Query Rewriting Algorithms for Computing Credible Query Answers over Annotated Inconsistent Database

Aihua Wu
Dept. C.S. of Shanghai Maritime University, Shanghai, China
Email: 061021058@fudan.edu.cn

Abstract—Managing and querying inconsistent database is a challenge problem: approaches of picking sure part or selecting one from the conflicting tuples result in information lose, while methods of computing all possible query answers can be meaningless because of the little probability of each possible query answer. We present an approach named Annotation Based Query Answer over Inconsistent Database which tries to calculate proper answer by distinguishing inconsistent data from consistent ones in the answer with annotations. It can correctly tell user inconsistency of query result down to attribute level when only functional dependency is considered. In this approach, information is preserved while query answer is one single. In this paper, we propose a method of query rewriting to compute Annotation Based Query Answer for any given SQL query without aggregation function and correlated sub query. Through the query rewriting, this approach doesn’t require a new query language and can be easily embedded into existing database applications. Except for the information preserving, the experimental results both on TPC-H database and synthesized database show the effectiveness and applicability of our approach.

Index Terms—data quality; inconsistency; uncertain data; certain query answer

I. INTRODUCTION

Although integrity constraints are adopted to guarantee consistency of data for long time, inconsistent data still exists in wide range applications from data integration [1], data exchange, data cleaning, information retrieval [2], to sensor networks [3]. Uncertainty implied in query answer over inconsistent database makes it incredible. And computing proper query answer over them is tougher than over conventional databases, even only constraint of functional dependency is violated. Major challenges include finding proper semantics for their query answers, developing efficient query evaluation algorithms, and preserving as much information as we can in the query results.

Inconsistent database is considered to be correspond to a set of deterministic database, and so do query answers over them. Although from the user’s perspective, a single sure query answer would be desirable in most cases. The probabilistic nature of inconsistent data makes it difficult to find such query answer. Approaches of data cleaning with insert or update [9] are limited by accuracy and human intervener, approaches of data cleaning with delete and that of consistent query answer [5] result in information loss.

On the other hand, instead of a single sure query answer, it would be significant if all inconsistent data of the query answer are marked out. User can learn which part of the query answer is credible and which is not, or even deduce the true value of incredible ones.

We present a weak representation with annotation for inconsistent relational database that may violate a set of functional dependencies (FDs for short below) but have only one candidate key in [6]. In this representation, inconsistent attribute values in both data source and query results are attached with annotations. We call such relation Annotationed Relation, such database Annotationed Database, and the query answer Annotation Based Query Answer (AQA for short below). The approach can avoid information loss.

For a given Annotationed Database and a query over it, can AQA be figured out in way of evaluating SQL queries in current DBMS? No. To give a formally solution, seven basic algebra operations are defined in [6]: selection, selection with domain equality, projection, join, join with domain equality, union and difference. Queries can be represented as these operations or their combination. Soundness and completeness of the approach are proved in [6].

But for any SQL query, how to compute its AQA? A strategy is to extend or rewrite the query evaluation module of current DBMS [17,19]. But the modified DBMS can not efficiently manage database managed by commercial DBMS. Therefore, it needs to develop a middleware which accepts user’s SQL queries, translates it into one or a set of SQL queries and returns AQA.
In this paper, we propose rewritten algorithms to calculate AQA. The main advantage of our method is its supporting of attribute-level inconsistency. Furthermore, our approach doesn’t require a new query language and can be easily embedded into existing database applications. Still more, our approach can deal with databases from different DBMS.

Contributions. The main contributions include:

We present algorithms for rewriting SQL queries. Except for creating Annotated Database, the approach doesn’t need neither pre- & post-processing nor modification of current database system. This enables the technique applicable to databases in many applications. Further more, it almost doesn’t change the original database and lose no information.

We present rules for calculating valid FDs on the query result for any given query over any given database schema.

We present a performance study using both data and queries of the TPC-H benchmark and those generated by our data generator. We compare performance of the approach against database with different degrees of inconsistency and in different scale to show its adaptability.

We present an optimization technique so that it is practical for join queries between many large tables.

Organization. Section 2 is related work, section 3 briefly introduces the annotation based data model and outline main idea of approach of AQA discussed in [6]. Section 4 presents algorithm for rewriting Select-Project-Join queries, and followed by algorithms for union and difference queries in section 5. Section 6 states experimental evaluation. And last is the conclusion.

II. RELATED WORK

Problems of computing “clean” or credible query answers on inconsistent, incomplete and uncertain database have received renewed attention in the last few years. Generally, there are three strategies to solve this problem: data cleaning [7-9], consistent query answer (CQA) [5,10,11] and probabilistic databases [1, 12-15]. Data cleaning focus on algorithms to correct data errors so that “clean” answer can be evaluated against “clean” data source. It is useful in many applications, but it usually requires user’s interference, and no algorithm can assure 100% correctness when insertion or modification is used. CQA tries to compute consistent query answer without modification of inconsistent data source. Here consistent query answer is defined as the common part of answers to the query on all repairs [5]. It avoids correcting inconsistent data, but produces sure query answers.

Both approaches of data cleaning with deletion and CQA are unavoidable of Information loss. The former loses tuples with inconsistent attribute, even they are consistent on all attributes of the query answer. While the latter ignores tuples who are inconsistent on one attribute of the query result, even its other attributes are credible. Our approach doesn’t modify or filter data, but add an extra annotation dimension for each attribute value. It loses nothing.

Information loss doesn’t exist in methods based on probabilistic database, too. However, possible answers can be exponentially large in size and the probability associated with each single answer is extremely small. Furthermore, the techniques view that the probability of each attribute value is equal to the probability of the whole tuple. But in fact, those attribute are different in reliability. Techniques of probabilistic database aim at likelihood of each query answer, but our goal is maximum consistent data in the query answer.

III. ANNOTATION BASED QUERY ANSWER OVER INCONSISTENT DATABASE

We present the framework of approach stated in [6], and related basic concept in this section. It defines inconsistent database as those that violates any of its integrity constraints. And it supposes that the database only violates FDs and all determine attributes are creditable. Determine attributes are those that appear as left side of a FD.
Figure 1. Two annotated relation of database Student

<table>
<thead>
<tr>
<th>R</th>
<th>CName</th>
<th>CNameA</th>
<th>Phone</th>
<th>PhoneA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ar#05</td>
<td>38215420</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ar#05</td>
<td>65466460</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ar#01</td>
<td>6881234</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An inaccurate query answer of Q1

<table>
<thead>
<tr>
<th>R*</th>
<th>CName</th>
<th>CNameA</th>
<th>Phone</th>
<th>PhoneA</th>
<th>Major</th>
<th>MajorA</th>
<th>Tutor</th>
<th>TutorA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ar#05</td>
<td>38215420</td>
<td>Class</td>
<td>Name</td>
<td>Art</td>
<td>Life</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ar#05</td>
<td>65466460</td>
<td>Class</td>
<td>Name</td>
<td>Art</td>
<td>Kiwi</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ar#01</td>
<td>6881234</td>
<td>Art</td>
<td>x</td>
<td>Bobby</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ar#01</td>
<td>6881234</td>
<td>Magic</td>
<td>x</td>
<td>Bobby</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#our annotation based query answer of Q1

Figure 2. Query answers of Q1 over Student
3.1 Data Model

Inconsistency is a property of data, and can be described. We extend relation data model by adding a description dimension: for each attribute $X$, attribute $X_A$ are added to record inconsistency of each tuple on $X$, e.g in annotated Class of figure 1, $t_5(Major)$ conflicts with $t_6(Major)$ according to $CName->Major$, so annotation $\ldots^*$ is assigned to $t_5(MajorA)$ and $t_6(MajorA)$.

For any given relation and its FD set, it is easy to judge inconsistency of each attribute value. But if nothing changed, annotations in the query result can't correctly denote inconsistency of its corresponding attribute value. For example, if we apply query Q1 over annotated relations shown in figure 1 and simply rewrite it as Q1', annotations can be wrongly returned back. As shown in figure 2(a), cell phones of two tutors of class Art05 are inconsistent. The FD $CName->Phone$ that they violate is not valid on input tables but valid on the query result. Tuples in the query result should be verified again for the "new born" FD.

\[Q1: \text{select } CName, \text{ Phone} \]
\[\text{From Class, Teacher} \]
\[\text{where class.Tutor=} \text{Teacher.Tname and major=} \text{Art'} \]
\[Q1': \text{select } CName, \text{ CNameA, Phone, PhoneA} \]
\[\text{from Class, Teacher} \]
\[\text{where class.Tutor=} \text{Teacher.Tname and major=} \text{Art'} \]

To recognize cell values who are consistent in input table but inconsistent in query result, we use determine attribute of the "new born" FD it violates as the annotation. For example, in figure 2(b), "Class.CName" table but inconsistent in query result, we use inconsistency w.r.t. is assigned to the first two tuples of $R'$, denoting their annotation. For example, in figure 2(b), "Class.CName" table but inconsistent in query result, we use inconsistency w.r.t. is assigned to the first two tuples of $R'$, denoting their annotation. For example, in figure 2(b), "Class.CName" table but inconsistent in query result, we use inconsistency w.r.t. is assigned to the first two tuples of $R'$, denoting their annotation. For example, in figure 2(b), "Class.CName" table but inconsistent in query result, we use inconsistency w.r.t.

\[ Definition1 \text{ uncertain data: Given a relation } R, \text{ a set of FD } \psi \text{ on it, } \forall(x \rightarrow y) \in \psi, \forall t_1, t_2 \in R, \text{ if } t_1[X]=t_2[X] \text{ and } t_1[Y]=t_2[Y], \text{ we say that piece of data } t_1[Y] \text{ and } t_2[Y] \text{ is uncertain data w.r.t. } x \rightarrow y. \]

\[ Definition2 \text{ Annotated Relation: Given a relation } R \text{ and its FD sets } F, \text{ if all uncertain pieces of data of } R \text{ w.r.t. } F \text{ are attached with one or more marks, } R \text{ is an annotated relation w.r.t. } F. \text{ Similarly, for any query } Q \text{ over } R, \text{ if all uncertain pieces of data of } Q(R) \text{ are attached by marks, } Q(R) \text{ is Annotation Based Query Answer.} \]

In annotated relation, certain piece of data has no mark with it while uncertain piece of data can have one or more marks with it. There are two types of annotation: mark \(*^*\) and determine attribute name. We call the former static mark and the latter dynamic mark. Static mark can't be changed, while dynamic marks can be attached to or eliminated from the data after another query expression.

3.2 Derived Functional Dependency

Data in the query result are assigned with dynamic mark because they violate derived FD. And derived FDs can be implied with domain equality between attributes.

\[ Definition3 \text{ domain equality (DEQ): Given database schema } D, \text{ domain equality statement } X=D \text{ is true iff for any instance of } D \text{ and any tuple } t \text{ of the instance, } \exists t', \text{ that } t[X]=t'[Y], \text{ here } t \text{ and } t' \text{ can be same tuple.} \]

\[ Definition 4. \text{ Given a query } Q \text{ and a set of FD } F, \text{ suppose } U_1,U_2,...,U_n \text{ are attributes appear in select, where, group by, having, order by of } Q, \text{ projection of } F \text{ on } Q \text{ is project of } F \text{ on } R(U_1,U_2,...,U_n). \text{ Similarly, projection of domain equality DEQ on } Q \text{ is projection of DEQ on } R(U_1,U_2,...,U_n). \]

The next rules can be used to compute DEQ for any given query expression over relational database.

Let $s$ be a schema, let $e$ be an expression over $s$. The derivation rules producing new domain equalities on $e$ are as follows (where "$|-'$" means "derives") (based on [20]):

1. $X=Y |- Y=X$
2. $X=Y, Y=D |- X=D$
3. $|- X=D$
4. $X=Y |- X=Y$
5. $Z->A_1, Z->A_2, \forall t_1,t_2 \text{ if } t_1[Z]=t_2[Z], t_1[A_1]=t_2[A_2] |- A_1=A_2$

Let $s$ be a database schema, $F$ be FD set in $s$ and $e$ a query expression over $s$. The set Drv($e$) of derivable constraints on $e$ is defined by the following rules which use induction on operations in $e$ when only domain equality considered (based on [20]).

1. \[ \text{Drv}(R): \text{Picture each of } R \text{ FDs as FD tree [20], then take the closure.} \]
2. \[ \text{Drv}(e[X])(\text{projection): Take all DEQs } Y=D \text{ where } X[Y]=X[Z] \text{ is in Drv(e), and all FDs } Z->A \text{ that } X[Z]->X[A] \text{ is in Drv(e).} \]
3. \[ \text{Drv}(e[X= Y])(\text{selection with domain equality): Add } D \text{ to } X=\text{ Drv(e) and take the closure.} \]
4. \[ \text{Drv}(e1 \triangleleft e2)(\text{join): Rename the constraints in Drv(e2) according to the degree of } e1, \text{ i.e. a DEQ } X=Y \text{ becomes } X+k=Y+k (k=\text{degree(e1)}) \text{ and an FD } Z->A \text{ becomes } Z+k->A+k. \text{ Then add renamed Drv(e2) to Drv(e1) and take the closure.} \]
5. \[ \text{Drv}(e1 \cup e2)(\text{union): A DEQ } X=Y \text{ is in Drv(e1) and Drv(e2) if it is in both Drv(e1) and Drv(e2). If } Z->A \text{ is in Drv(e1) and } Z->A \text{ is in Drv(e2) , } e1.Z \text{ is domain equal to } e2.Z \text{ and } e1.A \text{ is domain equal to } e2.A, Z->A \text{ in Drv(e1 u e2).} \]
6. \[ \text{Drv}(e1\setminus e2)(\text{difference): Use Drv(e1).} \]

From the above, it can be proved that Drv($e$) is projection of $F^+$ on schema of $e$. Here we call those FD Derived FD which does not belong to input FD set $F$ but belong to $F^+$ according to given DEQs, denoted as Drvd($F^+$,DEQs). Derived FDs can be computed by the following method:

1. Replace every determined attribute of $F$ with its domain equal attribute and add the new FD to FD set.
2. Replace every determine attribute of $F$ with its domain equal attribute and add the new FD to FD set.
3. For any functional $A->B$, $C->D$, if $B$ is domain equal to $C$, add $A->C$, $A->D$, $B->D$ to the FD set.
4. Repeated 3) until the FD set unchanged.
5. Remove duplicate FDs and input FDs . The left are
Derived FDs.

3.3 Annotation Based Query Answer

For a given database $D$ and query $Q$, suppose that all derived FD on $Q(D)$ is known. $AQQA$ is the evaluation result of $Q$ over correctly dynamic marked $D$ by verifying it w.r.t. all derived FD. The next are examples of $AQQAs$.

$$Q_{2}(\text{Student})$$

<table>
<thead>
<tr>
<th>SID</th>
<th>SIDA</th>
<th>SName</th>
<th>SNameA</th>
<th>Age</th>
<th>AgeA</th>
<th>Class</th>
<th>ClassA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>Jack</td>
<td>18</td>
<td></td>
<td></td>
<td>Art05</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Elindi</td>
<td>21</td>
<td>*</td>
<td>Art081</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Elindi</td>
<td>20</td>
<td>*</td>
<td>Art081</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. AQA of Q2.

$Q_{2}$: select * from Student where Age<=18

$Q_{3}$: select CName, Tutor from Class

$Q_{4}$: select Phone from R where CName='Art05'

In [6], we present evaluation rules for any algebra query, which are proved to be sound and valid. Based on those rules, the problem we try to solve in this paper is how to compute $AQQA$ by query rewriting for given SQL queries when valid set of FD on the query result is known and the base database is annotated.

IV. SPJ QUERIES

In this section, we present rewriting strategy for SPJ queries without aggregation or grouping. We illustrate the rewriting strategy with the next examples with DEQ $Tutor=Tname$.

Example1: let’s start with a simplest query which asks for all classes.

$Q_{5}$: select * from Class

Rewritten query of $Q5$ is $Q5^\prime$. Notice that * in $Q5$ and * in $Q5^\prime$ denote to different set of attributes, the latter includes all attributes of annotations. FDs are not checked on the query result, because no Derived FD exists here.

Example2: consider a query which retrieve all class whose major is “Art”.

$Q_{6}$: select CName, Tutor from Class where Major=’Art’

$Q_{6}^\prime$: select CName, CNameA, Tutor, TutorA from Class where Major=’Art’

$Q_{6}^\prime$"select CName, CNameA, Tutor, TutorA, Major, MajorA from Class

where Major=’Art’ or CName in (select CName from Class where Major=’Art’ and MajorA is not null)

Naturally, AQA for $Q6$ is thought as {('', 'Lee', '*'), ('Art05', '', 'Kim'), ('Art081', '')} which can be obtained by evaluating $Q6^\prime$. Notice that Major of $t5$ is actually unknown. If all possible classes are considered, $t5$ should also be included. While if only exact classes are considered, $t5$ should be excluded. Here we take the narrow semantic of incomplete database that classes satisfy $Q6$ can only be those whose major is “Art” or those who conflict with a class whose major is “Art”. Furthermore, attribute Major and MajorA are also returned so that user can know inconsistency of records on condition attributes.

Now, let’s discuss rewrite strategy of join queries. Derived FDs are usually implied in join result. Thus, we need to recheck inconsistency of the join result according to the derived FDs. Furthermore, as for tuples who are inconsistent on join attributes, they will join with those tuples who satisfy join condition with value of himself or of his conflicting values. In evaluation of $Q1$, $t3$ will join with $t10$ and $t11$, and $t4$ will also join with $t10$ and $t11$.

An optimization technique can be used to reduce unnecessary inconsistent checking and dynamic marking: departing the data source into two parts according to its possibility of violating Derived FD, computing dynamic annotations for the former, calculating query answer with both of them and returning the union query result.

Example3 gives the rewritten query of $Q1$ through 3 steps. Firstly, it joins tuples who share same determine attribute value with other tuples because they may violate a Derived FD. Notice that join condition is modified from $Tname=Tutor$ to $Tname$ equal to any $Tutor$ in the conflicting Tutor set of the Class. Secondly, it checks derived FD and attaches dynamic annotation to the temp table. Thirdly, it apply query condition on each part of data source and union them together.

Example3: Rewritten query of $Q1$ is as follows

1) Select C.*, T.* Into tmpR1
   From Class C, Teacher T
   Where (Tname = any (select Tutor from Class C2 where Cname=C.CName))
   and CName in (select CName from Class group by CName having count(*) >1);

2) update T.
   where exists (select B.CName from tmpR1 B
   where T.CName=B.CName
   group by B.CName.
   having count(distinct B.Phone) >1);

   select CName, CNameA, Phone, PhoneA,
   Major, MajorA, Tutor, TutorA
   from tmpR1
   where major=’Art’ or CName in (select CName From Class
   Where Major=’Art’ and MajorA is not null))

   Union
   select CName, CNameA, Phone, PhoneA,
   Major, MajorA, Tutor, TutorA
   from ((select * from Class C1 where CName in (select CName from Class group by CName

© 2013 ACADEMY PUBLISHER
having count(*) =1) C, Teacher T
where Tutor=Tname and (major=‘Art’ or CName in ( select CName From Class
Where Major=‘Art’ and MajorA is not null));
Example4: Irrelevant sub query in Q7 is rewritten to return all possible TName.
Q7: select Cname, Major from class
where Tutor in ( select name from Tutor Where city=‘Brea’)
Q7′: select Cname, CNameA, Major, MajorA, Tutor, TutorA from class
where Tutor in (Where city=‘Brea’ or name in ( select name From teacher
Where city=’Brea’ and cityA is not null));

The next is our algorithm for rewriting SPJ queries. It first rewrite all \( a \theta v \) (v is not attribute, \( \theta \) is predicate and \( a \) is determined attribute) in where clause so that tuples who are uncertain on condition attributes can be return back. In the rewritten query of \( a \theta v \), L1,L2,…are all determine attributes that Li->a is valid on some relation Ri in from clause. Secondly, if no derived FDs are valid on the query, it will be directly rewritten to return both value and annotation of attributes not only in select but also in where. If derived FDs are valid on the query, normally there is more than one table in from clause, it will be translated into a series queries in three levels: query to create table for records which may violate Derived FD, queries to update annotations w.r.t Derived FD, and query to retrieve records with annotations.

The first level of query joins the tables on rewritten join condition so that tuples who are inconsistent on join attributes can be joined correctly. After join, a super relation who includes all required attributes can be build, and annotations can be updated over it. By the rewritten join conditions, tuple from Ri will be joined not only with tuples from Rj who satisfy original join condition but also tuples who conflict with those in Rj on join attribute. In the algorithm, if \( \theta \) is predicate \( \omega,1 \), here \( \omega = \text{NULL} \) in with S;}\n
Or cname in ( select cName From class
where TutorA is not null and Tutor in ( select name From teacher
where city=’Brea’ and cityA is not null));

Suppose \( W_1,W_2,\ldots,W_n \) are attributes that appear in \( \omega \) but not in \{A1,A2,\ldots,An\}. Output: AQA of Q or rewritten query \( \varphi \)

1. Rewrite \( \omega \) to \( \varphi \) according to the next rule:
   a) For each \( a \theta v \in \omega \) (v isn’t attribute and \( \theta \) is predicate and \( a \) is determined attribute).
   Replace \( a \theta v \) with ( \( a \theta v \) or L1 in (select L1 from Ri Where \( a \theta A \) and \( a \) is not null) or L2 in (select L2 from Rj Where \( a \theta A \) and \( a \) is not null) )
   b) For each sub query \( \delta \) in \( \omega \)
   Replace \( \delta \) with SPJ( \( \Omega' , \Psi' , \delta , 1 \) ), here \( \Omega' \) and \( \Psi' \) are projection of \( \Omega \) and \( \Psi \) on \( \delta \) respectively.
2. Set \( \Psi' = \text{Projection of } \Psi \text{ on } Q \)
3. If \( \Psi' = \text{NULL} \)
   If (I=0)
     Execute the next query and return the query answer:
       select A1,A2,\ldots,An, A1A, A2A,\ldots,AnA, W1, W1A,\ldots,Wm,WmA
       From R1,R2,\ldots,Rk Where \( \varphi \) Order by \( \kappa \)
   else
     Depart \( \Psi' \) into two part: \( \Psi'_1 \) which includes all \( Ri.A0Rj.B \) where Ri and Rj \( \in \{R1,\ldots,Rk\} \), A and B are attributes, \( \theta \) is predicate, and \( \delta \) of the left.
   c) For each \( Ri.A0Rj.B \) in \( \Psi'_1 \)

   { S=""; 
   Suppose B-related FDs are \( B1->B,\ldots,Bn->B \) and A-related FDs are \( A1->A,\ldots,An->A \)
   if \( Ri.B \) is determined attribute
   \( S=Ri.A \theta any \) (select B from Ri T
   where T.B1=Rj.B1 and \ldots and T.Bn=Rj.Bn)
   If Ri.A is determined attribute
   \( S=""; \)
   \( S=Ri.B \theta any \) (select A from Ri T
   where T.A1=Ri.A1 and \ldots and T.An=Ri.An)
   Else
   \( S=\lor \) \( Ri.B \theta any \) (select A from Ri T
   where T.A1=Ri.A1 and \ldots and T.An=Ri.An)
   Replace \( Ri.A0Rj.B \) with \( S; \)
   d) Execute the next query where \( Ei \in \Omega' \), and \( fdli \) is determine attribute of a FD in \( \Psi' \);
       Select R1.*, R2.* into tmpR From R1,R2,\ldots,Rk
       Where \( \varphi_1 \) and fdli in (select fdli from Ri group by fdli
       having count(distinct fdri)>1)
       and fdli2 in (select fdli2 from Rj group by fdli2 having count(distinct fdri2)>1)
   e) For each FD \( fdli->fdri \) in \( \Psi' \), execute the next query update T set T.fdriA=T.fdriA+ ‘fdli’ from tmpR T
   where exists (select b.fdli from tmpR b
   where T.fdli=b.fdli group by b.fdli
   having count(distinct b.fdli)>1)
   f) If (I=0)
     Execute the next query and return the query answer:
V. UNION AND DIFFERENCE QUERIES

In this section, we will present the rewriting algorithms for union and difference queries.

<table>
<thead>
<tr>
<th>CName</th>
<th>CNameA</th>
<th>Major</th>
<th>MajorA</th>
<th>Tutor</th>
<th>TutorA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSD881</td>
<td>CSE</td>
<td>Judy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0801</td>
<td>EE</td>
<td>Nancy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1T08</td>
<td>CIT</td>
<td>David</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N0801</td>
<td>NE</td>
<td>David</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FDs  \( \text{Cname} \rightarrow \text{Major, Teacher} \)

Class2

<table>
<thead>
<tr>
<th>CName</th>
<th>CNameA</th>
<th>Major</th>
<th>MajorA</th>
<th>Tutor</th>
<th>TutorA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSD881</td>
<td>CSE</td>
<td>Nancy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0801</td>
<td>EE</td>
<td>Nancy</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1T08</td>
<td>CIT</td>
<td>Philip</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Example of Union over Annotated table

Example 5: considering query Q8 and Q9 over inconsistent tables Class1 and Class2 in Figure 6.

Q8: select * from Class1
Union
Select * from Class2
Q9: select Cname, major from class1
where Tutor = 'Nancy'
select Cname, major from class2
where major= 'EE'

As for Q8, according to traditional semantic of Union, the query result should be \{t1,t2,t3,t4,t5,t6,t7\}. But notice that: 1) \( t1[Tutor] \) and \( t5[Tutor] \) are inconsistent in the query result, they should be annotated, and 2) Although \( t2 \) is consistent in Class1, it should be annotated for conflicting with \( t7 \) on attribute Tutor, and after annotating, it should be removed because it is completely equal to \( t6 \).

Now, let's look into query Q9, of course, \( t2 \) is the only record in the result, but notice that \( t7 \) in Class2 implies another version of class 'Art05', which conflict with \( t2 \) on Tutor. Therefore, \( t2[Tutor] \) should be marked with "*".

According to the above discussion, we present our algorithm to compute AQA for query \( R_1 \cup R_2 \). It first extends the query condition so that all possible tuples will be involved in. Then a series of rewritten query are executed to compute AQA: 1) queries to get tuples who satisfy query condition from \( R_1 \) and \( R_2 \) into \( tmpR1 \) and \( tmpR2 \) respectively; 2) queries to verify consistency of tuples who are consistent in \( R_1 \) or \( R_2 \) against with tuples in the other table, and to attach marks to inconsistent ones, and 3) queries to get answer from merged and remarked \( tmpR1 \) and \( tmpR2 \). Though these queries, tuples who satisfy query condition will be correctly marked out and selected as query answer.

Algorithm: Union

Input: FD set \( \Psi \), user query in form of:

\[
\text{select } A1,\ldots,A_n \text{ from } R \text{ Where } \vartheta
\]

union

\[
\text{select } A1,\ldots,A_n \text{ from } R \text{ Where } \omega
\]

Suppose \( W_1,\ldots,W_n \) are determined attributes in \( \omega' \) or \( \omega'' \), but \( \notin \{A1,A2,\ldots,A_n\} \).

Output: AQA

1. Rewrite \( \omega' \) to \( \vartheta \) according to the next rule.

\[
\text{For each } a \in \omega' \text{ where } v \text{ is const.}
\]

\[
(\text{Suppose } L_1 \rightarrow a, L_2 \rightarrow a, \ldots, L_n \rightarrow a \text{ are FDs on } R_1)
\]

Replace \( a \omega \) with \( a \omega' \text{ or } L_1 \in (\text{select } L_1 \text{ from } R_1)
\]

Where \( aW \) and \( a \) is not null) or \( L_2 \) in (select \( L_2 \text{ from } R_1
\]

2. Simarily rewrite \( \omega'' \) to \( \vartheta'' \)

3. Execute the next queries:

a) select \( A1,A1,\ldots,An1,An1,An2,An2,\ldots,Wm1,Wm11 \)

into \( tmpR1 \) from \( R1 \) where \( \vartheta'' \);

b) select \( A1,A1,\ldots,An1,An1,An2,An2,\ldots,Wm2,Wm22 \)

into \( tmpR2 \) from \( R2 \) where \( \vartheta'' \);

4. For each \( A1, A1, \ldots, Wm1, Wm11, Wm2, Wm22 \)

in \( tmpR1 \) and \( \vartheta'' \), execute the next queries

a) update \( T \text{ set } T.fdr = T.fdr + 't'

from \( tmpR1 T \)

where \( T.fdr \) is null and exists (select \( A.fdr \)

from \( R2 A \) where \( T.fdr = A.fdr \)

group by \( A.fdr \) having count (distinct \( A.fdr \) > 1);

b) update \( T \text{ set } T.fdr = T.fdr + 't'

from \( tmpR2 T \)

where \( T.fdr \) is null and exists (select \( A.fdr \)

from \( R1 A \) where \( T.fdr = A.fdr \)

group by \( A.fdr \) having count (distinct \( A.fdr \) > 1);

5. Last execute the next query and return query result:

a) select \( A1,A1,\ldots,An1,An1,An2,An2,\ldots,Wm2,Wm22 \)

from \( \text{select } A1, A1, \ldots, Wm1, Wm11 \)

union

\[
\text{select } A1, A1, \ldots, Wm2, Wm22 \text{ from } tmpR1
\]

union

\[
\text{select } A1, A1, \ldots, Wm2, Wm22 \text{ from } tmpR2
\]

T
In union operation, the two relations describe the same entity. Original annotations in both relations cannot exactly denote to its inconsistency without global checking. It’s similar to relations involved in difference operation. In the rewritten algorithm of query R1-R2, we first recheck and remark cell values in R1 against records in R2, then doing difference to exclude tuples in R1 who are equal to or value equal to a tuple in R2.

VI. EXPERIMENTAL EVALUATION

We mainly state the experimental evaluation of query rewriting algorithms presented in this paper, to compare performance of AQA queries and SQL queries, and different AQA queries over different scale database with different ratio of inconsistent data.

Experimental environment. Settings of the experiment are: Intel Celeron 420 2.0GHZ CPU, 1GB memory, XP+SP2, C#/VC6.0 and SQL Server 2000.

Data set generation. To test the efficiency of AQA on different size of data sets, we developed a synthetic data generator which can be run with two parameters, the scaling factor (database size, ds) and the inconsistency factor (dirty ratio, dr) that controls ratio of “dirty” tuples. All generated data conform to schema shown in figure 1.

The two group data sets used in the experiments are shown in table1. The first group data sets are in size of 1GB but with different dr of 1%, 5%, 10% and 15%. While the second group data sets are in size of 0.1GB, 0.5GB, 1GB and 1.5GB, and with dr of 5%. All the data sets are sorted by primary key attribute in advance.
Time performance of AQA rewriting. The first group experiments compare time performance of q1-q11 (SQL queries) and their corresponding rewritten query(AQA queries). As shown in figure 7, for queries over a single table where no derived FDs are implied, performance of AQA is close to SQL. But for join query, evaluation of AQA queries need much more time than SQL queries. That is because when derived FD exists, computations of annotation require table scanning for each derived FD. Furthermore, the execution time goes more sharply as more tables are joined together. In fact, mark maintaining is the most time consuming operation.

The second group of experiments test performance of AQA queries over database with different $ds$ and different $dr$. As figure 8(a) shows, when only $dr$ changes, queries without join changes little, while time of join queries is polynomial against the $dr$. The reason is that more inconsistency validation is executed as more dirty tuples exist in the database. On the other side, when $dr$ keeps no change and $ds$ changes, as shown in figure 8 (b), time of query q3, q4, q5, q9 and q10 changes little because of index on $cname$, time of q2, q6 and q7 goes sharply because full scan time goes sharply, while q8 and q11 go sharper with database scale.

Query optimization of AQA. To improve the time performance join query, we present an optimization of AQA. Difference between AQA and its optimization is the query sequence. In AQA, we first do query for all tuples to form the “possible world” and compute their annotations, then filter them with query condition. Meanwhile, in the query optimization, we first filter tuples with query condition, then compute their annotation. For those who don’t conflict with a record in the query result, we will validate its consistency in the “possible world”. Take query Q1 as an example, with the optimization, its rewritten queries are:

```sql
Select C.*, T.* into tmpR
From Class C, Teacher T
Where
(major='Art' or majorA is not null)
and (Tname = any (select Tutor from Class C2
where Cname=C.CName));
update T
set T.PhoneA=T.PhoneA+'Class.CName'
from tmpR T
where exists (select B.CName from tmpR B
where T.CName=B.CName
and T.Phone!=Teacher.Phone); update T
set T.PhoneA=T.PhoneA+'Class.CName'
from tmpR T
where exists (select * from Class C, Teacher T
where (Tname = any (select Tutor from Class C2
where Cname=C.CName))
and T.CName=C.CName
and T.Phone!=Teacher.Phone);
select CName, CNameA, Phone, PhoneA,
Major, MajorA, Tutor, TutorA
From tmpR;
```
The query optimization evidently improves AQA’s performance when a few records satisfy the query. As shown in figure 10 and figure 11, after optimization, performance of q8 and q11 are sharply improved and close to normal SQL query, regardless different level of database size, dirty ratio and tables joined together.

Information Loss. Here we analyzes the information preserving ability of CQA and database repairing with tuple deletion (RWD below), and compare them with AQA on query q1 and q7. Information loss rate is calculated as follows: total number of all lost attribute values divide total number of attribute values satisfying the query, for example, there are 4 tuples satisfy q7 with no consideration of inconsistency, and two of them conflict on Major but consistent on CName and Tutor which will not appear in query answer with method of RWD, so that information lost rate is 4/12=33.3%. The experimental results in figure 12 (a) and (b) show that although information loss varies among different queries, RWD and CQA loss lots of information while AQA lose nothing in any case.

Experiment evaluation for tpc-h data and queries. The next experiment compares time performance of AQA and SQL over a tpc-h database [18]. The database is stored in SQL Server 2000, and initial size of each table is listed in table 2. We don’t change its integrity constraints but add some noise data. For each of the first 6 tables, we copy a number of its records into another table, and update these records on specified attributes, and then append them back into the original table so that they must conflict with their corresponding records on the updated attributes. Number of the appended revised records and the updated...
attributes are listed in table 2. By this way, dirty ratio of the database is 5%.

Experiment result in figure 13 shows that AQA with optimization is close to SQL queries.

![Figure 13. Performance over TPC-H databases](image)

VII. CONCLUSION AND FUTURE WORK

Conflicting and incomplete information are implied in inconsistent data and query answers over it. Hot discussed problems include how to represent inconsistent data, what its query answer should be, and how to compute it. Although from the user’s perspective, a single sure query answer would be desirable in most cases. The probabilistic nature of inconsistent data makes such query answer impossible.

As previous work, we present a weak representation named AQA where inconsistent cell values in both data source and query results are attached with annotations. It can avoid information loss, a vital and common deficiency of many previous works in this area. In this paper, we focus on an implementation strategy of it. We propose algorithms to rewrite queries without aggregation and correlated sub query so that its AQA can be correctly computed. The main difference between our method and other related work is its support of attribute-level inconsistency. Furthermore, our approach doesn’t require a new query language and can be easily embedded into existing database applications. Still more, our approach can deal with databases from different DBMS.

Insofar, our approach is limited to constraint type of FD and SPJUD queries. As a future work, we will extend it so that it can deal with other type of constraint and aggregation queries.

REFERENCES


APPENDIX A   QUERIES USED IN THE EXPERIMENTS

The following are the 11 queries, adapted from the TPC-H specification, that were used in the experiments.

**Query 1:**

select l_returnflag, l_extendedprice as avg_price,

**Figure 13. Performance over TPC-H databases**

© 2013 ACADEMY PUBLISHER
l_quantity as sum_qty, l_quantity as avg_qty,
l_linenumber, l_extendedprice as sumBasePrice,
l_extendedprice*(1-l_discount) as sumDiscPrice,
l_extendedprice*(1-l_discount)*(1+l_tax) as sumCharge,
l_discount as avg_disc
)
from lineitem
where DAYS('1998-12-01')-DAYS(l_shipdate) >90
order by l_returnflag, l_linenumber;

Query 2:
select s_acctbal, s_name, n_name, p_partkey,
p_mfgr, s_address, s_phone, s_comment
from part, supplier, partsupp, nation, region
where p_partkey = ps_partkey
and s_suppkey = ps_suppkey
and p_size = 15
and p_type like '%BRASS'
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = 'EUROPE'
order by s_acctbal desc, n_name, s_name, p_partkey

Query 3:
select l_orderkey, o_orderdate, o_shippriority,
l_extendedprice * (1 - l_discount) as revenue
from customer, orders, lineitem
where c_mktsegment = 'BUILDING'
and c_custkey = o_custkey
and l_orderkey = o_orderkey
and o_orderdate < '1995-03-15'
and l_shipdate > '1995-03-15'
order by revenue desc, o_orderdate

Query 4:
select o_orderpriority
from orders, lineitem
where o_orderdate >= '1993-07-01'
and days(o_orderdate) <days('1993-07-01') + 90
and l_orderkey = o_orderkey
and l_commitdate < l_receiptdate
order by o_orderpriority

Query 6:
select l_extendedprice * l_discount as revenue
from lineitem, part
where p_partkey = l_partkey
and p_brand = 'Brand#23'
and p_container = 'MED BOX'

Query 7:
select c_custkey, c_name, c_acctbal, n_name, l_extendedprice * (1 - l_discount) as revenue,
c_address, c_phone, c_comment
from customer, orders, lineitem, nation
where c_custkey = o_custkey
and l_orderkey = o_orderkey
and o_orderdate >= '1993-10-01'
and days(o_orderdate) < days('1993-10-01') + 90
and l_returnflag = 'R'
and c_nationkey = n_nationkey
order by revenue desc

Query 8:
select c_name, c_custkey, o_orderkey,
o_orderdate, o_totalprice, l_quantity
from customer, orders, lineitem
where o_orderkey = l_orderkey
and c_custkey = o_custkey
and o_orderkey = l_orderkey
order by o_totalprice desc, o_orderdate

Query 10:
select c_custkey, c_name, c_acctbal, n_name,
l_extendedprice * (1 - l_discount) as revenue,
c_address, c_phone, c_comment
from customer, orders, lineitem, nation
where c_custkey = o_custkey
and l_orderkey = o_orderkey
and o_orderdate >= '1993-10-01'
and days(o_orderdate) < days('1993-10-01') + 90
and l_returnflag = 'R'
and c_nationkey = n_nationkey
order by revenue desc

Query 11:
select ps_partkey,
ps_supplycost * ps_supplycost as value
from partsupp, supplier, nation
where ps_suppkey = s_suppkey
and n_nationkey = n_nationkey
and n_name = 'GERMANY'
order by value desc

Query 17:
select   l_extendedprice / 7.0 as avg_yearly
from    lineitem,  part
where       p_partkey = l_partkey
and p_brand = 'Brand#23'
and p_container = 'MED BOX'

Query 18:
select  c_name,  c_custkey,  o_orderkey,
o_orderdate,  o_totalprice,  l_quantity
from  customer,  orders,  lineitem
where    o_orderkey = l_orderkey
and l_quantity > 300
and c_custkey = o_custkey
and o_orderkey = l_orderkey
order by  o_totalprice desc,   o_orderdate

Query 20:
select s_name, s_address
from supplier, nation, partsupp, part
where s_suppkey=ps_suppkey
and ps_partkey=p_partkey
and p_name like 'forest%'
and s_nationkey = n_nationkey
and n_name = 'CANADA'
order by s_name

Aihua Wu
Born in Jiangxi Province, China, 1976.7, and received her M.Sc. and PH.D. of computer science from International Database Center, Fudan University, China, in 2004 and 2010. In 2007 - 2008, she worked on uncertain database while visiting University of California at Santa Barbara. And now she is an associate professor of Shanghai Maritime University, China. Her current research interests include uncertain database, inconsistent XML, BPM, data mining and knowledge discovery. Dr. Wu is membership of China Computer Federation.