Detecting Data Inconsistency for Multidatabases

Ke Wang
Dept. of Info. Sys. & Comp. Sci.
National University of Singapore
Lower Kent Ridge Road, Singapore, 0511
wangk@iscs.nus.sg

Weining Zhang*
Dept. of Math. & Comp. Sci.
University of Lethbridge
Lethbridge, Alberta, Canada, T1K 3M4
zhang@cs.uleth.ca

Abstract

Traditional approaches to database integration require that a common key exist in all participating relations that model equivalent entities in the real-world, therefore, compromising the logical heterogeneity of multidatabases. The recent proposal of using knowledge to identify equivalent entities without requiring a common key gives rise to the issue of detecting potential inconsistency during entity identification. In this paper, criteria of data consistency are proposed and incremental tests in the process of updating data and knowledge are considered. The proposed framework and algorithms are tested by an experiment on three databases extracted from the real-world.

Key words. Inconsistency detecting, data integration, entity identification, multidatabase

1 Introduction

Multidatabase systems provide integrated, global access to autonomous, heterogeneous local databases via a simple global request. A central activity required for processing a global request is to resolve the logical heterogeneity as the result of the local autonomy of multidatabases, namely, *schema integration* and *data integration*. Schema integration resolves schematic heterogeneity such as differences in attribute name and domain, and differences in data format and structure. Data integration, on the other hand, has to solve the following problems

- *entity identification*: identify object instances in different databases that model the same real-world entities.
- *missing or inconsistent data*: some data items may be recorded in one database but not in others, or several databases record the same data item but give it different values.

*This author's work was supported by an NSERC Research Grant

Two types of inconsistent data have been addressed in the literature. The first type occurs when the same data is represented differently in different databases due to the schematic heterogeneity. Such inconsistency is typically *resolved* by renaming attributes, domain mapping, value conversion, and structure transformation. The second type of inconsistency occurs, due to the failure of maintaining databases, when equivalent data items in different databases, which are expected to have the same value, store different values. Data inconsistency of this type amounts to an error in modeling the real-world and should be *detected* at the time of updating data or knowledge.

We assume that local schemas have been translated into the relational model and that schematic heterogeneity has been resolved by schema integration. However, participating databases may not share a common key and it is not immediately clear which tuples in these databases model equivalent entities. To identify equivalent entities, we propose an inference process that derives missing values with respect to some given knowledge. We define a notion of consistency for the correctness of integration, given the inference power of knowledge.

Detecting algorithms of inconsistencies are presented. We focus on an incremental detecting method in the process of updating data and knowledge. With an incremental method, an update of data is tested at nearly zero communication cost and a number of table lookups bounded by the number of involved schemas. By indexing or hashing technique each table lookup needs only a small number of block accesses. Checking update of knowledge is less efficient than that of data in general, but it is affordable for knowledge that is usually less dynamic. An experiment is conducted on the proposed framework and algorithms based on a case study in the real life.

In [4, 12], constraint enforcement in heterogeneous multidatabases has been considered. They have described languages of specifying constraints across several databases and relational methods of testing constraints. However, the consistency constraint encountered in entity identification was not addressed in that work, except for the simple case where equivalent entities can be found by comparing directly key identifiers. In the absence of common key identifiers and the presence of background knowledge, the problem of inconsistency detecting can be formulated as constraints over database relations and knowledge and the distributed constraint checking techniques, e.g., [5, 14], can be applied. However, too many constraints, derived from all possible ways of deriving data, will be generated and checked. Our methods make use of special structures of knowledge tables so that update of relations can be done independently of the number of data derivations. The method uses materialized views [9] for incremental detecting of inconsistency among several databases.

The rest of the paper is organized as follows: Section 2 defines an inference engine that is used in entity identification. Sections 3 and 4 address the consistency problem in a single database and a multidatabase, with the former being the foundation of the latter. In each of these sections, the notion of consistency, inconsistency detecting, and the incremental implementation are presented. Section 5 describes an experiment of the proposed framework and algorithms on a case study. Section 6 concludes the paper.

2 Deriving Data Using Knowledge

Instead of using common keys, our approach uses additional knowledge to infer missing data items needed for identifying equivalent entities.

Example 2.1 (Running example) In Table 1, relations $r(R)$ and $s(S)$ (or simply r and s) store information about restaurants in different databases. Without additional knowledge, it is not possible to tell which of the first three tuples in s model the same restaurant as the first tuple in r, even though the restaurants they model have the same name. Given additional knowledge on each database stored in tables of form $M(X \rightarrow Y)$, where a tuple u in $M(X \rightarrow Y)$ means that if an entity has value $u[X]$ on X it also has value $u[Y]$ on Y, one can infer, based on the first tuple in each of $M(speciality \rightarrow cuisine)$ and $M(name, street \rightarrow speciality)$, that the first tuple in s models a Chinese restaurant, and the first tuple in r models a restaurant specialized in Hunan food, and so on. If the restaurants modeled by the multidatabase can be uniquely identified by values of $\{name, cuisine, speciality\}$, one can conclude that the first tuple in r and that in s model the same restaurant in the real-world, thus can be integrated to provide more complete information to global users. □

Definition 2.1 A *mapping schema* on a relation schema R is a statement of form $X \rightarrow Y$, where X and Y are non-empty sets of attributes such that $X \cap Y = \emptyset$ and $R \cap Y = \emptyset$. A *mapping* on $X \rightarrow Y$, denoted $M(X \rightarrow Y)$, is a function that, given values for all attributes in X, returns a value for each attribute in Y. □

In general, mappings encode common sense and other discovered knowledge about data, and are assumed to be on the same site where the data reside. In the rest of the paper, let r denote a relation with a schema R, Ω denote a collection of mapping schemas on R, and τ denote an assignment of mappings to mapping schemas in Ω.

We now formalize the process of inferring missing values. Let X be a set of attributes not necessarily in R. A sequence $< X_1 \rightarrow Y_1, \ldots, X_k \rightarrow Y_k >$ of mapping schemas in Ω, $k \geq 0$, is a *derivation* of X from R if $X_1 \subseteq R$ and $X_i \subseteq RY_1 \ldots Y_{i-1}$ for $1 < i \leq k$, and $X \subseteq RY_1 \ldots Y_k$. A derivation of X from R is *minimal* if removing any mapping schema from it does not result in a derivation of X from R. R^+ denotes the set of all attributes appearing in some derivation from R.

Mapping Process: Given (r, τ), values on attributes $R^+ - R$ can be derived as follows. Let $aug(r)$ be the relation on R^+ obtained from r by padding every tuple in r with NULLs for all attributes in $R^+ - R$. Whenever there exists some mapping $M(X \rightarrow Y)$ in τ such that $t[X] = u[X]$, where t is a tuple in $aug(r)$ and u is a tuple in $M(X \rightarrow Y)$, $t[Y]$ is replaced by $u[Y]$. Mappings in τ are applied to $aug(T)$ in this way until either a non-NULL value is replaced with a different non-NULL value, in which case a *conflict* occurs, or no more change can be made, in which case the final $aug(r)$ is denoted by r'.

Note that the mapping process is a conceptual not an implementational model.

Example 2.2 Consider (r, τ_1) in Example 2.1. By applying mappings in $M(name, street \rightarrow speciality)$, $M(street \rightarrow county)$, $M(name, county \rightarrow speciality)$, and $M(annual\_profit \rightarrow monthly\_profit)$ to $aug(r)$ in order, values on $speciality$, $county$, $monthly\_profit$ of r are derived. as shown in r' in Table 2. □

3 M-Consistencies: A Single Database

We first consider the consistency problem in a single database, where each attribute name has a unique meaning and derivation of conflict values signals an error in modeling the real-world.

$r(R)$

<u>name</u>	<u>cuisine</u>	street	annual-profit	owner
Twin Cities	Chinese	Co. B2	102k	Lee
It's Greek	Greek	Front Ave.	120k	Pangalos
Anjuman	Indian	LeSalle Ave.	240k	Raman
Village Wok	Chinese	Wash. Ave.	200k	Dong

$s(S)$

<u>name</u>	<u>speciality</u>	county	monthly-profit	owner
Twin Cities	Hunan	Roseville	8.5k	Lee
Twin Cities	Sichuan	Hennepin	9k	Lee
Twin Cities	Pizza	Roseville	11k	Thanos
It's Greek	Gyros	Ramsey	10k	Pangalos
Anjuman	Mughalai	Mpls.	20k	Raman
Wong's	Canton	Roseville	12k	Wong

$M(speciality \rightarrow cuisine)$ on S

speciality	cuisine
Hunan	Chinese
Sichuan	Chinese
Canton	Chinese
Gyros	Greek
Mughalai	Indian
Pizza	Italy

$M(street \rightarrow county)$ on R

street	county
LeSalle Ave.	Mpls.
Front Ave.	Ramsey
Co. B2	Roseville
Wash. Ave.	Roseville

$M(name, street \rightarrow speciality)$ on R

name	street	speciality
Twin Cities	Co.B2	Hunan
Anjuman	LeSalle Ave.	Mughalai

$M(name, county \rightarrow speciality)$ on R

name	county	speciality
It's Greek	Ramsey	Gyros
Twin Cities	Roseville	Hunan

$M(annual\_profit \rightarrow monthly\_profit)$ on R: $monthly\_profit = annual\_profit/12$
$M(monthly\_profit \rightarrow annual\_profit)$ on S: $annual\_profit = monthly\_profit * 12$

Table 1: The running example

Definition 3.1 (r, τ) is *M-consistent* (M for mapping) if no conflict is derived by the mapping process wrt (r, τ). $\Box$

For a given (r, τ), M-consistency may be tested by using the mapping process or by formulating the M-consistency as constraints and applying constraint checking techniques in the literature. However, both these approaches suffer from poor performances. In the following, we consider an incremental detecting method. The idea is to materialize tuples on R, not necessarily in r, that will lead to a conflict if mappings are applied to them. These tuples are completely determined by the given mappings.

Given a sequence $\lambda =< X_1 \rightarrow Y_1, \ldots, X_k \rightarrow Y_k >$ of mapping schemas, let $ATT(\lambda) = X_1Y_1 \ldots X_kY_k$. For sets X and Y of attributes such that $X \cap Y = \emptyset$, $\theta(X|Y)$ denotes the formula $(X = X) \wedge (Y \neq Y)$, where $X = X$ is conjunction of $A = A$ for all $A \in X$, and $Y \neq Y$ is disjunction of $A \neq A$ for all $A \in Y$. Let $Y \rightarrow Z$ and $Y' \rightarrow Z'$ be two mapping schemas such that $Z \cap Z' \neq \emptyset$. A *colliding derivation* for $Y \rightarrow Z, Y' \rightarrow Z'$ from R is a sequence of mapping schemas $\lambda =< W_1 \rightarrow X_1, \ldots, W_k \rightarrow X_k, Y \rightarrow Z, Y' \rightarrow Z' >$, where $< W_1 \rightarrow X_1, \ldots, W_k \rightarrow X_k >$ is a minimal derivation of YY' from R. Let $U = Y \cap Y'$ and $V = Z \cap Z'$. The *CS-relation* (*Conflict Source relation*) for the colliding derivation λ, denoted $CS(\lambda)$, is defined as $\prod_{R \cap ATT(\lambda)}(M(W_1 \rightarrow X_1) \bowtie \ldots \bowtie M(W_k \rightarrow X_k) \bowtie \prod_{YY'}(M(Y \rightarrow Z) \bowtie_{\theta(U|V)} M(Y' \rightarrow Z')))$, where $\bowtie$ is the natural join. Intuitively, $CS(\lambda)$ contains all possible values that will lead to a conflict if mappings are applied in the order of λ.

r'

name	cuisine	speciality	street	county	m_profit	a_profit	owner
Twin Cities	Chinese	Hunan	Co. B2	Roseville	8.5k	102k	Lee
It's Greek	Greek	Gyros	Front Ave.	Ramsey	10k	120k	Pangalos
Anjuman	Indian	Mughalai	LeSalle Ave.	Mpls.	20k	240	Raman
Village Wok	Chinese	NULL	Wash. Ave.	Roseville	17k	204k	Dong

Table 2: r' in Example 2.2

Theorem 3.1 (r, τ) is M-consistent if and only if $\prod_{R \cap ATT(\lambda)}(r)$ does not contain any tuple in $CS(\lambda)$ for any colliding derivation λ from R. □

Example 3.1 Consider (r, τ_1'), where r and τ_1' are as given in Example 2.1, except that τ_1' is obtained from τ_1 by adding a mapping $< name = Anjuman, county = Mpls., speciality = Gyros >$ into $M(name, county, \rightarrow speciality)$. Only mapping schemas $f : name, street \rightarrow speciality$ and $f' : name, county \rightarrow speciality$ have non-disjoint right-hand sides, with $\theta(U|V)$ being $name = name \wedge speciality \neq speciality$. The only colliding derivation λ for f, f' from R is $< street \rightarrow county, f, f' >$. It can be verified that

$$CS(\lambda) = \prod_{name,street}(M(street \rightarrow county) \bowtie \prod_{YY'}(M(f) \bowtie_{\theta(U|V)} M(f')))$$

returns tuple $< name = Anjuman, street = LeSalle\ Ave. >$, which is contained in $\prod_{name,street}(r)$. By Theorem 3.1, (r, τ_1') is not mapping-consistent. □

Incremental Detecting 3.1: Let (r, τ) be M-consistent. We wish to test if insertion and deletion of a tuple of r or τ preserves the M-consistency.

Insertion of tuple t into r. Compute t' by applying the mapping process to t. Initially, let $t' = t$ and $Z = R$, and all mapping schemas are marked *unprocessed*. If there is an *unprocessed* $X \rightarrow Y$ such that $X \subseteq Z$ and $Y \not\subseteq Z$, mark $X \rightarrow Y$ as *processed* and retrieve the tuple in $M(X \rightarrow Y)$ that has $t[X]$ on X. If no tuple is returned, do nothing; otherwise, expand t' by values on Y of the returned tuple and let $Z = ZY$. If the expansion replaces a constant by another constant, reject t and stop. Repeat the above steps until either there is no *unprocessed* $X \rightarrow Y$ such that $X \subseteq Z$ and $Y \not\subseteq Z$, or t is rejected. If t is not rejected, add it to r.

Deletion of tuple t from r. Free.

Insertion of tuple t into τ. Assume that t is inserted into mapping M. Let $CS(\lambda)(M/t)$ denote $CS(\lambda)$ in which M is replaced by $\{t\}$. If $\prod_{R \cap ATT(\lambda)}(r) \cap CS(\lambda)(M/t) \neq \emptyset$ for some $CS(\lambda)$ involving M, reject t and stop. If t is not rejected, add it to M.

Deletion of tuple t from τ. Free.

Efficiency of Incremental Detecting: Assume that the mappings are accessed through B^+-tree or hashing. The number of block accesses needed to insert a tuple into r is bounded by $s * d$, where s is the number of involved mapping schemas, and d is the maximal depth of B^+-trees of mappings or a small constant for hashing. d is usually no more than 5 and grows very slowly with the size of mappings. Although detecting for insertion into mapping M is more costly due to computation of $CS(\lambda)(M/t)$, the update to mappings are usually less frequent than that to relations.

4 EI-Consistency: A Multidatabase

We now consider multidatabases and assume that the schema integration has been completed. Since different entities may be modeled in different databases with identical keys, an extended key [11] will be used to model uniquely an entity in the multidatabase.

Definition 4.1 (Extended key) Let K_i be the key of R_i, $1 \leq i \leq m$. An *extended key*, denoted K, is a minimal set of attributes $K_1 \cup \ldots \cup K_m \cup W$, needed to uniquely model an entity in the global domain D, where $W \subseteq (R_1 \cup \ldots \cup R_m) - (K_1 \cup \ldots \cup K_m)$. □

Let K be the chosen extended key. Tuples modeling the same entity in different databases will be identified with the same K values.

Definition 4.2 (Entity identification) Consider M-consistent (r_1, τ_1), ..., (r_m, τ_m) from m different databases that model entities of the same type. Let $t_i' \in r_i'$ and $t_j' \in r_j'$ be derived from $t_i \in r_i$ and $t_j \in r_j$ by mappings. A *conflict* occurs in entity identification if $t_i'[K] = t_j'[K]$ and $t_i[A] \neq t_j[A]$ for some $A \in R_i \cap R_j - K$. If no conflict occurs and $t_i'[K] = t_j'[K]$, the entity identification infers that t_i and t_j model the same real-world entity which has value $t_i'[K]$ on K, value $t_i[R_i - K]$ on $R_i - K$, and value $t_j[R_j - K]$ on $R_j - K$. □

Example 4.1 The $< (r,\tau_1),(s,\tau_2) >$ in Example 2.1 can be shown to be EI-consistent, if $K = \{name, cuisine, speciality\}$. However, if s is updated by changing owner "Lee" in the first tuple of s into a different owner, say "Graham", although (s,τ_2) remains M-consistent, $< (r,\tau_1),(s,\tau_2) >$ is no longer EI-consistent. □

Let K be a chosen extended key, $X_{ij} = R_i \cap R_j - K$, and Q_K^i denote the set of K values derived by the mapping process wrt (r_i, τ_i).

Theorem 4.1 A $< (r_1,\tau_1),\ldots,(r_m,\tau_m) >$, where each (r_i,τ_i) is M-consistent, is EI-consistent if and only if, for every pair of i, j such that $i \neq j$ and $X_{ij} \neq \emptyset$, $(Q_K^i \bowtie r_i) \bowtie_{\theta(K|X_{ij})} (Q_K^j \bowtie r_j) = \emptyset$, where $\bowtie$ is the natural join. □

Incremental Detecting 4.1: Let $< (r_1,\tau_1),\ldots,(r_m,\tau_m) >$ be EI-consistent, we wish to test if an update to r_i or τ_i that preserves the M-consistency will also preserve the EI-consistency.

Insertion of tuple t into r_i. If $K \not\subseteq R_i^+$, Q_K^i is empty, so add t into r_i and stop. Assume that $K \subseteq R_i^+$. Expand t by mappings in τ_i. Let the result tuple be t' and contain constants on Z. If $K \not\subseteq Z$, add t into r_i and stop. Otherwise, send t' to all database sites j such that $K \subseteq R_j^+$ and $R_i \cap R_j - K \neq \emptyset$. At each site j receiving t': retrieve a tuple from Q_K^j by search key $t'[K_j]$, where K_j is the key for R_j. If no tuple is returned or if the returned tuple is not identical to $t'[K]$, do nothing. Assume that a tuple identical to $t'[K]$ is returned. Retrieve from r_j by search key $t'[K_j]$ the original tuple that derives the K value, say u. If $u[R_i \cap R_j - K] \neq t'[R_i \cap R_j - K]$ for any j, then reject t. If t is not rejected by any site j, add t to r_i and $t'[K]$ to Q_K^i.

Deletion of tuple t from r_i: Although the EI-consistency is not violated by the deletion, if $K \subseteq R_i^+$, all tuples in Q_k^i with value $t[K_i]$ on K_i should be deleted.

For update on mappings, we need the generalized extension join of [10].

Definition 4.3 Let $X \subseteq R_i^+$. An *extension join* covering X from R_i is $r_i \bowtie M(X_1 \rightarrow Y_1) \bowtie \ldots \bowtie M(X_k \rightarrow Y_k)$, where $< X_1 \rightarrow Y_1, \ldots, X_k \rightarrow Y_k >$ is a minimal derivation of X from R_i. □

Insertion of tuple t into τ_i: Assume that t is inserted into mapping M. If $K \not\subseteq R_i^+$, $Q_K^i = \emptyset$, so add t to τ_i and stop. Assume that $K \subseteq R_i^+$. Compute the union of projections onto K of all extension joins covering K from R_i that involve M, but with M replaced by $\{t\}$. Let the result be ΔK. In other words, ΔK contains the increment of Q_K^i due to insertion of t. Compute $r_i \bowtie \Delta K$ and let the result be $\Delta Expand$. That is, $\Delta Expand$ contains tuples in r_i expanded to K using the new tuple t. Send $\Delta Expand$ to all database sites j such that $K \subseteq R_j^+$ and $R_i \cap R_j - K \neq \emptyset$. Reject t if and only if $\Delta Expand \bowtie_{\theta(K|X_{ij})} (Q_K^j \bowtie r_j) \neq \emptyset$ for some site j, where $X_{ij} = R_i \cap R_j - K$. If t is not rejected, add t to M and ΔK to Q_K^i.

Deletion of tuple t from τ_i. Assume that t is deleted from mapping M. Remove t from M. Compute the union of projections onto R_i of all extension joins covering K from R_i that involve M, but with M replaced by $\{t\}$. Let the result be Δr_i. Δr_i is a superset of tuples in r_i whose contribution to Q_K^i are affected by the deletion. Some tuples in Δr_i may not use t in an "essential" way and their K values should not be deleted from Q_K^i. To find these tuples in Δr_i, compute the union of projections onto R_i of all extension joins covering K from R_i that do not involve M, but with r_i replaced by Δr_i. Let the result be *stay*. Then remove from Q_K^i all tuples that agree with some tuple in $\prod_{K_i}(\Delta r_i - stay)$ on K_i.

Efficiency of Incremental Detecting: For insertion into r_i, at the updating site i the number of B^+-tree retrievals performed is no more than the number of mapping schemas used in expanding t, and there are at most two B^+-tree insertions performed. At most one tuple is sent from site i to each remote site j. At each receiving site j, at most two retrievals on B^+-trees are performed. For deletion from r_i, the cost is at most two B^+-tree deletions.

Checking update of mappings could be expensive if there are many extension joins covering K. However, since these extension joins compute only new K values, they are expected to be cheaper than recomputing all old K values.

5 Experiments on A Case Study

We have conducted an experiment on a case study of three real life restaurant databases, with three objectives in mind: identify equivalent entities, incrementally check consistency, and confirm the performance of the detecting methods by comparing it with other methods. The databases are created by UNIX Ingres on IBM RS/6000 Model 560 machines. The performance is measured by the elapsed time on each database site (i.e., computation cost) and the number of tuples transmitted between database sites (i.e., communication cost).

Three relations, referred to as R, S, and T, from the databases are considered. The mapping schemas on R are $f_1, \ldots, f_4$, on S are $f_5, \ldots, f_7$ on S, and on T is f_8. The data in databases and knowledge in mappings are obtained from the Singapore yellow pages, local dining directories and street directories. Some restaurants are modeled in all three databases, and some are modeled only in one or two databases. Table 3 shows the structure and the number of tuples in these databases and mappings. A B^+-tree index is specified on the index key of each relation and mapping. The Ingres block size is 2K, with 44 Bytes of it reserved by the system, and the pointer size is 4 Bytes. With non-leaf nodes 70% full (the Ingres default value), the branching factor of B^+-trees is 25 for index keys of 50 Bytes. Therefore, for the data in Table 3, at most 4 block accesses are needed for each retrieval using dense indexing. Because of such a large branching factor, the depth of these B^+-trees grows very slowly as the size of relations or mappings increases.

We have conducted experiments on both M-consistency and EI-consistency. Due to space limitation, we report only the experimental results on EI-consistency.

The detecting of EI-consistency requires to construct and compute extension joins of the extended key. Two incremental strategies for detecting EI-consistency are compared.

1. the *Incremental Detecting 4.1.*
2. *Non-materialization* in which Q_K^i is not materialized. All deletions are free in this case. However, insertions of a tuple into r_i or τ_i requires the processing on r_i at the updating site and r_j at each remote site j.

Table 4 gives the average elapsed times and the number of tuples transmitted between databases for different types of updates. In Table 4, τ_A is the mapping assignments for mapping schemas of relation A. For each type the average elapsed time is taken over five randomly chosen tuples. The elapsed time is defined as "local time"+max{"remote time j"}, where the "local time" refers to the time spent on the updating site, and "remote time j" refers to the time spent on the remote site j. Since local operations are sequentialized with remote operations, the total time is the sum of these two times.

Table 4 indicates that Incremental Detecting 4.1 performs better than the non-materialization strategy for all insertions. If insertion is frequent, the 30% saving on insertion in Incremental Detecting 4.1 could be substantial. However, if deletion from mappings is frequent, the non-materialization strategy will be preferred. We also observe that in both strategies the amount of data transmission is quite small for all update cases except for insertion into τ_S. In general, at most two tuples will be transmitted for insertion of a tuple into any of relations R, S, T. Because of the large branching factor of B^+-trees, 25 in our case, the degrading of performance will be slow and gradual as the size of databases and mappings increases. These experiments show that the proposed framework has presented a practical solution to the inconsistency detecting problem in multidatabases.

6 Conclusion

The problem of detecting data inconsistency in multidatabases without common key is studied. We proposed a notion of data consistency based on additional knowledge about data and a method of incremental detecting of its violation during updates of user relations or the additional knowledge. The method makes use of materialized view and involves very low communication cost when the updates are on user relations. The incremental detecting for updates on knowledge is less efficient in general, but is affordable if it is infrequent. Our experiment on a case study of three real life databases has shown that the proposed method identifies equivalent entities and detects data inconsistency effectively and efficiently.

References

[1] R. Ahmed, P. DeSmedt, W. Du, B. Kent, M. Ketabchi, W. Litwin, A. Rafii, and M-C. Shan, "The pegasus heterogeneous multidatabase system", IEEE Computer, December 1991, pp. 19-27

[2] R. Agrawal, T. Imielinski, A. Swami, "Mining association rules between sets of items in large databases", ACM SIGMOD 1993, pp. 207-216

[3] A. Chatterjee and A. Segev, "Data manipulation in heterogeneous databases", SIGMOD Record, 20(4), December 1991, pp. 64-68

[4] S. Ceri, J. Widom, "Managing semantic heterogeneity with production rules and persistent queues", Conference of VLDB, 1993, Dublin, Ireland, pp. 108-119

[5] S. Ceri and J. Widom, "Deriving production rules for incremental view maintenance", Conference of VLDB, 1991, pp. 577-589

[6] U. Dayal, "Processing queries over generalization hierarchies in a multidatabase system", Conference of VLDB, 1983, pp. 342-353

[7] U. Dayal, H.-Y. Hwang, "View definition and generalization for database integration in a multidatabase system", IEEE Transactions on Software Engineering, SE-10, 11, 1984, pp. 628-645

relation or mapping	No. of tuples	record length (Byte)	index key (Byte)
R	9,808	70	name,area (50)
S	9,752	80	name (30)
T	9,761	105	name,cuisine (50)
$M(f_1)$	985	80	name,area (50)
$M(f_2)$	985	80	name,street (60)
$M(f_3)$	94	40	speciality (20)
$M(f_4)$	860	70	name,area (50)
$M(f_5)$	77	25	postcode (5)
$M(f_6)$	96	40	speciality (20)
$M(f_7)$	657	70	name,area (50)
$M(f_8)$	599	50	street (30)

Table 3: Size and index key of each initial table

updates	Incremental Detecting 4.1	Non-materialization	Tuples transmitted
insert a tuple into R	4.17	6.69	1
insert a tuple into S	3.97	6.78	2
insert a tuple into T	3.54	6.18	1
delete a tuple from R	0.62	free	0
delete a tuple from S	0.72	free	0
delete a tuple from T	0.63	free	0
insert a tuple into τ_R	28.45	33.17	5
insert a tuple into τ_S	49.39	65.8	131.6
insert a tuple into τ_T	24.85	25.63	11.2
delete a tuple from τ_R	23.33	free	0
delete a tuple from τ_S	88.59	free	0
delete a tuple from τ_T	19.61	free	0

Table 4: Average elapsed time for EI-consistency (in seconds)

[8] L.G. DeMichiel, "Resolving database incompatibility: an approach to performing relational operations over mismatched domains", IEEE Transactions on Knowledge and Data Engineering, 1(4), 1989, pp. 485-493

[9] A. Gupta and I.S. Mumick, "Maintenance of materialized views: problems, techniques, and applications", in IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2), June 1995

[10] P. Honeyman, "Extension joins", Conference of VLDB, Nontreal, 1980, pp. 239-244

[11] E. P. Lim, S. Prabhakar, J. Srivastava, J. Richardson, "Entity identification in database integration", IEEE Conference of Data Engineering, 1993, pp. 294-301

[12] M. Rusinkiewicz, A. Sheth, G. Karabatis, "Specifying interdatabase dependencies in a multidatabase environment", IEEE Computer, Vol. 24, No. 12, 1991, pp. 46-54

[13] Gregory Piatetsky-Shapiro, William J. Frawley, "Knowledge Discovery in Databases", AAAI Press/The MIT Press, 1991

[14] A. Segev and J. Park, "Updating distributed materialized views", IEEE Transactions on Knowledge and Data Engineering, 1(2): 173-184, June 1989