

SAS® DATA QUALITY-CLEANSE: TECHNIQUES FOR MERGE/PURGE ON VERY LARGE DATASETS

Michael Krumenaker

**Palisades Research
Bernardsville, New Jersey**

**PhilaSUG
Oct. 27, 2004**

MERGE/PURGE

Combining lists (or cleaning a single list)

and

Removing duplicates

THE PROJECT

Combining 3 lists from different types of sources

Common items: Elements of name, address, and phone

Disparate items: Property and Demographics

Pecking order for –

Different form of name and/or address info

Conflicting phone numbers, property or demographic info

PALISADES RESEARCH, INC.

We provide database and decision support solutions including data warehouse and data mart design and implementation, Internet-enabled information delivery for decision support, and on-line analytical processing (OLAP).

We also provide statistical analysis, data mining, web development for data processing, and list merge/purge development.

Also: Lots of VBA.



FUZZY MATCHING

Fuzzy (inexact) Matching:

Robert Smith = Bob Smith = Robin Smith?

Robert Smith = Robert Smeth?

Criteria: Combinations of factors

Name, street address, zip

Name, phone

How fuzzy is OK (sensitivity)?

False positives vs. Missed matching opportunities

SAS/DQ-Cleanse Works by Match Coding

SAS/DQ and DataFlux create match codes on names, addresses, and other data types.

Items that are fuzzy matches get the same match code (e.g., Robert Smith, Bob Smith).

Only items with the same match code are considered the same.

SAS/DQ does not compare items.

SAS/DQ-Cleanse vs. DataFlux

DataFlux: Interactive. Hands on.

SAS/DQ-Cleanse: Uses DataFlux intelligence (libraries, dfCustomize datatypes and rules). Provides functions and procedures for use in a SAS program, without interactivity.

S...L...O...W...

PREPROCESSING:

Things to do before matching

No clear line between preprocessing and matching steps.

FILES

Break national or regional files down by State. Save time on sorts and matching.

We can do this because we do not expect matches between records in different States.

PREPROCESSING

NAMES: INDIVIDUAL vs. BUSINESS

Use index() or Perl regular expressions in SAS to remove records that are recognizable as businesses, not individuals, by common strings such as Inc (Inc.), Corp (Corp.), etc., before applying SAS/DQ to this task.

Reason: Less for SAS-DQ* to do.

*dqIdentify(input-value,'identification-definition','locale')

PREPROCESSING

FULL NAMES (non-compound)

Full names or multiple names may not always be in the same order

Why break non-compound names down at all? Project requirement: Last name, first name, middle. Also: Householding.

Removing prefixes (Mr. , Mrs., Ms...), suffixes (Jr., Sr....) and titles (Dr., M.D., Capt., D.D.S....): Use SAS index functions or Perl regular expressions in SAS.

PREPROCESSING

SAS-DQ Parsing Functions

dqParse() parses full name, **dqParseTokenGet()** returns each token.

dqParse(input value, 'parse-definition', <locale>)

To parse full names, parse-definition = NAME
Does not return individual tokens

dqParseTokenGet(parsed value, 'token', 'parse definition', '<locale>')

Takes the output of dqParse() and returns the tokens

Example: parsed_value=dqParse(fullname, 'NAME', 'ENUSA')

PREPROCESSING

COMPOUND FULL NAMES

Strings and single characters

SAS/DQ-Cleanse is not always the best way to deal with this.

- **SSIXSSS: Potentially ambiguous - Jones John S & Mary Ann Smith or Jones John S and Smith Mary Ann. But LFM&FML in almost all cases in dataset.**
- **SSIXSSI : Jones John S & Smith Deborah G**
- **SSIXSIS : Jones John S & Mary A Smith**
- **SSIXSS: Ambiguous: Jones John S & Mary Ann or Jones John S & Mary Smith**
- **SSIXSI: Jones John S & Mary A**
- **SSIXS : Jones John A & Mary**
- **SSSX SIS : Ambiguous: Smith Adam Robert & Mary A Jones or Adam Robert Smith & Mary A Jones. But almost 100% of these in dataset have name #1 beginning with last name.**
- **SSIXSSS: Potentially ambiguous - Jones John S & Mary Ann Smith or Jones John S and Smith Mary Ann. But LFM&FML in almost all cases in dataset.**

PREPROCESSING

COMPOUND FULL NAMES (continued)

- **SSIXSSI** : Jones John S & Smith Deborah G
- **SSIXSIS** : Jones John S & Mary A Smith
- **SSIXSS**: Ambiguous: Jones John S & Mary Ann or Jones John S & Mary Smith
- **SSIXSI**: Jones John S & Mary A
- **SSIXS** : Jones John A & Mary
- **SSSX SIS** : Ambiguous: Smith Adam Robert & Mary A Jones or Adam Robert Smith & Mary A Jones. But almost 100% of these in dataset have name #1 beginning with last name.
- **SSSXSSS**: Ambiguous - Robert Daniel Smith & Mary Ann Jones or Smith Robert Daniel and Mary Ann Jones or Smith Robert Daniel and Jones Mary Ann
- **SSS&SS**: Ambiguous - LFMXFM - conflicts with SMSXFL
- **SSSXSI** : Jones John Sam & Mary A OR Jones J Sam & Mary A or Jones J S and Mary A

PREPROCESSING

COMPOUND FULL NAMES (continued)

- **SSIXSS:** Ambiguous - Jones John S & Mary Ann or Jones John S and Mary Smith
- **SSIXSS:** Ambiguous - Jones John S & Mary Ann or Jones John S and Mary Smith
- **SSSXS:** Ambiguous: John Adam Jones and Mary or Jones John Adam and Mary
- **SSXSI:** Ambiguous: John Jones & Mary A Smith or Jones John & Mary A Smith
- **SSXSSS:** Ambiguous: Jones Jones & Mary Ann Smith or John Jones & Mary Ann Smith
- **SSXSS:** John Jones & Mary Smith, OR Jones John & Mary Ann. In dataset, never John Jones & Mary Ann.
- **SSXSI:** Jones John & Mary A
- **SSXS:** Jones John & Mary (never John Jones and Mary in dataset)
- **SISXSI:** Ambiguous - Donald M. Jones and Mary I. Smith or Jones M. Donald and Mary I Smith

PREPROCESSING

COMPOUND FULL NAMES (continued)

- **SIXSSS:** Ambiguous - Billy B Smith & Mary Ann Jones OR Billy B Smith & Jones Mary Ann. Latter is not likely, but SAS parsing may re-scramble if ambiguous. So do not use SAS parsing for this pattern.
- **SIXSS:** Billy B Smith & Mary Jones. No Billy B Smith & Mary Ann.
- **SIXSIS:** Billy B and Mary A Smith
- **SIXSSS:** Billy B and Mary Ann Smith
- **SIXSS:** Billy B and Mary Smith
- **SIXS:** Smith B and Mary
- **SXSIS:** John and Mary A Jones
- **SXSSS:** John and Mary Ann Jones
- **SXSS:** John and Mary Jones
- **SXS:** Not individual: Atlantic and Pacific

PREPROCESSING

COMPOUND FULL NAMES:

Examples using (1) Perl Regular Expression and (2) SAS/DQ-Cleanse

SSIXSSI: LFMXLFM (Jones John S & Smith Deborah G) - Create pattern:

```
prxSSIXSSI=prxparse("/^[a-zA-Z][a-zA-Z]+\s[a-zA-Z][a-zA-Z]+\s[a-zA-Z]\s"|"(&|AND)\s[a-zA-Z][a-zA-Z]+\s[a-zA-Z][a-zA-Z]+\s[a-zA-Z]$/");
```

Using SAS/DQ-Cleanse function for compound names:

```
given_name1=dqparsetokenget(parsed_names,"Given Name 1",  
"Name (Two Names)");  
given_name2=dqparsetokenget(parsed_names,"Given Name 2",  
"Name (Two Names)");
```

PREPROCESSING

ADDRESSES

Get rid of PO Box addresses (Perl)

Remove unit numbers? Business issue!

COMBINE THE LISTS

After preprocessing, combine the three lists into one big list.

Remember, we are doing only one State at a time.

FUZZY MATCHING: SAS DQ-CLEANSE

SAS/DQ-Cleanse does not directly compare items of data (individual names, addresses, other data types).

Instead, it creates a match code each item.

Two items of data need not be exact matches to have the same match code.

Robert Smith and Bob Smith will (probably) have the same match codes.

Robert Smith and Robin Smith MIGHT have the same match code, depending on (1) whether DataFlux thinks Robin is a nickname for Robert, or, if not, (2) how we set the matching “sensitivity”.

Two items of data match only if they have EXACTLY the same match code.

FUZZY MATCHING: SAS DQ-CLEANSE

Choose “sensitivity” for each criterion. The sensitivity determines how many characters of the string SAS uses for fuzzy matching.

Also, SAS DQ-Cleanse draws on libraries for matching. These libraries “know”, e.g.,

- whether a name string (Bob or Smith or Kelly) is more likely (or just as likely) to be a given name or a surname.
- that certain names and nicknames are the same
- that certain combinations of letters sound the same

Rules (phonetic matches, nicknames, number of characters at each level of sensitivity) are customizable using `dfCustomize`. We can (and have) even create new data types. But we could talk about this for DAYS.

OUR MERGE/PURGE CHALLENGES FOR TODAY

1. **Significantly reduce processing time on hundreds of millions of records**
2. **Combine partially overlapping duplicate groups**

COMMON SENSE: REDUCE I/O

After SAS/DQ-Cleanse functionality, including match coding, parsing, tokenizing and standardizing routines, the next biggest time sink is I/O.

Reduce I/O:

The records will go through several data steps and procs before match coding is complete. Each such step or proc means reading all the records in and writing them all (with modifications) back to disk.

Look at a SAS log and compare CPU time to real time!

If after pre-processing we have records with many fields, such as demographic fields, that do not affect matching, strip them off, keep them aside, and re-join them to the match-relevant fields after match coding.

Put the same index number on the “skinny” record and the stripped portion, for re-joining.

CREATING MATCH CODES WITH SAS DQ-CLEANSE

Processor intensive – very slow

Eliminate as much match coding as possible!

For a given token, there are an enormous amount of EXACT matches. Every exactly matching gets the same code (of course).

Tokenize: Parse the name, address, etc., using SAS DQ-Cleanse AS LITTLE AS POSSIBLE.

Separately for each token: Sort and dedup exact matches.

Dramatically reduces the number of tokens SAS-DQ has to match-code.

PARSING STRINGS TO EXTRACT TOKENS WITHOUT SAS-DQ CLEANSE

Is token order known?

For example: A list in which all names are in the order first-middle-last name order, with any applicable prefixes, suffixes and professional designation.

Use elementary Base SAS functions (`scan()`, `indexw()`...) to parse for tokens.

PARSING STRINGS TO EXTRACT TOKENS WITHOUT SAS-DQ CLEANSE

Do the great majority of strings for this field conform to a limited group of patterns?

For example: Street addresses

11 Main Street

11 North Main Street

11 Main Street South

11 Main Street Apt. 3A

11 Main Street South Apt. 3A

To tokenize: Parse by matching the string against pattern templates, using Regular Expressions (SAS 8) or Perl Regular Expressions (SAS 9).

Then sort and dedup.

We reduced the number of strings needing match codes from over 2 million to just under a few tens of thousands.

EXAMPLES OF PERL REGEXP TO TOKENIZE STREET ADDRESSES

Format as in 11 MAIN ST

```
prxAddress1Street = prxparse("/^([0-9][0-9a-zA-Z]*) (\s)([a-zA-Z0-9][a-zA-Z0-9]+) (\s)(STREET|ST|AVENUE|AVE|LANE|LN|PARKWAY|PKWY|WAY|ROAD|RD|DRIVE|DR|PLACE|PL|CIRCLE|CIR|COURT|CT|PIKE|TERRACE|TER|TRAIL|TRL|TL|BOULEVARD|BLVD) $/");
```

Format as in 11 E MAIN ST

```
prxNumPreStreet = prxparse("/^([0-9][0-9a-zA-Z]*) (\s*) (E|EAST|W|WEST|S|SOUTH|N|NORTH) (\s*) ([a-zA-Z0-9][a-zA-Z0-9]+) (\s*) (STREET|ST|AVENUE|AVE|LANE|LN|PARKWAY|PKWY|WAY|ROAD|RD|DRIVE|DR|PLACE|PL|CIRCLE|CIR|COURT|CT|PIKE|TERRACE|TER|TRAIL|TRL|TL|BOULEVARD|BLVD) $/");
```

USING prxposn() TO TOKENIZE AFTER MATCHING TO A prx PATTERN

Each bracketed portion has a position, which is critical.

Format as in 11 MAIN ST

```
prxAddress1Street = prxparse("/^([0-9][0-9a-zA-Z]*)(\s)  
([a-zA-Z0-9][a-zA-Z0-9]+)(\s)(STREET|ST|AVENUE|AVE|LANE|  
LN|PARKWAY|PKWY|WAY|ROAD|RD|DRIVE|DR|PLACE|PL|CIRCLE|CIR|  
COURT|CT|PIKE|TERRACE|TER|TRAIL|TRL|TL|BOULEVARD|BLVD)$/" );
```

Positions:

- 1: ([0-9][0-9a-zA-Z]*)
- 2: (\s)
- 3: ([a-zA-Z0-9][a-zA-Z0-9]+)
- 4: (\s)
- 5: (STREET|ST|AVENUE|AVE|LANE|LN|PARKWAY|PKWY|WAY|ROAD|RD|
DRIVE|DR|PLACE|PL|CIRCLE|CIR|COURT|CT|PIKE|TERRACE|TER|
TRAIL|TRL|TL|BOULEVARD|BLVD)

Example of tokens:

1. 35
3. Main
5. Street

USING THE PARSE RESULTS

Need to create new data type through dfCustomize since we match coded the street name rather than the street full addresses.

Street addresses that we could not tokenize (the small percentage of address strings not matching any of our templates) had to be match coded as full addresses.

STEPS IN OUR MATCH CODING, AND HOW WE APPLIED THESE IDEAS

1. **Bring the three pre-processed lists (for a given State) together, while stripping off the data not involved in match coding. Use an index to rejoin skinny and fat record parts.**
2. **Name match coding, taking advantage of duplicates in whole names and last names, respectively.**
3. **Addresses:**
 - A. **Parse using Perl regular expressions where possible (90%). Tokens:**
 - House number
 - Street name
 - Street type (Road, Avenue, etc.)
 - B. **Street addresses not parsed successfully get the dqparse() treatment.**
 - C. **Using “Street Name” data type we created, do match code on street names, taking advantage of duplications.**
 - D. **House numbers are also deduped (trememndous savings!) and match coded.**
 - E. **Truncated match codes on street names and street numbers are concatenated into a full 15-character match code for the address.**
 - F. **Street types are not used.**

COMBINING DUP GROUPS CREATED USING TWO DIFFERENT SETS OF CRITERIA

Example

Some records match on name-address-zip5 but not name-phone.

Other records match on name-phone but not name-address-zip5.

Some records match on both.

COMBINING ALGORITHM NEEDED FOR OVERLAPPING DUPE GROUPS

Three persons (records)

MaryAnn Sasquatch 11 Maple Street 08711 731-258-0011

Mary Ann Sasquatch 11 Maple Street 08711

Maryan Sasquatch 731-258-0011

We want to put all three in a single “dupe group” and then keep only one record.

Suppose the first two records match (same index1), and form one group.

Suppose the first and third record match (same index2), but form another group, since the second and third records do not match.

COMBINING DUP GROUPS CREATED USING TWO DIFFERENT SETS OF CRITERIA

Obtained match codes for each name and address. Tokens that are “fuzzy” matches have the SAME exact match code.

Used two matching criteria, i.e., concatenated the match codes two different ways:

```
index1 = name_match_code || address_match_code || zip5  
index2 = name_match_code || phone
```

Two records match if their value of index1 or value of index2 EXACTLY match.

zip5 and phone are used exactly as they appear in each record.

COMBINING ALGORITHM NEEDED FOR OVERLAPPING DUP GROUPS

All 6 of these records belong in Same Dup Group,
i.e., they are the same person.

In this example, there is duplication both within list and between lists.

index1	index2	source
M*	X**	List 1
M	X	List 2
N	X	List 3
N	X	List 2
N	Y	List 1
P	Y	List 3

*Actually, 35 chars.

** Actually, 25 chars.

COMBINING ALGORITHM NEEDED FOR OVERLAPPING DUP GROUPS

THREE KINDS OF RECORDS:

1. Records having no duplicates: Each such record is its own dup group. Example:

A 1

where no other record has $\text{index1}=A$ and no other record has $\text{index2}=1$.

2. Records in a dup group based on one index in which no member belongs in a dup group based on the other index. Example:

A 1
A 2

where no other record has $\text{index1}=A$ or $\text{index2}=1$ or 2.

2. Records in a dup group in which one or more members also belongs to another dup group. Example:

A 1
A 2
B 2

COMBINING ALGORITHM – STEP 1

**MAKE IT RUN QUICKER:
SKINNY-UP LONG RECORDS**

WE ALREADY DID THIS, MIKE!

**CAN ALSO STRIP OFF NAME, ADDRESS
AND PHONE INFO IF USEFUL TO DO SO**

COMBINING ALGORITHM – STEP 2

Find dup groups based on index1 only.

Assign common cluster numbers to each such group

skinny dataset: work.try

<u>index1*</u>	<u>index2**</u>	<u>cluster number</u>
A	2	1
E	2	2
E	1	3
B	1	4
C	3	5
F	7	6
F	7	7

*Actually a 35 character string of concatenated match codes for name-address-zip5

**Actually a 25 character string of concatenated match codes for name-phone

COMBINING ALGORITHM – STEP 2

Identify dups on index1 and assign such dups to same clusters.

Add 2 new variables:

cluster_number1 (has unique value for each By group)

dupindex1 (to identify By groups with multiple values)

By groups with multiple records get dupindex1=0

By groups with a single record get dupindex1=1

```
proc sort data=try; by index1; run;
  %let cluster_field=cluster_number;
  data try;
    set try;
    by index1;
    retain &cluster_field.1;
    if first.index1 then &cluster_field.1=&cluster_field.;
    if first.index1.*last.index1=0 then dupindex1=1;          *unique:1;
    else dupindex1.=0;
  run;
```

COMBINING ALGORITHM – STEP 2

Identify duplicates on index1.

&cluster_number.1 (has unique value for each By group)

Dupindex1 (to identify By groups with multiple values)

By groups with multiple records get dupindex1=0

By groups with a single record get dupindex1=1

<u>index1</u>	<u>index2</u>	<u>cluster_number</u>	<u>cluster_number1</u>	<u>dupindex1</u>
A	2	1	1	0
B	1	4	4	0
C	3	5	5	0
E	2	2	2	1
E	1	3	2	1
F	7	6	6	1
F	7	7	6	1

COMBINING ALGORITHM – STEP 3

Remove records that are unique on both index1 & index2

These records are not in overlapping dup groups

Also: Add cluster_number2 to track dup groups on index2.

```
Proc sort data=try;  
  by index2;  
run;
```

```
data nodup(keep=index1 index2 cluster_number counter) dup;  
  set try;  
  by index2;  
  retain &cluster_field.2;  
  if first.index2 then do &cluster_field.2= &cluster_field.1;  
  if first.index2*last.index2=0 then dupindex2=1;  
  else dupindex2=0;  
  if (dupindex1+dupindex2)=0 then output nodup;  
  else output dup;  
  drop dupindex2 dupindex1;  
run;
```

*correction of article

COMBINING ALGORITHM – STEP 3

Only the fifth record has index1=C and only the fifth record has index2=5.
Record #5 is unique on both indexes. It goes into nodup. It's a 1-record group.
Processing continues on dataset dup.

<u>Index1</u>	<u>index2</u>	cluster_number	<u>cluster_number1</u>	<u>cluster_number2</u>	<u>output</u>
B	1	4	4	4	dup
E	1	3	2	4	dup
A	2	1	1	1	dup
E	2	2	2	1	dup
C	3	5	5	5	nodup
F	7	6	6	6	dup
F	7	7	6	6	dup

*based on index2

COMBINING ALGORITHM – STEP 4

Identify redundant dups (de-dup the list of all combinations of index1/index2 by both)

<u>Index1</u>	<u>index2</u>	cluster_number	<u>cluster_number1</u>	<u>cluster_number2*</u>	<u>output</u>
B	1	4	4	4	dup
E	1	3	2	4	dup
A	2	1	1	1	dup
E	2	2	2	1	dup
F	7	6	6	6	dup
F	7	7	6	6	dup

COMBINING ALGORITHM – STEP 4

Removed the second F-7.

```
proc summary data=dup;  
  by index2 index1;  
  output out=temp (keep=index2 index1);  
run;
```

Dataset dup

<u>Index1</u>	<u>index2</u>
B	1
E	1
A	2
E	2
F	7

Result: We eliminate the second F/7 record from dup.
dup goes from 6 records to 5.

COMBINING ALGORITHM – STEP 5

Separate out the “simple” dup groups – dup groups that do not overlap other dup groups.

Identify records with non-unique values for index2 and obtain a list of index1 value for those records.

<u>Dataset dup</u>	
<u>Index1</u>	<u>index2</u>
B	1
E	1
A	2
E	2
F	7 ← unique

COMBINING ALGORITHM – STEP 5

First: Get the set of records for which there are dups on index2.

```
proc sort data=temp;
  by index2 index1;
run;
data select(keep=index1);
  set temp;
  by index2;
  if first.index2*last.index2=0 then output;  *index2 not unique;
run;
```

Dataset select

<u>Index1</u>	<u>index2</u>
B	1
E	1
A	2
E	2

Fifth record in **dup** [F/7] was unique on both indexes, so it does not go into **select**.

COMBINING ALGORITHM – STEP 5

Next: List unique values of index 1 in the remaining records.

Find unique values of index1 in the dataset that has dups on combinations of index1/index2; i.e., a non-duplicating list of such index1's.

Dataset select

<u>Index1</u>	<u>index2</u>
B	1
E	1
A	2
E	2

Fourth record in **select** [E/2] has duplicate of index1 in the second record [E/1].

COMBINING ALGORITHM – STEP 5

Result: Unique values of index1 in the dataset that has dups on combinations of index1/index2; i.e., a non-duplicating list of such index1's.

```
proc sort data=select NODUPKEY;  
  by index1;  
run;
```

Dataset select with only unique value of index1

<u>Index1</u>	<u>index2</u>
B	1
E	1
A	2

Fourth record in **select** [E/2] had duplicate of index1 in the second record [E/1].

COMBINING ALGORITHM – STEP 6

Index1 values that are in dup groups (B, E, A): Which ones are in overlapping dup groups based on I2 and which ones are not?

For such records, identify records whose index1 value are paired with only one value of index2 and those records whose index1 are paired with more than one value of index2.

We previously obtained a list of index1 values in records with non-unique values for index2.

Dataset dup (step 3) contains the six records for which either index1 or index2 or both is not unique.

Dataset select (step 5) is a list of non-unique values for index2.

We separate dup (6 records) into two datasets:

- (1) records having values of index2 that match more than one value of index1 (complex – 4 records)**
- (2) all other records (simple – 2 records).**

COMBINING ALGORITHM – STEP 6

```
proc sort data=dup;
  by index1;
run;
data simple(keep=id index1 index2 &cluster_field.2
  rename=(&cluster_field.2=&cluster_field.)) complex;
  merge dup(in=a) select(in=b );
  by index1;
  if a and b then output complex;
  else output simple;
  drop &cluster_field.;
run;
```

If record is select (in=b): “complex”, record is part of a dup group that overlaps other dup groups on I2.

If record is not in select: “simple”, record is part of a dup group that DOES NOT overlap other dup groups on I2.

COMBINING ALGORITHM – STEP 6

Dataset complex: Not unique, with overlaps between groups

<u>index1</u>	<u>index2</u>	<u>cluster_number</u>	<u>cluster_number1</u>	<u>cluster_number2</u>
A	2	1	1	1
B	1	4	4	4
E	1	3	2	4
E	2	2	2	1

Dataset simple: Non-unique but not overlapping any other group

<u>index1</u>	<u>index2</u>	<u>cluster_number</u>
F	7	6
F	7	6

COMBINING ALGORITHM – STEP 7

The iterative process of collapsing dupe groups that overlap into non-overlapping dupe groups.

Combine overlapping dup groups by collapsing dataset complex until the number of dup groups does not change from one iteration to the next.

The iterative process is controlled by a %do %while loop.

Each of the macro variables &d1, &d2 and &d3, which identify cluster field number macro variables, is incremented by +1 in each iteration

COMBINING ALGORITHM – STEP 7

```
%macro dup_iterate(complex);
  %let d1=0;
  %let d2=1;
  %let d3=2;
  %let groupcnt1=1;
  %let groupcnt0=0;
  %do %while(&&groupcnt&d2 ne &&groupcnt&d1);
    %let d1=%eval(&d1+1);
    %let d2=%eval(&d2+1);
    %let d3=%eval(&d3+1); /*new cluster field - assignment is critical;*/

    proc sort data=&complex; by &cluster_field.&d1; run;
    data &complex;
      set &complex end=EOF;
      by &cluster_field.&d1; /*number of such by-groups decreases;*/
      retain &cluster_field.&d3;
      if first.&cluster_field.&d1 then do;
        &cluster_field.&d3=&cluster_field.&d2;
        groupcnt+1;
      end;
      if EOF then call symput("groupcnt&d2",left(trim(groupcnt)));
      drop groupcnt ;
    run;
  %end;
```

COMBINING ALGORITHM – STEP 7

Starting Point: complex

<u>index1</u>	<u>index2</u>	<u>cluster_number</u>	<u>cluster_number1</u>	<u>cluster_number2</u>
			[sort]	
A	2	1	1	1
B	1	4	4	4
E	1	3	2	4
E	2	2	2	1

complex After First Iteration of the Macro

<u>index1</u>	<u>index2</u>	<u>cluster_number1</u>	<u>cluster_number2</u>	<u>cluster_number3</u>
			[sort next]	[=cf2 if first cf1]
A	2	1	1	1
E	1	2	4	4
E	2	2	1	4
B	1	4	4	4

groupcount=3, i.e., the number of unique values in cluster_number1 is 3.

COMBINING ALGORITHM – STEP 7

complex After Second Iteration of the Macro

<u>index1</u>	<u>index2</u>	<u>cluster_number2</u>	<u>cluster_number3</u> [sort next]	<u>cluster_number4</u> [=cf3 if first cf2]
A	2	1	1	1
E	1	1	4	1
E	2	4	4	4
B	1	4	4	4

groupcount=2, i.e., the number of unique values in cluster_number2 is 2.

complex After Third Iteration of the Macro

<u>index1</u>	<u>index2</u>	<u>cluster_number3</u>	<u>cluster_number4</u>	<u>cluster_number5</u> [=cf4 if first cf3]
A	2	1	1	1
E	1	4	1	1
E	2	4	4	1
B	1	4	4	1

groupcount=2, i.e., the number of unique values in cluster_number3 is 2, which is the same as the previous iteration, so iteration **STOPS**.

COMBINING ALGORITHM – STEP 7

We then gather together all the records by –

- start with dataset complex (step we just did)
- appending the dataset simple to complex (non-overlapping groups)
- appending the records from nodup (1-record groups)

```
*Create final clusterfield variable;  
data &complex;  
set &complex(keep=index1 index2 id &cluster_field.&d3  
  rename=(&cluster_field.&d3.=&cluster_field.));  
run;  
  
*Append simple to complex;  
proc datasets nolist library=work;  
  append base=&complex data=simple(keep=&key_field1.  
  Index2 &cluster_field. id);  
run;  
  
*Append nodup to complex;  
proc datasets nolist library=work; append base=&complex data=nodup; run;
```

COMBINING ALGORITHM – STEP 7

Re-combine the “skinny” set (that we have processed) with the set containing the fields we did not need for this procedure.

```
*** Re-assemble dataset with all details (input set) ***;  
proc sort data=&complex; by id; run;  
data mylib.out_data;  
    merge &source_data &complex;  
run;  
%mend;  
%dup_iterate(complex);
```

TRY STRING COMPARISON FUNCTIONS

Compged()

Complex()

Compare()

Soundex()

Speedis()

Palisades Research, Inc.

75 Claremont Road
Bernardsville, NJ 07924

Phone: 908-953-8081

Fax: 908-953-8096

Email: mkrumenaker@palisadesresearch.com
gbukhbinder@palisadesresearch.com

Try the on-line OLAP demos:

<http://www.palisadesresearch.com>

Presentation: <http://www.palisadesresearch.com/sug>