

86-60

OODB 模型的形式定义和查询代数

丁宝康 陈彤兵

(复旦大学计算机系 上海 200433)

TP311.13

摘 要 This paper introduces the concept of object inheritance apart from the concept of class inheritance in the OODB model. We then introduce a formalization and query algebra of the OODB model. The query deals with homogeneous sets of object, and is closure and safety.

近十年来,面向对象数据库(OODB)的研究已越来越深入,对于类、对象标识、类继承性、查询操作等的研究也很多。本文介绍 OODB 中一个新的概念——对象继承性,这个概念发展了 OODB 类继承性概念,使 OODB 的形式定义和查询代数发生变化,而且更符合实际和易于存储操作。

一、“面向值”与“面向对象”

在关系模型中,实体类型用关系模式表示,实体集用关系表示,由用户为关系模式定义“主键”,实体是用主键值标识的,而且,关系模型中严格要求模式具有第一范式性质,即不允许属性值具有复合性质,例如元组或关系。所以关系模型又称为平面关系模型,如果允许关系的属性值又可以是关系,即允许一个关系类型作为另一个关系类型的构造成份,那么这种模型称为嵌套关系模型。如果进一步放宽在关系的定义层次上集合与元组必须严格交替出现的限制,那么就得到了复合对象模型。然而,这三种模型在本质上没有多大区别,都是“面向值”的。这种“面向值”的方法,在技术上存在三个问题。一是更新异常问题,如果在关系中修改了某个元组的外键值,很可能会违反“引用完整性”规则。二是在关系中不允许存在重复元组,这样,具有同样值的两个元组就不可能存储。三是在处理关系模式属性集的子集时显得不灵活,没有一种简单表示这份子集的好办法。

然而,在面向对象数据库模型中基本上克服了上述三个缺点。面向对象数据库模型尽管与复合对象模型很类似,前者中对象的属性值可以是基本类型的值、对象、集合,类似于后者中基本类型的值、元组、集合,但是 OODB 采用了“面向对象”方法。实体类型用“类”表示,实体用“对象”表示,系统给数据库中每个对象赋予一个“对象标识”,起唯一标识作用。

对象标识和对象的值之间是分离的,为数据库进一步数据共享和减少冗余创造了条件。

二、对象继承性

OODB 的类之间可以有“is-a”联系(例如数据的特化或数据的概括),这种结构称为“类层次”。其类继承性分为“单继承性”和“多继承性”,后者类间联系组成了“类格(class lattice)结构”。

在系统中定义一个类 OBJECT,作为类层次或类格的根。OBJECT 是数据库中其它系统定义的类或用户定义的类的直接超类或间接超类。

对象继承性是类继承性的扩充。下面先举一个例子,然后再来看对象继承性的优点。

例 1 图 1 是一个 OODB 模型,类 PERSON 有两个子类 EMPLOYEE 和 STUDENT。

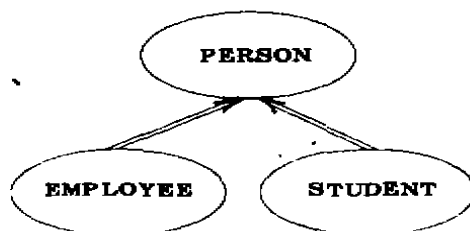


图 1

如果产生一个新的 employee 对象,那么系统会赋给该对象一个对象标识。然后把这个对象标识加入到 EMPLOYEE 对象集中。由于 EMPLOYEE 是 PERSON 的子类,因此任何 employee 对象也应是一个 person 对象。在把 employee 对象标识加入到 EMPLOYEE 对象集的同时,也要把这个对象标识加入到 PERSON 对象集中。显然,EMPLOYEE 对象集是 PERSON 对象集的子集。这种产生新对象的

丁宝康 副教授,研究方向:数据库理论和应用。陈彤兵 硕士,研究方向:面向对象数据库

方法是大多数 OODB 采用的方法。

本文采用另外一种方法。在产生新的 employee 对象时,系统自动在它的超类 PERSON 中也产生一个对象(另外一个对象标识)person,这个 person 对象称为 employee 对象的“超对象”,employee 对象为 person 对象的“子对象”。这种子对象与超对象间的联系称为“对象继承性”。

对象继承性可能使系统产生许多额外的标识,但是它有下列优点:

(1)扩充了“对象共享”的概念。例如某个人既是职工又是学生,那么这个人的 employee 对象和 student 对象具有同一个超对象 person,而 person 对象为两个子对象共享,而每个子对象不必各有一个超对象。这样的语义很确切,查询时也不会出现异常,同时也达到了“引用等价性”规则。

而在大多数 OODB 中则是另一种情况。在 student 对象和 employee 对象中虽具有同样的 person 对象的属性值,但是未说明它们的 person 是否是同一个对象。为了解决这个问题,在大多数 OODB 中都是再定义一个新的类 EMPLOYEE-STUDENT,它是类 EMPLOYEE 和类 STUDENT 的子类。这就在数据库中增加了许多类;而具有“对象继承性”的 OODB 可以少定义一些类。

(2)更有力地支持“引用完整性”规则。能避免“基于值”系统的更新异常。例如当一个人既是学生又是职工时,要修改 person 中的属性,只要在 person 对象中修改就行。但在传统的 OODB 中修改工作要在学生和职工两个对象中去做。因此“对象继承性”更有力地支持了“引用完整性”规则。

(3)删除操作的语义清楚。在大多数 OODB 中,如果要删除一个对象,而它又被其它对象引用,那么删除操作就有可能导致“悬挂引用”问题。一般采用“垃圾回收”机制,在一个对象被删时,只要存在对该对象的引用,这个对象就应保留存储空间。如果想删除一个对象,但又要保留它的超对象,那么在“垃圾回收”中就要从一个类中删除一个对象,而在其超类中保持信息,这里要做信息的迁移工作。

而在具有对象继承性的 OODB 中,删除操作就很简洁。例如要删除一个 employee 对象,不会影响它的超对象 person,因为这本来就是两个不同的对象。

(4)能确切定义封闭的面向对象查询代数。查询是在同质集上进行的,并提供投影和联接的确切有效的定义,把查询结果作为持久对象保存下来。

三、OODB 模型的形式定义

本节在数据库具有类继承性和对象继承性的情

况下对面向对象数据库模型进行形式定义。

1. 类的定义

(1)类定义为三元组〈状态,行为,继承〉。

(2)类的状态是形为 $[A_1 : a_1, \dots, A_n : a_n]$ 的元组类型,这里每个 A_i 是属性名, a_i 是属性值的类型。

属性分为原子属性和复合属性两类,原子属性是指基本数据类型的属性;复合属性或者是一个类,或者是元组,或者是一个集合。

(3)类的行为是为这个类明确定义的方法集。

(4)类的继承由这个类的直接超类组成。如果一个类没有直接超类,那么就把 OBJECT 作为直接超类。

2. 类实例的定义

(1)类实例是从这个类到特定值的映射。每个类实例有唯一的标识,即对象标识。因此“对象”和“类实例”是同一个概念。

(2)类的定义用下列形式表示:

$C = \langle [A_1 : a_1, \dots, A_n : a_n], \text{methods}, \{C_1, \dots, C_k\} \rangle$

这里 C 为一个类, $C_1, \dots, C_k$ 为类 C 的直接超类。

类 C 的类实例 c 可用下列三元组形式表示:

$c = \langle i, [v_1, \dots, v_n], [i_1, \dots, i_k] \rangle$

这里 i 为类实例 c 的对象标识, $v_1, \dots, v_n$ 是属性 $A_1, \dots, A_n$ 的值,而 $i_1, \dots, i_k$ 为类实例 c 的直接超对象的对象标识。 c 的后两个成份就是对象的状态和继承。

3. 类型的定义

类型有四种:原子类型,类类型,集合类型和元组类型。

(1)原子类型有整型、字符串型、布尔型等。原子类型也称为基本数据类型或预定义类型。

(2)类类型就是一个类定义。

(3)元组类型形为 $[A_1 : a_1, \dots, A_n : a_n]$, A_i 为属性名, a_i 为属性值的类型。

(4)集合类型限定集合中每个元素是某种类型。

4. 值的定义

由于类型有四种,因此相应的值也有下列四种:

(1)原子值是指原子类型的值。

(2)对象值是类实例的对象标识。

(3)元组值是从元组的属性到相应类型值的一个映射。

(4)集合值是特定类型值的集合。

5. OODB 模型的定义

(1)一个 OODB 模式是类定义的集合。

(2)一个类集是类实例的集合。

(3)一个 OODB 实例是类集的集合。

四、对于类的操作

在 OODB 中由于有了类继承性概念,可以在类定义中少写许多属性和方法。但需要有一些操作来求类的所有超类,所有属性、所有方法及类的值。下面分别介绍。

1. 求一个类的所有超类的操作 用下式定义:

$$\text{all_superclass}(C : \text{class}) = (C_1(\text{all_superclass}(C_1)), \dots, C_n(\text{all_superclass}(C_n)))$$

这里 n 是类 C 的直接超类的个数, $C_1, \dots, C_n$ 为其 n 个直接超类, 这个定义是一个递归定义。

2. 求一个类的所有属性的操作 用下式定义:

$$\text{all_attributes}(C : \text{class}) = \text{Att}(C) \cup \left(\bigcup_{i=1}^n \text{all_attributes}(C_i) \right)$$

这里 $\text{Att}(C)$ 是类 C 状态中的属性集, n 是类 C 的直接超类的个数, $C_1, \dots, C_n$ 是 n 个直接超类。

这个操作不仅返回类 C 及其所有超类中的属性,而且也返回这些属性中嵌套的属性,属性和嵌套的属性间用圆点来连结。属性 A 的长度 $\text{len}(A)$ 定义为属性表示中的最大圆点数。 $\text{type}(A)$ 表示属性 A 的类型。 $\text{atomic_attr}(B)$ 和 $\text{compond_attr}(B)$ 分别表示元组类型 B 的原子属性集和复合属性集。那么 $\text{all_level_attr}(B)$ 返回某类型 B 在各层次上的属性;

$$\begin{aligned} \text{all_level_attr}(B) = & \\ & \text{if atomic}(B) \text{ then } \{ \}; \\ & \text{if tuple}(B) \text{ then } \text{atomic_attr}(B) \cup \\ & \quad \text{compond_attr}(B) \cup \\ & \quad \left(\bigcup_{i=1}^m \{B_i, B_i | B_i \in \text{all_level_attr}(\text{type}(B_i))\} \right); \\ & \quad (m \text{ 是 } B \text{ 中复合属性的数目}) \\ & \text{if class}(B) \text{ then } \left(\bigcup_{i=1}^n \text{all_level_attr}(B_i) \right) \cup \\ & \quad \text{all_level_attr}(B, \text{state}); \\ & \quad (n \text{ 是 } B \text{ 的直接超类数目, } B_1, \dots, B_n \text{ 为 } n \text{ 个直接超类}) \\ & \text{if set}(B) \text{ then } \{A_i | A_i \in \text{all_level_attr}(A)\}. \\ & \quad (A \text{ 是集合类型名}) \end{aligned}$$

如果类层次或类格结构中有循环,那么上面的计算就陷入无穷循环。此时系统应规定嵌套的层数。

3. 求一个类的所有方法的操作 用下式定义:

$$\text{all_methods}(C : \text{class}) = C, \text{behaviour} \cup \left(\bigcup_{i=1}^n \text{all_methods}(C_i) \right)$$

此处 n 是类 C 的直接超类数目, $C_1, \dots, C_n$ 为 n 个直接超类。

4. 求值的操作 下面分别给出求值、求对象值、求元组值、求集合值的操作:

$$\text{UNFOLD}(\text{原子值}) = \text{原子值}$$

$$\text{UNFOLD}(o_id) = (a \text{ ppen} \bigcup_{i=1}^n \text{UNFOLD}(o_id, i)) \text{ append} (\text{UNFOLD}(o_id, \text{state}))$$

此处 o_id 是一个对象的对象标识, n 是这个对象的直接超类数目, o_id, i 为第 i 个直接超对象的对象标识。 append 为值的串接操作。

$$\text{UNFOLD}([A_1 : \text{value}_1, \dots, A_k : \text{value}_k]) = [A_1 : \text{UNFOLD}(\text{value}_1), \dots, A_k : \text{UNFOLD}(\text{value}_k)]$$

$$\text{UNFOLD}(\{\text{value}_1, \dots, \text{value}_k\}) = \{\text{UNFOLD}(\text{value}_1), \dots, \text{UNFOLD}(\text{value}_k)\}$$

五、OODB 的查询代数

查询语言是用户使用数据库的主要工具,在关系模型中,查询语言的理论是关系代数,是封闭的和安全的。其封闭性是指在基本结构下查询的输入输出都具有同样的结构,例如关系代数操作的输入和输出都是关系。关系代数的安全性是指各种操作不会产生无限关系和无穷验证。

OODB 的查询语言也应该具有封闭性和安全性。封闭性是指在类结构中定义的查询的输入输出都是对象集,安全性是指查询操作总能终止。由于我们在本文中既采用了类继承性概念又采用了对象继承性概念,因此查询代数与传统的 OODB 有所不同。下面分别介绍查询代数的集合操作(并、差、交)、投影和联接等操作。

1. 集合操作

在关系模型中,做并、差、交操作要求两个关系是相容的关系,即属性个数、属性意义都要一样。在 OODB 中也有类似操作,但对象的相等性是定义在对象标识的基础上,即只有两个对象的标识相等,才算两个对象相等。

例如要在例 1 中做 *employee* 类集和 *student* 类集的并操作。在大多数 OODB 中实际上求出 *employee* 和 *student* 对象标识的异质集(即不同结构的对象),这个异质集的类型就是类 *PERSON*。本文由于采用对象继承性,并操作的结果将是同质集(同一结构的对象)。做法如下:首先把 *employee* 和 *student* 类集转换到超类 *PERSON* 的类集(这个操作是下面定义的抽取操作),然后在 *PERSON* 类集中做并操作,因此导出的是同质集。

下面先定义抽取操作,再定义集合的并、差、交操作。

(1) 抽取操作 用下式定义:

$$\text{extract}(A, c) = \{\text{pick}(A, i) | i \in c\}$$

这里 c 是类 C 的一个类集, A 是类 C 的超类, $\text{pick}(A, i)$ 表示求出类集 c 中对象 i 的类型为 A 的超对象。这个抽取操作 extract 求出类集 c 在超类 A 中的所有超对象集合。

(2) 集合的并、差、交 设类 C_1, C_2 的类集分别

为 c_1 和 c_2 , A 是类 C_1, C_2 的最近公共超类, 那么并、差、交分别定义如下:

$$c_1 \cup c_2 = \text{extract}(A, c_1) \cup \text{extract}(A, c_2)$$

$$c_1 - c_2 = \text{extract}(A, c_1) - \text{extract}(A, c_2)$$

$$c_1 \cap c_2 = \text{extract}(A, c_2) \cap \text{extract}(A, c_2)$$

2. 投影操作

例2 考虑图2的类格结构。

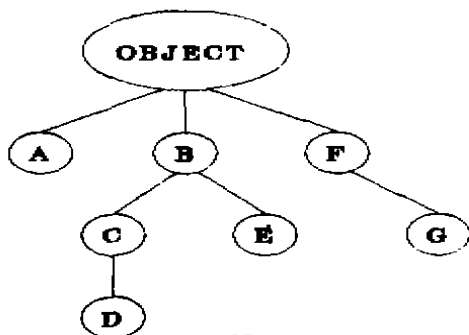


图2

投影操作 $\Pi_x(d)$ 表示计算类 D 的类集 d 在属性集 X 上的投影。操作结果是产生一个新类 D' , 它包含 X 中所有属性。关键问题是求出类 D' 在类格结构中的地位。这在各种 OODB 中的处理是不一样的。在有的系统中^[2], 规定投影产生的新类 D' , 始终是系统定义的 OBJECT 的直接子类。在有的系统中, 规定先考虑 D' 能不能是类 D 的直接超类, 如果不行, 再把 D' 作为 OBJECT 的直接子类。这些考虑都较粗糙。由于本文考虑了对象继承性, 因此可以找到投影产生的新类 D' 在类格结构中最恰当的位置。

譬如图2中, 类 A 有属性 $A_1, \dots, A_n$, 类 B 有属性 $B_1, \dots, B_m$, 类 C 有属性 $C_1, \dots, C_p$, 类 D 有属性 $D_1, \dots, D_k$, $X = \{A_1, \dots, A_n, B_1, \dots, B_m, C_1, D_2\}$ 。可以看出 X 中包含类 A 和类 B 的全部属性, 而只包含类 C 和类 D 中的部分属性。大致可看出 $\Pi_x(d)$ 产生的新类 D' 可以作为类 A 和类 B 的直接子类。因此 D' 在类格结构中的位置比较恰当。这样, 在新类 D' 中就不必包含 X 中所有属性, 也就是不必包括 D' 的超类中的属性。

投影操作 $\Pi_x(d)$ 所产生的新类 D' 形为 $(\text{new_attributes}, \text{new_methods}, \text{new_inheritance})$ 。new_inheritance 是 D' 的直接超类集, new_attributes 是 D' 的属性集, 应该是不包括在 D' 的超类中但在 X 中的属性, new_methods 是由定义在 new_attributes 上的方法集和继承的超类中的方法集组成。下面先求出 new_inheritance 和 new_attributes, 然后再定义 $\Pi_x(d)$ 操作。

(1) 求 D' 的直接超类 用下列算法实现:

算法: 求出 $\Pi_x(d)$ 产生的新类 D' 的直接超类。

集合 $A = \{ \}$;

集合 $B = \{D\}$;

repeat

for 集合 B 中每一个类 C_i do

$B = B - \{C_i\}$;

for C_i 的每个直接超类 C_{ij} do

if C_{ij} 不是 OBJECT

then $B = B + \{C_{ij}\}$;

}

for 集合 B 中每一个类 C_i do

if C_i 中的属性和 C_i 继承的属性都在 X 中

then $\{ B = B - \{C_i\};$

if C_i 不是集合 A 中某个类的直接超类

then $A = A + \{C_i\};$

}

until 集合 B 为空集;

return(集合 A);

算法的输出即是 new_inheritance。

$$(2) \text{new_attributes} = X - \bigcup_{i=1}^n \text{all_attributes}(C_i),$$

这里 n 为集合 A 中类的数目, $C_1, \dots, C_n$ 为这 n 个类。

(3) 类实例 i 在属性集 X 上的投影 具体定义如下: 类 D 在属性集 X 上的投影产生的新类为 D' , $D' = \langle [A', a'_1, \dots, A', a'_n], \text{new_methods}, \{C'_1, \dots, C'_j\} \rangle$, 那么类 D 的类实例 i 在 X 上的投影用 $i[X]$ 表示, 设 $i[X] = \langle [v'_1, \dots, v'_n], [i'_1, \dots, i'_j] \rangle$ 。

这里 $v'_1, \dots, v'_n$ 是类实例 i 在 $A'_1, \dots, A'_n$ 上的值, $i'_1, \dots, i'_j$ 为类实例 i 在 X 的投影 (也是一个对象) 的直接超对象的对象标识。实际上, $i'_g = \text{pick}(C'_g, i) (1 \leq g \leq j)$ 。

(4) $\Pi_x(d)$ 的定义 设 d 是类 D 的类集, X 是投影的属性集, 那么 d 在 X 上的投影用下式定义:

$$\Pi_x(d) = \{i[X] | i \in d\}$$

这里 $i[X]$ 返回一个新产生的对象标识。

3. 联接操作

联接有“ θ 联接”和“自然联接”两种, 大多数 OODB 实现的联接都是 θ 联接。在文[2]中, 两个类做联接操作, 返回一个新的类, 它具有操作类的属性集, 并且新的类作为 OBJECT 的直接子类在类格结构中出现。这样的联接结合律不一定成立。在文[3]中, 把联接操作的结果看成是一个参项型元组, 它由两个成份组成 (即两个操作类), 这个元组有一个新产生的对象标识。如果两个元组做联接, 一个是有 m 个属性, 另一个是有 n 个属性, 那么联接结果是 $(m+n)$ 个属性的元组。如果一个元组和一个类做联接, 元组有 m 个属性, 那么联接结果是 $(m+1)$ 个属性的元组。显然, 这样定义的联接操作在类结构中不具有“封闭性”。

在本文的 OODB 形式定义和对象继承性概念下, 联接操作还带有自然联接性质。如果被联接的两个类有共同的超类, 那么要求被联接的两个实例必

须满足它们共同的超对象。此外,联接得到的新类在类结构中有一个恰当的位置。

下面先定义自然联接导出的类的结构,再定义自然联接操作,最后举一例说明。

(1)自然联接导出的类的定义 设 c_1 是类 C_1 的类集, c_2 是类 C_2 的类集, c_1 和 c_2 的自然联接用 $c' = c_1 \bowtie c_2$ 表示, C' 是自然联接导出的类集 c' 的类类型, 那么 C' 用下列方法之一求得:

- ①如果 $C_1 = C_2$, 那么 $C' = C_1$ 。
- ②如果 C_2 是 C_1 的某一个超类, 那么 $C' = C_1$ 。
- ③如果 C_1 是 C_2 的某一个超类, 那么 $C' = C_2$ 。
- ④在其它情况时, $C' = \langle C_1. \text{state append } C_2. \text{state}, C_1. \text{behaviour} \cup C_2. \text{behaviour}, \text{new_inheritance} \rangle$

其中 new_inheritance 由满足下列条件之一的类组成: i) C_1 的直接超类, 且它不是 C_2 的直接超类的某一个超类; ii) C_2 的直接超类, 且它不是 C_1 的直接超类的某一个超类。

(2)自然联接操作的定义 据上述类定义的四种情况, 自然联接操作也有四种情况:

- ①如果 $C_1 = C_2$, 那么 $c_1 \bowtie c_2 = \{i | i \in c_1 \wedge i_2 \in c_2 \wedge i_1 = i_2\}$ 。
- ②如果 C_2 是 C_1 的某一个超类, 那么 $c_1 \bowtie c_2 = \{i | i_1 \in c_1 \wedge i_2 \in c_2 \wedge \text{pick}(C_1, i_1) = i_2\}$ 。
- ③如果 C_1 是 C_2 的某一个超类, 那么 $c_1 \bowtie c_2 = \{i_2 | i_1 \in c_1 \wedge i_2 \in c_2 \wedge \text{pick}(C_1, i_2) = i_1\}$ 。
- ④在其它情况时, $c_1 \bowtie c_2 = \{i | i_1 \in c_1 \wedge i_2 \in c_2 \wedge F\}$ 。

此处 F 是一个条件, 具体叙述如下:

对自然联接导出的新类 C' 的每个直接超类 B_i , 有下述三种情况之一发生:

- i) B_i 是 C_1 的直接超类且又是 C_2 的直接超类, 那么 $i. \text{inheritance}. B_i = \text{pick}(B_i, i_1) = \text{pick}(B_i, i_2)$;
- ii) B_i 是 C_1 的直接超类, 但不是 C_2 的直接超类。此时设 X 是一个集合, 其元素即既是 C_2 的某个超类又是 B_i 的某个超类的类。且对 X 中每个类 A_p , 有 $\text{pick}(A_p, i_1) = \text{pick}(A_p, i_2)$, 那么有 $i. \text{inheritance}. B_i = \text{pick}(B_i, i_1)$;
- iii) B_i 是 C_2 的直接超类, 但不是 C_1 的直接超类。此时, 设 X 是一个集合, 其元素即既是 C_1 的某个超类又是 B_i 的某个超类的类。且对 X 中每个类 A_p 有 $\text{pick}(A_p, i_2) = \text{pick}(A_p, i_1)$, 那么有 $i. \text{inheritance}. B_i = \text{pick}(B_i, i_2)$ 。此处 F 表示联接的两个类实例必须有共同的超对象。

例3 在图2中, 设类 D 的类集是 d , 类 E 的类集是 e , 现在要计算 $d \bowtie e$, 据上述定义可知, 联接的两实例应该在类 B 上公共超对象, 导出的类 J 应是类

C 的直接子类(图3)。新类 J 的定义如下:

$J = \langle D. \text{state append } E. \text{state}, D. \text{behaviour} \cup E. \text{behaviour}, \{C\} \rangle$

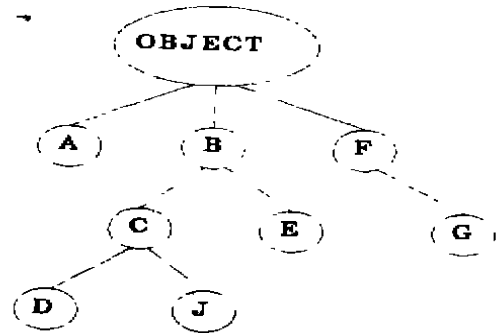


图3

显然, 这里的自然联接操作满足结合律:

$$(d \bowtie e) \bowtie g = d \bowtie (e \bowtie g).$$

六、结束语

本文主要在 OODB 中介绍了一种新的概念——对象继承性。虽然这样会在数据库中增加许多对象标识, 但是这个概念有许多优点, 使 OODB 的语义更加符合现实世界, 扩充了“对象共享”概念, 并在执行修改和删除操作时保持数据库的一致性。在此基础上定义的查询代数是封闭的和安全的, 比其它文献提出的查询代数更能表达查询结果。

参考文献

- [1] 施伯乐、丁宝康、楼荣生, 数据库系统导论, 高等教育出版社, 1994 年第一版
- [2] Kim, W., A model of querier for object-oriented database, Proc. 15th Int'l. Conf. on VLDB, 423-432, 1989
- [3] Shaw, G. M., Zdonik, S. B., An object-oriented query algebra, Data Engineering, Vol. 12, No. 3, 29-36, 1989
- [4] Loizou, G., Pouyioutas, P., A query algebra for an extended object-oriented database model, DATABASE SYSTEMS FOR ADVANCED APPLICATIONS'91, World Scientific Publishing Co. 89-98
- [5] Meseguer, J., Xiaolei, Q., A logical semantics for object-oriented databases, Proceedings of 1993 ACM SIGMOD Vol. 22, No. 2, 89-98