软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(5):1167−1182 [doi: 10.3724/SP.J.1001.2012.04079]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

不一致数据库上带信任标记的查询结果*

吴爱华[1+], 谈子敬[2], 汪 卫[2]

[1](上海海事大学 信息工程学院,上海 201306)

[2](复旦大学 计算机科学技术学院,上海 200433)

Query Answer over Inconsistent Database with Credible Annotations

WU Ai-Hua[1+], TAN Zi-Jing[2], WANG Wei[2]

[1](College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China)

[2](School of Computer Science, Fudan University, Shanghai 200433, China)

+ Corresponding author: E-mail: 061021058@fudan.edu.cn

Wu AH, Tan ZJ, Wang W. Query answer over inconsistent database with credible annotations. *Journal of Software*, 2012,23(5):1167−1182. http://www.jos.org.cn/1000-9825/4079.htm

Abstract: Inconsistent data is confusing and conflicting. Computing credible query answers over such data is significant. However, previous related works lose information. The approach of annotation based query answer (AQA) introduces confidence annotation to differ consistently and inconsistently in attribute value. Thus, a credible query answer can be computed and information loss can also be avoided. This is limited, however, in functional dependencies. This paper extends the approach to applications where multi constraints are involved, and no attribute is definitely credible. This paper redefines its representing model and query algebra, discusses the rules for calculating valid implied constraints of the above types on query result for any query algebra, proposes a cost based heuristic algorithm to repair, and annotates the initial database. The experiments show that time performance of extended AQA is almost similar to that of SQL for any query without join, and close to SQL for join queries after optimization, but it doesn't loss information.

Key words: uncertain data; data quality; consistent query answer; integrity constraints; data cleaning

摘 要: 不一致数据无法正确反映现实世界,其上的查询结果内含错误或矛盾,而现有的很多不一致数据查询处理相关研究都存在信息丢失的问题. AQA(annotation based query answer)针对这一问题采用信任标签在属性级别上区分一致和不一致数据,避免了信息丢失.但 AQA 假设记录在依赖左边属性上的分量可信,且只针对函数依赖一种约束,具有应用局限性.在综合约束(函数依赖、包含依赖和域约束)范围内、不确定属性任意的情况下扩展了 AQA,重新审视了 AQA 的数据模型及其上的查询代数,讨论了任意约束在查询结果上的蕴含约束计算问题.实验结果表明,扩展后的AQA非连接类查询的性能和普通的SQL基本相同,连接查询经优化后性能接近普通SQL查询,但AQA不丢失信息,与部分同类研究相比有很大优势.

关键词: 不确定数据;数据质量;一致的查询回答;完整性约束;数据清洗

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 上海海事大学校基金(20110042)

收稿时间: 2011-01-02; 修改时间: 2011-03-21; 定稿时间: 2011-07-01

给定数据库Φ及其完整性约束集Σ,若Φ的任意元组都不违反Σ的任意约束,则Φ一致;反之则不一致.

虽然完整性约束用于防止不一致已有很长时间,但由于各类原因,不一致数据仍广泛存在于各类现实应用中.它包括:数据整合与数据交换、隐私保护、RFID 无线网络、Web 数据抽取、科学数据管理等.

由于不一致数据无法正确反映现实世界,其上的查询结果内含错误或自相矛盾,极大地降低了数据的利用价值.另一方面,实际应用中很难纠正不一致数据:一来,人工纠正需耗大量人力时间,无法纠正大容量数据库,尤其当数据的正确取值无从了解时,人工纠错失效;二来各清洗算法至今仍无法确保修复的正确性.

虽然人们希望查询结果唯一确定,但不一致数据的概率本质决定了这样的结果不存在,除非舍弃部分数据.如:基于记录删除的数据清洗策略[1],在查询之前先删除不一致记录,再在"干净的"数据库上计算查询,虽能得到唯一确定的结果,但丢失了部分有价值的信息.另外,Consistent Query Answer over Inconsistent Database(简称CQA)[2]也能计算出唯一确定的查询结果,但它同样丢失了信息.前者丢失的是任意属性上不一致的元组,而后者丢失的是任意查询结果属性上不一致的元组.实际上,信息丢失在相关研究中普遍存在.

实际上,只要能在数据库中把一致的和不一致的分量标识出来,并在查询回答中正确地保持标识,那么无论是一致的还是不一致的信息,只要它符合查询条件,都将保留在查询结果中,信息丢失问题自然不存在.

例 1:若用不同的标记区分图 1 数据库的一致和不一致分量,结果如图 2 所示,并以它为数据源回答 Query 2,则根据查询结果,用户可知该学生的姓名和成绩确定(无标记);而对于其班级,即使无背景知识,也知其不一致.

Student

	SID	SName	SCore	Class
$t1$	1	Sam	561	CSE081
$t2$	1	Sam	561	CSE082
$t3$	2	John	620	EE081
$t4$	3	Eva	586	CIT08
$t5$	4	Lily	543	EE082
$t6$	4	Lily	578	EE082

FDs: SID→SName, Score, Class

Class

	CName	Major	Teacher
$t7$	CSE081	CSE	Judy
$t8$	CSE082	CSE	Judy
$t9$	EE081	EE	Nancy
$t10$	EE081	EE	Philip
$t11$	EE082	EE	Cathy
$t12$	EE082	CEE	Cathy
$t13$	CIT08	CIT	David
$t14$	NE08	NE	David

FDs: Cname→Major, Teacher

Teacher

	Tname	Title	Cellphone
$t15$	Judy	Lecture	5638987
$t16$	Nancy	T.A.	5686437
$t17$	Philip	A.P.	6423535
$t18$	Cathy	Lecture	6449192
$t19$	David	T.A.	6425151

FDs: Tname→Title, Cellphone

Query 1: select SID, SName
From student

Query 2: select SName, Score, Class
From student
where SID=1

Fig.1 An inconsistent relational database and two queries over it

图 1 一个不一致的关系数据库及其上的两个查询

Student′

	SID	SName	SCore	Class
$t1$	1	Sam	561	CSE081*
$t2$	1	Sam	561	EE082*
$t3$	2	John	620	EE081
$t4$	3	Eva	586	CIT08
$t5$	4	Lily	543*	EE082
$t6$	4	Lily	578*	EE082

Query 2 (student′)

SName	SCore	Class
Sam	561	CSE081*
Sam	561	EE082*

Fig.2 Annotated Student and result of Query 2 over it

图 2 标记过的 Student 及其上 Query 2 的查询结果

基于上述考虑,针对不一致关系数据库,AQA[3]通过对不一致数据加标注,并在查询计算中正确地保持标注的语义,使得查询结果可信有意义.与相关工作相比,AQA 具有以下特点:

- 不一致描述细致到属性一级.AQA 并不把包含不一致分量的元组整体标示为不一致,而只标识不一致分量,无标记的分量仍可信,从而在不一致认定的准确性和查询结果的信息保持两方面都有提高.
- 没有信息丢失.AQA 既不修改数据库,又不过滤信息,只对不一致加以描述,不存在信息丢失问题.

文献[3]针对函数依赖下主属性值可靠的不一致关系数据,阐述了 AQA,解决了 3 个层面的问题:不一致关系数据基于标记的表示模型(包括初始检测和查询结果上合法依赖的计算等)、该模型上的查询代数及其正确性

和完备性证明.但现实应用中数据依赖更丰富,除了函数依赖以外,还有包含依赖、域约束等.另外,引起不一致的字段是任意的.以函数依赖为例,设对于 $x\rightarrow y$,元组 $t1$ 和 $t2$ 有 $t1[x]=t2[x]$但 $t1[y]\neq t2[y]$,引起数据不一致的可能是 x 或 y.而问题范围的扩大,对不一致数据检测和其后的查询处理均提出了新要求,这包括:

- 如何计算新增加的各类约束在某查询表达式上的“新生”约束?这些“新生”约束对基于标记的查询结果有哪些影响?是否有必要根据这些“新生”约束,检验查询结果并修改其上的不一致标记?
- 不同的约束对同一元组的同一属性的不一致性认定是否相互矛盾或互有关联?
- 同一分量若违反了不同约束,其上附加的标记是否需体现各具体约束,AQA 上的两类标记是否够用?
- 对于函数依赖,当被依赖属性本身不可靠时,应该如何对违反依赖的数据进行不一致标记?

本文在综合约束(函数依赖、包含依赖和域约束)范围内、不确定属性任意的情况下,扩展了文献[3]提出的AQA,重新审视了 AQA 的数据模型及其上的查询代数.讨论了任意约束在查询结果上的蕴含约束计算问题,并用实例说明了综合约束下不一致数据的初始检测和标记算法.

本文第 1 节是相关工作.第 2 节在综合约束范围内,阐述并扩展不一致数据基于标记的表示模型及其上的查询代数及初始标记算法.第 3 节是相关实验.第 4 节是总结和未来工作展望.

1 相关工作

针对不一致关系数据库上的查询处理,目前有 3 种主流解决方案:第 1 种通过数据清洗,在“干净的”数据库上计算“干净的”查询结果;第 2 种是 CQA;第 3 种是近年来较受关注的基于概率数据库的查询处理.

第 1 种方案试图寻找不一致数据库的最优修复,并以此回答查询,如文献[4–8].实际研究主要关注多项式时间内的启发式修复算法.虽然数据清洗可发现并纠正某些错误,但无法确保不引入脏数据,或会丢失信息.

CQA 试图找出不一致数据库的所有修复,以查询在每个修复上的回答的交集为最终结果.但不一致数据库的修复数量巨大,求解所有修复的时间消耗在指数级.与前者不同的是,CQA 并不改变原数据,其一定范围内的正确性和完备性也得以证明.但 CQA 也存在信息丢失问题.CQA 认为,只要某元组有不一致属性,则整体不一致.在这一点上,它和基于删除的数据清洗是一致的.而 AQA 是不一致数据的弱表示系统,通过区分查询结果中的一致和不一致部分,在不丢失任何信息的前提下,给出有价值的查询结果.

不确定数据管理的方法,有的也能应用到不一致数据管理中,如近年来研究最热的基于概率的可能世界模型[9–12].该方案也不丢失信息,它用概率维来描述记录的不确定,概率一般加在记录级别,各属性在概率上处于均等地位.但实际上,各属性对记录概率的贡献并不同.而在 AQA 中,属性的正确可能性不均等.基于概率的研究着重说明每个查询结果发生的可能性,而 AQA 的重点是尽可能找出查询结果中所有可靠的部分.另外,基于概率数据库的方法存在查询结果概率太小的问题,尤其是涉及多张表的连接查询.

2 带标记的不一致关系数据的查询处理

本节首先介绍基于标记的不一致关系数据模型,然后在综合约束范围内扩展该模型及其上的查询代数以及初始标记算法.我们认为,不一致数据是违反其上任意完整性约束的分量.这里的完整性约束指函数依赖、包含依赖和域约束.对于违反了函数依赖的元组,它在依赖相关的所有属性上的分量都可能不一致.

2.1 相关术语和基本数据模型

AQA 把查询结果的求解分成两步:检测并标注不一致分量,形成标记过的关系;在标记过的数据库中计算查询结果.而在后一步中,标记从输入表到输出表的推演又与约束在查询结果中的推演分不开.下面是基于标记的不一致关系数据模型相关的一些概念.

定义 1(标记过的关系). 给定关系 R 及其上的约束集 F,若 R 中所有对于 F 的不一致分量都标注有至少 1 个标记,则 R 是依据 F 标记过的关系,记作 R^F.类似地,对于 R 上的任何查询 Q,若 $Q(R)$中所有不一致数据都被至少 1 个标记标注了,则 $Q(R)$是一个基于标记的查询结果,记作 $Q(R)^F$.

比如,设图 1 中关系 student 的约束集叫 $F1$,那么图 2 中的关系 student′就是 student^{F1},即所有违反了 $F1$ 的分量值都已正确标记过了.类似地,图 2 中的 Query 2(student)可以用代数表示为$\Pi_{2,3,4}(\sigma_{sid=1}(\text{student}))^{F1}$.

数据的不一致有两类情形:

- 一种是原始数据库中的不一致数据,这类数据的不一致性不会随着查询表达式的不同而不同.
- 另一种则是双目查询中那些在原数据库中一致却在查询结果中不一致的数据,这种数据的不一致性有可能随着另一个查询运算的执行而不存在.如图 3 中 EE081 班的老师手机号,它在 Teacher 中一致,但在 Query 3(Class,Teacher)中因违反函数依赖 CName→Cellphone 而不一致,应该有标记;而在Πcellphone (Query 3(Class,Teacher))中没有违反任何约束,其上的标记又应该取消.前者的标记称为静态不一致标记,后者的标记则称为动态不一致标记.文献[3]没有规定这两类标记,但在实验中,静态标记用符号“*”表示,动态标记则为分量违反的函数依赖的左部属性名,如前述两个可能 Cellphone 的标记都是“R.CName”.

Query 3(Class,Teacher) (表R)

CName	Major	Teacher	Cellphone
CSE081	CSE	Judy	5638987
CSE082	CSE	Judy	5638987
EE081	EE	Nancy*	5686437+
EE081	EE	Philip*	6423535+
EE082	EE *	Cathy	6449192
EE082	CEE *	Cathy	6449192
CIT08	CIT	David	6425151
NE08	NE	David	6425151

Query 3:

```
select Class.*, Cellphone
from Class C, Teacher T
where C.Teacher=T.TName
```

Fig.3 An example of error annotated query result by traditional query evaluation

图 3 一个使用传统查询估值但查询结果一致性被错误标注的例子

在标记过的关系中,元组是数据值和标记的混合.在下文中,符号 t 表示带标记的元组,t_v 表示 t 的值,t^a 表示 t 上的标记,$t^a[X]$表示 t 在属性 X 上的标记列表,$t[X]_v$ 表示 t 在属性 X 上的分量(不包括标记).对于任意标记过的关系中的任意两个元组 $t1$ 和 $t2$:(1) 若 $t1_v=t2_v$,且对于任意属性 X,$t1^a[X]=t2^a[X]$,则 $t1$ 等于 $t2$,记作 $t1=t2$;反之,若 $t1_v=t2_v$ 但 $t1\neq t2$,则 $t1$ 和 $t2$ 值相等,记作 $t1 \underline{v} t2$;(2) 若 $t1_v=t2_v$,且对于任意属性 X,若 $t1[X]$被标注,$t2[X]$必然也被标注,则 $t1$ 带标记包含于(mark-included)$t2$.

从数据模型上看,带标记的不一致关系数据在普通关系模型上对每个分量增加了一个标记维.标记维的一个可行的实现方案是:对于关系 R 的每一个属性 X,在 R 中增加一个属性 XA,以存储任意记录在 X 上的分量的不一致标记.该方案看上去增加了很多空间开销,但实际上,若把属性 XA 定义为 varchar 类型,则多出的空间开销取决于不一致分量的数量.而一般情况下,只有很少量的数据违反了约束,需用标记注明.

定义 2(域等(domain equality,简称 DEQ)). 给定一个数据库模式 D,若其属性 X 和 Y 拥有相同的域和语义,则它们域等.也就是说,如果对于 D 的任意实例中的任意元组 $t1$,都存在某个元组 $t2$,使得 $t1[X]=t2[Y]$,那么属性 X 和 Y 域等,记作 $X \overset{D}{=} Y$.这里,$t1$ 和 $t2$ 可以是相同元组,也可以是不同元组.

比如,在图 1 的数据库中,student.class 和 class.cname 就是域等属性.

根据查询结果上成立的域等关系,可以推导出其上成立的蕴含约束.而蕴含约束是动态标记的依据.

2.2 扩展后的不一致标记

文献[3]研究函数依赖下的不一致数据,只需一对标记.但当约束扩展到多种数据依赖时,不同的约束在蕴含依赖推导中是否相互影响?是否需以不同的静态和动态标记区分分量违反的不同约束?

数据被动态标记的原因是且只能是它违反了只在查询结果上成立的蕴含依赖.而不同类型的约束有不同的蕴含约束计算规则,互不影响.每种约束的蕴含依赖只能在本类型范围内推导.函数依赖的推理规则已经在文

献[3]中阐述了,这里仅说明域约束和包含依赖的推理规则.

设表达式 e 是数据库 R 上的任意查询,$Dom(A)$是属性 $A(A\in R)$上的域约束,$Drv(e(R))$是查询结果 $e(R)$上成立的蕴含约束集,那么:

(1) $\forall A\in e(R), Dom(A)\in Drv(e(R))$;

(2) $\forall A,B\in e(R)$,如果 $A\overset{D}{=}B$,那么 $Dom(B)\in Drv(e(R))$;

(3) $\forall A\in R$,但 $A\notin e(R)$,那么 $Dom(A)\notin Drv(e(R))$.

根据这个推理规则,查询结果中可能存在蕴含的域约束,需要加以检验和标记.本文使用标记"#"为域约束的静态标记,而使用〈与之域等的属性〉作为其动态标记.

另外,设表达式 e 是数据库 R 上的任意查询,$IND(A,B)$是属性 $A,B(A,B\in R)$上的包含依赖 $A\subseteq B$,$Drv(e(R))$是查询结果 $e(R)$上成立的蕴含约束集,那么,

(1) $Drv(R)$:$\forall IND(A,B)\in R, IND(A,B)\in Drv(R)$;$\forall C\in R$,如果 $A\overset{D}{=}C$,则 $IND(C,B)\in Drv(R)$,求闭包.

(2) $Drv(e[X])$(投影):选取 $Drv(e)$中所有满足条件 $X[Y]\overset{D}{=}X[Z]$的域等关系 $Y\overset{D}{=}Z$,以及所有满足条件 $X[Z]\subseteq X[A]$的 $IND(X[Z],X[A])$,根据规则 1)求闭包.这里:X 是将被投影出的所有属性组合;Y 和 Z 是属性或者属性组;$X[Y]$指 Y 在 X 上的投影;同理,$X[Z]$是 Z 在 X 上的投影.以此类推.

(3) $Drv(e[X\overset{D}{=}Y])$(带域等的选择):把 $X\overset{D}{=}Y$,$IND(X,Y)$和 $IND(Y,X)$加入 $Drv(e)$,并求闭包.

(4) $Drv(e1\triangleright\triangleleft e2)$(自然连接):设连接条件是 $X\overset{D}{=}Y$,把 $X\overset{D}{=}Y$,$IND(X,Y)$和 $IND(Y,X)$加入 $Drv(e)$,求闭包.

(5) $Drv(e1\infty e2)$(其他连接)=$Drv((e1)\cup Drv((e2)$.

(6) $Drv(e1\cup e2)$(并):如果域等关系 $X\overset{D}{=}Y$ 既属于 $Drv(e1)$又属于 $Drv(e2)$,那么它属于 $Drv(e1\cup e2)$;同理,如果 $IND(X,Y)$既属于 $Drv(e1)$又属于 $Drv(e2)$,那么 $IND(X,Y)\in Drv(e1\cup e2)$.并计算闭包.

(7) $Drv(e1-e2)$(差)=$Drv(e1)$.

上述计算规则可求出查询结果上成立的所有包含依赖.不难证明,在查询结果中,除了规则(1)中的"新生"包含依赖以外,其他蕴含包含依赖不会改变查询结果的标记.而 $Drv(R)$上的蕴含依赖可以在数据库初始标记中加以检验和标记,因此只要静态标记即可,这里也使用"*"为其静态标记.

2.3 查询处理规则

在一个已经初始标记过的关系上提交用户查询,如何才能得到正确标记过的查询结果?本节给出了综合约束范围内,标记过的不一致数据库上的 7 种基本查询操作的定义.用户提交的任意传统关系代数查询都可以转换为这 7 种查询或其综合表达式,从而得到正确标记过的查询结果.

需要说明的是,若元组 $t1$ 和 $t2$ 违反了函数依赖 $X\rightarrow Y$,同时 $t1[X]$或 $t2[X]$上有不一致标记,因为决定因素不确定,不能根据函数依赖的语义推出被决定因素不确定,这里采用封闭世界假设(closed world assumption),不再重新计算 $t1[Y]$和 $t2[Y]$的不一致标记.

下面是标记过的不一致数据库上的 7 种基本查询操作定义,其中,符号 $\sigma,\Pi,\cup,-,\triangleright\triangleleft$的语义与传统关系代数中的相同,$r$ 和 r'是参与查询操作的关系,$F1$ 和 $F2$ 分别 r 和 r'上成立的约束集,$Q(R)^F$ 是查询 Q 在 R 上的基于标记的查询结果.分量 $t'[A]\leftrightarrow t[A]$,表示存在一个函数依赖 $X\rightarrow A$,使得 $t'[X]=t[X]$但 $t'[A]\neq t[A]$.

- 没有域等的选择查询$\sigma_\omega(r)^{F1}$(R1):这里,ω是选择条件.设 A 是ω中的属性或属性组,$\forall t\in r$,若:(1) $\omega(t_v)$为真;或(2) $\exists t'\in r$,使得 $\omega(t'_v)$ 为真,$t^a[A]\neq\varnothing$且 $t'[A]\leftrightarrow t[A]$;或(3) $t^a[A]\neq\varnothing$且 A 是被依赖属性,则 $t\in\sigma_\omega(r)^{F1}$.
- 带域等的选择 $\sigma_{A\overset{D}{=}B}(r)^{F1}$ (R2):A,B 是 r 的属性,设

$\tau=\{t|(t\in r\wedge(t[A]_v=t[B]_v)\vee(\exists t1(t1\in r\wedge t1[A]_v=t[B]_v\wedge t1[A]\leftrightarrow t[A])\vee(\exists t2(t2\in r\wedge t2[B]_v=t[A]_v\wedge t2[B]\leftrightarrow t[B]))))$.

$\forall x\rightarrow y\in(Drv(\sigma_{A\overset{D}{=}B}(r))-Drv(r)),\forall t1,t2\in\tau(t1\neq t2)$,若 $t1[x]_v=t2[x]_v$ 且有 $t1[y]_v\neq t2[y]_v, t1^a[x]$和 $t2^a[x]=\varnothing$,则在τ中将"$r.x$"加入 $t1^a[y]$和 $t2^a[y]$;同时 $dom(A)/dom(B)\in Drv(\sigma_{A\overset{D}{=}B}(r))-Drv(r),\forall t\in\tau$,若 $t^a[A/B]\neq\varnothing$,且 $t[A/B]_v$ 不满足

$dom(A/B)$,则把“$\langle B\rangle$”/“$\langle A\rangle$”加入 $t^a[A/B]$.由此,$\sigma_{\overset{D}{A=B}}(r)^{F1}=\tau$.

- 投影$\Pi_A(r)^{F1}$(R3):设$\tau=\{t[A]|t\in r\}$:(1) 若 r 是初始表,则$\pi_A(r)=\tau$;(2) 若 r 是查询估值中的临时表,且设 B 是前面某个选择查询的条件属性,$\forall t\in\tau,x\in A$,“Y”$\in t^a[x]$,若 $A\cup B\subseteq r''$(r''是 r 的源表)且 $Y\notin r''$,则在τ中将“Y”和”$\langle Y\rangle$”从$t^a[x]$中移出,并删除等于或者标记包含于其他元组的元组,得到的τ就是$\pi_A(r)$.

- 笛卡尔积$(r\times r')^{F1,F2}$:$(r\times r')^{F1,F2}=\{t_1\overset{\frown}{}t_2\mid t_1\in r\wedge t_2\in r'\}$.

- 连接查询 $(r\underset{A\theta B}{\times}r')^{F1,F2}$ (R4): $(r\underset{A\theta B}{\times}r')^{F1,F2}=\{t_1\overset{\frown}{}t_2\mid t_1\in r\wedge t_2\in r'\wedge((\theta(t_1[A]_v,t_2[B]_v)=true\vee(\exists t(t\in r'\wedge\theta(t_1[A]_v,t[B]_v)=true\wedge t2[B]\leftrightarrow t'[B])\vee(\exists t'(t'\in r\wedge\theta(t'[A]_v,t_2[B]_v)=true\wedge t1[A]\leftrightarrow t[A]))\vee t1^a[A]\neq\varnothing$且 A 是被依赖属性$\vee t2^a[B]\neq\varnothing$且 B 是被依赖属性$\}$,这里,A 和 B 分别是 r 和 r'的属性,θ是比较数值.

显然,$(r\underset{A\theta B}{\times}r')^{F1,F2}=\sigma_{A\theta B}(r\times r')^{F1,F2}$.

- 自然连接$(r\triangleright\triangleleft r')^{F1,F2}$(R5):设 r 和 r'的公共属性为 A,$\tau=\{\widehat{t[\overline{A}]s}\mid t\in r\wedge s\in r'\wedge(t[A]_v=s[A]_v\vee(\exists t1(t1\in r'\wedge t[A]_v=t1[A]_v\wedge t1[A]\leftrightarrow s[A]\vee(\exists t2(t2\in r\wedge t2[A]_v=s[A]_v\wedge t2[A]\leftrightarrow t[A])))$,且$\forall x\rightarrow y\in(Drv(r\triangleright\triangleleft r')-(Drv(r)\cup Drv(r')))$,$\forall t1,t2\in\tau$,若$t1[x]_v=t2[x]_v$ 但 $t1[y]_v\neq t2[y]_v$ 且 $t1[x]^a$ 和 $t2[x]^a=\varnothing$,则将标记“$\tau.x$”加入 $t1^a[y]$和 $t2^a[y]$.同时,$dom(r.A)/dom(r'.A)\in(Drv(r\triangleright\triangleleft r')-(Drv(r)\cup Drv(r')))$,$\forall t\in\tau$,若“#”和“$\langle x\rangle$”$\notin t^a[r.A/r'.A]$,且$t[A/B]_v$不满足$dom(r.A)/dom(r'.A)$,则把“$\langle B\rangle/\langle A\rangle$”加入 $t^a[r.A/r'.A]$,$(r\triangleright\triangleleft r')^{F1,F2}=\tau$.

类似地,$(r\triangleright\triangleleft r')^{F1,F2}=\sigma_{\overset{D}{A=B}}(r\times r')^{F1}$.

- 并$(r\cup r')^{F1,F2}$(R6):(1) 将 r 和 r'并入 r'',并删除 r''中等于或者标记包含于其他元组的元组;(2) $\forall x\rightarrow y\in Drv(r\cup r')$,$\forall t1,t2\in r''(t1\neq t2)$,如果 $t1[x]_v=t2[x]_v$但 $t1[y]_v\neq t2[y]_v$ 且 $t1^a[x]$和 $t2^a[x]\neq\varnothing$,把标记“*”加入空的 $t1^a[y]$和空的 $t2^a[y]$中.$(r\cup r')^{F1,F2}=r''$.

单张表上一致的信息可能在合并以后不一致,因此,标记过的关系上的并操作不仅需要从值的角度合并两张表,还要重新验证所有记录的一致性,但不追加被依赖属性上不一致的记录的标记.

- 差$(r-r')^{F1,F2}$(R7):设$\tau=\{t|t_v\in r_v\cap r'_v$ 且 r'中的 t^a 不为空$\}$,$\omega=\{t|t\in r$ 且$(\exists t'\in\tau,\exists x\rightarrow y\in F1$,使得 $t[x]_v=t'[x]_v$ 但 $t[y]_v\neq t'[y]_v$ 且 $t'^a[x]\neq\varnothing)\}$,那么$(r-r')^{F1,F2}=\{t|t_v\in r_v$ 且 $t_v\notin r'_v$ 且 $t\notin\omega\}$.

2.4 初始标记面临的新问题

第 2.3 节说明了如何在一个带标记的关系上计算带标记的查询结果.本节主要关注如何在普通关系表上加标记,使其成为带标记的关系.也就是说,对于任意给定的关系R及其上的约束集,如何判断R的哪些分量不确定.本文认为这些不确定分量不一致.

文献[3]在问题范围内,初始标记算法简单、直接.但显然,在综合约束范围内,可信属性值的判断更困难.而且,当被依赖属性也可能不确定时,函数依赖在不一致检测方面就失去其原有作用.文献[3]的初始标记算法在问题范围扩大后不可行,哪怕只有函数依赖一种约束.

尽管本文不能根据约束断定不可信分量,但违反了数据依赖的记录仍有错误.这些记录不管错在哪,总会卷入某种冲突.可通过增加、修改或删除记录来纠正这些冲突.如图 4 中,$t0$ 和 $t4$ 是个冲突对,它们违反了约束 phno→name 和 phno→street.究其原因,可能是它们中有一个的 phno 错了,也可能是它们中有一个的 name 和 street 错了.相应地,可以有 6 种纠正该错误的方案.但不同方案的修复代价不同,且某些方案可能同时纠正多个冲突,而有的方案又可能引发新冲突.可以认为,修复代价越小、能纠正的冲突数量越多、引发新冲突越少的方案,其操作对象错误的可能性越大.本文借鉴文献[13]中的不一致 XML 文档启发式修复算法,通过寻找所有冲突的最优解决方案,比较其修复代价来判定记录的不确定分量.

本文的前期工作[14]已详细阐述了这节要说明的初始标记算法.但初始标记算法是不一致数据查询处理不可缺少的一部分,为了论文的完整性,接下来通过一个完整的例子来说明该算法及其相关概念.

Cust

	phno	Name	Street	City	State	Zip	*wt*
t_0	949-1212	Alice Smith	17 bridge	Midville	AZ	05211	2
t_1	555-8145	Bob Jones	5 valley rd	Centre	NY	10012	2
t_2	555-8195	Bob Jones	5 valley rd	Centre	NJ	10012	1
t_3	212-6040	Carol Black	9 mountain	Davis	CA	07912	1
t_4	949-1212	Ali Stith	27 bridge	Midville	AZ	05211	1

Equip

	phno	serno	eqmfct	eqmodel	instdate	*wt*
t_5	949-1212	AC13006	AC	XE5000	Mar-02	2
t_6	555-8145	L55001	LU	ze400	Jan-03	2
t_7	555-8195	L55011	LU	ze400	Mar-03	1
t_8	555-8195	AC22350	AC	XE5000	Feb-99	1
t_9	949-2212	L32400	LU	ze400	Oct-01	1

IND: ic1) equip[phno]⊆cust[phno]
ic2) cust[phno]→cust[name]
ic3) cust[phno]→cust[street]
FD: ic4) cust[phno]→cust[city]
ic5) cust[phno]→cust[state]
ic6) cust[phno]→cust[zip]
ic7) cust[zip]→cust[city]
ic8) cust[zip]→cust[state]
ic9) cust[name,street,zip]→cust[phno]
ic10) equip[serno]→equip[phno]
ic11) equip[serno]→equip[eqmfct]
ic12) equip[serno]→equip[eqmodel]
ic13) equip[serno]→equip[instdate]
ic14) equip[phno,eqmfct,eqmodel]→equip[serno]
DD: ic15) cust[state]≠'NJ'

Fig.4 An inconsistent database modified from Ref.[4] and integrity constraints over it

图 4 修改自文献[4]的一个不一致数据库及其上的完整性约束

2.4.1 相关概念及代价模型

首先,本文将违反了约束的元组或元组集组成的原子单位称为违反包.如图 4 中,违反包 $C1=\{(t9),ic1)\}$表明,存在一条记录 $t9$ 违反了约束 ic1;而违反包 $C2=\{(t0,t4),ic2)\}$则表明,$t0$ 和 $t4$ 共同违反了 ic2.另外,图 4 中还存在着下列违反包:$C3=\{(t0,t4),ic3)\}$,$C4=\{(t1,t2),ic8)\}$,$C5=\{(t1,t2),ic9)\}$和 $C6=\{(t2),ic15)\}$.

函数依赖的违反包包含 1 组记录,包含依赖和域约束的违反包则包含 1 条记录.

违反包是修复的对象.每个违反包都可通过若干种修复方案得到修复.修复方案由修复对象上的系列操作组成:插入、删除记录、或修改记录的属性值.它们都是该违反包的候选修复方案,其目标值可以是变量.

如,违反包 $C4$ 可通过 6 个候选修复得到修复(见表 1 的 $r16$~$r21$).它们分别是:(1) 修改 $t1$[state]为'NJ';(2) 修改 $t2$[state]为'NY';(3) 修改 $t1$[zip]为变量 x,且 $x\neq$'10012';(4) 修改 $t2$[zip]为 x,且 $x\neq$'10012';(5) 删除 $t1$;(6) 删除 $t2$.

其次,违反包 C 的候选修复 r 的修复代价被定义为 $RC(r)=\sum\lambda\times\Delta(t,t')$,其中,$t$ 是 C 中的任意记录;λ是 t 的可信度,由用户输入;t'是 t 的修复;$\Delta(t,t')$是 t 和 t'不同分量的个数.

比如,设图 4 任意记录的可信度都为 1.对于违反包 $C1$,$t9$[phno]在 cust[phno]中不存在,违反了 ic1,可通过下列候选修复之一得到修正:(1) 修改 $t9$[phno],使其满足约束;(2) 删除 $t9$;(3) 在 cust 中插入一条记录 t',使得 t'[phno]='949-2212'.但候选修复:(1) 只要修改一个分量,修复代价为 1×1=1;(2) 需修改 5 个分量,代价为 5×1=5;(3) 需修改 6 个分量,代价为 6×2=12.类似地,违反包 $C4$~$C6$ 的候选修复及其修复代价见表 1.

Table 1 Repair candidate of conflict class $C4$~$C6$ and its cost

表 1 违反包 $C4$~$C6$ 的候选修复及修复代价

违反包	修复编号	候选修复	修复代价
$C4$	$r16$	$t1$[state]=x (x='NJ')	2
	$r17$	$t2$[state]=x (x='NY')	1
	$r18$	$t1$[zip]=x ($x\neq$'10012')	2
	$r19$	$t2$[zip]=x ($x\neq$'10012')	1
	$r20$	删除 $t1$	12
	$r21$	删除 $t2$	6
$C5$	$r22$	$t1$[phno]=x (x='555-8195')	2
	$r23$	$t2$[phno]=x (x='555-8145')	1
	$r24$	$t1$[name]=$x1$ && $t1$[street]=$x2$ && $t1$[city]=$x3$ ($x1\neq$'Bob Jones', $x2\neq$'5 valley rd', $x3\neq$'centre')	6
	$r25$	$t2$[name]=$x1$ && $t2$[street]=$x2$ && $t2$[city]=$x3$ ($x1\neq$'Bob Jones', $x2\neq$'5 valley rd', $x3\neq$'centre')	3
	$r26$	删除 $t1$	12
	$r27$	删除 $t2$	6
$C6$	$r28$	$t2$[state]=x ($x\neq$'NJ')	1
	$r29$	删除 $t2$	6

违反包的修复之间可能相互包含.如一个候选修复要求删除某元组 t,另一个则要修改 t 的某属性,那么它们可以合并为前者.设两个候选修复都包含对属性值 $t[A]$的修改,则可使用区间合并的方法进一步确定 $t[A]$的取值范围.在整个不一致数据的修复中,若两个候选修复在区间合并中矛盾,则摒弃该修复.

一般情况下,修复代价小的候选修复对象比代价大的候选修复对象错误的可能性更高.但须注意:有的候选修复不仅能修复本违反包,还能同时修复其他违反包,如在表 1 中,r17 不但可以修复 $C4$,也可以修复 $C6$,即 r17 同时覆盖了 $C4$ 和 $C6$.由此可以认为,$t2$[state]错的可能性比 $t1$[state]要大.

这里,我们把候选修复 r 的修复代价 $RC(r)$和 r 能修复的违反包个数的商称为 r 的覆盖修复代价.

假设一共有 m 个违反包,它们共有 n 个候选修复,可以用数组 $a[n][m]$来记录它们之间的覆盖关系.$a[i][j]=0$,说明候选修复 i 不能覆盖违约包 j,反之覆盖.表 1 所列候选修复的覆盖修复代价见表 2.

Table 2 Cover cost and related cover cost for repair candidate in table 1

表 2 表 1 候选修复的覆盖和关联覆盖修复代价

	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	修复代价	覆盖代价	关联覆盖代价
$r16$	0	0	0	1	0	0	2	2	2
$r17$	0	0	0	1	0	1	1	0.5	0.5
$r18$	0	0	0	1	0	0	2	2	3
$r19$	0	0	0	1	0	0	1	1	2
$r20$	0	0	0	1	1	0	12	6	7
$r21$	0	0	0	1	1	1	6	2	4
$r22$	0	0	0	0	1	0	2	2	3
$r23$	0	0	0	0	1	0	1	1	3
$r24$	0	0	0	0	1	0	6	6	6
$r25$	0	0	0	0	1	0	3	3	3
$r26$	0	0	0	1	1	0	12	6	7
$r27$	0	0	0	1	1	1	6	2	4
$r28$	0	0	0	0	0	1	1	1	1
$r29$	0	0	0	1	1	1	6	2	4

某候选修复的覆盖代价越小,说明为其修复对象错误的可能性越大,应予以纠正.不一致数据库的修复是所有违反包的修复组合.候选修复的代价之和可以成为衡量整个数据库修复方案好坏的标准.同时,一个违反包的修复可能导致多个违反包的出现.因此,还要考虑其引发的违反包修复代价.

设候选修复 r 的覆盖修复代价为 $CC(r)$,由 r 而新增加的所有违反包的最小修复代价之和为 $AC(r)$,那么候选修复 r 的关联覆盖修复代价 $RCC(r)=CC(r)+AC(r)$.

表 1 的所有候选修复的关联覆盖修复代价见表 2,如对 $r23$,假设 $t2$[phno]的值修改为“515-8145”,那么在 equip 中,$t7$ 和 $t8$ 将违反包含依赖,其最小修复代价为 2.因此,$r23$ 的关联修复代价为 3.

2.4.2 初始标记算法

根据前一节,在一个违反包的众多候选修复中,关联修复代价最小的那个修复对象导致错的可能性最大,应予以标记.进一步来说,初始标记就是要为给定数据库启发式地选择一个关联覆盖修复代价最小的候选方案集,并对其中每个候选方案的修复对象加上静态标记.由于初始标记的主要目的不是修复所有错误,因此在计算关联覆盖修复代价时只考虑修复引发的一级违反包.另外,错误明显的违反包可直接修复而不附上标记.即,若候选修复的关联覆盖修复代价在该违反包中最小且小于用户给定的参数κ(κ<1),且修复目标值确定,则可以直接按照它去修复违反包.只有不能直接修复的最可能出错的分量或记录才被附上标记.

算法. 初始标记算法(GreedyRepair&Mark).

输入:不一致数据 R、R 上的约束集 Ics、修复阈值κ.

输出:带标记的 R.

1. 规约违反包栈 CC,并计算每个违反包的候选修复及其代价(栈 RP)
2. $RelateCover(RP)$;
3. /*选择 $RCost+AC$ 最小且小于κ且不与已选修复冲突的候选修复,若存在多个代价相同的候选修复,则随机选择一个,推入 $FRepair$,同时,其他所有覆盖该违反包的候选修复不再是它的候选修复.置相

应的 $a[i][j]$为 0,并从中删除其中只覆盖 CC'的候选修复,重新计算 $RC[i].t$. */

If $((rp.RCost+rp.AC)<\kappa$ && $\neg\exists rp'((rp'.RCost+rp'.AC)<rp.RCost+rp.AC)$ && $\neg\exists rp''(rp''\in FRepair$ && $rp''\nleftrightarrow rp))$

{push (min(rp),$FRepair$);

$a[m][n]=0$(m 任意,n 使得 $a[m'][n]=1$,m'为 rp 所在的行);

在 3 个冲突类栈中删除满足下面条件的违反包δi 及其修复:$a[m][i]=0$(m 任意);

//重新计算各候选修复的代价,已删除的 $RC[i]$不算

$rp.RCost= rp.\omega/\sum a[i][j]$ (i 是 rp 的序号,$0\leqslant j<n$)}

4. 重复步骤 3,直到候选修复栈为空,或没有符合条件的候选修复为止
5. for each rp in $FRepair$

   根据 rp,修复数据库
6. for each 违反包δ in CC

   如果δ存在关联覆盖代价最小的修复 rp

   {如果 rp 是删除记录 t,那么在 t 的所有属性值上都附加不一致标记“*”;

   如果 rp 是修改属性值 $t[A]$且违反的约束是函数依赖,那么在 $t[A]$上附加不一致标记“*”,且在子表的相应属性值上也附加不一致标记“*”;

   如果 rp 是修改属性值 $t[A]$且违反的约束是域约束,那么在 $t[A]$上附加不一致标记“#”,并且在子表的相应属性值上也附加不一致标记“#”;}

   否则

   在违反包内的所有记录的任意属性上附加不一致标记“*”

上面给出的 GreedyRepair&Mark 算法,即初始标记算法,由 4 部分组成:

- 首先,根据给定约束,按其语义找出数据库的所有违反包,并为每个违反包计算其所有候选修复,及每个候选修复的修复代价.
- 其次,判断候选修复和违反包之间的覆盖关系,初始化 $a[i][j]$.如果修复 i 能覆盖违反包 j,那么 $a[i][j]=1$,否则 $a[i][j]=0$.并计算每个候选修复的覆盖修复代价和关联覆盖修复代价.
- 然后,启发式地在所有未处理候选修复中选择关联覆盖修复代价最小、小于κ,且不与已采纳修复冲突的候选修复,推入已采纳修复队列中.同时,把它能覆盖的违反包从待处理违反包集中删除,并更新剩余候选修复的关联覆盖代价和 $a[i][j]$,进行下一轮选择,直到无符合条件的候选修复存在或违反包集为空为止.再在已采纳候选修复列表中合并同一对象上的候选修复,并缩小修复变量的取值范围.若某个修复变量值确定,则按它修复数据库;否则,把它覆盖的违反包推回违反包集合.
- 最后,对剩下的违反包,启发式选择关联覆盖修复代价最小的 1 个或几个候选修复,将其修复对象加上静态标记.若没有代价最小的候选修复,则把违反包中的所有记录都标记为不一致.

需要说明的是,根据修复对象的不同,不一致标记的具体做法也有所不同:

- 若候选修复是修改某分量值,则在该分量上附加静态标记;
- 若候选修复是删除某元组,则在该元组的所有分量上都附加静态标记;
- 若候选修复是插入一条记录,则在新插入元组的所有分量上都附加静态标记.

不难分析,算法的复杂度在候选修复数量的平方级.

假设修复阈值为 0.5,那么启发式修复和初始标记之后,图 4 的数据库变成了图 5 所示,其中,$t4$[phno]和 $t2$[state]已经修复为图中灰色标注的值,这两个值的纠正修复了 $C1$,$C2$,$C3$,$C4$ 和 $C6$ 这 5 个违反包;而在 $C5$ 的候选修复中,因为 $r22$ 和 $r23$ 的关联覆盖修复代价并列最小,因此将 $t1$[phno]和 $t2$[phno]同时标记为不一致.

Cust

	phno	Name	Street	City	State	Zip	*wt*
t_0	949-1212	Alice Smith	17 bridge	Midville	AZ	05211	2
t_1	555-8145*	Bob Jones	5 valley rd	Centre	NY	10012	2
t_2	555-8195*	Bob Jones	5 valley rd	Centre	NY	10012	1
t_3	212-6040	Carol Black	9 mountain	Davis	CA	07912	1
t_4	949-2212	Ali Stith	27 bridge	Midville	AZ	05211	1

Equip

	phno	serno	eqmfct	eqmodel	instdate	*wt*
t_5	949-1212	AC13006	AC	XE5000	Mar-02	2
t_6	555-8145	L55001	LU	ze400	Jan-03	2
t_7	555-8195	L55011	LU	ze400	Mar-03	1
t_8	555-8195	AC22350	AC	XE5000	Feb-99	1
t_9	949-2212	L32400	LU	ze400	Oct-01	1

Fig.5 Basic marked database of Fig.4

图 5 初始标记后的图 4 数据库

3 实 验

3.1 初始标记的实验性能

- 实验环境:下面的所有实验都在如下配置中完成:Intel Celeron 420 2.0GHZ CPU,1GB 内存,windows XP+SP2 操作系统.编程语言是 C#,DBMS 是 SQL Server 2000.
- 实验数据:实验以 TPC-H 数据库为基础,该数据库每张表的初始记录数见表 3.数据库上的约束设置除了遵从 TPC-H2.9.0 的说明文档[15]以外,还加了两个域约束:p_brand≠“00000”和 c_address≠“yizhangdengzi”.综上,共有 50 个函数依赖、9 个包含依赖和 2 个域约束.实验在数据库中人为地加入噪声数据,使其不一致.根据噪声数据的不同,有 4 个数据源.它们都按主码和外码建立了索引,其.mdf 文件大小都约为 4GB.
- 噪声数据:实验以 tpch 数据库为基础,对表 3 所述前 6 张表的每一张,按一定比例,将其 top *X* 个记录复制到新表,在新表中修改其在指定字段(表 3 的冲突字段)上的值,使其与原来不同,再把修改后的记录插回原表中,产生不一致.对于 p_brand 和 c_address,实验有意识地使其违反域约束.具体违反包数量见表 3.

Table 3 Noise data in four TPC-H databases

表 3 4 个 TPC-H 数据库的噪声数据

表	Tpch		Tpch 1		Tpch 2		Tpch 3		Tpch 4	
	记录(万)	冲突字段	违反 FD	违反 DD	违反 FD	违反 DD	违反 FD	违反 DD	违反 FD	违反 DD
s	1	s_nationkey	72	0	144	0	216	0	288	0
ps	80	ps_supplycost	0	0	0	0	0	0	0	0
p	20	p_brand	240	120	480	120	720	240	960	360
o	150	o_custkey	400	0	800	0	1 200	0	1 600	0
c	15	c_address	120	60	240	60	360	120	480	180
l	600	l_quanlity, l_shipdate	0	0	0	0	0	0	0	0
n	25 条		0	0	0	0	0	0	0	0
r	5 条		0	0	0	0	0	0	0	0
违反包含依赖			30		45		68		108	

- 启发式算法的性能:启发式算法的时间消耗和违反包数量关系很大.事实上,启发式算法的时间主要和候选修复数量有关,而候选修复的多少又取决于违反包的数量和约束类别.图 6 表明,初始标记的时间在违反包数量的平方级别.主要性能瓶颈在内存限制,整个启发式计算中,若在内存存储候选修复,则算法直接受候选修复数量的影响,而即使在外存存放候选修复,因算法需在内存中维持数组 *a*,内存仍是其性能瓶颈.

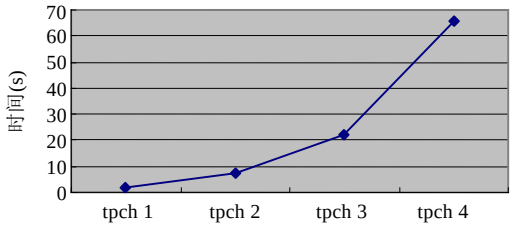


Fig.6 Basic marking time of four TPCH databases

图 6 4 个不一致数据库上的初始标记时间

3.2 查询性能分析

- 实验数据:以下实验数据采用文献[3]中的 8 个人工数据库,它们都遵从图 1 的数据库模式,预先都在主码上建了索引,具体参见表 4.其中,*ds* 是数据库大小(单位 GB);*dr* 是脏数据比例,*dr*=不一致分量数/所有分量数.

Table 4 Generated data used in the experiments

表 4 实验中用到的人工数据

	数据源名称	*ds*	*Dr* (%)	标记后多增加的空间(MB)
组 1	DB11	1GB	1	0.552
	DB12	1GB	5	2.714
	DB13	1GB	10	5.546
	DB14	1GB	15	8.233
组 2	DB21	106MB	5	0.316
	DB22	536MB	5	1.461
	DB23	1GB	5	2.714
	DB24	1.5GB	5	4.378

- 实验用查询:实验用查询共 11 个.查询 *q*1~*q*6 都是单表上的 SPJ 查询,*q*7 是一个嵌套查询,*q*8 是两张表之间的自然连接查询,*q*9 是 3 张表之间的自然连接查询,*q*10 是并查询,*q*11 是差查询.

*q*1: select cname, major
from class

*q*2: select cname, major
from class
where cname=‘c150’ or major=‘m53193’

*q*3: select major,cname
from class
where cname=‘c100’ and major=‘m13276’ and teacher>=‘t1000’

*q*4: select *
from class
where cname like ‘c1000%’

*q*5: select *
from class
where cname is null

*q*6: select major, cname
from class
where cname=‘c100’
and major>=‘m31000’
or major<=‘m100’

*q*7: select *
from class
where teacher in (select teacher.tname
from teacher
where ttitle='Lecture')

*q*8: select cname, major, teacher, ttitle, cellphone
from class, teacher
where cname='c1000'
and class.teacher=teacher.tname

*q*9: select *
from student,class,teacher
where student.class=cname
and teacher=teacher.tname and cname='c200'

*q*10: select cname, major
from class where cname='c10'
union
select cname, major from class where cname='c100'

*q*11: select cname, major from class where cname in ('c10', 'c100')
minus
select cname, major from class where cname='c10'

- 不一致标记的存储:对于任何属性 X,实验添加属性 XA,以存储记录在 X 上的不一致标记.
- AQA 查询实现方法及其优化:AQA 至少有两种实现方法:
  - 一是改写传统 RDBMS 的查询处理模块,按第 2.3 节给出的 7 种查询扩展传统的$\sigma,\Pi,\cup,-,\triangleright\triangleleft$运算,并定义与之匹配的查询语言.
  - 二是在现有数据库的基础上进行查询改写:首先扩展现有数据库结构,使其能够容纳标记;再对任意用户给定的普通 SQL 查询,通过映射模块,将其翻译成能够正确计算查询结果及其标记的 SQL 查询.接下来,所有实验都采用查询重写方法实现.

例 2:查询 Query 4 可严格按第 2.3 节 R1~R7 这 7 种查询定义,重写为一组 3 个级别的查询 Query 4-1,Query 4-2 和 Query 4-3,返回基于标记的查询结果:

Query 4:
Select CName, Cellphone
From class, teacher
Where class.teacher=teacher.TName and Major="EE"

Query 4-1:
Select *c*1.*, *t*1.*
Into tmpR1
From Class *c*1, Teacher *t*1
Where (Class.TeacherA is not null or Teacher.TnameA is not null
Or Tname=any (select teacher from Class *c*2 where CName=*c*1.CName))
and 1<(select *count*(*) from Class *C* where *C*.CName=Class.CName)
and *c*1.CNameA is not null;

Query 4-2:

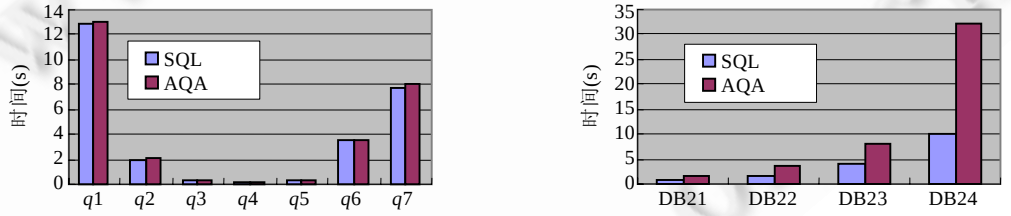
```
update T
set T.CellphoneA=T.CellphoneA+'Class.CName'
from tmpR1 T
where exists (select B.CName from tmpR1 B        where T.CName=B.CName
    group by B.CName      having count(distinct b.Cellphone)>1);
```

Query 4-3:

```
select CName, CNameA, Cellphone, CellphoneA, Major, MajorA, Teacher, TeacherA
from tmpR1
where major='EE' or (CName in (select CName From Class Where Major='EE') and MajorA is not null)
Union
select CName, CNameA, Cellphone, CellphoneA, Major, MajorA, Teacher, TeacherA
from Class, Teacher
where (Class.TeacherA is not null or Teacher.TnameA is not null
        Or Tname=any (select teacher from Class c2 where CName=c1.CName))
    and (1=(select count(*) from Class C where C.CName=Class.CName) or CNameA is Null)
    and (major='EE' or (CName in (select CName From Class Where Major='EE') and MajorA is not
        null))
```

其中,第 1 组查询只有 1 个,用于查询可能连接大表中所有可能违反蕴含约束的记录;第 2 组查询可有多个,对每个蕴含依赖产生一个更新语句,更新违反了该蕴含依赖的记录的标记;第 3 组查询也只有 1 个,用于查找符合条件的标记后的记录,并返回最终结果.

- AQA 查询性能:为了测试查询重写的性能,实验在相同数据源上比较 AQA 及其相应 SQL 查询的时间性能.图 7(a)表明,当无蕴含依赖时,AQA 的时间消耗与 SQL 相差无几.但对存在蕴含依赖的连接查询来说,如图 7(b)所示,AQA 比 SQL 更耗时.其原因是 SQL 查询无须连接所有记录,并计算不一致标记,但 AQA 却需对每个蕴含依赖进行一次全表扫描.这种时间差别会随着数据库的增加和参与连接的表数量的增加而增加.



(a) 非连接查询的 AQA 和 SQL 在 DB14 上的时间性能　　(b) *ds* 不同时查询 *q*9 的 AQA 和 SQL 的时间性能

Fig.7　Time performance comparison of AQA and SQL

图 7　AQA 和 SQL 的时间性能比较

为了提高 AQA 的性能,本文优化了 *q*8 和 *q*9,把第 1 组查询修改为查找可能违反蕴含约束且可能满足查询条件的记录.优化后减少了因"可能世界"太大而引起的全表扫描和不必要的不一致检验和标记修改操作.图 8 和图 9 表明,调整以后,无论数据源怎样变化,连接查询的 AQA 和普通 SQL 的时间性能都非常接近.

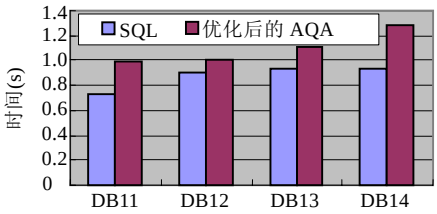


Fig.8 Time performance of *q*8 with optimized AQA and SQL when *dr* differs

图 8 *dr* 不同时,*q*8 的优化后 AQA 和 SQL 的时间性能

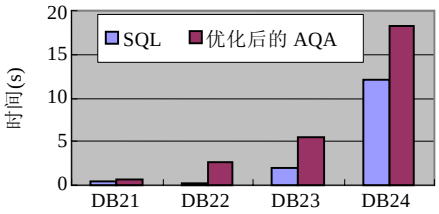


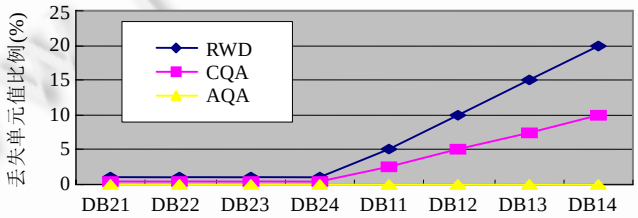
Fig.9 Time performance of *q*9 with optimized AQA and SQL when *ds* differs

图 9 *ds* 不同时,*q*9 的优化后 AQA 和 SQL 的时间性能

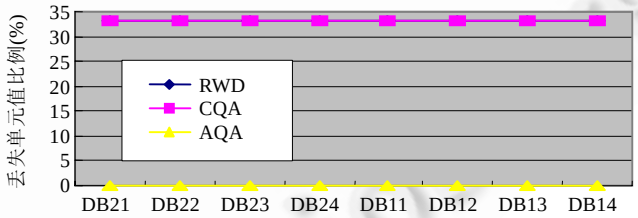
- 信息保持方面的性能:实验比较了 AQA 和主要相关方案——基于元组删除的数据清洗方法(RWD)和 CQA 在信息保持方面的性能.实验分别对 *q*4 和 *q*7,比较它们在信息保持方面的性能.信息丢失比例的计算方法为

所有满足查询条件但丢失的属性值/所有满足查询条件的属性值.

比如在 DB12 中,如果不考虑不一致性,共有 4 条记录满足 *q*7 的查询条件,但其中两条记录在 Major 上不一致但在 CName 和 Teacher 上一致,根据 RWD 的计算方法,这两条记录(共 4 个属性值)将不会出现在查询结果中.所有满足查询的 4 条记录共有 12 个属性值,信息丢失比例为 4/12=32.3%.图 10(a)和图 10(b)表明,RWD 和 CQA 在信息保持方面的能力因查询的不同而变化很大,但 AQA 在任何情况下都不会丢失信息.



(a) 3 种方法下查询 *q*4 的信息丢失比例



(b) 3 种方法下查询 *q*7 的信息丢失比例

Fig.10 Performance comparison of RWD, CQA and AQA on information preserve

图 10 RWD,CQA 和 AQA 在信息保持方面的性能比较

- 扩展后的 AQA 性能:本文在 tpch 4 上,采用 TPC-H 查询,对扩展前后的 AQA 时间性能作了比较,结果如图 11 所示.扩展前后的 AQA 时间性能相差不大,且扩展后的时间消耗还略少于扩展前.这是因为扩展后无须修改那些在被依赖属性上不一致记录的标记,而扩展后需检验的域约束在查询结果上的蕴含约束又为空.

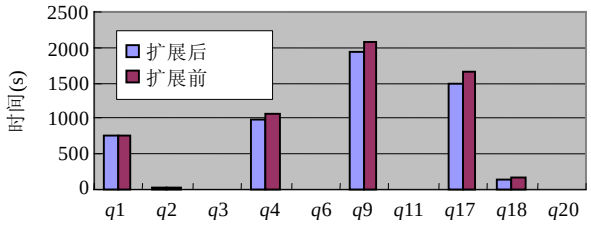


Fig.11 Time performance of unextended AQA (only FD) and extended AQA (multi IC) over tpch 4

图 11 tpch 4 数据库上扩展前(只有 FD)和扩展后(多种依赖)AQA 的时间性能

4 总结和未来展望

在现实应用中,数据常违反它应遵从的语义约束,特别是在数据整合和数据交换日益频繁的今天.如何回答这类数据库上的查询、保证查询结果的可信度并不丢失信息,是一个亟需解决也具有实际意义的问题.

针对这一问题,本文在文献[3]提出的 AQA 的基础上,把这一方案从单一的函数依赖扩展到了综合约束范围内.文中扩展了 AQA 的数据模型,重新定义了标记过的关系上的 7 种基本的代数查询及其估值规则.

目前,在关系数据库领域,本文只解决了非统计查询的基于标记的查询结果计算问题.未来将针对统计查询,计算其基于标记的查询结果.另外,把这一研究方案扩展到 XML 领域,也是未来的工作之一.

References:

[1] Kolahi S, Lakshmanan LVS. On approximating optimum repairs for functional dependency violations. In: Fagin R, ed. Proc. of the 12th Int'l Conf. on Database Theory (ICDT 2009). New York: ACM Press, 2009. 53−62. [doi: 10.1.1.169.8322]

[2] Arenas M, Bertossi LE, Chomicki J. Consistent query answers in inconsistent databases. In: Proc. of the 18th ACM Symp. on Principles of Database Systems (PODS'99). New York: ACM Press, 1999. 68−79. [doi: 10.1.1.14.2369]

[3] Wu AH, Tan ZJ, Wang W. Annotation based query answer over inconsistent database. Journal of Computer Science and Technology (JCST), 2010,25(3):467−479. [doi: 10.1007/s11390-010-9338-9]

[4] Bohannon P, Fan WF, Flaster M. A cost-based model and effective heuristic for repairing constraints by value modification. In: Özcan F, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2005). New York: ACM Press, 2005. 143−154.

[5] Lopatenko A, Bravo L. Efficient approximation algorithms for repairing inconsistent databases. In: Proc. of the 23th Int'l Conf. on Data Engineering (ICDE 2007). Washington: IEEE Computer Society, 2007. 216−225. [doi: 10.1109/ICDE.2007.367867]

[6] Wijsen J. Making more out of an inconsistent database. In: Atzeni P, Caplinskas A, Jaakkola H, eds. Proc. of the 8th Advances in Databases and Information Systems. LNCS 3255, Heidelberg: Springer-Verlag, 2004. 291−305. [doi: 10.1.1.60.998]

[7] Cong G, Fan WF, Geerts F, Jia XB, Ma S. Improving data quality: Consistency and accuracy. In: Koch C, *et al*., eds. Proc. of the 33rd Int'l Conf. on Very Large Databases (VLDB 2007). New York: ACM Press, 2007. 315−326. [doi: 10.1.1.109.1550]

[8] Andritsos P, Fuxman A, Miller RJ. Clean answers over dirty databases: A probabilistic approach. In: Liu L, *et al*., eds. Proc. of the 22nd Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2006. 30. [doi: 10.1109/ICDE.2006.35]

[9] Wang DZ, Michelakis E, Garofalakis M, Hellerstein JM. BayesStore: Managing large, uncertain data repositories with probabilistic graphical models. Proc. of the VLDB Endowment, 2008,1(1):340−351. [doi: 10.1.1.140.6348]

[10] Cheng R, Singh S, Prabhakar S. U-DBMS: A database system for managing constantly-evolving data. In: Böhm K, *et al*., eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB 2005). New York: ACM Press, 2005. 1271−1274. [doi: 10.1.1.97.1967]

[11] Antova L, Koch C, Olteanu D. MayBMS: Managing incomplete information with probabilistic world-set decompositions. In: Proc. of the 23th Int'l Conf. on Data Engineering (ICDE 2007). Washington: IEEE Computer Society, 2007. 1479−1480.

[12] Zhou AY, Jin CQ, Wang GR, Li JZ. A survey on the management of uncertain data. Chinese Journal of Computers, 2009,32(1): 1−16 (in Chinese with English abstract).

[13] Wu AH, Wang XS, Tan ZJ, Wang W. A cost-based heuristic algorithm for repairing inconsistent XML document. Journal of Software, 2009,20(4):918−929 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3225.htm [doi: 10.3724/SP.J.1001.2009.03225]

[14] Wu AH. Annotating algorithm for inconsistent initial relational database. Computer Research and Development, 2010,47(Suppl.): 208−214 (in Chinese with English abstract).

[15] T. P. C. (TPC). TPC benchmark H: Standard specification. 2009. http://www.tpc.org/tpch

附中文参考文献:

[12] 周傲英,金澈清,王国仁,李建中.不确定性数据管理技术研究综述.计算机学报,2009,32(1):1−16.

[13] 吴爱华,王先胜,谈子敬,汪卫.基于代价模型的不一致 XML 数据修复启发式计算.软件学报,2009,20(4):918−929. http://www.jos.org.cn/1000-9825/3225.htm [doi: 10.3724/SP.J.1001.2009.03225]

[14] 吴爱华.不一致关系数据库上的初始信任标记算法.计算机研究与发展,2010,47(Suppl.):208−214.

吴爱华(1976－),女,江西鹰潭人,博士,副教授,CCF 会员,主要研究领域为不一致数据查询处理,XML 数据库,BPM,知识发现,数据库系统.

谈子敬(1975－),男,博士,副教授,CCF 会员,主要研究领域为 XML 数据库,知识发现,数据库系统.

汪卫(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为复杂结构数据管理,数据挖掘,数据安全.