    %sysfunc(vartype(&dsid,&vnum));
  %if %upcase(&keytyp) = %upcase(c) %then
    %let keytyp = %;
  %else %let keytyp = ;
  %let t&hid = &keytyp;
  %let keylen =
    %sysfunc(varlen(&dsid,&vnum));
  %let rc = %sysfunc(close(&dsid));
  %if &pib1 > 6 %then %then %do;
    %let pib1 = 6;
    %put info: maximum value for pib1=
```

```
exceeded. pib1=&pib1 assumed.;
%end;
%let p&hid = &pib1;
do;
  array h&hid (0:&z&hid) &keytyp &keylen
    _temporary_;
  array l&hid (0:&z&hid) 8
    _temporary_;
  _r = &z&hid;
  eof = 0;
  do until (eof);
    set &data (keep=&key) end=eof;
    %if &keytyp = % %then %do;
      _h =
        mod(input(&key,pib&pib1..),&z&hid) + 1;
    %end;
    %else %do;
      _h = mod(&key,&z&hid) + 1;
    %end;
    if l&hid(_h) > . then do;
      l&hid:
        if &key = h&hid(_h) then
          continue;
        if l&hid(_h) ne 0 then do;
          _h = l&hid(_h);
          goto l&hid;
        end;
        do while (l&hid(_r) > .);
          _r ++ 1;
        end;
        l&hid(_h) = _r;
        _h = _r;
      end;
      h&hid(_h) = &key;
      l&hid(_h) = 0;
    end;
    eof = 0;
    drop _h _r;
  end;
%mexit;
%mend hload;

%macro hsearch (hid=, key=, match=);
  do;
    drop _h;
    %match = 0;
    %if &t&hid = % %then %do;
      _h =
        mod(input(&key,pib&p&hid..),&z&hid) + 1;
    %end;
    %else %do;
      _h = mod(&key,&z&hid) + 1;
    %end;
    if l&hid(_h) > . then do;
      s&hid:
        if h&hid(_h)=&key then %match = 1;
        else if l&hid(_h) ne 0 then do;
          _h = l&hid(_h);
          goto s&hid;
        end;
      end;
    end;
  end;
%mend hsearch;
```

The data= parameters require the name of a SAS dataset or view. The hid= parameters name the specific index being sized, loaded, and searched. The key= parameter identifies SAS variables that contain a value of the synthetic key either written to or matched to the index. The match= parameter names the Boolean result of an index search. These program excerpts show actual calls of %hsize, %hload, and %hsearch:

```
%hsize(data = mdx , hid = mdx , load = &lf );
data rslt.headid ...
```

```

%hload(data=mdx,hid=mdx,key= mdx,pibl=6);
.
.
do until (eof);
  infile header end=eof;
  input
    @ 01 rtype      $char01.
  .
  .
%hsearch(hid= mdx,key=mdx,match=m_mdx );
.
.
.

```

After screening, all of the rows of data selected from the huge dataset link to at least one of the rows of data in the search dataset, but some rows in the search dataset may not have linked to any of the rows in the huge dataset. A simple reversal of the screening process selects only those rows of data in the search dataset that match at least one row in the results of screening. We call this step "rescreening".

Where each row in the huge table has a very small chance of being a correct link, early elimination of unlikely candidates for linking greatly reduces the costs of later stages of the linkage process. Screening and rescreening cut large datasets down to manageable size.

Fuzzy Scoring and Ranking on Degrees of Similarity of Linkage Keys (AI)

Once screening has reduced a huge target dataset to, say, a mere million or so rows, and rescreening has reduced the number of rows in the search dataset by perhaps fifty to eighty percent, more intensive linkage methods become feasible. So-called probabilistic methods assign weights for individual field matches and mismatches for each pair of records, and sum the logs of these weights across fields. Estimated error rates in correct links, given a field match, and estimated coincidental links, given field frequencies, determine the composite weight or score. The higher the score for a pair of records, the higher the linkage program ranks them.

Probabilistic linkage and related methods have evolved over a span of some forty years into statistical modelling for control of both linkage failures and incorrect links. Winkler (2000) assesses the current state of record linkage methodology. Proceedings of a recent conference (Alvey and Jamerson, eds. 1997) on record linkage includes a historical perspective on development of specialized key linkage programs: OX-LINK (Oxford Medical Record Linkage System), GIRLS (Generalized Records Linkage System), from Statistics Canada; and, AUTOMATCH, originally developed by Matt Jaro at the US Bureau of the Census,

A relatively simple scoring program requires some guesswork about the values to assign to field match and mismatch weights and to a cut-off score. The values of weights generally increase with the relative importance of a field match to the chance of a correct match. Neutral weights for missing values help us focus on whatever subset of information a row of data contains.

In this implementation of a simple scoring program, a SAS macroprogram allows a user to assign variables names as parameters.

*** MATCH PROGRAM ***;

```

%macro mtch(      DSN1=      ,      DSN2=      ,
  SubmitID=      ,SourceID=      ,
  SSN1=      ,      SSN2=      ,
  LstName1=      ,LstName2=      ,
  FstName1=      ,FstName2=      ,
  MI1=      ,      MI2=      ,
  Sex1=      ,      Sex2=      ,
  Race1=      ,      Race2=      ,
  DOB1=      ,      DOB2=      ,
  DOD1=      ,      DOD2=      ,
  Zipcode1=      ,Zipcode2=      ,
  Zipcode3=      ,Zipcode4=      ,
  RECCODE1=      ,RECCODE2=      ,
  STATE1=      ,      STATE2=      ,
  key1=      ,      key2=      ,
  keyval1=      ,      keyval2=      ,
  Rectype=      ,      OutDSN=      ,
  c=      );

```

```

proc sql;
  create table &OutDSN as
  select t1.&SubmitID as SubmitID,
    t2.&SourceID as SourceID,
    t1.&STATE1 as STATE,
    t1.&RACE1 as RACE,
    t2.&RECCODE2 as RECCODE,
    t1.&Zipcode1 as ZIP,
    t2.&Zipcode2 as ZIPX,
    t2.&Zipcode4 as ZIPP,
  case when t1.&SSN1=t2.&SSN2 and t1.&SSN1 ne
    "000000000" and
    (soundex(UPCASE(t1.&LstName1))=
      soundex(UPCASE(t2.&LstName2))
    or t1.&DOB1=T2.&DOB2) then 1.0
    when t1.&SSN1=t2.&SSN2 and t1.&SSN1 ne
      "000000000" then 0.5
    when
      index(UPCASE(t2.&FstName2),substr(UPCASE(t1.&
        FstName1),1,3)) and
      soundex(UPCASE(t1.&LstName1))=soundex(UPCASE(
        t2.&LstName2)) and t1.&DOB1=T2.&DOB2 then 0.2
    when
      UPCASE(t1.&FstName1)=UPCASE(t2.&FstName2)
      and t1.&Sex1="F1" and t1.&DOB1=T2.&DOB2
      then 0.05
    else 0
  end as bonus,
  case when t1.&SSN1="000000000" or
    t2.&SSN2="000000000" then 0.4
    else max((1-
      (length(t1.&SSN1)*spedis(t1.&
        SSN1,t2.&SSN2)/200)),0.1)
  end as SSNcost,

```

```

case when
    UPCASE(t1.&LstName1)=
    UPCASE(t2.&LstName2)
        then 0.9
    when soundex(UPCASE(t1.&LstName1))=
        soundex(UPCASE(t2.&LstName2))
        then 0.6
    when t1.&Sex1 = "F1"
        then 0.4
    else 0.1
end as SDXLN,
case when
    UPCASE(t2.&FstName2)=UPCASE(t1.&FstName1)
        then 0.9
    when index(UPCASE(t2.&FstName2),
        substr(UPCASE(t1.&FstName1),1,3))
        then 0.6
    when index(UPCASE(t2.&FstName2),
        substr(UPCASE(t1.&FstName1),1,1))
        then 0.4
    else 0.2
end as FN2,

%if (&MI2 ne) %then
    case when substr(UPCASE(t1.&MI1),1,1)=
        substr(UPCASE(t2.&MI2),1,1)
            then 0.8
            else 0.2
    end as MI1,
;
%if (&Sex2 ne) %then
    case when t1.&Sex1=t2.&Sex2
        then 0.5
        else 0.2
    end as SexMch,
;
case when t2.&DOB2 = t1.&DOB1
    then 0.7
    when month(t1.&DOB1)=month(t2.&DOB2)
        and day(t1.&DOB1)=day(t2.&DOB2)
        then 0.6
    when t2.&DOB2 <= 1.05*t1.&DOB1
        and t2.&DOB2 >= 0.95*t1.&DOB1
        then 0.4
    else 0.2
end as BtwnDOB,
%if (&SourceID ne SSNC)
    %then t2.&SSN2 as SSNX, ;
        t1.&SSN1 as SSNC,
        t1.&LstName1,t2.&LstName2 as LNX,
        t1.&FstName1,t2.&FstName2 as FNX,
        t1.&Sex1,t1.&DOB1,t2.&DOB2 as DOBX,
        t2.&DOB2,calculated bonus +
        (calculated SSNcost*
        calculated SDXLN*
        calculated FN2*
        calculated BtwnDOB) as score
        from &DSN1 as t1,&DSN2 as t2
        where calculated score gt
            0.4*0.8*0.6*0.7 * &c
            and t1.&key1 = t2.&key2
            and t1.&keyval1 = t2.&keyval2
        order by calculated score
        DESCENDING,SubmitID,SourceID
;

quit;

%mend mch;

```

The structure of the SQL program makes it relatively easy to adapt to other purposes and to port to other SQL implementations.

Grouping Links and Decisions by Score Range (BA)

In some cases we expect more than one event row in a target dataset to link to one and the same person row in the search dataset; one event row in the target

dataset linked to more than one person row in the search dataset indicates at least one error. In lower score ranges, the number of cross-linked events should increase sharply. Clerical reviewers can verify small samples (≈ 300 links) of linked pairs drawn from different score ranges. Frequencies of reviewer decisions by scores make it possible to evaluate linkage performance within different ranges of scores.

Real database case study(5): During 2000 a particularly difficult linkage task required linkage of personal information on each member of a study cohort of around one-hundred forty-five thousand persons to a database of some twenty million exposure measurements containing names, demographic information, and a supposedly unique identifying number (US SSN) for each person. Some in the cohort should not link to any exposure measurements, and some should link to more than one. Researchers expected about ninety-seven thousand persons in the cohort to link to at least one exposure measurement. Roughly ninety thousand cohort records linked on the primary person key, SSN, to at least one exposure measurement.

Fuzzy linkage on a primary and on alternative keys linked the expected number of around ninety-seven thousand persons to at least one exposure measurement. About forty-five thousand of over two-hundred fifty thousand linked exposure measures required clerical reviews. A relatively large fraction of the ninety-seven thousand linked persons, 8.5%, linked to an exposure record on an alternative key, but not on the primary key.

Many linked on alternative keys had small errors in the primary key but had full or partial matches on names and demographic data. These almost certainly qualified as correct links. Around 2% or so of cases of records linked on identical primary keys, then failed to match on any alternative key or fragment of a key. Researchers reclassified these cases as linkage errors and dropped them from the set of linked records.

Conclusions

Fuzzy key linkage has an important role in data quality improvement of RDBMS's and other data repositories, and in linkage across databases. The computational burden of linkage of alternative keys means that such a task needs careful planning and good choices of resources. The SAS® System provides a rich variety of tools for conducting a linkage project and a basis for implementing new tools.

Acknowledgments

Paul Dorfman contributed many valuable ideas as well as programs and technical advice. Kellar Wilson and Lillie Stephenson tested variants of methods presented in the paper. Other Westat colleagues, especially Mike Rhoads and Ian Whitlock, and many contributors to SAS-L have for no fault of their own contributed to this effort.

References

- Alvey, W. and B. Jamerson, eds. "Record Linkage Techniques – 1997", Proceedings of an International Workshop and Exposition. Washington, DC, 1997.
- Arellano, M., Weber, G. "Issues in identification and linkage of patient records across an integrated delivery system", J. Healthcare Information Management, (3) Fall, 1998:43-52.
- Bentley, J. "SAS Multi-Process Connect: What, When, Where, How, and Why". Proceedings of SESUG 2K, Charlotte, NC, 2000.
- Beyer, K., J. Goldstein, R. Ramakrishnan and U. Shaft. "When Is 'Nearest Neighbor' Meaningful?", Proceedings 7th International Conference on Database Theory (ICDT99), pp.217-235, Jerusalem, Israel, 1999.
- Busch, M., K. Watanabe, J. Smith, S. Hermansen, R. Thomson, "False-negative testing errors in routine viral marker screening of blood donors", TRANSFUSION 2000;40:585-589.
- Doninger, C. "Multiprocessing with Version 8 of the SAS System", SAS Institute Inc, (2001) <ftp.sas.com/techsup/download/technote/ts632.pdf>
- Dorfman, P. "Table lookup via Direct Addressing: Key-Indexing, Bitmapping, Hashing", Proceedings of SESUG 2K, Charlotte, NC, 2000.
- Hermansen, S. 'Think Thin 2-D: "Reduced Structure" Database Architecture', Proceedings of SESUG 2K, Paper#1002, Charlotte, NC, 2000.
- McHugh, J., S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. SIGMOD Record, 26(3):54-66, September 1997.
- Patridge, C. "The Fuzzy Feeling SAS Provides: Electronic Matching of Records without Common Keys", 1998 <http://www.sas.com/service/doc/periodicals/obs/obswww15/index.html>
- Pierce, E., "Modeling Database Error Rates", DATA QUALITY 3(1) September, 1997.
- Rhoads, M., "Some Practical Ways to Use the New SAS Pattern-Matching Functions", Proceedings of the 22nd Annual SAS Users' Group International Conference: 72, San Diego, CA, 1997. <http://www2.sas.com/proceedings/sugi22/CODERS/PAPER72.PDF>.
- Tsur, D., Ullman, J., Abiteboul, S., Clifton, C., Motwani, R., Nestorov, S., Rosenthal, A. "Query flocks: A generalization of association rule mining", Proceedings of the 1998 ACM SIGMOD Conference on Management of Data, 1-12, Seattle, WA, June, 1998.
- Winkler, W. "The State of Record Linkage and Current Research Problems", Bureau of the Census, Suitland, MD, 2000.

Author Contact Information

Sigurd W. Hermansen
WESTAT, An Employee-Owned
Research Corporation
1650 Research Blvd.
Rockville, MD 20850
USA
phone: 301.251.4268
e-mail: hermans1@westat.com

Point, Set, Match (Merge) – A Beginner's Lesson

Jennifer Hoff Lindquist, Institute for Clinical and Epidemiologic Research,
Veterans Affairs Medical Center, Durham, NC

The phrase "Point, Set and Match" is used in tennis when the final game winning point is scored. Those terms are also special SAS techniques that can make you a champion SAS programmer. Projects can collect data from a number of different sources. Combining the information into a cohesive data structure is essential for resource utilization. One of the most valuable resources is your time. By combining and collapsing data into a small number of datasets, information can be accessed and retrieved more quickly and already properly linked. Two techniques used with manipulating existing data sets are SET and MERGE.

SET

The SET statement is often used in two ways – copying and appending.

Set – Copy

To avoid corrupting a permanent SAS data set, copy of the data set is desirable. Suppose the permanent SAS dataset In.Source A consists of 5 observation with 4 variables. The syntax to create a copy of the data set is Set <dataset name>.

```
SAS Code:   Data White;
            Set In.SourceA;
            Run;
```

The contents of the White data set are an exact replicate of the original In.Source A data set.

White data set

ID	GRP	AGE	ELIG
1	A	30	Y
2	A	40	N
3	A	50	N
4	A	60	Y
5	A	70	

Set – Append

The SET statement can be used to append or stack data sets.

Let data set Yellow consist of 3 observations with 3 variables

Yellow Data Set

ID	GRP	ELIG
3	B	Yes
5	B	No
6	B	Yes

```
SAS code: Data TwoSets;
            Set White Yellow;
            Run;
```

Any number of data sets could be listed. The white data set contributes 5 observations and the yellow data set tacks on 3 observations. All the variables in the two data sets are included. If any variables are only in one of the data sets, the variable is included in the concatenated dataset. The observations which originated from a dataset without a particular variable has the variable added to the observation with a missing value.

SAS Output:

ID	GRP	AGE	ELIG
1	A	30	Y
2	A	40	N
3	A	50	N

TUTORIALS

4	A	60	Y
5	A	70	
3	B	.	Yes
5	B	.	No
6	B	.	Yes

The observations are simply stacked starting with the data set listed first. SAS then continues tacking on the observations to the "bottom" of the list for each data set listed in the SET statement. This is especially useful when consolidating data sets with mutually exclusive records but the same variables.

MERGE

MERGE-GENERAL

Joining records "side by side" instead of stacking is another data consolidation technique. Many times your want to create a data set with one observation per patient/person. A merge statement then is more applicable. The remainder of the paper will be devoted to discussing the various types of merges – One to One Merge, Match Merge, One to Many Merge and Many to Many Merge.

MERGE-ONE TO ONE MERGE

The first type of merge is a one to one merge. The accidental or careless use of this merge can produce disastrous results. In all SAS merges a data set listed first (on the left) is overwritten by the corresponding data in the data set on the right. In a one to one merge the observations are conjoined by their relative positions-the first observation with the first observation, etc.

White Data Set					Yellow Data Set		
ID	GRP	AGE	ELIG		ID	GRP	ELIG
1	A	30	Y	↔	3	B	Yes
2	A	40	N	↔	5	B	No
3	A	50	N	↔	6	B	Yes
4	A	60	Y				
5	A	70					

```
SAS Code      Data MergeSet;
                Merge White Yellow;
                Run;
```

In this one to one merge, the values in the first three observations in the white data set are wiped out by the overlapping variables in the yellow data set even though they are NOT referring to the same individuals.

SAS Output

ID	GRP	AGE	ELIG
3	B	30	Yes
5	B	40	No
6	B	50	Yes
4	A	60	Y
5	A	70	

The resulting data set has "lost" patients with ids 1 and 2. The age for patient #3 appears to be 30 when it is actually the age for patient #1. Other errors include the ages for patients 5 and 6. Patient #5 has ages 30 years apart! Due to this potential to lose/corrupt data, the one to one merge is best avoided.

MERGE - MATCH MERGE

A refinement of the one to one merge is the match merge. Using the BY statement, variables are listed which SAS uses to match observations. Data sets must be sorted on the same variables as listed in the match merge BY statement. More than two data sets may be included in the merge statement. More than one variable can be listed on the BY statement. But only one BY statement is allowed for each Merge statement.

```

SAS Code:  Proc Sort data=white;
            By Id;
            Proc sort data=yellow;
            By Id;
            Run;

            Data Mergeby;
            Merge white Yellow;
            By ID;
            Run;

```

SAS Output

ID	GRP	AGE	ELIG
1	A	30	Y
2	A	40	N
3	B	50	Yes
4	A	60	Y
5	B	70	No
6	B		Yes

If two data sets have variables with the same name, the value of the variable in the data set named last will overwrite the value of the variable in the data set name previously. The ELIG for Patient #3 in the white data set was "N" but in the yellow data set ELIG was "Yes". Since the order in the Merge statement was white then yellow the value in the yellow data set appears in the merged dataset. However, due to the overwrite property this conflict of eligibility status is lost.

The match merge and the one to one merge differ in syntax only in the use of the BY statement. It is (too) easy to inadvertently leave off the BY statement. Results are NOT the same! SAS has acknowledged that this can be a problem. In version 8, there is a system option called MERGENOBY. It has 3 settings – None, Warning, Error. The Warning setting will write a Warning message in the log whenever a merge is performed without a BY statement but will continue processing. The Error setting will write an Error message in the log and halt processing. The None setting – no message is written in the log. I strongly recommend using at least the Warning option.

MERGE -IN Option

A useful option with both the SET and MERGE statements is the IN statement. The syntax is data set name (IN= temporary variable). The temporary variable is assigned a one if the observation came from that data set. The temporary variable receives a value of zero if the observation is not in that data set. The temporary variable exists only for the length of time it takes to process the observation. It is not accessible after the completion of the data step. If the information will be needed later, a regular variable can copy the value of the temporary variable;

```

SAS Code:  Data MergeSource;
            Merge White (IN=InWt) Yellow(IN=InYel);
            By Id;
            If InWt=1 and InYel=1;
            *Alternate if InWt=InYel;
            WtFileInd=InWt;
            Run;

```

An intermediate internal processing snapshot shows the values of the temporary variables.

ID	GRP	AGE	ELIG	InWt	InYel	ID	GRP	ELIG
1	A	30	Y	1	0			
2	A	40	N	1	0			
3	A	50	N	1	1	3	B	Yes
4	A	60	Y	1	0			
5	A	70		1	1	5	B	No
				0	1	6	B	Yes

Due to the subsetting if statement the observation must be in both the white and the yellow data sets to make the eligibility requirements for the MergeSource data set. The temporary variables InWt and InYel are not in the resulting data set. The problem remains with the second data set overwriting the first dataset.

Resulting Data set

TUTORIALS

ID	GRP	AGE	ELIG	WtFileInd
3	B	50	Yes	1
5	B	70	No	1

MERGE - RENAME Option

An option to avoid some of the overwrite problems is to rename the variables in the merge. The syntax is after the dataset name (rename=(old variable=new variable))

```
SAS Code      Data MergeRen;
                Merge White(rename=(ELIG=WELIG)) Yellow(rename=(ELIG=YELIG));
                By ID;
                Run;
```

The original data set supplies the value to the new variable as long as it was the dataset contributing the observation.

SAS Output

ID	GRP	AGE	WELIG	YELIG
1	A	30	Y	
2	A	40	N	
3	B	50	N	Yes
4	A	60	Y	
5	B	70		No
6	B			Yes

By using the rename option, it is possible to detect the inconsistency in patient #3 data.

MERGE - ONE TO MANY MERGE or MANY TO ONE MERGE

A third major category of merges is the One to Many and the closely related Many to One merges. The syntax is the same as the matched merge. However, it is important to know which data set has the "many" observations and which data set has the "one" observation. The order the data sets are listed in the MERGE statement makes a difference. The logistics of the merge is basically the same. The items in the right data set overwrites the data in the left data set.

```
SAS Code:      Data One2Many;
                Merge white green;
                By id;
                Run;
```

Visualizing the data sets side by side will help show what happens.

White Data Set

ID	GRP	AGE	ELIG
1	A	30	F
2	A	40	M
3	A	50	M
4	A	60	F
5	A	70	

Green Data Set

ID	GRP	TYPE
3	C	a
3	C	b
3	C	c
5	C	b
5	C	c

Results of a One to Many Merge

ID	GRP	AGE	ELIG	TYPE
1	A	30	Y	
2	A	40	N	
3	C	50	N	a
3	C	50		b
3	C	50		c

4	A	60	Y	
5	C	70		b
5	C	70		c

The values in variables AGE and ELIG are retained until the value of the BY GRP variable changes.

If the order is reverse, a Many to One merge results in a different data set.

```
SAS Code:      Data Many2One;
                Merge green white;
                By id;
                Run;
```

Looking at the data sets side by side, recall the data on the right overwrites the data on the left.

Green Data Set

White Data Set

ID	GRP	TYPE	ID	GRP	AGE	ELIG
			1	A	30	Y
			2	A	40	N
3	C	a	3	A	50	N
3	C	b				
3	C	c				
			4	A	60	Y
5	C	b	5	A	70	
5	C	c				

The results of the Many to One Merge

ID	GRP	TYPE	AGE	ELIG
1	A		30	Y
2	A		40	N
3		a	50	N
3	C	b		
3	C	c		
4	A		60	Y
5		b	70	
5	C	c		

A Many to One merge is possible. However, the values of AGE and ELIG were not retained. The Many to One and the One to Many data sets are different. The differences are highlighted.

Results of a One to Many Merge

ID	GRP	AGE	ELIG	TYPE
1	A	30	Y	
2	A	40	N	
3		50	N	a
3	C			b
3	C			c
4	A	60	Y	
5		70		b
5	C	70		c

TUTORIALS

Be aware the simple change in the order the data sets are listed DOES make a difference.

It is import to know your data. Look at the proc contents before performing merges. Print several observations of the original data sets and the merged dataset. Check and make sure you are getting the results you expect.

MERGE - MANY to MANY MERGE - General

The last category is a Many to Many merge. This type of merge prompts regularly recurring questions on SAS-L, a mail serve list for SAS questions. The problem is the same basic syntax does not yield the desired results for a Many to Many situation!

Green Data Set			Red Data Set	
ID	GRP	Type	ID	CAT
3	C	a	3	10
3	C	b	3	20
5	C	c	5	50
5	C	b	5	60
5	C	d		

The usual DESIRED results are a Cartesian cross product with 10 observations.

ID	GRP	TYPE	CAT
3	C	a	10
3	C	b	10
3	C	c	10
3	C	a	20
3	C	b	20
3	C	c	20
5	C	b	50
5	C	c	50
5	C	b	60
5	C	c	60

However, the expected SAS code does NOT produce the above results.

```
SAS Code:      Data Many2ManyERROR;
                Merge red green;
                By Id;
```

Run;

Results

ID	GRP	TYPE	CAT
3	C	a	10
3	C	b	20
3	C	c	20
5	C	b	50
5	C	c	60

Possible Solutions include using a SQL procedure or manipulate the dataset with the POINT command.

MERGE – MANY to MANY USING SQL

The SQL procedure implements the Structured Query Language. Using proc SQL a Cartesian cross product data set can be produced.

```
SAS Code:      Proc SQL;
                Create table manySQL as
                  Select *
                  From green, red
                  Where green.id=red.id
                ;
```

Quit;

Explanation of SQL code

The phrase "Create table many SQL as" creates a data set named manySQL, storing the results of the query expression.

The code "Select *" includes all the variables all of the data sets listed in the next snippet of code. An alternative is to name the variables you want to keep in the data set.
 The names of the source data sets are identified with "From green,red".
 The instructions "Where green.id=red.id" states the condition.

POINT in a MANY to MANY MERGE

Instead of using Proc SQL, it is possible to create the data set in a data step. This is accomplished through accessing observations in one data set sequentially and the observations in the other directly using the POINT= statement.

The vast majority of my work I process data sequentially. That is accessing the observations in the order in which they appear in the physical file. Usually every observation needs to be examined, and processed so sequential access is adequate. When working with small datasets sequential access is not a problem.

On occasion it is advantageous to access observations directly. You can go straight to a particular observation without having to handle all of the observations that come before it in the physical file. The POINT= option with the SET statement will communicate to jump to a particular observation. Suppose you had a large data set with 1 million observations named BigSet and you knew ELIG was missing for some observations in the last 100 observations. Instead of sorting or processing the 999,900 other observations you can go directly to the last 100 observations to identify those with missing ELIG values.

```
SAS Code:      Data FindMissing;
                  Do I = 999900 to 1000000;
                    Set BigSet point=I;
                    If ELIG="" then output;
                  End;
                  Stop;
Run;
```

When you use the POINT= option, a STOP statement must be included. If the STOP is inadvertently left off, a continuous or endless loop occurs. You are pointing to specific observations and SAS never will find/read the end of file indicator.

The Many to Many Merge data step solution using the Point option is given below. The Green data set is being processed sequentially. For each observation in the Green data set each observation in the Red data set is accessed directly in the Do Loop. The values read with the SET statement are automatically retained until another observation read from that data set. So each observation of the Red dataset is paired with the retained observation from the Green data set. If the tempid variable in the Green data set matches the id variable in the Red data set then the observation is outputted to the Cross product data set.

```
Data CrossProduct (drop=tempID);
  Set green(rename=(id=tempID));
  NumInRedSet=4;
  Do i=1 to NumInRedSet;
    Set Red Point=i;
    If tempid=id then output;
  End;
Run;
```

Green Data Set			Red Dataset	
ID	GRP	Type	ID	CAT
3	C	a	3	10
3	C	b	3	20
5	C	c	5	50
5	C	b	5	60
5	C	d		

Tempid	GRP	Type	i	ID	CAT	
3	C	a	1	3	10	Output

TUTORIALS

3	C	a	2	3	20	Output
3	C	a	3	5	50	
3	C	a	4	5	60	
3	C	b	1	3	10	Output
3	C	b	2	3	20	Output
3	C	b	3	5	50	
3	C	b	4	5	60	
3	C	c	1	3	10	Output
3	C	c	2	3	20	Output
3	C	c	3	5	50	
3	C	c	4	5	60	
5	C	b	1	3	10	
5	C	b	2	3	20	
5	C	b	3	5	50	Output
5	C	b	4	5	60	Output
5	C	c	1	3	10	
5	C	c	2	3	20	
5	C	c	3	5	50	Output
5	C	c	4	5	60	Output

Results

ID	GRP	TYPE	CAT
3	C	a	10
3	C	a	20
3	C	b	10
3	C	b	20
3	C	c	10
3	C	c	20
5	C	b	50
5	C	b	60
5	C	c	50
5	C	c	60

CONCLUSION

Trying to locate a particular data element in a large number of small, scattered data sets can be frustrating. Combining data sets with SET and MERGE statements can create data sets which are more comprehensive, cohesive and easier to utilize. The SET statement can be used to copy or append data sets. The MERGE statement used properly pulls together the common data elements for a unit of measurement. Usually a Matched Merge is preferred over a One to One Merge. Using the IN and RENAME options will refine newly created data sets. Many to Many merges are not necessarily intuitive. However, use the proc SQL or the Data Step point examples as templates will get you started on the right path. Combining and manipulating data sets does take a certain amount of skill. But just like tennis, with practice, POINT, SET, and MATCH (Merge), will become part of your winning game set.