第34卷 第10期
2011年10月

计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

Vol. 34 No. 10
Oct. 2011

一种优化关系型溯源信息存储的新方法

王黎维[1)] 鲍芝峰[2)] KOEHLER Henning[3)] 周晓方[3),4)] SADIQ Shazia[3)]

[1)](武汉大学国际软件学院 武汉 430072)

[2)](新加坡国立大学计算机学院 新加坡 117417)

[3)](昆士兰大学信息技术与电子工程学院 澳大利亚 4072)

[4)](数据工程与知识工程教育部重点实验室(中国人民大学) 北京 100872)

摘 要 现代数据管理必须处理来源不同、质量各异的数据,因此从系统层面支持数据溯源,让用户了解数据的来源及派生过程成为当前至关重要的一个研究课题.基于标注的方法是支持数据溯源的基本方法之一.这种方法的主要问题是存储空间开销,因为溯源信息可能会超过实际数据的大小.在该文中,作者提出了一个用与查询结构匹配的溯源树来表达和存储溯源信息从而避免数据派生过程中冗余存储的基本框架.基于这个框架,作者提出了一系列针对关系型查询的存储优化方法,选择查询树部分节点来存储溯源信息.这些优化算法对于查询大小是多项式时间,对于溯源信息大小是线性时间,在溯源信息的跟踪和优化方面均不会产生巨大的开销.这一框架是数据溯源研究的一个新思路,有着广泛的应用前景.

关键词 溯源树;溯源表;存储优化;最优削剪;规则 I&II 削剪

中图法分类号 TP311 **DOI号**:10.3724/SP.J.1016.2011.01863

An Approach for Optimizing Relational Provenance Storage

WANG Li-Wei[1)] BAO Zhi-Feng[2)] KOEHLER Henning[3)] ZHOU Xiao-Fang[3),4)] SADIQ Shazia[3)]

[1)](*International School of Software, Wuhan University, Wuhan* 430072)

[2)](*School of Computing, National University of Singapore, Singapore* 117417)

[3)](*School of Information Technology and Electrical Engineering, The University of Queensland, Australia* 4072)

[4)](*Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China) of Ministry of Education, Beijing* 100872)

Abstract Modern data management has to deal with data from different sources with different quality, therefore, supporting data provenance in the system level and allowing users to know where data comes from and how it was derived have become a critical research topic. Annotation is one of approaches to track provenance. However, storing fine-grained annotations can be expensive as the complete annotations for the data may outsize the storage space required for the data itself. In this paper, we propose a framework for storing provenance information relating to data derived via relational queries, using provenance trees which match the query structure to avoid redundant storage of information about the derivation process. Within this framework, we come up with a series of storage optimization methods against the relational queries to make good choices of query tree nodes where provenance information should be stored. Our optimization algorithms run in time polynomial in the query size and linear in the size of the provenance, thus enabling provenance tracking and optimization without incurring large overheads. This framework is a new idea for the data tracing study and has a wide range of applications.

Keywords provenance tree; provenance table; storage optimization; optimal reduction; rules I&II reduction

---

收稿日期:2011-07-18;最终修改稿收到日期:2011-09-26.本课题得到教育部博士点新教师基金(200804861067)和澳洲研究院(ARC)项目基金(LP0882957)资助.**王黎维**,女,1980年生,博士,讲师,研究兴趣为数据溯源、记录链接、科学工作流管理.E-mail:liwei.wang@whu.edu.cn.**鲍芝峰**,男,1983年生,博士,主要研究兴趣为数据库关键词检索、社会网络建模与分析、溯源数据管理.**KOEHLER Henning**,男,1976年生,博士,主要研究兴趣为数据依赖理论、规范化、数据集成、数据溯源、采样和多目标优化.**周晓方**(通信作者),男,1963年生,博士,教授,中国人民大学兼职教授,主要研究方向为空间和多媒体数据库、数据质量、高性能查询处理、网络信息系统和生物信息.E-mail:zxf@uq.edu.au.**SADIQ Shazia**,女,博士,副教授,主要研究兴趣为商业过程管理、企业治理、风险和法规遵从、数据质量管理、工作流系统、面向服务的计算.

1 引 言

随着互联网和数据自动采集设备的快速发展，人们可利用的数据源以及数据量在过去的十多年里成指数倍增长，然而，数据的质量却每况愈下，究其原因，大量数据由传感器网络、RFID 读入器和机器识别系统生成以及从网上自动收集，数据的完整性、一致性以及正确性无法得到保证. 而许多科学应用和大规模数据管理应用通常需要收集和处理大量不同来源的数据，数据来源复杂，质量参差不齐，使得这些应用的数据和结果的可信度受到质疑. 在系统层面上支持数据溯源，提供对数据来源及处理步骤有效方便地查询支持可以帮助用户理解数据和结果的可信度[1].

数据溯源信息(provenance)描述数据的来源和派生过程，在数据库环境中，数据派生过程是指查询过程，而科学应用环境中的派生过程则可用整个工作流图表示. 溯源信息对于判断结果的正确性以及可信度至关重要[2]. 在数据库环境中，它可以帮助解释意想不到的结果，并有助于数据集成[3]. 在科学应用环境中，它有助于诊断实验执行过程中的错误，探索未知的实验结果[4]. 在不确定性数据库中，它可以用来追踪概率变量之间的关联[5].

数据溯源研究从类型上大致分为数据级溯源和过程级溯源，而数据库领域的研究侧重于前者[6]. 从溯源信息的存储粒度上分为粗和细两个粒度. 在粗粒度层次上完成对溯源信息的追踪时，整个数据集的工作流图或查询过程作为数据派生过程存储. 在细粒度层次上，每一个"对象"(可能指在实验中派生的单个结果，或者在数据库中一个元组或属性的值)都需要记录与它关联的溯源信息. 粒度选择往往根据应用需求而定.

虽然存储粗粒度的溯源信息不必过于关注存储空间，细粒度的溯源信息存储则带来一系列的挑战，因为细粒度溯源信息的空间需求往往远超过实际数据的存储大小[7]. 处理这个问题的方法之一是只有需要数据集的溯源信息时才通过逆过程(inversion)计算出它们，从而避免预先存储溯源信息导致的高昂存储代价[8]. 然而这一方法的缺点是如果没有高效的逆过程，计算代价会很高，同时它也可能需要存储查询中间结果来帮助计算溯源信息[9].

本文主要关注关系数据库查询中溯源信息有效存储的问题，提出了一个类似于查询树的"溯源树"的数据结构用于溯源信息存储. 在此结构中，具有相同派生过程但不同输入的元组被分在一个组，因此避免有关派生过程信息的冗余存储. 虽然本文只陈述了关于数据库查询的溯源信息的存储方法，当任意派生过程能表示成树型结构时，这种存储溯源信息的方法也能适用.

另外，由于"溯源树"这种存储结构具有较强的灵活性，文中也探讨了降低溯源信息存储成本的优化方法，即选择查询树中的部分节点而非全部节点来存储溯源信息. 由于在一个节点中不存储溯源信息可能会导致其它节点溯源信息存储规模的增长，因此，考虑在哪些节点中存储是非常重要的，也是本文论述的重点.

本文第 2 节描述"溯源树"以及总体框架；第 3 节讨论选择查询树中的部分节点来存储溯源信息的优化方法；第 4 节简要概述相关的工作；第 5 节进行实验测试以及结果分析；第 6 节总结全文.

2 溯源树

本文主要考虑数据库查询的溯源信息. 溯源信息可以在数据库中以不同粒度的形式存储，即数值粒度、元组粒度或表粒度. 对于大多数数据库查询(不涉及用户自定义函数的查询)，数值粒度的溯源信息可以以一种有效的方式从元组粒度的溯源信息派生. 元组粒度的溯源信息也可以通过重新执行查询从表粒度的溯源信息中派生，或使用文献[9]中所述的逆过程派生. 然而，涉及大数据集和复杂查询时，通过表粒度这种方法计算元组的溯源信息代价比较大，因此，通常存储元组粒度的溯源信息.

目前已提出了许多不同的方式存储元组粒度的溯源信息，最常用的方法是将数据库查询的子查询作为溯源信息的一部分存储，然而，如果有多个元组共享相同的转换过程，那么在存储空间方面将产生巨大的开销. 本文通过使用具有嵌套数据类型的树结构来存储溯源信息有效地解决了这个问题.

定义 1(溯源类型/元组/表). 一个溯源类型(provenance type)是通过基本类型 tupleID 构成的元组类型和集合类型的嵌套而构成. 它的值域包含对元组的引用和一个特殊的空值⊥.

(1) tupleID 是一个基本溯源类型，简称为基本类型，表示一个基本元组的类型.

(2) 如果 $T_1, \cdots, T_n$ 是溯源类型，那么 $(T_1, \cdots, T_n)$ 是一个元组溯源类型，简称为元组类型.

(3) 如果 T 是一个溯源类型，那么 $\{T\}$ 是一个集合溯源类型，简称为集合类型.

一个溯源类型的值被称为溯源元组(provenance tuple)，类型为 T 的溯源元组的集合形成了一个类型为 T 的溯源表(provenance table).

(1) 引用数据元组或者溯源元组的元组和 $\perp$ 是类型为 tupleID 的溯源元组.

(2) 如果 $t_1,\cdots,t_n$ 分别是类型 $T_1,\cdots,T_n$ 的溯源元组，那么 $(t_1,\cdots,t_n)$ 是类型 $(T_1,\cdots,T_n)$ 的溯源元组.

(3) 如果 $t_1,\cdots,t_m$ 是类型 T 的所有溯源元组，那么 $\{t_1,\cdots,t_m\}$ 是类型 $\{T\}$ 的溯源元组.

例 1.　r_i 表示对元组的引用，那么

$$(r_1,\{(r_2,\perp),(r_3,r_4)\},\{\{r_5,r_6,r_7\}\}),$$

是一个溯源元组，它的溯源类型为

$$T=(\text{tupleID},\{(\text{tupleID},\text{tupleID})\},\{\{\text{tupleID}\}\}).$$

表 1 是一个类型为 T 的溯源表.

表 1　溯源表

r_1	$\{(r_2,\perp),(r_3,r_4)\}$	$\{\{r_5,r_6,r_7\}\}$
r_2	$\{(r_8,r_9)\}$	$\{\{r_1\},\{r_5,r_6\}\}$

一个数据元组的溯源信息可由一个溯源元组描述. 由于中间结果表中的所有数据元组都以同样的方式派生，即使用相同的操作，因此相应的溯源元组将拥有相同的溯源类型，所以，可以将他们存储在一个溯源表中. 所有的中间结果或最终的结果的溯源表形成了一个溯源树(provenance tree).

定义 2(溯源树).　假设一个拥有节点集 N 和根节点 R 的查询树 Q，Q 的一个溯源树表示成一个二元组 $P=(P_N,P_O)$，其中 P_N 是从节点 N 到该节点的溯源表的部分映射，其中，N 中的叶节点(即基本关系表)没有被映射. P_O 是 Q 中的输出元组到 $P_N(R)$ 中的溯源元组的全部映射.

例 2 详细阐述了溯源树的定义，其中，一个溯源树不包含数值，只包含对(数据或溯源)元组的引用. 这里只存储溯源树，而不存储中间结果.

例 2.　表 2 表示关系 *PropertyForRent*，其中"*ID*"表示引用元组的数据库内部分配的元组号.

表 2　关系 *PropertyForRent*

ID	*BranchNo*	*PropertyNo*
P.1	*B*001	*P*001
P.2	*B*001	*P*002
P.3	*B*002	*P*003

下面的 SQL 语句选择出租房产多于一处的机构和房产信息.

```
SELECT p.BranchNo, p.PropertyNo
FROM PropertyForRent p,
(SELECT BranchNo, PropertyNo FROM
PropertyForRent GROUP BY BranchNo HAVING
count(*) >1) s
WHERE p.BranchNo=s.BranchNo
```

图 1 是该查询对应的查询树，括号中的数字〈·〉用于标识节点.

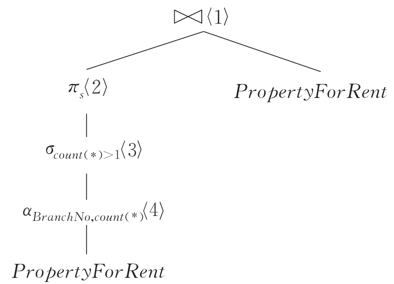

图 1　查询树

图 2 显示了例 2 执行的中间结果以及对应的溯源树. 其中，执行的中间结果显示在左边的树中，相应的溯源树显示在右边的树中. 节点 4 中 *ID* 为 4.1 的元组，它的溯源元组为({*P*.1，*P*.2}). 节点 3 中 *ID* 为 3.1 的元组，它的溯源元组为(4.1). 节点 1 中 *ID* 为 1.1 的元组，它的溯源元组为(2.1，*P*.1).

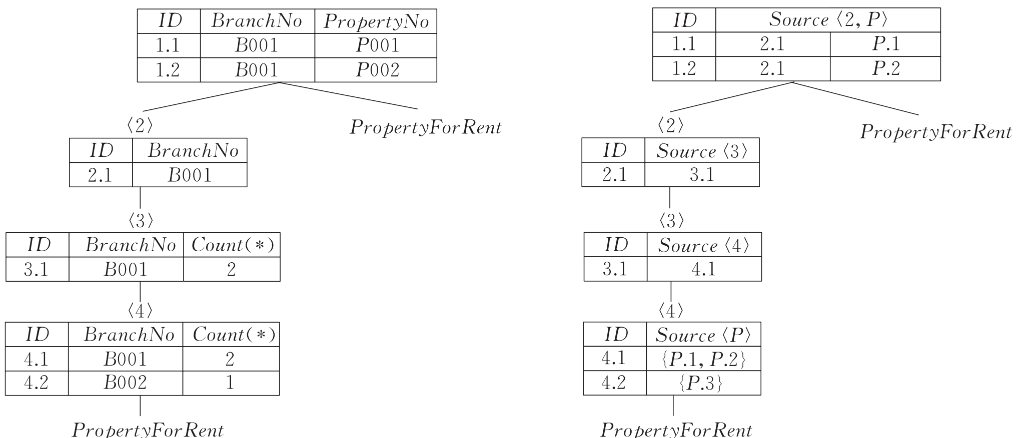
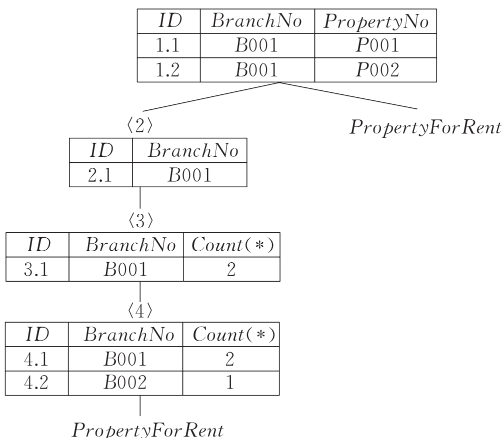

图 2　溯源树

定义 3(完全/冗余树). 一个溯源树 P 是完全(complete)的,如果每个输出数据元组被映射到一个溯源元组上,并且 P 中的溯源表上的每个不等于空值⊥的 tupleID 类型的数据项引用

(1) P 中的一个溯源元组,或者

(2) 对应于一个叶节点的基本表中的一个元组.

与输出数据元组不关联的溯源元组被称为冗余元组,即该溯源元组从没被引用过. 如果 P 中包含任何冗余元组叫树 P 是冗余的,否则就是非冗余的.

例如,例 2 中显示的溯源树是完全的,但是冗余的,因为溯源元组 4.2 没有被引用过. 本文在以下的论述中将仅仅考虑完全的溯源树.

2.1 溯源树的构造

本节描述了一个简单的方法构造完全非冗余的溯源树. 首先假设使用关系代数操作生成了一个查询树,为了迎合业界的 SQL 标准应该:

(1) 允许使用任意的选择函数.

(2) 使用多集(允许冗余)而不是集合. 在多集上的投影操作并不能消除冗余元组.

(3) 引入一个聚集操作 α,下标表示聚集的属性和分组函数.

(4) 引入一个替代操作 β,用一些其它的元组代替关系(多集)中的每个元组.

这些扩展不会对溯源信息的存储问题造成影响. 而集合上的投影操作能表达为先进行多集上的投影操作,然后再对投影之后的结果进行聚集操作. 并操作和交操作也是多集操作的一种.

每个操作的初始溯源树都能直接被构造. 每个操作的溯源类型在以下标出,即与使用这个操作派生的查询结果相关联的溯源表的溯源类型.

(1) 选择(selection)σ: tupleID;

(2) 投影(projection)π: tupleID;

(3) 差(minus)$-$: tupleID;

(4) 替代(replacement)β: tupleID;

(5) 并(union)$\cup$: (tupleID, tupleID);

(6) 连接(join)$\bowtie$: (tupleID, tupleID);

(7) 交(intersect)$\cap$: (tupleID, tupleID);

(8) 聚集(aggregation)α: {tupleID}.

本文阐述的溯源的概念与文献[6]中的术语"why-provenance"非常类似. 这表示溯源信息需要获取每个输出元组的所有输入元组. 对于选择、投影、差和替代操作,每个输出元组实际上都是从一个输入元组中计算得到. 对于连接和多集交操作,每个输出元组由两个输入元组创建. 对于并操作,一个元组由两个表中的其中一个拷贝得到,通过使用一个二元组,并用⊥值描述另一个源元组的缺失,能结构化的标识一个元组来自哪个输入关系. 多集交操作能被表达为先进行聚集操作用于消除冗余,然后进行集合交操作. 对于聚集操作,每个输出元组通过一组输入元组派生,因此,需要将所有这些元组存储起来. 当然,如果聚集操作是求最大最小值,那么只需要将最大最小的元素存储起来即可.

总之,执行查询时,首先自底向上构造初始的溯源树. 然后采用自顶向下的方式,消除多余的溯源元组. 这种方式获得的溯源树是完全非冗余的,也是进一步优化的出发点. 例如,将图 2 中多余的溯源元组 4.2 删除,便可得到完全非冗余的溯源树.

2.2 溯源树的存储

溯源树可以使用支持复杂类型,尤其是支持元组和集合类型嵌套的存储系统直接存储. 如果需要在关系数据库系统中存储溯源树,即无需使用外部结构直接处理溯源信息的计算、存储和查询,那么可以采用文献[10]中类似的方法,将溯源树的节点对应的溯源表映射成关系表的来存储溯源树. 例如,如果溯源表 R 中有一个元组 t,$(\{r_1,\cdots,r_k\})$是 t 的溯源信息,也即溯源元组,那么,关系表 R' 中将存储 k 个元组 $t_1,\cdots,t_k$,其中每个 t_i 对应一个 r_i,$1\le i\le k$. 这里的 r_i 通过元组的行号来引用元组. 本文主要探讨的是可以通过"溯源树"这种数据结构来存储溯源信息,而不是探讨溯源信息具体的物理存储,且本文的实现主要基于 Java 开发的一个关系查询引擎,因此,本文未着重关注溯源信息的物理存储结构.

3 优化溯源树

只在查询树的部分节点而非全部节点中存储溯源信息将会使得存储更高效.

3.1 极端的解决方法

当前的应用通常使用两种对立且极端的溯源信息储存方法. 第 1 种方法称为"存储最终",它只使用一个溯源表记录最终结果元组与基本元组之间的引用关系. 第 2 种方法称为"存储全部",这是另一个极端,因为它为每一个中间结果,即查询树的每个节点都存储一个溯源表.

一些现有的方法可以归纳到以上的两种类别中. 这些方法存储每次"转换"之后的溯源信息,这些信息是任意粒度的[11-12]. 将整个查询作为一个转换等同于"存储最终". 查询也可以使用 $\sigma,\pi,\bowtie$ 等操作来构成,如果将每个操作作为一个转换,那么就等同于"存储全部". 当然,也可以将查询分解成子查

询，对待每一个子查询作为一个转换. 因此，关键是找到一个很好的分解，将其划成子查询.

首先要分析存储最终和存储全部在总体性能上都比较差.

例 3.　考虑一个查询 $\sigma_1(\sigma_2(\cdots(\sigma_n(R))))$ 由 n 个连续的选择操作构成（或者其它具有 tupleID 类型的操作）. 那么"存储全部"的方法所需的存储空间是"存储最终"的 n 倍.

例 4.　考虑一个查询 $R_1 \bowtie \alpha(R_2)$ 和实例 r_1，r_2 以至于 r_1 和 r_2 分别包含 n 个元组，r_2 中所有的元组被聚集成一个元组 t_2，并且 r_1 中所有的元组都和 t_2 连接. 那么"存储全部"方法仅仅需要节点 $\alpha(R_2)$ 上的 n 个引用，加上最终节点 $R_1 \bowtie \alpha(R_2)$ 上的 n 个引用，然而，"存储最终"方法需要在最终节点上 n^2 个引用. 因此，"存储最终"方法需要 $\frac{n}{2}$ 倍"存储全部"的存储空间.

结合以上的两个例子，如 $\sigma_1(\sigma_2(\cdots(\sigma_n(R_1 \bowtie \alpha(R_2)))))$，两种极端的方法都不能产生很好的结果.

由于这两种方法总体上都不是很好的解决方法，下面将探讨另一种方法，它只在查询树上的几个但不是所有的节点上存储溯源表. 这里的关键问题是选择哪些节点存储溯源表.

3.2　基于规则的溯源树削剪

对于削剪方法，首先根据 2.1 节的描述来构造"存储全部"的溯源树. 然后通过将多个表中的溯源信息合并为一个溯源表来改进它. 通过复制溯源元组，而不是引用它们来合并溯源表，显然有益于存储空间. 这就引入了以下规则.

规则 I.　当一个溯源表中的所有元组最多被引用一次时，复制所有元组的溯源信息而不是引用它们.

规则 II.　当引用一个 tupleID 类型的溯源表时（即被引用的元组只包含一个引用），复制被引用元组的溯源信息.

规则 I 表示不引入任何冗余，并避免存放额外的溯源表. 考虑规则 II 的原因是引用一个 tupleID 类型的溯源信息不会比复制它所用的存储空间更小，同时也避免了存储额外的溯源表. 引用被复制后，立即删除不再被引用的溯源元组，也就是不与输出元组对应的元组. 构建削剪的溯源树的方法一般包括 3 个步骤：

1. 构造初始的溯源树.
2. 移除不被引用的溯源元组.
3. 使用规则 I 和规则 II 合并溯源表.

对于最后一步，规则 I 和 II 可以以不同的顺序应用，这导致了不同的结果. 根据 2.1 节，初始的溯源树是完全非冗余的，因此，将图 2 中的溯源元组 4.2 删除后得到的是初始的溯源树，再对该树以不同的顺序应用规则 I 和 II，可以得到图 3 中显示的两个削剪的溯源树. 其中，左面的树是先应用规则 II 得到的，而右边的树是先应用规则 I 得到的.

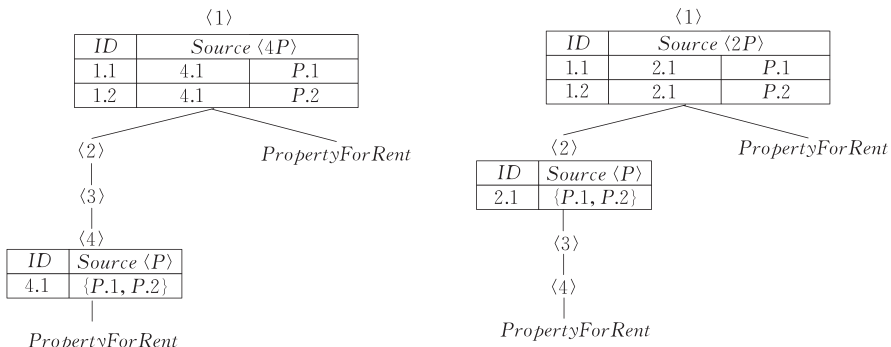

图 3　削剪的溯源树

左右两个削剪的溯源树虽然形式不同，但却是同构的，因此，可以将溯源表移动到溯源树上不同的节点中存储，并重新命名引用，但不能改变它的内容. 为了避免过于复杂的同构的定义，首先将溯源树转换成引用树（reference tree），然后定义引用树之间的同构概念.

定义 4（子类型/子元组）.　溯源类型（元组）之间的子类型（子元组）关系 $<P$（$<T$）是以下规则的传递扩展.

（1）$p_i < P(p_1, \cdots, p_n)$；　（1）$t_i < T(t_1, \cdots, t_n)$；

（2）$p < P\{p\}$；　（2）$t_i < T\{t_1, \cdots, t_n\}$.

如果一个子类型（元组）直接遵循以上规则，而不是通过传递扩展得到，那么称它为直接子类型（元组）.

定义 5（引用树）.　溯源树 P 的引用树是一个有向无环图（DAG），它由以下方式构建. 每个被引用的基本元组为一个节点. 对每个溯源元组 t 增加一个节点，节点上标记它所在溯源表的位置，同时，

为 t 的每个子元组增加一个无标号的节点.最后,在每个被引用的元组和引用它的元组之间增加一条弧,同时也为每个直接子元组关系增加弧.对于形式(1)中的直接子元组关系,用子元组在溯源元组 t 中的相应的位置 i 来标记弧.

将图3中削剪的溯源树转换成了引用树,如图4所示.

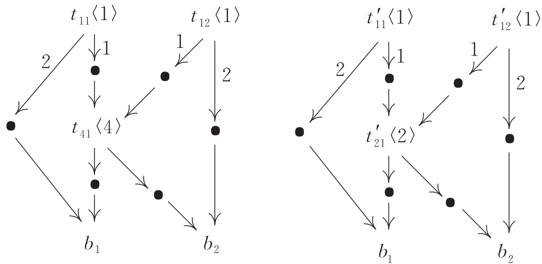

图4 引用树

虽然在图4中,无标记的节点·似乎是没必要的,但是,当溯源元组包含嵌套的集合类型时却非常重要.例如,({tupleID},{tupleID}).

定义6(同构). P_1 和 P_2 是同一个查询树上的两个溯源树,双射 ϕ:tupleID→tupleID 是 P_1 和 P_2 的同构映射,如果

(1) ϕ 是 P_1 和 P_2 的引用树之间的一个图同构映射,其中 P_1 和 P_2 的引用树保留弧标记且与节点标记相容,即节点有相同的标记.节点有相同的标记当且仅当通过 ϕ 的映射,两节点同像;

(2) ϕ 固定基本(输入)元组,并且每个输出元组 t 被映射到相应的溯源元组:$\phi(P_1(t))=P_2(t)$.

如果存在一个这样的 ϕ,那么称 P_1 和 P_2 是同构的.

容易看出同构是一个等价关系,如果 ϕ_{12} 是 P_1 和 P_2 的同构映射,ϕ_{23} 是 P_2 和 P_3 的同构映射,那么 $\phi_{23}\circ\phi_{12}$ 是 P_1 和 P_3 的同构映射.

定理1. P 是一个溯源树.以任意的顺序应用规则 I 和规则 II 削剪 P,直到不能再应用任何规则为止,那么由此产生的削剪的溯源树是同构的.

证明. 规则 I 和 II 的应用,可以被看作是同构的溯源树上的削剪步骤.根据著名的 Church-Rosser 定理[13],当两个不同的削剪步骤应用到同构的溯源树上并由此产生不同的溯源树时,可以通过进一步应用削剪步骤使它们达到同构.每一个削剪步骤(即规则 I 和 II 的应用)与一个非空的溯源表相关联,并且应用削剪步骤到相应的溯源表保留了溯源树之间的同构关系.接下来对一个溯源树进行削剪,使之产生一个不同但同构的溯源树.

现在,假设 P_1 和 P_2 是两个同构的溯源树,δ_1、δ_2 是两个分别应用到 P_1 和 P_2 中的溯源表 T_1、T_2 上的削剪步骤.如果 T_1 和 T_2 不邻近,也就表示 T_1 和 T_2 之间没有直接引用关系,那么 δ_1 和 δ_2 并不冲突,即 δ_1 的应用不会阻碍随后 δ_2 的应用,反之亦然.因此,应用 δ_2 到 $\delta_1(P_1)$ 以及应用 δ_1 到 $\delta_2(P_2)$ 可以获得同构的溯源树.

最后,假设 T_1 和 T_2 邻近,且 $T_1\rightarrow T_2$.如果 T_1 中的每个元组仅被引用一次(δ_1 使用规则 I),δ_2 的应用不可能改变这个性质,所以 δ_1 和 δ_2 并不冲突.同样的,如果 T_2 的溯源类型是 tupleID(δ_2 使用规则 II),δ_1 的应用也不能改变这个性质,因此 δ_1 和 δ_2 仍然是不冲突的.同理,应用 δ_2 到 $\delta_1(P_1)$ 以及应用 δ_1 到 $\delta_2(P_2)$ 可以获得同构的溯源树.

还需要考虑一种情况,如果 T_1 具有类型 tupleID(δ_1 使用规则 II),同时,T_2 中的每个元组被仅仅引用一次(δ_2 使用规则 I),那么,应用了 δ_1 之后,T_2 中的元组可能被引用多次,因此,规则 I 不能被应用.同样的,当应用了 δ_2 之后,新产生的 T'_1 不具有类型 tupleID,因此,规则 II 不能被应用.所以,δ_1 和 δ_2 是冲突的.然而,$\delta_1(P_1)$ 和 $\delta_2(P_2)$ 已经是同构的:与它们相关的同构映射 ϕ' 能被构造成 $\phi'=\phi_{\Delta}\circ\phi$,其中 ϕ_{Δ} 将 T_2 中每个元组的 tupleID 类型映射到引用这个元组的 T_1 中的元组的 tupleID 类型上.

削剪一个溯源树可以在 $O(N)$ 时间内完成,其中 N 是初始的溯源树中元组引用的总数.

3.3 最优溯源树削减

虽然上面所述的削剪规则算法执行速度快且容易实现,但由此产生的削剪溯源树可能不是最优的.3.3节将描述一个多项式时间算法,当溯源树的确是一个树,而不是一个任意的 DAG 时,该算法采用了动态规划方法来寻找最优解.

查询树中除根和叶节点之外的每个节点都必须考虑是否需要存储溯源表,需要存储的节点称为物化节点.这里采用 tupleID 类型的数量来衡量溯源表的大小,因此每个溯源表的大小是指向每个最近的物化的后代节点中元组的 tupleID 数量的总和.

决定是否物化一个节点 N 时,只需要考虑其最近的物化祖先以及最近的物化后裔,而不必考虑任何兄弟节点,因为祖先的大小将受到 N 的影响,而后裔会影响节点 N 的大小.这里称任何非祖先非后代的节点为兄弟节点.虽然兄弟节点的物化,可能会影响到一个共同的最近的物化祖先的大小,但这种物化对祖先表大小的影响是相互独立的.首先介绍一些概念.

定义7(子树). T 是一个树,N_1,N_2 是 T 上的节点,N_1 是 N_2 的一个祖先(考虑每个节点都是它自

己的祖先和后裔)，表示为 $N_1 \to N_2$. 集合 (N_1, N_2) 包含除 N_1 和 N_2 之外的 N_1 的后裔和 N_2 的祖先.

$T[N_1]$ 为 T 的子树，包含 N_1 的所有后裔，同时 $T[N_1 \to N_2]$ 为 $T[N_1]$ 的子树，包含所有 N_2 的祖先和后裔.

假设 T 中节点的一个集合 S，并且 $s_1, s_2 \in S$，如果 $s_1 \to s_2$，$s_1 \neq s_2$，并且不存在 $s' \in S \setminus \{s_1, s_2\}$，$s_1 \to s' \to s_2$，那么 s_1 是 s_2 的直接祖先，表示成 $s_1 \to^d s_2$.

例 5. 图 5 从左到右显示了树 T 以及它的两个子树 $T[N_2]$ 和 $T[N_2 \to N_5]$.

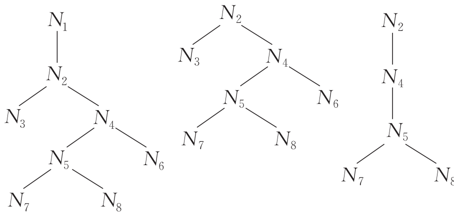

图 5　子树

对于 $S=\{N_1, N_4, N_5, N_8\}$，节点 N_1 是 N_4 的直接祖先，是 N_5、N_8 的祖先但非直接祖先.

定义 8(部分结果). P 是初始的溯源树，N_1 和 N_2 是 P 上的节点且 $N_1 \to N_2$. $ref(N_1, N_2)$ 表示 N_1 的溯源表对 N_2 中的溯源元组引用的数量. 假设需要物化 P 中节点的一个集合，S 的代价是

$$cost(S) := \sum_{s_1 \to^d s_2} ref(s_1, s_2).$$

如果 S 包含一个子树 P' 的根和所有叶节点，那么称 S 对于 P' 是有效的. $opt(N_1)$ 表示在 $P[N_1]$ 的所有有效的节点集合中最小化 $cost(S)$ 的集合 S，$opt(N_1, N_2)$ 表示在 $P[N_1 \to N_2]$ 的所有有效的节点集合中最小化 $cost(S)$ 的集合 S 且 $S \cap (N_1, N_2) = \varnothing$，即 N_1 和 N_2 之间没有任何物化节点.

可以将 $opt(N, N)$ 简写成 $opt(N)$. 现在需要寻找 $opt(root)$，其中的 $root$ 是溯源树的根，即需要寻找一个物化节点集使得溯源信息的存储成本最小.

定理 2. N_1 是 N_2 的祖先，N 是一个节点.

(1) 如果 N_2 是一个叶节点(基本表)，那么 $opt(N_1, N_2) = \{N_1, N_2\}$.

(2) 如果 N 不是一个叶节点，那么 $opt(N) = opt(N, C_1) \cup \cdots \cup opt(N, C_k)$，其中 $C_1, \cdots, C_k$ 是 N 的孩子节点.

(3) 如果 $N_1 \neq N_2$ 并且 N_2 不是一个叶节点，那么 $opt(N_1, N_2) = \{N_1\} \cup opt(N_2)$ 或者 $opt(N_1, N_2) = opt(N_1, C_1) \cup \cdots \cup opt(N_1, C_k)$，其中 $C_1, \cdots, C_k$ 是 N_2 的孩子节点.

证明.

(1) 根据定义 8，$opt(N_1, N_2)$ 一定包含节点 N_1 和 N_2，并且 N_1 和 N_2 之间没有任何物化节点，因为，N_2 是一个叶节点，因此 $opt(N_1, N_2)$ 只包含节点 N_1 和 N_2.

(2) 因为 N 不是一个叶节点，因此，根据定义 8，$opt(N)$ 表示包含根节点 N 和 N 的所有后裔节点集合中最小化 $cost(S)$ 的集合 S. 同时，$opt(N, C_i)$，$1 \le i \le k$，表示包含根节点 N 以及 C_i 的所有后裔节点集合中最小化 $cost(S_i)$ 的集合 S_i，并且 N 和 C_i 之间没有任何物化节点，$cost(S_i) := \sum_{s_1^i \to^d s_2^i} ref(s_1^i, s_2^i)$，$1 \le i \le k$，且 s_1^i 和 s_2^i 是 $N \cup \{C_i$ 的所有后裔节点集$\}$. 那么 $cost(S) := \sum_{i=1}^{k} \sum_{s_1^i \to^d s_2^i} ref(s_1^i, s_2^i)$，$1 \le i \le k$，由此可得，$cost(S) := \sum_{i=1}^{k} cost(S_i)$，$1 \le i \le k$，因此，$S = S_1 \cup \cdots \cup S_k$.

(3) 根据定义 8，$opt(N_1, N_2)$ 是包含节点 N_1 以及 N_2 的所有后裔节点中的最优解，又因为 $N_1 \neq N_2$，且 N_2 不是一个叶节点，如果最优解中包含 N_2，那么，$opt(N_1, N_2) = \{N_1\} \cup opt(N_2)$；如果最优解中不包含 N_2，结合定理 2(2)，那么 $opt(N_1, N_2) = opt(N_1, C_1) \cup \cdots \cup opt(N_1, C_k)$.

这个定理可以转换成算法 1，该算法能通过存储 $opt(N_1, N_2)$ 和 $cost(opt(N_1, N_2))$，而不是重新计算来找到最小物化节点集.

算法 1. DP-opt.

输入：P, N_1, N_2

输出：$opt(N_1, N_2)$

1. 如果 N_2 是一个叶节点，那么返回 $\{N_1, N_2\}$.
2. $O_2 :=$ DP-opt$(P, N_1, C_1) \cup \cdots \cup$ DP-opt(P, N_1, C_k).
3. 如果 $N_1 = N_2$，那么返回 O_2.
4. $O_1 := \{N_1\} \cup$ DP-opt(P, N_2, N_2).
5. 如果 $cost(O_1) \le cost(O_2)$，那么返回 O_1.
6. 否则返回 O_2.

引理 1. n 表示节点的数量，N 是初始溯源树中元组引用的数量. 算法 1 能在 $O(n \cdot N)$ 时间内计算出具有最小存储代价的溯源树.

证明. 给定一个节点 N，以自顶向下的方式计算出节点 N 对任何后裔表 N' 中每个元组的引用数量，对每个后裔表，计算总的引用数量，得到 $ref(N, N')$，这需要 $O(N)$ 的时间. 为所有的祖先-后裔对计算 $ref(N_1, N_2)$ 可能需要 $O(n \cdot N)$ 时间. 然后，每次算法 DP-opt 的调用可在固定时间内进行. 如果存储算法 1 的中间计算结果，那么最多需要花费 $O(n^2)$ 时间完成算法的调用，将以上 3 个时间求和，显然，$O(n \cdot N)$ 是算法 1 完成的时间复杂度.

以上提出的动态规划方法只适用于树型查询图. 今后将进一步探讨是否存在优化任意查询 DAG

的一个多项式时间算法.

3.4 对比研究

本节将探讨由简单的削剪规则获得溯源表的存储大小是否接近3.3节中的最优解.考虑如下存储溯源信息的策略:

(1)最优削剪.在部分节点上存储溯源表,使得存储空间最小.

(2)基于规则的削剪.对所有的溯源表应用一些削剪规则:

①完全削剪.仅在根节点上存储溯源表.

②不削剪.在所有节点上存储溯源表.

③使用规则I和/或规则II的削剪.

不同削剪规则之间的区别可能很大,见例3和例4.显然,最优的削剪方法至少不能比任何基于规则的削剪方法差.为了简化分析,假定存储引用的集合(类型为{tupleID}的一个值)需要至少相当于一个额外的单独引用的存储空间.假设不同的(合理的)大小可以影响将在以下确立的存储空间因子的范围,但仍然在一个固定的有限边界内.

定理3. 基于规则II的削剪方法需要少于最优削剪方法2倍的存储空间.

证明. 规则II应用到溯源树中后,每个被引用的溯源元组都具有元组类型或者集合类型,称这个削剪的溯源树为P_{II}.

考虑P_{II}中至少被引用了两次的溯源元组.考虑最优削剪方法,删除这样的元组并不能降低整体存储空间,可以假设它们仍出现在最优削剪的溯源树上.因此,所有对这些元组的引用也必须出现在最优削剪的溯源树上.同样,对基本元组的引用也是不可避免的,称对基本元组的引用或被引用多次的元组的引用为不可避免引用,所有其它引用为可避免引用.

由于P_{II}中每个可避免的溯源元组引用了至少两个其它的元组(根据刚才所述,假设集合类型的数据项需要一个额外的单独引用的存储空间),这里可以形成一个引用的有向无环图,其中可避免的引用数量小于不可避免的引用数量.因为最优的削剪算法只能消除可避免引用,或者说,通过消除不可避免的引用并不能减少引用总数,那么剩余的引用数量是基于规则II削剪算法得到的引用数量的一半以上.

正如以上所述,仅使用规则II已经达到了最优溯源存储的近似2倍.在此基础上应用规则I,可以进一步减少存储成本,但不降低理论边界.

4 相关工作

目前,溯源信息的存储、管理和查询变得越来越重要,涌现了大量的研究工作.溯源信息追踪主要包括标记(annotation)和逆过程(inversion)两种方法.逆过程主要是应用逆向查询或者逆向函数,由结果数据溯源到其源数据.而标记的方法则是将一个数据的派生历史搜集起来作为元数据,与数据一起存放在数据库中.这里仅仅给出简要概述,详细的内容请参见文献[2,14].

4.1 溯源的语义

对于溯源的语义,文献[4,6]分别给出了不同的解释.文献[6]主要从数据库的角度来定义"溯源",认为"溯源"是数据源以及数据如何出现在数据库中的描述,并提供了对溯源语义的分类,将"Why-provenance"和"Where-provenance"严格区分开,其中"Why-provenance"描述了哪些源元组参与了结果元组的派生,例如,为什么元组出现在结果集里,"Where-provenance"仅仅提供了对源元组中属性值的引用,描述了结果元组的值从哪里拷贝而来.文献[4]将"溯源"定义成一种元数据,用于记录实验工作流的过程,是对实验过程的标记.文献[15]提出派生一个数据值所应用的转换过程也同样重要,并引入了基于数据值(而不是基于元组)的"Why-provenance"的变体叫"What-provenance".

文献[14]给出了"溯源"的一般性说明,认为"溯源"是一个数据的派生信息,它主要包括两个重要的特征:派生这些数据的源数据以及经历的转换过程.从数据库的角度而言,转换过程也就是查询.本文探讨了关系型溯源信息的存储以及元组出现在结果集中的原因,因此,主要关注的是"Why-provenance".

另外,很多研究工作[16-19]也将溯源的语义扩展到解释为什么查询的结果集中没有出现所希望的元组.

4.2 标 记

当前,大部分的溯源信息管理系统采用标记来进行溯源信息的追踪.文献[10,20]为元组中的每个属性添加一个额外的列,用于存储属性值的标记信息,其中文献[10]将SQL扩展成了包含标记信息的PSQL,允许用户使用PSQL说明如何跟踪和迁移标记信息,即执行查询时,源元组属性上的标记信息会自动地迁移到结果元组的相关属性上.PERM[21]将标记的迁移作为常规的SQL查询的一部分,通过在关系表中存储额外的溯源属性以及查询重写来实现溯源信息的跟踪.文献[20]则更深入地探讨了关系代数操作的各种标记信息的迁移规则.具体而言,对于选择和投影操作,结果元组上属性值的标记信息也就是源元组上对应的属性值;对于连接操作,结果元组上属性值的标记信息可能来自于多个源元组

上对应的属性值.对于聚集操作,由于聚集操作的结果并非拷贝而来,而是通过计算得到,因此,结果元组上属性值的标记信息来自所有涉及计算它的源元组上的属性值.然而,以上这种存储标记信息的方法显然需要非常大的存储空间,通常要超出数据本身的存储大小,并且这种方法仅仅能处理属性级别上的标记信息,而不能处理元组或关系级别上的标记信息.

在科学应用中,GridDB[12]也采用了类似的溯源信息跟踪方法,它通过存储每一个处理步骤的标记信息,并使用递归查询来检索源数据.CHIMERA系统[22]以推导图的形式跟踪整个数据集的溯源信息,它使用虚拟数据语言构造工作流,在执行过程中,工作流为每个计算程序自动创建触发对象,触发对象连接输入数据和输出数据,它们一起构成了表示数据派生过程的标记模式,因此,必要时可用于重新生成这些数据集.另外,一个通用的用于跟踪和查询溯源信息的半环模型在文献[23-24]中被提出.

文献[6]以树形结构(如 XML)存储溯源信息,并使用根节点到特定数值节点的路径作为溯源信息.然而,这个方法并没有探讨溯源信息查询处理的高效性和存储代价.和本文的工作类似,文献[7]提出了高效的溯源信息存储方法,但目标是要存储科学应用产生的复杂数据的溯源信息.由于这些数据具有相似或者相同的结构信息,因此结构继承、可选节点分解等优化技术可用于最小化溯源信息的存储.这些优化技术虽然并不适用于数据库查询,但其节点和参数分解技术具有与本文所提出的溯源树存储模型类似的目标.

文献[25]探讨了专业数据库的溯源信息管理,由于专业数据库的内容通常需要专业科学家从不同的数据源中手动拷贝,为了追踪构建专业数据库的用户行为(溯源信息),该文提出了自动追踪数据源到目标数据库的方法,将用户行为,也即目标数据的拷贝路径记录在一个溯源表中,并采用多种存储优化技术对此溯源表进行优化.显然,该方法跟踪的溯源类型与本文论述的溯源类型非常不同,因此这个优化技术并不适用于解决本文提出的这个问题.

4.3　逆过程

逆过程(包括逆查询和逆函数)从存储角度看,特别对大量细粒度的数据似乎是更佳的选择.文献[7]中首先探讨了通过逆函数计算溯源信息,而不是将溯源信息预先存储起来的思想.逆过程方法虽然减少了细粒度溯源信息的存储代价,但其适用性被限制在具有高效的逆过程中.如果没有这样的逆过程,该方法等价于重新执行查询.文献[26]将与关系数据库查询相关的溯源信息的逆查询显示存储起来,用于实现高效的数据源查询.然而,这种方法仅仅当存在这样一个逆查询时才有用.

文献[9]研究了关系型 ASPJ 视图中逆查询用于溯源信息跟踪的方法,其基本思想是通过物化中间结果,确保了逆查询的存在,并利用结果元组中的属性值执行逆查询来跟踪数据源.之后,文献[3]对此进行了改进,将其扩展到更一般的转换操作中.以上的方法使用逆查询作为核心,而不是标记,与本文的工作有很明确的相似之处,物化中间结果类似于本文提出的溯源表.与本文方法不同的是,首先,这个方法用属性值而不是元组引用作为溯源信息;其次,这个方法的执行效率取决于结果元组中可用的属性值的多少以及逆查询的关系操作类型;最后,这个方法只有当需要结果元组的溯源信息时,才实时产生逆查询,当数据量很大时,其效率可能不高.这个方法和本文所提出的方法都只选定查询树的部分节点存储数据,只是节点的选择策略并不相同,因此,本文中所提到的优化技术可以被纳入文献[3,9]的方法中,反过来也一样.具体来说,如果需要标识一个元组的源元组的集合,而这个元组是对一些数据值使用聚集方法得到的一个聚集的结果.如果这些源元组是基本关系中的元组,采用以上方法可能比维持元组引用集合更高效.探讨以上两种方法的混合策略是今后的主要研究工作.

4.4　其它研究

一些研究工作将数据值的溯源信息和数据的不确定性结合起来.TRIO 系统[11]不仅跟踪数据的溯源信息,也表达数据的不确定性.其中,溯源信息在查询(或程序运行)时被计算出来,然后存储在一个单独的溯源信息关系中,对每个数据元组,存储一个溯源元组.关于结合溯源信息和不确定性的进一步的研究[5]是基于源数据的概率值和结果元组的溯源信息计算出结果元组正确的概率值.以上方法的难点是设计一个包括数据精确性和溯源信息的简单数据模型以及一个扩展的 SQL 查询语句处理数据精确性和溯源信息的管理和查询.

另外,有些研究工作也使用标记信息来判断数据的可信度.文献[27]使用标记信息完成不同数据库间可靠数据的迁移,也即源数据库的数据根据映射关系迁移到目标数据库时,需要通过标记信息判断目标数据库是否信任该数据,不信任的数据不能进行迁移.同样,在文献[28]中,标记信息用于跟踪用户关于数据的信任度.文献[29]也提出一个模型来追踪用户标记信息的可信度.

5 实验评估

本文实验的目标主要是讨论第3节中提到的不同优化模式下的溯源信息存储需求,它们分别是最优的削剪、使用规则I的削剪和使用规则II的削剪、不削剪、完全削剪.最后,实验评估了溯源信息跟踪和优化方法的效率.

实验建立. 本文使用Java建立了一个关系查询引擎,它实现了主码和外码属性上的Hash连接以及基于B树索引的连接,其中,溯源元组和溯源表都是作为Java中的一个对象存储.因为本文主要评估溯源信息的存储代价,因此目前关系查询引擎的实现还未考虑查询优化.该查询引擎支持执行所需的SQL风格的查询以及在查询执行过程中溯源树的建立,并且已实现了第3节中提出的所有削剪方法.实验运行在2.0GHz双核2GB RAM内存的Windows 7机器上.查询引擎返回的查询结果的正确性已通过MySQL 5.1核实.

数据集和查询集. 本文选择了两个数据库:Wildfinder数据库(www.worldwildlife.org)和医疗数据集(www.medicare.gov).实验采用了Wildfinder数据库的6张表,分别是动物名称表*cn*、动物保护状态表*cs*、动物生态区物种表*es*、动物生态区表*e*、领域表*rc*、濒危物种表*rs*;以及医疗数据集中的2张表,分别是医院名称表*h*、医院死亡率表*hm*.详细的数据集见表3.

表3 数据集

表模式缩写	表模式	元组个数
cn(*cn_cn*, *cn_si*)	*common_names*(*cn_common_name*, *cn_species_id*)	1164
cs(*cs_cs*, *cs_csd*)	*conservation_sts*(*cs_conservation_status*, *cs_conservation_status_desc*)	3
es(*es_ec*,*es_si*)	*ecoregion_species*(*es_ecoregion_code*, *es_species_id*)	255
e(*e_ec*, *e_en*, *e_cs*, *e_rc*)	*ecoregions*(*e_ecoregion_code*, *e_ecoregion_name*, *e_conservation_status*, *e_realm_code*)	83
rc(*rc_rc*,*rc_r*)	*realm_code*(*rc_realm_code*, *rc_realm*)	8
rs(*rs_rlc*, *rs_wsi*)	*redlist_species*(*rs_red_list_category*, *rs_wwf_species_id*)	1289
h(*h_pn*, *h_hn*, *h_x*)	*hosp*(*h_prov_number*, *h_hospital_name*, *h_xstate*)	4546
hm(*hm_pn*,*hm_hn*,*hm_x*, *hm_mr*,*hm_np*)	*hosp_mortalit_xwlk*(*hm_prov_number*,*hm_hospital_name*, *hm_xcondition*, *hm_mortality_rate*, *hm_nr_patients*)	8191

从表3中可以看出所使用的数据集不太大,这是因为实验主要是测试本文所提出的方法的正确性以及不同优化模式下的溯源信息存储代价.

查询和查询计划是根据实验目的设计的,其目标是获得"实际"的查询和避免"不好"的查询计划.查询的设计主要考虑连接操作和聚集操作,因为这两个操作是影响溯源信息存储代价的主要因素.同时,查询计划也只针对树型结构设计,以便于高效地使用算法1计算出最佳的解决方案,并将此作为实验中比较的基准.系统原型的源代码、实验中所使用的数据集和查询集在文献[30]中可以找到.

5.1 溯源信息的存储代价

当评估优化方法的成效时,不直接比较本文的方法与现有的方法对于溯源信息存储的研究结果,其原因如下,首先,有些方法并不是专门为关系查询设计的,如果将查询简单当成工作流的查询会产生巨大的开销,也会使得比较不公平;其次,有些方法也探讨了关系查询的溯源信息的追踪,它们主要使用"标记"或"逆过程"的方法."标记"方法的侧重点并非讨论溯源信息的存储,而是探讨溯源信息的使用和管理,因此,在存储上是将溯源信息作为数据本身的一部分来存储,根据4.2节的分析,这个方法的弊端显而易见.而"逆过程"方法虽然其思想与本文有相似之处,但其侧重点也是探讨逆过程的产生,通过逆过程计算出溯源信息,而非溯源信息的存储.

下面将详细陈述6个有代表性的查询的结果,wwf_1,med_1,med_2为复杂的查询,而wwf_2,wwf_3,wwf_4只是连接查询.图6显示了各个查询对应的查询树,为了简单表达查询树,部分投影操作的投影属性以及连接操作的连接属性没有在查询树中具体描述.详细的查询集信息可以在文献[30]中找到.其中wwf_1主要是查询大洋洲濒危动物的名字、个数、它所属的生态区名字以及保护状态,涉及多达8个投影、2个选择、7个连接和1个聚集操作;med_2主要是查询病人数不小于100且平均死亡率不小于150的医院名称;med_2主要是查询因心脏病发作引发死亡率大于平均死亡率的医院名称.wwf_2,wwf_3,wwf_4只是分别连接Wildfinder数据库中的3张表、5张表和6张表.

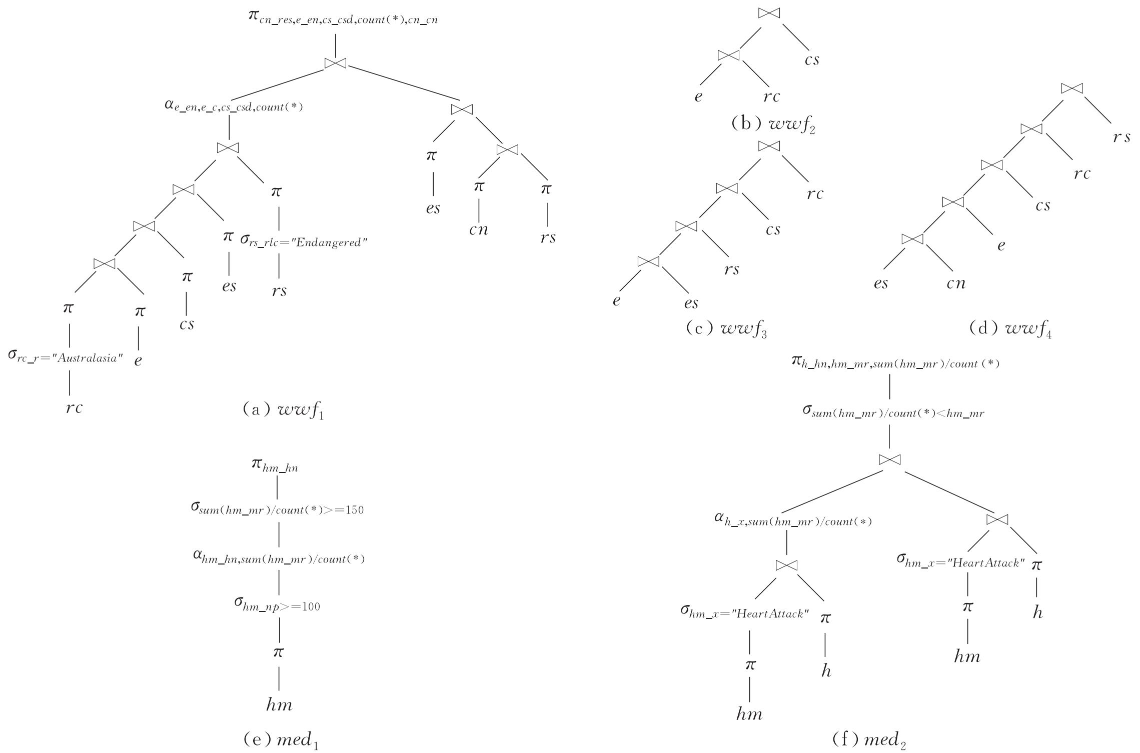

图 6 6 个样例查询的查询树

表 4 记录了 6 个查询的溯源树被初始构建时，删除冗余的元组后(不削剪)、完全削剪、使用规则 I 和/或规则 II 削剪及最优削剪后溯源树的大小. 溯源树的存储需求可以通过溯源树中的所有溯源表中的元组引用的数量来衡量. 本文所提出的这种方法的存储空间需求的上下界值可以通过不削剪、完全削剪方法来确定.

表 4 溯源树的存储大小

削剪规则	wwf_1	wwf_2	wwf_3	wwf_4	med_1	med_2
初始的树	12091	332	2032	2490	12217	65124
不削剪	4170	332	2032	2490	393	40827
完全削剪	8162	249	1270	1494	109	477530
规则 I	2068	249	1270	1494	109	13609
规则 II	3028	332	2032	2490	175	21353
规则 I&II	1950	249	1270	1494	109	13609
最优	1875	166	508	498	109	13367

从以上的表中我们可以观察到：(1) 基于规则 I 及 II 的削剪方法得到的结果往往是接近于最优解的，并且基于规则 I 及 II 的削剪明显优于不削剪和完全削剪. (2) 即使单独使用规则 *I* 的效果并没有进行理论的分析和保证，但在实际中，它往往取得比单独应用规则 II 更好的效果. (3) 在许多情况下，基于规则 I 和 II 的解决方案确实能取得最优. 这主要归因于这个事实：出现在实际和本文的查询集合中的大多数连接是键值连接. (4) 虽然不削剪会带来明显的开销且难于取得最优解，但开销也不会到达一个极端值，通常是最优解的 2～4 倍(存储代价因子为 2～4). 出现这个稳定性的理由是代价因子的大小被查询树的高度界定. (5) 完全削剪有时可以取得最优解(简单的查询更容易产生)，但也可能会导致极高的存储成本. 这种极端的情况通常涉及一个聚集操作后面出现一个连接操作.

除此以外，本文还做了大量的查询测试，完整的查询集在文献[30]中可以找到. 表 5 给出了整个查询集的测试结果，因为不同的查询对存储溯源信息的空间需求相差很大，因此表 5 总结了相对于最优解的不同方法的存储空间，同时给出了平均和最大(最坏情况)的结果.

表 5 溯源信息的存储代价率

方法	平均代价率/%	最大代价率/%
最优削剪	100	100
规则 I & II 削剪	104	112
不削剪	376	539
完全削剪	345	3572

5.2 算法的效率

本文还需要测试方法的时间效率. 前面已经证明了方法的时间复杂度的理论界限：所有算法是所访问元组数目的线性或近线性时间算法，是查询树的大小，即节点数量的低阶多项式时间算法. 对于以上每一个样例查询，可以将时间划分为 SQL 查询执行时间、初始溯源树构建时间以及各种削剪策略的处理时间. 这 6 个样例查询的时间如图 7 所示，时间是 log 规模的.

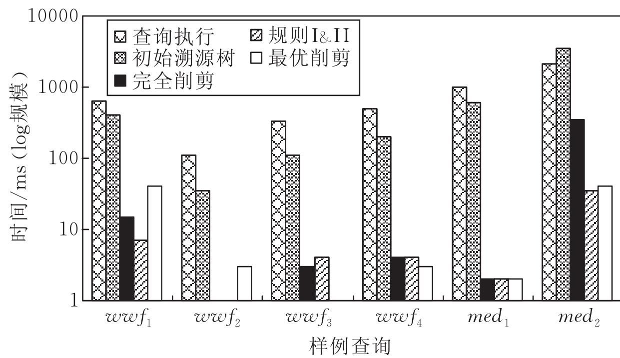

图 7 样例查询的效率

从图 7 中我们可以看到:(1) 初始溯源树的构建需要与查询执行大致相同的时间. 这是可以预料的,因为在初始的溯源树中的溯源元组的数量实际上是中间结果和最终结果中数据元组的总数. (2) 所有的削剪方法比查询执行花费的时间更少,虽然这主要依赖于查询,但通常查询执行时间是削剪方法所用时间的一个或两个数量级. 其主要原因是初始溯源树中冗余元组的删除使得时间大大的缩短. (3) 最优削剪算法明显比基于规则 I&II 的削剪算法慢,但从来不会超过一个量级. 这是因为本文提出的动态编程方法的多项式时间的性质决定的,如引理 1 所示.

整个查询集合[30]的结果在表 6 中显示,是以查询执行的代价作为基准,仍然用平均和最大(最坏情况)的结果来评估. 结果可以解释为由跟踪或优化溯源信息而间接导致的计算负载. 本文所提出的优化溯源信息存储的方法是正交于查询执行的,可以无缝地整合到现有的数据库引擎中.

表 6 计算时间效率

算法	平均效率/%	最大效率/%
查询执行	100.0	100.0
初始溯源树	94.0	245.0
规则 I&II 的削减	1.4	3.1
最优削剪	5.2	7.3

整个实验可以得到以下的结论. 对于基于削剪的优化,两个比较突出的方法是基于规则 I&II 的削剪和最优削剪. 由于计算效率通常不是一个问题,当查询是树型结构时,最优削剪方法的效果最好,而基于规则 I&II 的削剪方法能取得与最优结果最接近的结果.

6 结 论

本文介绍了一种关系数据库中溯源信息存储的新方法,即通过一个与查询结构匹配的初始溯源树来表达和存储溯源信息. 考虑到溯源信息存储代价,本文提出了基于规则 I&II 的削剪方法,并证明了该方法能达到最优溯源存储的近似 2 倍,因此能够普遍适用于支持溯源信息的应用. 此外,本文也提出了一个多项式时间的最优削剪算法,该算法能为树状结构的查询计算出一个最优解. 实验结果表明本文提出的方法可以显著减少存储需求,同时溯源信息的跟踪和优化的计算开销也是合理的.

参 考 文 献

[1] Zhou X F. Data quality: A core issue in modern databases and information systems research. China Computer Federation Communication, 2009, 5(2): 49-51(in Chinese)
(周晓方. 数据质量:现代数据库与信息系统研究的一个核心问题. 中国计算机协会通讯, 2009, 5(2): 49-51)

[2] Bose R, Frew J. Lineage retrieval for scientific data processing: A survey. ACM Computing Surveys, 2005, 37(1): 1-28

[3] Cui Y, Widom J. Lineage tracing for general data warehouse transformations. The VLDB Journal, 2003, 12(1): 41-58

[4] Greenwood M, Goble C, Stevens R et al. Provenance of e-science experiments-experience from bioinformatics//Proceedings of the AHM. Nottingham, UK, 2003: 223-226

[5] Benjelloun O, Sarma A D, Halevy A Y, Widom J. Uldbs: Databases with uncertainty and lineage//Proceedings of the VLDB. Korea, Seoul, 2006: 953-964

[6] Buneman P, Khanna S, Tan W C. Why and where: A characterization of data provenance//Proceedings of the ICDT. London, UK, 2001: 316-330

[7] Chapman A, Jagadish H V, Ramanan P. Efficient provenance storage//Proceedings of the SIGMOD Conference. Vancouver, BC, Canada, 2008: 993-1006

[8] Woodruff A, Stonebraker M. Supporting fine-grained data lineage in a database visualization environment//Proceedings of the ICDE. Birmingham, UK, 1997: 91-102

[9] Cui Y, Widom J. Practical lineage tracing in data warehouses//Proceedings of the ICDE. San Diego, CA, USA, 2000: 367-378

[10] Bhagwat D, Chiticariu L, Tan W C, Vijayvargiya G. An annotation management system for relational databases//Proceedings of the VLDB. Toronto, Canada, 2004: 900-911

[11] Widom J. Trio: A system for integrated management of data, accuracy, and lineage//Proceedings of the CIDR. Asilomar, CA, USA, 2005: 262-276

[12] Liu D T, Franklin M J. The design of GridDB: A data-centric overlay for the scientific grid//Proceedings of the VLDB. Toronto, Canada, 2004: 600-611

[13] Church A, Rosser J B. Some properties of conversion. Transactions of the American Mathematical Society, 1936, 39(3): 472-482

[14] Simmhan Y L, Plale B, Gannon D. A survey of data provenance in e-science. ACM SIGMOD Record, 2005, 34(3): 31-36

[15] Velegrakis Y, Miller R J, Mylopoulos J. Representing and querying data transformations//Proceedings of the ICDE. Tokyo, Japan, 2005: 81-92

[16] Huang J, Chen T, Doan A, Naughton J F. On the provenance of non-answers to queries over extracted data. Proceedings of the PVLDB, 2008, 1(1): 736-747

[17] Chapman A, Jagadish H V. Why not? //Proceedings of the SIGMOD. Providence, Rhode Island, USA, 2009: 523-534

［18］ Tran Q T，Chan C-Y. How to ConQueR why-not questions//Proceedings of the SIGMOD. Indiana，USA，2010：15-26
［19］ Herschel M，Hernndez M A，Tan W C. Artemis：A system for analyzing missing answers. Proceedings of the PVLDB，2009，2(2)：1550-1553
［20］ Cong G，Fan W，Geerts F. Annotation propagation revisited for key preserving views//Proceedings of the CIKM. Arlington，Virginia，USA，2006：632-641
［21］ Glavic B，Alonso G. Perm：Processing provenance and data on the same data model through query rewriting//Proceedings of the ICDE. Shanghai，China，2009：174-185
［22］ Foster I T，Vockler J-S，Wilde M，Zhao Y. Chimera：Avirtual data system for representing，querying，and automating data derivation//Proceedings of the SSDBM. Edinburgh，Scotland，UK，2002：37-46
［23］ Green T J，Karvounarakis G，Tannen V. Provenance semirings//Proceedings of the PODS. Beijing，China，2007：31-40
［24］ Karvounarakis G，Ives Z G，Tannen V. Querying data provenance//Proceedings of the SIGMOD Conference. Indianapolis，Indiana，USA，2010：951-962
［25］ Buneman P，Chapman A，Cheney J. Provenance management in curated databases//Proceedings of the SIGMOD Conference. Chicago，Illinois，USA，2006：539-550
［26］ Srivastava D，Velegrakis Y. Intensional associations between data and metadata//Proceedings of the SIGMOD Conference. Beijing，China，2007：401-412
［27］ Green T J，Karvounarakis G，Ives Z G，Tannen V. Update exchange with mappings and provenance//Proceedings of the VLDB. Vienna，Austria，2007：675-686
［28］ Gatterbauer W，Balazinska M，Khoussainova N，Suciu D. Believe it or not：Adding belief annotations to databases. Proceedings of the PVLDB，2009，2(1)：1-12
［29］ Ni O，Bertino E. Credibility-enhanced curated database：Improving the value of curated databases//Proceedings of the ICDE. Long Beach，California，USA，2010：784-795
［30］ Tested query set. http://www. comp. nus. edu. sg/～baozhife/provenancetree/

WANG Li-Wei，born in 1980，Ph.D.，lecturer. Her main research interests mainly include data provenance，record linkage，and scientific workflow management.

BAO Zhi-Feng，born in 1983，Ph. D.，research fellow. His research interests spread over keyword search over databases，social network modeling and analysis，and provenance data management.

KOEHLER Henning，born in 1976，Ph. D.. His main research interests include database dependency theory，normalization，data integration，data provenance，sampling and multi-target optimization.

ZHOU Xiao-Fang，born in 1963，Ph. D.，professor，adjunct professor. His research interests include spatial and multimedia databases，data quality，high performance query processing，Web information systems and bioinformatics.

SADIQ Shazia，Ph. D.，associate professor. Her main research interests are business process management，governance，risk and compliance，data quality management，workflow systems，and service oriented computing.

Background

This work is supported by Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 200804861067) and National Grant of Australian Research Council (No. LP0882957).

With the fast development of World Wide Web，the quantity of data explodes rapidly while the quality of data also decreases sharply；the data quality issue has become one of the major factors in deciding the success of a web information system. Some hot topics in data quality area include record lineage，data provenance，and data uncertainty. In this paper，we mainly focus on problems brought by data provenance.

Modern data management usually need to deal with data from different sources with different quality，therefore，supporting data provenance and allowing users to know where data comes from and how it was derived have become a critical research topic. The solutions of tracking data provenance in the literature usually involve annotations that comprise of the derivation history of a data product and inversion that generates a “reverse” query to find the origins supplied to derive a data product. However，storing fine-grained annotations can be expensive as the complete annotations for the data may outsize the storage space required for the data itself. Inversion seems to be more optimal from a storage perspective since an inverse function or query identifies the provenance for an entire class of data. However，it computes provenance data only when requested，rather than storing it. A drawback is that without good inverse functions this can be expensive，and it may require intermediate query results to be stored.

In order to make up for the shortcomings of the above methods，this paper proposed a new method for generating and storing tuple-based provenance data via database queries，using provenance trees which match the query structure to avoid redundant storage of information about the derivation process. Then this paper presented two rule-based reduction mechanisms to reduce the storage space. Furthermore，this paper proposed a polynomial time algorithm which computes an optimal solution for tree-structured queries by choosing only some of query tree nodes to store provenance information. Experiments showed the approach reduces storage requirements significantly，while the computational overhead for provenance tracking and optimization is reasonably small. This method is a new idea for the data tracing study and has a wide range of applications.