Data Cleaning and Deduplication

By David Adams

Technical Note 06-20

Overview

Databases almost inevitably come to include duplicate records. As a common example, people and address tables often include duplicates for a combination of reasons:

- Data entry adds typos, transcription errors, or other mechanical mistakes.
- People change names or addresses and the system doesn't recognize the change as a modification, so a new record is added.
- Data from different systems is combined imperfectly.
- Different name and address formats are used at different times, preventing the system from recognizing duplicates.

These, and other problems, predictably contribute to a decay in data quality. Fortunately for us as 4th Dimension developers, we're not alone in trying to address this problem. Virtually everyone maintaining a database, be it in 4th Dimension, FileMaker, SQL Server, MySQL, DB2, or Oracle, has to contend with the same issues. In fact, the larger the database, the more important data quality becomes. Because this is a general, industry-wide problem, there is a wealth of best practices, techniques, and research for us to draw on. This technical note reviews the basic steps involved in data cleaning and deduplication, offers specific links and implementation suggestions, and describes how the components and sample code in the FuzzyTools demonstration database can be applied to data deduplication projects. Before looking at any of the details, we should review why improving data quality matters.

Who Cares About Bad Data?

This may seem like a senseless question because everyone cares about bad data, in theory. In practice, improving data quality requires time, effort, and money. It needn't be as hard or costly as people might fear, but it's not free. Rephrasing the initial question puts the problem into perspective:

> How much are you willing to invest in improving data quality?

To justify work and expense, there must be tangible benefits. Some common benefits are listed below:

- Increasing data accuracy improves the accuracy of any reports derived from that data. Axiomatically, a report can never be more accurate than its least accurate data.
- Reducing duplicate addresses can likewise reduce mailing costs.
- Consolidating related customer records can improve a business's ability to provide good service and targeted promotions.
- Reducing duplicates can make lookups more efficient and improve productivity.
- Improving data quality simplifies linking records with external systems. Multi-system reporting and reconciliation is a legal requirement for a variety of systems.

While a compelling business case can't always be made for cleaning data, it's often not hard to justify.

A One-Minute Summary of Record Matching

Data deduplication is not that involved conceptually, usually falling into the following steps:

1) **Cleaning and standardization**, such as making sure addresses all use common abbreviations.
2) **Querying** to reduce the number of relevant records at any one time to a manageable number.
3) **Matching** potentially duplicate records.
4) **Combining** true duplicate values.

We'll review these steps in more detail throughout this technical note.

Included and Related Materials

This technical note provides an overview of some data cleaning and deduplication techniques, with special emphasis on the fuzzy matching tools described in Technical Note 06-19 **The FuzzyTools Component**. The sample database included with this technical note is also documented in The FuzzyTools Component technical note. The source code for the FuzzyTools system is documented in the Technical Note 06-18 **Fuzzy Matching in 4th Dimension**.

Data Cleaning and Standardization

It is much easier to find duplicates in clean, consistent data than in inconsistent data. For example, consider these two (fictitious) addresses:

18 Wayland Center, PMB 301 Wayland, MA 01178	18 Wayland Ctr PMB 301 Wayland, Mass 01778

It's easy for a human to see that the addresses are the same but not so easy for software. The fuzzy matching and word list tools included in the sample database can do a good job of recognizing the similarities between these two addresses, but why make them work harder than necessary? Instead, the data should be standardized before any duplicate hunting takes place. Below the two addresses shown above are presented in the correct format, according to the USPS (United States Postal Service):

18 WAYLAND CTR
PMB 301
WAYLAND, MA 01778-1005

Address Standardization

While regularizing addresses is only one example of data standardization, it's an important one. Address data seems to be particularly prone to errors, either from data entry mistakes or not getting updated when people or businesses move. Fortunately, there are many resources and tools available to assist with data cleanup. In the case of address data, the postal service of each country defines abbreviations, field layouts, required elements, and other details of a standard address. There are also numerous commercial products available for data cleanup, either through integration with a desktop application or by remote Web Service calls. Some postal services also offer free software or Web Services. For example, the USPS provides free XML-based remote services for address verification. For more information, visit this page:

http://www.usps.com/webtools/technical.htm

UPS (United Parcel Service) offers its own free address validation system, as well:

http://www.ups.com/e_comm_access/gettools_index?loc=en_US

We'll review some commercial address cleaning options in a moment but, for now, let's address the topic of creating your own data standardization code.

Note *The USPS Web Service mentioned above use XML messages passed as long URLs instead of SOAP messages. Within 4D, you can use URL-based services either by calling* `cURL` *through* **LAUNCH EXTERNAL PROCESS** *or by writing some simple HTTP* `GET` *code with 4D Internet Commands.*

Rolling Your Own Data Standardization Code

In some systems, or portions of some system, you may be able to build your own data standardization system. If you maintain standard abbreviations, for example, it can be relatively straightforward to maintain them. For help with this process, check the *WordList_GetWordsFromText* method. This routine takes a block of text and parses it into a list of words. Then you can step through the array of words, replacing them with abbreviations when appropriate. After the array has been processed, reassemble the completed address. Note that this strategy is more laborious that simply calling **Replace string** repeatedly but avoids **Replace string** matching partial words. For abbreviations, you typically want to work with full words at a time.

Warning: Address Standardization Schemes Are Complex

The complete details of address standardization can be quite complex. For example, the USPS's **Publication 28 - Postal Addressing Standards** is 136 pages long. There are seven pages of English street name words alone. There are a variety of special cases that make simple approaches, like blind word replacements, inadequate. For example, consider case 233.23 on addresses that contain two directionals:

> When two directional words appear consecutively as one or two words, before the street name or following the street name or suffix, then the two words become either the pre- or the postdirectionals. Exceptions are any combinations of *NORTH-SOUTH* or *EAST-WEST* as consecutive words. In these cases the second directional becomes part of the primary name and is spelled out completely in the primary name field.

If this rule strikes you as dense and confusing it's because this rule *is* dense and confusing. If possible, find a postal-certified address cleaning and validation system to integrate rather than rolling your own. If you do implement your own system, proceed conservatively and check your outputs against known valid results. For example, you can check an address at a time at the USPS site on this page:

> http://zip4.usps.com/zip4/welcome.jsp

Commercial Address Standardization Software

Commercial address cleaning software has been around for decades and is only getting better and cheaper. Below are a few suggestions but, given this is a fast-moving and competitive software category, you should check for current prices and options. Try, for example, googling any of these word combinations:

> address cleanup software
> address standardization software
> address validation software

PostCode Anywhere
http://www.postcodeanywhere.co.uk/

PostCode Anywhere is a UK-based system that offers a wide range of address verification and data cleaning products and services. They provide desktop, Web, and Web Service interfaces. Services can be licensed on a pay-as-you go approach or for unlimited use at a particular location. Their Web site and range of services are fast, well-organized, and impressive.

4th Dimension developer Balinder Walla generously provides, for free, a 4th Dimension component that simplifies integrating with PostCode Anywhere:

> http://www.developer-source.org/4d/

Once you register for a PostCode Anywhere account, you can experiment using Balinder Walla's sample database or any other tools you prefer. When you register, PostCode Anywhere provides several free lookup credits to give you a chance to try the service out.

AddressDoctor
http://www.addressdoctor.com/

AddressDoctor claims to be able to correct address data for 240 countries and territories. It appears to be a mature and powerful product and is embedded in several other popular address-cleaning related products. Currently, the pay-as-you go pricing structure for AddressDoctor is relatively high but, given how quickly this field is changing, it's worth checking current figures.

DesertSoft
http://www.desertsoft.net/

DesertSoft offers a straightforward series of address cleaning related services at a considerably lower per-transaction price than AddressDoctor.

Again, keep in mind that prices change, software changes names, and products come on and off the market. The links above are intended as a starting point, but you should double-check current prices and offerings before committing to a package.

Querying to Reduce the Number of Records

Overview

Ultimately, finding duplicate records involves comparing lots of records. If you have a table with 10,000 records, it takes roughly 50,000,000 record comparisons to test exhaustively for duplicates. Expressed as a formula, the number of record-pairs that require comparison is equal to:

(records in selection * records in selection) / 2

In fact, the number is slightly lower than this (49,995,000) and the formula slightly more complex. We'll look at calculating the number of comparisons in more detail later but, for now, it should already be clear that as the number of records in the selection increases, the number of record-pair comparisons needed rises almost exponentially. The corollary of this principle is that reducing the number of records in the selection also decreases the number of comparisons nearly exponentially. Reducing the number of records in the starting selection is often critical to making duplicate hunting practical in the real world.

Note *In the traditional data cleaning and reconciliation literature, reducing the initial selection is called "blocking" because you create a "block" of records.*

Strategies for Reducing the Starting Selection

Before performing a duplicate-hunt, there are many ways to reduce the starting selection. A sound strategy is to first search on a field that you know has relatively reliable data. For example, if a gender field in a patient database is nearly always entered correctly, first search by gender. This approach quickly eliminates many records from consideration without much reducing the chance of finding true duplicates. The exact rules depend on your data, but some more examples may offer food for thought:

- Find all addresses within the same ZIP Code.
- Find all addresses within a ZIP Code and all of its neighboring ZIP Codes. (This requires a ZIP Code data file that lists adjacent ZIP Codes.)
- Find all patients with birthdates within the same week.
- Find all patients with identical birthdates, or birthdate entered with day and month reversed, if legal. (For example, match 03/04/62 with 04/03/62.)
- Reduce the selection randomly to some percentage of the records in the table.

Pros and Cons of Reducing the Starting Selection

Reducing the starting selection is clearly a trade-off because it is possible you'll exclude duplicates from consideration. The benefit is that the duplicate hunting process is a great deal faster. If you can do more duplicate hunts on smaller selections, you may end up with better results overall. This strategy also gives you the chance to learn which of your duplicate hunting rules find the most true positives. If you wondered about the example, *Reduce the selection randomly to some percentage of the records in the table*, it isn't as strange as it sounds. If you don't know how high/low in quality your data is, it can be helpful to do a very thorough set of duplicate checks. If the table is large, you may learn more from performing several tests on 5% of the records than you can from a single test on 100% of the records.

Matching Duplicate Records

Matching potentially duplicate records can be a challenge but it's also both interesting and rewarding. Duplicate matching strategies can be as simple as scanning through records looking for exact duplicates and as complex as using bipartite graph theory (whatever that is), machine learning, and neural networks. Historically, 4th Dimension developers have largely been limited to relatively simple matching schemes, such as:

- Scanning for exact duplicates.
- Scanning for near duplicates based on substrings, such as finding contacts where the first four characters of the last name match and the ZIP/post code is the same.
- Scanning for near duplicates based on common soundex codes.

These techniques are a vast improvement over not checking for duplicates at all, but they can be improved considerably. The FuzzyTools component, included with this

technical note and used in the accompanying sample database, offers a wide range of tools to enhance fuzzy/approximate string comparisons:

- Four variants of the Soundex phonetic transcription system.
- The Metaphone phonetic transcription system, a more accurate alternative to Soundex.
- Four statistical string similarity estimating algorithms developed by the US Census and tested against databases with billions of records.
- The edit distance (Levenshtein distance) algorithm, commonly used in spell-checkers to identify and sort spelling correction suggestions.
- The Longest Common Subsequence (LCS) algorithm, widely used in DNA analysis to identify similar gene sequences and estimate the degree of similarity between DNA strings.

While some of these techniques sound a bit impressive, they're all easy to use. For example, the code fragment below shows what's needed to find the edit distance between two strings expressed as a percentage:

C_REAL($edit_distance)
$edit_distance:=*Fuzzy_GetDistancePercentage* ("Edit";"Smithe";"Smythe")

Next, we'll look at some strategies for apply fuzzy matching tools to duplicate matching problems. Before getting into more details, let's review an example.

Note *See Technical Note 06-19* **The Fuzzy Tools Component** *for documentation on FuzzyTools routines and Technical Note 06-18* **Fuzzy Matching in 4th Dimension** *for more details on the algorithms used.*

Duplicates Report Demonstration

If you want to get a feeling for duplicate matching in action, open the Show People demonstration in the sample database. The [Person] table includes 500 records of which 50 pairs (100 records) are duplicates. Press the Report Duplicates button to start the duplicate comparison process. When the duplicate checking code is finished, a results window appears with a textual summary of the comparisons performed and any duplicates found. A sample of the report's output, formatted for clarity, is shown below:

```
------------------------------------------------------------
Possible Duplicates Report
------------------------------------------------------------

Seconds taken:                  321
Records tested:                 500
Record comparisons:             124,750
Possible duplicate pairs:       53

Values are sorted by similarity weight.

A Note About WordList Scores:
When a possible duplicate is found, a WordList comparison percentage is
calculated and included in the results for reference. In this example, the score
is derived from the percentage of common words in the combined values of the
First_Name, Last_Name, Street, and ZIP_Code fields.

Possible duplicate record pair #1
Overall similarity score:       165
WordList similarity score:      100.00%

ID:             246                         218
First:          Sawyer                      Sawyer
Last:           McDonald                    McDonald
Street:         201 Lake Rd.                201 Lake Rd.
Zip:            19859                       19859
Phone:          369-601-7741                369-601-7741
Email:          Sawyer.McDonald@hotmail.com Sawyer.McDonald@hotmail.cOm
```

Tip *Remember that comparing 500 records can take some time, so run the demonstration compiled.*

Comparison Rules Table

The fuzzy duplicate matching system assigns similarity weights to record pairs. The weights are the combined totals of many comparisons. For example:

- The edit distance between first names.
- The edit distance between last names.
- The edit distance between Metaphone codes built for last names.

Within any comparison, different weighting can be assigned, depending on the degree of closeness and the significance of the field. For example, matches on last names are generally more meaningful than matches on first names. The rules for the sample report are all hard-coded into the *Demo_DuplicatesReportRun* method.

Hard-coding is a poor way to implement this kind of behavior as the rules are really data. The example is written in this manner as it makes the code simple and easy to follow. The imaginary rule table below illustrates conceptually what a data-driven rule table might contain:

Field	Match	Rank
Address	All fields identical	100
	Street name Lynch weight is 95% or higher	90
	Street name Metaphone4 match and City and State match	80
	City and State match exactly	70
	City edit distances less or equal to 2	60
	Country doesn't match	-20
	Any other address configuration	0
Name	All Fields identical	100
	Surname Lynch weight is 95% or higher	80
	Surname Metaphone4 match and first name match	70
	Surname Lynch weight is 95% or higher and first name Lynch weight is 85% or higher	50
	No match on any field	-100
Date of Birth	Identical	80
	Any 1 component (day, month or year) +/-1	30
	Within 18 months	10
	Difference between 5 and 10 years	-30
	Difference > 8 years	-80
	Any other date of birth configuration	0

You can develop rules that are a great deal simpler or a great deal more complex than those shown above. Whatever rules you end up using, instead of depending on one type of comparison, have several. The FuzzyTools component includes many phonetic and string comparison tools, and you can try them all.

Notice that the imaginary rules above assign both positive and negative scores. In the imaginary example above, birth dates more than 8 years apart suggest the records describe completely different people so 80 points are deducted from the overall score. The principle of using negative scoring should not, however, be overlooked. They are as simple to define as positive scores and at least as useful.

Building a Proper Rule Set System

Ultimately, it is ideal to have rule tables defined in data and a GUI for editing the data. Unless more advanced probabilistic and machine learning techniques are available, rule sets need to be tweaked by hand to reduce false positives and missed matches. With a data-driven rule set system, it is possible to automatically create sample rule sets and permutations of a given rule set. This technique is outside of the scope of this technical note but, if you are contemplating such a system, please feel free to write me at dpadams@island-data.com to exchange ideas about possible designs.

Make Data Quality Improvement an Ongoing Process

If you are undertaking a data quality improvement project, make the most of it by aiming for continuous improvement. Here are some suggestions:

- Do a thorough audit of the data before changing anything to develop an estimate of the degree of duplication currently in the system. You can use fuzzy matching rule sets for this process. This audit also produces a baseline value for estimating the number of true matches a rule set should find.
- For any particular rule set, use human reviewers to verify the data. Record how many false positives there are. You can use the baseline audit results to estimate how many true positives were missed. Over time, measure how much better/worse your rule sets are performing.
- Try several rules sets on a fraction of the records in the table. This step speeds up testing and comparing different rule sets.
- Over time, remeasure the degree of duplication to test if the situation is improving or decaying again.
- Accept that it is sometimes impossible to develop matching rules that don't result in false positives without also missing true duplicates. Over time, you may discover that a rough percentage of false positives can be tolerated.
- Run reports on new data nightly to train users not to create duplicates. We'll expand on this important point more, next.

Running Duplicate Checks, Training Users, and Learning from Users

Although it is ideal to avoid duplicates during data entry, it's not always possible. Also, full duplicate checking can be a lengthy process. Therefore, it is commonplace to run duplicate-checking sweeps at off-peak periods, such as at night or on weekends. When possible, run duplicate checks every night that test any records added during the day. If users entered duplicates, review the mistakes with them while their memory is still fresh. If you wait days or weeks to show the correction, no real training or improvement should be expected. (A user might just end up feeling wrong without really learning anything.) Over time, you'll very likely find that not only do the people entering records improve, they also get much savvier about where duplicates come from, how to find them, and, just as important, how to avoid false positives. Listen to these observations and use them to improve the duplicate matching code. Likewise, you may also discover that users aren't really to blame. Instead, it may be that something about the system makes it too slow or difficult to accurately avoid duplicates.

Look at the Data When Building Rule Sets

When you're developing duplicate checking rule sets, it really helps to see what problems are causing duplicates in your data. For example, in the sample database, several of the duplicates suffer from first/last name swaps, such as the pair below (highlighted for clarity):

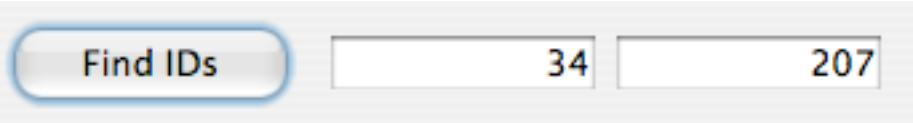
```
Possible duplicate record pair #50
Similarity score:  110
ID:                331                          339
First:             Audrey                       Jackson
Last:              Jackson                      Audrey
Street:            4086 Station St.             4086 Station St.
Zip:               48752                        48752
Phone:             201-460-4988                 201-460-4988
Email:             Audrey.Jackson@hspc.edu      Audrey.Jackson@hspc.edu
```

If you don't know that there are several first/last name swaps in the data, it's easy to build a set of rules that let these slip through. In this example, the address and phone number fields are less likely to indicate a person uniquely, so the weighting is heavily biased towards name comparisons. If you compare first and last names when the values are reversed, their similarity is typically zero. In this case, it's necessary to add rules that recognize first/last name swaps in the weighting. Although this is a common situation to keep in mind, I raise the point in a general way. It always helps to look at the data when building duplicate checking rules. Likewise, it's only by running sample rule sets and considering the results that it becomes possible to develop reasonable thresholds for what represents a highly probable duplicate.

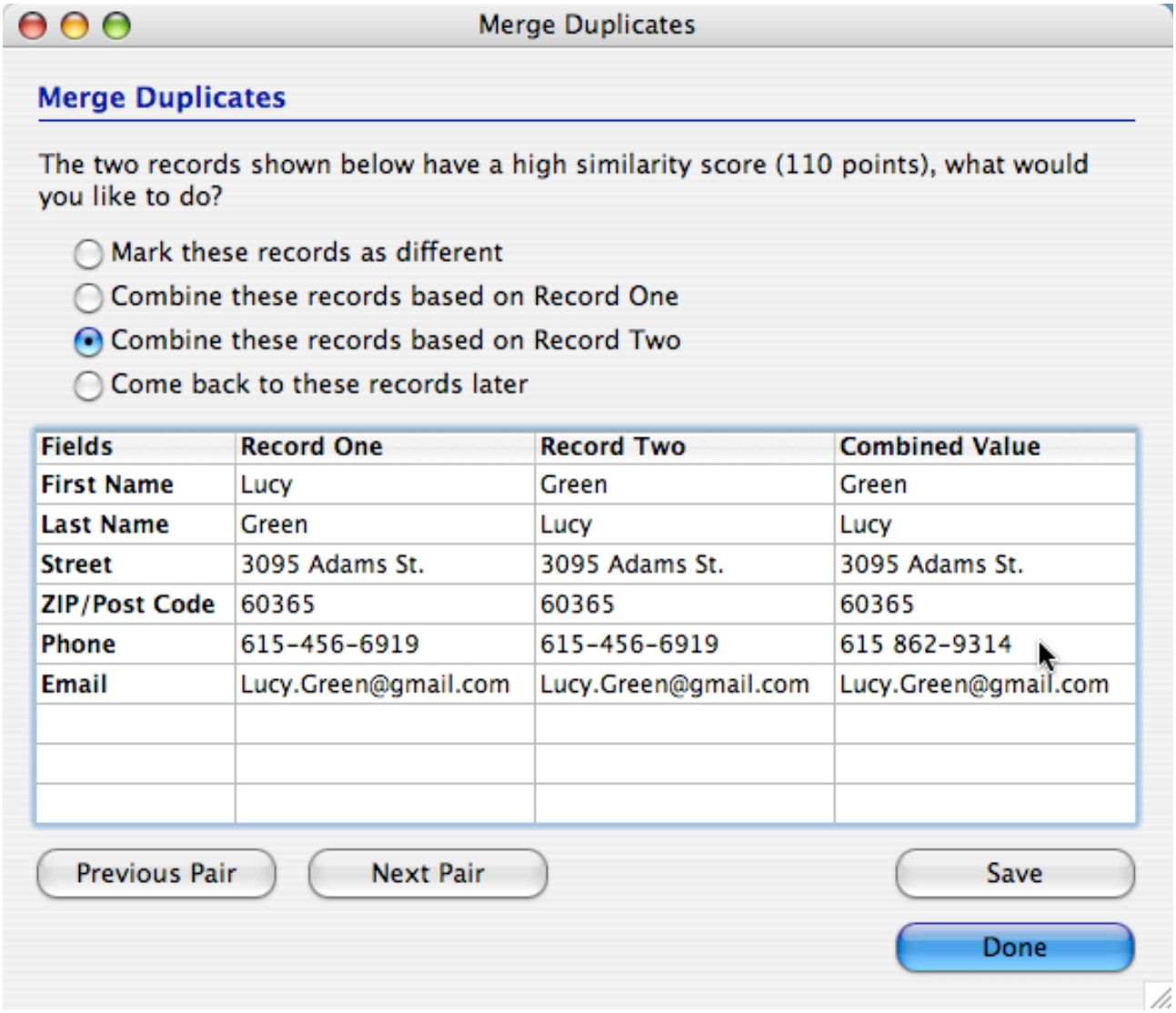
Combining Duplicate Records

Introduction

Fuzzy matching can largely automate the identification of duplicate records, but you will probably, at least at first, want users to review the matches before consolidating data. The sample report format from the demonstration, shown above, is meant to make it easy for a user to scan through the results and pick out true duplicates. The Find IDs button and entry areas at the bottom of the Person output list make it easy to type in two Person ID values and view the matching records.



This system of building a textual report and simplifying the loading of matching pairs is fine for a demonstration system. In a production system, this behavior can easily be enhanced. The mock-up below shows an idea for a screen that lets a user handle potential duplicate records found by the system:



In the screen shown above, the two records are shown side-by-side and the user is able to pick one as the basis of a merged record. In this example, the phone number is also being updated by hand. The screen also lets the user skip a record pair or mark it as a false positive. Both of these steps are important. If you force users to combine records, they may merge records that shouldn't be. Marking false positives is valuable as it lets you skip the same record pair on the next duplicate hunt and to estimate the percentage of false positives found by a particular rule set. To support all of these features, you only need a few pieces of data:

Rule set ID
First record ID
Second record ID
Similarity weight score or percentage
Is this pair a false positive? (Are they actually distinct?)

These values can be stored in a simple 4th Dimension table, updated by fuzzy duplicate hunts, and used to drive a record reconciliation interface.

Merge Data Thoroughly

Be sure to merge data carefully. When combining records that have related values, be sure to update all related values at the same time. Otherwise, you will end up with orphans. Record merging almost always needs to be performed within a transaction so that incomplete changes can be rolled back. A few points to watch out for:

- If you are consolidating records that have children, update all children.
- If you are consolidating records that are children of different parent records, determine which parent the link should be assigned to.
- If you are consolidating records that are children of another record, be sure that any related summaries are updated.

Additional Tools, Notes and Ideas

Probabilistic Record Matching

The record matching strategies outlined in this technical note depend on someone picking sensible match tests and assigning appropriate weights. This kind of ad hoc approach can be efficient and effective but there are ways to automatically refine the system. As an example, it may be obvious that Social Security Number (taxpayer ID) is a more significant field in an employee database than first name, but how do you detect significant fields automatically? And how do you quantify their relative significance? Is a match on Social Security Number twice as significant as a match on first name? Ten times more significant? Using probabilistic matching techniques can automatically answer these kinds of questions, at least provisionally. This subject is huge and well beyond the scope of this note, but a few more points can be made. Probabilistic methods often depend on building **frequency tables**. Code scans through a table, or a subset of a table, and inspects the field values. In the case of Social Security Numbers versus first names, such a scan could automatically detect that Social Security Numbers are more unique than first names. In this manner, code can automatically determine that a match on Social Security Number is more significant (more likely to indicate identical records) than a match on first name.

Frequency tables can also be used to automate weighting based on specific value pairs. Imagine a database that includes last names. If the database is in Omaha, Nebraska, the last name "Smith" is likely to occur far more often than the last name "Chu". Therefore, if you're checking for duplicates and find two instances of "Chu", this is more significant than finding a match on "Smith". In this particular database, a match on "Smith" is more likely to represent a false positive than a match on "Chu", given the relative frequencies of these names. With stored relative frequency tables and a bit of math (nothing more complex than calculating standard deviation), match pairs can be weighted based on actual values. This refined scoring system can be used in place of, or in addition to, the kind of weighting system discussed earlier and used in the demonstration report.

A last point is worth making about relative frequencies: they depend entirely on the data in the database. In the previous example, "Chu" is more significant because it appears less frequently in the Omaha database. The weighting would be different if the database were based in Beijing, where "Chu" would be far more common a name than "Smith". Notice, again, that frequency table based weighting systems can make these calculations automatically and without direct human intervention.

Note *If you are interested in developing a probabilistic record matching system, please feel free to write me to discuss designs, and for relevant research papers and formulas.*

Comparing Concatenated Fields Naively May Work Poorly

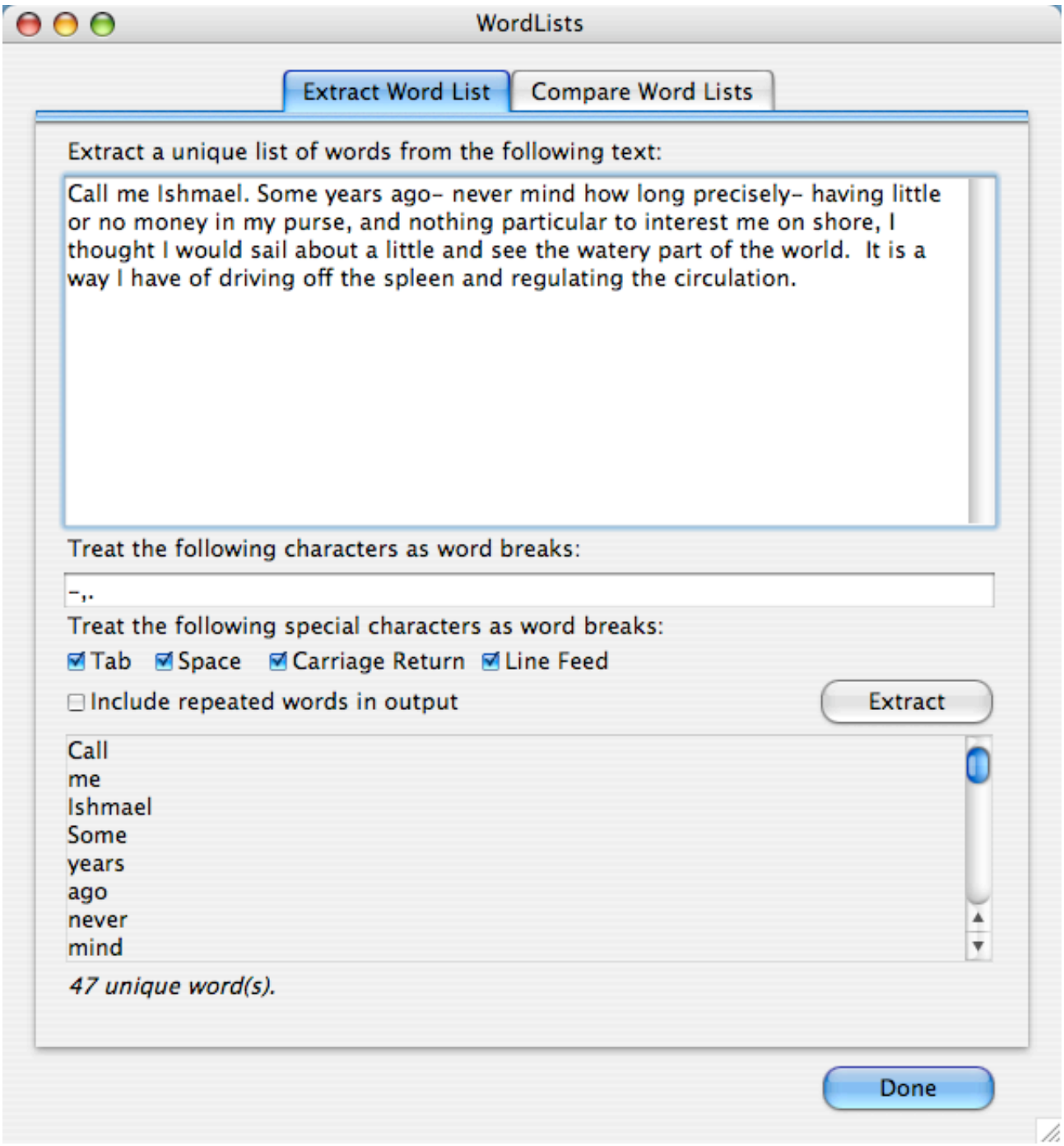
A simple strategy for comparing records is to compare the combined values of several fields. For example, combine first name, last name, and the string of date of birth in each of two records and then compare the two resulting text blocks. This approach is often used with "n-gram" analysis, a technique not implemented in FuzzyTools or the sample database. This approach is not as sensitive or reliable as many of the techniques discussed above, but it can be easy to implement. If you do take this approach, you may get reasonable results with *Fuzzy_GetEditDistance* or the WordList system, discussed next.

Bonus: WordList Comparisons

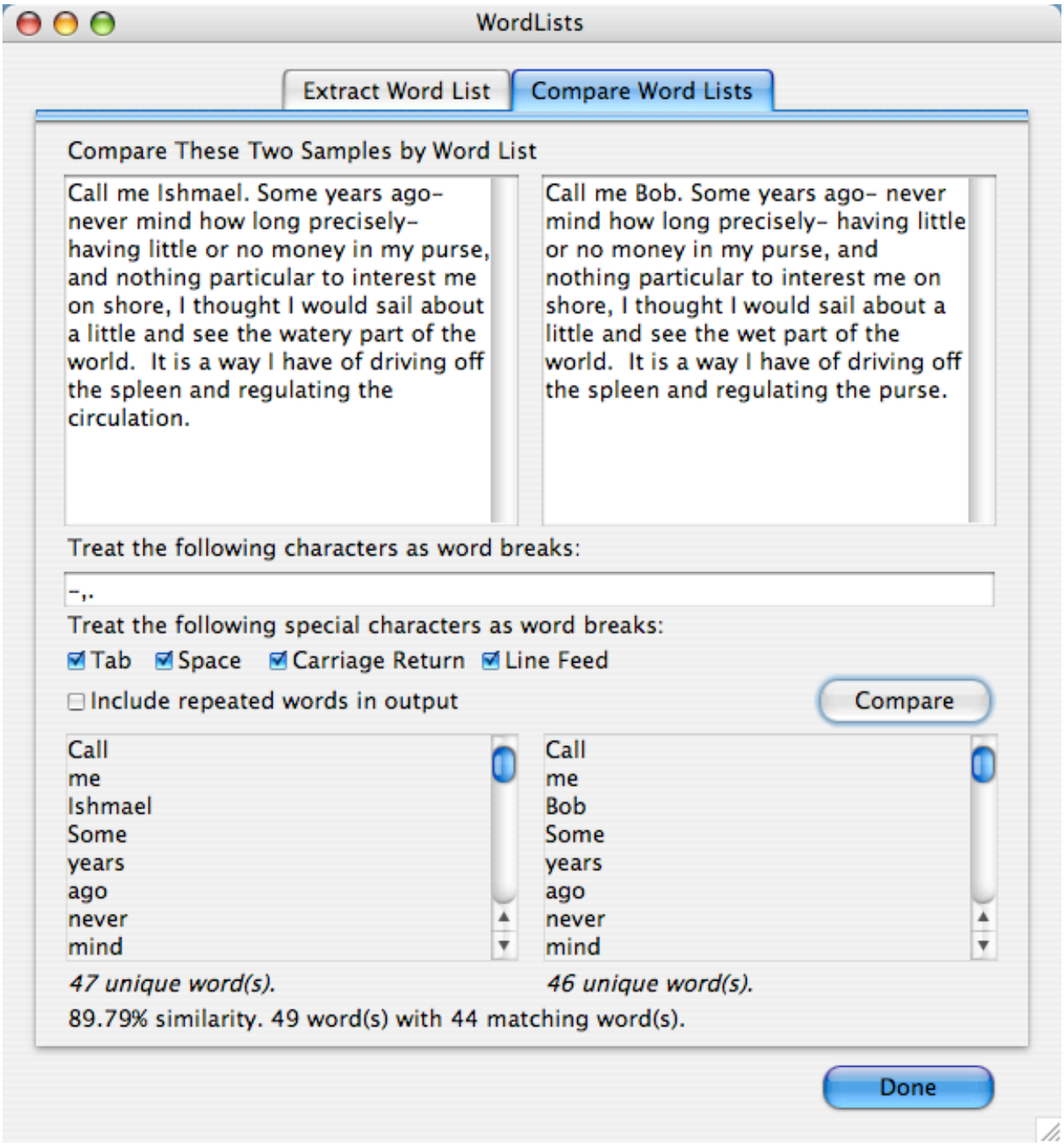
The sample duplicates report includes a "WordList similarity score" for each possible duplicate pair. To produce these scores, the system compares the number of unique words shared between the First_Name, Last_Name, Street, and ZIP_Code fields. This style of ranking is particularly helpful when data is in bad enough condition that fields are often reversed, such as, when first and last names are in the wrong fields. To produce a WordList score, the system follows these steps:

- Combines the values of the First_Name, Last_Name, Street, and ZIP_Code fields in the base record and treats them as a single block of text.
- Extracts the unique set of words found in the block of text.
- For each comparison record, the same steps are followed, giving us two arrays of words (WordLists) to compare.
- The WordLists are compared and the percentage of shared words returned.

If you are interested in experimenting or developing this form of comparison, review the WordList demonstration. It consists of a two page form. The first page allows you to extract the words from a block of text. Notice that the characters that are considered word breaks are configurable:


WordLists
Extract Word List
Compare Word Lists
Extract a unique list of words from the following text:
Call me Ishmael. Some years ago– never mind how long precisely– having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation.
Treat the following characters as word breaks:
–,.
Treat the following special characters as word breaks:
Tab
Space
Carriage Return
Line Feed
Include repeated words in output
Extract
Call
me
Ishmael
Some
years
ago
never
mind
47 unique word(s).
Done

The second page of the form lets you compare two blocks of text, as pictured below:



The **Include repeated words in output** checkbox illustrates a useful feature of the underlying code. Instead of comparing the unique words in each block of text, the system compares all of the words in the text blocks. For example, the text blocks above each have 58 words and, between them, 61 unique words. Because some words are repeated, such as "I", the percentage of shared word varies when comparing unique words and all words. Depending on the texts, it is probably more accurate to compare all words. In the example above, an all words comparison reports 90.16% similarity.

Calculating The Number of Record Pair Comparisons Required

Let's return to a topic I earlier promised to address in more detail: calculating the exact number of record comparisons required to exhaustively test a set of data for duplicates. As mentioned, the number of comparisons needed is roughly

(records in selection * records in selection) / 2

When doing an exhaustive comparison, no record needs to be compared with itself, and no pair of records needs to be compared redundantly. This rule is easy to understand if we present a set of five values in a conceptual comparison matrix:

	Andrew	**Jackie**	**Joan**	**Mary**	**Peter**
Andrew	Andrew = Andrew	Andrew = Jackie	Andrew = Joan	Andrew = Mary	Andrew = Peter
Jackie	Jackie = Andrew	Jackie = Jackie	Jackie = Joan	Jackie = Mary	Jackie = Peter
Joan	Joan = Andrew	Joan = Jackie	Joan = Joan	Joan = Mary	Joan = Peter
Mary	Mary = Andrew	Mary = Jackie	Mary = Joan	Mary = Mary	Mary = Peter
Peter	Peter = Andrew	Peter = Jackie	Peter = Joan	Peter = Mary	Peter = Peter

Of course, there is no need to compare any value to itself, so several comparisons, such as Andrew=Andrew, may safely be skipped. The new comparison grid is shown below:

	Andrew	**Jackie**	**Joan**	**Mary**	**Peter**
Andrew		Andrew = Jackie	Andrew = Joan	Andrew = Mary	Andrew = Peter
Jackie	Jackie = Andrew		Jackie = Joan	Jackie = Mary	Jackie = Peter
Joan	Joan = Andrew	Joan = Jackie		Joan = Mary	Joan = Peter
Mary	Mary = Andrew	Mary = Jackie	Mary = Joan		Mary = Peter
Peter	Peter = Andrew	Peter = Jackie	Peter = Joan	Peter = Mary	

It makes no difference in which direction we compare values to test if they are similar, and therefore, you don't need to perform redundant checks. If Joan # Andrew, then Andrew # Joan. This allows us to eliminate additional tests from the matrix, as pictured below:

	Andrew	**Jackie**	**Joan**	**Mary**	**Peter**
Andrew		Andrew = Jackie	Andrew = Joan	Andrew = Mary	Andrew = Peter
Jackie			Jackie = Joan	Jackie = Mary	Jackie = Peter
Joan				Joan = Mary	Joan = Peter
Mary					Mary = Peter
Peter					

There are several ways to calculate the exact number of comparisons needed, such as the one implemented in the *Demo_CalculateComparisonsNeeded* routine:

((records in selection * records in selection) / 2) - (records in selection / 2)

This value is important to keep in mind because it emphasizes why it is worthwhile to reduce the number of records that must be compared sequentially. If you can positively exclude some records from comparison, you can significantly reduce the total work required. For every record you eliminate from consideration, you save approximately

(1 * records to test)-1 comparisons.

For example, testing 400 records requires 79,800 comparisons while testing 399 records requires only 79,401 records, 399 less. Remember, too, that it can be worthwhile to artificially reduce the selection down to, say, 5-10% of the total table to accelerate testing multiple rule sets.

Generate Multiple Output Formats with XSLT

If you do choose to produce textual duplicate checking reports, you'll quickly find that there are numerous useful formats. It doesn't make sense to write a new routine for each format. Instead, you can create a single XML report and then one XSLT program for each desired output format. 4th Dimensions **APPLY XSLT TRANSFORMATION** can automatically build each output desired.

Summary

Data cleaning and deduplication improves data quality, improves report quality, and can reduce costs. Data cleaning can often be partially or fully automated, particularly for common data, such as addresses. Duplicate data matching can largely be automated, and made highly accurate, by combining a variety of fuzzy matching techniques with a simple scoring system.