一种支持对象代理数据库高效查询处理的索引结构

黄泽谦[1)] 彭智勇[2)] 李 越[2)] 彭煜玮[2)]

[1)](武汉大学软件工程国家重点实验室 武汉 430072)

[2)](武汉大学计算机学院 武汉 430072)

摘 要 文中为对象代理数据库提出了一种新的索引结构——路径导航索引(Path Navigation Index,PNI),能够克服路径表达式计算开销大的缺点,使对象代理数据库跨类查询与代理对象查询具备高效的查询性能. PNI 索引建立在代理层次的路径实例之上,包括 Path-Instance-Table,Identity-Index 和 Attribute-Index 3 个组成部分. Path-Instance-Table能够物化存储路径实例,避免查询处理过程中冗余的对象导航遍历. Identity-Index 与 Attribute-Index 用于对路径实例进行关联检索,能够避免对象导航过程中的条件判断. 通过实验分析了影响路径表达式计算的不同因素,实验结果表明,利用 PNI 索引计算路径表达式的方法在多数情况下性能要优于现有计算方法,尤其适用于带谓词的路径表达式计算.

关键词 对象代理数据库;路径导航索引;路径表达式;查询处理

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2010.01446

An Index Structure for Efficient Query Processing in Object Deputy Database

HUANG Ze-Qian[1)] PENG Zhi-Yong[2)] LI Yue[2)], PENG Yu-Wei[2)]

[1)](*State Key Laboratory of Software Engineering*, *Wuhan University*, *Wuhan* 430072)

[2)](*Computer School*, *Wuhan University*, *Wuhan* 430072)

Abstract This paper proposes an index structure: Path Navigation Index (PNI), which can reduce the cost of evaluating path expression, leading to efficient query processing of cross-class query and deputy object query in object deputy database. Path Navigation Index consists of Path-Instance-Table, Identity-Index and Attribute-Index. Path instances are materialized in Path-Instance-Table, avoiding redundant object traversal in query processing. Identity-Index and Attribute-Index facilitate associative search of path instances, avoiding predicate evaluation during object traversal. The experiments are used to analyze the influential factors of path expression evaluation, and the experimental results demonstrate that evaluation of path expression with this index outperforms the other methods in most cases, especially for the path expressions with predicate conditions.

Keywords object deputy database; path navigation index; path expression; query processing

1 引 言

面向对象(OO)数据模型解决了传统关系模型难以建模复杂数据的问题,满足了诸多高级数据库应用的需求,如计算机辅助设计(CAD)、地理信息系统(GIS)、生物数据管理等. OO 模型虽然可以直接建模复杂对象,提供丰富的语义,但是对象的封装

收稿日期:2010-06-11. 本课题得到国家"九七三"重点基础研究发展规划项目基金(2007CB310806)、国家自然科学基金重大研究计划项目(90718027)、湖北省自然科学基金重点计划项目(2008CDA007)和中央高校基本科研业务费专项资金(6082011)资助.**黄泽谦**,男,1983 年生,博士研究生,主要研究方向为数据库和科学工作流. E-mail: zeqian@mail.whu.edu.cn. **彭智勇**,男,1963 年生,教授,博士生导师,主要研究领域为先进数据库管理和 Web 信息管理. **李 越**,男,1987 年生,硕士研究生,主要研究方向为数据库和生物信息管理. **彭煜玮**,男,1980 年生,博士,讲师,主要研究方向为数据库和地理信息系统.

性使得对象不具有关系模型中关系的柔软性，因为对象难以分割和重组；此外，OO 模型缺乏对现实世界实体多角色特性与动态特性的建模能力. 对象代理(Object Deputy，OD)模型[1-2]是在传统 OO 模型基础上提出的一种新的数据模型，它利用代理对象(deputy object)的概念增强了 OO 模型的柔软性和建模能力. 近年来，OD 模型被广泛应用于数据仓库[3]、工作流视图[4]、科学工作流[5]、生物数据管理[6]以及地理信息系统[7]等领域. OD 模型与数据库相结合产生了对象代理数据库(Object Deputy Database，ODDB). ODDB 继承了诸多面向对象数据库(OODB)的优点，如直接建模复杂对象、把类组织成代理层次等. ODDB 允许定义灵活的对象视图、支持对象动态分类，并且提供特殊的跨类查询等新机制. ODDB 使用代理对象增强了对象的柔软性，允许对已有对象进行分解、组合和扩充，比 OODB 具有更加柔软的建模能力. 实践表明，ODDB 管理复杂数据的能力强，特别是在生物数据库、空间数据库、多媒体数据库等应用方面，比 OODB 更高效.

文献[8]为 ODDB 提出了一种声明式查询语言——对象代理查询语言. 该语言使用方便，具有许多新特性，如代理对象查询、跨类查询等. 众所周知，查询处理的效率严重影响着数据库系统的性能，代理对象查询与跨类查询是 ODDB 两种重要的查询机制，减少这两类查询的开销，有助于提高数据库系统的整体性能. 在 ODDB 中，代理对象的模式定义为代理类，其中继承定义的属性称为虚属性，不占有实际的物理存储；扩展定义的属性称为实属性，它们占有真正的物理存储. 代理对象查询的特殊之处在于虚属性计算. 每个虚属性定义包含一个读切换表达式定义，声明了如何由被继承的源对象属性计算得到该虚属性的值. 一个读切换表达式可以转换为一个表达式计算树，树的非叶子结点为计算操作符，树的叶子结点为数值常量或路径表达式，其中的路径表达式声明了从该虚属性到达一个源对象属性的路径. 虚属性查询处理的关键是路径表达式计算. 此外，对象代理查询语言使用路径表达式表达跨类查询. ODDB 中代理对象与源对象间存在双向指针链接. 跨类查询是指基于对象间的双向指针链接，从某个类(代理类)的对象(代理对象)出发，沿着类路径到达一个与其存在直接或间接代理关系的类(代理类)，并对该类上相关联的对象(代理对象)进行查询. 跨类查询的核心是路径表达式计算. 可以看到，路径表达式是对象代理查询语言的重要查询设施，提高路径表达式的计算效率是 ODDB 高效查询处理的关键.

路径表达式的概念并不是 ODDB 所特有，早在 OODB 查询语言[9-10]中就已经存在，用于表达嵌套对象的导航遍历. 目前，在 OODB 中针对路径表达式计算的基本方法有 3 种：正向指针跟踪算法、反向指针跟踪算法和显式连接算法. 正向指针跟踪算法利用对象引用指针，从路径起始类的对象出发，沿着嵌套关系路径进行对象遍历. 反向指针跟踪算法的思想与正向指针跟踪算法的思想类似，不过是从路径末结点类的对象出发进行对象遍历，如果 OODB 中被引用对象存在指向引用对象的反向指针，则对象遍历的执行过程与正向指针跟踪算法对象遍历的执行过程一样；如果 OODB 中被引用对象不存在反向指针指向引用对象，则反向指针跟踪算法实际上是一个隐式连接算法，需要将路径后一层对象的标识符与前一层对象的引用指针进行相等比较. 显式连接算法利用传统数据库的连接操作，将路径表达式转换成一个显式连接表达式进行计算. 文献[11]提出了一个代价模型用于评估不同方法的计算代价，并使用启发式规则选取由不同计算方法组成的高效组合计算策略. 相对于上述 3 种集中式的计算方法，文献[12]提出了路径表达式的并行算法，通过将一个路径表达式转换为一个级联式半连接表达式实现并行计算. ODDB 路径表达式与 OODB 路径表达式的区别在于：ODDB 路径表达式不是用于表达嵌套对象的导航遍历，而是用于表达代理层次中的对象导航遍历，两者的计算方法并不相同.

支持 OODB 嵌套对象查询、优化 OODB 路径表达式计算的技术主要是索引技术. 文献[13]提出在类的聚集层次中任意两个具有引用关系的类之间建立 Multiple Index，将被引用对象映射到引用对象. 文献[14]提出的 Join Index 与 Multiple Index 相类似，只不过在两个具有引用关系的类之间建立两个映射关系，一个是从被引用对象映射到引用对象，另一个是从引用对象映射到被引用对象. 为支持嵌套谓词计算，文献[15]提出了 Nested Index，可以由一个嵌套属性值直接映射到嵌套层次的根对象本身. Path-Index[16]在索引结构上比 Nested Index 复杂，索引记录不仅保存嵌套层次的根对象，而且还将嵌套路径上的各个对象都保存起来. Path Dictionary[17]提出了一种基于路径词典的索引机制，用于支持对象遍历和关联检索. 文献[18]提出的 Access Support Relation 采用独立的结构物化对象间的引用，实际上是一种一般化的 Join Index. 上述各种应用于 OODB 路径表达式的索引技术并不完全适用于

ODDB 路径表达式，而且到目前为止，还没有针对 ODDB 路径表达式计算的索引技术研究.

本文针对 ODDB 路径表达式计算问题，提出了一种新的索引结构——路径导航索引(Path Navigation Index，PNI)，并基于 PNI 设计实现了计算路径表达式的方法. PNI 索引建立在任意长度的路径上，核心思想是物化路径实例，允许在路径谓词的谓词属性上建立属性索引，以根据谓词属性值快速定位路径实例. 基于 PNI 的路径表达式计算方法，有效避免了路径表达式计算过程中的对象遍历操作，能快速过滤不满足谓词的路径实例，减少路径谓词计算的开销. 本文通过实验验证了 PNI 索引能有效支持路径表达式计算，所提出的基于 PNI 的计算方法能实现高效的路径表达式计算，从而达到在 ODDB 中高效处理涉及路径表达式的代理对象查询和跨类查询的目的.

本文第 2 节对对象代理数据库进行介绍；第 3 节介绍路径导航索引 PNI，并提出使用 PNI 计算路径表达式的方法；第 4 节通过实验验证所提出的基于 PNI 计算路径表达式方法的有效性，并将该方法与其它几种方法进行分析比较；第 5 节对本文工作进行总结.

2 对象代理数据库

2.1 基本概念

对象代理数据库的核心概念包含对象、代理对象、类和代理类等，本节首先对这些概念进行简单描述，其详细定义参见文献[1]. 另外，为方便文章的讨论，本节还将给出路径表达式等相关概念的定义.

对象用于表示现实世界的实体，包括属性集和方法集. 对象的属性值表示实体的状态，而对象的方法则用于表示实体的行为. 代理对象基于对象或其它代理对象定义，同样具有自身的属性集和方法集. 一个(代理)对象可以有多个代理对象，前者称为后者的源对象；多个源对象可共享一个代理对象. 代理对象与对象间的代理关系有别于传统的对象继承关系，即代理对象可以选择性继承源对象的部分或全部属性、方法，也可以根据需要增加扩展属性和扩展方法的定义. 每个对象和代理对象都具有唯一的标识符.

具有相同属性和方法的对象用类来定义其模式. 作为对象模式，类包括类名、外延和类型. 外延是属于该类的对象集，类型是对象的属性和方法的定义. 代理类定义代理对象的模式，同样包括代理类名、外延和类型. 代理类可由源对象所属类导出，后者称为前者的源类. 代理类定义中，继承的属性称为虚属性，并不占有实际物理存储. 每个虚属性定义有两个基本切换操作，使得对虚属性的读操作将切换至对继承属性的读，对虚属性的写操作也将切换到对继承属性进行更新. 代理类定义中扩展定义的属性与类定义中的属性一起统称为实属性. 实属性数据占有实际物理存储空间，支持直接的数据读写操作. 与属性相类似，类定义中的方法与代理类中扩展定义的方法支持直接方法调用，而代理类中继承定义的方法上定义有切换操作，对其调用会切换至对继承方法的调用.

ODDB 中可以定义 4 种类型的代理类，分别是 Select、Join、Group 和 Union. Select 代理类只从一个类(代理类)选择源对象以派生代理对象；Join 代理类在几个类(代理类)的连接结果中选择派生代理对象；Group 代理类对一个类(代理类)的对象(代理对象)进行分组，进而派生代理对象，一个代理对象代理一组源对象；Union 代理类从若干个源类选择 Select 代理派生的代理对象，再并在一个代理类中. 图 1 给出了一个音乐数据库的模式，它包含 5 个类：*Artist*(歌手)、*Music*(音乐)、*MTV*(音乐录影带)、*Lyrics*(歌词)和 *Picture*(专辑图片)；还定义有 5 个代理类：包括有 1 个 Select 代理类 *Personal*(个性化音乐)，3 个 Join 代理类 *Music_Lyr*、*Music_MTV* 与 *Album_Pic* 以及 1 个 Group 代理类 *Album*(专辑). 图中不带下划线标识的属性为实属性，而带下划线标识的属性为虚属性. 以代理类 *Album* 为例，歌手名 *artist* 和专辑名 *album* 是由 *Music_Lyr* 继承的虚属性，而专辑发布日期 *pubdate* 和专辑介绍 *intro* 则是扩展定义的实属性.

为便于后续讨论，下面给出 ODDB 中与路径表达式相关的概念定义.

定义 1(代理层次). 由类(代理类)以及类(代理类)之间的代理关系所形成的有向图称为代理层次. 代理层次中的结点表示类(代理类)，而有向边则表示任意两个结点之间的直接代理关系. 令 DH 为一个代理层次，$Class(DH)=\{C \mid C$ 是 DH 中的类或者代理类$\}$，表示代理层次中所有类与代理类的集合.

定义 2(路径和逆路径). 给定代理层次 DH，称 $P=C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_n\ (n\geq 2)$ 为代理层次 DH 的一条路径，当且仅当满足以下 2 个条件：(1) $C_i\in Class(DH)$，$1\leq i\leq n$；(2) C_i 与 C_{i+1} 之间存在直接代理关系，即 C_i 是 C_{i+1} 的代理类或者 C_{i+1} 是 C_i 的代理

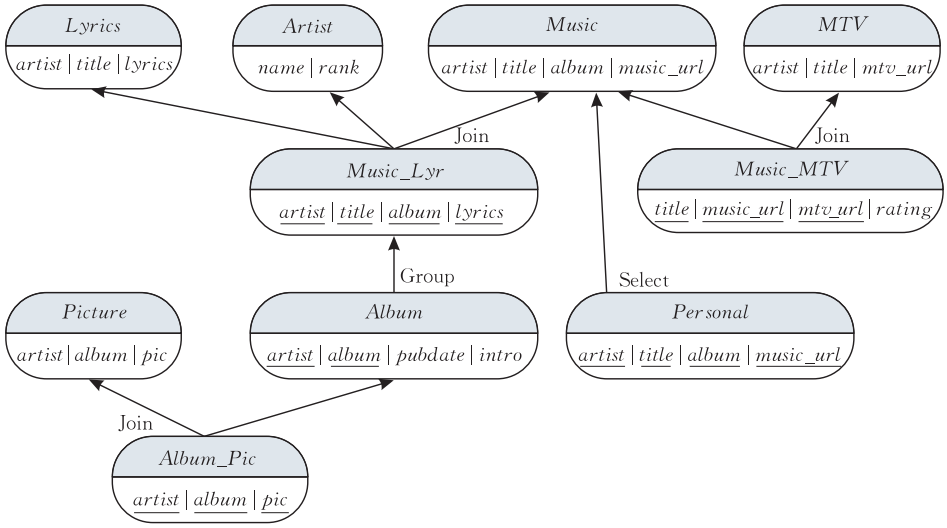


图 1　音乐数据库模式

类，$1\leq i<n$. $len(P)=n-1$ 表示路径 P 的长度. $class(P)=\{C_i \mid 1\leq i\leq n\}$ 表示路径 P 所包含的类与代理类的集合. 对于给定的路径 $P_1=C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_n$，称路径 $P_2=C_n\rightarrow C_{n-1}\rightarrow\cdots\rightarrow C_1$ 为路径 P_1 的逆路径，反之也成立.

定义 3(路径实例). 给定路径 $P=C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_n$，由 n 个对象(代理对象)组成的对象序列 $PI=o_1\rightarrow o_2\rightarrow\cdots\rightarrow o_n$ 称为 P 的一个实例，当且仅当：(1) o_i 是 C_i 的一个实例，$1\leq i\leq n$；(2) o_i 与 o_{i+1} 之间存在直接代理关系，即 o_i 是 o_{i+1} 的代理对象或者 o_{i+1} 是 o_i 的代理对象，$1\leq i<n$.

定义 4(路径表达式). 给定代理层次 DH，称 $PE=(C_1\{p_1\}\rightarrow C_2\{p_2\}\rightarrow\cdots\rightarrow C_n\{p_n\}).expr(n\geq 2)$ 是在路径 $P=C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_n$ 上定义的路径表达式，当且仅当：(1) P 是 DH 的一条路径；(2) p_i 是定义在 C_i 属性上的谓词(p_i 可以为空，$1\leq i\leq n$)；(3) $expr$ 是由 C_n 的属性与常量通过算术或逻辑运算符组成的合法表达式. 其中，$C_1\{p_1\}\rightarrow C_2\{p_2\}\rightarrow\cdots\rightarrow C_n\{p_n\}$ 称为 PE 的导航路径，$expr$ 称为 PE 的目标表达式.

图 1 的数据库模式是一个代理层次，$P=Music\rightarrow Music\_Lyr\rightarrow Album$ 是该代理层次的一条路径，$(Music\{artist='U2'\}\rightarrow Music\_Lyr\rightarrow Album).intro$ 是定义在路径 P 上的一个路径表达式.

2.2　对象代理查询语言

对象代理查询语言是对象代理数据库的查询语言，它在语法上与 SQL92/99 类似，是一种声明式查询语言，一个查询命令的基本格式与 SQL 查询命令的基本格式一样，由 SELECT 子句、FROM 子句和 WHERE 子句构成. 下面根据图 1 的数据库模式分别举例说明.

例 1.　针对 *Music_MTV* 查询："歌曲 foo 的音乐链接与音乐录影带链接"，该查询采用对象代理查询语言表示如下：

```
SELECT music_url, mtv_url FROM Music_MTV
WHERE title='foo';
```

这个例子涉及代理对象查询，因为 *Music_MTV* 是一个代理类，其所有实例都是代理对象，每个代理对象的属性都是虚属性且不存储这些属性的值. 在查询语言的表达上，查询语句可以用于查询类和代理类中的对象. 如果在一个查询语句的 SELECT 子句或 WHERE 子句中存在虚属性，查询处理需调用虚属性的读切换操作来计算它们的值. 例如在代理类 *Music_MTV* 的定义中，虚属性 *music_url* 由类 *Music* 的实属性 *music_url* 直接继承，该属性上定义的读切换操作由读切换表达式 $(Music\_MTV\rightarrow Music).music\_url$ 表示；同理，虚属性 *mtv_url* 所定义的读切换操作由 $(Music\_MTV\rightarrow MTV).mtv\_url$ 表示，虚属性 *title* 的读切换操作由 $(Music\_MTV\rightarrow Music).title$ 表示. 例 1 的查询最终会被转化为

```
SELECT (Music_MTV→Music).music_url,
       (Music_MTV→MTV).mtv_url
FROM Music_MTV
WHERE (Music_MTV→Music).title='foo';
```

该查询涉及 3 个路径表达式的计算.

除了查询虚属性时，由于查询处理模块自动将虚属性替换为其读切换表达式，导致在查询语句中隐式引入路径表达式外，对象代理查询语言允许在查询语句中显式使用路径表达式. 如果路径表达式出现在 SELECT 子句中，表示根据路径实例上的对象双向指针链，在一个代理层次范围内跨类查询相关联的对象(代理对象).

例 2. 针对 *Music* 查询:"歌曲 foo 的歌词",该查询可以表示如下:

SELECT (*Music*→*Music_Lyr*→*Lyrics*).*lyrics*
FROM *Music* WHERE title='foo';

或者

SELECT (*Music*{*title*='foo'}→
Music_Lyr→*Lyrics*).*lyrics*
FROM *Music*;

以上查询所依赖的路径是 *Music*→*Music_Lyr*→*Lyrics*,即从 *Music* 中的对象出发,沿着路径实例上的对象双向指针链,跨类查询 *Lyrics* 对象的歌词信息.以上两个不同的路径表达式表达同一查询目的.

路径表达式如果出现在 WHERE 子句中,表示检索对象(代理对象)时,需要依赖同一个代理层次中相关联的其它对象(代理对象).

例 3. 针对 *Music* 查询:"排名前十的歌手的所有专辑名称",该查询可以表示如下:

SELECT *album* FROM *Music*
WHERE (*Music*→*Music_Lyr*→*Artist*).*rank*<=10;

虽然查询的目标是 *Music* 中的 *album* 属性,但是却需要依赖 *Artist* 中的 *rank* 进行谓词判断.

此外,路径表达式也可以同时在 SELECT 子句与 WHERE 子句中存在.

2.3 路径表达式计算

路径表达式计算是 ODDB 查询处理的关键,因为涉及虚属性的代理对象查询以及跨类查询的核心都在于路径表达式计算.路径表达式的计算需依赖对象间的双向指针链接.对象间的双向指针链接是指每个代理对象记录有指向其源对象的指针,每个对象(代理对象)也存在指向其代理对象的指针,其实质是对象间的代理关系映射,方便由一个对象(代理对象)找到其源对象及代理对象.目前,ODDB 采用标识符(*OID*)映射表来记录对象间的代理关系映射,对象间的双向指针链接由系统自动维护,并保存在该标识符映射表中.标识符映射表的模式为一个二元关系:

[*SourceObject*:*OID*, *DeputyObject*:*OID*],

其中 *SourceObject* 为源对象的对象标识符,*DeputyObject* 为代理对象的对象标识符.

实现 ODDB 路径表达式计算的主要步骤是:首先根据路径表达式所依赖的路径,检索满足路径各层所定义谓词的路径实例,再由结果路径实例得到路径上最后一个类(代理类)的对象(代理对象),最后根据路径表达式的目标表达式进行投影计算.计算路径表达式的算法主要有两种:指针跟踪算法与显式连接算法.

在指针跟踪算法中,路径第 1 个类(代理类)C_1 的一个对象(代理对象)首先被存取,并检查是否满足该类(代理类)的谓词 p_1.对于匹配的对象(代理对象)O,通过对象间的双向指针链接检索可能的路径实例上的下一个对象,即利用标识符映射表检索路径下一层结点的实例,再判断是否满足相应的谓词.这种指针跟踪一直进行到发现已遍历的对象并不属于一个路径实例(即路径上某层不存在与上一层实例有代理关系的实例)、或在路径实例上有某个谓词为假、或路径实例上最后一个对象(代理对象)被碰到.如果路径实例上最后一个对象(代理对象)被找到,则通过投影计算得到目标表达式的值,从而完成一次计算流程.然后取 C_1 中下一个对象(代理对象)继续上述处理流程,直到 C_1 中所有对象(代理对象)被扫描处理完毕为止.指针跟踪算法是一种面向记录的算法.在根据双向指针链接进行对象遍历的过程中如果不需要进行谓词判断,则只需检索标识符映射表,否则需要存取对象实例进行谓词计算.如果谓词条件包含有虚属性,需通过虚属性的读切换表达式计算其值,这又涉及到嵌套计算其它路径表达式.

显式连接算法是把路径转换为一个显式连接表达式,通过该表达式计算得到满足所有谓词的路径实例,最后投影计算目标表达式.在转换后的显式连接表达式中,如果路径上不存在谓词,那么只需要将路径第一个和最后一个类(代理类)以及标识符映射表进行连接;如果路径某一层存在谓词,则需将该层所对应的类(代理类)也加入到连接表达式中.对于长度为 n 的路径,标识符映射表需自连接 n 次.相对于指针跟踪算法,显式连接算法是面向集合的算法.

3 路径导航索引 PNI

本节介绍一种支持路径表达式高效计算的索引结构——路径导航索引(Path Navigation Index,PNI).PNI 建立在代理层次的路径之上,主要由 3 部分组成:Path-Instance-Table、Identity-Index 和 Attribute-Index. Path-Instance-Table 用于存储路径实例,Identity-Index 与 Attribute-Index 建立在 Path-Instance-Table 之上,用于支持快速的路径实例检索.PNI 减少了路径表达式计算过程中对象遍历以及谓词判断的开销.

3.1 PNI 索引结构

Path-Instance-Table 是一个路径实例表,物化存储了给定路径的所有实例,表的每个元组对应一

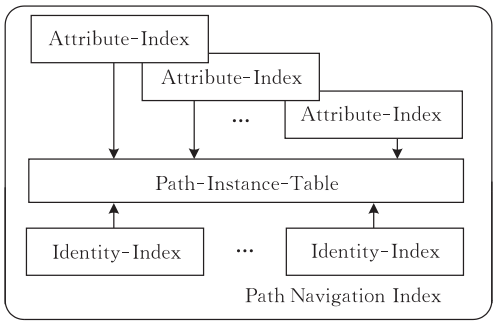


图 2　路径导航索引 PNI

个路径实例. 简单来说，ODDB 通过标识符映射表维护对象间的代理关系，在一个代理层次中，给定一个路径，利用对象间的代理关系可以确定其所有的路径实例. 对于频繁查询的路径，其路径实例可以存储在 Path-Instance-Table 中，避免每次计算路径表达式时都需要通过对象遍历来检索路径实例，减少数据库存取开销. Path-Instance-Table 不存储完整的对象实例，只保存组成路径实例的所有对象(代理对象)的 *OID*. 如图 3 所示，给定一条路径 $P=C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_n$，其对应的 Path-Instance-Table 的模式为$[S_1:OID,S_2:OID,\cdots,S_n:OID]$，即长度为 $n-1$ 的路径所对应的 Path-Instance-Table 一共有 n 列，每列 S_i的数据类型为 *OID*，其取值范围为 C_i所有实例的 *OID* 集合，$1\leq i\leq n$，Path-Instance-Table 的每一行对应 P 的一个路径实例.

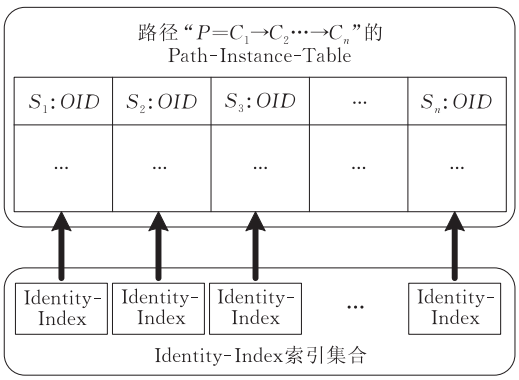
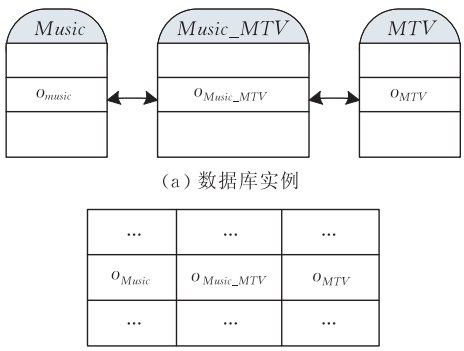


图 3　Path-Instance-Table 结构

Path-Instance-Table 在设计时主要考虑如下：(1) 互为逆路径的两条路径，只存储一个 Path-Instance-Table，以节省存储开销. 对于任意路径 P 及其逆路径 P'，将组成 P 的每一个路径实例的对象序列进行顺序置换，可以得到 P'的路径实例，反之亦然，因此可以通过将 P 的 Path-Instance-Table 进行列置换得到 P'的 Path-Instance-Table. (2) Path-Instance-Table 存储完整的路径实例. 在计算路径表达式时，需要找到路径实例上最后一个对象实例，并进行投影计算，因此 Path-Instance-Table 另一种可能的设计是只存储路径实例上的第一个和最后一个实例. 但是由于在路径的每一层都可能存在谓词，如果要判断路径实例是否满足谓词，还需要进行一次对象遍历以存取路径实例上相应的对象实例，只存储第一个和最后一个实例的 Path-Instance-Table 无法直接支持这样的操作. 出于此目的，Path-Instance-Table 存储完整的路径实例. (3) 支持对象的双向遍历. 因为对象间的指针链接是双向的，因此对象的遍历也可以是双向的. 根据(1)和(2)，显然可以有效地支持对象间的双向遍历. Path-Instance-Table 在实现时是通过一个表来实现的，对于表的每一行而言，其每一属性列所记录的 *OID* 按照列的顺序组成的一个对象序列，构成 Path-Instance-Table 所对应路径的一个路径实例，如图 4 所示，对于路径 $Music\rightarrow Music\_MTV\rightarrow MTV$，存在一个路径实例 $o_{Music}\rightarrow o_{Music\_MTV}\rightarrow o_{MTV}$，该路径在 Path-Instance-Table 中有相应的一条记录$[o_{Music},o_{Music_MTV},o_{MTV}]$.

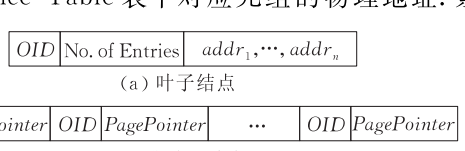


(a) 数据库实例

(b) 路径$Music\rightarrow Music_MTV\rightarrow MTV$所对应的Path-Instance-Table

图 4　Path-Instance-Table 示例

Path-Instance-Table 利用 *OID* 来记录路径实例，因此常常需要根据一个给定的 *OID* 来检索相关联的路径实例，为此，在 Path-Instance-Table 上建立 Identity-Index，以支持快速的路径实例检索. Identity-Index 具体通过在 Path-Instance-Table 的每一列上建立一个 B+ 树索引来实现，即 Identity-Index 是一个索引集合，每一个索引是 Path-Instance-Table 的一个列索引，以 *OID* 作为键值. Identity-Index 索引的叶子结点如图 5(a)所示，其中 $addr_1$与 $addr_n$是包含有键值 *OID* 的路径实例在 Path-Instance-Table 表中对应元组的物理地址. 索



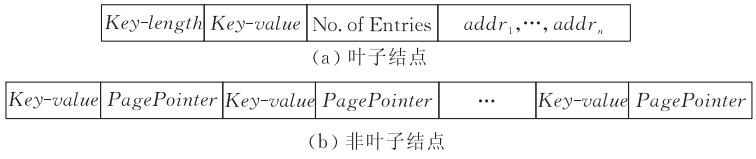
(a) 叶子结点

(b) 非叶子结点

图 5　Identity-Index 索引树结构

引的非叶子结点如图 5(b)所示,其中 *PagePointer* 是指向索引树中下一层结点的指针.

虽然 Path-Instance-Table 通过物化存储路径实例,支持快速的对象遍历,但 Path-Instance-Table 本身无法直接支持谓词判断.计算路径表达式时,需要根据路径上声明的谓词过滤不满足条件的路径实例.路径上每一层的谓词都是定义在该层对象的属性值上的,为计算路径上某一层的谓词,需要在对象遍历的过程中存取路径实例上该层的对象,从而利用对象的属性值进行条件计算.如果在过滤路径实例的过程中总是需要存取路径实例的中间对象,会增大路径表达式计算的开销.Attribute-Index 就是针对路径实例的条件过滤而设计的属性索引.不同于一般类上的属性索引将属性值直接映射到对象本身,Attribute-Index 是将路径上某个类(代理类)的对象(代理对象)的属性值映射到包含该对象(代理对象)的路径实例信息上.具体而言,Attribute-Index 索引建立在 Path-Instance-Table 之上,采用 B+ 树结构实现,索引的键值是 Path-Instance-Table 对应路径上某个类的属性值,Attribute-Index 索引将该类上对象的属性值映射到包含该对象的路径实例所对应 Path-Instance-Table 元组的地址.图 6(a)给出了 Attribute-Index 索引的叶子结点,其中 *Key-length* 是索引属性的长度,*Key-value* 为索引键值,$addr_1$ 与 $addr_n$ 是属性值等于索引键值的对象所对应 Path-Instance-Table 元组的地址.Attribute-Index索引的非叶子结点如图 6(b)所示,其中 *PagePointer* 是指向索引树中下一层结点的指针.



(a) 叶子结点

(b) 非叶子结点

图 6 Attribute-Index 索引树结构

给定一条路径 P,为其创建的 Path-Instance-Table 只有一个.而根据路径 P 定义的路径表达式可以有许多个,每个路径表达式在路径上每一层都可能声明各种谓词.根据实际路径表达式计算的需要,可以在 Path-Instance-Table 上建立不同的 Attribute-Index 索引,一个 Attribute-Index 索引可以被一条路径上不同的路径表达式计算时使用.

3.2 PNI 索引的使用

本小节介绍如何使用 PNI 索引进行路径表达式计算.在讨论之前,我们假设路径表达式所依赖的路径上建有 PNI 索引.由于路径表达式在导航路径的不同的层次上可能定义有谓词,为简化讨论,我们假设每个谓词只有一个谓词属性.在路径表达式的各个谓词属性上,可能建有 Attribute-Index 索引,也可能没有.下面我们基于图 1 给出的代理层次分情况进行讨论.

(1)导航路径上不带谓词的路径表达式计算

在这种情况下,路径表达式的导航路径不存在任何谓词.例如路径表达式(*Music*→*Music_Lyr*→*Lyrics*).*lyrics*,用于从 Music 类的对象出发,查询 *Lyrics* 类中相关联对象的 *lyrics* 属性的值,路径 *Music*→*Music_Lyr*→*Lyrics* 不含有任何谓词.令路径 *MML*=*Music*→*Music_Lyr*→*Lyrics* 上定义的 PNI 索引所包含的 Path-Instance-Table 表为 $PIT_{MML}[OID_{Music}, OID_{Music\_Lyr}, OID_{Lyrics}]$,则计算路径表达式(*Music*→ *Music_Lyr*→*Lyrics*).*lyrics* 的过程是:由 PIT_{MML} 投影得到 *MML* 路径实例中所有 *Lyrics* 对象的 *OID* 集合,进一步可以获取对应的 *Lyrics* 的对象集合,再对该集合中的每个 *Lyrics* 对象投影计算 *lyrics* 属性的值.

(2)导航路径上只有一个谓词的路径表达式计算

在这种情况下,计算路径表达式的过程需要过滤不满足条件的路径实例.如果在谓词属性上建有 Attribute-Index 索引,那么可以直接利用索引扫描获取满足谓词的路径实例;否则,需要获取路径实例上的中间对象以判断谓词.例如路径表达式(*Music*{*title*='foo'}→*Music_Lyr*→*Lyrics*).*lyrics*,其 PNI 索引的 Path-Instance-Table 与情况(1)中定义相同.假设在 *Music.title* 上建有 Attribute-Index 索引,那么计算该路径表达式时,首先根据 *Music.title* 上的 Attribute-Index 索引检索 PIT_{MML} 中所有满足谓词"*title*='foo'"的元组集合 S_1,再根据 *Lyrics* 对象的 *OID* 将 *Lyrics* 与 S_1 进行等值连接操作,得到所需 *Lyrics* 对象的集合 S_2,最后再对 S_2 中的每个对象投影计算 *lyrics* 属性的值.如果在 *Music.title* 上不存在 Attribute-Index 索引,则需要通过扫描 *Music* 类找到所有 *title* 属性值为"foo"的对象集合 S_0,再根据 *OID* 将 S_0 与 PIT_{MML} 进行等值连接操作,得到 PIT_{MML} 的元组集合 S_1,后续操作与上一种情况一样.

(3) 导航路径上有多个谓词的路径表达式计算

在这种情况下，计算路径表达式时需要多次进行谓词计算，以过滤不满足条件的路径实例. 对于路径表达式$PE=(C_1\{p_1\}\rightarrow C_2\{p_2\}\rightarrow\cdots\rightarrow C_n\{p_n\}).expr$，假设在路径$C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_n$上建有 PNI 索引，且 PNI 索引包含在部分谓词的谓词属性上创建的 Attribute-Index 索引，那么对于每个谓词属性上建有 Attribute-Index 索引的谓词，可以利用 Attribute-Index 索引扫描得到的一个满足该谓词的 Path-Instance-Table 结果集合，而对于每个不能利用 Attribute-Index 索引的谓词，则需要通过扫描谓词所在类的实例，再与 Path-Instance-Table 进行连接操作得到一个满足该谓词的 Path-Instance-Table 结果集合. 最后通过将这些 Path-Instance-Table 结果集合求交集，得到满足所有条件的路径实例. 例如路径表达式($Music\{title=$'foo'$\}\rightarrow Music\_Lyr\rightarrow Lyrics\{lyrics=$'loo'$\}).artist$，其 PNI 索引的 Path-Instance-Table 与(1)中定义相同，假设在 *Music.title* 上建有 Attribute-Index 索引，而在 *Lyrics.lyrics* 上并没有 Attribute-Index 索引，那么计算该路径表达式的过程为：首先根据 *Music.title* 上的 Attribute-Index 索引检索PIT_{MML}，得到所有满足谓词"$title=$'foo'"的结果集合 S_0；其次通过扫描 *Lyrics* 类找到所有 *lyrics* 属性值为"loo"的对象集合 S_1，再根据 *OID* 将 S_1 与 PIT_{MML} 进行等值连接操作，得到 PIT_{MML} 的结果集合 S_2；再求 S_0 与 S_2 的交集得到集合 S_3；再根据 *Lyrics* 对象的 *OID* 将 *Lyrics* 与 S_3 进行等值连接操作，得到所需 *Lyrics* 对象的集合 S_4，最后再对 S_4 中的每个对象投影计算 *lyrics* 属性的值.

3.3　PNI 索引的维护

数据库的各种变化会影响到 PNI 索引，本节主要讨论 PNI 索引的维护问题. 在数据库中，类与代理类的创建和删除，会导致代理层次发生变化，尤其是删除类或代理类时，会导致路径失效. 如果一个路径上某个类或代理类被删除，那么在该路径上所创建的 PNI 索引需要被删除. 另外，由于对象与代理对象的新增、删除和更新等操作，会导致路径实例发生变化进而影响 PNI 索引. 下面以数据更新操作(UPDATE)为例，讨论 PNI 索引的维护.

在对象代理数据库中，数据更新操作分为两大步骤：一是更新数据属性值，二是对象更新迁移. 在对象更新迁移过程中，可能导致新代理对象的派生或已有代理对象的删除，其本质是新增或删除对象(代理对象)间的代理关系. 数据更新操作导致 PNI 索引的更新主要分以下两种情况考虑：(1)只有数据属性值发生变化；(2)数据属性值发生变化，而且引起对象更新迁移.

(1) 只有数据属性值发生变化的情况

在这种情况下，由于没有发生对象更新迁移，对象(代理对象)间的所有代理关系并没有发生变化，路径实例所赖以存在的对象间双向指针链接并没有失效，因此数据更新操作不影响路径实例，PNI 索引的 Path-Instance-Table 和 Identity-Index 都不需要进行更新，可能被影响的只有 Attribute-Index 索引. 如果 PNI 索引的任一 Attribute-Index 不是建立在被更新属性上，那么数据更新操作并不影响 PNI 索引. 否则，相应的 Attribute-Index 索引需要被更新.

假设在路径$P=C_1\rightarrow C_2\cdots\rightarrow C_n$上建有 PNI 索引，而且对于路径上的类(代理类)$C_i(1\leq i\leq n)$的属性 w 建有 Attribute-Index 索引. 当更新 C_i 的一个对象(代理对象)q 的属性 w 时，令包含 q 的路径实例所对应的 Path-Instance-Table 表元组的物理地址为 *addr*，对 Attribute-Index 索引的更新包括两个步骤：首先扫描 Attribute-Index 索引，从对应 w 的旧属性值的叶子结点中删除 *addr*；再次扫描 Attribute-Index 索引，将 *addr* 插入到 w 的新属性值所对应的叶子结点中.

(2) 数据属性值发生变化并引起对象更新迁移的情况

对于这种情况，除了数据的属性值被更新，还发生了对象更新迁移. PNI 索引的更新主要分为两个过程，一是伴随数据属性值更新过程的 PNI 索引更新，一是伴随对象更新迁移过程的 PNI 索引更新. 对于伴随数据属性值更新过程的 PNI 索引更新与情况(1)相同，下面主要考虑对象更新迁移过程的 PNI 索引更新.

对象更新迁移过程涉及到新增代理对象或删除已有代理对象，从而导致新增或删除对象(代理对象)间的代理关系，进一步引起新增路径实例或已有路径实例失效，其对 PNI 索引的影响主要是由于路径实例的变更所造成的，因此我们只需考虑对象更新迁移过程中造成路径实例变更的两个基本操作：$ins_bipointer(o^s,o^d)$与$del_bipointer(o^s,o^d)$. 令 o^s 是类(代理类)C^s 的实例，o^d 是代理类 C^d 的实例，C^d 是 C^s 的一个代理类，且 o^d 是 o^s 的一个代理对象，$ins_bipointer(o^s,o^d)$表示新增 o^s 与 o^d 间的直接代理关系，即新增两对象间的双向指针链接；$del_bipointer(o^s,o^d)$表示删除 o^s 与 o^d 间的直接代理

关系，即删除两对象间的双向指针链接.

不失一般性，考虑路径 $P=C_1\rightarrow\cdots\rightarrow C_i\rightarrow C_{i+1}\rightarrow\cdots\rightarrow C_n$，假设 C_{i+1} 是 C_i 的代理类，o_i 是类(代理类) C_i 的实例，o_{i+1} 是代理类 C_{i+1} 的实例. *ins_bipointer*(o_i,o_{i+1})操作引起的 PNI 索引更新的过程为：(1)从 o_i 出发，根据路径 $P'_1=C_1\rightarrow C_2\rightarrow\cdots\rightarrow C_i$ 逆向遍历对象双向指针链，确定含有 o_i 的 P'_1 的路径实例集合 *PPI.new*；(2)从 o_{i+1} 出发，根据路径 $P'_2=C_{i+1}\rightarrow\cdots\rightarrow C_n$ 前向遍历对象双向指针链，确定含有 o_{i+1} 的 P'_2 的路径实例集合 *SPI.new*；(3)逐一取出 *PPI.new* 中的每一个路径实例 *ppi*，将其与 *SPI.new* 中任一路径实例 *spi* 进行连接，即 *ppi*→*spi*，构成路径 P 的新的路径实例集合 *PI.new*；(4)逐一取出 *PI.new* 的每一个路径实例 *pi*，构造相应的一条元组添加到 Path-Instance-Table，同时更新 Identity-Index 与 Attribute-Index. 令新增的每一个路径实例 *pi* 所对应的 Path-Instance-Table 元组的物理地址为 *newaddr*. 对 Identity-Index 的更新主要是根据 Path-Instance-Table 元组对应列的 *OID* 值，扫描该 Identity-Index，将 *newaddr* 添加到与该 *OID* 相对应的叶子结点中. 对于 PNI 中建立在 $C_k(1\leq k\leq n)$ 的属性 w 上的 Attribute-Index 索引，找到 *pi* 中属于 C_k 的对象实例，进一步获取属性 w 的值，再通过扫描 Attribute-Index 索引，将 *newaddr* 添加到与属性 w 的值相对应的叶子结点中.

del_bipointer(o_i,o_{i+1})操作所引起的 PNI 索引更新过程为：(1)根据 o_i 扫描相应的 Identity-Index，得到 Path-Instance-Table 的一个元组集合 PI_1；(2)根据 o_{i+1} 扫描相应的 Identity-Index，得到一个 Path-Instance-Table 的元组集合 PI_2；(3)求 PI_1 与 PI_2 的交集 *PI*；(4)从 Path-Instance-Table 中逐一删除 *PI* 的每一个元组 *pit*，同时更新 Identity-Index 与 Attribute-Index. 令所删除的每一个元组 *pit* 的物理地址为 *oldaddr*. 对 Identity-Index 的更新主要是根据 *pit* 对应列的 *OID* 值，扫描该 Identity-Index，将 *oldaddr* 从与该 *OID* 相对应的叶子结点中删除. 对于 PNI 中建立在 $C_k(1\leq k\leq n)$ 的属性 w 上的 Attribute-Index 索引，找到 *pit* 中在相应列所记录的 C_k 实例的 *OID*，根据该 *OID* 得到具体的 C_k 实例，进一步获取属性 w 的值，再通过扫描 Attribute-Index 索引，将 *oldaddr* 从与属性 w 的值相对应的叶子结点中删除.

4 实 验

本文提出的 PNI 索引已在对象代理数据库中实现. 我们通过实验来分析各种影响路径表达式计算的因素，验证 PNI 索引方法的有效性，并将该方法与多种不同方法进行性能比较. 所有实验都是在对象代理数据库系统 TOTEM 中完成.

4.1 实验环境

实验使用的测试环境是一台 PC 机，其配置如下：Intel Celeron 2.0 GHz CPU，2GB 内存容量，200GB 硬盘，Ubuntu 9.10 操作系统，TOTEM 2.0 数据库系统. 实验采用的测试数据库是一个音乐数据库，数据库模式如图 1 所示. 实验使用不同规模的测试数据集，每个数据集中各个类(代理类)所包含的对象(代理对象)数量如表 1 所示.

表 1 测试数据集

数据集	类(代理类)								
	Artist	*Music*	*MTV*	*Lyrics*	*Picture*	*Album*	*Music_Lyr*	*Music_MTV*	*Album_Pic*
DS1	500	5000	5000	5000	1000	1000	5000	5000	1000
DS2	1000	10000	10000	10000	2000	2000	10000	10000	2000
DS3	2000	20000	20000	20000	4000	4000	20000	20000	4000
DS4	3000	30000	30000	30000	6000	6000	30000	30000	6000
DS5	5000	50000	50000	50000	10000	10000	50000	50000	10000
DS6	8000	80000	80000	80000	16000	16000	80000	80000	16000

目前对象代理数据库系统 TOTEM 中，路径表达式计算采用的是 2.3 节所述的指针跟踪(PT)方法. 实验将对 PNI 索引方法与 PT 方法进行性能比较. 另外，OODB 领域提出了多种索引结构，如 Multiple Join，Nested Index 和 Path Index 等，用于支持高效的对象遍历，提高 OODB 中路径表达式计算的效率. 虽然这些方法不能直接应用于 ODDB 路径表达式计算，但这些索引的设计思想值得借鉴. 因此我们根据 Multiple Join，Nested Index 与 Path Index 等 3 种索引的核心思想，设计了另外 3 种可以用于支持 ODDB 路径表达式计算的辅助结构，分别是 MJ-Index、N-Index 以及 P-Index，用于减少路径表达式计算过程中对象遍历的开销. MJ-Index 为路径的每条边建立一个辅助表，存储路径实例上前后两个对象实例间的代理联系. N-Index 是一个只存储路径实例上第一个和最后一个对象实例间映射关

系的辅助表. P-Index 是一个物化存储完整路径实例的辅助表，其结构与 Path-Instance-Table 相同. 以上 3 种辅助结构都只记录对象实例的 *OID*，并不存储实际完整的对象实例.

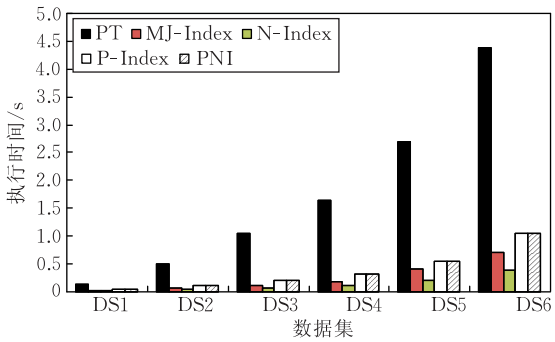
利用 MJ-Index、N-Index 与 P-Index，我们实现了 3 种计算路径表达式的方法. 其中，N-Index 方法只适用于不带谓词的路径表达式计算. MJ-Index 与 P-Index 可用于各种情况的路径表达式计算，不过这两个方法在计算带谓词的路径表达式时，需要获取路径实例的中间对象实例进行谓词计算，以过滤不满足条件的路径实例. 采用 N-Index 计算不带谓词的路径表达式时，只需扫描 N-Index 一次就能获得路径实例上最后一个对象实例，进一步投影计算目标表达式的值. MJ-Index 方法计算不带谓词的路径表达式时，需要将路径上每条边的 MJ-Index 进行连接，从而得到完整的路径实例，再由路径实例上最后一个对象实例投影计算目标表达式. MJ-Index 方法计算带谓词的路径表达式，需要首先扫描定义有谓词的类(代理类)，得到满足条件的该类(代理类)的实例集合，再将该实例集合与路径上每条边的 MJ-Index 进行连接，从而得到满足条件的路径实例，进一步再由实例上最后一个对象实例投影计算目标表达式. P-Index 方法计算不带谓词的路径表达式时，通过扫描 P-Index 可获取路径实例上最后一个对象实例，从而计算目标表达式的值. 当计算带谓词的路径表达式时，与 MJ-Index 方法相似，同样需要先扫描定义有谓词的类(代理类)，得到满足条件的该类(代理类)的实例集合，再将该实例集合与 P-Index 进行连接，从而得到满足条件的路径实例，再由路径实例上最后一个对象实例投影计算目标表达式.

在上述工作基础上，根据以下实验将 PNI 方法与 PT、MJ-Index、N-Index 和 P-Index 等方法进行比较分析.

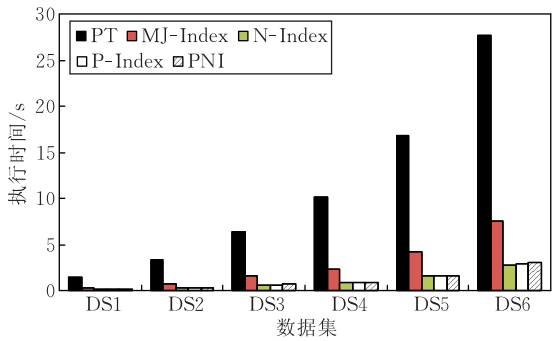
4.2　实验结果与分析

4.2.1　路径长度对路径表达式计算的影响

第 1 组实验研究路径长度对不同路径表达式计算方法的影响，在不考虑带谓词条件与虚属性的情况下，分别以路径长度为 3 和 6 的路径表达式作为测试例子，提交到不同数据集中进行计算，得到不同数据规模下各个方法的执行时间. 从图 7 可以看出，在路径长度一定的情况下，每种方法计算路径表达式的时间随着数据规模的增长而增加. 对比图 7(a)与图 7(b)，在相同数据规模的数据集中，每种方法针对路径较长的路径表达式的计算时间，都比路径较短的路径表达式计算的时间要长，即路径表达式计算的时间随着路径长度的增加而增加. 从实验的结果可以看到，在不考虑带谓词条件的情况下，在各种数据规模的数据集中，不论是短路径或长路径的路径表达式，PT 方法的计算效率是最差的，这是因为 PT 方法以一种面向记录的方式，每次计算一个结果，在计算过程中需要不断地进行对象遍历，大量的对象遍历操作导致总的计算代价较大. MJ-Index 方法对于短路径的路径表达式计算较有效，而对于长路径的路径表达式计算，其性能较差，其原因是 MJ-Index 方法需要将多个 MJ-Index 辅助表进行连接，路径越长，所需连接操作越多，代价越大. PNI 方法与 P-Index 方法性能相当，这是因为两种方法都将路径实例进行了物化存储. 这两种方法比 N-Index方法效率差，这是因为 N-Index 只存储路径实例上首尾两个对象实例，扫描辅助设施的代价最小，因此效率在大多数情况下较高.



(a) 路径表达式计算(路径长度=3)



(b) 路径表达式计算(路径长度=6)

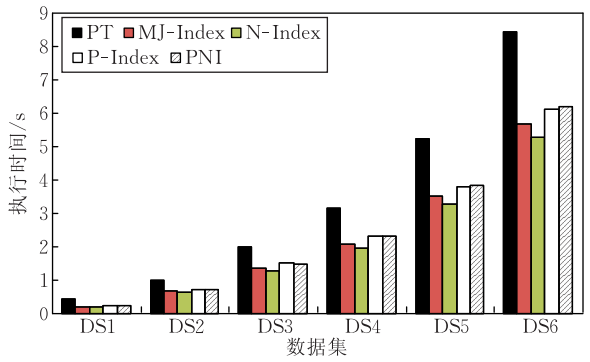
图 7　路径长度对路径表达式计算的影响

4.2.2　虚属性对路径表达式计算的影响

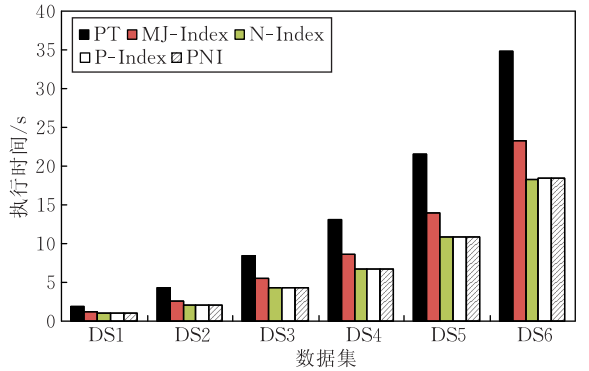
在不考虑带谓词条件的情况下，第 2 组实验研究虚属性对路径表达式计算的影响. 虚属性可能作为谓词属性在路径表达式中出现，也可能是路径表达式的目标表达式中的属性. 本组实验以目标表达式是虚属性的情况为例，分别以长度为 3 和 6 的路

径表达式作为测试例子，在不同数据集中提交给不同的方法进行计算. 本组实验的测试例子，与第 1 组实验的用例在路径表达式的导航路径部分完全一样，区别只在于第 1 组实验用例的目标表达式是实属性，而本组实验用例的目标表达式是虚属性.

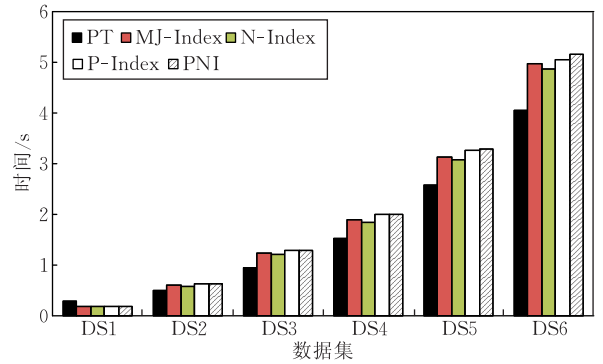
从图 8(a)与(b)可以得出，对于含有虚属性的路径表达式，第 1 组实验得出的各个结论依然成立. 图 8(c)与(d)由图 7(a)、(b)与图 8(a)、(b)的测试结果得到，图 8(c)显示了在路径长度为 3 的情况下，各种方法在计算具有相同导航路径的路径表达式时，由于将目标表达式由实属性转换为虚属性所带来的计算开销增长. 图 8(d)则对应于路径长度为 6 的情况. 由这两个图可得，虚属性增加了各种方法计算路径表达式的开销. 这是因为在计算涉及虚属性的某个路径表达式时，由于虚属性的原因，需要嵌套计算在虚属性切换表达式定义中所使用的其它路径表达式，计算代价由此随之上升.



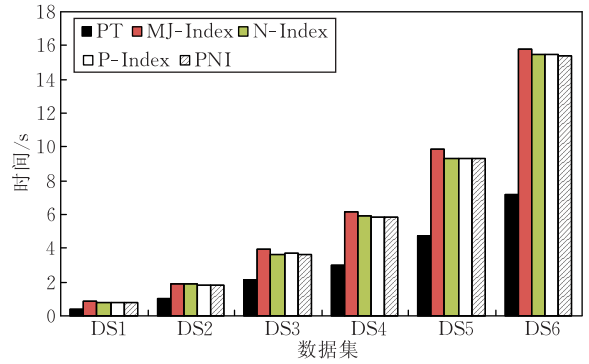
(a) 带虚属性路径表达式计算(路径长度=3)



(b) 带虚属性路径表达式计算(路径长度=6)



(c) 虚、实属性路径表达式计算时间差(路径长度=3)



(d) 虚、实属性路径表达式计算时间差(路径长度=6)

图 8 虚属性对路径表达式计算的影响

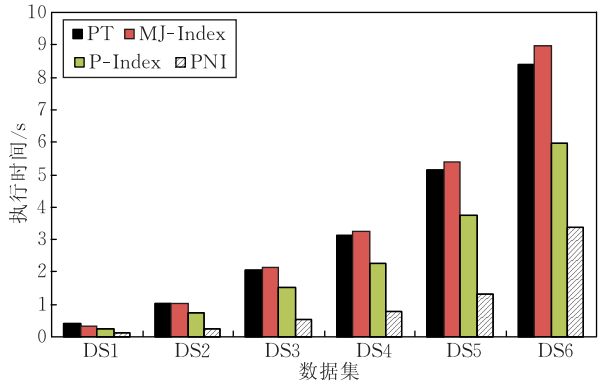
由第 1 组和第 2 组实验的结果可以得出以下结论：在不考虑带谓词条件的情况下，PT 方法的效率是最差的；MJ-Index 方法较适用于短路径的路径表达式计算，在长路径的路径表达式计算时，相对于 PNI 方法、N-Index 方法和 P-Index 方法没有优势. N-Index 方法表现较好，而 PNI 方法与 P-Index 方法次之，后两者的性能相当.

4.2.3 谓词条件对路径表达式计算的影响

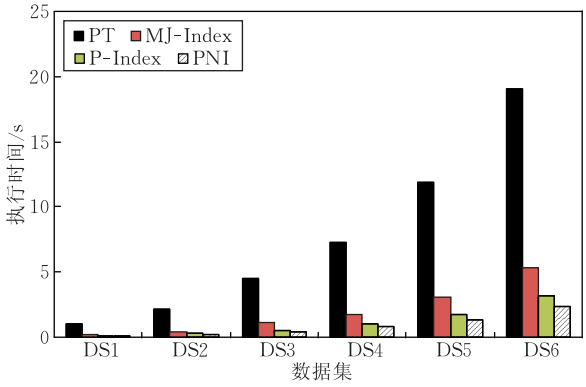
第 3 组实验研究谓词条件对路径表达式计算的影响. 由于 N-Index 不支持带谓词的路径表达式计算，因此本组实验只针对其它 4 种方法进行比较分析. 实验在路径一致、目标表达式一致的情况下，以不同谓词组合的路径表达式作为测试用例，分别测试了带一个实属性谓词、带一个虚属性谓词以及带两个虚属性谓词的路径表达式. 涉及 PNI 方法的测试，都在 PNI 索引中为谓词属性创建了相应的 Attribute-Index 索引.

从实验结果来看，PNI 方法受虚属性谓词和实属性谓词的影响最少，对于带谓词条件的路径表达式计算，其性能是 4 种方法中最好的，这主要受益于 PNI 索引中的 Attribute-Index 索引，可以避免不必要的谓词计算，快速过滤不满足条件的路径实例. PNI 方法相对于其它方法的的优势，在涉及虚属性谓词的路径表达式计算时体现得更加明显. 在带谓词条件的情况下，PT 方法的效率不一定是最差的，如图 9 所示，虽然在带一个虚属性谓词的情况下，PT 方法的性能是最差的，但是在带两个虚属性谓词的情况下，其性能仅次于 PNI 方法. 这是因为 PT

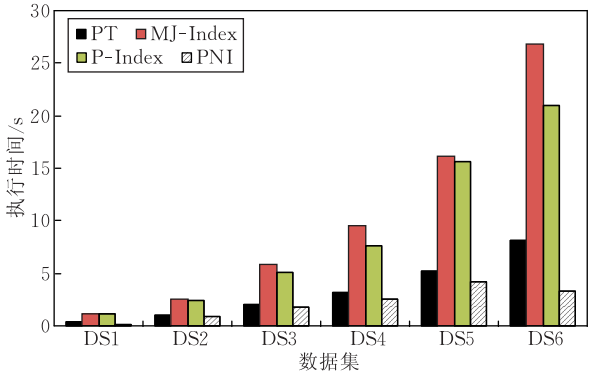
方法是面向记录的方法，在对象遍历的过程中，一旦发现谓词条件不满足，就及时中止该趟计算流程. 谓词条件判断得越早，PT 方法就能避免越多不必要的对象遍历开销. 之所以 PT 方法在两个虚属性谓词的情况下性能反而比一个虚属性的情况下的性能要好，这主要是由于测试路径的选择，使得 PT 方法能在前一种情况下及时中止对象遍历. 对于 MJ-Index方法与 P-Index 方法来说，由于两种方法在进行谓词条件判断时都需要获取中间对象实例，因此谓词数量越多，两种方法在谓词计算方面的开销就越大，从而影响了总的计算开销，这可以通过比较图 9(b)与(c)的结果得出. 至于出现如图 9(a)与(b)所示结果，MJ-Index 方法与 P-Index 方法在实属性谓词情况下比虚属性谓词情况下计算开销的增加，这完全是由于路径谓词的选择所造成的. 虚属性谓词情况下，过滤得到满足条件的路径实例较少，使路径表达式的后续计算开销较小；而实属性谓词的情况下，过滤得到满足条件的路径实例较多，所以后续计算的开销较大.



(a) 带一个实属性谓词的路径表达式计算



(b) 带一个虚属性谓词的路径表达式计算



(c) 带两个虚属性谓词的路径表达式计算

图 9　谓词条件对路径表达式的影响

综合以上 3 组实验的结果可以得出结论：PNI 索引能够提高路径表达式计算的效率，PNI 方法在不考虑谓词条件的情况下，性能表现良好；对于存在谓词条件的路径表达式计算，PNI 方法的性能最优，相较于其它方法具有明显的优势.

4.2.4　PNI 索引存储开销

本次实验主要研究 PNI 索引的存储开销. 实验以图 1 所示音乐数据库为背景，为路径 $P=Picture\rightarrow Album\_pic\rightarrow Album\rightarrow Music\_Lyr\rightarrow Music\rightarrow Music\_MTV$ 创建 PNI 索引，该 PNI 索引包含一个 Path-Instance-Table 表，各 Identity-Index索引以及为类 *Music* 的“*title*”属性所创建的一个 Attribute-Index 索引. 实验分别测试了在不同数据规模下的 PNI 索引的存储代价，其结果如图 10 所示. 从实验的结果分析，对于一个路径的 PNI 索引，随着数据规模的增大，索引所占存储空间也随之增加. 这是因为：数据量的增大，必然增加了路径实例的数量，Path-Instance-Table 表需要存储更多的路径实例. Path-Instance-Table 表的数据量一增大，Identity-Index 索引规模也随之增加. 一旦 PNI 索引建有 Attribute-Index 索引，随着 Attribute-Index 索引属性所在类的数据规模的增大，Attribute-Index 索引的规模也会随之增加.

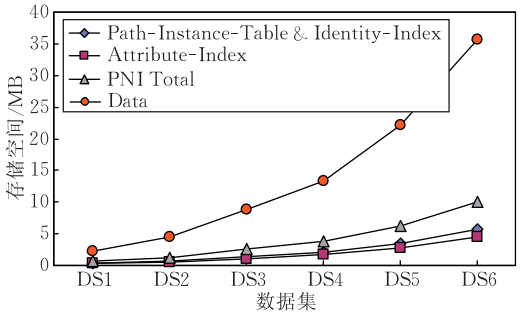


图 10 PNI 存储空间开销

5 总 结

在代理层次中进行对象导航遍历是对象代理数据库的一个重要特点，对象代理查询语言的路径表达式可以有效满足对象遍历的表达需求，是支持诸如跨类查询、代理对象查询等查询功能的重要设施. 对象代理数据库需要研究有效的存取方法以支持在代理层次中进行高效的对象导航遍历，本文正是在这一背景下，提出了一种新的索引结构——路径导航索引 PNI 来计算代理层次中的路径表达式.

PNI 索引物化了路径实例，并利用属性索引技术支持对路径实例进行关联检索. 通过物化路径实例，在计算路径表达式的过程中可以尽量减少对象遍历的开销；对路径实例关联检索的支持可避免谓词计算给路径表达式计算所带来的开销. 正是从对象遍历与谓词计算两个侧面进行改进，PNI 索引提高了路径表达式的计算效率. 本文的实验对各种可能影响路径表达式计算的因素进行了研究，分析了路径长度、虚属性与谓词条件等因素对路径表达式计算效率的影响. 实验验证了 PNI 索引对路径表达式计算的有效性，尤其是针对具有谓词的路径表达式，PNI 索引能实现高效计算. 同时我们也必须认识到，利用 PNI 索引进行路径表达式计算的方法不能完全取代现有对象代理数据库中指针跟踪的计算方法，因为 PNI 索引并不是在所有情况下性能表现都是最优的. 而且 PNI 索引在提高路径表达式计算效率的同时，也会增加系统的存储开销. 本文的建议是，对于频繁查询且效率低下的路径表达式，可以合理选择 PNI 索引来提高路径表达式计算的效率.

参 考 文 献

[1] Peng Z Y, Kambayashi Y. Deputy mechanisms for object-oriented databases//Proceeding of the 11th International Conference on Data Engineering. USA, 1995: 333-340

[2] Kambayashi Y, Peng Z Y. Object deputy model and its applications//Proceedings of the 4th International Conference on Database Systems for Advanced Applications. Singapore, 1995: 1-15

[3] Peng Z Y, Li Q, Feng L, Li X H, Liu J Q. Using object deputy model to prepare data for data warehousing. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(9): 1274-1288

[4] Peng Zhi-Yong, Luo Yi, Shan Zhe, Li Qing. Realization of workflow views based on object deputy model. Chinese Journal of Computers, 2005, 28(4): 651-660(in Chinese)
(彭智勇，罗义，单喆，李青. 基于对象代理模型的工作流视图实现. 计算机学报，2005，28(4)：651-660)

[5] Wang Li-Wei, Huang Ze-Qian, Luo Min, Peng Zhi-Yong. Data provenance in a scientific workflow service framework integrated with object deputy database. Chinese Journal of Computers, 2008, 31(5): 721-732(in Chinese)
(王黎维，黄泽谦，罗敏，彭智勇. 集成对象代理数据库的科学工作流服务框架中的数据跟踪. 计算机学报，2008，31(5)：721-732)

[6] Peng Z Y, Shi Y, Zhai B X. Realization of biological data management by object deputy database system. Transactions on Computational Systems Biology, 2006, LNCS 4070: 49-67

[7] Peng Z Y, Peng Y W, Zhai B X. Using object deputy database to realize multi-representation geographic information system//Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems. Seattle, Washington, 2007: 43

[8] Zhai B X, Shi Y, Peng Z Y. Object deputy database language//Proceedings of the 4th International Conference on Creating, Connecting and Collaborating through Computing. USA, 2006: 88-95

[9] Cattell R G G. The Object Database Standard: ODMG-93. USA: Morgan Kaufmann, 1993

[10] Kifer M, Kim W, Sagiv Y. Querying object-oriented databases//Proceedings of the ACM SIGMOD International Conference on Management of Data. USA, 1992: 393-402

[11] Gardarin G, Gruser J R, Tang Z H. Cost-based selection of path expression processing algorithms in object-oriented databases//Proceedings of the 22th International Conference on Very Large Data Bases. Bombay, India, 1996: 390-401

[12] Wang Guo-Ren, Yu Ge, Zhang Bin, Zheng Huai-Yuan. The parallel algorithms for path expressions. Chinese Journal of Computers, 1992, 22(2): 126-133(in Chinese)
(王国仁，于戈，张斌，郑怀远. 路径表达式的并行算法研究. 计算机学报，1992，22(2)：126-133)

[13] Maier D, Stein J. Indexing in an object-oriented database//Proceedings of the IEEE Workshop on Object-Oriented Data Base Management Systems, USA, 1986: 171-182

[14] Xie Z, Han J. Join index hierarchies for supporting efficient navigations in object oriented databases//Proceedings of the 20th International Conference on Very Large Data Bases. Chile, 1994: 522-533

[15] Bertino E, Foscoli P. Index organizations for object-oriented database systems. IEEE Transactions on Knowledge and Data Engineering, 1995, 7(2): 193-209

[16] Bertino E, Guglielmina C. Path-Index: An approach to the efficient execution of object-oriented queries. Data and Knowledge Engineering, 1993, 10(1): 1-27

[17] Lee W C, Lee D L. Path Dictionary: A new access method for query processing in object-oriented databases. IEEE Transactions on Knowledge and Data Engineering, 1998, 10(3): 371-388

[18] Kemper A, Moerkotte G. Access support in object bases// Proceedings of the ACM SIGMOD International Conference on Management of Data. USA, 1990: 364-374

HUANG Ze-Qian, born in 1983, Ph. D. candidate. His current research interests include database and scientific workflow.

PENG Zhi-Yong, born in 1963, professor, Ph. D. supervisor. His main research interests include advanced database management and web information management.

LI Yue, born in 1987, M. S. candidate. His current research interests include database and biological information management.

PENG Yu-Wei, born in 1980, Ph. D., lecturer. His current research interests include database and geographical information system.

Background

Object deputy model was proposed as an extension to the traditional object-oriented model by introducing the new concept of deputy object. The previous research has shown that the object deputy model has advantages over the traditional object-oriented model in many fields. TOTEM, the first object deputy database system based on the object deputy model, has been developed by the database group in Wuhan University.

Efficient query processing is the most important issue for the performance of a database system. With regard to the object deputy database system, deputy object query and cross class query are two typical types of query. The object deputy query language provides the path expression as the basic facility to support these two types of query. Efficient evaluation of path expression will lead to efficient processing of deputy object query and cross class query. In the field of object-oriented database, path expression evaluation has been recognized as a key problem to query processing. There are three basic methods for path expression evaluation: forward pointer chasing, reverse pointer chasing and join-based method. A few kinds of index have been also proposed for query optimization in previous literatures, such as multiple index, join index, nested index, path index, et al. However, there are differences between path expression of object deputy database and that of object-oriented database, and those methods could not be applied directly.

In the object deputy database system TOTEM, forward pointer chasing method has been implemented, while there is still no research on the index issue. This study aims at improving the efficiency of path expression evaluation in object deputy database based on the index method. The research in this paper proposed the new type of index, path navigation index, for path expression in object deputy database. Evaluation of path expression with this index avoids redundant object traversal and reduces the cost of predicate evaluation. The experimental results of this study show that query processing in the object deputy database takes the advantages of the path navigation index.

This work is supported by the Major State Basic Research Development Program of China under grant No. 2007CB310806, the Major State Research Program of National Natural Science Foundation of China under grant No. 90718027, the Major Program of Natural Science Foundation of Hubei Province under grant No. 2008CDA007, and the Fundamental Research Funds for the Central Universities under grant No. 6082011.