Name Matching for Data Quality Mediator

by

Jin Mo Kim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author ....
Department of Electrical Engineering and Computer Science
May 30, 1995

Certified by ....
Richard Wang
Co-Director for Total Data Quality Management (TDQM) Reasearch Program and
Associate Professor of Information Technologies
Thesis Supervisor

Accepted by ....
Leonard A. Gould
Chairman, Departmental Committee on Graduate Students

Name Matching for Data Quality Mediator

by

Jin Mo Kim

Submitted to the Department of Electrical Engineering and Computer Science
on May 30, 1995, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Name matching refers to the process of identifying names that are equivalent but not necessarily identical. A pair of names are equivalent when they both refer to the same entity. For example, "M.I.T." and "Massachusetts Institute of Technology" are equivalent but not identical expressions. The topic of this thesis is a general theory to name matching which exploits syntactic, domain, and contextual knowledge to match names from two separate input tables. A computer program is implemented which performs name matching specifically on company names. Empirical analysis using the program shows that the algorithm can achieve 100% accuracy with a small number of user queries.

Thesis Supervisor: Richard Wang

Title: Co-Director for Total Data Quality Management (TDQM) Research Program and Associate Professor of Information Technologies

Acknowledgments

Professor Rich Wang has been my mentor and guide throughout the writing and implementation of this thesis. I am deeply indebted to him for his generosity in sharing his time and insight into matters beyond just those concerning my thesis but also concerning my future and direction as well.

I am also forever grateful to my parents, Paul and Anne Kim, who have provided the foundation of love and family that has carried me through my years at MIT. They have poured their lives into our family, and their sacrifice has been my eternal blessing.

I also wish to thank the one who has literally been at my side as I wrote this thesis. She has shared in the burden of its making as well as in the joy of its completion. I owe more than I can express to her.

Contents

List of Tables

Chapter 1

Introduction

Modern information systems have the capability to collect and process information from multiple sources. Any time data is collected from multiple sources, however, it becomes necessary to be able to recognize when references are being made to the same entity or when duplicate entries exist. This may be the case, for instance, when merging two lists of mailing addresses. In order to avoid mailing multiple copies of an item to the same household, duplicate addresses need to be purged. This is a problem when working with disparate databases because primary-foreign key relationships are not well defined.[1] Standards for notation and abbreviation may vary significantly across databases, and even misspellings confuse the process of identifying duplicates. For these reasons it is a non-trivial task to identify records that are referring to the same entity, and name matching is a topic of research which addresses this issue.

To make the issue a little more concrete, the following is an example of when name matching could be used. A join operation is desired between a Fortune 500 database and a Car Manufacturers database of companies that sell stock (see Table 1). The Fortune Database may have an entry with "Ford Co." while the Car Manufacturers database has "Ford Company." They both refer to the same company, but a simple DBMS join operation would be unable to recognize the match.

[1] A primary key is the attribute in a relation which uniquely identifies a record, and a foreign key is an attribute in a relation which is also a primary key in another relation.

	Fortune 500	Car Manufacturers	
⇒	Ford Co.	IBM	
	Apple Computers	Ford Motor Co.	⇐
	IBM	Microsoft	

Table 1.1: Name Matching Example

1.1 Thesis Topic and Goals

Name matching refers to the process of identifying names that are equivalent but not necessarily identical. A human operator would have little trouble recognizing that the example given above is a match. However, a human operator does not have the time to manually inspect thousands or even millions of possible matches, and a computer algorithm becomes useful.

Therefore, the goal of this thesis is twofold. The first goal is to *develop a theory* to name matching which can be applied to any class or domain of names, and the second goal is to actually implement a computer program which *applies the theory* to a specific domain. The domain chosen for this thesis is company names.

1.2 Thesis Organization

A theory to name matching utilizing syntactic, domain, and contextual knowledge is presented in the next chapter. This general theory is then applied to a specific domain, namely company names in chapter 3. Chapter 3 includes the details of software implementation and specific algorithms used in the program. Chapter 4 gives the results of empirical analysis performed on the name matching program using synthetic and real world data sets. The fifth and final chapter presents conclusions from this effort and points to several areas of possible future research and development.

Chapter 2

Overview and General Theory

Name matching is fundamentally a comparison test. When a person is asked to perform name matching on a pair of names, he or she will typically gather as much information as possible from the name itself, any knowledge which he/she may have concerning that name from previous experience, and also any attributes that may be associated with the name. These three areas outline the types of knowledge which are available and exploitable for the purpose of name matching: syntactic, domain, and contextual knowledge respectively.

2.1 Uniqueness Assumption

Before exploring name matching in more detail, however, it is necessary to define what is meant by two names *matching.* If the question seems at first too simple, the reason, most likely, is that people hardly make a distinction between a word and what it means or represents. When humans are presented with verbal information the words are immediately and transparently tied to their meanings. Computers, however, have no built-in capability to tie a word with what it represents unless a system of knowledge representation has been implemented. Name matching attempts to match the entities behind the words, but unfortunately only the representation of it will be available.

Simple comparison testing for computers, therefore, requires making an assumption which we will call the *uniqueness assumption.* The uniqueness assumption states that for any particular name, x, there is only one unique entity, y, which x refers to. In other words, a one-to-one correlation between representation and meaning is assumed. If a pair of names are identical, then they refer to the same object or entity. The corollary to this is that a single entity will not have more than one name identifying it.

The uniqueness assumption may appear like a reasonable assumption to make at first since names are often chosen to identify the entity which it refers to from a vast variety of other similar entities. If names did not have this characteristic than then they would be useless as identifiers and no longer serve one of the main purposes for which they were created.

Conceding their purpose for creation, however, some classes of names, nevertheless, clearly lack uniqueness. This happens especially as the number of similar entities increase without a like increase in the number of possible different names. People names are a prime example of this. As the population of English speaking people rapidly increases, the number of English names that are used remains relatively stable. This means that more and more people will end up with similar names. The problem does not arise because more permutations of letters do not exist. It is rather that when parents come to naming their child, they often draw from a pool of names which have been used before. In other words, people names are often "recycled." To varying degrees, this is common to all classes of names for which people are the primary namers.

Due to ambiguities introduced by naming constraints such as the one mentioned above, the uniqueness assumption does not hold strongly for many classes of names. More specifically, there are two cases in which relying *solely* on the uniqueness assumption will lead to incorrect conclusions. The first case is *representation overload* or when two different names refer to the same entity (e.g., Bob and Robert may refer to the same person). In this case the uniqueness assumption will fail to see a match and thus produce a false mismatch. The second case is *semantic overload* which is when two exactly matching names refer to different entities (e.g., Paris is a city in

France but also a city in Texas). The example given above concerning English names is a case of semantic overload. In this case the uniqueness assumption falsely assumes that the names refer to the same entity producing a false match.

The success of comparison testing for computers will, therefore, lie in being able to identify those situations in which the uniqueness assumption fails and being able to correct for it appropriately. This will be accomplished by employing the three types of knowledge which are described in the following sections.

2.2 Three Types of Knowledge

There are three types of knowledge which can be leveraged in order to determine if a pair of names are equivalent. The first type of knowledge is *syntactic knowledge* found in the name itself. Syntactic knowledge attempts to know nothing about the meaning of a particular name but only looks at how the name is constructed, and a high degree of similarity between two names is a strong indicator that they are equivalent. The second type of knowledge is *domain knowledge.* Domain knowledge captures all that can be known about a class of names from previous experience. This includes conventions of abbreviation and notation as well as knowledge of nicknames or aliases which have previously been used. Domain knowledge will be important in determining which form of a particular name is to be used as its canonic form. The third type of knowledge is *contextual knowledge* that may or may not be available for a given name. Contextual knowledge here refers specifically to secondary information found in the adjacent columns of the table in which the name resides. For example, in a directory database the name of a person is usually supplemented by contextual information such as street address, city, state, ZIP, and phone number. Contextual knowledge has the potential to provide the highest degree of discriminating power.

There is actually a fourth type of knowledge which a computer will have available to it, and this is the human operator. When all else fails, the computer will default to the human operator to make the final call as to whether or not a pair of names are equivalent.

2.3 Syntactic Knowledge

Syntactic knowledge is simply knowledge about the construction of a name. All names are constructed of letters, spaces, and punctuation, and it is the structure of how these elements are combined that compose syntactic information. This information is used to test pairs of names for similarity using string matching. Because syntactic knowledge, by definition, separates itself from the meaning of words, it is unable to identify or correct for semantic overload. It is, however, able to check for several cases of representation overload.

It is possible to have representation overload due to misspelling. A misspelling is likely to alter the original syntax of a word by a small degree. The computer's task will be to decide if in fact the difference is such that a misspelling can be inferred. To infer a misspelling further infers that the user who entered the name was actually referring to the entity that the correctly spelled name refers to. Thus a match can be assumed.

Due to its lack of semantics, syntactic knowledge has relatively low discriminating power compared with the other types of knowledge that will be discussed.

It is sometimes the case that a name is purposely altered to a similar form, but the alteration is not illegal, meaning it follows some accepted convention. For example, initials are often used in people names. "Susan May Jones" can also be written "Susan M. Jones" or "S. M. Jones." This type of knowledge falls into domain knowledge which will be discussed in the following section, but it is mentioned here because the same string matching algorithm can be used to apply this type of knowledge. In other words, implementation will not fall cleanly into the lines of theory that are being built but will combine and recycle ideas in order to optimize work.

2.4 Domain Knowledge

Domain is a very flexible term. The key to defining it correctly is striking a balance between being restrictive enough to provide some discriminating power and liberal enough to be able to use in real world applications. For the purposes of name matching, *domain* is defined to be a class of names which has a naming convention. Conventions are rules of practice which are widely accepted and followed. Several examples of such classes are company names, people names, addresses, and university names.

If domain is defined to be a class of names with a naming convention then domain knowledge refers to all that can be known about those conventions. In essence, the name matching program is a knowledge-based system which attempts to capture the knowledge that an expert would have about matching company names, and it is domain knowledge which captures the bulk of the "knowledge" that is represented in the system.

Domain knowledge is represented in two forms. The first representation is captured in *rules* which use a particular convention to condition all names in an input file to a canonic form. Such rules would, for example, take out all punctuation and drop common endings such as "Inc." and "Co." The second representation is a *system table.* The system table is different from rules in that the table will contain knowledge of specific names where as rules are generic and apply to all names. The system table will contain three types of information. First it will have a name which is defined to be the canonic form of a particular company name. Second, the table will contain any number of aliases which are associated with the company name. The third type of information will be unique specifiers which are discussed in the next section.

The use of an system table enables the program to gain knowledge as the user may decide to add entries to the system table which is not be captured by the generic conditioning rules. The program may also allow the user to add a generic rule which gives the user the ability to tailor the program to the application at hand. This will enable the name matching program to significantly improve its performance with time.

The use of system tables and rules are both attempts to remedy representation overload because they both start with the assumption that several representations are possible and try to reduce the possibilities down to a canonic form for a particular name.

Note that it is possible for a single class of names to have multiple conventions. For example, people names are usually given in order of first name followed by surname. However, in asian cultures the convention is to give the surname followed by the first name. A name matching algorithm should identify and use only those conventions which are unique to a class of names.

2.5 Contextual Knowledge

Context is another term with a wide range of possible meanings, but again, for the purposes of name matching the definition of context is strictly defined to be secondary information that may or may not be available as adjacent columns to the column being matched. In database terminology, this refers to information in accompanying fields of the same record as the name being matched. Contextual knowledge would not be available if name matching is being performed on a list of names rather than a column from a table.

2.5.1 Keys: Unique Specifiers

Contextual knowledge has the potential to provide the highest degree of discriminating power among the three types of knowledge discussed. This is because the problem of trying to decide if two names are equivalent is not a new problem that has arrived with the use of disparate databases. Even before networks made access to disparate databases possible, people have recognized the need for additional specifiers that are unique to the entities that names refer to. These specifiers are names for the which the uniqueness assumption holds perfectly. For example, the social security number is a unique key for people that is widely used. Companies that sell stock have a ticker tape symbol or a disclosure number that are designed to be unique keys.

A unique key column such as the ones described above has perfect discriminating power, meaning that if that key is available for all names being matched, then every pair of names can be unambiguously resolved as a match or mismatch. A unique key solves both representation overload and semantic overload.

2.5.2 Non-Unique Specifiers

Unique keys will not always be available, but even without them contextual knowledge plays a very important role in name matching. A previous example of contextual knowledge mentioned fields such as address, telephone, and zip. These are not unique keys in the two ways. First, a match in one of these fields may not necessarily imply a match in the names (similar to semantic overload). For example, two companies may have the same zip code. Second, a non-match in one of these fields may not necessarily mean that two separate entities are being referred to (representation overload). A single company may have several telephone numbers which are used. Although non-unique fields lack the definitive authority of unique keys, taken together or in subsets, these specifiers can still provide a high degree of discriminating power.

For example, in order to deal with semantic overload, a comparison test may combine syntactic knowledge with contextual knowledge. A comparison of two similar names may show that their addresses are in two different states. This may be used as evidence to support the hypothesis that the two names are actually referring to two different companies. The same kind of example applies to representation overload. Two names may be dissimilar, but if all other contextual information indicate that the two names are actually referring to the same company, this may be used as evidence to declare a match.

Chapter 3

Implementation and Algorithms

3.1 Introduction

The previous chapter outlined the three types of knowledge which a name matching system should exploit in order to achieve maximum discriminating power. This chapter describes the Name Matcher in the Data Quality Mediator which has been implemented specifically to show that the three knowledge areas are adequate to build a computer algorithm for name matching which can achieve accuracy rates of 95 - 100 percent.

3.2 Data Quality Manager

Name matching will be implemented as part of a larger design for data management called the Data Quality Manager (DQM). DQM will have the capability to query data from multiple databases, measure and analyze the quality of that data, and finally modify the data to improve its quality. Name matching will be one option available to the user for improving data quality.

3.3 Basic Implementation

DQM and correspondingly the DQM Name Matcher is implemented using Microsoft Access and its macro language, Access Basic. Access has several characteristics which were useful for implementing the DQM. Access allows users to build graphical user interface (GUI) applications on top of existing databases, and it also allows users to attach tables from remote databases through ODBC.

An important aspect of Access Basic is that it is an event driven programming language. Rather than typing a start command and waiting for the program to run its course, the user is in control of how the program runs by performing actions with either the mouse or the keyboard on the active window.

3.4 Salient Features

When the user selects "Name Matching" from the DQM Main Menu, the Name Matching window is opened on the screen. (See figure 3-1) The following sections describe in detail each section of the window and their corresponding functions.

3.4.1 Tables and Field Lists

The boxes that appear directly below the Name Matching logo are used to specify tables and columns to be used in the name matching process. These boxes appear in pairs so that the boxes on the left hand side are concerned with the first input table (Table A) and the right side boxes concern the second input table (Table B).

3.4.2 Buttons

There are three rows of buttons normally visible to the user; two rows right below the Tables and Field Lists, the third midway from the top and bottom on the right hand side of the window. The first row of buttons directly below the Tables and Field Lists are used to walk through a name matching session. The second row of buttons directly below these are used to modify the system tables which control the behavior

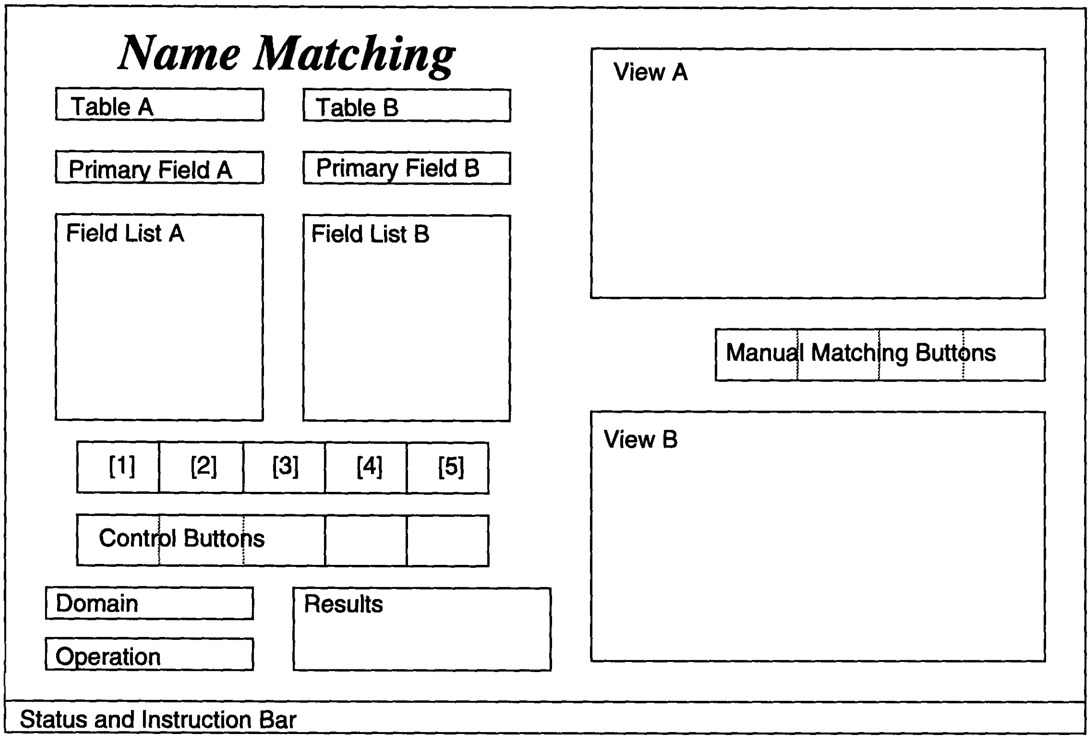

Figure 3-1: Layout of the DQM Name Matching Window

of the name matching process as well as allow the user to quit, start over, or cancel. The third row on the right hand side are used when the user chooses to manually declare matches from the table views and also to control the views themselves.

3.4.3 Views

The views occupy the right half section of the name matching window. There are two modes of view; a double window and a single window view. The default is the double window view which displays two tables at once. The single-window view occupies the entire right half of the window and is used only when displaying the system table for modifications or displaying the output table at the conclusion of a name matching session.

3.4.4 Option Boxes

There are two option boxes,and they are located on the lower left corner of the window. The first option box on top allows the user to specify the domain in which the name matching is to be executed. For this thesis, only the "Company Names" option has been implemented. The second option box allows the user to specify the type of join operation that is desired. There are four possibilities:

- Inner Join
- Right Join
- Left Join
- Merge

3.4.5 Results Window

The sunken box located directly to the right of the option boxes displays a running total of how records have been matched, how many records are left unmatched in the two input tables, and how many records will be included in the final output table depending the type of join the user has specified.

3.5 Running a Session

This section provides a general description of how a session is executed in the name matching window. Appendix B contains a more detailed description using a sample tables and a scenario.

3.5.1 Specifiying Tables and Fields

The first step in running a name matching session is to specify which tables contain the name fields which are to be matched. Clicking on the pulldown button located flush right on the boxes labeled Table A and Table B displays a list of all available tables in the current database. As soon as a table is chosen, the name matcher looks up the table definition and displays a list of field names in the field list box. A constraint on table definitions using Access requires that field names for any specific table be unique. It may, however, be the case that Tables A and B use the same name for any one or more of its fields. For example, both tables may use the name "Company_Name." To allow both fields to be included in the join table, the Name Matcher appends a '"A_" and "B_" to the beginnings of all field names for their respective tables.

The next step after specifying the tables is to specify the fields which contain the names that are to be matched. These are called the *primary columns.* This can be done by simply clicking on the appropriate field name in the field list box. Now the user is ready to begin matching.

3.5.2 Button [1]: Exact Matching

The first matching that is executed is exact matching. None of the other four out of five matching buttons are enabled until exact matching has first been performed. Exact matching simply performs a run-of-the-mill DBMS join operation on the primary columns specified.

There are several important initialization procedures that occur at this time. First of all, the input tables are copied into temporary work tables NM_UnMatched_A and NM_UnMatched_B, respectively. Following this, the results of the join operation are used to make a new output table called NM_Matched. Finally, the records that were joined are purged from the UnMatched tables. After exact matching has been executed, the unmatched tables will appear in the views to the right of the screen.

3.5.3 Button [2]: Context Matching

Context matching can be used when columns containing the same unique-specifier exists for both tables. For example, if both Tables A and B contain fields for the ticker tape symbol of the company, the user can select those fields from the field lists and press the Context Matching button. If no such fields exist, then pressing the

Context Matching button without having selected fields will move the session to the next step which is Canonic matching.

If the user has selected fields, then the name matcher first performs a check to see that the fields are indeed unique. A join operation is executed, and any record that is joined more than once indicates that the field is not unique. The user is informed that the fields are not unique and matching is not performed. If no record is joined more than once, then the results of the join are appended to the match table and purged from their respective unmatched tables.

Notice that the name matcher cannot check to see that the selected columns are *by design* unique-specifiers like ticker tape symbols or Social Security numbers. The only thing that the can be checked is to see that no value occurs twice in the same table. This means that a field like "city" can serve as a unique-specifier if no other records in that table and no more than one record on the other table contains the same value.

3.5.4 Button [3]: Canonic Matching

The purpose of canonic matching is four fold:

1. take out all punctuation
2. convert common words such as "and" to a canonic form "&"
3. truncate common endings such as Inc, Co, etc...
4. check if the name is an alias

The name matcher applies various string operations to carry out the above tasks, and the resulting string is stored in a column appended to the original unmatched table. The columns are named A_NM_CANONIC and B_NM_CANONIC, respectively. Following the conditioning process, the resulting name is checked against the alias table to see if there is a match, and if there is, the name is replaced by its canonic form as defined in the alias table. Finally, a join operation is executed on the appended columns. As in the previous cases, the results of the join are appended to the matched table and purged from the unmatched tables. Upon successful execution of canonic matching, the name matcher allows the user to move on to either keyword matching or Soundex matching.

3.5.5 Button [4]: Keyword Matching

Keyword matching performs further conditioning on the canonic form of the names and stores the results in a second appended field called A_NM_KEYWORD and B_NM_KEYWORD. What keyword does specifically is return the first word that is not a single letter and not a common word as defined by the user. "Common" words are stored in a system table called the Condition Table. The user has access to this table and can add, delete, or modify entries in the table. Some common words include First, National, General, and articles. If the first word happens to be a single letter then the program returns the next word as well. In exactly the same manner as canonic matching, a join operation is executed on the newly filled fields and appropriate append and purge operations are executed.

3.5.6 Button [5]: Soundex Matching

Soundex matching uses an algorithm called the Soundex algorithm to convert the canonic form of the name into a string of characters and numbers. The soundex algorithm is described in detail along with the code in Appendix C. As with canonic and keyword matches, the resulting string of characters and numbers are inserted into an appended column. Then a join operation is used to find matches that are to be added to the match table and deleted from the unmatched tables.

3.6 Improving Matching Performance

The user has several options for improving the performance of the name matcher after he/she gains an understanding of the domain in which the name matching is being performed and the procedure which the DQM Name Matcher follows in matching names. All options for improving performance relies on the user to modify one or more system tables which specify exactly what the DQM Name Matcher looks for during different phases of the process.

3.6.1 Alias Table

The system table contains a list of aliases for specific company names which would not otherwise be detected by the name matcher. For example, acronyms cannot be detected by the name matcher unless specified in the alias table. The alias table provides a mechanism for handling the exceptions that always occur within conventions of nearly any sort. A single company can have up to four aliases.

In order to modify the alias table, the user presses the button labeled "Alias Table," and the table will appear in single-table mode in the view section of the window. With the mouse, the user can click on either an existing record to modify or add aliases to an already existing company, or the user may click on a blank record to add a new company and an alias. When finished, pressing the now depressed button will restore the button and return the session to where it left off .

3.6.2 Condition Tables

There are two condition tables which specify how canonic and keyword matches are carried out. The first is the discard table and the second is the replace table.

The discard table consist of two columns which contain the first and end tokens discarded, if found, during the keyword and canonic matching stages respectively. During canonic matching, the end token is found by finding the last space which occurs before the end of the name and discarding everything before the space and including the space. If no space is found then an empty string is returned. If the end token is in the column of end tokens specified in the condition table, then it is discarded from the name. The process is similar for keyword matching except that the first token is used instead of the end token.

The replace table also consists of two columns and both columns are used to convert a name to its canonic form. The first column specifies a search string, and the second column specifies a replace string. Anytime the search string is found in a name, it is replaced by the replace string. This table is used mostly to replace punctuation with spaces, but it is also used to replace common words such as "and" to a canonic form such as "&." In the latter case, the choice of which form to make canonic is arbitrary and unimportant as along as all occurrences of the word are converted to a common form.

The condition tables can be modified in similar fashion to the alias table. The button labeled "Condition Tables" will cause the discard table to appear in the top view and the replace table in the bottom view.

Chapter 4

Empirical Analysis

Empirical analysis was performed on the DQM Name Matcher using two different types of data: synthetic data and real world data. Synthetic data was built specifically to test that the designed features of the name matcher were working properly. Section 1 describes how the synthetic data was prepared and also gives the results of performing a DQM Name Matching session on the set. Section 2 describes the real world data that was used to test the DQM Name Matcher.

4.1 Synthetic Data Set

4.1.1 Making the Set

The synthetic tables, consisting of Synthetic Table A and B, were constructed by first querying the Fortune 1000 '93 database to return all companies that had a primary SIC code of 3571. The SIC code categorizes company's according to the type of service or product that they offer, and 3571 refers to companies that manufacture electronic computers. There was no particular reason for choosing this subset of Fortune companies except that the names were familiar to the author and the size query was adequate for the purpose of testing. This query returned 16 records which makes up Synthetic Table A.

To prepare Synthetic Table B, records in Table A was modified to test a certain

Synthetic Table A	Synthetic Table B	
A_Company_Name	B_Company_Name	Match Type
Dell Computer Unisys	Dell Computer Unisys	Exact
Hewlett Packard Digital	Hewl. Pack. Dig. Equip. Corp.	Context
Apple Computer Ast Research	Apple Computer Inc Ast Research Incorporated	Canonic-End Token
International Business Machines Silicon Graphics	IBM SGI	Canonic-Alias
Gateway 2000 Sun Microsystems	Gateway Sun Micro Inc.	Keyword
Compaq Computer Tandem Computers	Compac Computer Tandam Computers	Soundex
	Sun Diamond Growers	Mismatch

Table 4.1: Synthetic Tables

feature of the name matcher. For example, end tokens such as Inc. and Co. were added to some names to test Canonic matching. Canonic matching was also tested by converting some names to their aliases which were input into the system alias table. Parts of names were deleted or added to test Keyword matching, and some names were purposely misspelled to test the Soundex algorithm. Some names were left unmodified to test exact matching. Finally, a miscellaneous record was added to Table B which does not have a match in Table A. The modified table was then saved as a second table called Synthetic Table B. Table 4.1 shows the names in Synthetic Table A and B along with the type of match which they represent indicated in the third column. The ticker tape symbol was included in both tables as a primary key on which to perform context matching.

4.1.2 Testing the Set

After synthesizing Tables A and B, saving the tables into the DQM database made them available for the DQM Name Matcher to examine. Notice that remote tables can also be attached to the DQM database meaning that tables need not physically reside in the machine which is running the DQM. Following the procedure for running a name matching session described in the previous chapter, Synthetic Tables A and

A_Company_Name	B_Company_Name
Dell Computer	Dell Computer
Unisys	Unisys

Table 4.2: Exact Matches

A_Company_Name	B_Company_Name
Dell Computer	Dell Computer
Unisys	Unisys
Hewlett Packard	Hewl. Pack.
Digital	Dig. Equip. Corp.
Apple Computer	Apple Computer Inc
Ast Research	Ast Research Incorporated
International Business Machines	IBM
Silicon Graphics	SGI
Gateway 2000	Gateway
Sun Microsystems	Sun Micro Inc.
Compaq Computer	Compac Computer
Tandem Computers	Tandam Computers

Table 4.3: Context Matches

B were selected, and buttons [1] through [5] were used to perform the matching. Table 4.2 shows the results of performing exact matching on the input tables. Table 4.3 shows the results of performing context match. Table 4.4 shows the results of performing canonic match. Table 4.5 shows the results of performing keyword match, and Table 4.6 shows the results of performing soundex match. Notice that when a primary key is available, no other matching algorithm is necessary.

A_Company_Name	B_Company_Name	Canonic Form
Apple Computer	Apple Computer Inc	Apple Computer
Ast Research	Ast Research Incorporated	Ast Research
International Business Machines	IBM	International Business Machines
Silicon Graphics	SGI	Silicon Graphics

Table 4.4: Canonic Matches

A_Company_Name	B_Company_Name	Keyword
Gateway 2000	Gateway	Gateway
Sun Microsystems	Sun Micro Inc.	Sun

Table 4.5: Keyword Matches

A_Company_Name	B_Company_Name	Soundex Code
Compaq Computer	Compac Computer	
Tandem Computers	Tandam Computers	

Table 4.6: Soundex Matches

4.2 Real World Data Set

The real world data sets were obtained from two separate sources: the Worldscope database and Fortune 1000 '93 database. Just as with the synthetic data sets, the primary SIC code was used to query a subset of the company tables from their respective databases. For the Worldscope database, an additional specificier was added to restrict the query to U.S. companies only. Included in both queries was the ticker tape symbol which provided a way of determining how many true matches were in the two input tables created by the queries. After determining the number of true matches, name matching was performed on the input tables without using the ticker tape symbol so that performance of the rest of the name matching algorithm could be measured. The data sets tested are listed along with their output tables in Appendix B.

Tables 4.7 and 4.2 gives the vital statistics of the name matching performed on the input tables enumerated in Appendix B. The first column of Table 4.7 gives the SIC Code which was queried. The second column gives the number of true matches existing in the two input tables. This was obtained by performing a join operation using the ticker tape symbols as specifiers. The third column gives the number of actual matches that were obtained without using the ticker tape symbol with the DQM Name Matcher. The fourth column is the percentage of names which were matched (Actual/True * 100). The fifth column indicates how many of the matches that were made required the user's input. This is essentially the number of matches made using keyword and soundex since both of these require the user to make the final decision. Finally, the last column gives the percentage of matches made requiring user input from all matches made. Table 4.2 breaks down the matches by type.

SIC Code	True Matches	Actual Matches	% Matched	Queried Matches	% of Matches from Queries
6331	16	16	100%	5	31%
2911	13	13	100%	1	7.7%
3571	5	5	100%	1	20%
4813	9	9	100%	1	11%

Table 4.7: Matching Statistics

SIC Code	Exact Matches	Canonic Matches	Keyword Matched	Soundex Matches
6331	0	11	5	0
2911	0	12	1	0
3571	0	4	1	0
4813	1	7	1	0

Table 4.8: Matching Statistics II

Chapter 5

Conclusion

The goals initially set for this thesis were succesfully reached. The main contribution of this thesis is a framework for viewing all the available information that can be exploited toward determining if two names within a specified domain are equivalent. This framework includes the syntactic, domain, and contextual knowledge that is either embedded in the name itself, in a known convention, or in accompanying fields of the record, respectively. The DQM Name Matcher has been implemented to show that these concepts can be coded into a computer algorithm for actual use in real working environments.

The emprical analysis shows that the algorithm implemented in DQM Name Matcher can achieve 100% accurary with real world data. There are, however, two important details that point to possible future research and development of the DQM Name Matcher. The first detail is that although the test set included company names from a variety of domains, the names came only from two databases, Worldscope and Fortune 1000. Both these databases were likely to have naming conventions which would have constrained the types of mis-matches that the name matcher encountered. Therefore, further testing with a wider variety and greater number of sources is needed before giving confidence to the DQM Name Matching algorithm.

The second detail concerns the percentage of matches that required user input. The Name Matcher required the user's final decision for an average 17.4% of its matches. This was not a problem for tables in the test set which had no more than

16 matches. For tables with number of matches in the thousands or more, however, 17.4% proves to be quite a burden for the user to have to manually inspect. An improvement in the keyword and soundex matching algorithms could greatly reduce this burden. Currently, the system queries the user whenever a keyword or soundex match is found. This is because those matches are not strong enough in themselves to conclude a definitive match. If, however, these matches were combined with some other measure of equivalency, the two measures together could be strong enough to declare a match without asking the user. One such measure of equivalency could come from the non-unique context fields that are currently unused by the system.

Appendix A

Code

```
Option Compare Database   'Use database order for string comparisons

'*** CONSTANTS **************************************************************

Const TABLE_COLOR = 32768, P_COLOR = 16711808, S_COLOR = 4194432
Const COMPANY_NAMES = 1, ADDRESSES = 2, PEOPLE_NAMES = 3
Const INNER_JOIN = 1, OUTER_JOIN = 2, LEFT_JOIN = 3, RIGHT_JOIN = 4
Const PASS = 1, FAIL = 0
Const EXACT_MATCH_STAGE = 0, UNIQUE_CONTEXT_STAGE = 1, CANONIC_STAGE = 2
Const KEYWORD_STAGE = 3, USER_STAGE = 4, FINISHED_STAGE = 5, QUERY_STAGE = 6
Const MB_OK = 0, MB_OKCANCEL = 1    ' Define buttons.
Const MB_YESNOCANCEL = 3, MB_YESNO = 4
Const MB_ICONSTOP = 16, MB_ICONQUESTION = 32    ' Define icons.
Const MB_ICONEXCLAMATION = 48, MB_ICONINFORMATION = 64
Const MB_DEFBUTTON2 = 256, IDYES = 6, IDNO = 7    ' Define other.
Const IDCANCEL = 2, IDOK = 1

'*** VARIABLES *************************************************************

'Recordset Variables
Dim MyWS As WorkSpace
Dim MyDb As Database
Dim matched_set As Recordset, un_one As Recordset, un_two As Recordset
Dim t1 As String  'name of Table A
Dim t2 As String  'name of Table B
Dim p1 As String  'name of primary field in table A
Dim p2 As String  'name of primary field in table B
Dim s1 As String  'name of secondary field in table A
Dim s2 As String  'name of secondary field in table B
Dim man_match As Integer     'number of records matched manually
Dim man_nonmatch As Integer
Dim FIELD_LIST_A As String, FIELD_LIST_B As String
Dim exact_matches As Integer
Dim context_matches As Integer
Dim canonic_matches As Integer
Dim keyword_matches As Integer
Dim soundex_matches As Integer
Dim total_matches As Integer


'*** FLAGS **********************************************************

Dim KEYWORD_MATCHED As Integer
Dim CANONICIZED As Integer
Dim SOUNDEXED As Integer
Dim EXACT_MATCHED As Integer
Dim MATCH_VIEW_CREATED As Integer
Dim A_VIEW_CREATED As Integer
Dim B_VIEW_CREATED As Integer

Sub btn_Cancel_Click ()
   On Error GoTo Err_btn_Cancel_Click
   DoCmd Close

Exit_btn_Cancel_Click:
   Exit Sub

Err_btn_Cancel_Click:
   MsgBox Error$
   Resume Exit_btn_Cancel_Click

End Sub

Sub btn_Canonic_Click ()
On Error GoTo Err_btn_Canonic_Click

   If Not CANONICIZED Then

         Write_STAT "CANONIC STAGE: Please Wait.  Primary fields are being
                transformed to canonic form."
         DoCmd Hourglass True

         Condition_Set "NM UnMatched One", "A_CANONIC"
         Condition_Set "NM UnMatched Two", "B_CANONIC"
         canonic_matches = Update_Tables("A_NM_CANONIC", "B_NM_CANONIC", False)
         Update_Results

         If Not STAGE = FINISHED_STAGE Then
            [btn_Keyword].Enabled = True
            [btn_Soundex].Enabled = True
            [btn_Keyword].SetFocus
            [btn_Canonic].Enabled = False
            Write_STAT "Next, KEYWORD STAGE: Canonic forms will be
                parsed for keyword.  Press [4]."
         End If

         CANONICIZED = True
```

```
  End If

Exit_btn_Canonic_Click:
  DoCmd Hourglass False
  Exit Sub

Err_btn_Canonic_Click:
  CANONICIZED = False
  Select Case Err
      Case 13 'Type Mismatch
        Write_STAT "The two selected fields have incompatible types."
      Case Else
        MsgBox Error$
  End Select
  Resume Exit_btn_Canonic_Click

End Sub

Sub btn_Context_Click ()
On Error GoTo Err_btn_Context_Click

      s1 = Me!field_one_list
      s2 = Me!field_two_list

      If (s1 = "") Or (s2 = "") Then
        If Not CANONICIZED Then
            [btn_Canonic].Enabled = True
            [btn_Canonic].SetFocus
        End If
        GoTo Exit_btn_Context_Click
      End If

      'Check to see if keys are unique
      Dim TempRS As Recordset
      Set TempRS = MyDb.OpenRecordset("SELECT [NM UnMatched One]." &
            s1 & " FROM [NM UnMatched One] INNER JOIN [NM UnMatched
            Two] ON [NM UnMatched One]." & s1 & " = [NM UnMatched
            Two]." & s2 & " GROUP BY [NM UnMatched One]." & s1 & "
            HAVING (Count([NM UnMatched One]." & s1 & ") > 1);")
      If Not (TempRS.EOF) Then
        MsgBox "The chosen fields are not unique keys."
        GoTo Exit_btn_Context_Click
      End If
      TempRS.Close

      context_matches = Update_Tables(s1, s2, False) + context_matches

      Update_Results

Exit_btn_Context_Click:
  'context_matches = (Val([txt_result_1].Caption) - exact_matches)
  [field_one_list] = ""
  [field_two_list] = ""
  Exit Sub
```

```
Err_btn_Context_Click:
  context_matches = 0
  Select Case Err
      Case 13 'Type Mismatch
        Write_STAT "The two selected fields have incompatible types."
      Case Else
        MsgBox Error$
  End Select
  Resume Exit_btn_Context_Click


End Sub

Sub btn_Context_KeyPress (KeyAscii As Integer)
  btn_Context_Click
End Sub

Sub btn_done_Click ()
      Me![btn_view].Enabled = True
      Me![table].Form.allowediting = False
      Me![table].Form.defaultediting = 3   'Read Only
      Me![table2].Form.allowediting = False
      Me![table2].Form.defaultediting = 3   'Read Only
      [btn_match].Enabled = False
      [btn_manual].Enabled = True
      [btn_manual].SetFocus
      [btn_done].Enabled = False
End Sub

Sub btn_Exact_Click ()
On Error GoTo Err_btn_Exact_Click

  If Not EXACT_MATCHED Then

      If ([field_one_list] = "") Or ([field_two_list] = "") Then
        MsgBox "Please choose two fields to match", 48
        GoTo Exit_btn_Exact_Click
      ElseIf Me!table_one_list = Me!table_two_list Then
        MsgBox "Choose Different Tables", 48
        GoTo Exit_btn_Exact_Click
      End If

      t1 = Me!table_one_list
      t2 = Me!table_two_list
      p1 = Me!field_one_list
      p2 = Me!field_two_list

      'DoCmd Hourglass True
      If Initialize_NM() = FAIL Then
        MsgBox "Failed to initialize tables", 16
        GoTo Exit_btn_Exact_Click
      End If
```

```
        [table_one_list].Locked = True
        [table_two_list].Locked = True
        [txt_primary_1].Caption = p1
        [txt_primary_2].Caption = p2
        [btn_view].Enabled = True
        [btn_manual].Enabled = True

        [field_one_list] = ""
        [field_two_list] = ""

        Update_Results

        EXACT_MATCHED = True
        Remove_from_list 1
        Remove_from_list 2
        Write_STAT "CONTEXT STAGE: Select fields that are unique and
              press [2].  Press [3] when none."
        [btn_Context].Enabled = True
        [btn_Context].SetFocus
        [btn_Exact].Enabled = False
        [btn_Finish].Enabled = True
        exact_matches = Val([txt_result_1].Caption)
    End If

Exit_btn_Exact_Click:
    Exit Sub

Err_btn_Exact_Click:
    exact_matches = 0
    EXACT_MATCHED = False
    MsgBox "Procedure:  btn_Exact_Click" & Chr$(13) & Error$
    DoCmd Hourglass False
    Resume Exit_btn_Exact_Click
End Sub

Sub btn_Exact_KeyPress (KeyAscii As Integer)
    btn_Exact_Click
End Sub

Sub btn_finish_Click ()

Dim fname As String, MyQuery  As QueryDef, LogSet As Recordset

    fname = InputBox$("Save output table as:", , "NM_Out")
    If fname = "" Then
        Exit Sub
    End If
    'Del_from_TabledDefs fname

    Del_from_QueryDefs "NM Update Matches"
    Set MyQuery = MyDb.CreateQueryDef("NM Update Matches")

    [cmb_operation].SetFocus
    Select Case [cmb_operation].Text
```

```
        Case "Inner Join"
            GoTo After_Select
        Case "Right Join"
            MyQuery.SQL = "INSERT INTO [NM Matched] SELECT [NM UnMatched
                Two].* FROM [NM UnMatched Two];"
        Case "Left Join"
            MyQuery.SQL = "INSERT INTO [NM Matched] SELECT [NM UnMatched
                One].* FROM [NM UnMatched One];"
        Case "Merge"
            MyQuery.SQL = "INSERT INTO [NM Matched] SELECT [NM UnMatched
                One].* FROM [NM UnMatched One];"
            MyQuery.Execute
            MyQuery.SQL = "INSERT INTO [NM Matched] SELECT [NM UnMatched
                Two].* FROM [NM UnMatched Two];"
    End Select

    MyQuery.Execute

After_Select:
    MyQuery.Close

    Set LogSet = MyDb.OpenRecordset("NM Log")
    LogSet.AddNew
    LogSet!TABLE_A = t1
    LogSet!TABLE_B = t2
    LogSet!Primary_field_A = p1
    LogSet!Primary_field_B = p2
    LogSet!Output_Table = fname
    LogSet!Date = Date$
    LogSet!Time = Time$
    [cmb_operation].SetFocus
    LogSet!Operation_Type = [cmb_operation].Text
    [cmb_domain].SetFocus
    LogSet!Domain_Type = [cmb_domain].Text
    [btn_new].SetFocus
    LogSet!exact_matches = exact_matches
    LogSet!context_matches = context_matches
    LogSet!canonic_matches = canonic_matches
    LogSet!keyword_matches = keyword_matches
    LogSet!soundex_matches = soundex_matches
    LogSet!total_matches = total_matches
    LogSet!User_Queries = man_match + man_nonmatch
    LogSet!Matches_Declared = man_match
    LogSet!NonMatches_Declared = man_nonmatch
    LogSet!UnMatched_in_A = [txt_result_2].Caption
    LogSet!UnMatched_in_B = [txt_result_3].Caption
    LogSet!Output_Records = [txt_total].Caption
    LogSet.Update
    LogSet.Close

    If [table2].sourceobject = "NM Matched" Then
            [table2].sourceobject = "NM Empty Form"
    End If
    Del_From_TableDefs fname
    DoCmd Rename fname, A_TABLE, "NM Matched"
    MyDb.tabledefs.Refresh

    Me![table_one_list].SetFocus
    Me![btn_Exact].Enabled = False
    Me![btn_Context].Enabled = False
    Me![btn_Canonic].Enabled = False
    Me![btn_Keyword].Enabled = False
    Me![btn_Soundex].Enabled = False
    Me![btn_manual].Enabled = False
    Me![btn_match].Enabled = False
    Me![btn_done].Enabled = False
    Me![btn_Cancel].Enabled = True
    Me![btn_new].Enabled = True
    Me![btn_Finish].Enabled = False
    Write_STAT "Matching COMPLETE!!!!"
    Initialize_View fname, "Final Output"

    Del_From_TableDefs "NM UnMatched One"
    Del_From_TableDefs "NM UnMatched Two"
    Del_From_TableDefs "NM Temp Matched"
    Del_From_TableDefs "NM Query Match"
    Del_from_QueryDefs "NM Update Matches"
    Del_from_QueryDefs "NM Update Table A"
    Del_from_QueryDefs "NM Update Table B"

    DoCmd OpenForm "NM Result"
End Sub


Sub btn_Keyword_Click ()
On Error GoTo Err_btn_Keyword_Click

    If Not KEYWORD_MATCHED Then
        DoCmd Hourglass True

        Condition_Set "NM UnMatched One", "A_KEYWORD"
        Condition_Set "NM UnMatched Two", "B_KEYWORD"
        keyword_matches = Update_Tables("A_NM_KEYWORD", "B_NM_KEYWORD", True)
        Update_Results

        If STAGE = FINISHED_STAGE Then
            [btn_new].SetFocus
        Else
            [btn_new].SetFocus
            [btn_Keyword].Enabled = False
            Write_STAT "Next, SOUNDEX STAGE: Uses approximate matching
                algorithm.  Press [5]"
        End If

    KEYWORD_MATCHED = True

    End If
```

```
Exit_btn_Keyword_Click:
    DoCmd Hourglass False
    Exit Sub

Err_btn_Keyword_Click:
    KEYWORD_MATCHED = False
    MsgBox "Procedure:btn_Keyword_Click" & Chr$(13) & Error$
    Resume Exit_btn_Keyword_Click
End Sub

Sub btn_Keyword_KeyPress (KeyAscii As Integer)
    btn_Keyword_Click
End Sub

Sub btn_manual_Click ()
    If Me![table2].sourceobject = "NM Matched" Then
            Make_ViewB_to_B
    End If
    Me![btn_view].Enabled = False
    Me![table].Form.defaultediting = 4   'Can't Add Record
    Me![table].Form.allowediting = True
    Me![table2].Form.allowediting = True
    Me![table2].Form.defaultediting = 4   'Can't Add Record
    [btn_match].Enabled = True
    [btn_done].Enabled = True
    [btn_done].SetFocus
    [btn_manual].Enabled = False
End Sub

Sub btn_manual1_click ()
On Error GoTo Err_btn_manual1_click

    If [btn_manual].Caption = "Manual Matching" Then
            If Me![table2].sourceobject = "NM Matched" Then
                Make_ViewB_to_B
            End If
            Me![btn_view].Enabled = False
            Me![table].Form.defaultediting = 4   'Can't Add Record
            Me![table].Form.allowediting = True
            Me![table2].Form.allowediting = True
            Me![table2].Form.defaultediting = 4   'Can't Add Record
            [btn_match].Enabled = True
            [btn_done].Enabled = True
            [btn_manual].Caption = "Done"

    Else
            Dim MyQuery1 As QueryDef, MyQuery2 As QueryDef, MyQuery3 As QueryDef
            Dim FIELD_LIST As String

            MyWS.BeginTrans
            Set MyQuery1 = MyDb.OpenQueryDef("NM Update Matches")
            Set MyQuery2 = MyDb.OpenQueryDef("NM Update Table A")
            Set MyQuery3 = MyDb.OpenQueryDef("NM Update Table B")
```

```
        FIELD_LIST = Build_Field_List()
        MyQuery1.SQL = "INSERT INTO [NM Matched] SELECT " & FIELD_LIST &
               " FROM [NM UnMatched One] INNER JOIN [NM UnMatched Two]
               ON [NM UnMatched One].[A_NM_MATCH_NUM] = [NM UnMatched
               Two].[B_NM_MATCH_NUM];"
        MyDb.Execute (MyQuery1.name)

        MyQuery2.SQL = "DELETE DISTINCTROW [NM UnMatched One].* FROM [NM
               UnMatched One] WHERE [NM UnMatched One].[A_NM_MATCH_NUM]
               = " & man_match & ";"
        MyQuery3.SQL = "DELETE DISTINCTROW [NM UnMatched Two].* FROM [NM
               UnMatched Two] WHERE [NM UnMatched Two].[B_NM_MATCH_NUM]
               = " & man_match & ";"
        MyDb.Execute (MyQuery2.name)  'Run query.
        MyDb.Execute (MyQuery3.name)  'Run query.
        MyQuery1.Close
        MyQuery2.Close
        MyQuery3.Close

        Update_Results

        MyWS.CommitTrans

        Me![btn_view].Enabled = True
        Me![table].Form.allowediting = False
        Me![table].Form.defaultediting = 3   'Read Only
        Me![table2].Form.allowediting = False
        Me![table2].Form.defaultediting = 3   'Read Only
        [btn_match].Enabled = False
        [btn_done].Enabled = False
        [btn_manual].Caption = "Manual Matching"
   End If

Exit_btn_manual1_click:
   Exit Sub

Err_btn_manual1_click:
   MyWS.Rollback
   MsgBox "Procedure: btn_manual1_click" & Chr$(13) & Error$
   Resume Exit_btn_manual1_click
End Sub

Sub btn_match_Click ()
On Error GoTo Err_btn_match_Click
Dim MyQuery1 As QueryDef, MyQuery2 As QueryDef, MyQuery3 As QueryDef
Dim FIELD_LIST As String

        man_match = man_match + 1
        Set un_one = [table].Form.RecordsetClone
        Set un_two = [table2].Form.RecordsetClone
        un_one.Bookmark = [table].Form.Bookmark
        un_two.Bookmark = [table2].Form.Bookmark

        un_one.Edit
        un_two.Edit
        un_one![A_NM_MATCH_NUM] = man_match
        un_two![B_NM_MATCH_NUM] = man_match
        un_one.Update
        un_two.Update

        Set MyQuery1 = MyDb.OpenQueryDef("NM Update Matches")
        Set MyQuery2 = MyDb.OpenQueryDef("NM Update Table A")
        Set MyQuery3 = MyDb.OpenQueryDef("NM Update Table B")

        MyWS.BeginTrans

        FIELD_LIST = Build_Field_List()
        MyQuery1.SQL = "INSERT INTO [NM Matched] SELECT " & FIELD_LIST &
                " FROM [NM UnMatched One] INNER JOIN [NM UnMatched Two]
                ON [NM UnMatched One].[A_NM_MATCH_NUM] = [NM UnMatched
                Two].[B_NM_MATCH_NUM];"
        MyDb.Execute (MyQuery1.name)

        MyQuery2.SQL = "DELETE DISTINCTROW [NM UnMatched One].* FROM [NM
                UnMatched One] WHERE [NM UnMatched One].[A_NM_MATCH_NUM]
                = " & man_match & ";"
        MyQuery3.SQL = "DELETE DISTINCTROW [NM UnMatched Two].* FROM [NM
                UnMatched Two] WHERE [NM UnMatched Two].[B_NM_MATCH_NUM]
                = " & man_match & ";"
        MyDb.Execute (MyQuery2.name)  'Run query.
        MyDb.Execute (MyQuery3.name)  'Run query.
        MyQuery1.Close
        MyQuery2.Close
        MyQuery3.Close
        MyWS.CommitTrans
        Update_Results

Exit_btn_match_Click:
        Exit Sub

Err_btn_match_Click:
   man_match = man_match - 1
   MyWS.Rollback
   MsgBox "Procedure btn_match_Click." & Chr$(13) & Error$
   Resume Exit_btn_match_Click

End Sub

Sub btn_match1_Click ()
On Error GoTo Err_btn_match1_Click


        man_match = man_match + 1
        Set un_one = [table].Form.RecordsetClone
        Set un_two = [table2].Form.RecordsetClone
        un_one.Bookmark = [table].Form.Bookmark
        un_two.Bookmark = [table2].Form.Bookmark
        un_one.Edit
        un_two.Edit
        If IsNull(un_one![A_NM_MATCH_NUM]) Then
           If IsNull(un_two![B_NM_MATCH_NUM]) Then
                 un_one![A_NM_MATCH_NUM] = man_match
                 un_two![B_NM_MATCH_NUM] = man_match
           Else
                 un_one![A_NM_MATCH_NUM] = un_two![B_NM_MATCH_NUM]
           End If
        Else
           If IsNull(un_two![B_NM_MATCH_NUM]) Then
                 un_two![B_NM_MATCH_NUM] = un_one![A_NM_MATCH_NUM]
           Else
                 MsgBox "Both records are already matched to other records."
                 GoTo Exit_btn_match1_click
           End If
        End If

        un_one.Update
        un_two.Update


Exit_btn_match1_click:
        Exit Sub

Err_btn_match1_Click:
   MsgBox "Procedure btn_match_Click." & Chr$(13) & Error$
   Resume Exit_btn_match1_click

End Sub

Sub btn_New_Click ()
   Form_Load
End Sub

Sub btn_Soundex_Click ()
On Error GoTo Err_btn_Soundex_Click

   If Not SOUNDEXED Then
        DoCmd Hourglass True

        Condition_Set "NM UnMatched One", "A_SOUNDEX"
        Condition_Set "NM UnMatched Two", "B_SOUNDEX"
        soundex_matches = Update_Tables("A_NM_SOUNDEX", "B_NM_SOUNDEX", True)
        'Update_Tables "A_NM_SOUNDEX_VAL", "B_NM_SOUNDEX_VAL", False
        Update_Results

        If STAGE = FINISHED_STAGE Then
           [btn_new].SetFocus
        Else
           [btn_manual].SetFocus
           [btn_Soundex].Enabled = False
           Write_STAT "Next, Do Manual matching."
        End If
        SOUNDEXED = True
    End If

Exit_btn_Soundex_Click:
    DoCmd Hourglass False
    Exit Sub

Err_btn_Soundex_Click:
    SOUNDEXED = False
    MsgBox "Procedure:btn_Soundex_Click" & Chr$(13) & Error$
    Resume Exit_btn_Soundex_Click

End Sub

Sub btn_view_Click ()
On Error GoTo Err_btn_view_Click

    If Me![table2].sourceobject = "NM Matched" Then
        Make_ViewB_to_B
    Else
        Make_ViewB_to_C
    End If

Exit_btn_view_Click:
    Exit Sub

Err_btn_view_Click:
    MsgBox Error$, , "Table Editing - btn_view_Click"
    Resume Exit_btn_view_Click

End Sub

Function Build_Alias ()
On Error GoTo Err_Build_Alias

    Dim N As Integer, M As Integer, I As Integer, Max As Integer, result
        As String, fname As String

    M = MyDb.tabledefs(t1).Fields.Count
    N = MyDb.tabledefs(t2).Fields.Count
    result = ""

    If N > M Then
        Max = N
    Else
        Max = M
    End If

    For I = 0 To Max - 1
        If I < M Then
            fname = MyDb.tabledefs(t1).Fields(I).name
            result = result & "[" & t1 & "].[" & fname & "] AS A_" &
                fname & ", "
        End If
```

```
            If I < N Then
                fname = MyDb.tabledefs(t2).Fields(I).name
                result = result & "[" & t2 & "].[" & fname & "] AS B_" &
                        fname & ", "
            End If
    Next I

    Build_Alias = Left$(result, Len(result) − 2)

Exit_Build_Alias:
    DoCmd Echo True
    Exit Function

Err_Build_Alias:
    MsgBox Err & " :   " & Error, , "Build_Alias"
    Build_Alias = ""
    Resume Exit_Build_Alias



End Function

Function Build_Alias_Field_List (table_name As String, Prefix As String)
            As String
    On Error GoTo Err_Build_Alias_Field_List

    Dim N As Integer, M As Integer, result As String, fname As String

    N = 0
    M = MyDb.tabledefs(table_name).Fields.Count
    result = ""

    Do
            fname = MyDb.tabledefs(table_name).Fields(N).name
            result = result & "[" & table_name & "].[" & fname & "] AS " &
                        Prefix & fname & ", "
            N = N + 1
    Loop Until N >= M

    Build_Alias_Field_List = Left$(result, Len(result) − 2)

Exit_Build_Alias_Field_List:
    DoCmd Echo True
    Exit Function

Err_Build_Alias_Field_List:
    MsgBox Err & " :   " & Error, , "Build_Alias_Field_List"
    Build_Alias_Field_List = ""
    Resume Exit_Build_Alias_Field_List

End Function

Function Build_Field_List () As String
    On Error GoTo Err_Build_Field_List
    Dim N As Integer, M As Integer, I As Integer, Max As Integer, result
            As String, fname As String


    M = MyDb.tabledefs("NM UnMatched One").Fields.Count
    N = MyDb.tabledefs("NM UnMatched Two").Fields.Count
    result = ""

    If N > M Then
            Max = N
    Else
            Max = M
    End If

    For I = 0 To Max - 1
            If I < M Then
                fname = MyDb.tabledefs("NM UnMatched One").Fields(I).name
                result = result & "[NM UnMatched One].[" & fname & "], "
            End If
            If I < N Then
                fname = MyDb.tabledefs("NM UnMatched Two").Fields(I).name
                result = result & "[NM UnMatched Two].[" & fname & "], "
            End If
    Next I

    Build_Field_List = Left$(result, Len(result) - 2)

Exit_Build_Field_List:
    DoCmd Echo True
    Exit Function

Err_Build_Field_List:
    MsgBox Err & "  :   " & Error, , "Build_Field_List"
    Build_Field_List = ""
    Resume Exit_Build_Field_List

End Function

Function Build_Temp_Field_List () As String
    On Error GoTo Err_Build_Temp_Field_List

    Dim N As Integer, M As Integer, result As String, fname As String

    N = 0
    M = MyDb.tabledefs("NM Temp Matched").Fields.Count
    result = ""

    Do
            fname = MyDb.tabledefs("NM Temp Matched").Fields(N).name
            result = result & "[NM Temp Matched].[" & fname & "], "
            N = N + 1
    Loop Until N >= M
    N = 0

    Build_Temp_Field_List = Left$(result, Len(result) - 2)

Exit_Build_Temp_Field_List:
    DoCmd Echo True
    Exit Function

Err_Build_Temp_Field_List:
    MsgBox Err & " :  " & Error, , "Build_Temp_Field_List"
    Build_Temp_Field_List = ""
    Resume Exit_Build_Temp_Field_List

End Function

Sub Clear_STAT ()
    Me![txt_status].Caption = ""
End Sub

Function Condition_Record_to_Canonic (txt As String) As String
On Error GoTo Err_Condition_Record_to_Canonic

Dim LastSpc As Integer, Length As Integer
Dim EndToken As String, Temp As String
Dim Criteria As String, MySet As Recordset, ReplaceSet As Recordset

    Set MySet = MyDb.OpenRecordset("NM Condition Table",
                    DB_OPEN_DYNASET)    ' Create Recordset.
    Set ReplaceSet = MyDb.OpenRecordset("NM Replace Table",
                    DB_OPEN_DYNASET)    ' Create Recordset.

    Temp = txt
    ReplaceSet.MoveFirst
    Do Until ReplaceSet.EOF
            Criteria = Mid$(ReplaceSet![Find String], 2,
                    Len(ReplaceSet![Find String]) - 2)
            If InStr(Temp, Criteria) Then
                Replace_String Temp, Criteria, Mid$(ReplaceSet![Replace
                    String], 2, Len(ReplaceSet![Replace String]) - 2)
            End If
            ReplaceSet.MoveNext
    Loop

    Length = Len(Temp)
    LastSpc = Find_Last_Space(Temp)
    If LastSpc > 0 Then
            EndToken = Mid$(Temp, LastSpc + 1, Length - LastSpc)
            Criteria = "EndToken = '" & EndToken & "'"   'Define search criteria.
            MySet.FindFirst Criteria    ' Locate first occurrence.
            If Not MySet.NoMatch Then
                Temp = Left$(Temp, LastSpc - 1)
            End If
            MySet.Close
    End If
```

```
Exit_Condition_Record_to_Canonic:
   Condition_Record_to_Canonic = Temp
   Exit Function

Err_Condition_Record_to_Canonic:
   MsgBox "Procedure: Condition_Record_to_Canonic" & Chr$(13) & Error$
   Resume Exit_Condition_Record_to_Canonic
End Function

Function Condition_Record_to_KeyWord (txt As String) As String
On Error GoTo Err_Condition_Record_to_KeyWord

Dim Length As Integer, FirstSpc As Integer
Dim Temp As String, FirstToken As String
Dim Criteria As String, MySet As Recordset

   Temp = txt

   Length = Len(Temp)
   FirstSpc = InStr(Temp, Chr$(32))
   If (FirstSpc > 1) Then
         FirstToken = Left$(Temp, FirstSpc - 1)
         Criteria = "FirstToken = '" & FirstToken & "'"   'Define search
                  criteria.
         Set MySet = MyDb.OpenRecordset("NM Condition Table",
                  DB_OPEN_DYNASET)   ' Create Recordset.
         MySet.FindFirst Criteria   ' Locate first occurrence.
         If Not MySet.NoMatch Then
            Temp = Mid$(Temp, FirstSpc + 1)
         End If
         MySet.Close
   End If

Exit_Condition_Record_to_KeyWord:
   Condition_Record_to_KeyWord = Find_FirstToken(Temp)
   Exit Function

Err_Condition_Record_to_KeyWord:
   MsgBox "Procedure: Condition_Record_to_KeyWord" & Chr$(13) & Error$
   Resume Exit_Condition_Record_to_KeyWord

End Function

Function Condition_Record_to_Soundex (txt As String) As String
On Error GoTo Err_Condition_Record_to_Soundex

Dim Length As Integer, FirstSpc As Integer, I As Integer
Dim Temp As String, FirstToken As String, Char As String, Last_Char As String

   Temp = ""
   Last_Char = ""
   Char = ""
   Length = Len(txt)
   If Length <= 0 Then
         GoTo Exit_Condition_Record_to_Soundex
   End If
   For I = 1 To Length
         Char = Mid$(txt, I, 1)
         If (Char Like "[a-z]") Then
            If Last_Char <> "" Then
                  Select Case Char
                     Case "a", "e", "i", "o", "u", "h", "w", "y"
                           Char = Last_Char
                     Case "b", "f", "p", "v"
                           Char = "1"
                     Case "c", "g", "j", "k", "q", "s", "x", "z"
                           Char = "2"
                     Case "d", "t"
                           Char = "3"
                     Case "l"
                           Char = "4"
                     Case "m", "n"
                           Char = "5"
                     Case "r"
                           Char = "6"
                  End Select
            End If
            If Last_Char <> Char Then
                  Temp = Temp & Char
                  Last_Char = Char
            End If
         Else
            If (Temp <> "") And (Char = Chr$(32) And (Last_Char <> Char)) Then
                  Temp = Temp & Chr$(32)
                  Last_Char = ""
            End If
         End If
   Next I


Exit_Condition_Record_to_Soundex:
   Condition_Record_to_Soundex = Temp
   Exit Function

Err_Condition_Record_to_Soundex:
   MsgBox "Procedure: Condition_Record_to_Soundex" & Chr$(13) & Error$
   Resume Exit_Condition_Record_to_Soundex
End Function

Sub Condition_Set (Tbl As String, MorphType As String)
On Error GoTo Err_Condition_Set

Dim MySet As Recordset, Temp As String

         Set MySet = MyDb.OpenRecordset(Tbl)
```

```
      If MySet.EOF Then
        GoTo Exit_Condition_Set
      Else
        MySet.MoveFirst
        Do Until MySet.EOF
            MySet.Edit
            Select Case MorphType
              Case "A_CANONIC"
                  MySet![A_NM_CANONIC] =
            Condition_Record_to_Canonic(CStr(MySet![A_NM_CANONIC]))
                  MySet![A_NM_CANONIC] =
            LookUp_Alias(CStr(MySet![A_NM_CANONIC]))
              Case "A_KEYWORD"
                  MySet![A_NM_KEYWORD] =
            Condition_Record_to_KeyWord(CStr(MySet![A_NM_CANONIC]))
              Case "B_CANONIC"
                  MySet![B_NM_CANONIC] =
            Condition_Record_to_Canonic(CStr(MySet![B_NM_CANONIC]))
                  MySet![B_NM_CANONIC] =
            LookUp_Alias(CStr(MySet![B_NM_CANONIC]))
              Case "B_KEYWORD"
                  MySet![B_NM_KEYWORD] =
            Condition_Record_to_KeyWord(CStr(MySet![B_NM_CANONIC]))
              Case "A_SOUNDEX"
                  Temp =
            Condition_Record_to_Soundex(CStr(MySet![A_NM_CANONIC]))
                  MySet![A_NM_SOUNDEX] = Temp
                  MySet![A_NM_SOUNDEX_VAL] = Val(Temp)
              Case "B_SOUNDEX"
                  Temp =
            Condition_Record_to_Soundex(CStr(MySet![B_NM_CANONIC]))
                  MySet![B_NM_SOUNDEX] = Temp
                  MySet![B_NM_SOUNDEX_VAL] = Val(Temp)
            End Select
            MySet.Update
            MySet.MoveNext
        Loop
        MySet.MoveFirst
        MySet.Close
      End If

Exit_Condition_Set:
  Exit Sub

Err_Condition_Set:
  MsgBox "Procedure:  Condition_Set:" & Chr$(13) & Error$
  GoTo Exit_Condition_Set


End Sub

Function Create_NM_form (table_name As String)
  On Error GoTo Err_Create_NM_form
```

```
   Dim frmcreate As Form, x As Integer, y As Integer, mycontrol As Control
   Dim N As Integer, M As Integer, result As String, fname As String

   DoCmd Echo False

   x = 0
   y = 0

   Set frmcreate = CreateForm()
   frmcreate.viewsallowed = 2
   frmcreate.defaultview = 2
   frmcreate.recordsource = table_name
   M = MyDb.tabledefs(table_name).Fields.Count
   N = 0
   frmcreate.section(0).Height = M * 300
   Do
        fname = MyDb.tabledefs(table_name).Fields(N).name
        Set mycontrol = CreateControl(frmcreate.name, 109, 0, "", fname,
                   x, y, 1000, 200)
        mycontrol.controlsource = fname
        mycontrol.name = fname
        y = y + 300
        N = N + 1
   Loop Until N >= M

Exit_Create_NM_form:
   DoCmd Echo True
   Exit Function

Err_Create_NM_form:
   MsgBox Error$, 0 Or 48, "Create_NM_form"
   Resume Exit_Create_NM_form

End Function

Sub Del_from_QueryDefs (q_name As String)
   Dim I As Integer

        For I = 0 To MyDb.QueryDefs.Count - 1
          If MyDb.QueryDefs(I).name = q_name Then
                 MyDb.QueryDefs.Delete q_name
                 Exit For
          End If
        Next I
End Sub

Sub Del_From_TableDefs (table_name As String)
   Dim I As Integer

        For I = 0 To MyDb.tabledefs.Count - 1
          If MyDb.tabledefs(I).name = table_name Then
                 MyDb.tabledefs.Delete table_name
                 Exit For
          End If
```

```
      Next I
End Sub

Function Find_FirstToken (txtstr As String) As String

Dim FrstSpc As Integer, ScndSpc As Integer, Length As Integer

  Find_FirstToken = txtstr
  Length = Len(txtstr)
  If Length = 0 Then
      Exit Function
  Else
      FrstSpc = InStr(txtstr, Chr$(32))
      If (FrstSpc <= 0) Then
        Exit Function
      ElseIf (FrstSpc <= 2) Then
        ScndSpc = InStr(Mid$(txtstr, FrstSpc + 1, Length), Chr$(32))
        If (ScndSpc > 0) Then
            Find_FirstToken = Left$(txtstr, (FrstSpc + ScndSpc − 1))
        End If
      Else
        Find_FirstToken = Left$(txtstr, FrstSpc − 1)
      End If
  End If

End Function

Function Find_Last_Space (txt As String) As Integer

Dim SpcPos As Integer, LastPos As Integer

  SpcPos = 0
  Find_Last_Space = 0

  Do While Not (txt = "")
      SpcPos = InStr(SpcPos + 1, txt, Chr(32))
      If SpcPos = 0 Then
        Find_Last_Space = LastPos
        Exit Function
      Else
        LastPos = SpcPos
      End If
  Loop

End Function

Sub Form_Load ()
  Me![table_one_list].Locked = False
  Me![table_two_list].Locked = False
  Me![table_one_list].rowsource = get_table_list()
  Me![table_two_list].rowsource = get_table_list()
  Me![btn_Exact].Enabled = True
  Me![btn_Context].Enabled = False
  Me![btn_Canonic].Enabled = False
```

```
   Me![btn_Keyword].Enabled = False
   Me![btn_Soundex].Enabled = False
   Me![btn_Finish].Enabled = False
   Me![btn_Exact].Enabled = True
   Me![txt_result_1].Caption = "0"
   Me![txt_result_2].Caption = "0"
   Me![txt_result_3].Caption = "0"
   Me![txt_total].Caption = "0"
   Me![btn_match].Enabled = False
   Me![btn_done].Enabled = False
   Me![btn_manual].Enabled = False
   Me![btn_view].Enabled = False
   Me![field_one_list].rowsource = ""
   Me![field_two_list].rowsource = ""
   Me![field_one_list] = ""
   Me![field_one_list] = ""
   Me![txt_primary_1].Caption = ""
   Me![txt_primary_2].Caption = ""
   Me![table2].Visible = True
   Me![table].Height = 3478
   Me![table].sourceobject = "NM Empty Form"
   Me![table_label].Caption = ""
   Me![table2].sourceobject = "NM Empty Form"
   Me![table2_label1].Caption = ""
   Me![table2_label2].Caption = ""
   MATCH_VIEW_CREATED = False
   A_VIEW_CREATED = False
   B_VIEW_CREATED = False
   EXACT_MATCHED = False
   CANONICIZED = False
   KEYWORD_MATCHED = False
   SOUNDEXED = False
   man_match = 0
   man_nonmatch = 0
   exact_matches = 0
   context_matches = 0
   canonic_matches = 0
   keyword_matches = 0
   soundex_matches = 0
   total_matches = 0
   Set MyWS = DBEngine.Workspaces(0)
   Set MyDb = MyWS.Databases(0)

   Write_STAT "Choose the tables and primary fields which are to be matched."
End Sub

Function Initialize_NM () As Integer
'Return PASS if successful, FAIL if not
On Error GoTo err_init
Dim myset1 As Recordset, myset2 As Recordset
Dim FIELD_LIST_A As String, FIELD_LIST_B As String, FIELD_LIST As String

      Write_STAT "Checking for exact match in primary fields."
```

```
Set myset1 = MyDb.OpenRecordset(t1, DB_OPEN_TABLE)
Set myset2 = MyDb.OpenRecordset(t2, DB_OPEN_TABLE)

Del_From_TableDefs "NM Matched"
Del_From_TableDefs "NM UnMatched One"
Del_From_TableDefs "NM UnMatched Two"
Del_from_QueryDefs "NM Update Matches"
Del_from_QueryDefs "NM Update Table A"
Del_from_QueryDefs "NM Update Table B"


Dim MyQuery1 As QueryDef, MyQuery2 As QueryDef, MyQuery3 As
        QueryDef, SQL1 As String, SQL2 As String, SQL3 As String
Dim orig As String, orig_p2 As String

Set MyQuery1 = MyDb.CreateQueryDef("NM Update Matches")
Set MyQuery2 = MyDb.CreateQueryDef("NM Update Table A")
Set MyQuery3 = MyDb.CreateQueryDef("NM Update Table B")

orig_p1 = Right$(p1, Len(p1) − 2)
orig_p2 = Right$(p2, Len(p2) − 2)

FIELD_LIST_A = Build_Alias_Field_List(t1, "A_")
FIELD_LIST_B = Build_Alias_Field_List(t2, "B_")
MyQuery2.SQL = "SELECT DISTINCTROW " & FIELD_LIST_A & ", [" & t1
        & "].[" & orig_p1 & "] AS A_NM_CANONIC INTO [NM
        UnMatched One] FROM [" & t1 & "] LEFT JOIN [NM Matched]
        ON [" & t1 & "].[" & orig_p1 & "] = [NM Matched].[" & p1
        & "] WHERE ([NM Matched].[" & p1 & "] Is Null);"
SQL1 = "SELECT DISTINCTROW " & FIELD_LIST_B & ", [" & t2 & "].["
        & orig_p2 & "] AS B_NM_CANONIC INTO [NM UnMatched Two]
        FROM [" & t2 & "] LEFT JOIN [NM Matched] ON [" & t2 &
        "].[" & orig_p2 & "] = [NM Matched].[" & p2 & "] WHERE
        ([NM Matched].[" & p2 & "] Is Null);"
Debug.Print SQL1
MyQuery3.SQL = SQL1
FIELD_LIST = Build_Alias()
SQL1 = "SELECT DISTINCTROW " & FIELD_LIST & ", [" & t1 & "].[" &
        orig_p1 & "] AS A_NM_CANONIC, [" & t2 & "].[" & orig_p2
        & "] AS B_NM_CANONIC INTO [NM Matched] FROM [" & t1 & "]
        INNER JOIN [" & t2 & "] ON [" & t1 & "].[" & orig_p1 &
        "] = [" & t2 & "].[" & orig_p2 & "];"
MyQuery1.SQL = SQL1
Debug.Print SQL1
Debug.Print MyQuery2.SQL
Debug.Print MyQuery3.SQL
MyQuery1.Execute
MyQuery2.Execute
MyQuery3.Execute
MyDb.tabledefs.Refresh

MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [A_NM_MATCH_NUM] SHORT;"
MyQuery1.Execute
```

```
MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [B_NM_MATCH_NUM] SHORT;"
MyQuery1.Execute
MyQuery2.SQL = "ALTER TABLE [NM UnMatched One] ADD COLUMN
        [A_NM_MATCH_NUM] SHORT;"
MyQuery3.SQL = "ALTER TABLE [NM UnMatched Two] ADD COLUMN
        [B_NM_MATCH_NUM] SHORT;"
MyQuery2.Execute
MyQuery3.Execute

MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [A_NM_KEYWORD] TEXT;"
MyQuery1.Execute
MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [B_NM_KEYWORD] TEXT;"
MyQuery1.Execute
MyQuery2.SQL = "ALTER TABLE [NM UnMatched One] ADD COLUMN
        [A_NM_KEYWORD] TEXT;"
MyQuery3.SQL = "ALTER TABLE [NM UnMatched Two] ADD COLUMN
        [B_NM_KEYWORD] TEXT;"
MyQuery2.Execute
MyQuery3.Execute

MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [A_NM_SOUNDEX] TEXT;"
MyQuery1.Execute
MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [B_NM_SOUNDEX] TEXT;"
MyQuery1.Execute
MyQuery2.SQL = "ALTER TABLE [NM UnMatched One] ADD COLUMN
        [A_NM_SOUNDEX] TEXT;"
MyQuery3.SQL = "ALTER TABLE [NM UnMatched Two] ADD COLUMN
        [B_NM_SOUNDEX] TEXT;"
MyQuery2.Execute
MyQuery3.Execute

MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [A_NM_SOUNDEX_VAL] SHORT;"
MyQuery1.Execute
MyQuery1.SQL = "ALTER TABLE [NM Matched] ADD COLUMN
        [B_NM_SOUNDEX_VAL] SHORT;"
MyQuery1.Execute
MyQuery2.SQL = "ALTER TABLE [NM UnMatched One] ADD COLUMN
        [A_NM_SOUNDEX_VAL] SHORT;"
MyQuery3.SQL = "ALTER TABLE [NM UnMatched Two] ADD COLUMN
        [B_NM_SOUNDEX_VAL] SHORT;"
MyQuery2.Execute
MyQuery3.Execute

MyQuery1.Close
MyQuery2.Close
MyQuery3.Close

myset1.Close
```

```
        myset2.Close


        Initialize_View "NM UnMatched One", "A"
        Initialize_View "NM UnMatched Two", "B"
        Initialize_View "NM Matched", "C"

        Initialize_NM = PASS

exit_init:
    Exit Function

err_init:
    Select Case Err
        Case 13 'Type Mismatch
            MsgBox "The two selected fields have incompatible types."
        Case Else
            MsgBox "Procedure:Initialize NM." & Chr$(13) & Error$
    End Select
    Initialize_NM = FAIL
    Resume exit_init

End Function

Sub Initialize_View (table_name As String, view As String)
On Error GoTo Err_Init_View

    DoCmd Echo False
    DoCmd SetWarnings False

    Dim dummy As Variant, F As Form

    dummy = Create_NM_form(table_name)
    Set F = screen.activeform
    F.defaultediting = 3   'Read Only
    F.allowediting = False
    F.allowupdating = 1
    SendKeys table_name & "{enter}", False
    DoCmd DoMenuItem 3, a_file, a_saveformas, , a_menu_ver20
    DoCmd Close

    If view = "A" Then
        Me!table_label.Caption = "Table A:   " & table_name
        Me!table_label.ForeColor = TABLE_COLOR
        Me!table.sourceobject = table_name
    ElseIf view = "B" Then
        Make_ViewB_to_B
    ElseIf view = "Query" Then
        Me!table_label.Caption = "Table Query Match:      " & table_name
        Me!table_label.ForeColor = TABLE_COLOR
        Me!table.sourceobject = table_name
        Me!table2.sourceobject = "NM Empty Form"
    ElseIf view = "Final Output" Then
        [table].Height = 7954
        [table].sourceobject = table_name
        [table_label].Caption = "Final Output Table:   " & table_name
        [table_label].ForeColor = S_COLOR
        [table2].sourceobject = "NM Empty Form"
        [table2].Visible = False
    ElseIf view = "System" Then
        '[table].sourceobject = table_name
        '[table_label].Caption = "System Table:   " & table_name
        '[table_label].ForeColor = S_COLOR
        '[table].Form.allowediting = True
        '[table].Form.defaultediting = 2    'Allow Edits
        '[table2].sourceobject = "NM Empty Form"
        '[table2].Visible = False
    End If

Exit_Init_View:
    DoCmd Echo True
    DoCmd SetWarnings True
    Exit Sub

Err_Init_View:
    MsgBox "Procedure:  Initialize_View " & Chr$(13) & Error$
    Resume Exit_Init_View

End Sub

Function LookUp_Alias (txt As String) As String
On Error GoTo Err_LookUp_Alias
Dim MyTable As Recordset, I As Integer

    LookUp_Alias = txt
    Set MyTable = MyDb.OpenRecordset("NM System", DB_OPEN_TABLE)

    For I = 1 To 4
        MyTable.Index = "Alias_" & I
        MyTable.Seek "=", txt
        If Not MyTable.NoMatch Then
            LookUp_Alias = MyTable!COMPANY_NAME_CANONIC
            Exit Function
        End If
    Next I

    MyTable.Close            ' Close table.

Exit_LookUp_Alias:
    Exit Function

Err_LookUp_Alias:
    MsgBox "Procedure:  LookUp_Alias" & Chr$(13) & Error$
    Resume Exit_LookUp_Alias
End Function

Sub Make_ViewB_to_B ()
        Me![btn_view].Caption = "View Matches"
        Me!table2_label1.Caption = "Table B:"
        Me!table2_label2.Caption = "NM UnMatched Two"
        Me!table2_label1.ForeColor = TABLE_COLOR
        Me!table2_label2.ForeColor = TABLE_COLOR
        Me!table2.sourceobject = "NM UnMatched Two"
End Sub

Sub Make_ViewB_to_C ()
    Me![btn_view].Caption = "View Table B"
    Me![table2_label1].Caption = "Output Table:"
    Me![table2_label2].Caption = "NM Matched"
    Me![table2_label1].ForeColor = S_COLOR
    Me![table2_label2].ForeColor = S_COLOR
    Me![table2].sourceobject = "NM Matched"
End Sub

Sub Remove_from_list (contrl As Integer)
    Dim list As String, item As String

    If contrl = 1 Then
        list = Me!field_one_list.rowsource
        item = p1
    ElseIf contrl = 2 Then
        list = Me!field_two_list.rowsource
        item = p2
    End If

    Replace_String list, item & ";", ""

    If contrl = 1 Then
        Me!field_one_list.rowsource = list
    Else
        Me!field_two_list.rowsource = list
    End If

End Sub

Sub Table_one_list_AfterUpdate ()
    TABLE_A = table_one_list
    Me!field_one_list.rowsource = get_NM_field_list(TABLE_A, "A_")
End Sub

Sub Table_one_list_KeyDown (KeyCode As Integer, Shift As Integer)
    Table_one_list_AfterUpdate
End Sub

Sub table_two_list_AfterUpdate ()
    TABLE_B = table_two_list
    Me!field_two_list.rowsource = get_NM_field_list(TABLE_B, "B_")
    'button89_Click
End Sub

Sub Table_two_list_KeyDown (KeyCode As Integer, Shift As Integer)
    table_two_list_AfterUpdate
```

```
End Sub

Sub tgl_Condition_AfterUpdate ()
   If [tgl_Condition] Then
         [table].sourceobject = "NM Condition Table"
         [table_label].Caption = "System Table:  NM Condition Table"
         [table].Form.allowediting = True
         [table].Form.defaultediting = 2   'Allow Edits
         [table2].sourceobject = "NM Replace Table"
         [table2_label1].Caption = "System Table:"
         [table2_label2].Caption = "NM Replace Table"
         [table2].Form.allowediting = True
         [table2].Form.defaultediting = 2   'Allow Edits
   Else
         [table].sourceobject = "NM Empty Form"
         [table_label].Caption = ""
         [table2].sourceobject = "NM Empty Form"
         [table2_label1].Caption = ""
         [table2_label2].Caption = ""
   End If
End Sub

Sub tgl_System_AfterUpdate ()
   If [tgl_System] Then
         [table2].Visible = False
         [table3].Visible = True
         [table3_label].Visible = True
         [table3].sourceobject = "NM System"
         [table3_label].Caption = "System Table:  NM System"
         [table3].Form.allowediting = True
         '[table3].Form.defaultview = 1   'Continuous
         [table3].Height = 7954
         [table3].Left = 6870
         [table3].Top = 540
         [table3].Width = 7530
         [table3].Form.scrollbars = 3
   Else
         [table2].Visible = True
         [table3].Visible = False
         [table3_label].Visible = False
   End If
End Sub

Sub Update_Results ()
         Set matched_set = MyDb.OpenRecordset("NM Matched")
         If matched_set.EOF Then
            Write_STAT "No exact matches found."
         End If
         Set un_one = MyDb.OpenRecordset("NM UnMatched One")
         Set un_two = MyDb.OpenRecordset("NM UnMatched Two")

         If matched_set.EOF Then
            Me![txt_result_1].Caption = "0"
         Else
```

```
    matched_set.MoveLast
    Me![txt_result_1].Caption = matched_set.RecordCount
    total_matches = matched_set.RecordCount
End If

If un_one.EOF Then
    Me![txt_result_2].Caption = "0"
    STAGE = FINISHED_STAGE
Else
    un_one.MoveLast
    Me![txt_result_2].Caption = un_one.RecordCount
End If

If un_two.EOF Then
    Me![txt_result_3].Caption = "0"
    STAGE = FINISHED_STAGE
Else
    un_two.MoveLast
    Me![txt_result_3].Caption = un_two.RecordCount
End If

matched_set.Close
un_one.Close
un_two.Close

[cmb_operation].SetFocus
Select Case [cmb_operation].Text
    Case "Inner Join"
        [txt_total].Caption = Val([txt_result_1].Caption)
    Case "Right Join"
        [txt_total].Caption = (Val([txt_result_1].Caption) +
                Val([txt_result_3].Caption))
    Case "Left Join"
        [txt_total].Caption = (Val([txt_result_1].Caption) +
                Val([txt_result_2].Caption))
    Case "Merge"
        [txt_total].Caption = (Val([txt_result_1].Caption) +
                Val([txt_result_2].Caption) +
                Val([txt_result_3].Caption))
End Select

If STAGE = FINISHED_STAGE Then
    btn_finish_Click
Else
    If Me![table2].sourceobject = "NM Matched" Then
        Make_ViewB_to_C
    Else
        Make_ViewB_to_B
    End If
    Me!table.sourceobject = "NM UnMatched One"

    If [btn_manual].Caption = "Done" Then
        Me![table].Form.defaultediting = 4   'Can't Add Record
        Me![table].Form.allowediting = True
```

```
                Me![table2].Form.allowediting = True
                Me![table2].Form.defaultediting = 4   'Can't Add Record
            End If
        End If
End Sub

Function Update_Tables (field1 As String, field2 As String, Query_Flag
        As Integer) As Integer
On Error GoTo Err_Update_Tables
Dim MyQuery1 As QueryDef, MyQuery2 As QueryDef, MyQuery3 As QueryDef
Dim temp_matched_set As Recordset
Dim FIELD_LIST As String

        MyWS.BeginTrans

        [table].sourceobject = "NM Empty Form"
        [table2].sourceobject = "NM Empty Form"
        Set MyQuery1 = MyDb.OpenQueryDef("NM Update Matches")
        Set MyQuery2 = MyDb.OpenQueryDef("NM Update Table A")
        Set MyQuery3 = MyDb.OpenQueryDef("NM Update Table B")

        MyQuery2.SQL = "CREATE INDEX NM_INDX_" & field1 & " ON [NM
                UnMatched One](" & field1 & ") WITH IGNORE NULL;"
        MyDb.Execute (MyQuery2.name)
        MyQuery3.SQL = "CREATE INDEX NM_INDX_" & field2 & " ON [NM
                UnMatched Two](" & field2 & ") WITH IGNORE NULL;"
        MyDb.Execute (MyQuery3.name)
        MyDb.tabledefs.Refresh

        Del_From_TableDefs "NM Temp Matched"
        FIELD_LIST = Build_Field_List()
        MyQuery1.SQL = "SELECT DISTINCTROW " & FIELD_LIST & " INTO [NM
                Temp Matched] FROM [NM UnMatched One] INNER JOIN [NM
                UnMatched Two] ON [NM UnMatched One].[" & field1 & "] =
                [NM UnMatched Two].[" & field2 & "];"
        Debug.Print MyQuery1.SQL
        MyDb.Execute (MyQuery1.name)
        MyDb.tabledefs.Refresh

        If Query_Flag Then
            Dim MySet As Recordset, Response As Integer, Msg As String

            FIELD_LIST = Build_Temp_Field_List()
            Set MySet = MyDb.OpenRecordset("SELECT DISTINCTROW " &
                FIELD_LIST & " FROM [NM Temp Matched] WHERE " & field1 &
                " In (SELECT [" & field1 & "] FROM [NM Temp Matched] As
                Tmp GROUP BY [" & field1 & "],[" & field2 & "] HAVING
                Count(*)>=1  And [" & field2 & "] = [NM Temp Matched].["
                & field2 & "]) ORDER BY [" & field1 & "], [" & field2 &
                "];")
            MyDb.tabledefs.Refresh

            Do Until MySet.EOF
                Msg = "Is this pair a match?" & Chr$(13) & Chr$(13)
            Msg = Msg & "(1)  " & MySet.Fields(p1) & Chr$(13)
            Msg = Msg & "(2)  " & MySet.Fields(p2) & Chr$(13)
            Response = MsgBox(Msg, MB_YESNOCANCEL + MB_ICONQUESTION,
                  "NM User Query")
            Select Case Response
              Case IDYES
                  man_match = man_match + 1
              Case IDNO
                  man_nonmatch = man_nonmatch + 1
                  MySet.Delete
              Case IDCANCEL
                  Write_STAT "NAME MATCHING was Cancelled."
                  MyWS.Rollback
                  GoTo Exit_Update_Tables
            End Select
            MySet.MoveNext
        Loop
        MySet.Close
      End If

      MyQuery2.SQL = "DELETE [NM UnMatched One].* FROM [NM UnMatched
            One] INNER JOIN [NM Temp Matched] ON [NM UnMatched
            One].[" & field1 & "] = [NM Temp Matched].[" & field1 &
            "];"
      MyQuery3.SQL = "DELETE [NM UnMatched Two].* FROM [NM UnMatched
            Two] INNER JOIN [NM Temp Matched] ON [NM UnMatched
            Two].[" & field2 & "] = [NM Temp Matched].[" & field1 &
            "];"
      MyDb.Execute (MyQuery2.name)  ' Run query.
      MyDb.Execute (MyQuery3.name)  ' Run query.
      MyQuery1.SQL = "INSERT INTO [NM Matched] SELECT [NM Temp
            Matched].* FROM [NM Temp Matched];"
      MyDb.Execute (MyQuery1.name)

      MyQuery1.Close
      MyQuery2.Close
      MyQuery3.Close

      Set MySet = MyDb.OpenRecordset("NM Temp Matched")
      MySet.MoveLast
      Update_Tables = MySet.RecordCount
      MySet.Close

      MyWS.CommitTrans

Exit_Update_Tables:
      Exit Function

Err_Update_Tables:
  Select Case Err
      Case 13 'Type Mismatch
        Write_STAT "The two selected fields have incompatible types."
      Case Else
        MsgBox "Procedure Update_Tables." & Chr$(13) & Error$ &
```

```
                  Chr$(13) & Err
   End Select
   Update_Tables = 0
   MyWS.Rollback
   GoTo Exit_Update_Tables

End Function

Sub Write_STAT (Msg As String)
   Me![txt_status].Caption = Msg
End Sub
```

Appendix B

Real World Data Set

Company	Ticker
Acmat Corporation	Acmt
Alfa Corporation	Alfa
Allied Group, Inc.	Algr
Allmerica Property & Casualty	Alpc
American Indemnity Financial C	Aifc
American International Group,	Aig
American Premier Underwriters,	Apz
Argonaut Group, Inc.	Agii
Ari Holdings, Inc.	Ari
Avemco Corporation	Ave
Baldwin & Lyons, Inc.	Bwina
Berkley, W.R. Corporation	Bkly
Berkshire Hathaway Inc.	Brk
Chubb Corporation	Cb
Cigna Corporation	Ci
Cincinnati Financial Corporati	Cinf
Cna Financial Corporation	Cna
Continental Corporation (The)	Cic
Emc Insurance Group Inc.	Emci
Foremost Corporation Of Americ	Fcoa
Fremont General Corporation	Fmt
Geico Corporation	Gec
General Re Corporpation	Grn
Harleysville Group, Inc.	Hgic
Leucadia National Corporation	Luk
Loews Corporation	Ltr
Merchants Group, Inc.	Mgp
Mercury General Corporation	Mrcy
Meridian Insurance Group, Inc.	Migi
Midland Company	Mla
Milwaukee Insurance Group, Inc	Milw
Nac Re Corp.	Nrec
Navigators Group, Inc.	Navg
Nobel Insurance Limited	Nobl
Nymagic, Inc.	Nym
Old Republic International Cor	Ori
Orion Capital Corporation	Oc
Progressive Corporation	Pgr
Re Capital Corporation	Rcc
Reliance Group Holdings, Inc.	Rel
Riverside Group, Inc.	Rsgi
Rli Corp.	Rli
Safeco Corporation	Safc
Seibels Bruce Group, Inc. (The	Sbig
Selective Insurance Group, Inc	Sigi
St. Paul Companies, Inc. (The)	Spc
Sunstates Corporation	Atn
Transamerica Corporation	Ta
Travelers Corporation	Tic
Trenwick Group, Inc.	Tren
Unicare Financial Corp.	Ufn
United Fire & Casualty Company	Ufcs
Usf & G Corporation	Fg
20th Century Industries	Tw

Table B.1: Worldscope : SIC=6331

Company_Name	Ticker
Allmerica Property & Casualty Cos	Apy
General Re	Grn
Itt	Itt
Leucadia National	Luk
Loew S	Ltr
Ohio Casualty	Ocas
Old Republic International	Ori
Progressive	Pgr
Reliance Group Holdings	Rel
Safeco	Safc
St Paul Cos	Spc
Transamerica	Ta
Transatlantic Holdings	Trh
United Services Automobile Association	D.Uzd
Usf&G	Fg

Table B.2: Fortune 1000 : SIC = 6331

A_Company	B_Company_Name
American International Group,	American International Group
American Premier Underwriters,	American Premier Underwriters
Berkshire Hathaway Inc.	Berkshire Hathaway
Chubb Corporation	Chubb
Cigna Corporation	Cigna
Geico Corporation	Geico
Leucadia National Corporation	Leucadia National
Progressive Corporation	Progressive
Reliance Group Holdings, Inc.	Reliance Group Holdings
Safeco Corporation	Safeco
Transamerica Corporation	Transamerica
Allmerica Property & Casualty	Allmerica Property & Casualty Cos
Cincinnati Financial Corporati	Cincinnati Financial
General Re Corporpation	General Re
Old Republic International Cor	Old Republic International
St. Paul Companies, Inc. (The)	St Paul Cos

Table B.3: Output Table for Insurance Companies

Company	Ticker
Amerada Hess Corporation	Ahc
Ashland Oil, Inc.	Ash
Atlantic Richfield Company	Arc
Crown Central Petroleum Corp.	Cnp A
Diamond Shamrock, Inc.	Drm
Du Pont (E.I.) De Nemours And	Dd
Fina, Inc.	Fi
Holly Corporation	Hoc
Hondo Oil & Gas Company	Hog
Lyondell Petrochemical Company	Lyo
Murphy Oil Corporation	Mur
Pacific Resources, Inc.	Na
Petrolite Corporation	Plit
Phillips Petroleum Company	P
Sun Company, Inc.	Sun
Tesoro Petroleum Corporation	Tso
Unocal Corporation	Ucl
Valero Energy Corporation	Vlo
Wainoco Oil Corporation	Wol

Table B.4: Worldscope: SIC=2991

Company_Name	Ticker
Amerada Hess	Ahc
Amoco	An
Ashland Oil	Ash
Cenex	D.Czc
Chevron	Chv
Citgo Petroleum	D.Czh
Coastal	Cgp
Crown Central Petroleum	Cnpa
Diamond Shamrock	Drm
E I Du Pont De Nemours &	Dd
Fina	Fi
Holly	Hoc
Kerr Mcgee	Kmg
Louisiana Land & Exploration	Llx
Lyondell Petrochemical	Lyo
Mapco	Mda
Murphy Oil	Mur
Pennzoil	Pzl
Phillips Petroleum	P
Shell Oil	D.Sgu
Sun	Sun
Tesoro Petroleum	Tso
Tosco	Tos
Total Petroleum North America Ltd	Tpn
Ultramar	Ulr
Unocal	Ucl
Valero Energy	Vlo

Table B.5: Fortune 1000: SIC=2991

A_Company	B_Company_Name
Amerada Hess Corporation	Amerada Hess
Ashland Oil, Inc.	Ashland Oil
Crown Central Petroleum Corp.	Crown Central Petroleum
Diamond Shamrock, Inc.	Diamond Shamrock
Fina, Inc.	Fina
Holly Corporation	Holly
Lyondell Petrochemical Company	Lyondell Petrochemical
Murphy Oil Corporation	Murphy Oil
Phillips Petroleum Company	Phillips Petroleum
Tesoro Petroleum Corporation	Tesoro Petroleum
Unocal Corporation	Unocal
Valero Energy Corporation	Valero Energy
Sun Company, Inc.	Sun

Table B.6: Output Table for Petroleum Refining Companies

Company	Ticker
Advanced Logic Research, Inc.	Aalr
Amplicon, Inc.	Ampi
Apple Computer, Inc.	Aapl
Atari Corporation	Atc
Convex Computer Corporation	Cnx
Decision Industries Corporatio	Na
Dell Computer Corporation	Dell
Digital Communications Associa	Dca
Evans & Sutherland Computer Co	Escc
Everex Systems, Inc.	Evrx
Hewlett-Packard Company	Hwp
Inmac Corporation	Inmc
Intermec Corporation	Intr
Mai Systems Corporation	Mco
Micom Systems, Inc.	Na
Miltope Group Inc.	Milt
Oracle Systems Corporation	Orcl
Paradyne Corporation	Na
Recognition International Inc.	Rec
Stratus Computer, Inc.	Sra
Sun Microsystems, Inc.	Sunw
Tandem Computers Incorporated	Tdm
Telxon Corporation	Tlxn
Ungermann-Bass, Incorporated	Na
Wang Laboratories, Inc.	Wan B

Table B.7: Worldscope: SIC=3571

Company_Name	Ticker
Amdahl	Amh
Apple Computer	Aapl
Ast Research	Asta
Compaq Computer	Cpq
Cray Research	Cyr
Data General	Dgn
Dell Computer	Dell
Digital Equipment	Dec
Gateway 2000	Gate
Hewlett Packard	Hwp
Intergraph	Ingr
International Business Machines	Ibm
Silicon Graphics	Sgi
Sun Microsystems	Sunw
Tandem Computers	Tdm
Unisys	Uis

Table B.8: Fortune 1000: SIC=3571

A_Company	B_Company_Name
Apple Computer, Inc.	Apple Computer
Dell Computer Corporation	Dell Computer
Hewlett-Packard Company	Hewlett Packard
Sun Microsystems, Inc.	Sun Microsystems
Tandem Computers Incorporated	Tandem Computers

Table B.9: Output Table for Electronic Computer Companies

Company	Ticker
Alc Communications Corporation	Alc
Bell Atlantic Corporation	Bel
Bellsouth Corporation	Bls
C-Tec Corp.	Ctex
Centel Corporation	Cnt
Century Telephone Enterprises,	Ctl
Contel Corporation	Ctc
Gte Corporation	Gte
International Telecharge, Inc.	Iti
Lincoln Telecommunications Co.	Ltec
Mci Communications Corporation	Mcic
Nynex Corporation	Nyn
Pacific Telecom, Inc.	Ptcm
Pacific Telesis Group	Pac
Rochester Telephone Corporatio	Rtc
Southern New England Telecommu	Sng
Southwestern Bell Corporation	Sbc
Sprint Corporation	Fon
Telephone And Data Systems, In	Tds
U S West, Incorporated	Usw

Table B.10: Worldscope: SIC=4813

Company_Name	Ticker
Alltel	At
American Telephone & Telegraph	T
Ameritech	Ait
Bell Atlantic	Bel
Bellsouth	Bls
Gte	Gte
Mci Communications	Mcic
Nynex	Nyn
Pacific Telesis Group	Pac
Southwestern Bell	Sbc
Sprint	Fon
U S West	Usw
Williams Companies	Wmb

Table B.11: Fortune 1000: SIC=4813

A_Company	B_Company_Name
Pacific Telesis Group	Pacific Telesis Group
Bell Atlantic Corporation	Bell Atlantic
Bellsouth Corporation	Bellsouth
Gte Corporation	Gte
Mci Communications Corporation	Mci Communications
Nynex Corporation	Nynex
Southwestern Bell Corporation	Southwestern Bell
Sprint Corporation	Sprint
U S West, Incorporated	U S West

Table B.12: Output Table for Telephone Communications Companies

Bibliography

[1] Almudena Arcelus. Name management strategy for database integration. Master's thesis, Massachusetts Institute of Technology, 1990.

[2] James M. Kennedy Howard B. Newcomb. Record linkage. *Communications of the ACM*, September 1962.

[3] Herber Regal Norckauer Jr. Duplicate entry detection in mailing and participation lists. Master's thesis, Massachusetts Institute of Technology, 1990.

[4] William E. Winkler. Exact matching lists of businesses: Blocking, subfield identification, and information theory. *Record Linkage Techniques, Proceedings of the Workshop on Exact Matching*, 1985.