Inconsistent Data Repairs in Database Integrations

Bo Liu
Department of Computer Science
Jinan University
Guangzhou, China

Abstract—**In the operations of multiple database integrations, one of the most prominent data quality problems is the existence of inconsistencies. Due to independence of data management among different information sources, there exist several possibilities for repairs of integrated data. In order to get good repairs and preserve the correct data, this paper introduces a new algorithm for repairing inconsistent data among multiple tables based on functional dependencies, which keeps integrity of source databases without deletions of tuples. The algorithm determines the tuples that are potentially updated by analyzing related attribute statistic measures, and is of objective, accurate and effective properties. A repairing system structure is presented, and some comparisons with other related work are given at last.**

Keywords- integration; functional dependency; inconsistency; repair

I. INTRODUCTION

The main quality problems existed among several data sets include inconsistencies, duplicates, and so on. An integrated database is produced from multiple sources, even though each source has the same scheme and the same data constraints, by operations of the integration, violations of integrity constraints may emerge.

Current database management systems (e.g. Oracle) can define data constraints as a part of a database scheme, which prevents violations of constraints from going into the database However, some violations may happen in many ways[1]. For example, new constraints may be added without being checked for violations by legacy data; or, integrity constraints may be turned off temporarily for backup; or, constraints fail to hold after several databases have been merged.

Functional dependency (FD) is one kind of database integrity constraints. Let R be a relational scheme of an instance D, $X \subseteq R$, $Y \subseteq R$, an FD $X \rightarrow Y$ holds on D, denoted as $D \models X \rightarrow Y$, if for every two tuples t_1 and t_2 in D, such that $t_1[X]=t_2[X]$, $t_1[Y]=t_2[Y]$.

Example 1: Consider two instances in table 1 and table 2 from two database sources, which have the same scheme and satisfy a functional dependency (referred as fd :$X \rightarrow Y$) respectively. Tuples are labeled as $t_1, \ldots, t_8$.

TABLE I. INSTANCE D_1

	X	Y	Z
t_1	a_1	b_1	c_1
t_2	a_2	b_2	c_2
t_3	a_3	b_3	c_3
t_4	a_1	b_1	c_4

TABLE II. INSTANCE D_2

	X	Y	Z
t_5	a_1	b_3	c_5
t_6	a_2	b_4	c_6
t_7	a_4	b_4	c_7
t_8	a_5	b_5	c_7

Suppose an integration view: $v = D_1 \cup D_2$. Apparently, v violates fd: $X \rightarrow Y$. There are some repairs of v, such as:(1) delete t_5 from D_2; (2) update $t_5[Y]=b_1$ from D_2; (3) delete t_2 from D_1.

In order to solve quality problems, data repairs are usually carried out by insertions, deletions or modifications of tuples. Literature [2] proposes the concept of active integrity constraint (AIC), and insertion or deletion tuples should be given under specific constraint conditions. Jef Wijsen presents the method based on modifications, which replaces the errors with variables[3]. Literature [4] proposes a sampling method based on modifications with the minimum number of updated tuples. Literature [5] introduces the update-based repairing algorithms for satisfying FDs and INDs (Inclusion dependencies).

By analyzing several repairing approaches according to functional dependencies, it is easily to discover that the deletion-based method is comparatively simple, but part of useful information from the sources may be missing; and the update-based method ensures completeness of source information, but the choice of modified values is a difficulty problem. Literature [5] proves that computing the minimum-cost repair based on non-deletions is NP-complete. So current methods based on modifications have limitations in efficiency and accuracy.

The paper will study a repairing method based on FDs. The background of the work is similar to that of literature [5], which intends to repair violations according to FDs by modifications. However, the method proposed in [5] needs

The research is supported by Guangdong Natural Science Foundation under Grant No.S2012010008831 and State Key Laboratory of Software Engineering under Grant No. SKLSE2012-09-37.

some subjective parameters for computing and comparing cost of repairs, which effect accuracy and practicability. The paper introduces a new objective heuristic algorithm in the light of attribute statistic measures related to FDs.

The remainder of the paper is organized as follows. Section Ⅱ gives some related concepts. An algorithm for repairing inconsistent data that violate FDs is introduced in section Ⅲ. A universal repairing system structure is given in section Ⅳ. In section Ⅴ, comparisons with related work are studied. Finally, it concludes with general remarks on the work and the further direction for future research.

II. PRELIMINARIES

In general, given a set of FDs $\sum$ over a relation R, and two instances r and r′ of R, a repair of an inconsistent instance r is another instance r′ that satisfied $\sum$. Here only repairs obtained by modifications are considered, and the criterion of minimal changes is used. Some definitions of repairs are given as follows according to [4], where $\triangle$(r, r') is the difference set between instances r and r′, and |$\triangle$(r, r')| is the number of tuples in $\triangle$(r, r').

Definition 1. (Repair by modifications) Given a set of FDs $\sum$ over a relation R, and two instances r and r′ of R, r′ is a repair of r w.r.t $\sum$ if r′$\models\sum$ and TIDs(r′)= TIDs(r), where TIDs(r) is the set of all identifiers of tuples in r.

Definition 2. (Cardinality-Minimal Repair) A repair r′ of r is cardinality-minimal iff there is no repair r″ of r such that |$\triangle$(r, r'')| <|$\triangle$(r, r')|.

Definition 3. (Set-Minimal Repair) A repair r′ of r is set-minimal iff there is no repair r″ of r such that |$\triangle$(r, r'')|$\subset$|$\triangle$(r, r')| and for each attribute cell c$\in\triangle$(r, r''), r''(c)= r'(c).

In the paper, we use the concept of repairs in definition 2, and cardinality-minimal repair is referred as repair simply.

Definition 4. (Violation set) Given an FD fd over a relation R and an instance r, a violation set over fd on r is a set of culprits[5], denoted as Vio_FD (r, fd), where c $\in$ Vio_FD(r, fd), c$\subseteq$r, $\forall$ $t_1, t_2\in$c, t_1[X]=t_2[X]$\wedge t_1$[Y]$\neq t_2$[Y].

Example 2: In table 1 and table 2, given an FD fd: X→Y, and an integration view: v= $D_1\cup D_2$, according to definition 4, Vio_FD(v, fd)={{t_1, t_5},{t_4, t_5} ,{t_2, t_6}}.

In the following, definition 5 and definition 6 describe two confidence measures of tuples.

Definition 5. (tuple confidence) Given an FD fd: X→Y over a relation R, and an instance r, and a tuple t ∈ r, confidence of t over fd is

$$Conf(t, fd)=\frac{P(t[X],t[Y])}{P(t[X])} \quad (1)$$

where $P(t[X], t[Y])$ is frequency of $(t[X],t[Y])$ in r, and $P(t[X])$ is frequency of $(t[X])$ in r.

Definition 6. (tuple relative confidence) Given an FD fd: X→Y over a relation R, and an instance r, and a tuple t∈r, relative confidence of t over fd is

$$RConf(t, fd)=\frac{P(t[X],t[Y])}{P(t[Y])} \quad (2)$$

where $P(t[X], t[Y])$ is frequency of $(t[X], t[Y])$ in r, and $P(t[Y])$ is frequency of $(t[Y])$ in r.

Theory 1. Given an FD fd: X→Y over a relation R, and an instance r, for a tuple t∈r, if t$\notin$ Vio_FD(r, fd), then *Conf*(t, fd)=100%.

Prove: Let t′∈r, and t′[X]= t[X], because t$\notin$ Vio_FD (r,fd), then t′[Y]= t[Y], therefore

$$P(t[X], t[Y])= P(t[X])$$

$$Conf\ (t, fd) = \frac{P(t[X],t[Y])}{P(t[X])} = 100\,\%$$

However, *RConf*(*t*, *fd*) is probably not 100%. Because $P(t[X])$ may be less than $P(t[Y])$, in addition, $P(t[X], t[Y])\leqslant P(t[X])$ and $P(t[X], t[Y])\leqslant P(t[Y])$, so $RConf(t, fd)\leqslant$100%.

III. A REPAIRING ALGORITHM

A. Train of Thought

Given a violation set Vio_FD (r, fd), and fd: X→Y, for c∈ Vio_FD (r, fd), if there are n number of tuples in c, then n-1 number of tuples in c will be repaired. However, it is difficult to decide which tuples are modified to satisfy low cost and result in high accuracy. The main idea of the paper is selecting the tuples with the lower confidence to repair. In other words, firstly, choose the tuples with the highest confidence based on fd in each element c∈Vio_FD (r, fd), if there are more than one such tuples, then from them choose the one with the highest relative confidence based on fd, and let the tuple be t, denote Refvalue(c) as (t[X],t[Y]), such that inconsistent data of the other tuples in c which are different from t are repaired by Refvalue(c).

For example, in example 2, we get Vio_FD($D_1\cup D_2$, fd)={{t_1, t_5},{t_4,t_5} ,{t_2, t_6}}.

Let c_1={t_1, t_5}, c_2={t_4, t_5} , c_3={t_2, t_6}.

For c_1, due to *Conf*(t_1, fd)> *Conf*(t_5, fd), so Refvalue(c_1)=(a_1, b_1), and t_5[Y] is repaired by b_1, the same situation to c_2 .

For c_3, *Conf*(t_2, fd)= *Conf*(t_6, fd), but *RConf*(t_2, fd)> *RConf*(t_6, fd), so Refvalue(c_3)=(a_2, b_2), and t_6[Y] is repaired by b_2.

B. Repair Algorithm

The repair algorithm based on FDs is described in Fig.1. Some explanations are shown in the following.

Algorithm FD-REPAIR
Input: Database instance r, FD set F.
Output: Database repair D′
Method:
1. for each fd in F
2. if (Vio_FD (r, fd) is not empty) then
3. for each c in Vio_FD (r, fd)
4. *bestconf*={0, 0};
5. for each t∈c do
6. if (*Conf*(*t*, *fd*), *Rconf*(*t*, *fd*))>> *bestconf*) then
7. *bestconf*=(*Conf*(*t*, *fd*), *Rconf*(*t*, *fd*));
8. Refvalue(c)=(t[X], t[Y]); //fd: X→Y
9. end if;
10. end for;
11. Repair(c, Refvalue(c));
12. end for;
13. end if;
14. end for;
15. return D′.

Figure 1. FD-REPAIR Algorithm

In line 2, Vio_FD (r, fd) returns a violation set over fd on r;

In line 4, *bestconf* is initialized, which gives the minimal values of confidence and relative confidence;

In line 6 to line 8, use operator ">>" to compare (*Conf*(t, fd), *Rconf*(t, fd)) with current *bestconf*. If *Conf*(t, fd) is larger than the corresponding value in *bestconf*, or *Conf*(t, fd) is equal to the corresponding value in *bestconf* and *Rconf*(t, fd) is larger than the corresponding value in *bestconf*, then *bestconf* is replace by (*Conf*(t, fd), *Rconf*(t, fd)), successively, Refvalue(c) is given or updated;

In line 11, the tuples except t in c are repaired by Refvalue(c).

C. Time Complexity of FD-REPAIR Algorithm

In FD-REPAIR algorithm, let the number of tuples in r be n, the complexity of computing violation set is O(|F|n), the complexity of computing *bestconf* is O(|F|n^2), so time complexity of FD-REPAIR algorithm is O(|F|n^2), where |F| is the number of dependencies.

D. Implementation of the Main Procedures Based on SQL

The main procedures of repairing inconsistent data can make use of SQL standard language to implement. Suppose an fd: X→Y over instance r, then some procedures of FD-REPAIR algorithm by expanded SQL sentences are shown as follows.

(1) Function Vio_FD (r, fd)

```
Vio_FD=NULL;
for each tuple t in r
   insert into c
      select * from r as t1
        where t[X]= t1[X] and
              not (t[Y]= t1[Y]));
   Vio_FD= Vio_FD ∪ c;
end for;
```

In the above sentences, Vio_FD is the returned result, and c is a culprit of tuples.

(2)Compute confidence *Conf*(*t*, *fd*) and *RConf*(*t*, *fd*),where t∈c, c∈Vio_FD.

```
select count(*) as n1
   from r where r.X=t[X];
select count(*) as n2
   from r where r.Y=t[Y];
select count(*) as n
    from r where r.X=t[X] and r.Y=t[Y];
Conf(t, fd)=n/ n1;
RConf(t, fd)=n/ n2;
```

IV. A QUALITY REPAIRING SYSTEM STRUCTURE

Quality repairing is one of tasks in database integrations. It is based on source information systems, quality rules or data constraints and integration requirements. Fig.2 shows a quality repairing system, where the quality rule database stores FDs, and can be expanded to store other kinds of data constraints or quality rules, such as inclusion dependencies, domain rules, etc. Violation detection is implemented before data repair.

The quality repairing system is universal, independent, and extendable. It can be embedded into an integration application, and the source applications are not effected by it. The quality rule database can also be modified.

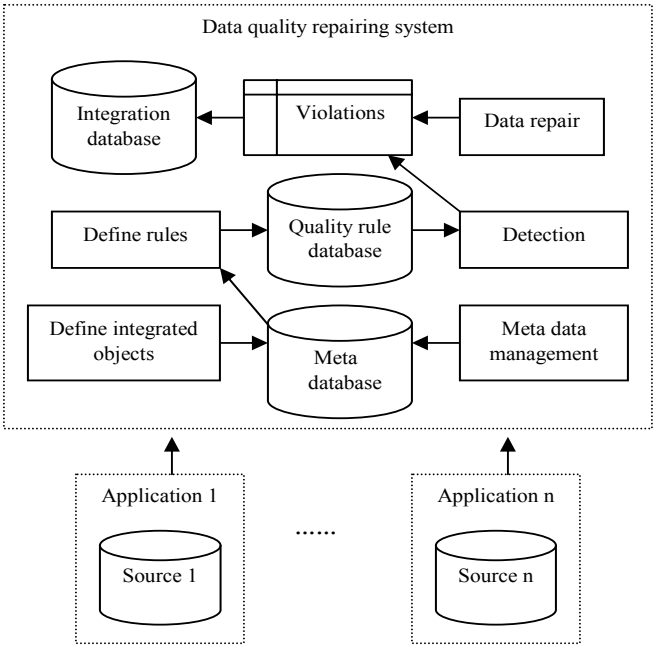

Figure 2. Structure of A Quality Repairing System

V. COMPARISON WITH RELATED WORK

There are some related research work introduced in section Ⅰ, which repair inconsistencies by modification methods. Especially, the work in literature [5] is based on FDs and INDs (Inclusion Dependencies). Although our work doesn't study repairing methods in respect to INDs, it can be compared with the work in literature [5]. Firstly, two points of differences between the work in literature[5] and our work are given in the following.

(1)Different criterion of choosing repaired objects

Literature [5] introduces a heuristic method based on the minimal repairing cost. The cost of repairing a tuple is evaluated by equation (3), where *inscost* is the cost of inserting tuple t (for INDs-based repairing), and dis(D(t, A), D′(t, A)) is the difference between the old value of attribute A and the new value of attribute A for tuple t, w(t) is confident weight of tuple t defined by users.

$$\text{cost(t)}=\begin{cases} inscost(R_i) & if \quad t \in new(R_i) \\ w(t)\cdot\sum_{A\in Attr(R_i)} dis(D(t,A),D'(t,A)), & otherwise \end{cases} \quad (3)$$

There is a big disadvantage over the equation (3), the cost of evaluating a tuple is determined by a set of subjective parameter weights given by users and a similarity computation method chosen by users. In contrast, computing cost of repairing a tuple and setting parameters are unnecessary in our method. Making use of the objective confidence measures which are simple and accurate.

(2) Different time complexity

Time complexity of GEN-REPAIR algorithm presented in [5] is $O(|C|^2n^3meg)$, where |C| is the number of dependencies, n is the number of tuples of a database instance, meg is the number of equivalent classes. Apparently, time complexity of FD-REPAIR presented in section Ⅲ is lower.

The main ideas of other related work are also different from ours. The follows give more details about them.

The method proposed in literature [2] requires repairing rules with explicit operations, including insertions, deletions and updates; i.e., based on a special form of integrity constraints, called active integrity constraint (AIC), whose body consists of a conjunction of literals which should be false and the head contains the actions which have to be performed if the body is true (i.e., the constraint is violated). Although the AICs can lead to fewer repairs to be considered, it is difficult to define an explicit repairing action in an AIC.

Jef Wijsen presents the method which uses variables to substitute error data according to constraints and constructed a core table containing all possible repairs[3]. For full dependencies and conjunctive queries, all updating repairs can be summarized into a single tableau *G* such that the consistent answer to any conjunctive query can be obtained by executing the query on *G*. The tableau *G*, called nucleus, will be homomorphic to all updating repairs and will be maximal in the sense that any other tableau that is homomorphic to all updating repairs, is also homomorphic to *G*. The approach is only suitable for consistent queries.

Literature [4] proposes a sampling method, which generates a random sample of cardinality-set-minimal repairs. In order to avoid repairing a tuple that violates one FD may introduce a new violation of another FD, the presented algorithm performs the repair of one cell at a time, rather than one tuple at a time. As to efficiency challenge, it introduces a mechanism that partitions the input instance into blocks that can be repaired independently. However, the algorithm can't be sure to get good results.

Solmaz Kolahi and Laks V.S Lakshmanan[6] introduce an approximate algorithm that for a fixed set of functional dependencies and an arbitrary input inconsistent database, produces a repair whose distance to the database is within a constant factor of the optimum repair distance. It gives V-repairs which are databases that contain variables representing incomplete information. A V-repair reflects two types of changes made to the original database to resolve functional dependency violations: changing a constant to another constant whenever there is enough information for doing so, and changing a constant to a variable whenever a constant can't be suggested for an incorrect value. V-repairs are homomorphic to a consistent relation, but they are not necessarily homomorphic to the original relation.

In a whole, our method is comparatively simple and effective in the practical applications.

VI. EXPERIMENTAL STUDY

A synthetic dataset BankCust(Bank-id, ID-number, LoanID, Amount, Province, City) is created for checking efficiency and repairing results of FD-REPAIR, and 20000 tuples are produced, in which error rate is 1‰. The algorithm was implemented in PL/SQL with Oracle DBMS, and run on a PC with Intel Pentium Dual CPU 1.6GHz, 2GB RAM and Window XP. The FDs are as following:

(BankID, CustID)→LoanID;

City→Province;

LoanID→Amount.

The results in Fig.3 shows that as the dataset size changes, running time is almost increased linearly. In addition, all error data are repaired correctly, because errors in the instance are in low confidence. These demonstrate the accuracy and efficiency of our method.

VII. CONCLUSION

From description and analysis of the above, it can be concluded that our work is simple, valid and efficient. The main contributions of the research include the following two aspects:

1) The paper proposes an algorithm of repairing violations, which depends on objective evaluation measures (ie., confidence, relative confidence) of tuples in the database

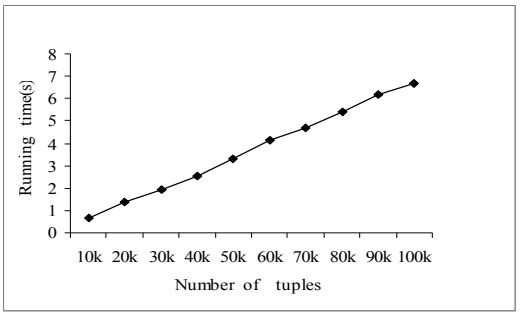

Figure 3. Running time of DQR-REPAIR

instance, repairing a violation set by the tuple with the highest confidence.

2) The paper gives some simple forms of extended SQL query sentences to support efficient violation detections and conference computations of tuples.

The work will be expanded to repair databases which violate the other kinds of constraints, such as CFDs(Conditional Functional Dependencies) [7], INDs and so on, in the future.

REFERENCES

[1] Hendrik Decker, Davide Martinenghi, "Inconsistency-tolerant integrity checking," IEEE Transactions on Knowledge and Data Engineering, vol. 23, pp. 218-234, 2011.

[2] Luciano Caroprese, Segio Greco, "Active integrity constraints for database consistency maintenance," IEEE transaction on Knowledge and Data Engineering. vol. 21, pp .1042-1058, 2009.

[3] Jef Wijsen, "Database repairing using updates," ACM Transactions on Database Systems, vol. 30, pp .722-768, 2005.

[4] George Beskales, Ihab F. Ilyas,Lukasz Golab, "Sampling the repairs of functional dependency violations under hard constraints," Proceedings of the VLDB Endowment, vol. 3, pp. 197-207, 2010.

[5] Philip Bohannon, Wenfei Fan, Michael Flaster, Rajeev Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," SIGMOD 2005, pp. 143-154, 2005.

[6] Solmaz Kolahi, Laks V.S Lakshmanan, "On approximating optimum repairs for functional dependency violations," Proceedings of the 12th International Conference on Database Theory, pp. 53-62, 2009.

[7] Wenfei Fan, Floris Geerts, Xibei Jia, Anastasios Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," ACM Transactions on Database Systems, vol. 33, pp. 444–491, 2008.