A Token-Based Data Cleaning Technique for Data Warehouse Systems

Timothy E. Ohanekwu
School of Computer Science,
University of Windsor

C.I. Ezeife
School of Computer Science,
University of Windsor,
Windsor, Ontario,
Canada N9B 3P4.
cezeife@cs.uwindsor.ca

***Abstract*— Data cleaning is a process for determining whether two or more records defined differently in a database, actually represent the same real world object. During data cleaning, multiple records representing the same real life object are identified, assigned only one unique database identification, and only one copy of exact duplicate records is retained. Most existing work on data cleaning, identify record duplicates by computing match scores compared against a given match score threshold. Some use the entire records for long string comparisons that involve a number of passes. Determining optimal match score threshold in a domain is hard and straight long string comparisons with many passes is inefficient.**

This paper proposes a technique that eliminates the need to rely on match threshold by defining smart tokens that are used for identifying duplicates. This approach also eliminates the need to use the entire long string records with multiple passes, for duplicate identification.

Keywords: Data Warehouse, Data Cleaning, Token-Based search, Data Dirt

I. Introduction

A data warehouse, is a database made up of **integrated, subject-oriented, time-variant and non-volatile** data designed for business decision, multidimensional querying [7]. Integrated data sources of a data warehouse may reside on different hardware and software platforms and may have different data models. A data warehouse based on the star schema consists of (1) a fact table, which contains the integrated, subject-oriented and time-variant records, and (2) a set of dimension tables describing subject, foreign key attributes in the fact table. 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 paper presents a technique for efficiently cleaning data for data warehouse tables, which uses well-defined tokens for record matching.

Data cleaning, is an automated method for examining the data, detecting missing and incorrect values and correcting them [12]. It focuses on eliminating variations in data contents and reducing data redundancy aimed at improving the overall data consistency [3]. Data cleaning first detects dirty records by determining whether two or more records represented differently refer to the same real world entity, and then, it cleans the dirty records by either (i) collapsing them to get a consolidated whole devoid of missing parts, (ii) unifying them with a single entity identity and (iii) retaining only one copy of records that are exact duplicates. 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", i.e., 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". While all other causes of data dirt can be as a result of "oversight" or "human errors", synonyms and homonyms are not. For example, a document collection center in a unit of an organization may decide to use entity acronyms/abbreviations, "ACM", while another center may write it in full, as "Association for Computing Machinery". Homonymous dirt arises when the same "term" or "expressions" refer to two or more entities, e.g., many occurrences of "John Smith" in a data source may refer to different persons.

A. Related Work

Bitton et al. [1] sort on designated fields to bring potentially identical records together in a large data file. However, sorting is based on "dirty" fields, which may fail to bring matching records together, and its time complexity is quadratic in the number of records. Hernandez et al. [5], [6] solves the merge/purge problem in a large database by forming keys from some selected fields, sorting the entire data set on the keys, clustering the sorted records and using a scanning window of a fixed size to reduce the number of comparisons. Record comparison is still based on the original dirty records. The equational theory used in the multi-pass version of the work is a time consuming process. The basic field matching algorithm [10] extracts and sorts atomic strings within fields, finds the number of strings that match and computes the match score used to decide if the two fields are the same. The accuracy of the the basic field matching algorithm is dependent on the match score threshold for deciding if any two input string match. The work described in [9] introduces the idea of field pre-processing with external data source, prior to sorting and comparison phases as well as "tokenizing of fields". Pre-processing the dirty records with external data source like birth registry may not always be feasible and final comparison of strings for a match still involves the entire long strings. Work in [4], [11] enhance the data integration and cleaning process with declarative operators that allow for dynamic and interactive cleaning.

B. Contributions

While existing techniques have used tokens for bringing likely duplicate records together [5], [6], [9], used pre-determined match score thresholds to decide on a match between two input strings [9], [10], depended on external or interactive input during duplicate detection [4], [9], [11], 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 paper contributes to.

This paper proposes a token-based data cleaning algorithm, TB cleaner, which first defines smart tokens from most important fields of records, compares and identifies duplicate records with those tokens. *Token-based* technique achieves a better result than the *record-based* techniques of comparable algorithms. By using short lengthened

TABLE I

TSA, A TRANSACTION HISTORY FOR SA CUSTOMERS

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 II

TCA, A TRANSACTION HISTORY FOR CA CUSTOMERS

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

tokens for record comparisons, a high recall/precision is achieved. The technique also drastically lowers the dependency of the data cleaning on match "threshold" choice.

C. Outline of the Paper

The rest of this paper is organized as follows. An example of a yet to be cleaned data warehouse is given in section 2. Section 3 presents the token-based data cleaning algorithm with an example, while section 4 presents experimental performance study to support the contributions of this work. Section 5 presents conclusions and future work.

II. AN EXAMPLE

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 data warehouse fact table (FT), shown in Table III and the customer dimension table (CDT) given as Table IV are yet to be cleaned. Tables I and II 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 CDT (Table IV) and FT (Table III) are described in section 2.2, while the tasks accomplished by the proposed algorithm are outlined in section 2.3. The data warehouse schema, which represents an integration of the savings account and checking account data sources is shown below.

```
FT(WID, Trans-code, Account-code, Trans-time,
                                        Amount)
CDT(WID, Name, Sex, Phone, DBirth, Address)
Transactions (Trans-code, Trans-name)
Accounts(Account-code, Account-name)
Times (Trans-time, Day, Month, Year)
```

This data warehouse consists of the main fact table, (FT) (Table III) and four dimension tables, only one of which, CDT (Table IV) will be cleaned in the example.

A. Dirt in the Customer Dimension and Fact Tables

Two levels of dirt exist in Table IV, namely, *field or attribute level dirt and record level dirt*. The field level dirt is the dirt that occurs when each field in a record is considered in isolation. For example, the "WID" field of Table IV has "*type-mismatch*" dirt, since different data types are used to represent potentially 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 IV include: (i) typographic errors, (ii) different addressing conventions (in address field), etc. 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 IV 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 III) contains only field level dirt in the "WID" field. As it is, using different WID to represent the same customer makes it difficult to determine all the transactions conducted by the same customers.

TABLE III

THE YET TO BE CLEANED FACT TABLE

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 IV

THE YET TO BE CLEANED CUSTOMER DIMENSION TABLE

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. 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. Rd. M8C 8S4

B. The Cleaning Tasks

Two cleaning tasks to be carried out on the customer dimension table (Table IV) are: (i) duplicate detection, and (ii) duplicate elimination. Duplicate detection requires a combination of (pieces of) information from two or more fields to find if two or more records

TABLE V

A TARGET FACT TABLE AFTER CLEANING

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

TABLE VI

A TARGET CUSTOMER DIMENSION TABLE AFTER CLEANING

Row	WID	Name	Sex	Phone	Birth	Address
1	122570JOS	John Smith O	M	(519) 111-1234	25-Dec-70	Sunset # 995 N9B3P4
2	010175EOT	Tim E. Ohanekwu	M	2566416	01-Jan-75	ABCD St. No. 695 n9b 2t7
3	080864CJ	Colette Jones	M	123-4567	08/Aug/64	600 XYZ apt 5a5 N7C4K4
4	111172ADD	Ambrose A. Diana	F	519 6669999	Nov/11/72	4 Church Rd. N8K6t6
5	103078JS	Smith John	F	519 560 3626	30-Oct-78	182 Haven Ave M9B3J7
6	052367EOT	Edema Tom Obi	M	977-5950	23-Mar-1967	98 Haven Rd. M8C 8S4

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 III) and the customer dimension table (Table IV) is to unify the entity identity, 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 "Tim E. Ohanekwu", "Ambrose A. Diana", etc. The solution to this problem is to use the same identity value for the same real world entity. For example, "122570JOS" will be used for all the 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", "S004" and "1003". Section 3 discusses the process of producing the desired fact table (Table V) and customer dimension table (Table IV) after cleaning Tables III and IV respectively, with the proposed TB cleaner.

III. THE PROPOSED TOKEN-BASED DATA CLEANING ALGORITHM

The proposed token-based data cleaning algorithm, TB cleaner, accepts "dirty" source tables, such as Tables I (recent SA transactions), II (recent CA transactions), III (dirty FT), IV (dirty CDT) and returns "cleaned" data warehouse tables, such as Tables V (cleaned FT) and VI (cleaned 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 VII). 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 VIII (tokens sorted on birth day) and IX (tokens sorted on Name). **Token-records** in close neighborhood are compared for a match and warehouse id (WID) is generated for records. The four steps (in sequence) in the algorithm are described below.

Step 1: Selection and Ranking of Fields: The user familiar with application domain, is expected to select and rank 2 or 3 fields that could be combined to perfectly discriminate one record from another. In our banking domain, selected are the following fields from Table 6: "Birth", "Name" and "Address" and ranked them in the order given.

Step 2: Extraction and Formation of Tokens: This step entails decomposing a divisible element, e into its divisible and/or indivisible tokens. Further decomposition is needed for the divisible tokens. The indivisible tokens are recomposed in a desired order for the element, e. Conversion from one data type to another may be necessary where desired. For example, a date element, (like, "19-Dec-1978") is decomposed into three members: day (19), month (Dec) and year (1978). The month, "Dec" is converted to its numeric equivalent, while the year (1978) is decomposed further into the "century" (19) and "Year" (78) parts and only the day, month and year parts (not including the century part) are used. The final token is "121978", sorted in ascending order. Some tokens in the original element are considered unimportant, hence will be discarded. For example, "#", ".", "(", ")", "–", "/", "′" are unimportant in the "date", "address", and "telephone" elements. Title tokens like "Mr.", "Ms", and "Dr." are excluded in "name" elements, while stop words like "the", "of", "for", "in" are considered unimportant in publication titles. We recognize and define three types of tokens, as follows.

(i) **Numeric Tokens:** These tokens comprise only digits (0,1,2,3,4,5,6,7,8,9) and are obtained from numeric-dominated fields such as "birth date", "telephone numbers", "social insurance number", "student numbers".

(ii) **Alphabetic Tokens:** Tokens in this category consist only of alphabets (aA - zZ) and are obtained from fields consisting mainly of alphabets, e.g., "name of persons", "company names", "journal names", "publication titles". A function is defined, which takes string of input like "Dr. Christy I. Engings" or "C.I. Engings" or "Engings Iny C." and returns the following alphabetic token: "CEI". To form the alphabetic token, the first character of each word in the field is obtained, and the token is made up of these characters sorted in an order.

(iii) **Alphanumeric Tokens:** These tokens comprise both numeric and alphabetic tokens and could be obtained from fields containing both numbers and strings, e.g., an "address" field. A function is defined, which (1) decomposes a given alphanumeric element into its constituent members, (2) scans through the set of members and selects only tokens that are either numeric or alphanumeric in nature, (3) further decomposes each of the alphanumeric part to its numeric and alphabetic parts, and (4) sorts the set of tokens in certain order (e.g., ascending) to get an alphanumeric token key. For example, the function described above takes "600 XYZ blvd apt 585 N7C4K4" and returns "585600744NCK" as the alphanumeric

TABLE VII

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 VIII

TOKENS SORTED IN ASCENDING ORDER OF BIRTH 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

token. This is obtained from one of the numeric and alphanumeric constituent members of this address, 600 585 744 NCK. When these are sorted in ascending order, the token 585600744NCK is obtained. Applying the token extraction procedure on the "name", "birth" and 'address" fields of Table IV produces Table VII.

Step 3: Sorting of Tokens: The table of tokens (Table VII) is sorted separately on the two most uniquely identifying fields according to the ranking by the user. Therefore, sorting respectively on the "birth" and "name" token fields produces Tables VIII and IX respectively. Records "1003" and "S004" in Table VIII (highlighted) are not in 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 IX due to "token inequality". It can also be said that records "1003" and "S004", which are spread apart in Table VIII are close neighbors in Table IX. Conversely, records "S001" and "1001", which are not close neighbors in Table IX are brought to close neighborhood in Table VIII. This is the essence of sorting on two tokens. 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

TABLE IX

TOKENS SORTED IN ASCENDING ORDER OF NAME 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**

TABLE X

DUPLICATE RECORD LISTS FOR WID GENERATION

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

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).

(i) Detection of Duplicates: The "token-based" record match is based on the following valid argument:

> *If tokens are sufficiently adequate to bring potential duplicates 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 R_1 and R_2 are (i) perfect match (if SMC is 1.0), (ii) near perfect match (if SMC is between 0.66 and 0.99), (iii) maybe a match (if SMC is between 0.33 and 0.67) and (iv) 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 token pair match or not. Given two tokens t_1 and t_2 with m and n characters respectively. 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 $SMR \geq 0.67$. Once the SMR 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 VIII is the following pairs of duplicate records: (S001,1001), (S002,1004), (S003,1002). Similarly, applying the same procedures to the second token table, Table IX, 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).

(ii) Elimination of Duplicates and Generation of WID: Table X shows the final list of record duplicates obtained. The WID is formed from the first record in the duplicate set of Table X by concatenating the two most powerful 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 VI) and an entity-unified fact table (Table V). In addition, a log table is generated and stored for subsequent cleaning tasks.

IV. Performance Analysis

This section presents some results of the experiments conducted to measure the performance of the proposed token-based algorithm (TB Cleaner) in comparison with two other algorithms, namely, the basic field matching algorithm (Basic) [10] and the algorithm described in [9] (Lee's). 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.

A. Performance Parameters Used

The performance of each algorithm was measured against four parameters, namely, (i) recall (RC), (ii) false-positive error (FPE), (iii) reverse false-positive error (RFP) and (iv) threshold. Recall is the a 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 duplicates, 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, FPE = $\frac{number\ of\ wrongly\ identified\ duplicates}{total\ number\ of\ identified\ duplicates} \times 100$. We introduce **reverse false-positive error, (RFP)** in this paper as a performance measure and it indicates the number of duplicates that a given algorithm could not identify. Formally, RFP = $\frac{number\ of\ duplicates\ that\ escaped\ identification}{total\ number\ of\ duplicates} \times 100$. 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 paper that a good data cleaning algorithm should: (i) have a high recall, (ii) have a very low (better if zero) FPE, hence high precision, (iii) a very low (better if zero) RFP and (iv) maintain a steady behavior as threshold varies.

B. Four Case Experiments

The results of four case studies are given in this section. We used a small sized input data to enable us 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.) [10], the algorithm described in [9] (Lee's alg) are compared and the results for the four case study are given as CASE 1 to CASE 4 in Table XI. Each of these experiments is described next.

Experiment 1: 20 rows of records, 4 pairs of duplicates, trivial data dirt - the results as CASE 1 of Table XI.
Experiment 2: 40 rows of records, 7 pairs of duplicates, slightly less trivial data dirt - Table XI, CASE 2.
Experiment 3: 80 rows of records, 10 pairs of duplicates, advance data dirt - results are in Table XI, CASE 3.
Experiment 4: 120 rows of records, 14 pairs of duplicates, advance data dirt - results in Table XI, 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 XI, at a threshold of 0.44, 4 pairs of duplicates exist in the experimental data in Case 1 and TBcleaner found all 4 pairs of duplicates, while the other two algorithms found 3 pairs. TBcleaner also found all 7 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 8 pairs. Results also show that 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.

TABLE XI
Experiments : Recall(RC), False positive error (FPE) and Reverse false positive (RFP) error at Varied Thresholds produced by 3 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

V. Conclusions and Future Work

A token-based algorithm for cleaning a data warehouse is presented. The notion of "token records" was introduced for record comparison. Existing algorithms use token keys extracted from records for only sorting and/or clustering. The results from the experiments show that the proposed token-based algorithm outperforms the other two 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 [5], [6], 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, and semi-structured data.

VI. Acknowledgment

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

[1] D. Bitton and D.J. Dewitt. Duplicate Record Elimination in Large Data Files. ACM Transactions on Database Systems, Vol. 8, No. 2, PP 255 - 265, June 1983.

[2] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data Integration in Data Warehousing. Int'l Journal of Cooperative Information Systems, 2000.
[3] B. Delvin. Data warehouse from architecture to implementation, Addison-Wesley, 1997.
[4] H. Galharda and D. Florescu and D. Shasha and E. Simon and C. Saita. Declarative Data Cleaning: Language, Model and Algorithms. Proceedings of the 27th VLDB conference, Roma, Italy, 2001.
[5] M.A Hernandez and S.J Stolfo. The Merge/Purge Problem for Large Databases. In Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, pp 127 - 138, May 1995.
[6] M.A Hernandez and S.J Stolfo. Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem. Data Mining and Knowledge Discovery, 2, 9 - 37 1998.
[7] W.H Inmon. Building the Data Warehouse, second edition, John Wiley, 1996.
[8] R. Kimball. Dealing with Dirty Data. DBMS Online, vol. 9, no. 10, September 1996. Available at URL http://www.dbmsmag.com/9609d14.htm.
[9] M.L. Lee and L. Hongjun and W.L Tok and T.K Yee. Cleansing Data for Mining and Warehousing. In Proceedings of the 10th Int'l Conference on Database and Expert Systems Applications (DEXA 99), Florence, Italy, August 1999.
[10] A.E Monge and C.P Elkan. The Field Matching Problems: Algorithms and Applications. Proceedings of the 2nd Int'l Conference on Knowledge and Data Mining pp 267 - 270, 1996.
[11] V. Raman and J.M. Hellerstein. Potters Wheel: An Interactive Framework for Data Cleaning and Transformation. Proceedings of the 27th VLDB conference, Roma, Italy, 2001.
[12] E. Simoudis and B. Livezey and R. Kerber. Using Recon for Data Cleaning. In Proceedings of KDD 1995, pp 282-287.