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 *Annotated Relation*, such database *Annotated Database*, and the query answer *Annotation Based Query Answer* (AQA for short below). The approach can avoid information loss.

For a given *Annotated 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*.

[*] Supported by National Natural Science Foundation of China under Grant No. 61202022 and Science & technology program of shanghai Maritime University No. 20110042

doi:10.4304/jsw.8.2.375-386

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 Batabase*, 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 loss 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 time performance of evaluating SQL queries and their rewritten ones. We test 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.

Class

	CName	Major	Tutor
t1	MA08	Math	Alex
t2	MA09	Math	Alex
t3	Art05	Art	Lee
t4	Art05	Art	Kimi
t5	Art081	Art	Bobby
t6	Art081	Magic	Bobby
t7	CIT08	CIT	Ella
t8	EE08	EE	Ella

FDs: Cname-> Major, Tutor

Teacher

	Tname	City	Email	Phone
t9	Alex	Brea	Alex@ ucs.edu	5651565
t10	Lee	Olive	Lee@ucs.edu	3822820
t11	Kimi	Gorita	Kimi@ucs.edu	6544460
t12	Bobby	Brea	Bobby@ucs.edu	6881234
t13	Ella	Olive	Ella@ucs.edu	6425151

FDs: Tname-> Title, Email, Phone

Student

	SID	SName	Age	Class
t14	1	Ada	20	MA08
t15	1	Ada	20	MA09
t16	2	Jack	18	Art05
t17	3	Jane	19	CIT08
t18	4	Elindi	21	Art081
t19	4	Elindi	20	Art081

FDs: SID->SName, Age, Class

Class (Annoated)

	CName	CNameA	Major	MajorA	Tutor	TutorA
t1	MA08		Math		Alex	
t2	MA09		Math		Alex	
t3	Art05		Art		Lee	*
t4	Art05		Art		Kimi	*
t5	Art081		Art	*	Bobby	
t6	Art081		Magic	*	Bobby	
t7	CIT08		CIT		Ella	
t8	EE08		EE		Ella	

Teacher(Annotated)

	Tname	TnameA	City	CityA	Email	EmailA	Phone	PhoneA
t9	Alex		Brea		Alex@ ucs.edu		5651565	
t10	Lee		Olive		Lee@ucs.edu		3822820	
t11	Kimi		Gorita		Kimi@ucs.edu		6544460	
t12	Bobby		Brea		Bobby@ucs.edu		6881234	
t13	Ella		Olive		Ella@ucs.edu		6425151	

Figure 1. Two annotated relation of database Student

R

CName	CNameA	Phone	PhoneA
Art05		3822820	
Art05		6544460	
Art081		6881234	

(a) an inaccurate query answer of Q1

R'

CName	CNameA	Phone	PhoneA	Major	MajorA	Tutor	TutorA
Art05		3822820	Class.CName	Art		Lee	*
Art05		6544460	Class.CName	Art		Kimi	*
Art081		6881234		Art	*	Bobby	
Art081		6881234		Magic	*	Bobby	

FDs: Cname->Phone

(b) 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 *XA* are added to record inconsistency of each tuple on *X*, e.g in annotated *Class* of figure 1, *t5[Major]* conflicts with *t6[Major]* according to *CName->Major*, so annotation "*" is assigned to *t5[MajorA]* and *t6[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 against the "new born" FD.

Q1: *select CName, Phone*
From Class, Techaer
Where class.Tutor=Teacher.Tname and major='Art'
Q1':*select CName, CNameA, Phone, PhoneA*
from Class, Techaer
where class.Tutor=Teacher.Tname and major='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" is assigned to the first two tuples of R', denoting their inconsistency w.r.t. *CName->Phone*.

The following is a formal definition of our data model.

Definition1 ***uncertain data***: Given a relation *R*, and a set of FD ψ on it, $\forall(x->y)\in\psi, \forall t1,t2\in R$, if *t1[X]=t2[X]* and *t1[Y]!=t2[Y]*, we say that piece of data *t1[Y]* and *t2[Y]* is uncertain data w.r.t. *x->y*.

Definition2 ***Annotated Relation***: Given a relation *R* and its FD sets *F*, if all uncertain pieces of data of *R* w.r.t. *F* are attached with one or more marks, *R* is an *annotated relation* w.r.t. *F*. Similarly, for any query Q over R, if all uncertain pieces of data of *Q(R)* are attached by marks, *Q(R)* 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 ***domain equality (DEQ)***: Given database schema *D*, domain equality statement $X \overset{D}{=} Y$ is true iff for any instance of *D* and any tuple *t* of the instance, $\exists$ *t'*, that *t[X]=t'[Y]*, here *t* and *t'* can be same tuple.

Definition 4. Given a query *Q* and a set of FD *F*, suppose *U1,U2,…,Un* are attributes appear in *select, where, group by, having, order by* of *Q*, ***projection of F on Q*** is project of *F* on *R(U1,U2,…Un),* similarly, ***projection of domain equality DEQ on Q*** is projection of *DEQ* on *R(U1,U2,…Un).*

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) $X \overset{D}{=} Y$ |- $Y \overset{D}{=} X$
2) $X \overset{D}{=} Y$, $Y \overset{D}{=} Z$ |- $X \overset{D}{=} Z$
3) |- $X \overset{D}{=} X$
4) $X \overset{D}{=} Y$ |- *X->Y*
5) *Z->A1, Z->A2,* $\forall$ *t1,t2 if t1[Z]=t2[Z], t1[A1]=t2[A2]* |- $A1 \overset{D}{=} A2$

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) Drv(*R*): Picture each of *R*'s FDs as FD tree [20], then take the closure.
2) Drv(*e[X]*)(projection): Take all DEQs $Y \overset{D}{=} Z$ where $X[Y] \overset{D}{=} X[Z]$ is in Drv(*e*), and all FDs *Z->A* that *X[Z]->X[A]* is in Drv(*e*).
3) Drv(*e*[$X \overset{D}{=} Y$])(selection with domain equality): Add $X \overset{D}{=} Y$ to Drv(*e*) and take the closure
4) Drv(*e1* $\bowtie$ *e2*)(join): Rename the constraints in Drv(*e2*) according to the degree of *e1*, i.e. a DEQ $X \overset{D}{=} Y$ becomes *X+k=Y+k* (*k*=degree(*e1*)) and an FD *Z->A* becames *Z+k->A+k*. Then add renamed Drv(*e2*) to Drv(*e1*) and take the closure.
5) Drv(*e1* U *e2*)(union): A DEQ $X \overset{D}{=} Y$ is in Drv(*e1* U *e2*) if it is in both Drv(*e1*) and Drv(*e2*). If *Z->A* is in Drv(*e1*) and *Z->A* is in Drv(*e2*) , *e1.Z* is domain equal to *e2.Z* and *e1.A* is domain equal to *e2.A*, *Z->A* in Drv(*e1* U *e2*).
6) Drv(*e1-e2*)(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 FD with its domain equal attribute and add the new FD to FD set.
2) Replace every *determine attribute* of FD 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. *AQA* 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 *AQAs*.

Q2(Student)

SID	SIDA	SName	SNameA	Age	AgeA	Class	ClassA
2		Jack		18		Art05	
4		Elindi		21	*	Art081	
4		Elindi		20	*	Art081	

Figure 3. AQA of Q2.

R1

O	OX	P	PX	Q	QX
1		A	*	g	*
1		B	*	e	*
1		A	*	n	*
4		D		v	

R2

O	OX	P	PX	Q	QX
1		A		g	*
1		A	*	n	*
4		D		v	

R1-R2

O	OX	P	PX	Q	QX
4		D		v	

FDs: O->P, O->Q

Figure 4. An example of difference between 2 annotated tables

*Q2: select * From Student Where Age<=18*

Q3: $\prod_{phone}(R)$

Phone	PhoneA
5651565	
3822820	
6544460	
6881234	
6425151	

Q4: $\sigma_{CName=\text{"Art05"}}(\prod_{phone}(R))$

Phone	PhoneA
3822820	R.CName
6544460	R.CName

R=Class∞Teacher

Figure 5. Suppose *R* is inner join result of *Class* and *Teacher*, annotation-based Query Answer of Q3, Q4.

Q3:select Phone from R
Q4: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 *AQA* by query rewriting for given SQL queries when valid set of FD on the query result is know 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.

Q5: *select * from Class*
Q5': *select * from Class*

Rewritten query of Q5 is Q5'. Notice that * in Q5 and * in Q5' 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".

Q6: *select CName, Tutor from Class*
where Major='Art'
Q6': *select CName, CNameA,Tutor,TutorA from Class*
where Major='Art'
Q6'':*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 {*('Art05','','Lee', '*'), ('Art05','','Kimi','*'), ('Art081','','Lee','')*} which can be obtained by evaluating Q6'. 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 rewritten 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
   set T.PhoneA=T.PhoneA+'Class.CName'
   from tmpR1 T
   where exists (select B.CName from tmpR1 B
           where T.CName=B.CName
           group by B.CName.
           having count(distinct b.Phone)>1);
2)  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
```

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 tname from Tutor Where city='Brea')
Q7':select Cname,CNameA,Major,MajorA, Tutor,TutorA
from class
where Tutor in (Where city='Brea' or tname in (
select tname From teacher
Where city='Brea' and cityA is not null))
Or cname in (
select cname From class
where TutorA is not null and Tutor in (
select tname From teacher
where city='Brea' or tname in (
select tname 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, θ 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 Lj->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 θ is predicate >,<,>=,<=,= or <>, $\bar{\theta}$ would be <,>,<=,>=,= or <> respectively.

The bunch of queries in the second level updates annotations. Each of them checks one *Derived FD*. In this example, only one *Derived FD* needs to be verified.

The last query answer is made up of two parts: one is potentially inconsistent w.r.t. *Derived FD* and the other can not violate any *Derived FD*. The third level query unions them together.

Algorithm: SPJ(Ω, Ψ, Q, *I*)

Input: domain equality Ω, derived FD set Ψ, return type *I* (0 for query answer, 1 for rewritten query), and user query Q in form of:

Select *A1,A2,…,An*
From *R1,R2,…Rk*
Where ω
Order by κ

Suppose W1,W2,…Wn are attributes that appear in ω *but not in { A1,A2,…An}.*

Output: AQA of Q or rewritten query φ

1.Rewrite ω to ϖ according to the next rule:

a)For each $a\theta v$ *in* ω (*v* isn't attribute and θ is predicate and a is determined attribute).

Replace $a\theta v$ *with (* $a\theta v$ *or L1 in (select L1 from Ri Where* $a\theta t$ *and* a *is not null) or L2 in (select L2 from Rj Where* $a\theta t$ *and* a *is not null) …)*

b)For each sub query δ in ω

Replace δ with SPJ(Ω', Ψ', δ, 1), here Ω' and Ψ' are projection of Ω and Ψ on δ respectively.

2.Set Ψ'=Projection of Ψ on Q

3.If Ψ'=NULL

If (I=0)

Excute the next query and return the query answer:

select A1,A2,…,An, A1A, A2A,…AnA, W1, W1A,…Wm,WmA
From R1,R2,…Rk Where ϖ *Order by* κ

else

{ φ *= select A1,A2,…,An From R1,R2,…Rk Where* ϖ
Return φ *}*

Else

{ a)Ω'={E | $E \in$ Projection of Ω on R1 $\cup$ R2 $\cup$ … $\cup$ Rk}

b) Depart ϖ into two part: ϖ_1 which includes all $Ri.A\theta Rj.B$ *where Ri and Rj* $\in \{R1,...,Rk\}$, *A* and *B* are attributes, θ is predicate, and ϖ_2 of the left.

c) For each $Ri.A\theta Rj.B$ *in* ϖ_1

{ S="";

Suppose *B*-related *FDs* are *B1->B,…,Bn->B* and *A*-related *FDs are A1->A,…,An->A*

if Rj.B is *determined attribute*

S=Ri.A θ *any (select B from Rj T*
where T.B1=Rj.B1 and … and T.Bn=Rj.Bn)

If Ri.A is determined attribute

If S=""

S= Rj.B $\bar{\theta}$ *any (select A from Ri T*
where T.A1=Ri.A1 and … and T.An=Ri.An)

Else

S=S+ or Rj.B $\bar{\theta}$ *any (select A from Ri T*
where T.A1=Ri.A1 and … and T.An=Ri.An)

Replace $Ri.A\theta Rj.B$ *with S;}*

d) Execute the next query where *Ei* $\in \Omega'$, and *fdli* is determine attribute of a FDi in Ψ':

Select R1., R2.* Into tmpR From R1,R2,…Rk*
Where ϖ_1 *and fdl1 in (select fdl1 from Ri group by fdl1 having count(distinct fdr1)>1)*
and fdl2 in (select fdl2 from Rj group by fdl2 having count(distinct fdr2)>1);

…

e) For each FD fdl->fdr in Ψ', execute the next query

update T set T.fdrA=T.fdrA+'fdl' from tmpR T
where exists (select b.fdl from tmpR b
where T.fdl=b.fdl group by b.fdl
having count(distinct b.fdr)>1)

f) If (I=0)

Excute the next query and return the query answer:

select A1,A2,…,An, A1A, A2A,…AnA, W1, W1A,…Wm,WmA
From tmpR Where ϖ_2 *Order by* κ
Union
select A1,A2,…,An, A1A, A2A,…AnA, W1, W1A,…Wm,WmA
*From ((select * from R1 where R1.key in (*
select key from R1
group by fdl1,fdl2,…fdln
having count(*) =1)) R1, …,
*((select * from Rk where Rk..key in (*
select key from Rk
group by fdl1,fdl2,…fdln
having count(*) =1)) Rk
Where ϖ *Order by* κ
else
{ φ *= select A1,A2,…,An From tmpR Where* ϖ_2
Union
select A1,A2,…,An
*From ((select * from R1 where R1.key in (*
select key from R1
group by fdl1,fdl2,…fdln
having count(*) =1)) R1, …,
*((select * from Rk where Rk..key in (*
select key from Rk
group by fdl1,fdl2,…fdln
having count(*) =1)) Rk
Where ϖ }

V. Union and Difference Queries

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

Class1

	CName	CNameA	Major	MajorA	Tutor	TutorA
t1	CSE081		CSE		Judy	
t2	EE081		EE		Nancy	
t3	CIT08		CIT		David	
t4	NE08		NE		David	

FDs: Cname-> Major, Teacher

Class2

	CName	CNameA	Major	MajorA	Tutor	TutorA
t5	CSE081		CSE		Nancy	
t6	EE081		EE		Nancy	*
t7	EE081		EE		Philip	*
t8	CIT08		CIT		David	

Figure 6. Example of Union over Annotated table

Example5: 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'
union
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 $R1 \cup R2$. 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 *R1* and *R2* into *tmpR1* and *tmpR2* respectively; 2) queries to verify consistency of tuples who are consistent in *R1* or *R2* 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 Ψ , user query in form of :
select A1,….An from R1 where ω'
union
select A1,….An from R2 where ω''
Suppose W1,…,Wn are determined attributes in ω' *or* ω'' *but* $\notin$ *{ A1,A2,…An}* .
output: AQA
----

1. Rewrite ω' to ϖ' according to the next rule:
For each $a\theta$v in ω' where *v* is const.
{ Suppose *L1->* a , *L2->* a ,… , *Ln->* a are FDs on R1.
Replace $a\theta t$ with *(* $a\theta$v or *L1* in *(*select *L1* from *R1* Where $a\theta$v and a is not nul*l)* or *L2* in (select *L2* from *R1* Where $a\theta$v and a is not null*)…)* }
2. Similary rewrite ω'' to ϖ''
3. Excute the next queries:
a) select *A1,A1A,….An,AnA, W1,W1A,…,Wm1,Wm1A* into *tmpR1 From R1* where ϖ' *;*
b) select *A1,A1A,….An,AnA, W1,W1A,…,Wm2,Wm2A* into *tmpR2 From R2* where ϖ'' ;
4. For each *fdl->fdr* $\in \Psi$ in *R1(R2), fdl* $\in$ *tmpR1* and *fdr* $\in$ *tmpR1*, execute the next queries
a) update *T* set *T.fdr =T.fdr*+',*'
from *tmpR1 T*
where *T.fdr* is null and exists (select *A.fdl*
from *R2 A* where *T.fdl=A.fdl*
group by *A.fdl* having count(distinct *A.fdr*)>1);
b) update *T* set *T.* =*T.fdr*+',*'
from *tmpR2 T*
where *T.fdr* is null and exists (select *A.fdl*
from *R1 A* where *T.fdl=A.fdl*
group by *A.fdl* having count(distinct *A.fdr*)>1);
5. Last execute the next query and return query result:
select *A1,A1A,….An, AnA W1,W1A,…,Wm2,Wm2A*
from (select * from *tmpR1*
union
select * from *tmpR2*) *T*

In union operation, the two relations descript the same entity. Original annotations in both relations can not 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*.

---

algorithm: difference

input: FD set Ψ ={fdli->fdri (i=1,2,…,n)} and user query in form of :

select A1,….An from R1 where ω'

minus

select A1,….An from R2 where ω''

Suppose W1,…,Wn are determined attributes in ω' *or* ω'' *but* $\notin$ *{ A1,A2,…An}* .

output: AQA

1.Rewrite ω' to ϖ' according to the next rule:

For each $a\theta t$ *in* ω' *where t is const.*

{ Suppose *L1->* a *, L2->* a *,… , Ln->* a are FDs on *R1,*

Replace $a\theta$v with *(* $a\theta$v *or L1 in (select L1 from R1 Where* $a\theta$v *and* a *is not null) or L2 in (select L2 from R1 Where* $a\theta$v *and* a *is not null)…)*}

2. Similary rewrite ω'' to ϖ''

3. Execute the next query:

select A1,A1A,….An,AnA, W1,W1A,…,Wm1,Wm1A into tmpR1 from R1 where ϖ'

4. For each fdl->fdr $\in\Psi$ in R1, fdl, fdr $\in$ tmpR1, execute the next queries to remark tmpR1:

```
update T set T. =T.fdr+',*'
from tmpR1 T
where T.fdr not like '%*%' and exists (select A.fdl
  from R2 A where ϖ'' and T.fdl=A.fdl
    group by A.fdl having count(distinct A.fdr)>1);
```

5. Excute the next query and return query result.

```
select A1,A1A,….An ,W1,W1A,…,Wm2,Wm2A
from tmpR1
where not exists ( select * from R2 where ϖ'' and
    R2.A1=tmpR1.A1 and … and R2.An=tmpR1.An)
```

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.

TABLE I.

Data used in the experiments

	name	Size	Dirty ratio
group1	DB11	1GB	1%
	DB12	1GB	5%
	DB13	1GB	10%
	DB14	1GB	15%
group2	DB21	103MB	5%
	DB22	536MB	5%
	DB23	1GB	5%
	DB24	1.5GB	5%

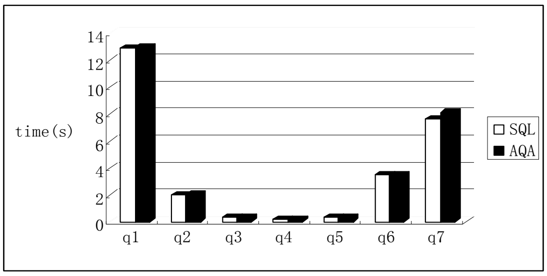
TABLE II.

TPC-H data used in the experiments

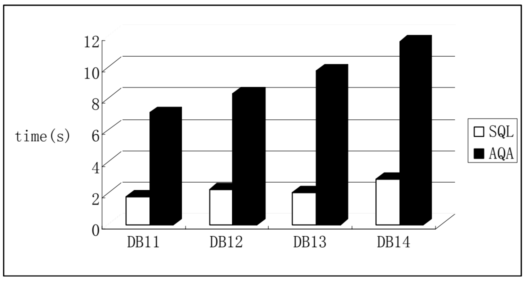
table	Records	noise records	Updated Attributes
supplier	10,000	1,750	s_nationkey
partsupp	800,000	100,000	ps_supplycost
part	200,000	45,000	P_brand
orders	1500,000	337,500	o_custkey
customer	150,000	30,000	c_address
lineitem	6000,000	1200,000	l_quanlity, l_shipdate
nation	25	0	
region	5	0	

Queries. 11 queries are used in the experiment. Queries *q1-q6* are about table *Class* and without join, *q7* is a nested query, *q8* and *q9* are join operations. Query *q10* and *q11* are union and difference.

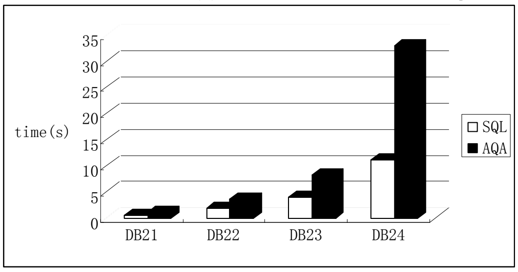
```
q1: select cname, major
    from class
q2: select cname,major
    from class
    where cname='c5' or major='m93'
q3: select major,cname
    from class
    where cname = 'c2' and major = 'm3' and Tutor >= 't1500'
q4: select *
    from class
    where cname like 'c2000%'
q5: select *
    from class
    where cname is null
q6: select major,cname
    from class
    where cname = 'c5'
      and major >= 'm21035'
      or major <= 'm1000'
q7: select *
    from class
    where Tutor in (
      select tname from teacher
      where city='Brea')
q8: select cname,major, Tutor, city,Phone
   from class,teacher
  where cname = 'c2000' and class.Tutor = teacher.tname
q9: select *
   from student s,class c,teacher t
   where s.class=cname and Tutor=t.tname and cname='c65'
q10: select cname,major
     from class
     where cname = 'c18'
     union
     select cname,major
     from class
     where cname = 'c108'
q11: select cname,major
     from class
     where cname in ('c18','c108')
     minus
     select cname,major
     from class
     where cname = 'c18'
```



(a) Time of AQA and SQL over DB14



(b) Time of AQA and SQL of *q8*

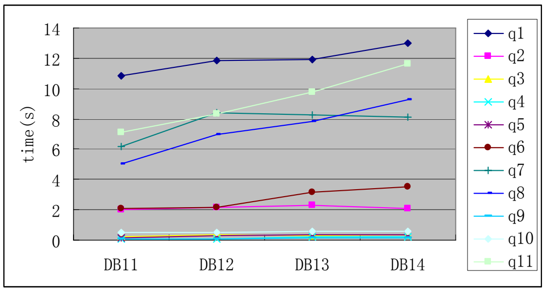


(c) Time of AQA and SQL of *q11*

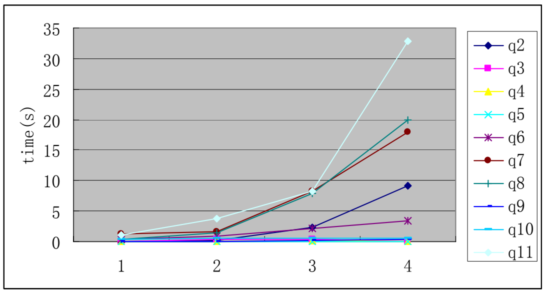
Figure 7. Performance of AQA and normal SQL

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 difference *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.



(a) time of AQA when noisy data changed

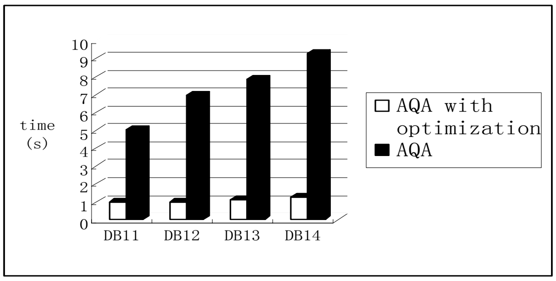


(b) time of AQA when database size changed

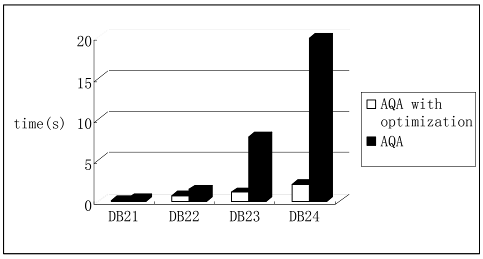
Figure 8. Time performance of q1-q11 over databases with different *dr* and different *ds*.

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:

```
 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
         group by B.CName.
        having count(distinct b.Phone)>1);
 update T
 set T.PhoneA=T.PhoneA+'Class.CName'
 from tmpR T
 where T.PhoneA is null and exists
          (select *  from Class C, Teache
                 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;
```

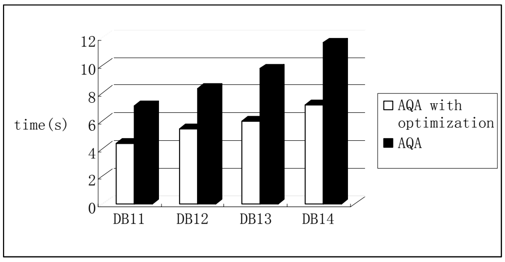


(a) result when noisy data changed

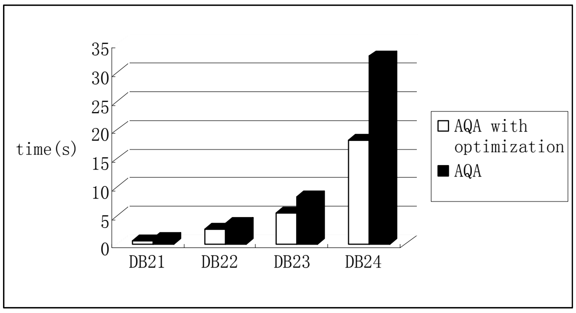


(b) result when database size changed

Figure 9. Time performance of AQA and its optimization of q11



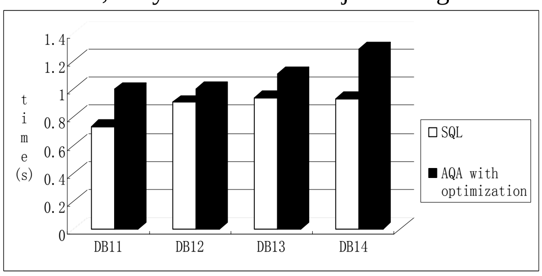
(a) result when noisy data changed



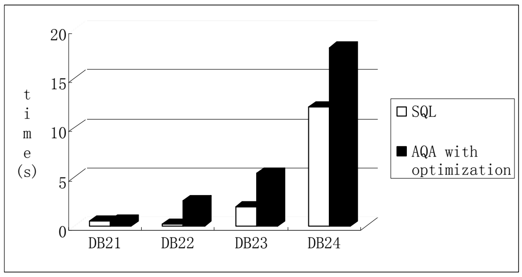
(b) result when database size changed

Figure 10. Time performance of AQA and its optimization of q8

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.



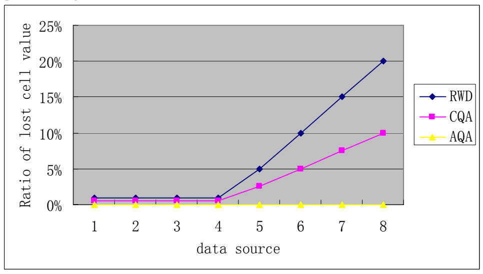
(a) SQL and AQA with optimization of *q8*



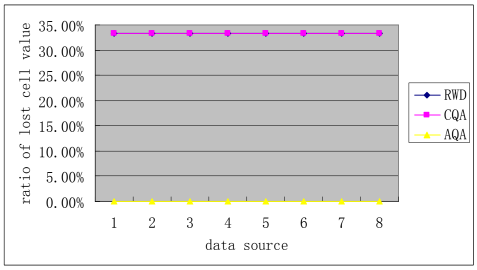
(b) SQL and AQA with optimization of *q11*

Figure 11. Time performance of SQL and AQA with optimization

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.



(a) Information loss of RWD, CQA and AQA for *q1*



(b) Information loss of RWD, CQA and AQA for *q7*

Figure 12. Information loss compare of RWD, CQA and AQA

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.

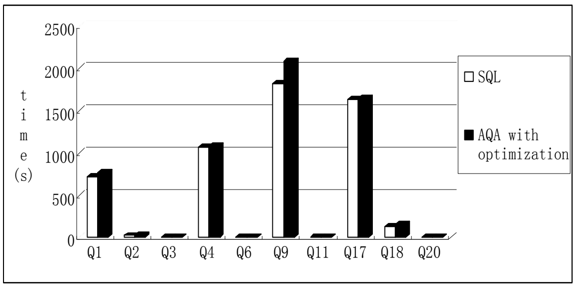


Figure 13. Performance over TPC-H databases

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

[1] Periklis Andritsos, Ariel Fuxman, Ren´ee J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In Proc. of the 22nd Intl. Conference on Data Engineering, April 3-8, 2006, Atlanta, USA. 2006, pp.30.

[2] Prithviraj Sen, Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In Proc. of the 23rd Intl. Conference on Data Engineering Istanbul, Turkey, April 15-20, 2007, ICDE, pp.596-605.

[3] Amol Deshpande, Carlos Guestrin, Sam Madden, etc. Model-driven data acquisition in sensor networks. In Proc. of the 30th Intl. Conference on Very Large Data Bases, Toronto, Canada, 8.31 – 9.3, 2004, pp. 588—599.

[4] Serge Abiteboul, Richard Hull, Victor Vianu: Foundations of Databases. Addison-Wesley 1995

[5] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, USA, 1999, pp.68-79.

[6] Aihua Wu, Zijing Tan, Wei Wang. Annotation Based Query Answer over Inconsistent Database. Journal of Computer Science and Technology. 2010, 25(3):467-479.

[7] Philip Bohannon, Michael Flaster, Wenfei Fan, etc. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In Proc. of the 24th ACM SIGMOD intl. conference on Management of data, June 14-16, 2005, Baltimore, Maryland, pp.143-154.

[8] L. Bertossi, L. Bravo, E. Franconi, etc. Complexity and Approximation of Fixing Numerical Attributes in Databases under Integrity Constraints. In Proc. of 10th Intl. Symposium on Database Programming Languages, Trondheim, Norway, August 28-29, 2005, pp. 262-278.

[9] J. Wijsen. Database Repairing using Updates. ACM Transactions on Database Systems, 30(3):722-768, 2005.

[10] L. Bravo and L. Bertossi. Logic programs for consistently querying data integration systems. In Proc. of the 18th Intl. Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, pp. 10-15.

[11] J. Chomicki, Consistent Query Answering: Five Easy Pieces. In Proc. of the 11th Intl. Conference of Database Theory, Barcelona, Spain, January 10-12, 2007, pp.1-17.

[12] Xi Zhang, Jan Chomicki: On the semantics and evaluation of top-k queries in probabilistic databases. In Proc. of the 24th Intl. Conference on Data Engineering Workshops, Cancún, México, April 7-12, 2008, pp. 556-563.

[13] Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In Proc. of the 25th Intl. Conference on Data Engineering, Shanghai, China, March 29 - April 2, 2009, pp. 305-316.

[14] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In Proc. of the 30th Intl Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3, 2004, pp. 864-875.

[15] M. Hua, J. Pei, W. Zhang, X. Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In Proc. of the 24th Intl. Conference on Data Engineering, Cancún, México, April 7-12, 2008,1403-1405.

[16] Aihua Wu, Zijing Tan, Wei Wang. Query Answer over Inconsistent Database with Credible Annotations. Journal of software (China), 2012,23(5):1167-1182.

[17] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions. In: Proc. 23rd Intl. Conference on Data Engineering (ICDE2007), Istanbul, Turkey, April 15-20: IEEE Computer Society , 2007, 1479-1480.

[18] T. P. C. (TPC). TPC Benchmark H: Standard Specification, 2009. http://www.tpc.org/tpch.

[19] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In: Proc. 2nd Biennial Conference on Innovative Data Systems Research (CIDR2005), Asilomar, USA, January 4-7, 2005, 262-276.

[20] Anthony C. Klug: Calculating Constraints on Relational Expressions. ACM Trans. Database Syst. 5(3): 260-290 (1980).

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,

```
        l_quantity as sum_qty, l_quantity as avg_qty,
        l_linestatus, 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_linestatus;
```

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
where   l_shipdate >= ’1994-01-01’
      and days(l_shipdate) < days(’1994-01-01’)+365
   and l_discount >= 0.06 - 0.01
   and l_discount <= 0.06 + 0.01
   and l_quantity < 24
```

Query 9:

```
select  n_name as nation,
     YEAR(o_orderdate) as o_year,
     l_extendedprice * (1 - l_discount) -
              ps_supplycost * l_quantity as amount
from   part,  supplier,  lineitem,  partsupp,  orders,  nation
where     s_suppkey = l_suppkey
      and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey
      and p_partkey = l_partkey
      and o_orderkey = l_orderkey
      and s_nationkey = n_nationkey
      and p_name like ’gr%’
order by  nation,  o_year desc
```

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_availqty as value
from  partsupp,  supplier,  nation
where    ps_suppkey = s_suppkey
     and s_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.