复杂对象数据库的递归查询语义*

陈 睿

(华南理工大学计算机系 广州 510641)

摘要 本文用代数规范方法定义了复杂对象数据库中的对象标识、对象值、查询谓词和数据库状态语义,通过对查询路径的分析,给出了个体对象查询条件匹配的语义,特别是在个体对象上施用递归查询的语义.最后,给出了OODB中选择操作的语义.

关键词 代数规范,复杂对象数据库,递归查询,形式语义.

1NF关系模型中,关系的结构犹如二维表.二维表固然是一种简单和易于操作的数据结构,但对于OIS和CAD等新应用却缺乏适当的建模能力.这是→1NF关系模型及复杂对象模型提出的动因.

→1NF关系模型和复杂对象模型都试图提供比二维表更为复杂的数据结构,而在复杂的数据结构中递归结构是不可避免的.文献[1]是对→1NF关系模型研究的理论总结,给出了扩充关系代数和扩充关系演算,但该文通过限制关系模式的定义以避免递归结构.文献[2]将递归操作列为复杂对象的基本操作,后来这种思想被融合在ORION的查询语言中.[3]文献[4]给出了一个类似关系演算的对象演算系统,名为OOPC,其中包含一个表示递归查询路径的原语ρ.其中ORION查询语言(一种语法类似SQL的OODB查询语言)和OOPC代表了当前OODB查询语言研究的最新成果,二者所提供的递归查询能力从直观上很容易判断是等价的,但由于缺乏精确的语义定义,要在数学上证明这一点则很困难.

OODB模型的形式化研究是目前的一个重要的国际课题.本文的目的正是用代数规范方法来定义复杂对象数据库递归查询的形式语义.我们不妨将OODB看作是复杂对象数据库加上类层次结构,这样OODB和复杂对象数据库在递归查询特性上是共同的.这里,我们只考虑OOPC和ORION所支持的递归查询能力,在文献[3]中称为"线性递归查询".关于代数规范方法的细节,见文献[5].

1 个体和集体

我们不妨将数据库中的基本聚合单位称作集体(Collection).集体是由一系列个体(Entity)构成的.在关系数据库中,关系是集体,元组是个体.在大多数OODB模型中,类与特定类型相关联(内涵),并隐式地指称该类型的所有实例(外延),在此意义下,类的外延是

* 本文研究得到广东省科学基金项目资助.作者陈睿,1968年生,讲师,主要研究领域为面向对象系统,数据库系统.
本文通讯联系人:陈睿,广州510641,华南理工大学计算机系
本文1995-03-15收到修改稿

集体,对象是个体.对一个集体施加一个查询,其结果是另一个集体,这就是所谓的查询封闭性.关系数据库的查询封闭性是严格保持的;而对于 OODB 和复杂对象数据库,则是一个迄今尚未有公认结果的研究焦点,充满争议.这里,我们要想先避开这个问题,就不得不考虑个体的作用.我们注意到,施加于集体上的查询最终是通过判断集体中各个体是否满足查询条件所实现的,因而集体在查询中的作用实际上包括 3 方面:指明查询范围;提供迭代机制;聚合查询结果.

集体所提供的迭代机制将查询施加于每一个个体,可见,与查询条件相比较的是个体而非集体.递归查询的处理也不例外.因此,我们先在对象一级讨论递归查询条件的匹配问题,然后再回到集体层次.

2 值,对象和数据库状态

关系数据库是基于值的,因此更便于形式化.复杂对象数据库是基于对象的,与作为数学抽象的值概念相比,对象是一个完全不同的概念.对象可创建和销毁,有生存期,具有可变的状态(可表示为值)和不可变的唯一标识.在对象概念中,对象标识是一个关键特性,因此,对象标识被认为是复杂对象数据库的一项基本特性,是区分¬1NF 关系数据库与复杂对象数据库的重要标准.我们可以用一个代数规范 OID 来定义对象标识的语义.

OID≡{ sort oid
nil:→oid,
……}

oid 类子(sort)可解释为所有对象标识的集合.对于它的构造我们不感兴趣,只强调一个特殊的对象标识,它由零元函数 nil 生成,是 oid 中的一个常元,直观上表示空对象(不存在的对象).

根据 Khoshafian 和 Coperland 的对象模型[6],对象值分为 3 类:

1)原子值(Atom):整数,字符串,布尔值等(这里只考虑整数);

2)元组(Tuple):形为$[A_1:i_1,\cdots,A_n:i_n]$,其中 $A_1,\ldots,A_n$ 为属性名,$i_1,\ldots,i_n$ 为对象标识;

3)集合(Set):形如$\{i_1,\ldots,i_n\}$,其中 $i_1,\ldots,i_n$ 为对象标识.

我们用代数规范 VALUE 来定义所有这 3 类值及其关系操作的语义:

VALUE≡{
based on INTEGER,BOOLEAN,STRING,OID
sort value,A
atomic,tuple,set:value→bool,
_ =i_,_>i_,_<i_,_⊇s_,_⊆s_:value×value→bool,
_ ∈_,_:oid×value→bool,
_ @_:value×A→oid
……}

我们假定 INTEGER,STRING 和 BOOLEAN 是已定义的 3 个基本代数规范.类子 value 解释为所有值的集合,A 为属性名集合.这里省去类子 value 的构函数.由于所列出函

数的语义是直观且易于理解的,我们省去了所有公理,只作一些非形式的解释.谓词函数atomic,set 和 tuple 用于判定每个值是原子值、元组值还是集合值.中缀形式的=i,>i 和<i操作符表示整数原子值间的相等或大小关系,其它类型原子值的基本关系运算符没有列出.⊇s,⊆s 表示对象标识集合的包含关系.@表示元组值取属性的运算.这些操作符都是作用于整个值集 value 上的,如果操作数类型不合适,则结果无定义,因此 VALUE 是一个偏代数规范.例如,我们有公理:set((v1)=false⇒undefined(v1⊆$_s$,v2)

即若 V1 不是集合值,则 v1⊆$_s$,v2 无意义.同样,若 v1 不是元组值,则 v1@a1 无定义.若 v1 是元组,但不包含 a1 属性,v1@a1 也无定义.

数据库状态是由所有现存对象及其值构成的.任意的数据库状态都是由初态经过一系列变换得到的.这些变换可以是:创建对象、销毁对象、消息传递.

由于上述变换语义的定义较复杂,本文不对其进行形式定义.这部分工作在另文中论述,本文只把初态和变换都视作类子 state 的构造函数,并在 STATE 规范中略去.这并不影响后面的讨论,因为本文并不考虑查询对数据库状态的影响,只考虑"无副作用"的查询.

STATE ≡{

based on VALUE

sort state,P

atomic,tuple,set:value→bool,

=$_p$,>$_p$,<$_p$,⊇$_p$,⊆$_p$,= =,=$_s$,=$_d$:→P,

υ:state×oid→value,

comp:state×oid×P×oid→bool

……}

类子 P 解释为查询条件中可以包含的基本查询谓词.其中= =,=s 和=d 分别表示对象等同、浅相等和深相等(Object Identical,Swallow Equal,Deep Equal,见文献[6]),函数 v 映射给定数据库状态下的对象值.comp 映射对象间比较运算的逻辑值,这是查询所需的基本操作,作用于个体对象上.STATE 规范包含以下相关公理:

∀ s∈state;i1,i2∈oid,

comp(s,i1,<$_p$,i2)=υ(s,i1)<$_i$υ(s,i2)

comp(s,i1,=$_p$,i2)=υ(s,i1)=$_i$υ(s,i2)

comp(s,i1,>$_p$,i2)=υ(s,i1)>$_i$υ(s,i2)

comp(s,i1,⊇$_p$,i2)=υ(s,i1)⊇$_s$υ(s,i2)

comp(s,i1,⊆$_p$,i2)=υ(s,i1)⊆$_s$υ(s,i2)

而= =,=$_s$ 和=$_d$ 的语义在文献[6]中已有详细描述.

3　查询路径

由于复杂对象数据库支持复杂的对象结构,其查询本质上是导航式的,这导致了查询路径的复杂化.考察 ORION 查询语言及 OOPC 中查询路径的构造,可以发现每一查询路径都由一些路径片段构成,而路径片段可分为下列几种:

①直接路径片段:由一系列属性名相连而成;②量化集合路径片段:在一直接路径片段

前冠以全称或存在量词，而以一类型为集合的属性名结束；③递归路径片段：在一直接路径片段前冠以递归原语(在 ORION 中为 recursive，OOPC 中为 ρ). 由于递归路径片段定义了一个对象集合而非单个对象，因此也应量化.

例如，有类 employee 定义如下：

```
class employee={name:string;
                manager: employee;
                salary: integer;
                collegue: setof(employee);}
```

并有下列查询：Q1. 查找其顶头上司名为 John 的那些雇员

```
select x from employee where x.manager.name='John'
```

Q2. 查找其某一同事名为 John 的那些雇员

```
select x from employee where x.some(collegue).name='John'
```

Q3. 查找其某一上司名为 John 的那些雇员

```
select x from employee where x.some_recursive(manager).name='John'
```

Q4. 查找其所有上司工资超过 15000 元的那些雇员

```
select x from employee where x.all_recursive(manager).salary>15000
```

其中 Q1 中包含一直接路径片段 manager.name；Q2 中包含一个存在量化的集合路径片段 some(collegue)；Q3 中的 some_recursive(manager)和 Q4 中的 all_recursive(manager)分别是存在量化的和全称量化的递归路径片段. Q2，Q3，Q4 的查询路径中均有 2 个片段，后面的片段都是直接路径片段.

考察 3 类路径片段，可见它们的基本成分都是属性序列. 代数规范 PATH 定义了查询路径的生成：

```
PATH ≡{
    based on STATE
    sort pseg, path=pseg*
    dp, ss, as, sr, ar: A*→pseg,
    access: state×oid×A*→oid}
```

上述规范用到一个类子构造符“*”，其作用是构造序列域.

定义. 对任意类子 s，s^* 也是一个类子，并解释为类子 s 中的元素可能构成的所有序列的集合，包括空序列. s^* 上有下述运算及相关公理：

```
sort s*
    ε: s*      (空序列)
    [_]: s→s*      (单元素序列)
    _·_: s*×s*→s*      (连接)
    ∀ s1, s2, s3∈s*;
    s1·ε=ε·s1=s1, s1·(s2·s3)=(s1·s2)·s3      □
```

类子 path=pseg* 表明查询路径是由路径片段序列构成的. 而路经片段 pseg 的生成函数中，dp 由属性名序列构造直接路径片段，ss，as，sr 和 ar 分别由属性名序列构造存在量化

和全称量化的集合路径及存在量化和全称量化的递归路径.access 函数反映了对象的导航式存取方式,其相关公理如下:

$\forall s\in state; i\in oid; a\in A; b\in A^*$

$access(s,i,\varepsilon)=i$

$access(s,i,[a]\cdot b)=access(s,(v(s,i))@a,b)$

查询处理所涉及的对象一级的操作实际上是对个体对象施用一个布尔函数以判断该对象是否满足查询条件.查询条件则由一个特定谓词(类子 P 的元素)和比较对象构成.比较对象可以是常数值(可视作常对象的值),也可以是从指定对象按指定的存取路径所到达的对象.因此,我们可以确定上述布尔函数的基调(Signature)为:

$$cond: state\times oid\times path\times P\times oid\times A^*\to bool$$

其中第 1 个 oid 参数指明被测试对象,第 1 个 path 参数是查询路径,第 2 个 oid 参数和 A^* 用以指明比较对象.由于路径的复杂性,要对 cond 函数公理化,我们还需要若干辅助函数.首先定义一个直接指定比较对象的布尔函数 cond′:

$$cond': state\times oid\times path\times P\times oid\to bool$$

另外再定义下列辅助函数:

$$recursive: state\times oid\times A^*\times value\to value,$$

$$some, all: state\times value\times path\times P\times oid\to bool$$

recursive 函数的目的是将那些从指定对象按递归路径所能到达的所有对象的标识聚集在一个集合中,生成一集合类型的值.

$\forall v\in value; i\in oid; b\in A^*; s\in state$

$recursive(s,i,b,v)=$ if $access(s,i,b)\in v$ or $access(s,i,b)=nil$ then v

else $recursive(s,access(s,i,b),b,v\cup access(s,i,b))$ fi

显然,$recursive(s,i,b,\varnothing)$正是在状态 s 下,从 i 出发通过存取路径 b 所能递归到达的所有对象标识的集合,其中∅表示空对象标识集.some 和 all 函数满足下列公理:

$\forall l\in path; i1,i2\in oid; p\in P; v,v1,v2\in value; s\in state;$

$set(v)=false\Rightarrow undefined(some(s,v,l,p,i))\wedge undefined(all(s,v,l,p,i))$

$some(s,\varnothing,l,p,i2)=false$

$some(s,\{i1\},l,p,i2)=cond'(s,i1,l,p,i2)$

$some(s,v1\cup v2,l,p,i2)=some(s,v1,l,p,i2)\vee some(s,v2,l,p,i2)$

$all(s,\varnothing,l,p,i2)=true$

$all(s,\{i1\},l,p,i2)=cond'(s,i1,l,p,i2)$

$all(s,v1\cup v2,l,p,i2)=all(s,v1,l,p,i2)\wedge all(s,v2,l,p,i2)$

cond′ 函数满足下列公理:

$\forall l\in path; i1,i2\in oid; p\in P; b\in A^*; s\in state;$

$cond'(s,i1,[dp(b)]\cdot l,p,i2)=cond'(s,access(s,i1,b),l,p,i2)$

$cond'(s,i1,[ss(b)]\cdot l,p,i2)=some(s,v(s,access(s,i1,b)),l,p,i2)$

$cond'(s,i1,[as(b)]\cdot l,p,i2)=all(s,v(s,access(s,i1,b)),l,p,i2)$

$cond'(s,i1,[sr(b)]\cdot l,p,i2)=some(s,rescursive(s,i1,b,\varnothing),l,p,i2)$

$$cond'(s,i1,[ar(b)]\cdot l,p,i2)=all(s,rescursive(s,i1,b,\varnothing),l,p,i2)$$

上面的公理表明，cond′函数的求值过程实际上是沿着查询路径推进的，每一步向前推进一个路径片段，对于有限长的查询路径，这一过程总会结束，于是可根据下面公理求出 cond′的值：

$$cond'(s,i1,e,p,i2)=comp(s,i1,p,i2)$$

其中 comp 函数在 STATE 规范中已定义. 我们现在可以对 cond 进行公理化了：

$$cond(s,i1,l,p,i2,b)=con'(s,i1,l,p,access(s,i2,b))$$

现在，我们把 cond 及所有辅助函数和相关公理纳入规范 MATCH.

MATCH ≡{

based on Path

cond：state×oid×path×P×oid×A*→bool，

cond′：state×oid×path×P×oid→bool，

recursive：state×oid×A*×value→value，

some，all：state×value×path×P×oid→bool}

至此，对象一级上的查询操作就定义好了. 从上述公理可看出对于递归查询路径，我们是采用与量化集合路径同样的方法来处理的.

4 集体上的查询

这一节将讨论如何利用 cond 函数在集体一级上进行查询处理. OODB 中集体的形式语义至今未有公认的定义，而大多数现有的 OODB 系统以类为集体，以对象集合作为类的语义. 我们注意到这种模型有 2 个问题，一是查询封性问题，二是类并不满足集合的数学特性. 关于第 1 个问题，文献[4]有初步的讨论，但未圆满解决；关于第 2 个问题，我们认为现有模型不能刻画消息传递的副作用，其解决办法之一是把 OODB 中的集体视作是特定数据库状态下的对象标识序列，由 COLLECTION 规范中所定义的 db 函数构造：

COLLECTION≡{

based on MATCH

sort collection

db：state×oid*→collection，

ids：collection→oid*，

select1：collection×path×P×oid→collection，

select2：collection×path×P×A*→collection，

……}

函数 select1 是施加于集体上的选择操作，查询条件中的比较对象是由对象标识指定的（对应于关系数据库中根据常值所进行的选择操作）；select2 的查询条件中比较对象是由存取路径所指定的（对应于关系数据库中根据属性所进行的选择操作）. 它们的公理化很简单：

$\forall l\in path;i1,i2\in oid;p\in P;b\in A^*;s\in state;c\in oid^*$；

$ids(db(s,c))=c$

$select1(db(s,e),l,p,i1)=db(s,e)$

$select1(db(s,[i1]\cdot c),l,p,i2)=$ if $cond'(s,i1,l,p,i2)=true$

then db(s,[i1]·ids(select1(db(s,c),l,p,i2)))
else select1(db(s,c),l,p,i2) fi

select2(db(s,ε),l,p,b)=db(s,ε),

select2(db(s,[i1]·c),l,p,b)=if cond(s,i1,l,p,i1,b)=true
then db(s,[i1]·ids(select2(db(s,c),l,p,b)))
else select2(db(s,c),l,p,b) fi

可见,select1 和 select2 是 2 个封闭的运算. 我们可以把类看成是隐式定义的集体,其对象标识序列中包含了所有当前数据库状态下存在的该类实例的标识,它们按某种特定的顺序,例如,创建的先后次序排列. 涉及到多个类的查询处理,依赖于更复杂的迭代机制,在此不再讨论. 第 3 节中的查询 Q4 可表为 collection 类子的基项:

Q4 select1(employee,[sr([manager])]·[dp([name])],$>_p$,15000)

5 结　语

本文用代数规范方法定义了复杂对象数据库中的对象标识、对象值、查询谓词、查询路径、数据库状态及查询操作的语义,特别是在个体对象上施用递归查询的语义. 个体和集体在查询中的作用是不同的,在认识到这一点后,我们成功把递归查询处理局部化在个体对象层次,从而简化了递归查询语义的定义.

参考文献

1 Roth M A, Korth H F, Silberschatz A. Extended algebra and calculus for nested relational databases. ACM-TODS, Dec. 1988,13(4):389～417.
2 Kim W, Chou H T, Banerjee J. Operations and implementation of complex objects. IEEE Trans. on Software Engineering, July 1988,14(7):985～995.
3 Kim W. Introduction to Object－oriented Databases. MIT Press, 1990.
4 Bertino E, Negri M, Pelagatti G *et al*. Object－oriented query languages: the notion and the issues. IEEE Trans. on Knowledge and data engineering, June 1992,4(3):223～237.
5 陆汝钤. 计算机语言的形式语义. 北京:科学出版社,1992.
6 Khoshafian S, Coperland G. Object identity. OOPSLA'86 Proc., 1986. 406～416.

FORMAL SEMANTICS OF RECURSIVE QUERIES IN COMPLEX OBJECT DATABASES

Chen Rui

(*Department of Computer Science South China University of Technique Guangzhou* 510641)

Abstract This paper has defined the formal semantics of object identities, values, query predicates and database states for complex object databases with a hierarchy of algebraic specifications. Semantics of queries, especially recursive queries on individual objects is presented based on insights into query paths. Finally, the semantics of selection operations on OODBs is given.

Key words Algebraic specification, complex object database, recursive query, formal semantics.