SESUG 2022 Paper 148

Finding Duplicate Names and Addresses in Your Consumer Database

Joe DeShon, Boehringer Ingelheim Animal Health

ABSTRACT

Despite the best efforts of all, duplicates are certain to be created in any consumer database. Because of variations in spelling and formatting of names and addresses, simply processing the database with PROC SORT NODUPKEY is not enough. Many vendors offer services to help de-dupe such databases, but the process can be very inflexible and expensive. This paper describes a system written in Base SAS which identifies the most important parts of each name and address and thus produces a de-duped database with results comparable to most outsourced solutions. This is appropriate for any company with a large consumer database and limited resources needing to detect and respond to duplicate names and addresses in a consumer database.

HOW DUPLICATES MAY BE INTRODUCED

Duplicates can be introduced into a database in several ways, usually by accident, but sometimes on purpose.

It may be desirable to introduce duplicates if a particular customer fulfils two different roles. For example, an individual may be both a customer and a vendor, and require separate records, one for each role.

A customer may be duplicated if *something* has changed with the customer – such as a credit rating – which requires that history be kept separately for two instances of a customer. The preferred way to handle such *on purpose* cases will vary greatly from one company to the next.

Accidental duplicates are usually created while setting up a new customer without realizing that customer is already in the database. This may be done by an inexperienced employee or by an employee who didn't do a deep enough search to be aware that the customer was already there.

Duplicate records may be created if customers are allowed to "self-enroll" online. Customers rarely have any incentive (or ability) to check first to see if their own record already exists. They may even believe that it's to their advantage to have a duplicate version of their record on file. It may be best to allow the duplicate record to be created in real time so it can be de-duped later in a batch mode.

Duplicate records can also be created if records are obtained from multiple sources, such as when outside mailing lists are purchased.

WHY NODUPKEY AND DUPOUT= OPTIONS ARE NOT GOOD ENOUGH

One might be tempted to eliminate duplicates by simply building a de-dupe key and then performing a PROC SORT with NODUPKEY and/or a DUPOUT= option. However, this is rarely sufficient.

Consider the following examples:

```
Jim Patersson     123 Main Parkway  Mytown    State  12345-6789
James Patterson   123 Main Pkwy     Anytown   State  12345-6789
```

The following should be intuitively obvious:

"Jim" and "James" are probably synonyms for the same person.

"Patersson" and "Patterson" are certainly the same surname, varying only in the double letters. It is impossible to tell which one is correct, although, given the identical address, they are certainly the same family.

"Pkwy" is an abbreviation for "Parkway".

"Mytown" and "Anytown" may be the same location, especially since the ZIP+4s are identical. (They are probably neighboring cities that share the same post office, or perhaps one is a suburb of the other.)

If a de-dupe key were constructed from these two addresses without any pre-processing, they would not be de-duped.

Therefore, it is important to do pre-processing of the data before creating any de-duping key. The exact processing required will vary. This paper breaks it into the following three steps:

- Standardize characters
- Mitigate potential for typos and variant spellings
- Determine the most important parts of the keys

Important note:

This paper assumes a *consumer* (not B2B) set of data where the names are in separate first_name/last_name fields. There are several methods for parsing first_name/last_name from full_name fields, but that is beyond the scope of this paper.

Also, the only fields that are needed for this de-duping process are the last name (assumed to be the family name), the street address, and the ZIP code.

STANDARDIZE CHARACTERS

The first step is to *simplify* the data. This makes sure that any minor differences in character sets, punctuation, or styles are mitigated.

First, upper-case all characters and remove any leading or trailing spaces.

```
if last_name ne ' ' then work_last = strip(upcase(last_name));
work_address = strip(upcase(street));
work_zip     = zip_code;
```

Notice that the first step establishes work_ variables to preserve the original variable values.

Next, use the BASECHAR function to convert any accented or diacritic characters to their base form. This guarantees, for example, that "GARCIA" and "GARCÍA" will be treated as the same.

```
work_last    = basechar(work_last);
work_address = basechar(work_address);
```

It's important to note that the intent is NOT to *correct* the string, rather, to *simplify* it. The process does not know or care whether "GARCIA" or "GARCÍA" is correct, only that by simplifying the string, there is a greater chance that they will match.

MITIGATE PROBLEMS WITH TYPOS AND VARIANT SPELLINGS

Most typos and variant spellings occur when a letter is doubled (or NOT doubled) when it is not supposed to be (or IS supposed to be). For that reason, many typos can be detected simply by removing all doubled letters. (Remember, the intent is to *detect duplicates*, not to correct the spelling.)

```
work_last = tranwrd(work_last,'AA','A');
work_last = tranwrd(work_last,'BB','B');
work_last = tranwrd(work_last,'CC','C');
...
work_last = tranwrd(work_last,'ZZ','Z');


work_address = tranwrd(work_address,'AA','A');
work_address = tranwrd(work_address,'BB','B');
work_address = tranwrd(work_address,'CC','C');
...
work_address = tranwrd(work_address,'ZZ','Z');
```

DETERMINE THE MOST IMPORTANT PARTS OF THE KEY

The next step is to build a key that will be used for deduping. First, build a key containing all the variables. This won't be used as such but will force the syntax to create a variable long enough to contain all the values. The actual key will be much shorter:

```
dedup_key =
  last_name  ||
  street     ||
  zip_code
  ;
```

We could stop there and just use the name, address, and ZIP for a de-dupe key. But that wouldn't find many potential duplicates that have a slightly variant spelling.

Instead, this paper suggests to further reduce the fields to only their most significant values.

ZIP CODE

Of all the components in an address, the ZIP code is by far the most accurate. It is short (only five or nine characters), and it is entirely numeric (ignoring the dash). For those reasons, there are simply fewer opportunities to get it wrong.

The best quality of the ZIP code is that it is completely non-ambiguous. A single ZIP code represents a finite set of delivery destinations. No locations beyond those delivery destinations can have the same ZIP code, and all the locations within that set of delivery destinations have that exact same ZIP code.

The ZIP code is considered *so accurate* that the US Postal Service completely ignores the city name when sorting long-distance mail. (City names are notorious for variant spellings and irregular borders.) The first sort of all mail by the USPS is always the ZIP code (or, more specifically, the *first three bytes* of the ZIP code).

LAST NAME

This process assumes that two households must have the *same last name* (at least, within certain "fuzziness" allowances such as double letters) to be considered duplicates.

Since so many women choose to retain their previous surname after marriage, it is more common than ever to have a household with multiple surnames. This could spark a discussion on the wisdom of *splitting* (rather, *NOT combining*) such a household into two different entities.

However, it is important to realize that the raw data has no **context**. If two records have the exact same address with different last names, there is no way of knowing which is the *correct* last name, or whether it is appropriate to combine them.

If two people have married but retained separate last names, they may have chosen to keep separate identities – perhaps even separate bank accounts and separate lives.

Therefore, it is best to play it safe and keep the records with different last names separate.

STREET ADDRESS (PARTIALLY)

The street address, of course, is the single piece of data that determines the exact delivery location for mail. But not all parts of the street address are created equally. And not all parts are equally significant when determining whether it should be included in a de-duping key.

A street address usually consists of three to five characters of a house number, followed by an optional directional indicator, a street name, and a street designator (such as "Street" or "Avenue").

Of all the components, the most important – and most accurate – part of any street address is the house number. Even if two street addresses vary by only one digit, if the ZIP code and family name agree, there is a very high likelihood that the two are, indeed, two separate households that should not be combined. They are probably two family members living across the street or next door to each other.

But, as we go farther to the *right* in the address, two things happen:

First, the address becomes less accurate. The more characters there are, the more opportunities exist to make typos and other mistakes. Variant spelling of street names usually begins a few letters *into* the name.

Second, each succeeding character becomes less relevant. Most street names are unique after the first three or four characters. All the characters beyond that are potentially ambiguous noise.

In fact, this is also true of the *last name*; only the first few characters are needed for de-duping purposes. The rest are noise.

This process exploits those facts. Knowing this, we can now create relatively small de-duping keys, which are *more accurate* and which *process faster*, than if we had used the entire address as a key.

THE SAS CODE TO BUILD THE DE-DUPE KEY

Now that we know the relevant parts of the address, we are ready to build a key that can be used to de-dupe the entire file.

We have already built an initial version of the key, just to set the maximum length and avoid truncation errors.

Let's review the components that will make up the key:

- Last name (only the left-most characters and after eliminating double characters)
- Street address (only the left-most characters and after eliminating double characters)
- ZIP code (first five bytes)

First, keep only the first five bytes of the ZIP code:

```
work_zip = subpad(work_zip,1,5);
```

Next, prepare the street address further by left justifying it and removing all spaces. Then remove all other characters except letters and digits:

```
work_address = strip(compress(work_address,,'kan'));
```

With the address in this state, create another work variable, which we'll need a little later:

```
work_addressx = work_address;
```

Change all the remaining spaces to asterisks. This isn't necessary, but it can make later debugging easier. Then keep only the leftmost seven digits. This will be enough for three to five house numbers plus a few letters of the street directional and name. Our tests have shown that seven is a reasonable number which gives almost no false positives; it is easy to experiment with this number and adjust it for your data as necessary:

```
work_address = tranwrd(work_address,' ','*');
work_address = subpad(work_address,1,7);
```

If a key were created with just the above three components, it would be very effective in determining duplicates, but it would miss one special case – those with an apartment number at the end of the address line, thus:

```
Jim Smith   123 Main Street, Apt. 456
Mary Smith  123 Main Street, Apt. 789
```

This format violates one of the assumptions made so far – that the characters in the address become increasingly more irrelevant and inaccurate the further to the right they are. If "123 Main Street" is an apartment building, obviously you wouldn't want to combine Jim's and Mary's records because they live on separate floors in the same building but only coincidentally have the same last name.

The following code solves this problem. It considers *numbers* as significant *only* if they are at the *end* of the street address.

First, reverse the address to put the *ending* characters at the *beginning*. Then keep only the digits 1-9 (with the "KD" option of the COMPRESS function). Left justify everything, replace any leftover spaces with asterisks for ease of debugging, and keep only the first seven digits:

```
work_addressx = reverse(work_addressx);
work_addressx = compress(work_addressx,,"kd");
work_addressx = strip(work_addressx);
work_addressx = tranwrd(work_addressx,' ','*');
work_addressx = subpad(work_addressx,1,7);
```

Finally, we can build our de-dupe key by combining all the pieces that we have assembled so far:

```
dedup_key =
  substr(work_last,1,5)      ||
  substr(work_address,1,7)   ||
  substr(work_addressx,1,7)  ||
  work_zip;
```

PROCESS THE FILE APPROPRIATELY

Now that the de-dup key has been created, the file can be sorted by the key and processed according to your needs.

Although it is possible to dedupe the file with NODUPKEY and DUPOUT= options, we prefer to do the processing through a data step for greater flexibility:

```
data work02;
  set work01;
  by dedup_key;
  if not (first.dedup_key and last.dedup_key) then do;
    /** Do appropriate action for duplicates here.  **/
    end;
```

Notice that the NOT modifier means the parentheses are required for the logic to work correctly.

POSSIBLE IMPROVEMENTS

This paper presents only the most fundamental aspects of creating a de-dupe key and processing it. There are many possible improvements and enhancements including:

Including a "tie-breaker" to keep or delete some duplicates based on some other criteria. For example, in the case of a tie, you may want to keep the one that is already a customer or keep the one with the most purchases.

Further pre-processing of name strings is possible, such as standardizing "ST", "STREET", "SAINT(E)" or "STATE".

Removing the space after "MC" and "MAC" for consistency.

Removing (or other processing) records that would cause over-householding such as "N/A", "NO NAME", "NULL", or "UNKNOWN".

Processing variant forms of streets with ordinal names, such as "Second Street" and "2nd Street".

Processing variant forms of street directionals such as "N" and "North"

Processing other formats of address lines such as post-office boxes or rural route addresses.

CONCLUSION

Managing messy consumer data is not easy, but you don't need expensive and difficult-to-configure off-the-shelf tools to do it. The powerful string-manipulating capabilities of Base SAS is more than adequate to handle the most challenging customer data issues that you might have.

CONTACT INFORMATION

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

Joe DeShon
816-210-0950
joedeshon@yahoo.com