计算机科学1999Vol. 26№. 7

基于嵌入式关系的一种 Multi-join 算法

A Multi-join Algorithm Based on Nested Relations

何 伟 洪晓光 王海洋 董继润

（山东大学计算机科学系 济南 250100）

Abstract The nested relational model has a better way to represent complex objects than 1NF relational model. Detailed examination and analysis has proven that the nested relational model is equally robust. The join operation is one of the most expensive operations in nested relational database as well as in 1NF relational databases. In this paper we first introduce a form of join operator in the nested relational model. After that, an algorithm for computing multi-join among nested relations on decomposed storage model is proposed.

Keywords Nested relation, Multi-path, Storage model, Query optimism

去掉第一范式"原子属性"的限制条件而得到的关系模式称作嵌入式或扩展的关系模式。嵌入式关系模型已经被数据库理论界接受为一种规范的关系模型，它可以有力地支持存储树型（层次）或图形（网状）数据的系统，比如办公自动化、多媒体系统等许多新的应用领域。试验及分析证明嵌入式关系模型与符合第一范式的关系模型同样严格，换句话说，只要遵守更高的规范形式，删除1NF关系"原子属性"的限制不会导致关系数据库产生不合法的或不一致的数据。嵌入式关系模型同1NF关系模型相比主要有以下优点：

·数据库模式更加简单，定义、维护较少的表和视图；

·嵌入式数据库模式更直接地反映数据之间固有的关系，从而使数据库模式更容易定义和理解；

·仅仅用来存储关联的数据表可以删除，从而节省存储空间；

·可以避免许多代价颇高的连接操作；

·访问数据更加简单、直接。

连接操作是关系数据库中最常用、代价最高的查询操作之一。由于嵌入式关系模型比1NF关系模型表达复杂对象的能力更强，在嵌入式关系模型中许多复杂对象的信息不必分布于多个关系中，因此可以避免很多在1NF关系模型中常用的连接操作。但是，任何数据存储模型都不可能满足所有的查询要求，连接操作在嵌入式关系的某些查询中同样是必需的，因此很有必要研究如何将关系数据库的连接技术扩展到嵌入式关系数据库系统。

本文首先介绍文[1]提出的一种嵌入式关系的连接操作（基于路径的连接 P-join）；基于此，提出一种有效的嵌入式关系内的多连接算法。

一、基本概念和 P-join 连接操作

一个嵌入式关系模式可以看作一棵有根节点的树，称作模式树（用 T_R 表示关系模式 R 的模式树），树的节点标记为属性名。

以下给出定义在嵌入式关系上的连接操作 P-join，我们首先介绍扩展于嵌入式关系的笛卡尔积 $\times^*$ 和为 P-join 提供取值范围（域）的基于路径的笛卡尔积 $\times^P$。

1.1 扩展笛卡尔积 $\times^*$

扩展的笛卡尔积 $\times^*$ 是从两个嵌入式关系的顶层开始做笛卡尔积，遇到同一层次相同名称的非叶节点属性，将其合并，对它们表示的子关系递归形成 $\times^*$。例如，存在两个关系模式分别为：

$R(A, X, (B, C), Y(D, E))$ 和 $Q(F, X(B, C), Y(D, G))$，那么 $R \times^* Q = (A, F, X(B_1, C_1, B_2, C_2), Y(D_1, E, D_2, G))$

其中 B_1 和 B_2 用来区分两个关系中的属性 B，C_1 和 C_2 区分两个关系中的属性 C。

1.2 基于路径的扩展笛卡尔积 $\times^P$

定义1 设 T 是关系模式 R 的模式树，如果 N_1 是 $root(T)$ 的子节点并且 N_k 是一个非叶节点，那么，路径 $P_t = (N_1, \cdots, N_k)$ 就称作 R 的连接路径。

基于路径的笛卡尔积$\times^p$形成的基本思路是沿着连接路径对子模式内部应用扩展笛卡尔积$\times^e$。假设关系r和q的两条连接路径分别是$P_r=(R_1,\cdots,R_k)$和$P_q=(Q_1,\cdots,Q_m)$，不妨设$k\geqslant m$，我们从关系r的R_{k-m}属性（R_0属性表示$root(R)$）和Q_0（即$root(Q)$）开始，向下分别沿着路径P_r和P_q对子关系内部应用扩展笛卡尔积$\times^e$，两条连接路径中相对应的属性合并为同一属性。

1.3 基于路径的连接操作(P-Join)

由于连接路径$P_r(N_1,\cdots,N_k)$完全可以由属性N_k决定，以后我们用$r(N_k)$标识$r(P_r)$；其中N_k称作连接路径P_r的路径决定节点。

定义2 设r和q是两个关系，连接路径分别是P_r和P_q。在关系$r(N_R)\times^p q(N_Q)$的模式$R(N_R)\times^p Q(N_Q)$中，如果：(1)$A^1,\cdots,A^m$是在模式R和Q二者都出现的属性，并且$A_r^i,A_q^i(1\leqslant i\leqslant n)$的父节点相同；(2)$B^1,\cdots,B^m$是在模式$R$和$Q$二者都出现的属性，并且$B_r^i,B_q^i(1\leqslant i\leqslant n)$的父节点不同；(3)如果$A$和$B$是属性名相同的叶子节点，则$A$和$B$具有如下关系：或者$A$的父节点是$B$的某一个祖先节点，或者$B$的父节点是$A$的某一个祖先节点，则：

$$r(N_R)\bowtie^p Q(N_Q)=\delta_{t2}(\Pi_X[\delta_{t1}[r(N_R)\times^p q(N_Q)]])$$

其中：$t1=(P^1\cdot A_r^1=P^1\cdot A_q^1)\wedge(P^2\cdot A_r^2=P^2\cdot A_q^2)\wedge\cdots\wedge(P^m\cdot A_r^m=P^m\cdot A_q^m)$，$P^i(1\leqslant i\leqslant m)$表示节点$A^i$的父节点；

X为模式$R(N_R)\times^p Q(N_Q)$除去属性$P^1\cdot A_q^1,\cdots,P^m\cdot A_q^m$；

$t2=(P_1^1\cdot B_r^1=P_2^1\cdot B_q^1)\wedge(P_1^2\cdot B_r^2=P_2^2\cdot B_q^2)\wedge\cdots\wedge(P_1^n\cdot B_r^n=P_2^n\cdot B_q^n)$，$P_1$、$P_2$分别表示节点$B_r^i$、$B_q^i(1\leqslant i\leqslant n)$的父节点。

例如，存在两个关系$R(A,X(B,Y(C,D)))$和$Q(C,Z(D,E))$，则有

$$R(Y)\times^p Q(Z)=(A,X(B,C_q,YZ(C_r,D_r,D_q,E)))$$

$$r(Y)\bowtie^p q(Z)=\delta_{YZ\cdot C_r=X\cdot C_q}(\Pi_{A,X(B,C,YZ(C,D-k))}[\delta_{Y2\cdot D_r=YZ\cdot D_q}[r(Y)\times^p q(Z)]])$$

至此，我们已经介绍了嵌入式关系的基本概念以及一种定义在嵌入式关系中的基于路径的连接操作P-join。下面我们将讨论计算P-join连接的算法。在此之前，我们先介绍本文算法所采用的存储模型。

二、一种嵌入式关系存储模型

文[5]给出了嵌入式关系数据库的几种存储模型。使用何种存储模型决定于查询分布，同时又对连接算法产生影响，本文讨论的连接算法基于分解存储模型。

所谓分解存储模型就是将一个嵌入式关系分割为几个1NF关系进行存储，嵌入式关系中同一层次的属性在同一个1NF关系中。例如关系$R(A,B,X(C,D))$按照分解存储模型存储为两个关系：$R_1(A,B)$和$R_2(C,D)$。

三、计算嵌入式关系连接的基本机制

假定存在两个嵌入式关系$R(A,X(B,C))$和$S(A,Y(B,D))$，需要计算R和S基于路径(X和Y)的自然连接$r(X)\bowtie^p s(Y)$。关系R和S分别存储为$R_1(A,P_1)$、$R_2(B,C)$和$S_1(A,P_2)$、$S_2(B,D)$，其中P_1、P_2表示指向子关系的指针字段。

计算$r(X)\bowtie^p s(Y)$的基本思路是：首先计算$R_1\bowtie S_1$，然后对其中每个元组的指针字段所指向的子关系做自然连接。

算法1

步1 计算$R_1\bowtie S_1$，输出结果为$Temp(A,P_1,P_2)$。

步2 将关系$Temp$的元组尽量读入内存块，对块中每个元组$t_i=(a_i,P_1(a_i),P_2(a_i))$，将指针$P_1(a_i)$和$P_2(a_i)$指向的子关系分别记为$X_i(B,C)$和$Y_i(B,D)$。计算$X_i\bowtie Y_i$。如果$X_i\bowtie Y_i$非空，将其元组存入关系$OUT_2(B,C,D)$；同时将元组$(a_i,P'(a_i))$存入关系$OUT_1(A,P')$，$P'(a_i)$指向$X_i\bowtie Y_i$的第一个元组。

步3 重复步2，直到$Temp$中所有元组被处理完。

以算法1为基础，文[1]介绍了一种计算两个嵌入式关系的P-join连接算法。但是，在文[1]中并没有讨论多个嵌入式关系的连接问题，然而嵌入式关系内的多连接要耗费更高的代价，所以优化问题显得更为突出。为此，本文提出一种计算嵌入式关系的多连接算法。

四、计算多个嵌入式关系的自然连接算法

基于路径的连接操作P-join与通常1NF关系的连接操作具有某些共同的特点，比如交换率、结合律、投影可交换性等，文[3]和[4]讨论了这些特点，这对于嵌入式关系模型的查询优化非常有用，同时也保证了下面算法的正确性。

本文将要提出的计算多个嵌入式关系的自然连接算法采用一种"连接代价最小的关系优先连接"的策略，基本思路是：始终在参与连接的关系序列中选取两个连接代价较小的嵌入式关系；根据前面定义

的 P-join 操作中的连接路径，对当前两个嵌入式关系从连接节点的顶层开始，应用算法1，可得到当前两个嵌入式关系的连接。以下是算法的描述。

算法 LCF (the Least Cost, the First join)

m 个嵌入式关系 $r_1, r_2, \cdots, r_m$ 按分解存储模型分别存储为：

$r_1^1, r_1^2, \cdots, r_1^{n1}$；

$r_2^1, r_2^2, \cdots, r_2^{n2}$；

……

$r_m^1, r_m^2, \cdots, r_m^{nm}$。

假定在每一个嵌入式关系 r_i 的($1 \leqslant i \leqslant m$)序列中，含有连接节点属性的分解关系位于关系序列的尾部。同时令任意两个关系 r_i 与 r_j($1 \leqslant i \leqslant m, 1 \leqslant j \leqslant m$)的第一对含有某些相同属性，并且这些属性的父节点在连接路径上的分解关系分别记做 $r_i(j)$ 和 $r_j(i)$。显然有，$r_i(j) \in \{r_i^1, r_i^2, \cdots, r_i^{ni}\}$，$r_j(i) \in \{r_j^1, r_j^2, \cdots, r_j^{nj}\}$。

步1 建立连接序列 S，包含所有参与连接的关系 $r_1, r_2, \cdots, r_m$ 及各自的连接路径 $path_1, path_2, \cdots, path_m$。

步2 从序列 S 中选取两个关系 r_i 和 r_j，使得 $r_i(j)$ 或 $r_j(i)$ 的元组数在所有 m 个关系中最小。并且 $r_i(j)$ 和 $r_j(i)$ 在两个关系序列 $\{r_i^1, r_i^2, \cdots, r_i^{ni}\}$ 和 $\{r_j^1, r_j^2, \cdots, r_j^{nj}\}$ 中分别是 r_i^k 和 r_j^l。($1 \leqslant k \leqslant n_i, 1 \leqslant l \leqslant n_j$)

步3 对关系 r_i^k 和 r_j^l 沿着连接路径应用算法1，直到到达连接路径的终点(即路径决定节点)，然后，分别修改 r_i^k 和 r_j^l 的上一层分解关系每个元组的指针属性值。

步4 从当前两个嵌入式关系 r_i 和 r_j 的顶层开始，对每一对含有相同属性的分解关系 r_i^s 和 r_j^t($1 \leqslant s \leqslant k-1, 1 \leqslant t \leqslant l-1$)做自然连接。

步5 在 r_i 和 r_j 连接的最终模式树中，修改相关的指针属性值。

步6 将 r_i 和 r_j 的连接结果 r' 及连接路径 $path'$(由 r_i 和 r_j 的连接路径 $path_i$ 和 $path_j$ 确定)插入序列 S，同时从序列 S 中删除关系 r_i 和 r_j 及二者的连接路径。

步7 重复步2～6，直到序列 S 中仅剩余一个关系，即为连接结果。

讨论 从算法 LCF 的描述我们可以看出，两个嵌入式关系自然连接的复杂性主要产生于算法 LCF 的步3，随着连接路径长度(嵌入式深度)的增长，复杂性将成数量级地增长。但是，在连接路径已经确定的情况下，沿连接路径参与连接的第一对关系的连接代价直接影响到整个步3的代价。

算法 LCF 就是基于以上事实，每次从嵌入式关系序列中取出一对这样的关系：它们沿连接路径参与连接的第一对关系规模最小，这样就使每一次连接的代价始终是较小的，从而减少整个算法的代价。

参 考 文 献

1 Liu Hong-Cheu, et al. An Efficient Join for Nested Relational Databases. LNCS 1134. In: Wagner · Thoma, eds. DEXA' 96, Database and Expert Systems Application Springer Press

2 Roth M A, et al. Extended Algebra and Calculus for non-1NF Relational Databases. ACM Transactions on Database Systems, 1988, 13(4)

3 Liu H C, Ramamohanarao K. Algebraic Equivalences among Nested Relational Expressions. In: Proceedings of Third Int. Conf. on Information and Knowledge Management. Gaitherburg, Maryland, 1994

4 Garnett L, Tansel A. Equivalence of the Relational Algebra and Calculus for Nested Relations. Computers Math Applic., 1992, 23(10)

5 Hafez A, Ozsoyoglu G. Storage structures for nested relations. IEEE Database Engineering, 1988, 11(3)

(上接第86页)

6)验证 $(A)_D$ 是由 D 签发的，此时所有的验证结束。

于是我们得到 e 为验明 a 的身份所走的一条认证路径：$\langle a\rangle_B(B)_A\langle A\rangle_D(D)_D$。

参 考 文 献

1 Garfinkel S, Spafford G. Web Security & Commerce O' Reilly & Associates, June 1997

2 ITU-T. ITU-T Recommendation X. 509. ITU-T, June 1997

3 Ford W, Baum M. Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption. Prentice Hall, 1997

4 Hughes J. Certificates Inter-operability. Entegrity Inc, July 1998