计算机科学1996 Vol.23 №.3
44-47

面向对象数据库中SQL语义模型的研究

陈洪亮 姜 涛
(中国科技大学电子技术部 合肥230026)

摘 要 This paper introduces one semantics model to realize SQL semantics in Object-Oriented Database(OODB). This model can keep SQL semantics in good with OODB kernel conception, also make great advantage from current Object-Oriented Programming Language's characteristics.

关键词 Object-oriented database, Kernel object conception, SQL semantics, Aggregation, View.

结构化查询语言SQL是关系数据库(RDB)中广泛使用的查询语言,主要有四大功能:查询,操纵,定义,控制,这构成了关系数据库的功能主体[1]。面向对象数据库是由面向对象数据模型定义的对象集合,该集合中的对象反映了面向对象程序设计所支持的对象语义[2]。核心的面向对象数据库系统可以用下列等式表达它的组成:

面向对象数据库系统=面向对象概念十传统数据库系统所支持的数据库特征十常驻内存的对象管理十OODB程序设计语言

面向对象数据库的查询模型与RDB中的查询模型有着显著的不同。RDB的查询是用基于值的方法从存于表中的数据集中导出查询视图,其查询路径是严格的层次路径,而OODB的查询是从一个对象到另一个对象遍历数据库,其聚集分层结构使查询路径实际上是可以有环的图,并且OODB模型中包括概括(is-a)结构,而RDB不存在这种关系结构。基于这些原因,OODB中的查询模型较RDB更为复杂,在OODB上实现SQL语言会产生许多语义和存储管理上的问题。

已有的对象数据库SQL模型普遍存在几个缺陷:

(1)定义查询操作时,往往很难确定操作结果类的类格位置,不同的模型有不同的定位策略,在不同的应用场合有各自的优点,但却难于统一。

(2)在SQL实现语义上,偏重考虑传统关系形式,没有完整的对象引用概念,在支持核心的对象语义上比较欠缺。

(3)在SQL实现语义上,未能很好结合面向对象程序语言(OOPL)中的对象程序设计机制,在具体操作实现上各有各的见解,但都不能够很好地利用现有的OOPL特征和工具。

因此,要提出支持核心对象特征的SQL语义模型,既需要支持核心的面向对象概念,又需要强调核心OODB中的对象语义特征及SQL语言特征和OOPL中的程序设计机制的结合,并针对已有SQL语言的缺陷,结合实现的难易和应用实际,修正OODB中的聚合和视图概念,同时与已有的OOPL在对象操作和实现上保持一致协调,以利于实现。

一、OODB的SQL语义模型

OODB中的一个重要概念是聚合概念,已有的聚合概念,大都认为聚合仅仅是包含多个对象的无序集合,对象之间没有什么明显的关系,这主要是受面向对象程序设计的影响,在移植到OODB中时没有恰当处理其在OODB中的特殊地位和具体含义,不能给出反映实际应用的较灵活的对象存取形式。因此需要给出比较恰当的OODB中的聚合概念:聚合是数据库中对象的最直接存取形式,聚合中的对象构成表用于记录聚合中的对象所来自的类,因此聚合体现了不同类中对象的联接关系,在类层次上反映类之间的关系。实际上,聚合主要是记录了类与类之间的联接关系和对象标识符。其表现形式可以是关系形式,也可以是对象形式(DAG)。

为了在OODB中扩展SQL语言,同时又与标准SQL语言保持兼容,在概括出SQL命令语义,表达

陈洪亮 副教授,研究生导师,主要从事计算机应用,超媒体数据库和面向对象数据库的研究。姜 涛 研究生,专业方向是面向对象数据库及超媒体数据库研究。

式语义和语法及命名机制的基础上，我们基本上保留了标准 SQL 语言的关键字和子句语法。

由于 SQL 语言是定义在 RDB 模型上，扩展到 OODB 时要概括它在 OODB 中的对应等价：OODB 中的聚合可以对应于 RDB 中的关系，OODB 中的属性类型对应于 RDB 中的属性，RDB 中的 from 子句对应于关系名表达式，OODB 中的 from 子句对应于聚合名表达式；RDB 的 Select 子句对应于属性表达式，OODB 中的 Select 子句也对应于属性表达式，但与 RDB 中基于值的查询不同，OODB 中的 Select 子句是基于对象标识符的引用。下面将详细介绍 OODB 中的查询操作和 SQL 子句。

1.1 查询操作

首先，定义查询操作的作用域为源域，源域可以是类层次结构、对象、聚合或是前次查询的结果（视图），是包含查询对象的集合。其次，定义查询操作的结果域为目标域。查询操作的作用是把源域转换为目标域，它包含以下几种操作：

1）联接（Join）：如果源域中包含的类不止一个，把每个类中的元素进行合并，最后组成单一的一个类的操作称为联接。

2）筛选（Filter）：源域可以通过对对象实例定义筛选标准而进行筛选。

3）分组（Group）：结果集合可以进行分组，以使组中成员对象在某些属性上具有相同的值。

4）排序（Sort）：结果集合可以基于一个或多个属性进行排序。

5）投影（Projection）：结果集合中每个成员可以通过函数变换转化成目标域中的每个属性。

1.2 SQL 子句

（A）Select 子句：用来说明目标域，其语法形式如下：

Select（属性名表达式）

Select 子句中的属性名表达式通常列举一些用户选择的属性类型，这些属性类型来自于 From 子句中聚合对象所拥有的属性类型，因此，如果 From 子句的对象没有定义或继承某一属性类型，则在 Select 子句中包含该属性类型是非法的。注意：在 Select 子句中的属性类型可以是 From 子句中对象的父类中的属性类型，但却不可以是 From 子句中对象的子类中的属性类型。

（B）From 子句：用来确定查询的范围，其语法形式如下：

From（聚合名表达式）

OODB 中 From 子句的处理比较复杂。由于类的属性可以嵌套，即属性可以是对象，而该对象又属于其他类，所以与标准 SQL 中只需 From 子句就可以确定查询范围不同，OODB 中有时是由 From 和 Where 子句共同确定了查询的范围。

首先，考虑在一个类中查询的情况。当只考虑类分层结构时，在一个类中的查询有两种情况：

（a）在当前类的当前所有实例中进行查询；

（b）在当前类的当前所有实例中进行查询，同时在当前类的所有子类中的所有实例中进行查询。由于子类一般情况下都继承了父类的属性，因而，这时的查询实现，是要把所有子类中的实例和当前类中的实例合并为一个临时结构，该结构可以是类，也可以是关系表或者是聚合。本文中把这种合并操作称为提交。提交的目标域是以当前类为根的类分层结构，产生一个临时结构，一般地，可以视为聚合结构，但更有实际意义的可能是临时关系表结构。

只考虑类的嵌套定义时，在一个类上的查询是在该类和该类一些属性定义域上的查询，可以表示为作用于该类及其嵌套属性对象所属类上的查询。此时的查询可以这样进行：设 A 类的某些嵌套属性来自于 B 类，在这些嵌套属性上的查询会使系统先对 B 类中满足条件的对象进行匹配（注意：这隐含着多重嵌套查询时的查询是类似于出入栈操作的查询路径），在匹配完成后得到 B 类满足条件的对象，此时根据聚合中的嵌套类对应关系回溯到 A 类中的对象查询。

其次，考虑二个类的情况：对类 A，类 B 进行查询。在 RDB 中，就对应于表 A，表 B 进行联接操作后再进行查询。在 OODB 中，与 RDB 不同，表 A，表 B 一般没有什么明显的逻辑关系（在建立联接操作之前），而类 A，类 B 却可以有以下几种关系：

（1）A，B 之间没有任何对应的属性或方法，即可将 A，B 视作由抽象类 Object 派生出来的二个正交类。

（2）A 是 B 父类，或 B 是 A 父类，这种情况可以退化为以父类为根的一个类查询的情况，只不过显式声明必须将子类对父类进行提交操作。

（3）A 中对象的某些嵌套属性是来自 B 类的对象（或 B 子类中的对象），此时，在这些嵌套属性上的查询会自然过渡到对 B 类的查询上去，根据 B 类中满足条件的"属性"对象再参照聚合中的类对应关系寻找 A 类中的对象，因而，这种情况下的查询是先对 B 类进行提交，再根据聚合中的 B 到 A 的对应关系

回溯到A类对象查询。

若A中对象的嵌套属性是B类父类的对象，此时根据Where子句中A类与B类的逻辑关系选择不同的方法。若二者是"相与"性关系(即一个类的查询结果的确定有助于另一类的查询范围的缩小)，此时应当先把B类提交到父类，之后先对B类进行查询，从而有助于减少A类的查询范围；若二者是"相或"性关系，此时就将二者视为一般情况进行查询。

以上就二个类的情况进行了初步分析。分析的首要目标是确定查询的语义，从而进行查询算法匹配和查询优化。这里的分析没有考虑二个类之间存在友类关系的情况，因为友类的存在牵涉到封装和隐藏的问题，而且友类主要用在对象的方法处理方面，在对象类的关系方面也就没有太大的意义。尽管如此，实际情况也可能复杂得多。例如，若A类对象实例的某些嵌套属性是B类中的对象，而另外一些嵌套属性是B类的友类的对象等等。

(C)Where子句(或For子句)：用来确定查询的筛选条件，其语法形式如下：

Where〈属性关系式〉

从以上分析可以知道，OODB中查询的源域包含对象实例的层次较多，因而，在Where子句进行第一步筛选很有必要，事实上Where子句中表达式的建立是查询优化的出发点。OODB中由于可以引入用户定义数据类型(UDT)，因而Where子句中的运算符可以是用户定义运算符或重载运算符，但运算的结果总是返回对象标识符的引用。

(D)Group子句：RDB中SQL的Group子句提供的分类功能是基于值的线性分类，完全由用户确定分类的层次和顺序。而OODB中由于对象有其固有的类格结构，因而，Group子句缺省时的分类顺序很自然地依赖于源域中对象的类结构和聚合组成。但由于OODB也可以支持"视图"，使查询结果存放于视图之中，因而RDB中所具有的分类功能，OODB也完全支持。

(E)SQL中的数据操作命令：Insert，Delete，Update—插入，删除，更新。标准SQL中的这些数据操作命令在OODB的SQL中是类似处理的，二者都是对查询结果一视图进行处理，区别主要在具体操作的处理步骤实现细节上，本文对此不再赘述。

1.3 SQL表达式

RDB中SQL表达式可以使用的数据类型比较有限，因而其表达式的形式比较固定，而OODB由于可以由用户定义自己的数据类型，表达式中的方法是在类上定义和实现的，所以其SQL表达式必须基于类方法的调用，以适应用户自定义数据类型的引入，而方法的执行由查询编译器在查询执行时进行联编来实现。

1.4 命名规则

在OODB的SQL中，查询所涉及到的变量命名，属性命名等命名规则应尽量遵循ANSI SQL标准，这些命名规则如下[3]：

(1)在From子句，单个命名始终认为是变量声明。

(2)在其它子句，单个命名可以解释为变量引用或属性引用。

(3)当表达式包含链式属性声明(A.B.C...)，则倒数第二个命名将被解释为属性引用。

(4)当单个命名，或者是链式属性命名的第一个元素，或者是函数/过程的一个参数，或者是只有它本身，系统先尝试将其解释为变量引用，如果解释失败，再尝试解释为属性引用。

(5)当变量引用解释失败后，系统尝试解释为属性引用。如果不存在该属性名，则查询中止。如果找到不止一个同名属性，系统根据From子句中的属性顺序来确定。(先出现者，先拥有该属性名定义。)

(6)From子句用逗号隔开基本表达式。基本表达式包含一个聚合表达式和一个可选的变量名。如果聚合表达式是个单名，系统尝试将其解释为聚合的永久名。(非临时别名)如果失败，再尝试将其解释为属性类型的永久命名。(该属性类型所在的类应当已引入聚合表达式。)

OODB中的SQL允许嵌套查询，被嵌套查询嵌入另一查询的Where子句，被嵌套查询命名环境继承于其包容查询，但可以扩展其自身变量定义。当被嵌套查询引入另一个与其包容查询同名变量时，该局部变量的作用域为该被嵌套查询。

二、OODB中的对象聚簇模式

对象聚簇的目的是为了减少对象存取磁盘I/O次数。一般情况下，磁盘传输数据的单位是页而不是一个独立对象。如果多个对象能够事先经过聚簇，当其中某一个对象被存取时，缓冲管理器能够将簇中的其他对象取入缓冲池，以提高对象存取性能[4]。

在OODB中，聚合是最基本的数据操作单元。聚合中的对象来自于不同的类，传统的存储管理是把同一类型的一组记录在磁盘上存放在一起，但这会

导致在存取聚合时引发过多的重构过程(如联接操作),因此,比较符合逻辑的是把不同类中的相关对象进行聚簇处理以获得可以接受的性能。

OODB的一个主要优点是通过对象标识符显式表示对象之间的关系,在聚合的建构过程中,聚合可以采取不同的结构形式如树状、线性链表状、网状等等,对象之间的关系可以作为对象属性的一部分给予定义,从而使系统能够根据这些信息把相关对象进行聚簇。

在大多数应用场合,对象之间的关系可以表示为层次结构或有向无环图,而在层次结构或有向无环图上进行的操作一般是通过对象结点之间的链进行浏览或者是求取给定对象的父类或子类,因此把层次结构或有向无环图结构中的对象按线性簇顺序进行聚簇可以有效提高这些操作的性能。在聚簇层次结构对象时,按深度优先顺序进行聚簇,使结点P的所有子结点存放于结点P的后面,当存取结点P和其所有子结点时,性能改善非常有效。在聚簇有向无环图结构对象时,首先设DAG有一个虚的根结点,把DAG结构转化为等效的层次结构,对具有多个父结点的结点,选择某一特定父结点使其与所有子结点构成一前向图,聚簇的结果是源于虚结点的树。

已有的OODB聚簇模式有以下几个特征:

(1)模式是静态的,即对象是建立时进行聚簇的,一旦建立,运行时就无法更改。

(2)模式使用磁盘页作为聚簇单位,不考虑磁盘页的物理相邻位置。

(3)对象聚簇模式以对象间某一特定的关系为基础进行聚簇,其他类型的关系将被忽略。

在已有的OODB聚簇模式中存在二个问题:

(1)静态聚簇模式没有考虑对象的动态模式进化,在数据库设计中,对象在设计周期的每一个阶段都要更新,频繁的更新可能会毁坏聚簇的初始结构,因此,为了进行有效的存取,需要重新组织对象的聚簇结构。

(2)层次结构或有向无环图结构中,对象之间可能存在多种关系,好的聚簇模式应当考虑多重关系,可以通过给关系赋以不同的权产生不同联接等级的对象聚簇方式。紧联接对象聚簇比松联接对象聚簇更为有效,实际上这样做同时改善了紧联接对象和松联接对象的存取性能。

无论何时对象的再次聚簇都应当使联接对象物理上进行聚簇。在读/写率高的应用中,可以采用完全动态聚簇策略,而在读/写率不高的应用中,频繁的重新聚簇可能会降低系统的整体性能,系统应当能够使重聚簇在后台自动进行。

三、OODB中的视图概念

在传统的RDB中,视图被定义为查询的输出结果,对视图可以象对关系表一样进行数据修改,更新。类似地,OODB中的视图概念也被定义为查询输出的结果,对OODB中的视图而言,有二种表示方法:"对象"形式;"关系"形式。用"对象"形式的表示方法可以表示视图中每一对象的类层次结构及相互关系;而"关系"形式采用RDB中的视图表示方法,也具有传统RDB中视图的操作功能。但在RDB中,查询一旦生成,视图的结构就无法更改,而OODB中查询输出结果是基于"对象标识符"来引用一个对象,因此,在同一查询结果的基础上,可以有不同的视图结构和表示方法。同样理由,由于基于"对象标识符"的引用,视图可以与另一视图或另一聚合再次进行查询,与RDB中基于值的引用不同,这种视图的"再查询"能够更清晰地表示对象的类分层结构,更好地保持数据引用完整性,也更容易跟踪对象的演化历程和查询的路线。

参 考 文 献

[1]胡志平,麦中凡:SQL的现状及其研究与发展,计算机科学,Vol. 21. No. 4. 1994

[2]W. Kim et al. ,Object-Oriented database support for CAD,ACM Trans. on Database,Vol. 22. No. 8. 1990

[3]Rajiv Gupta, et al. , Object-Oriented Database with Application to CASE,Networks,and VLSI CAD,Prtenice Hall. 1991

[4]N. Bhalla et al. ,Operation and queries in object-oriented database supporting complex object. Information and Software Technology, Vol. 35. No. 1. 1993