关系数据库 复杂对象代数 查询代数 O-O模型

计算机科学1996 Vol. 23 № 2

61-65

对象-关系模型下的复杂对象代数*)

李天柱 边小凡 肖计田

(河北大学计算中心 保定071002)

TP311.13

摘 要 The paper gave a complex object algebra based object-relational model and full object identity. The algebra ensures that operation results keep the object identities, and complete nested structure (for many other complex object algebra, the results are nested-flat mixed structure), to make the semantics of the operations and their results are clear and consistent. The algebra has rather stroing descriptive ability to express constructors, i. e., disassembly and assembly of complex objects, which make it easy to complete the transformation between complex object algebra and flat relational algebra.

关键词 Database, Object-Relationel Model, Algebra, Object identity.

1. 引 言

查询代数是关系模型的重要组成部分。对 NF^2 关系模型和 O-O 模型中的查询代数，很多人做了大量研究[1]。对 NF^2 模型，得出了类似于平面关系代数的结论，但并不能处理对象标识。文[2]中 O-O 查询代数施于复杂对象上，但在好多情况下，其运算结果是平面的或嵌套-平面混合式结构，对象标识语义不清，且重构能力不够。这些研究成果都借助了平面关系代数的研究成果。

商品数据库的发展也表明，O-O 数据库和关系数据库相互渗透。关系数据库在增加 O-O 数据库的能力和特征，而 O-O 数据库也需兼容关系数据库，故研究对象-关系代数是有重要意义的。

对 O-O 模型，复杂对象结构应是完全嵌套的形式，将嵌套结构平面化和嵌套-平面混合结构都不是规范的，操作对象和结果对象都应是完全嵌套的(本文不给出完全嵌套结构的形式定义)。

平面关系代数通过 Π 和 ▷◁ 运算可实现关系模式的重构；而 O-O 模型描述的是复杂的嵌套结构，O-O 查询代数也应能实现复杂嵌套结构的重构，绝不仅仅意味着嵌套运算(nest)和反嵌套运算(unnest)，还应有更强的能力(重构嵌套结构)。

O-O 查询代数不仅应能对嵌套对象进行运算，也应能对嵌套对象的成分对象进行运算，使能被平面关系代数描述的运算也能按 O-O 查询代数表达。这个问题不仅涉及到 O-O 查询代数的查询能力，而且对平面查询和 O-O 查询之间的互相转换也很重要[3]。

2. 复杂对象代数中的运算

本查询代数依据对象-关系模型特征，该数据模型包括的数据类型有：原子数据类型，原子数据类型的集合，元组的集合。每个对象都是由这些数据类型构成，并具有对象标识。对象标识为全对象标识<e-id, c-id i-id>，其中 e-id 为实体对象标识，用户可见，与型和存储位置无关；c-id 为概念对象标识，与型有关，与存储空间位置无关；i-id 为内对象标识，与存储空间有关[4]。

NF^2 关系模型所含数据类型也是原子数据类型和元组的集合，但没有其它类型的集合，也没有对象标识[1]。一般 O-O 模型包括原子数据类型、元组、集合、表(LIST)。表(LIST)是元组的有序集合，是元组集合的特例。其它集合可被看做单属性的元组集合，或附加一任意属性而变成元组之集合。另一方面，表达 is-a 语义的继承特性是 O-O 模型的重要特征，而集合类型没有继承特性[5]，元组是类(class)的最基本数据类型，以表达继承特性。

基于以上所述，本文对数据模型关于数据结构的假设是合理的，且兼有O-O模型、NF^2 模型之特

*)由国家自然科学基金和河北省自然科学基金资助

征，故叫做对象-关系模型。

本文以大写字母表示类(class)，以小写字母表示对象之集合或单个对象，以 F 表示加于对象上的条件谓词。

定义 给定一路经表达式 $A_i.B_j.\cdots.C_k.D_l$，高阶属性 D_l 中的任一成分对象 d，对应一个对象标识序列 $dA_i\text{-id}, dB_j\text{-id}, \cdots, dC_k\text{-id}, dD_l\text{-id}$，此标识序列叫做 d 的对象标识路径。

下面，给出本复杂对象代数的几种主要运算。

选择运算 σ'

$\sigma_F^1(r)' = \{t' \mid t \in r, F(t)\}$；

$\sigma_F^2(r, R_l)' = \{t' \mid t \in r, \Delta = \{s \mid s \in t.\cdots.R_l, F(s)\}, t' = t//\Delta, t'[id] = t[id]\}$.

其中 r 是类 R 的实例集，R_l 为 R 的高阶属性(可以是内嵌的)，$t.\cdots.R_l$ 是 r 中对象 t 在高阶属性 R_l 上的值($Domain(R_l)$型的元组对象集合)，$\sigma_F^2(r, R_l)$ 中的 F 是加在 $Domain(R_l)$ 中对象上的谓词，$t//\Delta$ 表示 t 的变形，即将 t 的高阶属性 R_l 的原来值改为 Δ，其它不变。

$\sigma_F^1(r)$ 是对完整对象的选择，$\sigma_F^2(r, R_l)$ 是对子对象的选择，即在高阶属性中的选择。如 $\sigma^2_{age>10}(person, children)$ 输出的每一对象"人"，其高阶属性 *children* 上的值只含年龄>10岁的子女。

投影运算 Π'

$\Pi_Q^k(r)' = \{t' \mid t \in r, t'[Q] = t[Q], t'[path] = path(t)\}$

其中 Q 为类 R 的顶层或内层嵌套属性集，t[Q]为 t 在 Q 上的值，Q 的典型结构为：

$Q_1 = A_{i1}, A_{i2}, \cdots, A_{ik}$(顶层)

$Q_2 = A_i.\cdots.B_j.C_m$(内层)

$Q_3 = A_i, A_j.\cdots.B_j.C_p$(混合式)

对 Q_1，投影运算是对一个类进行从顶到底的纵向分割；对 Q_2，投影运算是对一个类在内嵌的某层上的属性集上做投影；对 Q_3，投影运算是对一个类同时做上述二种投影。

投影结果中对象的对象标识如何确定是个重要问题。上式中的 t'[path]表示 t' 的标识路径，它与 t 的对象标识及 t 的高阶属性值中的对象标识有关，用 path(t)表示。t' 的标识路径可以有多种不同的表达方式，可以表达不同的查询语义，保留不同的嵌套结构信息，较为有用的有下列三种：

①全标识路径：如 Q_2，投影对象的标识路径包括标识 $A_i\text{-id}, \cdots, B_j\text{-id}$，若 C_m 为高阶属性，还包括 $C_m\text{-id}$；

②部分标识路径的特例之一——顶层标识：如对 Q_2，投影对象的标识路径只取 $A_i\text{-id}$ 和 $B_j\text{-id}$(C_m 为零阶属性)或 $C_m\text{-id}$(C_m 为高阶属性)；

③部分标识路径的特例之二——最内层标识：如对 Q_2，投影对象的标识路径只取 $B_j\text{-id}$(C_m 为零阶属性)或 $C_m\text{-id}$(C_m 为高阶属性)。

投影对象的对象标识就是标识路径中的最外层的那个对象标识，对②和③，隐含用于 C_m 中对象消重的并(∪)运算。

投影对象的标识路径中的各对象标识仍保持原来的嵌套形式。

例1 设有以下类定义

```
class Dept{
  D-NO:numer;
  D-name:string;
  D-project:project;
}
class Project{
  P-NO:number;
  P-name:string;
  members:{Emp};
}
class Emp{
  E-NO:number;
  E-name:string;
}
```

并设 Dept 中有若干复杂对象，并以嵌套形式表达，示于图1。这里没有单列出 Project 和 Emp 中的对象。事实上 P_1-P_3 是 Project 中对象，E_1-E_5 是 Emp 中的对象，没有列出这二个对象集，只是为了节省篇幅。

为叙述方便，将对应于上述标识路径情形①②③的三种投影分别叫做 Π^1、Π^2、Π^3。

设投影中取 Q = Dept. D-project. members. E-name，则取前述三种不同方式的标识路径所得结果示于图2—4。

上面保留不同嵌套信息的三个对象集，分别表达查询语义：①列出每个系所承担每个项目的参加人员；②列出每个系参加科研项目的人员(不关心参加那个项目)；③列出各系参加科研项目的人员(不

Did	D-NO	D-name	D-project					
			Pid	P-NO	P-name	members		
						Eid	E-NO	E-name
D_1	001	Comp	P_1	101	DB	E_1	201	Davie
						E_2	202	Smith
			P_2	102	PRG	E_2	202	Smith
						E_3	203	John
D_2	002	Math	P_3	103	Fun	E_4	204	James
						E_5	205	Anne

图 1 dept 复杂对象集

关心那个系，参加那个项目）。

D_1	P_1	E_1	Davie
		E_2	Smith
	P_2	E_3	Smith
		E_4	John
D_2	P_3	E_5	Jemes
		E_6	Anna

图2 保留全标识路径 $\Pi^1_{Q_1}$(dept)

D_1	E_1	Davie
	E_2	Smith
	E_3	John
D_2	E_4	Jemes
	E_5	Anna

图3 保留顶层标识 $\Pi^2_{Q_1}$(dept)

E_1	Davie
E_2	Smith
E_3	John
E_4	Jemes
E_5	Anna

图4 只保留最内层标识 $\Pi^3_{Q_1}$(dept)

D_1	Comp	P_1	DB	E_1	Davie
				E_2	Smith
		P_2	PRG	E_2	Smith
				E_3	John
D_2	Math	P_3	FUN	E_4	Jemes
				E_5	Anna

图5 $\Pi^2_{Q'}$(dept)

若将 Q 改 Q' = (Dept. D-name, Dept. D-project. P-name, Dept. D-project. members. E-name)，则 $\Pi^3_{Q'}$(d)与 $\Pi^3_{Q_1}$(d)相同，结果如图5所示。

上述基于三种标识路径的投影，形式②最重要，因顶层对象标识所标识的对象是查询所关心的最基本的对象，而且属性继承也是顶层概念。

形式③在文[2]中叫做 image 运算，在将 O-O 语言持久化而得的 O-O 数据模型中，对象的型与对象集是分离的，如例1中，可以只有类 Emp 的型定义，而没有 Emp 的对象集(collection 类)。复杂对象按 Dept 聚集，通过 Dept 而间接引入 Emp 对象并嵌套于 Dept 的对象中[8]。此时，引入 image(即 Π^3)是必要的，以获取 Emp 型对象集。在本文所用对象-关系模型中，类的概念包括类的型和类中对象集二部分。只要 Dept 中存在一个对象，其内嵌高阶属性 member 不空，一定存在 Emp 对象集。复杂对象的高阶属性值是通过引用高阶属性域中对象标识而建立(指针或 Join index)，Π^3运算可由后面讲到的$\bowtie$代之，因此，本代数中 Π^3(即 image 运算)是冗余的。

Π^1运算主要在建立复杂对象模型与平面关系模型之间联系中发挥作用，此问题在第3节论述。故本代数投影运算主要为 Π^2。

连接运算$\bowtie^1$

对嵌套的复杂结构来说，投影为"拆卸"，而连接为"组装"，连接运算分为以下几种情况：

$r_1 \bowtie^1 r_2 = \{t \mid t_1 \in r_1, t_2 \in r_2, t[R_1] = t_1[R_1], t[R_2] = t_2[R_2], t[id] = t_1[id] = t_2[id]\}$；

$r_1(J-R_2) \underset{F}{\bowtie^2} r_2 = \{t \mid t_1 \in r_1, t[R_1] = t_1[R_1], t[J-R_2] = \{t_2 \mid t_2 \in r_2, F(t_1, t_2)\} \neq \varnothing, t[id] = t_1[id]\}$。

$\bowtie^1$用于 r_1 和 r_2 为同一类层次中的类的情况，即 r_1 和 r_2 的对应对象描述同一实体，有相同的 e-id。其连接条件是基于具有相同的实体对象标识 e-id。由标识的唯一性假设，满足条件的对象(被连接对象)是1-1的。特别，$\bowtie^1$运算可将由 Π^2基于 Q_1 所得的二个类连接起来，恢复原状。关于$\bowtie^1$，在文[5]中已有论述，在那里还论述了$\bowtie^{11}$在属性继承中的应用。

$\bowtie^2$用于 r_1 和 r_2 的对应对象(被连对象)是1-m的情况，通常描述不同类型的实体。连接结果是类 R_2 作为类 R_1 的附加高阶属性 $J—R_2$ 的域，该高阶属性的值由条件 F 决定。$\bowtie^2$运算的例子在第3节会看到。

按说，还应有另一种连接运算，可以使 r_2 和 r_1 的任一内嵌高阶属性相连接。但这种运算的语义复杂，运算本身复杂，且改变了高阶属性的域。仅管可以参照$\bowtie^2$定义这种运算，但本文不予定义。

其它运算，如嵌套(nest)，反嵌套(unnest)，在第3节稍加讨论。并运算∪没有什么特殊之处，不再赘述。

顺便提一下，Π^3运算是冗余的。在例1中，$\Pi^3_{Q_1}$(dept)，Q = Dept. D-project，则此运算相当于 σ^1(project)，F = (self. project in Dept. D-project)，这里隐含有连接运算，也就是说 $\Pi^3_{Q_1}$(dept) = $\Pi^3_{Q_1}$(project $\underset{F}{\bowtie^2}$ dept)，F = (self. project in Dept. D-project)，Q' = (self. P-NO, self. P-name, self. members)。

3. 讨 论

①对象标识和嵌套结构。代数运算结果的对象标识的处理是困扰许多研究者的一个问题，直接影响到运算结果的嵌套化程度，也影响到对运算结果的语义解释。

如前所述，本文的讨论是基于文[4]中的全标识<e-id, c-id, i-id>。前述各种运算中的 id 指文[4]中的实体标识 e-id，只依赖于所描述的实体，而与型和存储空间位置无关。如果将运算结果存起来，除 σ^1 运算保持 e-id 和 c-id 不变外，其它运算(Π^1 和$\bowtie^{1*}$)只保持 e-id 不变(Π^2)，c-id 和 i-id 一般都会变。σ^1运算较简单，下面重点分析 Π^3和$\bowtie^{1*}$运算。

Π^3运算同文[2]中的 image，不赘述。Π^2运算结果的对象标识保留原对象的对象标识(顶层 e-id)，且保留了原嵌套关系(不一定原样)，结果是完全嵌套的。如，对例1，$\Pi^2_{Q_1}$(dept)，Q' = (D-name, D-project. P-name, D-project. members. E-name)，则 D-name, P-name, E-name 不在同一层上，E-name 在 P-name

所在层的内层，其层次关系为（D-id，D-name，{P-id，P-name，{E-id，E-name}}），如图5所示。按文[2]，投影的结果，E-name 和 P-name 在同一层上。

Π^1运算结果中的对象保留了标识全路径，完全保留了原来的嵌套结构，也保留了原来的对象标识 e-id（标识路径中的顶层对象标识，或叫第一个对象标识）。

连接运算结果中的对象标识是最困惑许多研究者的问题之一。有的认为运算结果的型为 n＋m tuple，即 n＋m 个属性的元组型，或 oid 对，并需另外附加新的对象标识[2,6]。这种情况下的运算结果中的对象往往是嵌套-平面结构（当 r_1和 r_2中的对象是1-m 关系时）。文[7]中的 extend 运算类似于本文的$\bowtie^2$运算，都认为结果为 n＋1 tuple。该文的作者声称保对象，但其对象标识语义不清，会造成实现上的困难；extend 的运算结果相当于生成了 R_1的子类，那么一个类中的一个对象移到其子类时，其对象标识改变否？本文引入的$\bowtie^1$和$\bowtie^2$运算保原对象标识（e-id），且结果对象是完全嵌套的。其语义为连接结果为 n＋1 tuple，是对 r_1中的对象所描述的实体对象的不同描述。通常，连接运算是动态地建立对象间联系的一种手段。R_2是 r_1的由于连接而附加的高阶属性 J-R_2的域，而高阶属性正是复杂对象中表达对象间联系的一种手段，二者是一致的。也就是说，连接运算建立了 r_1和 r_2中对象间的一种联系，且用高阶属性表之。

关于$\bowtie^1$运算作用在第二节已述，且其运算结果也是完全嵌套的，保 r_1中对象的对象标识。

②对复杂对象的描述能力。复杂对象代数不仅应能在保持整对象结构的条件下处理复杂对象，还应能改变嵌套结构，并保持完全嵌套形式。

Π^1运算可以用来拆卸复杂对象成所希望的形式，并保持完全嵌格式。$\bowtie^2$可将不同的完全嵌套结构的对象再嵌套，并保持完全嵌套格式。

按照不同的顺序连续使用$\bowtie^2$运算，可得到不同的嵌套结构。如 $r_1\bowtie^2(r_2\bowtie^2 r_3)$，使得 R_1、R_2、R_3在不同嵌套层次上，R_3在最内层；而 $r_1\bowtie^2 r_2\bowtie^2 r_3$使得 R_2、R_3在同一层次上，同为 r_1的附加高阶属性的域。

Π^2和$\triangleright^2$还可以用来逆换嵌套结构，这对以不同方式表达 n-m 联系，满足不同的查询语义，很有意义。

例2　对例1给定的关定义及对象集，设 D_2中的 P_1改为 P_2，则 Π_{S}(project $\bowtie^2$ dept)，F = (Project-self in Dept. D-project)，Q = (self. P-NO，self. P-name，self. J-Dept. D-NO，Self. J-Dept. D-name，self. J-Dept. D-project. members)，其结果是将例1中的 Dept 和 Project 的嵌套互换，如图6。

P_1	101	DB		D_1	001	comp	E_1	201	Davie
							E_2	202	smith
							E_3	203	John
P_2	102	DB		D_1	001	comp	E_1	201	Davie
							E_2	202	smith
							E_3	203	John
				D_2	002	Math	E_2	202	Davie
							E_4	204	John

图6　例1中 Dept 和 Project 嵌套互换

③与平面关系代数的兼容和转换。将对象标识看做对象的一个属性，当复杂对象退化成平面对象时，本文的 σ，Π^1，$\bowtie^1$运算仍适用，且等效于平面关系代数的相应运算。不同点在于必须包含对象标识这一属性。

$\bowtie^2$的运算结果为嵌套形式，即使复杂对象退化成平面对象时，$\bowtie^2$的运算结果仍为嵌套格式；而平面关系代数的运算结果永远为平面格式。运算之后，立刻进行 unnest（对象标识也参加此运算一见下面论述），可使$\bowtie^2$之结果成为平面格式。此时，可能出现重复的对象标识，但对象标识已失去其语义，只是语法上存在而已，仍可将 r_1的对象标识在语法上做为结果的对象标识（只是名为 Oid 的一个属性）。没有高阶属性（注意，此时参加运算的对象都已退化成平面对象），σ，Π^1，$\bowtie^1$语法上仍可执行，由此做到与平面关系代数的兼容。

通常复杂对象代数的运算结果可能是嵌套-平面混合格式，为了进行嵌套和平面结构间的相互转换，常引入嵌套（nest）和反嵌套（unnest）运算。本文的复杂对象代数是基于完全嵌套结构的，运算结果是完全嵌套的，故就复杂对象运算而言，可以不引入这二个运算。

为了支持复杂对象代数与平面关系代数的相互转换，需要引入 nest 和 unnest 运算。这里，此二运算基本上同 NF²关系模型中的 ν 和 μ 运算[1]，只不过标识路径中的对象标识也象普通属性一样参加运算。因 unuest 运算是为了转入平面关系代数，将其结果转为元组集合，不再是对象集合，故对象标识概念消失，尽管对象标识符号本身仍存在。因此 unnest 应连续使用，使结果完全平面化。

unnest 是针对高阶属性进行的，这容易由系统

自动判断并执行之，直至完全平面化。我们将连续使用 μ 运算直至完全平面化的运算叫做平面化运算(Flatten)，nest 运算也应连续执行，直至完全嵌套化。但此过程难以由系统自动完成，需人工干与。但第一次使用 nest，至少应对一个对象标识进行，以使此对象标识消重，变成顶层对象标识，即嵌套化后形成的对象的对象标识不再是纯元组)。这里 Π' 运算起重要作用。

例3 对例1，运算

Flatten(Π_{D}(dept)),Q=(Dept. D-NO,Dept. D-name);

Flatten(Π_{D}(dept)),Q=(Dept. D-project. P-NO,Dept. D-project. P-name);

Flatten(Π_{D}(dept)),Q=(Dept. D-Project. members. E-NO,Dept. D-project. members. E-name);

这三个运算得到三个平面关系，其第三个的结果示于图7，其它二个省略之。

D_1	P_1	E_1	201	Davie
D_1	P_1	E_2	202	Smith
D_1	P_2	E_2	202	Smith
D_1	P_2	E_3	203	John
D_2	P_3	E_3	204	Jemes
D_2	P_3	E_1	205	Anna

图7 Flatten 之例

此结果与文[3]得到的平面关系相同，后者不是通过代数运算得到，而只做了直观解释。

对 Π' 运算结果平面化所得到的关系中含有标识路径信息，便于和复杂对象代数中的路径表达式相对应，有利于相互转换。同时，标识路径中的标识实质上就是外关键字，保留了复杂对象所表达的实体间的联系，这种平面关系，相当于对应于联系实体的关系，并且，所得平面关系的规范化程度较高。这些都优于文[1]中的 μ 和文[2]中的 unnest。当然，平面化的方式不是唯一的。

参考文献

[1]Roth M. A. et al. ,Extended algebra and calculus for nested relational databases,ACM TODS,13(4),1988

[2]Shaw G. M. et al. ,An object-oriented query algebra,Proc. 2nd Int. Workshop on database programming language,1989

[3]Meng W. et al. ,Construction of a relational front-end for object-oriented database systems, IEEE Data Engineering Conference 1993

[4]李天柱，对象标识的语义及构造，计算机科学，20(5)，1993

[5]李天柱，NF² 关系模型与属性继承，软件学报，6(4)，1995

[6]Kim W. ,A model of queries for object-oriented databases,VLDB,1989

[7]Scholl M. H. et al. ,A relational object model, Proc. 3rd Int. Conf. on Database Theory,1990

[8]Kim W. ,Introduction to object-oriented databases,The TIM Press,1990

(上接第4)

哪行，包含它的源程序代码等信息，另外用户也可以通过点取交叉引用关系中的单元名，将它的下一级交叉引用关系显现出来；显示的另一种方式是一次就把整个软件系统的交叉引用关系从根结点到叶子结点全部显示给用户，它包含调用的单元名以及 WITH 的过程名、包名，USE 的包名。同样，用户也可以通过鼠标的点取操作来显示上一种方式所显示的那些信息。另外，用户还可以用不同的颜色把交叉引用关系图中的全部过程名及其它们之间的调用关系或各任务名及其它们之间的通讯关系突出显示出来，供用户对顺序执行的程序以及并行执行的程序进行分析、理解。

所产生的交叉引用表及未被使用过的单元名表(如果存在)也是以文件形式存于信息库中的。

ASCA 首先是在 VAX 机上开发的，现在已移植到 ALPHA 机上，有关的界面及图形将用 Motif 实现。我们力求为用户提供一个用户界面友好、操作简单方便、能真正帮助用户分析、理解大型 Ada 软件项目的工具，相信它会为提高 Ada 软件系统的开发效率、保证其可靠性发挥不小的作用。(参考文献共5篇略)