计算机科学 1996 Vol. 23 №. 4

演绎数据库优化技术新进展

袁 军 王能斌
(东南大学计算机系 南京 210096)

摘 要 Deductive database is an integration of logic programming with relational database technology. Poor performance is the very reason hindering the development of deductive database system. This paper focuses on the development of optimization technique of deductive database, and several new optimization approaches are discussed here.

关键词 Deductive database, Logic data language, Query optimization, Recursive query.

在现实应用中有一类应用不仅要求数据库系统能够存取大量的数据，而且要求数据库系统能够对所存贮数据进行知识推理，我们称这类应用是基于知识的数据密集型应用。很多迹象表明，关系数据库由于缺乏足够的语义表达能力，不适用于这类应用系统的开发，因此人们试图寻找一种新型数据库系统，它既能存贮数据，又能存贮知识推理规则，这种集数据管理和知识推理为一体的新型数据库系统就是演绎数据库系统。

演绎数据库系统将逻辑程序设计和关系数据库技术有机结合起来。这两种技术分别独立发展于七十年代，至八十年代已趋成熟，并得到了广泛应用。作为两者相结合的产物，演绎数据库系统被认为是数据库技术的一个重要发展方向。

关系数据语言(如：SQL)缺乏足够的表达能力，在大多应用系统的开发中人们不得不在过程性的程序设计语言(如：C 语言)中嵌入说明性的关系数据语言进行系统开发，而诸如 C 语言等程序设计语言是一种面向元组(ruple-oriented)的语言，与面向集合(set-oriented)的关系数据语言的计算模式不同，造成了所谓的阻抗不匹配问题，影响系统的效率。数据库研究人员一直在致力于寻找一种持久性的程序设计语言来解决这一问题，而语义强大的逻辑数据语言也许正是解决这一问题的一个有效途径。

演绎数据库系统的逻辑数据语言与关系数据语言相比，具有更强的语义表达能力。这种强大的语义表达能力主要体现在以下两点：

1. 逻辑数据语言可以表达递归查询，这是关系数据语言所不具备的，也是逻辑数据语言最具魅力的地方。

2. 逻辑数据语言通过构造函数符号的引入，可以表达较复杂的对象，突破了关系数据语言 1NF 的限制。

这种强大的语义表达能力给演绎数据库系统带来了如下优越性：

1. 逻辑数据语言具有说明性的语义，而且基于规则的表达方式简单、明了，更符合人的思维模式，使用更加方便。

2. 逻辑数据语言强大的语义表达能力使得演绎数据库系统特别适用于基于知识的应用系统开发。

3. 逻辑数据语言克服了关系数据库中的阻抗不匹配问题。

1 逻辑数据模型及其计算语义

一阶谓词逻辑早已用于定理证明和逻辑程序设计语言，直到七十年代后期才作为一种数据模型提出，这就是逻辑数据模型。在扩大数据库的查询(特别是递归查询)功能和提高数据库的推理能力方面，逻辑数据模型有不可取代的作用。

在逻辑数据模型中，每个关系对应一个谓词，例如关系 R(X,Y,Z)对应谓词 P(X,Y,Z)。设 a,b,c 是三个常量，谓词 P 的定义是：如果(a,b,c)∈R，则 P(a,b,c)为真，否则为假。谓词可表示两种关系，一种是实际存储在数据库中，称为外延数据库(简称 EDB)关系；另一种是只有定义，而其元组并不存储在数据库中，在需要时可以导出，这称为内涵数据库(简称 IDB)关系。通过 IDB 关系，可以在数据库中按需要定义各种导出关系。从逻辑数据模型的观点来看，数据库是一组谓词的实例的集合；查询和 IDB 关系都可以在用逻辑程序中用规则表示。事实加上规则就构成一个演绎数据库。

从数据库的观点来看，不但要知道什么是演绎数据库，而且还要有相应的算法去获取其语义，也就是要给逻辑数据模型一个计算定义，才有实际意义。

对一个演绎数据库求值时，必须要知道各谓词

间的依赖关系。这种依赖关系可用一个有向依赖图表示。对于非递归规则，可按依赖图的次序，自底向上，计算各 IDB 关系的值。为此，可将规则变换成对应的关系方程(Datalog 方程)。递归规则也可以变换成相应的 Datalog 方程，不过这是递归的。一条规则的计算就是 Datalog 方程的计算，对递归规则来说是一种迭代计算。迭代计算的终止条件是方程中的关系不再产生新元组，这样计算得的结果称为不动点。在一般情况下，一个递归 Datalog 方程可能有多个不动点。可以证明，在不考虑否定的情况下，用迭代计算计算 Datalog 方程所得到的不动点是最小不动点(简称 LFP)，即它是递归 Datalog 方程任何其他不动点的子集，而且最小不动点与最小模型是一致的。在逻辑数据模型中，就以最小不动点作为递归谓词所对应的 IDB 关系的值，这就是递归谓词的计算语义。

2 递归查询的计算模式

演绎数据库针对递归查询的计算有两类不同的计算模式，即自顶向下的计算方法和自底向上的计算方法。后者是从 EDB 事实出发，由规则体到头部应用规则，隐式地由树叶向上直到根构造一棵 IDB 元组树。这种逻辑的自底向上处理又称作向前链接法。semi_naive 方法就是一种自底向上的计算方法。自顶向下的计算方法是从想获得的目标出发，通过每条规则从头部到规则体的运用，扩展规则/目标树，通过合一(unification)实现信息传递，直至找到相应的 EDB 事实与经扩展的子目标相匹配。这种逻辑的自顶向下的处理方法又称作向后链接法。Prolog 关于逻辑的计算就是一种自顶向下计算方法。两种计算模式究竟孰优孰劣，在理论上有过一段时间的争论，到八十年代末，由于一些基于自底向上计算模式的优化算化(如 magic set)的出现，使人们逐步达成了共识，即自底向上的计算方法要优于自顶向下的计算方法，其优越性体现在以下几点：

1. 采用向前链接的推理方式简单、直观，更符合人类的思维方式。

2. 自底向上计算方法从事实出发，计算扩充的 Datalog(含有函数符号)规则，该方法只需一种简单的基本项匹配(ground term matching)算法就可以实现。而对自顶向下计算方法来说，需要较复杂的合一算法。

3. 能够保证安全的 Datalog 规则的计算可终结性，而自顶向下计算方法有可能会导致推理的无限循环。

4. 自底向上计算方法是一种面向大关系的运算，可以利用关系数据库中较成熟的大关系连接技术对计算作进一步的优化。

5. 对于一个安全的 Datalog 程序 P_1 来说，设 Q 是一个查询，则必定存在另一个 Datalog 程序 P_2，对 Q 来说，P_1 与 P_2 的查询结果相同，而且 P_2 采用 semi_naive 算法的计算效率不低于 P_1 采用自顶向下的 QRGT(queue-based rule goal tree)算法[1]。

出于以上的优点，所以大多数的演绎数据库原型系统(包括 LDL，Nail!，Adtit 等)对于递归查询的计算，均采用自底向上的计算方法。值得一提的是，引入 Magic Set 等优化方法以后的自底向上计算方法从某种意义上说已不再是原先那种单纯的自底向上的方法，实质上它是自底向上计算方法和自顶向下计算方法相结合的产物，只不过最后的计算仍是一种向前链接方式，所以仍将其归类于自底向上的计算方法。

3 演绎数据库优化策略

作为数据库的一个重要发展方向，演绎数据库有着诸多的优越性，但至今尚未有商品化的演绎数据库系统问世，效率低(尤其是递归查询的计算效率低)是阻碍其发展的重要原因。如何提高演绎数据库的效率，是演绎数据库研究领域所面临的一个主要问题。究竟应该从哪里入手进行演绎数据库的优化工作，是演绎数据库优化研究领域的一个基本问题。通过分析逻辑数据模型的计算语义，我们认为演绎数据库可以从以下五个方面进行。

3.1 减少无关计算

无关计算包括两方面含义，一个是无关数据的计算，另一个是无关操作的计算。

3.1.1 无关数据的计算。逻辑数据语言的计算首先将一阶谓词逻辑转换成相应的关系演算，然后再进行多种关系操作得到计算结果，因此逻辑数据模型的计算实质上仍是一种面向关系的运算。对于用户提交的具体查询，由于存在某些特定的约束条件，使得对一个关系中的大量数据来说，可能仅仅需要其中的一部分数据参与关系运算就可以得到正确的计算结果。关系中无需参与关系运算的那部分数据就是无关数据。尽可能多地将无关数据从关系中剔除将会在很大程度上提高计算的效率。魔集方法是到目前为止最为成功的一个基于这种思想的优化方法。魔集方法根据用户提交的具体查询中关于变元的约束，利用约束的旁路传递(sideway information passing)，尽可能地将约束向下传，延续关系数据库中将选择和投影尽可能下压的宗旨，使得参与计算的事实数量尽可能少。魔集方法的通用性和高

效性使其成为目前递归规则自底向上计算的首选优化方法。与魔集方法机理相同的一系列线性优化技术也是目前较为流行的优化方法，它们包括：右线性改写方法(right linear transformation)、左线性改写方法(left linear transformation)和计数方法(counting set)。

右线性改写方法和左线性改写方法是目前成功的两个面向线性递归规则的规则改写方法，这两种方法通过约束向下传递，减少了参与计算的事实数量，它们比魔集方法更为简洁、高效，但是对所适用的规则加了较多限制，具体表现在右线性递归规则必须保证规则头部的自由变元和规则体递归子目标中的自由变元存在一个变元名和变元位置上的一一映射关系；而左线性递归规则必须保证规则头部的约束变元和规则体递归子目标中的约束变元存在一个同样的一一映射关系。文[4]中提出了广义左、右线性递归规则组的定义，放宽了左、右线性递归规则对规则形式的限制。广义左线性递归规则只限制规则头部的约束变元在规则体递归子目标中也是约束变元；广义右线性递归规则只限制规则头部的自由变元在规则体递归子目标中也是自由变元；广义左、右线性递归规则并不要求变元在位置上有一个一一映射关系。广义左、右线性递归规则可以改写成左、右线性递归规则而不失任何语义。这样，高效的左、右线性规则改写方法可以适用于更广泛的规则。计数方法是又一个具有较高效率的面向线性递归规则的规则改写方法。传统的计数方法限制递归规则组中只有一条递归规则出现，而且还限制规则体中的有些子目标之间不允许有共享变元。文[2]中对传统的计数方法进行了拓广，通过将规则中的计数变元由简单的整数项推广为较复杂的函数项，使得计数方法能够适用于含共享变元的多递归线性规则组。

3.1.2 无关操作的计算。对于由一阶谓词逻辑转换而成的关系演算来说，如果从中去掉若干个关系操作并不改变计算的语义，则这些被去掉的关系操作就是无关操作。尽可能多地找出计算中的无关操作并删除之同样可以显著地提高计算的效率。

减少无关操作是语义查询优化的一个重要的优化思想。演绎数据库语义查询优化是根据数据库完整性约束条件中的语义信息，将用户提交的规则改写成语义等价的另一组规则，而新规则组比原始规则组有更高的执行效率。语义查询优化在演绎数据库领域有着独有的优越性，这是因为基于一阶谓词逻辑的逻辑数据语言具有很强的表达能力，数据库完整性约束条件与IDB谓词可以用一种较为统一的形式加以描述，使得在优化过程中寻找两者之间的联系变得相对简单，有利于优化方法的实现。

语义查询优化是建立在规则语义等价基础上的一种查询优化技术，一般可以通过规则改写方法来实现。如何发现规则中的多余子目标是语义查询优化的关键，同时也是一个十分复杂的问题，人们很难找到一个充要条件对其进行判断。Chakravarthy首先将语义查询优化的思想引入了演绎数据库领域并提出了基于Residue的语义查询优化的方法。基于Residue的语义查询优化方法有严格的形式定义和理论证明，但效率不高，这是因为：

①作为该语义查询优化方法的核心部分的Residue产生算法中需求解两条规则的最一般合一mgu，而任何一个合一算法的效率都不高。

②该语义查询优化方法虽然能够适用于递归规则的计算，但Residue的产生嵌入在每一轮递归迭代计算中，对于递归规则来说，该方法在计算中增添了相当高的额外计算代价。

在文[7]中，作者从分析若干个EDB谓词间的一种包含约束条件入手，通过定义规则输出变元的相关性和谓词的相关性，给出了判定规则中基于这种包含约束条件的多余子目标的充分条件，并提出了基于该条件的发现规则中多余子目标并删除该子目标的算法。文中提出的语义查询优化方法通过规则编译时对规则进行改写完成，避免了基于Residue的语义查询优化方法在递归规则计算中的额外计算代价；而且文中所提出的算法都具有多项式级的时间复杂性，与基于Residue的语义查询优化方法相比，具有更高的效率。

3.2 减少重复计算

重复计算是影响逻辑数据模型的计算效率的另一个重要因素。造成重复计算的原因包括用户提交的规则中对IDB的重复定义和Datalog方程在迭代计算中同一事实多次重复计算同一事实等。对于递归规则的自底向上的计算来说，从原始的naive方法到基于增量求解的semi_naive方法的改进就是为了降低计算中的重复计算量。可以说，如何降低规则计算中的重复计算量是优化演绎数据库计算的又一个重要研究方向。

3.3 函数项的计算优化

引入函数项是逻辑数据语言对关系数据语言的重要扩充，现有的研究实现了含函数项的逻辑数据语言的计算，但对其效率没有作更多的讨论。随着应用领域不断地提出新的要求，仅仅引入简单的函数项已经不能够满足某些应用的需求，最近人们又提出了将面向对象数据库和演绎数据库相结合新方向。所以，怎样正确有效地将含有函数项的一阶谓词

逻辑用关系演算来表达，并使其具有较高的计算效率不仅仅是演绎数据库优化的一个重要研究方向，同时也是为寻找导航式查询和面向集合查询之间的联系作一些技术上的准备。

集合因其丰富的语义、引入逻辑数据语言后提高了演绎数据库的语义表达能力，但同时也给演绎数据库的优化提出了新问题。集合项的匹配具有变元不确定性、幂等性和交换性等特性，而满足幂等律和交换律的项匹配问题很难从算法上进行彻底的优化。LDL 系统在对含集合符号的规则进行计算时，将含有集合符号的规则，改写成一组 MHMB(Multi-Head Multi-Body)规则进行计算，这种方法使得原规则中不含集合项的其它子目标也必须在改写后的多条规则中重复计算，而且 MHMB 形式的规则头部是子目标的析取形式，因此也可能带来语义的不确定性，在某些情况下会导致计算不终结。在文[5]中引入了一种集合项归约演算，尝试从另外一条途径去优化计算。基于这种归约演算，通过扩展集合项在数据库中的存贮模式，将集合项的唯一存贮按可能的模式进行扩展，使一般函数符号的项匹配算法就能完成集合项的匹配。集合项存贮模式的扩展可以在规则编译时完成，但这要求每次 EDB 修改后须进行相应的存贮元组改写，因此文[5]的方法更适用于 EDB 无需经常修改的应用。从本质上说，文[5]中提出的方法趋向于大关系的计算，而 LDL 的方法更趋向于将一个大关系的计算变成多个小关系的计算。

3.4 计算并行化

并行化是当今计算机领域的一个研究热点，这在演绎数据库领域也不例外。逻辑数据模型的计算过程中存在着许多可以并行化的成份，利用并行处理提高演绎数据库的计算效率有极大的潜能。对于关系数据库来说，将数据在多个处理机之间进行合理分布，能够很好地实现计算并行化。但对于含递归规则的逻辑数据语言来说，若采用此类并行策略(如：数据归约范例)，由于存在规则的迭代计算，一次计算的结果要作为下一次计算的输入，这就可能需要计算结果在处理机之间进行传递，即在计算的并行化的同时又带来了高额的传递代价。如何减少这种计算结果的传递是演绎数据库并行处理的一个研究热点，选择更合理的数据分布方案以及寻找新的并行计算模型是解决这一问题的两条途径。

在文[6]中受并行二分法的启发，提出了一种基于 Datalog 规则的并行策略——规则分解并行策略。该方法通过编译时对规则的改写，形成若干组不同的规则组，将这些规则组分别分配到不同的处理机上进行计算，从而实现递归 Datalog 规则计算的并行化。规则分解并行策略是一种新颖的递归规则并行计算策略，与以往的并行策略不同，它的规则改写不是强调数据的划分，而是强调规则计算过程的分解，把一个问题的求解转换成多个同样性质但规模降低的问题进行求解。同其它的并行策略相比(如：数据归约规范)，规则分解并行策略不需要将计算结果在处理机之间相互传递，具有更高的效率。基于规则分解并行策略，采用简单的规则代入方法对规则进行加工，文[6]中成功地实现了单线性递归规则组的并行计算。但同时，也存在非线性递归规则的并行计算和多递归规则组的并行计算两方面需要进行进一步的研究。

3.5 连接优化

传统的连接优化方法在处理含有较少数目连接的查询时具有很好的优化效果，而对长连接查询则考虑得不够。例如，System R 系统的动态规划算法在最坏的情况具有时间复杂性 O(2N)，这里，N 是连接操作的个数，当 N 超过 10 时，该算法不再适用。

连接操作是关系操作中最为常用也最为费时的一种操作，连接优化是任何一个关系数据库系统不可缺少的一部分。与传统的关系数据库查询相比，演绎数据库查询具有如下特点：

1. 在演绎数据库查询计算中，常常会产生大量含仅少量元组的临时关系，而关系数据库一般涉及少量含大量元组的永久性关系。

2. 由于逻辑数据语言本身的特点及逻辑程序变换需要，常常出现一条规则含有多个子目标，对该规则的计算包含了较多数目的连接操作，我们称此为长连接操作。

因此，演绎数据库的连接优化与关系数据库的连接优化的侧重点有所不同，演绎数据库更侧重于减少临时关系和长连接的优化两方面内容。

一种被称为三路连接或多路连接的方法能够一次完成多个连接操作，把它应用于长连接计算具有很好的优化效果。但是，采用三路连接或多路连接的方法使得长连接的优化变得复杂。在为连接操作(包括三路连接和多路连接)确定一个合理的执行顺序之前，必须要选择适当的三路连接或多路连接执行策略。影响连接次序的因素包括连接结果的预测和内、外存的调度等，由于长连接查询中连接操作数目多，搜索空间大，采用穷举法不再有效，对长连接的优化更多的是采用启发式方法。文[3]中提出了一种适用于演绎数据库系统的多路连接方法和判定多路连接顺序的判定算法。

(下转第 15 页)