Use of Smart Tokens in Cleaning Integrated Warehouse Data

Christie I. Ezeife, University of Windsor, Canada
Timothy E. Ohanekwu, University of Windsor, Canada

ABSTRACT

Identifying integrated records that represent the same real-world object in numerous ways is just one form of data disparity (dirt) to be resolved in a data warehouse. Data cleaning is a complex process, which uses multidisciplinary techniques to resolve conflicts in data drawn from different data sources. There is a need for initial cleaning at the time a data warehouse is built, and incremental cleaning whenever new records are brought into the data warehouse during refreshing. Existing work on data cleaning have used pre-specified record match thresholds and multiple scanning of records to determine matching records in integrated data. Little attention has been paid to incremental matching of records. Determining optimal record match score threshold in a domain is hard. Also, direct long record string comparison is highly inefficient and intolerant to typing errors. Thus, this article proposes two algorithms, the first of which uses smart tokens defined from integrated records to match and identify duplicate records during initial warehouse cleaning. The second algorithm uses these tokens for fast, incremental cleaning during warehouse refreshing. Every attribute value forms either a special token like birth date or an ordinary token, which can be alphabetic, numeric, or alphanumeric. Rules are applied for forming tokens belonging to each of these four classes. These tokens are sorted and used for record match. The tokens also form very good warehouse identifiers for future faster incremental warehouse cleaning. This approach eliminates the need for match threshold and multiple passes at data. Experiments show that using tokens for record comparison produces a far better result than using the entire or greater part of a record.

Keywords: data dirt; data disparity; data warehouse; token-based data cleaning

INTRODUCTION

Several independent data sources are integrated into a huge data repository called a data warehouse for complex analytical and querying purposes (Calvanese, Giacomo, Lenzerini, Nardi, & Rosati, 2000; Inmon, 1996). A data warehouse, therefore, is a database made up of integrated, subject-oriented, time-variant, and nonvolatile data designed for use in decision support querying (Inmon, 1996). Traditional databases are different from data warehouses because traditional databases: (1) are designed around enterprise functions (e.g., savings account database for maintaining

current savings account balance of each customer), and not organized around attribute subjects (like customers, account type, transaction type); (2) store only current transaction data (e.g., has current account balance for each customer, but does not remember the account balance before last deposit) because it is not historical and does not remember account balance an hour or a week ago; (3) undergo frequent updates (e.g., new deposit into savings account will change the current balance), meaning that their data are volatile; and (4) are usually stand-alone repository (e.g., has only savings account balance and not checking or other accounts), implying that they are not integrated.

Data sources integrated into the warehouse may be deployed on different hardware and software platforms (Ezeife, 2001), as well as different data models. For example, a financial institution in a given city may have separate databases for its different units. Its checking accounts unit may have a data source maintained as an XML file (as shown in Figure 1) deployed on an IBM PC, while its savings accounts department may have a relational database (shown as Figure 2) mounted on an Apple Macintosh computer. Yet, the credit card unit may store its client data in a flat file deployed on a UNIX-controlled mini computer. Figures 1 and 2 are examples of function-oriented data sources, which are good for answering simple and one-dimensional queries such as:

What is the balance of each customer in their savings account now?

How many customers now have balance greater than $1,500.00?

In order to answer queries like "Get account balance of all customers in both checking and savings account" or "Get the total balance maintained by each customer in all accounts every week for one year," it is necessary to visit both databases for the first query. For the second query, it is necessary to have stored a history of account balances as they change over time and better, in an integrated data source. To gain competitive advantage, the financial institution in our example may want to pose queries for finding information like: (1) when in a given year customers deposit huge amounts of money in their accounts, (2) the pattern/trend in the customers' needs/behaviors, (3) season-by-season analysis of transactions carried out in the bank, and (4) when in a given year the credit card unit makes its greatest profit. A data warehouse—being historical, integrated, subject-oriented, nonvolatile, and summarized data—provides the necessary platform for answering such business and management decision support queries.

The data taken from two or more sources are "dirty" in nature due to heterogeneity in representations. Therefore, data heading to the data warehouse needs to be cleaned for the warehouse to be reliable. This article presents two efficient token-based data cleaning algorithms for initial data warehouse cleaning and incremental data warehouse cleaning during warehouse refreshing.

DATA CLEANING OVERVIEW

Data cleaning is a technique for detecting missing and incorrect values and correcting them, as well as matching duplicate records (Simoudis, Livezey, & Kerber, 1995) in an integrated data warehouse or database table. It focuses on eliminating variations in data contents and reducing data

Figure 1. Instance of an XML data source

```
< CheckingAccounts >
    < Client >
      < cid > 1001 < /cid >
      < csex > M < /csex >
      < cbirth > June 4, 1975 < /cbirth >
      < cphone > 519 - 256 - 6416 < /cphone >
      < cname > Timothy Ohanekwu < /cname >
      < caddress > Randolph ave, 995N9B2T7 < /caddress >
      < coccupation > ITProfessional < /coccupation >
      < balance > 1000.54 < /balance >
    < /Client >
< /CheckingAccounts >
```

redundancy, and is aimed at improving the overall data quality and consistency (Delvin, 1997). Data cleaning determines first whether two or more records represented differently are referring to the same real-world entity, and secondly it performs any one (or combination) of the following actions (if the records represent the same object): (1) combining them to get a consolidated complete record, (2) unifying them with a single entity identifier, and (3) retaining only one copy of them. Data cleaning involves decomposing and re-assembling data (Kimball, 1996), and sometimes, "semantic enrichment," e.g., acquiring additional information from external source(s) to resolve conflicts (Parent & Spaccapietra, 1998).

Two main causes of "Dirt" or conflicts in data are synonyms and homonyms, though there are many others such as: "incomplete, missing, or null values"; "spelling, phonetic, or typing errors"; "mis-fielding" (e.g., a country's name in a state/province field); "noise or contradicting entry," such as values outside the accepted range (e.g., 31/9/99); "scanning errors" (e.g., alphabetic "I" instead of numeric "1" and vice versa); "type mismatch"; and so on. While all other causes of data dirt may be as a result of "oversight" or "human errors," synonyms and homonyms are not.

Figure 2. Instance of a relational data source

cno	S002
cname	T. Emenike Ohane
cbirth	04-06-75
caddress	#995 Roundup, Windsor, n9b 2t7
cphone	2566416
csex	M
occupation	IT Specialist
cbalance	500.50

For example, a document collection center in a unit of an organization may use entity acronyms/abbreviations (e.g., "ACM"), while another center may write it in full (e.g., "Association of Computing Machinery"). There may also be a situation where two different scales are used to express the same level of performance, for example, "A+" versus "99%." Homonymous dirt arises when the same "term" or "expression" refers to more than one entity, for example, many occurrences of "John Smith" in a data source may refer to different persons.

The main differences between data cleaning in a data warehouse and data cleaning in a single file is that for database systems, which integrate multiple data sources (like data warehouses, federated databases, and global Web-based information systems), data quality work to be done includes: all the single source data cleaning problems, as well as increased synonym (different names for same entity) and homonym (same name for different entities) problems. Also, while some duplicates need to be eliminated, others need to be retained after proper object identifier merging. Generally, no identified duplicates are eliminated from the warehouse fact table, but warehouse identification unification is required for all duplicates in the fact table. Integrated sources also have increased redundancy in data.

Omitting the data cleaning process in a data warehouse system will result in spurious results. For example, a warehouse built from the two data sources in Figures 1 and 2, but without the cleaning process, is unable to correctly answer queries like: "Get the total balance maintained in all accounts (checking and savings) by each customer." This is because in the checking database this customer is represented as Timothy Ohanekwu, but in the savings database he is represented as T. Emenike Ohane. There are other discrepancies in other fields as well. Thus, the two records for the same person (Timothy Ohanekwu) from the two data sources are treated as though they belong to different entities without data cleaning. This simple example demonstrates the importance of cleaning integrated data.

RELATED WORK

Bitton and Dewitt (1983) propose sorting records on some designated fields to bring potentially identical records together in a large data file. However, the drawback of this work is that the fields upon which sorting is based are "dirty," and thus may fail to bring matching records together. Secondly, the time complexity for the record comparison phase is quadratic in nature, requiring $N^2 - N*1/2$ comparisons (where N is the number of records in the data set). The sorted neighborhood method (SNM) in Hernandez and Stolfo (1995, 1998) solves the merge/purge problem in a large database by: (1) forming token keys from some selected fields of the database table, (2) sorting the entire data set on these keys, (3) clustering the sorted records to have records with same token keys in the same clusters, and (4) using a sliding window of a fixed size to compare records in each cluster and merge records if they are equivalent. Thus, given an integrated list of N records as demonstrated with four records shown in Figure 3.

The keys are formed from these records by concatenating the first three consonants of the record's last name with the first three letters of the first name, followed by the address number and all of the consonants of the street name, followed by the first three digits of the SSN. This will re-

Figure 3.

RecNum	FirstName	LastName	Address	SSN
1	Pat	Stalpe	415 Busy Street	123456789
2	Pat	Stiles	415 Bus Street	123458689
3	Pat	Stalfe	415 Busy Street	123456789
4	Pat	Stally	415 Busy Street	123456789

Figure 4.

RecNum	Tokenkey
1	STLPAT415BS123
2	STLPAT415BS123
3	STLPAT415BS123
4	STLPAT415BS123

sult in the following token keys for the four records (see Figure 4).

This means that each record has an extra field (Tokenkey). The records are sorted with these keys so that likely duplicate records are brought together. Comparison of records for equivalence is a complex inferential process, which takes into consideration more information from the records than the keys used to bring them together. The equational theory is used to define the logic (semantic) of domain equivalence with rules to determine if records close together are the same. The multi-pass version of this algorithm changes the sort key header field for each independent run through the dataset. For example, one run may use the student's contact address as the head of the key, and the next run may be based on key with the original record SSN as the head. The notion of sorting records on token keys is an attractive contribution by this work, but comparison based on the original records reduces its optimality since the records are dirty. The equational theory used in the multi-pass version of the work is also a time-consuming process.

The basic field matching algorithm (Monge & Elkan, 1996) first extracts and sorts atomic strings (words) of each field of each record, and second finds the number of strings from each of the two records that match by computing the matching score for pairs of strings. The string match scores are later combined to obtain the field match scores. The record match score is in turn computed from the field match scores, and the value of the record match score in comparison to a pre-selected threshold is used to decide if the two records being compared are the same. For example, after removing stop words like "of", the process of checking if two strings, A and B, are duplicates proceeds as discussed in Figure 5.

There are six strings, k {Comput, Sci, San, Diego, Univ, Calif} in A that match some strings in B. The overall match score is computed with the following formula:

$$(k/(|A| + |B|))/2 = (6(8 + 7))/2 = 6 * 2/15 = 0.8$$

If the pre-defined match threshold is 0.75, then A and B are duplicates since their match score of 0.8 is greater than the threshold.

Figure 5.

A = "Comput Sci. Eng. Dept. University of California, San Diego"
B = "Department Computer Science, Univ. Calif., San Diego"
After sorting words, the strings A and B become:
A = "California Comput Dept Diego Eng San Sci Univeristy"
B = "Calif Computer Department Diego San Science Univ"

The work described in Lee, Hongjun, Tok, and Yee (1999) is an enhancement on the SNM by Hernandez and Stolfo (1998), which introduces the idea of field pre-processing prior to sorting, tokenization, and comparison phases. Pre-processing the fields with an external data source, such as birth registry, before other data cleaning phases solves a lot of cleaning problems, but using an external data source to achieve that is simply infeasible because: (1) the external source may not be accessible; and (2) if it is accessible, it may most likely be through a network, which may take a lot of time. A brief version of one of the proposed token-based warehouse cleaning algorithms for initial data warehouse cleaning is presented in Ohanekwu and Ezeife (2003).

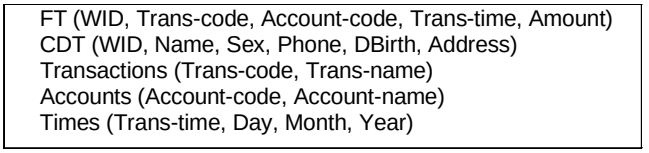
CONTRIBUTIONS

Existing techniques that have used tokens for bringing likely duplicate records together were presented by Hernandez and Stolfo (1995, 1998) and Lee et al. (1999). Work that used pre-determined match score thresholds to decide on a match between two input strings includes that by Lee et al. (1999) and Monge and Elkan (1996). Work that depends on external or interactive input during duplicate detection includes that by Galharda, Florescu, Shasha, Simon, and Saita (2001), Lee et al. (1999) and Raman and Hellerstein (2001). Achieving a high recall (cleaning accuracy) in a reasonable time, which is less dependent on match score thresholds and external intervention, are data cleaning research goals this article contributes to. Thus, this article first proposes a method for defining smart tokens composed from most important fields of records, which are effectively used for identifying duplicate records in data warehouses and other records. The smart token-based technique achieves a better result than the record-based techniques of comparable algorithms. By using short tokens for record comparisons, a high recall/precision is achieved. The technique also drastically lowers the dependency of data cleaning on match score "threshold" choice. This article also proposes a second algorithm for using already defined smart tokens to perform incremental cleaning of subsequent integrated data warehouse records during refreshing.

OUTLINE OF THE ARTICLE

The rest of this article is organized as follows. An example of a dirty data warehouse with its data sources is given next.

Figure 6.



We then formally describe the token-based data cleaning algorithms, followed by a presentation of a number of experimental cases for performance comparisons. We end with conclusions and future work.

AN EXAMPLE

Before presentation of the dirt to be cleaned, an example data warehouse schema is presented in the two types of tables in the data warehouse—integrated fact table and dimension tables.

Data Warehouse Schema

The example given is that of a data warehouse built from two data sources, the savings account (SA), and the checking account (CA) of a bank. The warehouse fact table keeps track of the amount of money involved in different kinds of transactions (not account balance) executed by bank customers over a period of time. The data warehouse fact table (FT) shown in Table 3 and the customer dimension table (CDT) given as Table 4 are yet to be cleaned. Tables 1 (TA transactions) and 2 (CA transactions) are used to record transactions executed on the two bank accounts, SA and CA, by customers for a short period of time. The "dirt" to be cleaned in Table 4 (dirty CDT) and Table 3 (dirty FT) are described in the first part of this section, while the tasks accomplished by the proposed algorithm are outlined in the second part. The data warehouse schema, which represents an integration of the savings account and checking account data sources, is shown in Figure 6.

This data warehouse consists of the main fact table, Table 3 (dirty FT), and four dimension tables, only one of which (Table 4, dirty CDT) will be cleaned in the example to demonstrate the proposed technique. Generally, smart tokens are first created on the main entity dimension table, and the warehouse identifier generated from these tokens is applied to both this table and the fact table for duplicate record handling. The rest of the dimension tables have limited single-source data quality problems that can be handled using token keys from the fields or any other approach.

Dirt in the Customer Dimension and Fact Tables

Two levels of dirt exist in Table 4, namely, field- or attribute-level dirt and record-level dirt. The field-level dirt occurs in each field in a record. For example, the "WID" field of Table 4 has "type-mismatch" dirt, since different data types are used to represent the same customer. There are also format differences in both "phone" and "birth" fields. For example, the phone number "2566416" in row 2 is

written as "5192566416" in row 9, while the date of birth of the customer in row 2 written as "01-Jan-1975" has a different format (1-1-75) in row 9. Other field-level dirt apparent in Table 4 include: (1) typographic errors, and (2) different addressing conventions (in address field). The implication of field-level dirt is that no particular field is *clean enough* to determine record match. Record-level dirt is the combination of all the fields' dirt in a given row. For example, row 1 of Table 4 is a record with the following content (excluding the Row and WID fields), "*John Smith O, M, (519) 111-1234, 25-Dec-70, Sunset # 995 N9B3P4.*"

This appears to be the same person as row 6, with the following content (excluding the Row and WID fields), "S. John, M, 1111234, 25-12-1970, 995 Sunset Ave, N9B 3P4." An obvious implication of record-level dirt is "that duplicates are not easily determined." The fact table (Table 3, dirty FT) contains only field-level dirt in the "WID" field as it is, using different WID to represent the same customer, and that makes it difficult to determine all the transactions conducted by the same customers.

The Cleaning Tasks

Two cleaning tasks to be carried out on the customer dimension table (Table 4, dirty CDT) are: (1) duplicate detection, and (2) duplicate elimination. Duplicate detection requires a combination of (pieces of) information from two or more fields to find if two or more records are the same. Duplicate elimination task ensures that only one copy of records found to be duplicates is retained.

The only way to establish a link between the fact table (Table 3, dirty FT) and

Table 1. Transaction history for SA customers (SA transactions)

Cid	Transaction	Date-And-Time	Amount
S005	Deposit	1/1/02, 9.30AM	525.25
S001	Deposit	11/1/02, 4.30PM	1005.53
S005	Withdraw	15/3/02, 1.45PM	125.44
S004	Withdraw	6/4/02, 11.00AM	325.50

Table 2. Transaction history for CA customers (CA transactions)

Cid	Transaction	Date-And-Time	Amount
1001	Deposit	2/1/02, 12.00PM	650.33
1004	Withdraw	5/3/02, 5.00PM	150
1005	Deposit	8/4/02, 9.45AM	1015.99
1002	Withdraw	15/2/02, 10.00AM	250.16
1003	Deposit	8/4/02, 7.00PM	450.50

Table 3. The yet-to-be-cleaned fact table (dirty FT)

Row	WID	Trans-type	Account	Trans-time	Amount
1	S005	D	SA	570A	525.25
2	1005	D	CA	585A	1015.99
3	1004	W	CA	1020P	150
4	S005	W	SA	825P	125.44
5	1001	D	CA	720P	650.33
6	S004	W	SA	660A	325.50
7	1002	W	CA	600A	250.16
8	S001	D	SA	990P	1005.53
9	1003	D	CA	1140P	450.50

Table 4. The yet-to-be-cleaned customer dimension table (dirty CDT)

Row	WID	Name	Sex	Phone	Birth	Address
1	S001	John Smith O	M	(519) 111-1234	25-Dec-70	Sunset # 995 N9B3P4
2	S002	Tim E. Ohanekwu	M	2566416	01-Jan-75	ABCD St. No. 695 n9b 2t7
3	S003	Colette Jones	M	123-4567	08/Aug/64	600 XYZ apt 5a5 N7C4K4
4	S004	Ambrose A. Diana	F	519 6669999	Nov/11/72	4 Church Rd. Rd. N8K6t6
5	S005	Smith John	F	519 560 3626	30-Oct-78	182 Haven Ave M9B3J7
6	1001	S. John	M	1111234	25-12-1970	995 Sunset Ave, n9b 3p4
7	1002	Jon Cole	M	Null	08-08-1964	XYZ No. 600 apt 585 n7c 4k4
8	1003	Ambo D. Dian	F	566-5555	10-11-1972	Church St. # 4 n8k 6t6
9	1004	Ohanekw T.E	M	519 2566416	1-1-75	# 695 abcd street N9B2T7
10	1005	Edema Tom Obi	M	977-5950	23-May-1967	98 Haven Rd. M8C 8S4

the customer dimension table (Table 4, dirty CDT) is to unify the entity identifier, such that it is possible to determine the transactions conducted by a given entity. This is not yet the case, because different identities (e.g., S001 from the savings account, and 1001 from the checking account) are used to represent the same entity (John Smith O and S. John). The effect of this is that it is impossible to obtain the correct total amount deposited in all accounts by "John Smith O." This is also the case with

Table 5. The target fact table after cleaning (clean FT)

Row	WID	Trans-code	Account-code	Trans-time	Amount
1	103078JS	D	SA	570AM	525.25
2	32367EOT	D	CA	585AM	1015.99
3	010175EOT	W	CA	1020PM	150
4	103078JS	W	SA	825PM	125.44
5	122570JOS	D	CA	720PM	650.33
6	111172ADD	W	SA	660AM	325.50
7	080864CJ	W	CA	600AM	250.16
8	122570JOS	D	SA	990PM	1005.53
9	111172ADD	D	CA	1140PM	450.50

"Tim E. Ohanekwu," "Ambrose A. Diana," and so forth. The solution to this problem is to use the same identifier value for the same real-world entity. For example, "122570JOS" will be used for all occurrences of "S001" and "1001" in the fact table to reflect the fact that "John Smith O" and "S. John" represent the same person. The same is done for records "S002" and "1004," and "S004" and "1003."

The next section discusses the process of producing the desired fact table (Table 5, clean FT) and customer dimension table (Table 6, clean CDT) after cleaning Tables 3 (dirty FT) and 4 (dirty CDT) respectively, using the proposed TB cleaner algorithm.

PROPOSED TOKEN-BASED DATA CLEANING ALGORITHMS

Two data cleaning algorithms are presented in this section. The first algorithm, suitable for cleaning data when a data warehouse is being built, is presented in the first subsection, while the second algorithm designed for subsequent/incremental cleaning of an existing data warehouse is described in the second subsection.

Initial Warehouse TB Cleaner Algorithm

The proposed warehouse token-based (initial TB cleaner) data cleaning algorithm accepts "dirty" source tables, such as Table 1 (recent SA transactions), Table 2 (recent CA transactions), Table 3 (dirty FT), and Table 4 (dirty CDT) and returns "cleaned" data warehouse tables, such as Tables 5 (clean FT) and 6 (clean CDT). Basically, a user selects two or three most important fields and ranks them based on their power to uniquely identify records. The elements in the selected fields are tokenized, resulting in a table of tokens (shown as Table 7). The two most uniquely identifying fields of the table are used as two different main sort keys on the table of "tokens" to produce two sorted token tables, which are shown as Tables 8 (to-

kens sorted on birth day—BD tokens) and 9 (tokens sorted on name—NM tokens). Token records in close neighborhoods are compared for a match, and warehouse ID (WID) is generated for each record. The four main steps (in sequence) in the algorithm are described in detail below, while the formal TB cleaner algorithm is presented as Figure 7 (TB-cleaner algorithm).

Step 1: Selection and Ranking of Fields

The user selects and ranks two or three fields that could be combined to most uniquely identify records. The condition for "fields selection and ranking" is that the user is very familiar with the problem domain, and can select and rank fields according to their unique identifying power. We assume that the user in our banking domain selected the following fields from Table 4 (dirty CDT)—"Birth," "Name," and "Address"—and ranked them in the order given.

Step 2: Extraction and Formation of Tokens

Essentially, in this step, a given attribute value is transformed into a smart token. A smart token is obtained by:

1. Decomposing an attribute value into special tokens (like date and acronyms) and ordinary tokens consisting of ordinary words, numbers, alphanumeric terms, punctuations, articles, salutations, and special characters.
2. Eliminating all unimportant tokens consisting of punctuations, special characters (like '/', '('), articles (like 'a', 'the'), salutations (like 'Dr', 'Mr'), and labels (like 'street', 'apt', 'blvd').
3. Further decomposing any special tokens to primitive terms (e.g., month day year)

Table 6. A target customer dimension table after cleaning (clean CDT)

Row	WID	Name	Sex	Phone	Birth	Address
1	12257 0JOS	John Smith O	M	(519) 111-1234	25-Dec-70	Sunset #995 N9B3P4
2	01017 5EOT	Tim E. Ohanekwu	M	2566416	01-Jan-75	ABCD St. No. 695 n9b 2t7
3	08086 4CJ	Colette Jones	M	123-4567	08/Aug/64	600 XYZ apt 5a5 N7C4K4
4	11117 2ADD	Ambrose A. Diana	F	519 6669999	Nov/11/72	4 Church Rd. N8K6t6
5	10307 8JS	Smith John	F	519 560 3626	30-Oct-78	182 Haven Ave M9B3J7
6	05236 7EOT	Edema Tom Obi	M	977-5950	23-Mar-1967	98 Haven Rd. M8C 8S4

Table 7. The table of tokens

Row	WID	Name	Birth	Address
1	S001	JOS	122570	934995NBP
2	S002	EOT	010175	927695NBT
3	S003	CJ	080864	74455600AKNC
4	S004	ADD	111172	4866NKT
5	S005	JS	103078	937182JMB
6	1001	JS	122570	934995NBP
7	1002	CJ	080864	744585600KNC
8	1003	ADD	101172	4866NKT
9	1004	EOT	010175	927695NBT
10	1005	EOT	052367	88498MCS

Table 8. Tokens sorted in ascending order of birth tokens (BD tokens)

Row	WID	Name	Birth	Address
2	S002	EOT	010175	927695NBT
9	1004	EOT	010175	927695NBT
3	S003	CJ	080864	74455600AKNC
7	1002	CJ	080864	744585600KNC
10	1005	EOT	052367	88498MCS
8	1003	ADD	101172	4866NKT
5	S005	JS	103078	937182JMB
4	S004	ADD	111172	4866NKT
1	S001	JOS	122570	934995NBP
6	1001	JS	122570	934995NBP

Table 9. Tokens sorted in ascending order of name tokens (NM tokens)

Row	WID	Name	Birth	Address
4	S004	ADD	111172	4866NKT
8	1003	ADD	101172	4866NKT
3	S003	CJ	080864	74455600AKNC
7	1002	CJ	080864	744585600KNC
2	S002	EOT	010175	927695NBT
9	1004	EOT	010175	927695NBT
10	1005	EOT	052367	88498MCS
1	S001	JOS	122570	934995NBP
5	S005	JS	103078	937182JMB
6	1001	JS	122570	934995NBP

and applying any necessary type conversions (e.g., January as 01) and vice versa.

4. Defining smart tokens from any of the three possible types of numeric, alphabetic, and alphanumeric tokens in the field (e.g., given the alphanumeric address "600 XYZ blvd apt 585 N7C4K4," rules for alphanumeric tokens will be applied to this field to create its smart token 585600744NCK). The three types of tokens in the field are formed as:

   a. **Numeric Tokens:** Each string of number tokens representing a real-life term (like phone number, social security number, street number, apartment number, etc.) is kept together as one numeric token in the order they appear originally, after removing unimportant characters. For example, tokens for dates 25-Dec-70 and 25/12/1970 are the same as 122570, obtained after converting month to numeric, and removing unimportant tokens and terms. With dates, the century part "19" if present in any date

is eliminated as an unimportant term. With phone numbers, international code and area codes are removed as unimportant terms.

b. **Alphabetic Tokens:** Each important alphabetic string of word tokens in the field representing a real-life term (like name, book title, etc.) is used to define an alphabetic token by selecting the first letter in each word in the field, sorting these letters in a specific order, and stringing them together to obtain the alphabetic token (e.g., the token from the name Ohanekwu Tim Emenike is EOT).
c. **Alphanumeric Tokens:** These are obtained from fields that are alphanumeric, such as address (e.g., 600 XYZ blvd apt 585 N7C4K4), by using only the numeric and alphanumeric tokens from the field (600 585 N7C4K4). Any more alphanumeric token is further decomposed into its numeric and alphabetic components (e.g., N7C4K4 is decomposed into 744 and NCK). Finally, the tokens in the field are sorted in order to obtain, for example, 585600744NCK. Applying the token extraction procedure on the "name," "birth," and "address" fields of Table 4 (dirty CDT) produces Table 7 (table of tokens).

Figure 7. Token-based data cleaning algorithm for data warehouse (TB-cleaner)

Algorithm 3.1 (Token-based Data Cleaning Algorithm for Data Warehouse)
Algorithm Token-Based()
Input: two or more tables containing "dirty" data
Output: Dirt-free Data Warehouse Tables
begin
(1) Selection and Ranking of Fields
Given a table, T with i number of attributes, A[i], select j
number of attributes, A[j] such that A[j]⊆A[i] and i⩾2 where A[j] have highest discriminatory power of all A[i]
(2) Extraction and Formation of Tokens
For row = 1 to the last row, n
For attribute = 1 to the last attribute, j
Tokens[row, attribute] = extract-tokens (T[row], A[attribute])
(3) Sorting tokens on 2 fields to get 2 token tables
Sort Tokens separately on the two most discriminating attributes in A[j] to give sorted-Tokens1 and sorted-$Tokens_2$
(4) Duplicate Detection, Elimination and Generating of WID
Using a sliding neighborhood window size, w, detect record duplicates in sorted-Tokens1 and sorted-$Tokens_2$, integrate detection results, eliminate duplicates, generate WID for the entities and apply WID appropriately
end // of TB-Cleaner//

Step 3: Sorting of Tokens

The table of tokens (Table 7) is sorted separately on the two most uniquely identifying fields according to ranking by the user. Therefore, sorting respectively on the "birth" and "name" token fields produces Tables 8 (record tokens sorted on birth tokens—BD tokens) and 9 (record tokens sorted on name tokens—NM tokens) respectively. Records "1003" and "S004" highlighted in Table 8 (BD tokens) are not in the immediate neighborhood of each other due to a "number difference" in the birth token. The same is the case with records "S001" and "1001" in Table 9 (NM tokens) due to "token inequality." It can also be said that records "1003" and "S004," which are spread apart in Table 8 (BD tokens), are close neighbors in Table 9 (NM tokens). Conversely, records "S001" and "1001," which are not close neighbors in Table 9 (NM tokens), are brought to close neighborhood in Table 8 (BD tokens). The purpose of sorting on two tokens is to catch possible duplicate records that are not brought together by one field token sort-key, and by so doing, the algorithm avoids the need to engage in numerous multi-passes at huge tables. The duplicate detection results from both token tables are eventually combined to give the final optimal result as explained later.

Step 4: Duplicate Detection, Elimination, and Generation of Warehouse Identification

The main cleaning tasks are accomplished in this step. The three sub-steps involved in cleaning are: detection of record duplicates, elimination of duplicates, and generation and applying of warehouse identification (WID).

Detection of Duplicates

The "token-based" record match is based on the following valid argument:

If tokens are sufficiently adequate to bring potential duplicate records together, then they can equally be used to determine record match.

The above argument is formalized as proposition 1:

Proposition 1: Two or more records from different sources within the same application domain would most likely have the same or nearly the same tokens if such tokens were extracted from the most uniquely identifying attributes of the records.

The following sequence is followed when two records are being matched. Given two records, R_1 and R_2, having m pairs of token fields, R_1t_1, R_1t_2,…, R_1t_m; R_2t_1,R_2t_2,…,R_2t_m. First, the similarity match count (SMC) is determined. SMC is the number n of corresponding token fields that match, divided by the total number m of token fields and ranges from 0.00 to 1.00. The value of SMC of a match is used to determine whether records R_1 and R_2 are: (1) a perfect match (if SMC is 1.0), (2) a near perfect match (if SMC is between 0.67 and 0.99), (3) maybe a match (if SMC is between 0.33 and 0.66), and (4) no match at all (if SMC is less than 0.33). When the SMC results in a "maybe match," a function further computes the "similarity match ratio" (SMR) of each of the pairs of tokens that did not match exactly. SMR is a character-level comparison that is used to determine whether the field token pair match or not. Given two tokens t_1 and t_2,

with m and n characters respectively, and also given that the number of characters common in t_1 and t_2 is c, SMR is defined as $\frac{2c}{n+m}$. Tokens t_1 and t_2 are considered a match if and only if SMR ≥ 0.67. Once the SMRs of the tokens are used to determine the number of tokens that match, the SMC of the records is now computed in order to declare the records a match or not. The outcome of applying the above duplicate detection procedures to tokens sorted on birth attribute in Table 8 (BD tokens) are the following pairs of duplicate records: (S001,1001), (S002,1004), (S003,1002). Similarly, applying the same procedures to the second token table, Table 9 (NM tokens), which is sorted on the name attribute, results in the following duplicates being identified: (S002, 1004), (S003, 1002), (S004, 1003). The use of two token sort keys serves to identify all possible duplicates. It can be seen that each of the token tables missed one duplicate that is identified by the other. Finally the duplicate results identified by each of the token tables are integrated to obtain the list of record duplicates as: (S001, 1001), (S002, 1004), (S003, 1002), and (S004, 1003).

Elimination of Duplicates and Generation of WID

The WID is formed from the first record in the duplicate set of Table 10 (duplicate record list table) by concatenating the two most important tokens used in the sorting of the table of tokens shown in this table. Only the first record in the duplicate set is retained in the customer dimension table, while the rest are deleted. The old WID of the corresponding record is overwritten with the newly generated WID. The old WID of the records in the fact table corresponding to the record(s) in the duplicate set are overwritten with the new WID. The final result is a duplicate-free and cleaned customer dimension table (Table 6, clean CDT) and an entity-unified fact table (Table 5, clean FT). In addition, a log table (Table 11) is generated and stored for subsequent cleaning tasks.

Second Proposed Algorithm for Cleaning an Existing Data Warehouse: Refresh TB Cleaner

This section presents a token-based algorithm (refresh TB cleaner) for cleaning an existing data warehouse. Assuming

Table 10. Duplicate record lists for WID generation (duplicate list)

Duplicate Set	First Record	Token1 (B.day)	Token2 (Name)
{1, 6}	1 (S001)	JOS	122570
{2, 9}	2 (S002)	EOT	010175
{3, 7}	3 (S003)	CJ	080864
{4, 8}	4 (S004)	ADD	111172
{5}	5 (S005)	JS	103078
{10}	10 (1005)	EOT	052367

Table 11. The log table for subsequent data warehouse cleaning (LOG table)

Row	OLD-WID	Name-Token	Birth-Token	Address-Token	NEW-WID
1	S001	JOS	122570	934995NBP	122570JOS
2	S002	EOT	1175	927695NBT	1175EOT
3	S003	CJ	8864	74455600AKNC	8864CJ
4	S004	ADD	111172	4866NKT	111172ADD
5	S005	JS	103078	937182JMB	103078JS
6	1001	JS	122570	934995NBP	122570JOS
7	1002	CJ	8864	744585600KNC	8864CJ
8	1003	ADD	101172	4866NKT	111172ADD
9	1004	EOT	1175	927695NBT	1175EOT
10	1005	EOT	52367	88498MCS	52367EOT

(as an example) that data to either refresh or expand the warehouse with are fetched from Tables 12 (refresh FT) and 13 (refresh CDT). Table 12 contains transactions carried out by persons in Table 13 (refresh CDT). This algorithm proceeds step-wise as follows.

Step 1: Token Composition

This algorithm (refresh TB cleaner), like the initial version (initial TB cleaner), starts token formation with a main dimension table, which in this case is Table 13 (refresh CDT). The first step is to compose tokens from the two most uniquely identifying fields, namely, "birth" and "name" for a given record in Table 13 in a similar manner as described earlier. For example, the tokens t_1 and t_2 formed from those two fields for the first record in Table 13 (refresh CDT) are "122570" and "JOS."

Step 2: Log Duplicate Cluster Formation

A cluster is formed from the log table (Table 11) using t_1 and t_2 to determine whether each of the new records in Table 13 (refresh CDT) is: (1) new to the warehouse (when cluster is empty), (2) has only one existing record match in the log table (when cluster has one record), or (3) has more than one existing record match in the log table (when cluster has more than one record). The cluster is defined as follows:

Select into cluster, c all records from Log Table 11 where Birth-Token = t_1 and Name-Token = t_2;

Step 3: Examine Cluster, Define WID and Refresh

Examine the cluster c returned by the query in step 2. The refresh action to take depends on the contents of c as follows. When c is empty, it indicates a new customer and thus, the two tokens t_1 and t_2 for the new transaction are simply concatenated and used as this customer's warehouse identifier, and applied to the dimension, fact, and log tables. When c has one existing record in the log table, the WID of this record is used as the WID of the new record for refreshing. When c has more than one existing record in the log table,

similarity match count of the new record and each record in the cluster is used to determine which record in the cluster the new record matches before using the matching record's WID to refresh the warehouse tables. The operation performed in this step is expressed formally as:

If c is empty, then generate WID from t_1 and t_2 for a new customer and refresh;

Else if c has only one element, then use WID of this existing customer to refresh new transaction;

Else if c contains more than one element, then perform additional task to determine the actual record and then perform action for an existing customer;

Applying this algorithm on Tables 12 (refresh transactions) and 13 (refresh CDT) discovers that customers "S001," "1001," and "CC001" already exist in the warehouse, while "1006" and "CC002" are new entrants into the warehouse. Our incremental warehouse cleaning algorithm records 100% subsequent cleaning efficiency.

Table 12. Transactions to refresh or expand warehouse with (refresh FT)

CID	Source	Activity	Time	Date	Amount
S001	SA	W	779A	1/4/02	250
1006	CA	D	717P	3/3/02	500
S001	SA	D	771A	5/4/02	300
1001	CA	W	915P	7/4/02	100
CC001	CC	D	1015P	1/4/02	250.50
CC002	CC	W	666A	10/4/02	100
1006	CA	D	611P	8/4/02	300

PERFORMANCE ANALYSIS

This section presents some results of experiments conducted to measure the performance of the proposed token-based algorithms (TB cleaner) in comparison with two other algorithms, namely, the basic field matching algorithm (Basic alg—Monge & Elkan, 1996) and the algorithm described in Lee et al. (Lee's alg—1999). All the experiments were performed on a 733 MHZ Intel Pentium III PC with 128 MB main memory running Windows 2000 Professional Edition. All the programs were coded in Java, and the input and output tables were kept in a database managed by Oracle 8i database management system, personal edition. The input data were real data taken from the telephone directory containing names of clients of Bell Canada. Some missing fields (e.g., date of birth) in the input data were added in order to arrive at the desired schema. We also carefully introduced a variety of "dirt" into the data, such as: (i) misspellings, (ii) transposition errors, (iii) inconsistent use of initials in names, (iv) different addressing schemes, (v) synonyms, (vi) homonyms, (vii) record duplications, and (viii) data format differences.

Performance Parameters Used

The performance of each algorithm was measured against four parameters, namely: (1) recall (RC), (2) false-positive error (FPE), (3) reverse false-positive error (RFP), and (4) threshold. Recall is the ratio indicating the number of duplicates correctly identified by a given algorithm. For example, if "x" number of duplicates were identified out of "y" number of dupli-

Table 13. The personal information of customers (refresh CDT)

CID	Name	Sex	Birth	Address
S001	John Smith O	M	25-Dec-70	Sunset # 995 N9B3P4
1006	Florenc e Mike	F	23-Nov-66	334 Rankin M7V 2J2
1001	S. John	M	25-12-1970	995 Sunset Ave, n9b 3p4
CC001	John Smith	M	December 25 1970	No. 995 Sunset Ave N9b 3p4
CC002	Zach Young	F	Jul-30-68	345 AP St P6C 6K1

cates, then the recall is $\frac{x}{y}$, which when expressed in percentage is $\frac{x}{y} \times 100$. False-positive error is a ratio of wrongly identified duplicates. Formally, false-positive errors:

$$\text{FPE} = \frac{\text{number of wrongly identified duplicates}}{\text{total number of identified duplicates}} \times 100.$$

We introduce reverse false-positive error (RFP) in this article as a performance measure; it indicates the number of duplicates that a given algorithm could not identify. Formally:

$$\text{RFP} = \frac{\text{number of duplicates that escaped identification}}{\text{total number of duplicates}} \times 100.$$

We want to see how a given algorithm fluctuates with varied thresholds when all other factors are constant. We arbitrarily chose three thresholds—0.25, 0.44, and 0.80. We maintain in this article that a good data cleaning algorithm should have: (1) a high recall; (2) a very low (better if zero) FPE, hence high precision; and (3) a very low (better if zero) RFP. It should also maintain a steady behavior as threshold varies.

Four Case Experiments

The results of four case studies are given in this section. We used a small-sized input data to enable us to evaluate the output of the experiments. For each experiment, we varied the threshold three times, starting from a low threshold of 0.25, to a medium-sized threshold of 0.44, and finally to a high threshold of 0.80. Our proposed algorithm (TB cleaner), the basic field matching algorithm (Basic alg—Monge & Elkan, 1996), and the algorithm described in Lee et al. (Lee's alg—1999) are compared, and the results for the four case studies are given as CASE 1 to CASE 4 in Table 14. Each of these experiments is described next.

Experiment 1: 20 rows of records, 4 pairs of duplicates, trivial data dirt—results in Table 14, CASE 1.

Experiment 2: 40 rows of records, 7 pairs of duplicates, slightly less trivial data dirt— results in Table 14, CASE 2.

Table 14. Experiments: Recall (RC), false positive error (FPE), and reverse false positive (RFP) error at varied thresholds produced by three algorithms

Algorithms	Match Score Thresholds								
	0.25			0.44			0.80		
	RC	FPE	RFP	RC	FPE	RFP	RC	FPE	RFP
CASE 1	Pairs of duplicates found from 4 actual pairs								
TB cleaner	4	0	0	4	0	0	4	0	0
Basic alg	3	5	1	3	1	1	0	0	4
Lee's alg	2	12	2	3	2	1	0	0	4
CASE 2	Pairs of duplicates found from 7 actual pairs								
TB cleaner	7	1	0	7	1	0	7	0	0
Basic alg	5	4	2	6	0	1	1	0	6
Lee's alg	5	17	2	5	3	2	0	0	7
CASE 3	Pairs of duplicates found from 10 actual pairs								
TB cleaner	10	3	0	10	2	0	10	0	0
Basic alg	7	12	3	7	0	3	1	0	9
Lee's alg	7	58	3	8	6	2	0	0	10
CASE 4	Pairs of duplicates found from 14 actual pairs								
TB cleaner	12	9	2	13	7	1	14	0	0
Basic alg	8	19	6	8	3	6	1	0	13
Lee's alg	8	111	6	8	13	6	0	0	14

Experiment 3: 80 rows of records, 10 pairs of duplicates, advance data dirt—results in Table 14, CASE 3.

Experiment 4: 120 rows of records, 14 pairs of duplicates, advance data dirt—results in Table 14, CASE 4.

It is evident from the experimental results that our algorithm (TB cleaner) achieves an optimal cleaning correctness in all cases when the threshold is 0.80. Looking at the Recall column of Table 14, at a threshold of 0.44, four pairs of duplicates exist in the experimental data in CASE 1, and TBcleaner found all four pairs of duplicates, while the other two algorithms found three pairs. TBcleaner also found all seven pairs and 10 pairs of duplicates in data CASES 2 and 3 respectively. While TBcleaner found 13 of the 14 pairs of duplicates in data CASE 4, the other two algorithms found only eight pairs. Results also show that the token-based algorithm maintained a steady behavior over a spectrum of thresholds because threshold is only needed at one point (i.e., "maybe match") in the course of cleaning. This is not so with Basic alg and Lee's alg, which depend on threshold at all cleaning points. Running the techniques on much larger datasets for different problem domains is a future work worth exploring to better ex-

plore any limitations of the technique and also to test its adequacy for refreshing data warehouses.

CONCLUSIONS AND FUTURE WORK

Two token-based data warehouse cleaning algorithms were proposed in this article. The first algorithm is suitable for cleaning when a data warehouse is being built, while the second algorithm is designed for subsequent or incremental cleaning of an existing data warehouse. The idea behind the techniques is to define smart tokens from two most important fields by applying simple rules for defining numeric, alphabetic, and alphanumeric tokens. Database records now consist of smart token records, composed from field tokens of the records. These smart token records are sorted using two separate most important field tokens. The result of this process is two sorted token tables, which are used to compare neighboring records for a match. Duplicates are easily detected from these tables, and warehouse identifiers are generated for each set of duplicates using the concatenation of its first record's tokens. These warehouse identifiers are later used for quick incremental record identification and refreshing.

The notion of "token records" was introduced for recording comparison. Existing algorithms only use token keys extracted from records for either sorting or clustering (or both). Results from experiments show that our token-based algorithm outperforms the other two comparable algorithms. It has a recall close to 100%, as well as negligible false-positive errors. We succeeded in reducing the number of token tables to a constant of 2, irrespective of the number of fields selected by the user. This is a great improvement over the algorithms described in Hernandez and Stolfo (1995, 1998), where the number of token key tables increases proportionally to the number of fields in use.

In addition, the smart tokens are more likely applicable to domain-independent data cleaning, and could be used as warehouse identifiers to enhance the process of incremental cleaning and refreshing of integrated data.

Future work should consider applying this token-based cleaning technique on unstructured (like complete text files) and semi-structured (like XML file) data. This approach can also be applied to stream and sensor network data cleaning, where immediate answers are needed. Applying to network dataset will also contribute to recent network intrusion issues like spams and viruses.

ACKNOWLEDGEMENTS

This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

REFERENCES

Bitton, D. & Dewitt, D.J. (1983). Duplicate record elimination in large data files. *ACM Transactions on Database Systems, 8*(2), 255-265.

Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., & Rosati, R. (2000). Data integration in data warehousing. *International Journal of Cooperative Information Systems.*

Delvin, B. (1997). *Data warehouse from*

architecture to implementation. Addison-Wesley.

Ezeife, C.I.. (2001). Selecting and materializing horizontally partitioned warehouse views. *Elsevier Journal of Data and Knowledge Engineering, 36*(2), 185-210.

Galharda, H., Florescu, D., Shasha, D., Simon, E., & Saita, C. (2001). Declarative data cleaning: Language, model and algorithms. *Proceedings of the 27th VLDB Conference,* Rome, Italy.

Hernandez, M.A. & Stolfo, S.J. (1995). The merge/purge problem for large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 127-138).

Hernandez, M.A & Stolfo, S.J. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery, 2*, 9-37.

Inmon, W.H. (1996). *Building the data warehouse* (2nd edition). New York: John Wiley & Sons.

Kimball, R. (1996). Dealing with dirty data. *DBMS Online, 9*(10).

Lee, M.L., Hongjun, L., Tok, W.L., & Yee, T.K (1999). Cleansing data for mining and warehousing. *Proceedings of the 10th International Conference on Database and Expert Systems Applications* (DEXA 99), Florence, Italy.

Monge, A.E. & Elkan, C.P. (1996). The field matching problems: Algorithms and applications. *Proceedings of the 2nd International Conference on Knowledge and Data Mining* (pp. 267-270).

Ohanekwu, T.E. & Ezeife, C.I. (2003, January). A token-based data cleaning technique for data warehouse systems. *Proceedings of the International Workshop on Data Quality in Cooperative Information Systems* (pp. 21-26), held in conjunction with the *9th International Conference on Database Theory* (ICDT 2003), Siena, Italy.

Parent, C. & Spaccapietra, S. (1998). Issues and approaches of database integration. *Communications of the ACM, 41*(5), 166-178.

Raman, V. & Hellerstein, J.M. (2001). Potters wheel: An interactive framework for data cleaning and transformation. *Proceedings of the 27th VLDB Conference,* Rome, Italy.

Simoudis, E., Livezey, B., & Kerber, R. (1995). Using recon for data cleaning. *Proceedings of KDD* (pp. 282-287).

Christie I. Ezeife received her MSc in computer science from Simon Fraser University, Canada (1988), and a PhD in computer science from the University of Manitoba, Canada (1995). She has held academic positions at a number of universities, and has been an associate professor of computer science at the University of Windsor, Canada, since 1999. She has authored several technical publications, including three journal articles in the International Journal of Distributed and Parallel Databases *and the* Journal of Data Mining and Knowledge Discovery *(Kluwer Academic Publishers). She has authored two books on problem solving and programs with C (Thomson Learning Publishers, Canada).*

Timothy Ohanekwu obtained his BSc (Honors) in computer science from the University of Ibadan, Nigeria (1997). He also received his Master of Information Science (MINFSC) from

the same university in 1998. Timothy later received an MSc in computer science from the Univeristy of Windsor (2002). While currently pursuing a Master in Education, he is interested in pursuing a PhD program in the future in the areas of computer information systems and automated learning.