面向数据集成的一种高效一致性查询方法

张晓刚[1,2],杨路明[1],潘久辉[2]
(1.中南大学信息科学与工程学院,湖南长沙 410083;2.暨南大学计算机系,广东广州 510632)

摘 要: 一阶查询的 SQL 可表达性使得基于数据库修复的一阶查询重写方法在解决不一致数据库上的一致性查询问题上更具实际应用价值,但现有方法生成的一致性查询重写的执行效率不够理想.本文重点考虑在数据集成环境下如何有效地提高一致性查询的执行效率,同样针对合取查询类 C_{forest} 提出了基于 OR-database 集成模式的一致性查询重写算法 ConsRewrite-OR.基于线性工作度量的查询代价分析以及 Oracle 10g 上的 TPC-H 模拟实验都充分地说明本文算法产生的一致性查询与 Fuxman 的查询重写相比在集成数据库上具有更好的执行效率.

关键词: 一致性查询; 完整性约束; 修复; 查询重写; 不一致集成数据库
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2014)08-1474-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.08.003

An Efficient Consistent Query Answering Method for Data Integration

ZHANG Xiao-gang[1,2], YANG Lu-ming[1], PAN Jiu-hui[2]
(1. *Institute of Information Science and Engineering, Central South University, Changsha, Hunan* 410083, *China*;
2. *Department of Computer Science, Jinan University, Guangzhou, Guangdong* 510632, *China*)

Abstract: The expressiveness of first-order queries makes first-order query rewriting based on database repair has more practical value on solving CQA problem over inconsistent database compared with other methods. However, the execution efficiency of consistent query rewriting generated from the existing methods might be unsatisfactory. How to effectively promote the performance of consistent query in the data integration environment is studied principally in this paper. Facing the same conjunctive query class C_{forest}, the consistent query rewriting algorithm ConsRewrite-OR based on OR-database integration schema is presented. Both query cost analysis based on the linear work metric and TPC-H simulated experiments on Oracle 10g adequately indicate that the consistent query rewriting produced by ConsRewrite-OR can obtain more optimized execution efficiency on integration databases, compared with Fuxman's consistent query rewriting.

Key words: Consistent query answering; integrity constraints; repair; query rewriting; inconsistent integration database

1 引言

完整性约束为数据库中数据提供与外部现实保持语义一致的途径,违反完整性约束的数据库被称为不一致数据库.不一致数据库上的一致性查询(Consistent Query Answering, CQA)[1]问题在数据集成环境下尤为普遍.数据集成为用户提供分布、自治及异构数据源数据的统一视图,来自各数据源的数据即使满足自身完整性约束,仍有可能违反全局模式的完整性约束.例如:不同数据源关于同一个雇员在薪水和地址信息产生不一致,而数据集成需要为用户查询返回该雇员一致的信息.

对不一致数据库上的 CQA 问题,Arenas 等人在文献[2]中首先提出了基于修复的查询重写方法,但是它仅考虑了无量化或无析取的一阶查询及二元完整性约束;Fuxman 在主键约束下的一阶查询重写上取得重要突破,针对基于查询连接图定义的合取查询类 C_{forest} 提出了一阶查询重写算法 RewriteConsistent[3] 并应用到 ConCuer 系统[4] 中; Wijsen 在这方面也做了大量工作[5,6],但主要是处理特殊的布尔合取查询.基于逻辑程序计算 CQA 是更一般化的方法[7,8],它能够处理任意一阶查询及全称完整性约束,但无法与现有商业数据库技术结合而得到广泛应用.

与逻辑程序相比,一阶查询重写由于能够表达为 SQL 而更具实际应用价值.但是,现有方法重写的一致性查询的执行效率不够理想.对数据集成环境下的 CQA,在数据集成构建时就可以充分考虑一致性查询的

收稿日期:2013-06-28;修回日期:2014-02-06;责任编辑:蓝红杰
基金项目:国家自然科学基金(No.61003056);广东省自然科学基金(No.S2012010008831);广东省教育部产学研结合项目(No.2007B090400095)

执行效率,本文通过 OR-database 集成模式有效地隔离集成数据库的一致性部分和不一致性部分,对同样的 C_{forest} 提出了基于 OR-database 的一致性查询重写算法.基于线性工作度量的查询代价分析以及 Oracle 10g 上的 TPC-H 模拟实验都表明本文的一致性查询重写与 Fuxman 的查询重写相比具有更好的执行效率.

2　OR-database 的数据集成及一致性查询

2.1　OR-database 数据集成模式

OR-database[9]是定义在包含简单常量和有限可选常量(即 OR-object)集合上的非确定性数据库,它代表了一个可能世界集合,即用确定的域可选常量替换每一个 OR-object 所获得的数据库实例集合.如果用 $D(A_i)$ 表示定义在属性集合 $U=\{A_1,\cdots,A_n\}$ 上的关系模式 $R(U)$ 的第 i 个属性的域,则 R 的实例 r 就是 $D(A_1)\times\cdots\times D(A_m)$ 的一个子集,而一个关系数据库模式是一个关系模式集合 $\Sigma=\{R_1,\cdots,R_n\}$,一个关系数据库实例则是 Σ 上的关系实例集合 $I=\{r_1,\cdots,r_n\}$.本文使用如下关系模式表示 OR-database,以便有效隔离数据库的一致性部分和非一致性部分.

定义 1　设 ATOMIC 常量域 D_{at} 和 OR-object 标识常量域 D_{or} 是两个不相交的可数无限常量集合,定义 $D=D_{at}\cup D_{or}$ 上表达 OR-database 的关系数据库模式 Σ 包含 ATOMIC 和 OR-object 两类关系,每个 ATOMIC 关系 R 都存在一个 OR-object 关系 R_or 与其对应.如果用 $Key(X)$ 和 $Nokey(X)$ 分别表示关系 X 的主键与非主键,则 Σ 的 R 及 R_or 应该满足:

(1) $D(Key(R\_or))\subseteq D_{or}\wedge D(Key(R))\subseteq D_{at}$

(2) $Key(R\_or)\subset Nokey(R)\wedge Nokey(R\_or)=Nokey(R)-Key(R\_or)$

OR-database 的关系模式 Σ 通过 OR-object 关系 R_or 存储对应 ATOMIC 关系 R 中的 OR-object,即违反主键约束的元组集合,R 仅保留一个元组并通过外键引用 R_or 中的 OR-object,同时设置 $Nokey(R)$ 的冲突属性值为未知.那么,基于 OR-database 的数据集成系统可以根据文献[10]做如下定义.

定义 2　一个基于 OR-database 的数据集成系统 $I=\langle\Sigma,\mathcal{S},\mathcal{M}\rangle$ 需要满足下面三个条件:

(1)全局模式 Σ 是表达 OR-database 的关系数据库模式,同时包含 ATOMIC 关系和 OR-object 关系.

(2)数据源模式 $\mathcal{S}$ 是非 OR-database 的关系数据库模式,仅包含 ATOMIC 关系.

(3)模式映射 $\mathcal{M}$ 是扩展源-目标(STD)[11]允许出现谓词递归的元组依赖集合,元组依赖具有形式 $\psi_\Sigma(\underline{\boldsymbol{x}},\boldsymbol{z}):-\varphi_S(\underline{\boldsymbol{x}},\boldsymbol{y}),\lambda_\Sigma(\underline{\boldsymbol{x}},\boldsymbol{w})$,其中 $\psi_\Sigma(\underline{\boldsymbol{x}},\boldsymbol{z})$ 和 $\lambda_\Sigma(\underline{\boldsymbol{x}},\boldsymbol{w})$ 都是模式 Σ 上的原子或原子否定构成的合取公式,而 $\varphi_S(\underline{\boldsymbol{x}},\boldsymbol{y})$ 是模式 S 上的任意一阶公式.

设 s,g 分别为 S 和 Σ 的数据库实例,如果 s 和 g 满足 $\mathcal{M}$,即对 $\mathcal{M}$ 中的每一条依赖规则 $\psi_\Sigma(\underline{\boldsymbol{x}},\boldsymbol{z}):-\varphi_S(\underline{\boldsymbol{x}},\boldsymbol{y}),\lambda_\Sigma(\underline{\boldsymbol{x}},\boldsymbol{w})$ 都有:对任意 $\boldsymbol{a},\boldsymbol{c}$ 如果存在某个 $\boldsymbol{e}$ 使得数据库实例 s 和 g 满足 $\varphi(\boldsymbol{a},\boldsymbol{c})\wedge\lambda(\boldsymbol{a},\boldsymbol{e})$,则必定存在一个 $\boldsymbol{v}$ 使得 g 满足 $\psi(\boldsymbol{a},\boldsymbol{v})$.

2.2　OR-database 全局模式上的一致性查询

完整性约束表达数据对应外部现实的语义一致性,DBMS 一般通过事务维护其有效性,最常用的完整性约束就是函数依赖和包含依赖,例如:主外键约束.如果属性集 Y 为关系 $R(U)$ 的主键即 $Key(R)=Y$,则主键约束 $Y\rightarrow U$ 是一个函数依赖;对关系 $R(U)$ 和 $S(V)$,外键约束 $R(W)\subseteq S(Z)$ 是一个包含依赖,其中 $W\subseteq U$,$Z\subseteq V$ 并且 $Z=Key(S)$.那么,OR-database 全局模式 Σ 上的完整性约束集合定义了来自各数据源的数据应该满足的语义一致性.

定义 3　OR-database 全局关系模式 Σ 上的完整性约束 C 仅包括主外键,Σ 上的集成数据库实例 g 如果满足 C 则称 g 关于 C 是一致的,记为 $g\models C$;否则称 g 关于 C 是不一致的,记为 $g\not\models C$.

定理 1　OR-database 数据集成系统 $I=\langle\Sigma,\mathcal{S},\mathcal{M}\rangle$,设数据源实例 s 关于 S 的主外键约束 C_s 是一致的,则集成数据库实例 g 关于 Σ 的完整性约束 C 不一致只需满足下面两个条件之一:

(1) 设 ATOMIC 关系 R_i 存在主键,如果与 R_i 对应的 OR-objects 关系 R_i_or 的实例不空,则有 $g\not\models C$.

(2) 设 ATOMIC 关系 R_i 的属性 $A\in Nokey(R_i)$ 是参考 ATOMIC 关系 R_j 的外键,如果它们的实例 r_i 和 r_j 满足 r_i 中存在元组 t 使得 $t[A]\notin D(Key(r_j))$ 或者与 R_i 对应的 OR-objects 关系 R_i_or 的实例 s 满足存在元组 t' 使得 $t'[A]\notin D(Key(r_j))$ 时,则有 $g\not\models C$.

对于 OR-database 全局模式 Σ 上的 CQA 问题,本文沿用文献[2]基于修复的观点加以解决,由于 Σ 的集成数据库实例 g 可能违反的完整性约束仅包括函数依赖和包含依赖,所以 g 的修复也只考虑元组删除[3].g 的一个修复 g' 就是删除 g 中违反完整性约束的一个最小事实子集而得到的,即删除 ATOMIC 关系中不存在引用的元组及对应 OR-object 关系中违反主键的元组.

定义 4　设 C 是 OR-database 全局模式 Σ 上的完整性约束,g 是 Σ 上的一个数据库实例(可能 $g\not\models C$),$q(\boldsymbol{z})$ 是 Σ 上的合取查询,如果一个元组 $\boldsymbol{t}$ 对 g 关于 C 的每一个修复 g' 都满足 $g'\not\models q[\boldsymbol{z}/\boldsymbol{t}]$ 则被称为 q 关于 C 的一个一致性回答,表示为 $\boldsymbol{t}\in CQA_C(q,g)$;对于 Σ 上的布尔查询 q 来说,如果 g 关于 C 的每个修复 g' 都有

$g' \not\models q$ 则称 $CQA_C(q,g)$ = true,如果至少存在一个修复 $g' \not\models q$ 则称 $CQA_C(q,g)$ = false.

例 1 设 OR-database 全局模式 $\Sigma = \{emp(ename, dept, sal, or\text{-}id), dept(dname, loc, or\text{-}id), emp\text{-}or(or\text{-}id, dept, sal), dept\text{-}or(or\text{-}id, loc)\}$, OR-objects 关系 *emp_or* 和 *dept_or* 分别存储雇员表 *emp* 和部门表 *dept* 出现的 OR-objects,每个关系的下划线属性为其主键,Σ 上的一个数据库实例 g 如图 1 所示.

注意,由于 *dept* 未出现 OR-object,所以图中未列出空的 *dept_or*. *emp* 的外键 *or-id* 使用 # 开头的数字串标识并引用 *emp_or* 中的 OR-object,同时冲突元组相关属性值被置为未知,而对于 *emp* 和 *dept* 中不出现 OR-object 的元组,其 *or-id* 属性值被置为不存在; *emp* 的另一外键 *dept* 引用 *dept* 的主键 *dname*. 接下来考虑几个具有式(1)形式的合取查询:

首先,对于检索薪水等于 3000 的雇员姓名的查询 $q_1(e) = \exists d, o.\ emp(\underline{e}, d, 3000, o)$,根据定义 4 可知其一致性回答为(Jennifer)和(Smith);同理,对于检索薪水大于等于 3000 的雇员姓名及其工作地点的查询 $q_2(e, l) = \exists d, s, o.\ emp(\underline{e}, d, s, o) \wedge \exists o.\ dept(\underline{d}, l, o) \wedge s \geq 3000$,一致性回答应包含(Jennifer, New York)和(Roman, Dallas);最后,对询问雇员 Smith 在 Chicago 工作是否正确的布尔查询 $q_3 = \exists d, o, s.\ emp(\text{"Smith"}, d, s, o) \wedge \exists o.\ dept(d, \text{"Chicago"}, o)$来说,其一致性回答显然应为 false.

ename	dept	*sal*	*or-id*
Jennifer	Accounting	3000	null
Smith	null	3000	#1
Jack	Sales	2800	null
Roman	null	null	#2
John	Research	null	#3

emp

dept	*loc*	*or-id*
Accounting	New York	null
Research	Chicago	null
Sales	Dallas	null
Operations	Dallas	null

dept

or-id	*dept*	*sal*
#1	Accounting	3000
#1	Research	3000
#2	Sales	3500
#2	Operations	3200
#3	Research	2800
#3	Research	2500

emp_or

图1 OR-database集成数据库实例

3 一致性查询重写算法

3.1 一致性查询

查询重写计算 CQA 的最大优势在于无需明确构建修复,而是通过查询变换得到新查询并直接从不一致数据库获取与原查询预期的一致性回答. OR-database 全局模式上的用户查询是基于 ATOMIC 关系,如果 ATOMIC 关系 R 存在元组 t 且其外键属性 or_id 非空并引用 OR-object 关系 R_or,由定理 1 可知 g 不一致. g 上的一致性查询重写就需要 R 连接 R_or 以获取一致性回答.

定义 5 设 Σ 为 OR-database 全局模式而 C 是 Σ 上的完整性约束集合, $q(\vec{z})$ 是 Σ 上的用户查询,对 Σ 上的每个数据库实例 g,如果 Σ 上存在一个一阶查询 Q 当且仅当 $\vec{t} \in CQA_C(q,g)$时满足 $g \models Q[\vec{z}/\vec{t}]$,则称一阶查询 Q 为 q 的一致性查询重写.

例 2 继续前面例 1,对全局模式 Σ 上的三个查询 q_1, q_2 和 q_3 来说,它们的一致性一阶查询重写 Q_1, Q_2 和 Q_3 可以分别定义如下:

$Q_1(e) = \exists d, o.\ emp(\underline{e}, d, 3000, o)$

$Q_2(\underline{e}, l) = \exists d, s, o.\ emp(e, d, s, o) \wedge (\exists o'.\ dept(d, l, o') \wedge s \geq 3000 \vee \forall d' \forall s'.\ emp\text{-}or(o, d', s') \rightarrow \exists o'.\ dept(d', l, o') \wedge s' \geq 3000)$

$Q_3 = \exists d, s, o.\ emp(\text{"Smith"}, d, s, o) \wedge (\exists o'.\ dept(d, \text{"Chicago"}, o') \vee \forall d' \exists s.\ emp\_or(o, d', s) \rightarrow \exists o'.\ dept(d', \text{"Chicago"}, o'))$

由于查询 q_1 仅涉及一个 ATOMIC 关系 *emp* 且不存在非键属性上的非等值选择条件,所以该查询不需要添加与 OR-object 关系 *emp_or* 连接的查询析取项,即 $Q_1 = q_1$. 也就是说查询 q_1 本身返回一致性回答;而对包含多个 ATOMIC 关系连接的查询 q_2 和 q_3,需要通过逐级嵌套来完成它们的一致性查询重写,在连接 *emp* 与 *dept* 的基础上,还需要添加 *emp* 与 *emp_or* 及 *dept* 的连接查询析取项,并且在计算嵌套查询时可以把常量选择直接下推到叶子节点以提高查询效率.

3.2 查询重写算法

对违反主外键约束的不一致数据库,文献[3]提出了合取查询类 C_{forest} 并对该查询类给出了多项式时间的一致性查询重写算法 RewriteConsistent,本文同样对 C_{forest} 提出基于 OR-database 全局模式的新一致性查询重写算法 ConsRewrite_OR:

算法 1 ConsRewrite_OR(q, C)算法

输入:查询 $q(z) = \exists \boldsymbol{w}, \boldsymbol{o}.\ \varphi(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{o}) \wedge \varphi$, C 为 q 中关系的主外键约束;

输出:Q,q 的一致性查询重写

设 G 为 q 的连接图,$T_1,\cdots,T_m$ 为 G 的连通分量;

for $i:=1$ to m do

设 $R_i(\underline{\boldsymbol{x}}_i,\boldsymbol{y}_i,o)$为 T_i的根节点关系,φ_i 为 T_i 的关系合取式,而 $\boldsymbol{w}_i$ 和 $\boldsymbol{z}_i$ 分别为 φ_i 的存在变量向量及自由变量向量,

$\boldsymbol{o}_i$ 为 φ_i 上的 *or_id* 属性变量向量,φ_i 则为 φ_i 上的非等值内建谓词,$q_i(\boldsymbol{x}_i,\boldsymbol{z}_i)=\exists \boldsymbol{w}_i.\varphi_i(\boldsymbol{x}_i,\boldsymbol{w}_i,\boldsymbol{z}_i,\boldsymbol{o}_i)\wedge\varphi_i$ 为 $q(\boldsymbol{z})$在 T_i 上的分查询;

$Q_i(\boldsymbol{x}_i,\boldsymbol{z}_i)=\text{RewriteTree\_OR}(q_i,C)$;

end for

$Q(\boldsymbol{z})=\exists \boldsymbol{w}.(\wedge_{i=1..m}Q_i(\boldsymbol{x}_i,\boldsymbol{z}_i))$;

算法 2　RewriteTree_OR(q,C)**算法**

输入:查询 $q(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},\boldsymbol{o}.\varphi(\boldsymbol{x},\boldsymbol{w},\boldsymbol{z},\boldsymbol{o})\wedge\varphi$,$C$ 为 q 中关系的主外键约束;

输出:Q,q 的一致性查询重写

设 T 为 q 的连接图,$R(\underline{\boldsymbol{x}},\boldsymbol{y},o)$为 T 的根节点关系,R 的 OR-object 关系为 R_or 且其实例大小为$|R_or|$;

if φ 仅包含一个关系谓词 then

$q(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},o.R(\underline{\boldsymbol{x}},\boldsymbol{y},o)\wedge\varphi$,$o$ 为 R 的 *or_id* 属性变量,φ 为 R 上的非等值内建谓词;

$Q(\boldsymbol{x},\boldsymbol{z})=\text{RewriteLocal\_OR}(q,C)$;

else

设 $q_l(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},o.R(\underline{\boldsymbol{x}},\boldsymbol{y},o)\wedge\varphi_0$,$o$ 为 R 的 *or_id* 属性变量,φ_0 为 R 上的非等值内建谓词,$R_1(\boldsymbol{x}_1,\boldsymbol{y}_1),\cdots,R_n(\boldsymbol{x}_n,\boldsymbol{y}_n)$为 R 在 T 中的孩子;

for $i:=1$ to n do

设 T_i 为 T 的第 i 个子树,φ_i 为 T_i 的关系合取式;$\boldsymbol{w}_i$ 和 $\boldsymbol{z}_i$ 分别为 φ_i 的存在变量向量及自由变量向量,$\boldsymbol{o}_i$ 为 φ_i 上关系的 *or_id* 属性变量向量,φ_i 则为它们上的非等值内建谓词,$q_i(\boldsymbol{x}_i,\boldsymbol{z}_i)=\exists \boldsymbol{w}_i,\boldsymbol{o}_i.\varphi_i(\boldsymbol{x}_i,\boldsymbol{w}_i,\boldsymbol{z}_i,\boldsymbol{o}_i)\wedge\varphi_i$ 为 $q(\boldsymbol{z})$在 T_i 上的分查询;

$Q_i(\boldsymbol{x}_i,\boldsymbol{z}_i)=\text{RewriteTree\_OR}(q_i,C)$;

end for

$Q(\boldsymbol{x},\boldsymbol{z})=q_l(\boldsymbol{x},\boldsymbol{z})\wedge_{i=1..n}Q_i(\boldsymbol{x}_i,\boldsymbol{z}_i)$;

if $|R\_or|>0$ then

$\boldsymbol{v}'$和$\boldsymbol{z}'$分别为 R_or 中与 R 连接其子节点属性对应的变量向量及其他非键属性对应的变量向量,

用 $\boldsymbol{v}'$和$\boldsymbol{z}'$中相关新变量替换φ_0中原有变量得到新内建谓词公式 φ'_0;

$Q(\boldsymbol{x},\boldsymbol{z})=q_l(\boldsymbol{x},\boldsymbol{z})\wedge(\wedge_{i=1..n}Q_i(\boldsymbol{x}_i,\boldsymbol{z}_i))\vee\exists \boldsymbol{w},o.R(\boldsymbol{x},\boldsymbol{y},o)\wedge\forall \boldsymbol{v}'\exists \boldsymbol{z}'.R\_or(o,\boldsymbol{v}',\boldsymbol{z}')\rightarrow\wedge_{i=1..n}Q_i(\boldsymbol{x}_i,\boldsymbol{z}_i)\wedge\varphi'_0)$;

end if

end if

算法 3　RewriteLocal_OR(q,C)**算法**

输入:查询 $q(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},o.R(\underline{\boldsymbol{x}},\boldsymbol{y},o)\wedge\varphi$,而 C 为 R 的主键约束;

输出:Q_l,q 的本地一致性查询重写

对查询 q,o 为 R 的 *or_id* 属性变量,φ 为 R 上的非等值内建谓词,R 的 OR-object 关系为 R_or 且其实例大小为$|R_or|$;

$\lambda=\gamma=\varnothing$;

for $i:=1$ to $|\boldsymbol{y}|$ do

if $\boldsymbol{y}[i]$上的变量 u 出现在$\boldsymbol{z}$ 中 then$\lambda=\lambda\wedge u\neq\text{null}$;

else if $\boldsymbol{y}[i]$上的变量 v 出现在$\boldsymbol{w}$ 及φ 中 and $v\notin\boldsymbol{x}$ then $\gamma=\gamma\vee v=\text{null}$;

end for

$Q_l(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},o.R(\underline{\boldsymbol{x}},\boldsymbol{y},o)\wedge\lambda$;

if$\gamma\neq\varnothing$and $|R\_or|>0$ then

$\boldsymbol{v}'$和$\boldsymbol{z}'$分别为R_or 与γ 对应的变量向量及其他非键属性对应的变量向量,用 $\boldsymbol{v}'$中新变量替换φ 中原有变量得到新内建谓词公式φ';

$Q_l(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},o.R(\underline{\boldsymbol{x}},\boldsymbol{y},o)\wedge\lambda\wedge(\varphi\vee(\gamma\wedge\forall \boldsymbol{v}'\exists \boldsymbol{z}'.R\_or(o,\boldsymbol{v}',\boldsymbol{z}')\rightarrow\varphi'))$

else

$Q_l(\boldsymbol{x},\boldsymbol{z})=\exists \boldsymbol{w},o.R(\underline{\boldsymbol{x}},\boldsymbol{y},o)\wedge\lambda\wedge\varphi$

end if

算法 ConsRewrite_OR 对 q 连接图的不同连通分量逐一调用 RewriteTree_OR 得到每个分量的查询重写 Q_i 并合取连接获取一致性查询重写 Q;RewriteTree_OR 分两种情况处理分量:对单个关系上的查询调用本地重写算法 RewriteLocal_OR 得到一致性查询重写 Q_i,主要是增加自由变量非空选择及更新内建谓词检查 OR-object 关系一致性的查询合取项;而对涉及多个关系的查询,首先划分出根节点及各子树的查询部分,然后调用递归算法 RewriteTree_OR 获取各子树查询重写并形成合取项,最后增加根节点的 OR-object 关系一致性检查析取项来得到 Q_i. 容易验证例 3 中的一致性查询重写 Q_1,Q_2 和 Q_3 能够通过 ConsRewrite_OR 获取.

4　一致性查询执行代价比较

考虑符合 C_{forest}的 SPJ(Select-Project-Join)查询 $q(\boldsymbol{z})=\exists \boldsymbol{w}R_1(\boldsymbol{x}_1,\boldsymbol{y}_1)\wedge\cdots\wedge R_n(\boldsymbol{x}_n,\boldsymbol{y}_n)\wedge\varphi$,$R_1\cdots R_n$ 为不重复的 n 个关系,它的 nonkey-to-key join 都是完全并且不构成环(即连接图为森林,连通分支为树). 由 RewriteConsistent 和 ConsRewrite_OR 分别得到 q 的一致性查询重写 Q 和 Q',采用线性工作度量[12]作为它们的代价模型并把扫描关系元组数作为度量标准,则它们的查询代价 Cost(Q)和 Cost(Q')根据连接图形成了不同的取值范围,注意,$T_1,\cdots,T_K(0\leqslant K\leqslant n-1)$为查询递归执行中形成的临时关系而 $R\_or_1,\cdots,R\_or_n$ 为与 $R_1\cdots R_n$ 对应的 OR-object 关系.

$$c\sum_{i=1}^{n}|R_i|\leqslant\text{Cost}(Q)\leqslant c\sum_{i=1}^{n}|R_i|^2+c\sum_{k=1}^{K}|T_k| \tag{1}$$

$$c\sum_{i=1}^{n}|R_i|\leqslant\text{Cost}(Q')\leqslant c\sum_{i=1}^{n}(|R_i|+|R\_or_i|)+c\sum_{k=1}^{K}|T_k| \tag{2}$$

式(1)和(2)的下界代表连接图为包含 n 棵树(仅有根

节点)的森林时,查询返回的自由向量 z 仅出现关系 $R_1\cdots R_n$ 的主键上,且 $R_1\cdots R_n$ 之间的关系连接仅为 key-to-key join,根据算法 RewriteConsistent 和 ConsRewrite_OR,Q 和 Q' 的查询代价相等即为 $R_1\cdots R_n$ 的元组数之和;如果 $\vec{z}$ 中有自由变量出现在 R_i 非主键上,Q 和 Q' 为了保证 R_i 非键属性值的一致性而花费的查询代价就不再相等而分别为 $c|R_i|^2$ 和 $c|R_i|$.式(1)和(2)的上界则表示当连接图为包含 n 个节点的树(即只有一个连通分支)时,q 涉及 $R_1\cdots R_n$ 这 n 个关系的连接,Q 和 Q' 在保证每个 R_i 一致的同时递归地形成临时关系,一致性检查是 Q 和 Q' 的最大不同. Q 对 R_i 的一致性检查是在 R_i 本身中完成的,其查询代价为 $c|R_i|^2$;而 Q' 则是通过 R_i 的 OR-object 关系 R_or_i 完成一致性检查,所以其查询代价仅为 $c(|R_i|+|R\_or_i|)$.

对一般查询 q,其连接图可能包含多个连通分量而查询代价则是介于上下界之间,如果在 $R_1\cdots R_n$ 中出现某一个关系 R_i 单独形成树的情况,并且 z 中变量出现在 R_i 的非主键上,此时 Q 和 Q' 在 R_i 上的查询代价是 $c|R_i|^2$ 和 $c|R_i|$,但如果 R_i 出现在多节点的树中即 R_i 涉及与其他关系连接时,Q 和 Q' 在 R_i 上的查询代价则是 $c|R_i|^2$ 和 $c(|R_i|+|R\_or_i|)$.通常,$|R_i|>>|R\_or_i|$ 甚至对 $|R_i|$ 来说 $|R\_or_i|$ 可以忽略不计,所以 $\mathrm{Cost}(Q)>>\mathrm{Cost}(Q')$.

5 实验

为了比较 ConsRewrite_OR 与 RewriteConsistent 产生的一致性查询在 Oracle 10g 数据库上的执行代价,我们基于 TPC-H 测试基准在 Oracle 10g 上分别创建 OR-database 全局模式与普通全局模式,然后在开源数据生成工具 DBGen 的基础上通过修改源代码为两个集成数据库分别产生大小为 0.25GB,0.5GB,1GB,2GB,4GB 的不一致数据.实验运行环境为 IBM THINKPAD T400,2.8 GHz Intel Core 2 Duo CPU,4GB DDR3 RAM,Seagate 160 GB (7200RPM),操作系统为 Windows 7.

首先,考虑 lineitem 表上的 SP 查询及 lineitem,part,partsupp,supplier 表上的 SPJ 查询,选择数据库大小为 1GB 且不一致数据比例从 5% 到 25% 变化,分别比较原始查询,算法 RewriteConsistent 的一致性查询重写 1 及算法 ConsRewrite_OR 的一致性查询重写 2 的查询执行代价,实验结果如图 2 和图 3 所示.

显然,对单表 SP 查询来说,查询重写 2 的执行代价与原查询相等并且基本不受不一致比例大小的影响,而查询重写 1 的执行代价至少是前者的两倍且随着不一致比例的增大快速上升;对 4 个表的 SPJ 查询,查询重写 2 的执行代价大于原查询并随着不一致比例增大逐渐上升,但明显低于查询重写 1 的执行代价.显然,单表的查询重写 2 由于不需要检查 OR-object 关系一致性仅在原查询基础上增加投影属性非空的条件而与原查询在执行代价上基本相等;多表的查询重写 2 则因为需要检查有关 OR-object 关系一致性而造成在执行代价上大于原查询.但无论哪种情况,查询重写 2 的执行代价都大大低于查询重写 1 的执行代价.

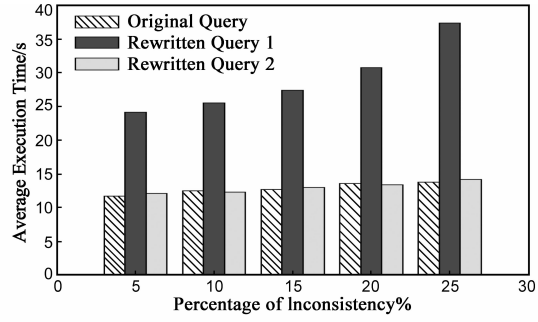

图2 单表SP查询上的性能对比

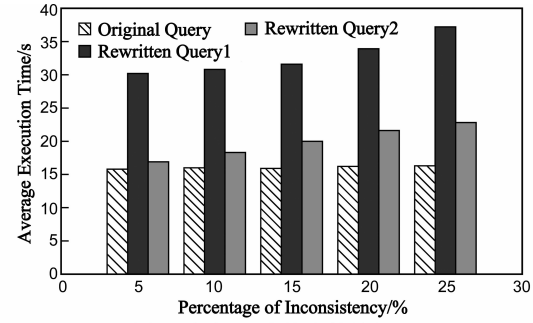

图3 多表SPJ查询上的性能对比

接着,对同样的单表 SP 查询和多表 SPJ 查询,实验比较它们在数据不一致比例为 15% 而数据库大小从 0.25GB 到 4GB 变化时的查询重写执行代价.实验结果图 4 及图 5 验证查询重写 2 的执行代价在单表 SP 查询和多表 SPJ 查询上不同的同时更清楚地表明了它们执行代价的整体变化趋势,即查询重写 2 的在执行代价增长速度上明显慢于查询重写 1.

最后,实验选择数据库大小为 1GB 并且数据不一致比例为 15% 的情况下,比较查询涉及关系数从 1 到 6 变化时查询重写 1 和 2 的平均执行代价(原始查询 SQL 详见附录 B).这里,实验选择元组最多的 lineitem 关系作为基础,每次扩充一个新的连接关系来实现关系数的递增,实验结果图 6 再次表明查询重写 2 的执行代价虽然略高于原查询但却远低于查询重写 1.这里需要特别说明的是,TPC-H 标准仅包含 8 个关系,由于后面增加的三个关系与前面关系相比在实例元组数上明显小很多,所以造成了图 6 中关系数从 4 到 6 增长时它们的平均执行代价增长速度呈现出不同程度的下降趋势.

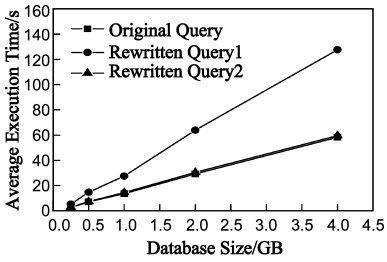

图4　SP查询的性能变化趋势

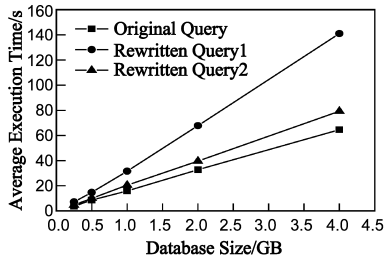

图5　SPJ查询的性能变化趋势

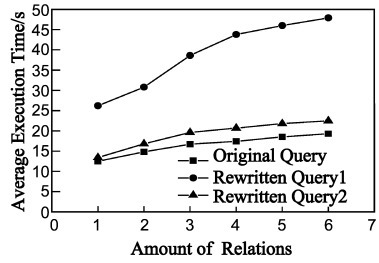

图6　关系数上的性能变化趋势

6　结论

本文重点考虑了在数据集成环境下如何有效提高不一致数据库上一致性查询的执行效率,对合取查询类 C_{forest} 提出了基于 OR-database 集成关系模式的一致性查询重写算法,采用线性工作度量的查询代价分析以及在 Oracle 10g 数据库上的模拟实验都表明,与 Fuxman 的 RewriteConsistent 算法产生的一致性查询重写相比,本文算法 ConsRewrite_OR 产生的一致性查询重写在 OR-database 集成数据库上会有更高的执行效率,从而使得本文的查询重写方法在数据集成环境下更适合用来解决不一致集成数据库上的一致性查询问题.

参考文献

[1] Chomicki J. Consistent query answering: Five easy pieces[A]. Proceedings of the 11th International on Database Theory[C]. Berlin: Springer, 2007. 1 – 17.

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

[3] Fuxman A, Miller R J. First-order query rewriting for inconsistent databases[J]. Journal of Computer and System Sciences, 2007, 73(4): 610 – 635.

[4] Fuxman A, Fazli E, Miller R J. Conquer: Efficient management of inconsistent databases[A]. Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data [C]. New York: ACM, 2005. 155 – 166.

[5] Wijsen J. On the consistent rewriting of conjunctive queries under primary key constraints[J]. Information Systems, 2009, 34 (7): 578 – 601.

[6] Wijsen J. Certain conjunctive query answering in first-order logic[J]. ACM Transactions on Database Systems (TODS), 2012, 37(2): 9.

[7] Eiter T, Fink M, Greco G, et al. Repair localization for query answering from inconsistent databases[J]. ACM Transactions on Database Systems (TODS), 2008, 33(2): 10. 1145 – 1366

[8] Caniupún M, Bertossi L. The consistency extractor system: Answer set programs for consistent query answering in databases [J]. Data & Knowledge Engineering, 2010, 69(6): 545 – 572.

[9] Imielinski T, Van Der Meyden R, Vadaparty K. Complexity tailored design: A new design methodology for databases with incomplete information[J]. Journal of Computer and System Sciences, 1995, 51(3): 405 – 432.

[10] Lenzerini M. Data integration: A theoretical perspective[A]. Proceedings of the Twenty-first ACM Symposium on Principles of Database Systems[C]. New York: ACM, 2002. 233 – 246.

[11] Libkin L, Sirangelo C. Data exchange and schema mappings in open and closed worlds[J]. Journal of Computer and System Sciences, 2011, 77(3): 542 – 571.

[12] Lee K Y, Son J H, Kim M H. Reducing the cost of accessing relations in incremental view maintenance[J]. Decision Support Systems, 2007, 43(2): 512 – 526.

作者简介

张晓刚　男,1974 年生于内蒙古,中南大学信息科学与工程学院博士研究生,主要研究方向为数据库、数据集成.
E-mail: jnuzxg@gmail.com

杨路明　男,1947 年生于江西,中南大学信息科学与工程学院教授,博士生导师,主要研究方向为信息系统、数据库技术.

潘久辉(通信作者)　男,1956 年生于湖南,暨南大学计算机系教授,博士生导师,主要研究方向为数据库、数据集成和程序语言理论.
E-mail: jhpan@jnu.edu.cn