面向对象 查询 隐式回答 (9)

45-50

计算机科学Vol.20No.4 1993

面向对象查询的完整隐式回答

Hava T. Siegelmann 和 B.R.Badrinath

TP311.13

Sieg, HT ⓓ 冯铃

摘 要

根据类公式化OODB中的查询，可检索某些满足类属性谓词的实例。OO数据模型的类层次概念允许依据类和实例，隐式地表达具有不同抽象级的查询结果。Shum和Munta〔SM88〕提出了基于DB分类的隐式表达方法，其算法在查询结果长度上是最佳的，但不够清晰。本文提出一种易理解的唯一相关回答。为了以自顶向下方式有效地获得并操纵相关回答，借助访问DB的查询算子，公式化用户查询并对标准查询算子进行重定义，这种相关回答的抽象表示，对处理复杂信息十分有用。

一、引言

对于支持复杂的数据密集型应用，当前工作集中在如何为OO数据模型的查询提供方法和手段。大多数方案以扩展的关系框架为基础，支持各种OO概念。典型地，一个查询的结果是满足特定谓词的对象集，既不能提供一个丰富的表达技术，又非一个表达复杂信息的有效方法。例如，考虑一个大型公司中人事数据库及其查询：

SELECT all employees with a salary greater than 30k

通常的结果由salary大于30k的employee对象集构成。如果有许多employee，其salary大于30k，则所有的engineer、manager和两个secretary的salary也大于30k。所以一种更精巧的回答为：

engineers+managers+objid (secretary1)+objid(secretary2)

上述结果称为隐式表达式或隐式回答。engineer与manager是先定义好的类，我们称常规回答中的对象（用object ids表示）为实例，实例组（如engineer和manager）为类。隐式回答的优点在于：

- 不必列出所有的实例；
- 对于复杂信息，用户未必对结果的详细情形感兴趣，因此，隐式回答能够允许在更高的抽象级上交换信息；
- 是决策支持系统的有用功能；
- 隐式回答能帮助用户获得关于回答的全部理解；
- 低抽象级上的修改并不与高抽象级上的检索相冲突，提高了并行度。

二、相关的工作

〔SM88〕在概念分类的数据库上，发展了隐式表达式的概念。隐式回答使用了预先定义的类名和实例名。具有最少数目的类名或对象的隐式回答称为优化表达式。我们使用了〔SM88〕中定义的隐式表达式模型，但重新定义了称为i-相关表达式的隐式表达式概念，i是表达式的简单参数，由用户选定。同时，我们还发展了基本的查询算子集，以提供相关回答。

〔KP90，Deu90，SZ90，SZ89，BKK88〕提出了基于扩展关系框架的面向对象查询语言。〔SZ89，SZ90〕描述了关系代数与OO概念相结合的查询代数。〔SZ89〕提出返回指定类型对象的嵌套运算与函数算子。〔KP90〕中的模型支持嵌套关系，其元组的属性既可是原子，又可是关系。嵌套关系代数能提供更强的表达能力，处理层次结构。

三、相关回答

我们旨在依据继承性提供关于类和实例的隐式表达式，而非单一的抽象级上的回答，如：完全的实例列。我们定义对象的有限集为$D=\{D_1, D_2, \cdots, \}$，$C=\{C_1, C_2, \cdots\}$是与D相关的类集，实体可以是一个实例，也可以是一个类。我们只处理有限的树状分类，其节点为实例或类，实例是叶子。包括根在内的内部节点为类。每个后续节点均包含在父母节点内。任一个非叶节点的后续节点总和等价于其双亲的类，同层的亲属互相排斥。实例集A可用T来分类，当且仅当T的根包含A。

图1是一个vehicle类，其中椭圆表示类，方框表示实例，所有实例都含属性year、color、owner和insure，属性域是字符串或整数，实例旁的标记为属性值，类旁的标记为其值约束。

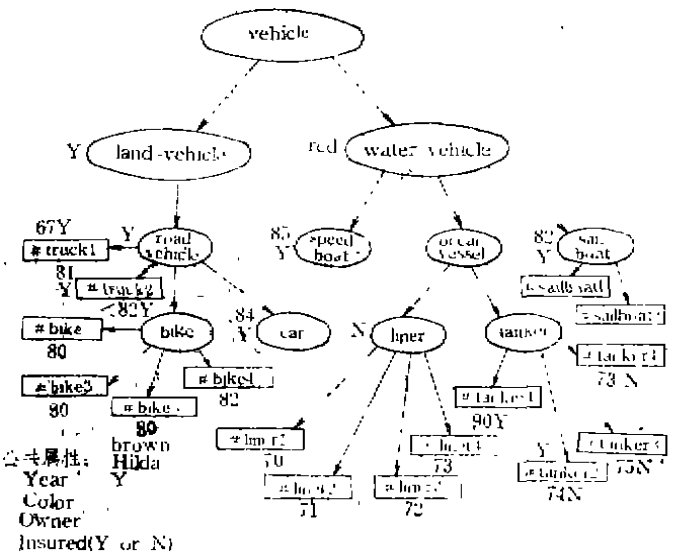


图1 具有属性值的vehicle模式

3.1 隐式表达式

下面列出了图1中包含类和实例的回答。

例1 找出所有1989年之前制造的water-vehicle，隐式回答为：

1. water-vehicle－#tanker1
2. ocean-vessel+sail-boat+speed-boat－#tanker1
3. (water-vehicle－ocean-vessel) +liner+(tanker－#tanker1)

由于上述表达式所代表的实例集既难以推演，又难以理解，因此，需要一种对用户是清晰的隐式表达式标记法，为此定义一些术语和概念。

定义1 关于分类T上的表达式中用到的字母由如下几部分构成：

- 类：C_1，C_2，…；
- 实例：D_1，D_2，…；
- 空：ε；空表达式；
- 符号：+，－；“+”表示包含，“－”表示不包含
- 括号：（ ）；

定义2 T上的表达式如下递归定义：

- 实体（实例或类）是一个表达式；
- ε是一个表达式；
- 若e是表达式，则（e）也是；
- 若e_1，e_2是表达式，则e_1+e_2，e_1-e_2也是。

定义3 T上的表达式代表的含义为实例集（0或多个），用meaning (exp) 表示，两个表达式e_1、e_2是等价的，若meaning (e_1) =meaning (e_2)。

定义4 分类上的一条路径P是始于根，终止于叶节点的连续节点集。一个关于P路径的表达式的子表达式是将表达式中所有不在P上的实体去掉后的结果。

定义5 表达式的长度是其实体的个数，表达式的大小是它所代表的实例个数。

3.2 相关表达式的定义和例子

为了便于理解，我们为隐式表达式规定一些限制，这些表达式称为相关表达式。我们分两步构造相关表达式：首先，引入简单表达式概念，其次，定义相关表达式。这两种表达式都是具有特定限制的隐式表达式，它们包含两种类型的项。

定义6 项是实体序列，有两种类型：

a. $(C_i—D_{i1}—\cdots—D_{il})$ $D_{ij}\in C_i$ 且 $l\geqslant 0$，称为概念项C_{term}；

b. D_k，称为单个项I_{term}或一个实例。

定义7 简单表达式是一个隐式表达式，由概念项和单个项构成，有如下限制：

a. 表达式中一个实体只可出现一次；

b. 对于每条路径，它的子表达式最多包含一个类；

c. 对任意I_{term}和C_{term}，$I_{term}\in C_{term}$。若$I_{term}\in$expression$\Longrightarrow C_{term}\notin$expression。

d. 分类中实体按从左到右的顺序出现（唯一性）。

例2 找出所有在1971～1975年间制造的ocean-vessels，该查询结果的等价简单表达式为：

1. #liner2＋#liner3＋#liner4＋#tanker2＋#tanker3＋#tanker4
2. (ocean-vessel—#liner1—#tanker1)
3. (liner—#liner1)＋(tanker—#tanker1)

定义8 相关表达式是具有如下限制的简单表达式：

a. 不包括类C_1，C_2，…，C_k，$k>1$，对于T的某一类C_j，有 meaning $(C_1+C_2+\cdots+C_k)$ =meaning (C_j)。

b. 没有概念项包含C_j，其中有C_j的祖先C_i，C_i与C_j的含义等价。

c. 不包含C_{term}，其中有一个比它短的等价的实例序列。

d. 不包含一个实例序列，其中有一个比它们短的等价的C_{term}。

例3 相关表达式的例子。

1. (ocean-vessel—#liner3)＋(speedboat)＋(sail-boat) 不是一个相关表达式，（违反了a）。
2. (water-vehicle—#liner3) 是一个相关表达式。
3. (road-vehicle—#bike2) 不是一个相关表达式，违反了b。
4. (land-vehicle—#bike2) 是一个相关表达式。
5. (tanker—#tanker3—#tanker4)不是一个相关表达式，违反了c。
6. #tanker1＋#tanker2是一个相关表达式。
7. #tanker1＋#tanker2＋#tanker3不是一个相关表达式，违反了d。
8. (tanker—#tanker4) 是一个相关表达式。

定义9 一个i-相关表达式是一个相关表达式，其C_{term}中负实例数均不超过i，但至少有一个C_{term}，它具有i个负实例。

例4 找出所有1972～1975年间制造的ocean-vessel，等价的i-相关表达式为：

1. (ocean-vessel—#liner1—#liner2—#tanker1) 3-相关表达式
2. #liner3＋#liner4＋(tanker—#tanker1) 1-相关表达式

3.3 i-相关表达式的特点

本节证明i-相关表达式的唯一性，即如果e_1和e_2是i-相关表达式，meaning (e_1)与meaning (e_2)等价，则$e_1=e_2$。一个i-相关表达式可通过自顶向下方式计算得到。

定理1 一个i-相关表达式是唯一的。

证明：略

对于一个给定的查询，获得i-相关表达式分两步。第一步，找出查询的正、负实例个数，通过从根开始的深度优先搜索完成，以决定满足查询条件的实例；第二步，描述算法，在树中的每一个节点上，或返回一个项，或递归进行，最终的结果就是项的序列。

0. current-node←the root。

1. if current-node是一个叶子（实例），若它满足查询条件，则返回实例，否则返回e。

2. if current-node仅有一个孩子，current-node's children←grandchildren。

3. 若current-node下的所有实例不满足查询条件，返回e。

4. 假定current-node下的p个实例满足查询条件，n个实例不满足查询条件，若$n\leqslant i$（i-概念项或更少），且$p>(n+1)$（概念项比等价的实例序列短），返回 (current-node—D_{j1}—D_{j2}—…—D_{jn})。

5. 若$n>i$或$p\leqslant(n+1)$，将算法运用

到current-node的所有孩子上。

下面我们证明上述算法的正确性。

定理2 获得一个i-相关表达式分两个步骤，第一阶段花费时间O(d+c)，d为实例的个数，c为类的个数，第二阶段花费时间O(hA)，h为分类的高度，A为回答集的大小。

证明：证明在于算法的正确性和复杂性。

正确性证明：略

引理1 若过程持续给current-node的儿子，因为$p\leqslant(n+1)$，$(n\leqslant i)$，最终回答中，没有类C_1，C_2，…，C_l集合，$l>1$，其中meaning$(C_1+C_2+\cdots+C_l)$与meaning(current-node)等价。

证明：略。

引理2 若处理过程持续给current-node的儿子（当$p\leqslant(n+1)$，$n\leqslant i$），S_k均是类，则它们并不全部包含在最终答案中。

证明：略。

复杂性：算法的第一阶段花费时间O(d+c)，第二阶段，算法仅扫描可出现在最终答案中的路径p上的实体，因而第二阶段的复杂性为O(h*A)，h是T的高度，A是答案所代表的实例个数。

四、查询代数

3.3节描述的算法既可用于根类，又可用于其它的类，一个类所代表的实例集称为目标集，目标集既可以作为一个类名字，又可作为隐式相关表达式给出，一个类名字是相关表达式的特殊情况，因而我们将查询算子定义在相关表达式而非实例集上。下面我们定义了一个基本的算子集，包含有select、两类project及两类join。

4.1 select算子

select算子既可以定义在目标集上，又可以定义在相关表达式上，它返回一个满足选择谓词P的实例集。

1．定义在目标集上的select，select(S，P)由i-相关算法计算得出。

2．定义在相关表达式上的select，select(e，P)如下计算：

$e=term_1+term_2+\cdots+term_i$，$term_i\in\{I_{term},\ C_{term}\}$

$select(e,\ p)=\bigcup_{i=1}^{i}select(term_i)$，$term_i\in e$

·若$term_i$是I_{term}，结果可以是实例本身，若它满足P，否则为e。

·若$term_i$是$C_{term}=(C-D_1-\cdots-D_k)$，将C的实例看成是更小目标集S(C)的成员，以导出结果，假定C_{term}的负实例不满足谓词P。

4.2 project算子

对于每一个将查询的实例，project算子返回属性的子集值。定义在i-相关表达式上的project算子具有形式：project(e，A_1，A_2，…，A_k)，属性A_1，A_2，…，A_k是目标集S的属性的子集。我们定义两种类型的project：s-project和t-project。前者的结果是元组集，每一个元组与查询出的e的一个实例的属性相匹配，它是基本的project；t-project的结果是一个元组，元组中每一个元素由e实例的属性的所有可能出现值构成，当指定类的属性值约束时，t-project的定义就可被采用到分类层上。

4.2.1 t-project

t-project(e，$\langle A_1,\ A_2,\ \cdots,\ A_k\rangle$)$=\langle a_1,\ a_2,\ \cdots,\ a_k\rangle$，$a_i$是表达式中$A_i$的所有取值的集合。

t-project计算如下：

$e=term_1+term_2+\cdots+term_i$

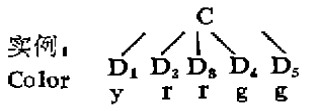
t-project(e，$\langle A_1,\ A_2,\ \cdots,A_k\rangle$)$=\biguplus_{i=1}^{i}$(t-project($term_i$，$\langle A_1,\ A_2,\ \cdots,\ A_k\rangle$))

其中$\biguplus_{i=1}^{3}(\langle a_1,\ a_2,\ a_3\rangle,\ \langle b_1,\ b_2,\ b_3\rangle)=\langle\{a_1,\ b_1\},\ \{a_2,\ b_2\},\ \{a_3,\ b_3\}\rangle$

1．如果$term_i=I_{term}$（一个实例），t-project(e，$\langle A_1,\ A_2,\ \cdots,\ A_k\rangle$)$=\langle D_{A1},\ D_{A2},\ \cdots,D_{Ak}\rangle$，$D_{Ai}$是实例D的属性$A_i$值。

2．如果$term_i$是$C_{term}(C-D_1-\cdots-D_m)$，我们不能用t-project(C)或t-project(C)—t-project($D_1+D_2+\cdots+D_m$)代替它，考虑下

列例子：

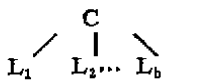


t-project((C−D_1−D_2),color)= ‹{red, green}›

t-project(C, color)= ‹{yellow, red, green}›

t-project(C,color)−t-project(D_1+D_2, color)= ‹green›

因此，我们必须递归地处理t-project：



L_i可以是类或实例

t-project((C−D_1−…−D_m), ‹A_1, A_2, …, A_h›)

$=\biguplus_{i=1}^{k}$(t-project (E_i, ‹A_1, A_2, …, A_h›))

E_i是表达式$L_i \setminus \{D_1+D_2+\cdots+D_m\}$, L_i是C的儿子。

3．若$term_i$是类C_i属性A_1, A_2, …, A_h是目标集S的实例的属性集子集，但是类C可能仅对这些属性的子集的值有约束制约，我们分别检查每一个属性，然后，将此值插入一个元组中。

t-project(C, ‹A_1, A_2, …, A_h›)$=\Diamond_{i=1}^{k}$ t-project(C, A_i)

(从k个值形成一个元组)

$$\text{t-project}(C, A_1)=\begin{cases} C.A & \text{若}A_1\text{对C有一个约束} \\ \bigcup_{i=1}^{k}\text{t-project}(L_i, A) & \text{否则} \end{cases}$$

4.2.2 s-project

该算子与标准的project定义类似：

s-project(e, ‹A_1, A_2, …, A_h›)={‹a_1, a_2, …, a_h›}

每一个元组由e实例的属性A_1, A_2, …, A_h值构成。

4.3 join算子

join算子返回一个新实例序列，它为实例A和B的笛卡尔积，我们定义两类join：d-join是目标集A和B间的联接，它们在叶子级上有不同的属性；s-join是特殊情况，所有实例的属性来自同一属性集合。

$e_1=term_{1,1}+term_{1,2}+\cdots+term_{1,n}$

$e_2=term_{2,1}+term_{2,2}+\cdots+term_{2,m}$

$terms\in\{C_{term}, I_{term}\}$

$join(e_1, e_2)=\bigcup_{i=1}^{n}\bigcup_{j=1}^{m} join(term_{1,i}, term_{2,j})$

不同项上join概念依赖于join类型。

4.3.1 s-join

s-join是两组实例的交集，各用一个相关表达式加以表示。

1．若有一个项是实例：

$$\text{s-join}(X, D_2)=\begin{cases} D_2 & \text{若}X=D_2\text{或}D_2\in X \\ e & \text{否则} \end{cases}$$

2．两者均为类：

$$\text{s-join}(C_1, C_2)=\begin{cases} C_1 & \text{if } C_1\subset C_2 \\ C_2 & \text{if } C_2\subset C_1 \\ e & \text{否则} \end{cases}$$

3．两者均为概念项（并不都是类）：

s-join((${C_1}-{D_{1,1}}-{D_{1,2}}-\cdots-{D_{1,h}}$), (${C_2}-{D_{2,1}}-{D_{2,2}}-\cdots-{D_{2,h}}$))

$=$s-join(C_1, C_2)−〔$D_{1,1}+D_{1,2}+\cdots+D_{1,h}+D_{2,1}+D_{2,2}+\cdots+D_{2,h}$‖s-join($C_1$, C_2)〕

“‖”表示受限运算，即仅对s-join代表的实例集实行该种操作。

4.3.2 d-join

d-join与一个自然联接相似，相关表达式代表的A、B实例组均由目标集推导出来。这些目标集具有不同的属性，图2为一分类，根是动物类，D_1是动物类的实例，D_2、D_3是vehicle目标类的实例。

1．两个项均为实例：d-join与通常的join相同，例如：

D_1=‹Tail:8 in, color: white, Friendly

50－54 时态数据库 数据库 时态关系代数 (10)

计算机科学 Vol.20 No.1 1993

一个时态关系代数

张师超

TP311.13

（国家智能计算机研究开发中心 广西师范大学数学系 541004桂林）

摘 要

In the current temporal relational models, it leads to inadequate of the operations that the temporal relational operations act temporal data as well as static data except relational union. This paper sets up a N1NF temporal relational algebra. Its operations are sufficient for the needs of varied applications.

1. 引言

随着越来越多的私人和政府部门等需要保存和查看过去的记录（或数据），时态数据库已成为一个活跃的研究领域。据统计，近十多年来，已有400余篇有关时间信息处理的文章发表。这些研究主要集中在：（1）时态数据模型；（2）时态询问语言；（3）存贮结构和实现技术。

由于时间具有一些特有的性质，它不象一般数据那样易于在数据库中模拟。所以，许多研究人员认为，有必要从零开始建立时态数据库的理论与基础[2]。L. E. Mckenzie和R. T. Snodgrass[1]用26种评价标准测试了12个典型的时态数据库后，其结果与上面一致。

从现有的时态关系模型看，它仅仅是传统的关系模型的一个简单扩展。除关系并外，其它的关系操作仍不能体现时态的特性，这导致时态操作的不充分性。本文建立一个

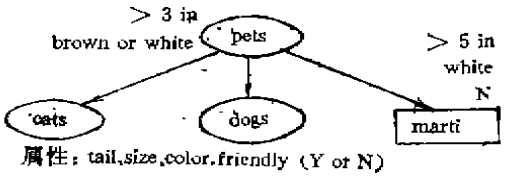


图2 动物模式

:N〉

D_2= 〈Year:1988, Insured:Y, Color: white, Owner:john〉

D_3= 〈Year:1990, Insured:Y, Color:yellow, Owner:john〉

d-join(D_1, D_2)= 〈Tail:8 in, Color: white, Friendly:N, Year:1988, Insured:Y, Owner: john〉

d-join(D_1, D_3)=ε

2. 有一个项是实例：分解为

d-join((C_1—$D_{1,1}$—$D_{1,2}$—…—$D_{1,k}$), D_2)=d-join(C_1, D_2)—〔d-join($D_{1,1}$, D_2)+…+d-join($D_{1,2}$, D_2)+…+d-join($D_{1,k}$, D_2)〕

3. 两个项均为具有缺省值的类，将项当成实例对待。

4. 两个项均为概念项：

d-join((C_1—$D_{1,1}$—$D_{1,2}$—…—$D_{1,k}$), (C_2—$D_{2,1}$—$D_{2,2}$—…—$D_{2,h}$))=d-join(C_1, C_2)—(d-join(C_1, $D_{2,1}$)+…+d-join(C_1, $D_{2,h}$))—(d-join(C_2, $D_{1,1}$)+…+d-join(C_2, $D_{1,k}$))

（参考文献略）

〔冯铃摘译自“Proceedings of the 17th Intl. Conf. on VLDB”，冯玉才校〕