数据修复与一致性查询处理研究

刘 波[1] 蔡 美[1] 周绪川[2]
(暨南大学信息科学技术学院计算机科学系 广州 510632)[1]
(西南民族大学计算机科学与技术学院 成都 610041)[2]

摘 要 在数据库以及集成系统中通常存在违背数据约束的不一致查询问题。修复是解决该问题的主要手段之一,但目前还缺乏基于修复、约束与查询的统一模型研究。提出了基于删除元组修复、满足多种类型约束的一致性查询算法;阐明了具有简洁特性的约束定义与查询语句结构;构建了新的查询与修复系统模型,将关系实例集、非空的约束集、查询定义、修复方法等统一到模型中,以产生满足一致性约束要求的查询结果。所研究的方法、语言以及模型通用性强、适用面广,不局限于特定质量问题的修复与查询。

关键词 数据修复,一致性,查询模型
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.1.050

Study on Data Repair and Consistency Query Processing

LIU Bo[1] CAI Mei[1] ZHOU Xu-chuan[2]
(Department of Computer Science, College of Information Science and Technology, Jinan University, Guangzhou 510632, China)[1]
(School of Computer Science and Technology, Southwest University for Nationalities, Chengdu 610041, China)[2]

Abstract There are usually query inconsistencies problems of disobeying data constraints in databases and integration systems. Repairing is one of the main approaches to tackle these problems, but few unified models have been studied based on repairing, constraints and queries at present. We proposed the consistency query algorithm based on deleting tuples and given constraints, which produces results being consistent with the constraints, and is suitable for query processing with the various kinds of data constraints (i.e., not specific one). It defines the concise syntax for constraint expressions and query sentences. A new query and repair system model was built, which includes relational instances, a non-null constraint set, query definitions and repair methods, so that the query results satisfy the constraints. The method, language syntax and model proposed in the paper are of universal and applicable properties, which are not limited to specific repair methods, constraints and queries.

Keywords Data repair, Consistency, Query model

1 引言

在数据库以及数据集成应用系统中,可能遇到数据重复、冲突、违背完整性约束等问题。当应用系统的数据来源于多个信息源时,不同信息源的数据模式、约束条件可能不同,不一致问题尤为突出。即使每一信息源采用相同的数据模式以及相同的完整性约束,由多信息源产生的集成视图也可能违背某些约束,产生不一致的查询结果。

例1 考虑表1和表2中两个信息源实例 r_1 和 r_2,r_1 和 r_2 均满足 $X\rightarrow Y$,且 X 为主键。定义视图:$V=\{\langle x,y,z\rangle \mid (x,y,z)\in r_1 \vee (x,y,z)\in r_2\}$。

显然,例1中定义的视图 V 违背了 r_1 和 r_2 的函数依赖和主键约束,并且存在重复结果。

如果按照 V 的定义给出查询,且要求结果与 r_1 和 r_2 的函数依赖和主键约束一致,则需要修复 V 结果。下面列举了2种采用删除元组方式的修复方案。

表1 实例 r_1

X	Y	Z
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_4	c_4

表2 实例 r_2

X	Y	Z
a_1	b_3	c_3
a_3	b_4	c_4
a_4	b_4	c_4

方案1 在 V 中除掉一条重复元组 (a_3,b_4,c_4),并删除 (a_1,b_1,c_1)。

方案2 在 V 中除掉一条重复元组 (a_3,b_4,c_4),并删除 (a_1,b_3,c_3)。

到稿日期:2014-11-18 返修日期:2015-03-19 本文受国家自然科学基金(U1431227,61379019),广东省科技计划项目(2013B010401017),广东省自然科学基金(S2012010008831)资助。

刘 波(1965—),女,硕士,教授,主要研究方向为信息集成、数据挖掘、数据库,E-mail:ddxllb@163.com;蔡 美(1990—),女,硕士生,主要研究方向为数据挖掘、信息集成;周绪川(1972—),男,博士,副教授,主要研究方向为计算机软件与理论。

上述两种修复方案均产生满足函数依赖和主键约束的一致性查询结果，如表 3 和表 4 所列。

表 3　结果 1

X	Y	Z
a_1	b_3	c_3
a_2	b_2	c_2
a_3	b_4	c_4
a_4	b_4	c_4

表 4　结果 2

X	Y	Z
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_4	c_4
a_4	b_5	c_5

在许多数据库与集成应用中，为了给用户提供高质量的查询服务，必须对数据库进行修复(Database Repair)，使得查询结果达到一致、无冗余等要求。目前，一致性查询方法分为两类：一类是基于各种约束条件重写查询，再执行重写的查询，得到一致性结果；另一类是对数据库进行增加、删除或修改操作，将非一致数据库 D 修改为满足一致性约束的数据库 D'，且使得 D 和 D' 的差别尽可能小，从而得到一致性查询结果。查询重写的方式，与被查询数据库的变化无关，但需要消耗查询重写的时间；修改数据库的方式，不需要消耗查询重写的时间，但被查询数据库变化后，需要重新计算修复结果。

本文研究的主要问题是如何修复不满足给定约束集的数据库查询结果，以及基于约束、修复的一致性查询系统模型。查询结果与约束不一致可能是因为单数据源本身违背了约束，也可能是由于多数据源之间的冲突造成的。不管哪种原因，总体思路都是采用查询处理与数据质量检测、修复相结合的方法，不修复被查询的关系本身，而从查询结果(或查询视图)中检测出不一致数据集，再对不一致结果集进行修复。本文所指的不一致数据集是指不满足给定约束的集合，约束(用一阶逻辑语言描述)可以为函数依赖、包含依赖、蕴涵关系、非重复条件等。

本文的主要贡献：

(1)将查询处理与数据质量检测、修复相结合，不进行查询重写，采用限制性修复方式最小化修复查询结果，提出了基于删除修复的一致性查询算法，其适合于满足多种类型约束的一致性查询处理；

(2)基于标准的 SQL 语言，研究了创建统一的约束形式和一致性查询语句结构，查询处理具有检测、修复不一致查询结果的能力；

(3)研究了查询与修复系统模型，将其定义为一个五元组，涉及关系实例集、非空的约束集、查询定义、修复方法、一致性查询结果等。

本文第 2 节叙述相关研究工作；第 3 节给出相关概念；第 4 节研究数据质量检测与修复处理的方法；第 5 节研究基于修复的 SQL 语言结构及查询与修复系统模型；最后总结全文并展望今后工作。

2　相关研究工作

一致性查询处理是 NP 难问题[1]，相关研究主要围绕如下几个方面展开：一致性查询重写、数据库修复算法、查询处理与语言等。

Marcelo Arenas、Leopoldo Bertossi 等在基于完整性约束的一致性查询研究方面具有一定的影响力。他们取得了一系列研究成果[2-5]，提出了满足给定多种完整性约束的查询重写理论模型，并针对多种查询分别提出了计算模型。关于查询重写的其他典型工作包括：文献[6]针对主键约束，通过查询连接图，提出了复杂度为多项式时间的查询重写算法；文献[7]提出 QPIAD(Query Processing over Incomplete Autonomous Databases)系统，利用近似函数依赖、值分布等重写及排序查询。

另一种一致性查询的思路是：最小化修改数据库，而不进行查询重写。对于查询数据集，为了满足完整性约束，实施 I 修复、D 修复或者 U 修复。文献[8]提出了主动完整性约束(Active Integrity Constraint，AIC)的概念，在特定的约束条件下指定需插入或删除的元组，是本文提出的限制性修复的特殊情况。Jef Wijsen 采用基于修改属性值的方法[9]，根据约束对数据表中的错误数据用变量替换并进行修改，将所有可能修改构成一个核心表 G(独立于具体查询)。文献[10]引入 V-Repair 数据库，允许其包含变量，依据一组函数依赖，用另一常数或变量改变原数据库中的数据值。文献[11]提出了抽样化的基于修改属性值的方法，该方法能够满足最少修改元组的要求。文献[12-16]提出了基于函数依赖修复的方法。

目前，基于一致性查询语言的研究主要是针对给定的约束条件进行语言扩展，如：Lyublena Antova 等研究了 I-SQL 语言[17]，提出了 World-set 关系代数，并对 SQL 进行了自然扩展，主要支持基于主键修复的非确定信息查询；J. Chomicki 等提出了一种近似查询语言[18]，研究包含相似度计算的 SQL 语句，在关系数据源上实现数据质量原语。

现有研究存在两方面的局限性：

(1)修复方法大多基于特定的一类约束提出，例如，文献[12]仅针对函数依赖这种约束来研究修复问题，缺乏针对多种类型约束的通用修复方法的研究。

(2)面对各种形式的完整性约束、数据修复要求，现有语言的查询定义与处理能力是不够的。流行的数据库管理系统中所提供的数据定义语句一般仅支持 5 种完整性约束，即 Check 约束、NOT NULL 约束、唯一性约束、主键约束、外键约束，目的是自动检查插入、修改等操作是否符合这些约束，即没有将这些约束用统一的形式表达，也没有将它们用于数据库的自动修复。

我们的研究工作较好地解决了上述两个问题，按照统一的形式描述多类约束，并提供了基于标准 SQL 查询语言修复数据的能力；提出了 Q_REPAIR 算法，其能够针对给定的约束产生所有 D 修复的查询结果。

3　基本概念

借鉴文献[2,10]中的有关概念，给出如下有关定义。

定义 1(关系的约束)　一个关系的约束是用一阶语言表示的公式，由原子构成。原子可以是如下形式之一：

- $t \in r$，其中 t 是元组变量，r 是关系；
- $t[x] \odot s[y]$，其中 t 和 s 是元组变量，x 是 t 所基于的关系模式中的属性，y 是 s 所基于的关系模式中的属性，$\odot$ 是比较运算符；
- $t[x] \odot c$，其中 t 是元组变量，x 是 t 所基于的关系模式中的属性，$\odot$ 是比较运算符，c 是属性 x 所属域中的常量。

在表达约束的公式中，元组变量可被量词符号 $\forall$、$\exists$ 修饰，原子可用与、或、非、蕴涵等符号联结。

例 2　考虑例 1 中给出的实例 r_1 和 r_2，以及基于它们定义的视图 V，用一阶语言表示如下约束：

$IC_1: \forall t_1, t_2(t_1 \in r_1 \wedge t_2 \in r_1) \wedge t_1[X]=t_2[X] \Rightarrow t_1[Y]=t_2[Y]$

$IC_2: \forall t_1, t_2(t_1 \in r_2 \wedge t_2 \in r_2) \wedge t_1[X]=t_2[X] \Rightarrow t_1[Y]=t_2[Y]$

$IC_3: (t_1 \in r_1 \wedge t_2 \in r_1) \wedge t_1[X]=t_2[X] \Rightarrow t_1=t_2$

$IC_4: (t_1 \in r_2 \wedge t_2 \in r_2) \wedge t_1[X]=t_2[X] \Rightarrow t_1=t_2$

$IC_5: \forall t(t \in r_1) \wedge t[Y]=b_4 \Rightarrow t[Z]=c_4$

$IC_6: \forall t(t \in r_2) \wedge t[Y]=b_4 \Rightarrow t[Z]=c_4$

$IC_7: \forall t_1, t_2(t_1 \in V \wedge t_2 \in V) \wedge t_1[X]=t_2[X] \Rightarrow t_1[Y]=t_2[Y]$

$IC_8: \forall t_1, t_2(t_1 \in V \wedge t_2 \in V) \Rightarrow t_1 \neq t_2$

$IC_9: \forall t(t \in V) \Rightarrow \exists t'((t' \in r_2 \vee t' \in r_1) \wedge t'[X]=t[X])$

上述约束的含义如下：

IC_1 和 IC_2 分别表示 r_1 和 r_2 的函数依赖：$X \rightarrow Y$；

IC_3 和 IC_4 分别表示 r_1 和 r_2 的主键为 X；

IC_5 和 IC_6 分别表示 r_1 和 r_2 中所存在的数据项之间的一个蕴涵关系；

IC_7 表示 V 的函数依赖：$X \rightarrow Y$；

IC_8 表示 V 中不存在重复记录；

IC_9 表示包含依赖：V 的 X 属性值都包含在 r_1 或 r_2 的 X 属性值域中。

本文用 $Num\_var(IC)$ 表示 IC 约束中涉及的元组变量个数，因此：

$Num\_var(IC_1)=Num\_var(IC_2)=2$；

$Num\_var(IC_3)=Num\_var(IC_4)=2$；

$Num\_var(IC_5)=Num\_var(IC_6)=1$；

$Num\_var(IC_7)=Num\_var(IC_8)=2$；

$Num\_var(IC_9)=2$。

定义 2（一致性）　一个数据库实例 r 如果满足约束 IC，则 r 基于 IC 是一致的，记为：$r \models IC$；否则，r 是不一致的，记为：$r \not\models IC$。

例 3　考虑例 1 中给出的实例 r_1 和 r_2，$V=\{t \mid t \in r_1 \vee t \in r_2\}$，根据例 2 中给定的约束，存在如下一致或非一致的关系：

$r_1 \models IC_1, r_1 \models IC_3, r_1 \models IC_5, r_1 \models IC_9$；

$r_2 \models IC_2, r_2 \models IC_4, r_2 \models IC_6, r_2 \models IC_{10}$；

$V \not\models IC_7, V \not\models IC_8$。

定义 3（差异）　对于两个数据库实例 r、r'，它们之间不同元组构成的集合称为差异，即：$\Delta(r,r')=(r-r') \cup (r'-r)$。

定义 4（修复）　给定两个数据库模式相同的实例 r、r' 以及一组约束集合 Σ，如果 $r \not\models \Sigma$，$r' \models \Sigma$，并且不存在另一与它们模式相同的数据库实例 r''，使得 $r'' \models \Sigma$ 且 $|\Delta(r,r')| > |\Delta(r,r'')|$，则 r' 是满足 Σ 的 r 的修复。

定义 5（一致性查询结果，一致性查询）　给定数据库实例集 $I=\{r_1, r_2, \cdots, r_m\}(m \geqslant 1)$ 及定义在其上的查询 Q，Σ 为约束集，满足 Σ 的查询结果称为一致性查询结果，记为 $Q_\Sigma(I)$ 或 $Q_\Sigma(r_1, r_2, \cdots, r_m)$；产生一致性结果的查询处理，称为一致性查询处理，简称一致性查询。

例 4　考虑例 1 中给出的实例 r_1 和 r_2，查询 $Q(r_1,r_2)$ 的定义如 V 表示，$V=\{t \mid t \in r_1 \vee t \in r_2\}$，$Q(r_1,r_2)$ 的结果如表 5 所列，该结果违背了例 1 中给出的 IC_7 和 IC_8，满足 $\Sigma=\{IC_7, IC_8\}$ 的一致性查询结果记为 $Q_\Sigma(r_1,r_2)$，表 3 和表 4 分别为 $V=Q(r_1,r_2)$ 的两个修复。

表 5　$Q(r_1,r_2)$

X	Y	Z
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_4	c_4
a_1	b_3	c_3
a_3	b_4	c_4
a_4	b_4	c_4

本文所述的一致性查询结果是指对给定数据库实例查询或修复后得到的一致性结果（不考虑数据库中存在空值或不确定值等情况）。修复操作有 3 种方式：插入元组、删除元组、修改元组属性值[3]，下文称这 3 种操作的修复分别为 I 修复、D 修复、U 修复。

由于修复方式不是唯一的，因此一致性查询结果也不是唯一的。通常情况下，并不要求求出所有修复，只需要按照指定的方式修复，因此下面提出限制性修复的概念。

定义 6（限制性修复）　给定一个数据库实例 r 和约束 φ，如果 $r \not\models \varphi$，且 r' 是按照指定方式（即 I 修复或 D 修复或 U 修复）得到的满足 φ 的 r 的修复，则 r' 称为限制性修复。

定义 7（D 修复）　给定一个关系实例集 I，对 I 的查询 $Q(I)$，查询结果约束集 Σ，$Q(I)$ 的一个 D 修复 $\omega \in \{r' \mid r' \subseteq Q(I) \wedge r' \models \Sigma \wedge (\neg(r'' \subseteq Q(I) \wedge r'' \models \Sigma \wedge \Delta(Q(I),r') \supset \Delta(Q(I),r'') \wedge |\Delta(Q(I),r')| > |\Delta(Q(I),r'')|))\}$。

例如：表 5 所列的查询结果违背了例 1 中给出的 IC_7 和 IC_8，表 3 和表 4 是按照 D 修复方式得到的限制性修复。如果按照 U 修复的方式，还可得到其他限制性修复，如：将 (a_1, b_3, c_3) 修改为 (a_1, b_1, c_3)，得到如表 6 所列的一种 U 修复。

表 6　U 修复结果

X	Y	Z
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_4	c_4
a_1	b_1	c_3
a_4	b_4	c_4

本文所指的修复均为限制性修复，并主要研究删除元组的 D 修复方式。

4　修复与一致性查询结果的计算

数据修复及一致性查询处理的总体思路是：首先，根据查询的定义，计算查询结果；然后，检测结果是否满足给定的约束；再依次根据每一约束修复不一致的结果。

4.1　修复算法

给定一个关系实例集 I、查询 Q 和查询结果约束集 Σ，当 $Q(I) \models \Sigma$ 时，需要修复 $Q(I)$，即计算 $Q_\Sigma(I)$。下面仅采用删除元组的方式来研究相关的修复算法。

给定约束集 Σ，对关系实例集 I 的查询结果进行 D 修复的步骤如下：

(1)执行 $Q(I)$，其结果用 $\aleph$ 表示，即：$\aleph=Q(I)$；

(2)对于 Σ 中的每一约束 $\varphi_i(1 \leqslant i \leqslant k)$，执行 $Upair(\aleph, \varphi_i)$，即基于约束 $\varphi_i(\varphi_i \in \Sigma)$ 修复 $\aleph$。按照如下步骤处理：

①检测$\aleph$中的元组子集是否满足φ_i，子集的大小为$Num_var(\varphi_i)$。

②若存在不满足φ_i的元组子集，则修复$\aleph$。计算方法是在$\aleph$中去掉最少数目元组后，得到满足φ_i的元组集。$\aleph$的修复集可能存在多个，将它们分别保存在$R_{i1}, R_{i2}, \cdots, R_{in}$ $(1 \leqslant i \leqslant k, n \geqslant 1)$中，令$R_i = \{R_{i1}, R_{i2}, \cdots, R_{in}\}$。

若$\aleph$本身已满足φ_i，则令$R_i = \aleph$。

按上述步骤得到的$R_i (1 \leqslant i \leqslant k)$是满足$\varphi_i$的查询一致性结果集，即$Upair(\aleph, \varphi_i) = R_i$，记$Upair(\aleph) = \{R_1, R_2, \cdots R_k\}$。

基于约束的查询与修复算法 Q_REPAIR 如下所示，其中：$Check(\aleph, \varphi_i)$返回基于φ_i检测不一致元组集的结果，$Upair(\aleph, \varphi_i, vio)$返回基于$\varphi_i$修复$vio$的结果。

算法名称：Q_REPAIR

```
输入：Q( r1,r2,…,rm);//查询表达式
      φ1,φ2,…,φk; //查询约束
方法：
  (1)ℵ=Q( r1,r2,…,rm);//执行查询
  (2) for i=1 to k
  (3)   vio=Check (ℵ,φi);
  (4)   if vio<>NULL then
  (5)     Ri=Upair(ℵ,φi);
  (6)   else Ri=ℵ;
  (7)end for;
输出：R1,R2,…,Rk
```

函数名称：Check

```
输入参数：关系ℵ，约束 φi
方法：
(1) vio=NULL;
(2)for each tset in ℵ
   //对ℵ中包含 Num_var(φi)个元组的子集 tset
(3)  if tset⊨ φi then Add tset into vio;
     //将违背约束的元组加入 vio 中
(4)end for;
返回：vio
```

函数名称：Upair

```
输入参数：关系ℵ，约束 φi，违背集 vio
方法：
(1) Ri=NULL;k=1;
(2) while k<|vio|
(3)   for each tsetk in vio
      //对 vio 中包含 k 个元组的子集 tsetk
(4)     if ℵ−tsetk⊨φi then
(5)       Add ℵ−tsetk into Ri;
          //添加元素到 Ri 中
(6)   end for;
(7)   if Ri=NULL then k++;
(8)   else k=|vio|;//跳出 while 循环
(9) end while;
返回：Ri
```

定理 1 给定一个关系实例集I的查询Q、约束φ_i，若$Upair(Q(I), \varphi_i)$的结果集为$R_i, r_k \in R_i, r_m \in R_i$，则$|r_k| = |r_m|$。

证明：若$k = m$，显然，定理成立。

若$k \neq m$，假设$|r_k| \neq |r_m|$，但$r_k \in R_i, r_m \in R_i$，即r_k和r_m都是满足约束φ_i的$Q(I)$修复。令$r_q = Q(I)$，若$|r_k| > |r_m|$，$|\Delta(r_q, r_m)| > |\Delta(r_q, r_k)|$，根据定义 4，$r_m$不是修复，与$r_m \in R_i$矛盾；同样，若$|r_k| < |r_m|$，$|\Delta(r_q, r_k)| > |\Delta(r_q, r_m)|$，根据定义 4，$r_k$不是修复，与$r_k \in R_i$矛盾。

也就是说，针对某一约束，D 修复如果存在多种结果，它们包含的元组数目相同。

4.2 Q_REPAIR 算法复杂度分析

Q_REPAIR 算法的时间复杂度主要由$Check$函数和$Upair$函数决定，分析如下。

(1)$Check(\aleph, \varphi_i)$

令$Num_var(\varphi_i) = k$，$|\aleph| = n$，$\aleph$中包含k个元组的子集数目为C_n^k，所以最坏情况为$k = 1$，$Check(\aleph, \varphi_i)$的复杂度为$O(n)$。

(2)$Upair(\aleph, \varphi_i)$

令$|vio| = n'$，从vio中去掉 1 个元组有n'种可能；从vio中去掉 2 个元组的结果有$C_{n'}^2$种可能，从vio中去掉k个元组的结果有$C_{n'}^k$种可能，所以$Upair(S_2, \varphi_i)$测试删除元组的子集数目为：$n' + C_{n'}^2 + \cdots + C_{n'}^k$，$Upair(\aleph, \varphi_i, vio)$的时间复杂度为$O(2^k)$，$k < n'$。

(3)Q_REPAIR 的复杂度

根据算法 Q_REPAIR 的描述，其复杂度为$O(m(n + 2^k))$，其中m为查询约束的个数，n为修复前查询结果$\aleph$中包含的元组数目，k为修复过程从$\aleph$中删除的元组数目。依据定理 1，按照 Upair 方法修复后所有可能结果中包含的元组数目相同，假设任一结果中包含的元组数目为p，则$k = n - p$。

从以上分析可知，当n远大于2^k时，$O(m(n + 2^k))$约为$O(mn)$。

4.3 一致性查询结果的计算

根据 4.2 节的 Upair 修复方法，得到满足Σ中各约束的查询一致性结果集$R_i (1 \leqslant i \leqslant k$，其中$k$为约束个数)，按照式(1)计算它们之间的交集，结果用$Q_c(I)$表示。

$$Q_c(I) = \{r \mid r = \bigcap_{i=1}^{k} S_i, S_i \in R_i\} \tag{1}$$

定理 2 给定一个关系实例集I的查询Q及约束集Σ，$Q_c(I)$返回满足Σ的查询结果。

证明：令$\Sigma = \{\varphi_1, \cdots, \varphi_k\}$，$k \geqslant 1$；

依据 4.1 节的修复方法，$Upair(Q(I), \varphi_i, vio)$的执行结果集$R_i$中的任意元素满足$\varphi_i$；

依据式(1)，对任一$r \in Q_c(I)$，$r \models \{\varphi_1, \cdots, \varphi_k\}$成立，即$r \models \Sigma$。

定理 3 给定一个关系实例集I的查询Q及约束集Σ，如果$Q(I) \not\models \Sigma$，$r \models \Sigma$，且r是$Q(I)$基于 D 修复方式得到的结果，则$r \in Q_c(I)$。

证明：因为r是$Q(I)$基于Σ的一个修复，所以$r \in Q_\Sigma(I)$。

令$\Sigma = \{\varphi_1, \cdots, \varphi_k\}$，$k \geqslant 1$，则$r \models \varphi_1, \cdots, r \models \varphi_k$

按照$Upair(Q(I), \varphi_i, vio)$的计算方法，$Upair(Q(I), \varphi_i, vio) = R_i$，$R_i$为满足$\varphi_i$的所有$Q(I)$的 D 修复结果集合。

因此，$r \in R_1, \cdots, r \in R_k$。

根据$Q_c(I)$的计算式，$r \in Q_c(I)$。

定理 2 和定理 3 说明$Q_c(I)$的计算是可靠和完备的。定理 2 说明$Q_c(I) \subseteq Q_\Sigma(I)$；而定理 3 表示$Q(I)$的每一个基于

D修复方式得到的结果都包含在$Q_c(I)$中。

例5　考虑例1中给出的实例r_1和r_2，关于r_1和r_2的查询定义为：$Q(r_1,r_2):\{t|t\in r_1 \vee t\in r_2\}$，结果用$V$表示，给出一致性查询约束集：$\Sigma=\{\varphi_1,\varphi_2\}$，其中：

$\varphi_1:\forall t_1,t_2(t_1\in V\wedge t_2\in V\wedge t_1[X]=t_2[X]\Rightarrow t_1[Y]=t_2[Y]$

$\varphi_2:\forall t_1,t_2(t_1\in V\wedge t_2\in V)\Rightarrow t_1\neq t_2$

求$Q_c(r_1,r_2)$的过程如下：

(1)执行$Q(r_1,r_2)$，得到结果$\aleph$，如表5所列。

(2)基于φ_1，修复$\aleph$，得到$R_1=\{R_{11},R_{12}\}$，R_{11}和R_{12}的结果分别如表7和表8所列；基于φ_2，修复$\aleph$，得到$R_2=\{R_{21}\}$，R_{21}如表9所列。

表7　R_{11}

X	Y	Z
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_4	c_4
a_3	b_4	c_4
a_4	b_4	c_4

表8　R_{12}

X	Y	Z
a_2	b_2	c_2
a_3	b_4	c_4
a_1	b_3	c_3
a_3	b_4	c_4
a_4	b_4	c_4

表9　R_{21}

X	Y	Z
a_1	b_1	c_1
a_2	b_2	c_2
a_1	b_3	c_3
a_3	b_4	c_4
a_4	b_4	c_4

(3)按照(式(1)，求$Q_c(r_1,r_2)$，即$R_{11}\cap R_{21}$，$R_{12}\cap R_{21}$，得到的最终查询结果如表3和表4所列。

4.4　Q_REPAIR的流水计算方式

上面提出的一致性查询结果的计算方法，首先针对每一约束检测与修复查询结果，然后对各自产生的结果进行交集运算，消耗了较多的读写时间。采用流水计算方式可以提高计算性能，基本思路是：将基于一个约束的检测与修复结果提供给基于下一个约束的检测与修复。

5　基于约束的一致性查询与修复模型

5.1　一致性查询语句结构

基于约束的一致性查询语句的形式如下：

```
SELECT      Selists | *
FROM        view_name|table_list
[WHERE      Conditions ]
[REPAIR BY  Conslist]
[WITH       Algorithm ];
```

上述语句结构与标准SQL查询语句相比，扩充了repair by和WITH关键字。view_name表示查询视图，table_list表示关系序列，Conditions是查询条件，Conslist是对view_name或table_list的约束列表；Algorithm是限制性修复算法，以view_name和Conslist为输入参数，输出一组满足各约束的修复关系。

Conslist中的约束作用与标准SQL语言中数据定义语句的约束作用不同。标准SQL语言把各种完整性约束作为数据库模式定义的一部分，表示关系表的主键、外键、非空列、属性值域或条件等；而Conslist可表达定义1中的约束，用于不一致数据库的检测与修复，因此，需要研究一致性查询语句中各种约束的统一表达形式。

容易观察到，蕴涵关系既可以表示函数依赖、主键、非重复等约束，也可以表示属性值域或条件等约束，如：例2中给出的$IC_1,IC_2,\cdots,IC_8$都是用蕴涵关系表示函数依赖及非重复约束的；又如，可以用$\forall t\ (t\in V)\Rightarrow V[X]\neq$NULL表示$X$值非空，用$\forall t\ (t\in V)\Rightarrow V[X]\geqslant 100$表示$X$值不小于100等。所以，这里考虑用蕴含关系表达一致性查询语言中的约束。

本文将蕴含关系的一般形式定义为：$A\Rightarrow B$，其中A和B是包含一个或多个元组变量的公式。该形式的约束也可看成以A为前提(可为空)、B为结论的规则，创建这种形式的约束语句格式为：

```
CREATE CONSTRAINT φ
FROM        view_name|table_list
TUPLE-VAR   variable_list
BY          [formula1],formula2;
```

其中，φ为约束名；view_name为查询表或视图；variable_list为view_name或table_list的元组变量列表，被$\exists$修饰的元组变量前用#字符标注；$formula_1$和$formula_2$为约束规则的前提和结论，没有前提的约束无$formula_1$。

例6　根据例5给出的查询和约束，创建如下查询视图、约束和查询语句。

创建查询视图：

```
CREATE VIEW R_view(r1.X,r1.Y,r1.Z) AS
SELECT      X,Y,Z
FROM        r1
UNION
SELECT      X,Y,Z
FROM        r2;
```

创建约束φ_1：

```
CREATE      CONSTRAINT φ1
FROM        R_viewTUPLE-VARt1,t2
BY          t1[X]=t2[X],t1[Y]=t2[Y];
```

创建约束φ_2：

```
CREATE      CONSTRAINT φ2
FROM        R_view
TUPLE-VAR   t1,t2
BY          t1≠t2;
```

查询语句：

```
SELECT      *
FROM        R_view
REPAIR BY   φ1,φ2;
WITH        Q_REPAIR
```

5.2　查询与修复系统模型

查询与修复系统可以定义为一个五元组：$\mathcal{R}=(I,\Sigma,R,Q,Op)$，其中$I$是有限元组数目的关系实例集；$\Sigma$是非空的约束集；$Q$是对$I$的查询；$Op$为对$Q(I)$的修复操作；$R$为$Op$操作的结果集，可能为空集，也可能是多个关系的集合。

按照定义7，D修复的查询结果表示如下：

$R=\{r'|\ r'\subseteq Q(I)\wedge r'\models\Sigma\wedge(\neg(\exists r''\subseteq Q(I)\wedge r''\models\Sigma\wedge\Delta(Q(I),r')\supset\Delta(Q(I),r'')\wedge|\Delta(Q(I),r')|>|\Delta(Q(I),r'')|))\}$

按照本文提出的D修复定义和修复算法，可以求出所有可能的基于D修复的一致性查询结果，定理2和定理3说明了这一点。

(下转第241页)

[12] Chieu H L, Lee Y K. Query based event extraction along a timeline[C]// ACM SIGIR. Sheffield, 2004: 425-432

[13] Rui Yan, Wan Xiao-jun, Otterbacher J, et al. Evolutionary timeline summarization: a balanced optimization framework via iterative substitution[C]// Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Beijing, 2011: 745-754

[14] Rui Yan, Liang Kong, Huang Cong-rui, et al. Timeline generation through evolutionary trans-temporal summarization[C]// Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Edinburgh, 2011: 433-443

[15] Page L, Brin S, Motwani R, et al. The PageRank Citation Ranking: Bringing Order to the Web[R]. Stanford Digital Library Technologies Project, 1999: 1-17

[16] Mei, Qiao zhu, Jian Guo, et al. DivKank: the Inte-rplay of Prestigo and Diversity in information Networks[C]// Special Interested Group on Knowledge Discovery in Databases. Washington, United States, 2010: 1009-1018

[17] Chen Ji-li, Niu Qin-zhou. Duplicated webpages deletion based on feature code[J]. Microcomputer Information, 2006(3): 113-115 (in Chinese)
陈基漓，牛秦洲. 基于特征码的网页去重[J]. 微计算机信息，2006(3): 113-115

[18] Xiong Zhong-yang, Ya Man, Zhang Yu-fang. Detection and climination of similar Web pages based on text structure and string of feature code[J]. Journal of Computer Applications, 2013, 33(2): 554-557(in Chinese)
熊忠阳，牙漫，张玉芳. 基于网页正文结构和特征串的相似网页去重算法[J]. 计算机应用，2013, 33(2): 554-557

(上接第 236 页)

然而，与 D 修复不同，I 修复或 U 修改的结果中可能包含 I 中不存在的元组属性值，插入元组一般根据指定的包含约束规则操作；修改元组的属性值一般用属性域内的常数或变量替代，有些 U 修改算法仅产生一个修复结果，有些产生多种修复结果[11]。因此，I 修复或 U 修改的不确定性较大。

总之，不管采用 D 修复方式还是 I 修复或 U 修复方式，我们都可以采用统一的查询语言结构以及修复系统模型。

结束语 本文将数据修复与一致性查询处理相结合，研究了基于 D 修复、满足多种类型约束的一致性查询方法。虽然 D 修复可能删去了原本正确的元组，但该方法能产生多种结果，使得所删去的元组可能在其他结果中出现。此外，扩展了标准 SQL 查询语言，采用蕴含关系表达多种类型的约束，定义了新的约束创建语句、一致性查询语句，提出了查询与修复系统模型。下一步将继续优化查询与修复算法，实现查询与修复系统功能。

参 考 文 献

[1] Greco S, Molinaro C. Querying and repairing inconsistent databases under three-valued semantics[C]// ICLP 2007. Springer Berlin Heidelberg, 2007: 149-164

[2] Arenas M, Bertossi L, Chomicki J. Consistent query answers in inconsistent databases[C]// Proceedings of the ACM Symposium on Principles of Database Systems. New York: ACM Press, 1999: 68-79

[3] Andrea R M, Bertossi L, Marileo M C. Consistent query answering under spatial semantic constraints[J]. Inf. Syst., 2013, 38(2): 244-263

[4] Bertossi L, Kolahi S, Lakshmanan Laks V S. Data cleaning and query answering with matching dependencies and matching functions[J]. Theory Comput. Syst., 2013, 52(3): 441-482

[5] Arenas M, Gottlob G, Pieris A. Expressive languages for querying the semantic Web[C]// Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems(PODS 2014). 2014: 14-26

[6] Fuxman A D, Miller R J. First-Order Query Rewriting for Inconsistent Databases[J]. Journal of Computer and System Sciences, 2007, 73(4): 610-635

[7] Wolf G, Kalavagattu A, Khatri H, et al. Query processing over incomplete autonomous databases: query rewriting using learned data dependencies[J]. The VLDB Journal, Springer Berlin Heidelberg, 2009, 18(5): 1167-1190

[8] Caroprese L, Greco S. Active integrity Constraints for Database Consistency Maintenance[J]. IEEE transaction on Knowledge and Data Engineering, 2009, 21(7): 1042-1058

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

[10] Kolahi S, Lakshmanan Laks V S. On approximating optimum repairs for functional dependency violations[C]// ICDT 2009. ACM Publisher, 2009: 53-62

[11] Beskales G, Ilyas Ihab F, Golab L. Sampling the Repairs of Functional Dependency Violations under Hard Constraints[J]. Proceedings of the VLDB Endowment, 2010, 3(1): 197-207

[12] Hu Yan-li, Zhang Wei-ming, Luo Xu-hui, et al. Dependencies Theory and its Application for Repairing Inconsistent Data[J]. Computer Science, 2009, 36(10): 11-15(in Chinese)
胡艳丽，张维明，罗旭辉，等. 基于数据依赖的数据修复研究进展[J]. 计算机科学，2009, 36(10): 11-15

[13] Cheng Lu-qing. Conditional functional dependency and data quality control[J]. Information System Engineering, 2009(11): 106-108(in Chinese)
程录庆. 条件函数依赖与数据质量控制[J]. 信息系统工程，2009(11): 106-108

[14] Geng Yin-rong, Liu Bo. Conditional functional dependencies for detecting data inconsistencies[J]. Computer Engineering and Applications, 2012, 48(3): 122-125(in Chinese)
耿寅融，刘波. 基于条件函数依赖的数据库一致性检测研究[J]. 计算机工程与应用，2012, 48(3): 122-125

[15] Beskales G, Ilyas I F, Golab L, et al. Sampling from repairs of conditional functional dependency violations[J]. The VLDB Journal, 2014, 23(1): 103-128

[16] Neehar C, Krishna T V S. Inconsistent relational data cleaning by detecting conditional functional dependencies[J]. International Journal of Computer Science and Information Technology & Security (IJCSITS), 2013, 3(1): 120-125

[17] Antova L, Koch C, Olteanu D. From complete to incomplete information and back[C]// SIGMOD'07. ACM, 2007: 713-724

[18] Chandel A, Hassanzadeh O, Srivastava D. Benchmarking Declarative Approximate Selection Predicates[C]// SIGMOD'07. ACM, 2007: 353-364