LNFDBS的查询优化算法及联邦条件下的考虑*

姚卿达 何 昕 黄晓春 李志方

(中山大学岭南(大学)学院 广州 510275)

摘要 查询是一个数据库管理系统的核心功能,一个功能强大而又高效的查询算法的设计是一个成功的数据库系统的关键.文章提出了一个数据库查询优化算法,并实现于联邦数据库系统LNFDBS(Lingnan federal database system).另外,对查询算法在联邦条件下的优化进行了探讨.

关键词 查询优化算法,连接,投影,选择,联邦数据库系统.

中图法分类号 TP311

本查询优化算法目标是实现LNFDBS(Lingnan federal database system)下标准SQL语言的SELECT语句的查询功能,它的基本形式为:

SELECT (目标列)
FROM (基本表)
[WHERE (条件表达式)]

本算法的实现环境是联邦数据库系统LNFDBS,其体系结构如图1所示.

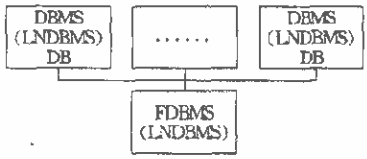

图1

LNFDBS采用类似于总线形网络的体系结构,各个DBMS为成员结点,LNFDBS相当于服务器.各个结点的DBMS具有自治性,同时也可通过LNFDBS访问其它结点.另外,它也允许任意结点之间的直接通信.目前,主要以ORACLE,FOXPRO,LNDBMS作为LNFDBS的3个成员.

在上述环境下,我们实现了对LNFDBS的查询优化.本文中的例子使用了以下4个基本表:

S(SNO, SNAME, DEPT, SEX): 学生的学号、姓名、所属系及性别;
C(CNO, CNAME, TEACHER): 课程的编号、课名和教师名;
SC(SNO,CNO,SCORE): 某学生的某门课的成绩;
SA(SNO,ADDRESS): 学生的地址

1 查询优化的目标与原则

查询优化的总的目标是:选择最有效的策略,以最小的时间、空间消耗来完成所要求的查询.对于SQL查询语句,只需要笛卡尔积(或连接)、选择、投影3种关系运算就足以获得其等价的表达式.关系代数表达式优化的一般原则是:①选择运算应尽可能先做;②执行连接前对文件进行适当的处理.例如:对文件排序或在连接属性上建立索引;③把投影运算和选择运算同时进行;④把投影同其前或后的双目运算结合起来;⑤把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算.⑥找出公共子表达式.本查询优化算法正是体现了上述查询优化原则.

2 算 法

2.1 设计思想

通常的优化算法首先建立关系表达式的查询语法树,然后,(1) 把形如$\delta_{F_1 \wedge \ldots \wedge F_n}(E)$的每一选择变换成串接形式$\delta_{F_1}(\ldots(\delta_{F_n}(E))\ldots)$;(2) 把每个选择尽量移向树的叶端;(3) 把每个投影尽量移向树的叶端;(4) 尽可能把选择和投

---

* 作者姚卿达,1937年生,教授,主要研究领域为数据库与知识库,网络与应用.何昕,1972年生,硕士,主要研究领域为数据库,分布式计算.黄晓春,女,1974年生,硕士生,主要研究领域为数据库与知识库.李志方,1973年生,硕士生,主要研究领域为数据库与知识库.

本文通讯联系人:姚卿达,广州 510275,中山大学软件研究所

本文1995-10-09收到原稿,1997-05-13收到修改稿

影的串接合并成单个选择、单个投影或一个选择后跟一个投影;(5) 把上述得到的语法树的内节点分组,每一双目运算和它所有的直接祖先为一组(这些祖先是投影、选择运算).如果其后代直到叶子全是单目运算,则也可将它们并入该组.但是当双目运算是笛卡尔积,而且其后的选择不能与之结合为等值连接时除外.这种情况下,把这些单目运算单独分为一组;(6) 生成一个程序,每组节点的计算是程序中的一步.各步的顺序是任意的,只要保证任何一组的计算不会在它的后代组之前计算.

而本算法则不需要预先建立关系表达式的查询语法树.它在进行查询连接的同时就进行优化,而不是象上述算法那样,先优化,再查询连接.

对于SQL的SELECT语句的查询,主要有选择、投影等运算,而在这些查询的各种运算中,最耗时的就是连接运算.另外,对于N个基本表的查询,要且仅要$N-1$次连接.因此,本算法的策略是以连接为中心,做$N-1$次连接,每次连接时,综合尽可能多的选择和投影.

2.2 数据结构

本算法中,对SQL查询语句所涉及的每一基本表都建立一个结构(内部表):

```
typedef struct {
    char TableName[TABLE_NAME_LENGTH];      //表名
    char Alias [TABLE_NAME_LENGTH];         //别名
    char TableType;                         //表的类型
    int TotalFieldNum;                      //表的域数
    int SelectedFieldNum;                   //本表被选中的域的个数
    int TotalLen;                           //本表的记录长度
    DBField Fields [FIELDNUM];              //本表各域的描述
    char Selected [FIELDNUM];               //本表各域的选中情况
    }PrimitiveTableUnit;
```

另外,对于在查询过程中出现的中间表,也定义了对应的结构:

```
typedef struct {
    char Status;                    //本表是否处理过
    char TableType;                 //本表是否基本表
    FILE *Handle;                   //文件句柄
    int FieldNum;                   //域数
    int Length;                     //记录总长度
    FieldTypeInt Fields [FIELDNUM]; //域描述
    int loc [FIELDNUM];             //各域的位置
    } TableArrayUnit;
```

2.3 相关的术语定义

当前条件:在算法执行中,指定一个或两个表为当前表.若一个子条件中涉及的域都在这些表中,则称之为当前条件.判断当前表的某个记录组合是否满足条件,仅需对当前条件进行判断,如果它们都为真,结果为真;否则,为假.

无效条件:对于已经使用过的条件,我们对它进行标记,称之为无效条件.

2.4 条件的分解

查询条件可以用逆波兰表达式、前缀表达式、跳转表等表示.但为了便于把条件分解成子条件,即$c_1 \wedge c_2 \wedge \ldots \wedge c_n$的形式,我们采用条件语法树来表示.条件语法树不同于查询语法树,其每个结点的定义如下:

```
struct tu {
    int Type;                       //节点类型
    union {
        struct tu *Left, *right;    //左、右指针(非叶结点时)
        char *valuePointer;         //常数指针
        FieldTypeInt FieldName;     //域代号
          }
        };
```

分解条件有两种方案:一种是把条件表达式分解成合取范式,这样做的复杂性较高,而运行效率也未必高;另一种简化的方法是直接按照现有的形式,把条件表达式分解成$c_1 \wedge c_2 \wedge \ldots \wedge c_n$的形式,其中$c_i(i=1,\ldots,n)$可以是原子条件表达式或最高层由$V$连接的条件表达式.这里,称每个$c_i$为一个子条件.例如,条件表达式$T1$ AND $T2$ AND($T3$ OR($T4$ AND $T5$))可以分解为3个子条件:$T1$,$T2$和$T3$ OR($T4$ AND $T5$).

2.5 具体算法

输入:待查询的基本表,分解后的条件表达式的语法树,目标列表.

输出:由目标列组成的表,表的内容为符合条件表达式的记录.

方法:

(1) 对输入信息进行预处理,填写基本表的内部表;

(2) 判断每个表是否本地表,选出非本地表.对于只涉及非本地表的子条件,把这些非本地表和子条件组合成SQL子查询(或其他协议),传输给相应的服务器;同时,更新内部表结构,并行等待其他服务器上的子查询结果(这一步骤针对联邦数据库,将在第3节"联邦条件下的考虑"详细阐述.);

(3) 对于每一个基本表,进行如下操作:

(3.1) 在条件语法树中,获得此表的当前条件;

(3.2) 建立新表;

(3.3) 标志应该保留的域;

(3.4) 对于本表的每一记录,根据当前条件进行判断.如果为真,则将(3.3)中标志的域写入新表;

(3.5) 更新各种内部表格,标志原表无效;

(4) 如果表数等于1,则结束,最后处理的表为结果表;否则,按下列顺序在未处理过的表中取两个表:

(4.1) 两表之间的连接关系具有等于关系;

(4.2) 两表之间的连接关系具有<,<=,>,>=等关系;

(4.3) 两表之间的连接关系具有其他类型的关系;

(4.4) 两表之间没有任何连接关系;

(5) 在条件语法树中,获得此二表的当前条件;

(6) 建立新表;

(7) 标志应该保留的域;

(8) 对于两表的每一记录的组合,对当前条件进行判断,如果为真,则写入新表;

(9) 更新各种内部表格,使原表无效,把当前条件标志为无效;

(10) 转步骤(4).

本算法步骤(3)的目的是对基本表进行选择、投影,并将结果存入中间表中.这种作法由于先做最底层的选择和投影,减少了后面连接的比较次数.但其代价是牺牲了一定的空间,因其建立了一些中间文件.另一种作法是:取消步骤(3),在读入记录时,先对该记录进行局部条件判断,如果该记录不能满足全部有关它的当前条件,则它不可能满足整个条件,故可以跳过此记录,这样做,直到找到符合条件的记录,才进行下一步.

算法中步骤(3.3)和步骤(7),标志应该保存的域的根据是选中域表或条件树,本表中出现的目标域固然要保留,另外,本表中在未被标志为无效或当前的子条件中出现的域(即将来还要用到的域),也应该保留.这样做,相当于把投影尽量向下移.这一步可以通过查找内部表格和条件语法树来实现.

算法中步骤(4)的作用是通过选择顺序来加快查询的执行速度.几个表的不同顺序的连接可以导致运算时间的巨大差异,例如:设例子中的S,C各有10条记录,而SC有20条记录,现有查询

SELECT * FROM S,C,SC WHERE S.SNO=SC.SNO AND C.CNO=SC.CNO

若先连接S,C,再与SC连接,则要做10*10+10*10*20=2 100次判断;而先连接S,SC,再与C连接,则做10*20+20*10=400次左右就够了.可见有必要对每次连接的表的顺序进行选择.选择表的连接顺序的依据是两表之间的连接关系,等于关系优先,大于、小于等关系次之,其他关系再次之,两表之间没有任何关系(例如:S和C),则放在最后做.

上述算法稍加修改即可利用索引来提高查询速度,进行物理优化.在步骤(4)选取表时,查看在两表的相应域上是否存在索引,如果没有,可根据情况建立索引;在步骤(8)连接表时,根据两表间连接关系的不同和索引情况,采取相应的策略.对连接关系为相等、非相等简单比较,利用索引进行连接;而对其他关系,则需对每一种记录组合进行判断.

2.6 实　例

下面举一个例子来说明该算法的执行过程.在S,C和SC进行如下查询

SELECT SNAME, CNAME, SCORE WHERE C.CNO=SC.CNO AND S.SNO=SC.SNO AND SCORE>80

构造条件语法树如图2所示.

由条件语法树可以得到3个子条件:S.SNO=SC.SNO,C.CNO=SC.CNO和SCORE > 80.

执行过程:

步骤(1)　填写各种内部表格;

步骤(3)　对每个基本表进行选择、投影,这里只对SC有操作,其结果保存在临时表1中,保存的域包括SNO,CNO,SCORE,子条件SCORE>80被标志为无效;

步骤(4.5)　选取两表:S和临时表1;当前条件为:S.SNO=SC.SNO;

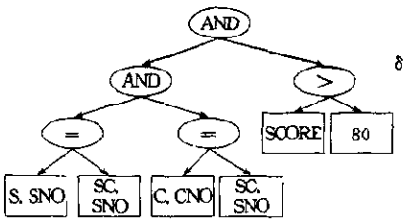

图2

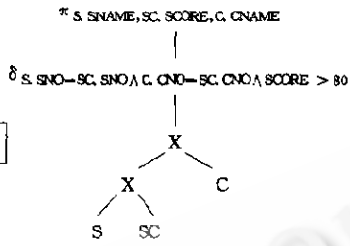

图3　原查询的语法树

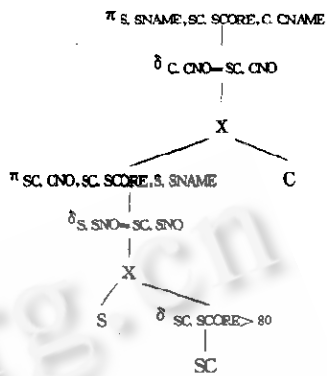

图4　本算法的执行过程构造的等价语法树

步骤(6.7)　建立新的临时表2,应保存的域有 S. SNAME, SC. CNO, SC. SCORE;
步骤(8)　对两表的每一记录的组合进行判断,按上面步骤的域表存入临时表2;
步骤(9)　标志 S. SNO=SC. SNO 无效,更新其他内部表格;
步骤(4.5)　选取两表:S 和临时表1;当前条件为:C. CNO=SC. CNO;
步骤(6.7)　建立新的临时表3,应保存的域有 S. SNAME, SC. SCORE, C. CNAME;
步骤(8)　对两表的每一记录的组合进行判断,按上面步骤的域表存入临时表3;
步骤(9)　标志 C. CNO=SC. CNO 无效,更新其他内部表格;
步骤(10)　结束,临时表3为结果.

一般的查询优化算法通过建立一棵查询语法树来达到把选择、投影下移的目的.本算法没有事先构造查询语法树,它是在查询、连接的同时进行优化.但实际上,其执行过程相当于由底向顶构造同样的一棵查询语法树.上述执行过程即可用下面等价的查询语法树来表示,如图3、4所示.

3　联邦条件下的考虑

由于本查询优化算法是在联邦环境(LNFDBS)下实现的,因此,在这样的联邦数据库系统中,查询优化算法的目标是实现联邦成员之间表的任意组合查询,即通过联邦,将对外地表的查询处理透明化,这样,用户查询外地表就跟查询本地表一样.

联邦查询的实现主要需要解决两个问题:①如何得到数据的分布情况和各个数据库的结构;②在查询算法中如何处理外地数据库.

问题1涉及到联邦数据库系统的实现方式.我们实现的联邦数据库系统 LNFDBS,是采用动态方式来实现的.动态方式就是直接通过一个 DBMS 实时地访问另一个 DBMS 所管理的数据库,当数据格式不一样时,就实时地进行转换.LNFDBS 提供的动态互操作工具 ImportTable 实现了 LNFDBS 对 ORACLE, LNDBMS, FOXPRO 数据库的动态操作.ImportTable 支持模式集成,它将其他数据库系统的表输入到 LNDBMS 中,但不象静态方式那样读入表的数据,而是定义一个输入模式.LNFDBS 根据其输入模式,通过所要操作的另一个数据库系统提供的动态语句来访问数据,将结果返回给动态语句中的主变量(动态语句是指应用程序执行时动态生成和执行的数据库语句).这样就可获得数据的分布情况和各个数据库的结构,用户操作其他异构数据库就象操作本地表一样.ImportTable 不需要真正的数据描述,只需要映射信息.当异构数据库系统修改自身表的结构与数据时,只需要修改相应的映射表的内容,而不影响 LNDBMS 用户对其操作,从而,以动态方式保证了成员的自治性.

问题2可以有多种方案.最简单的一种是:把外地表与本地表同样对待,只是在读取记录时采用不同的方法,一个是向相应的数据库服务器发查询请求,另一个是读取本地文件.这种方法实现简单,但是效率太低.这是因为在每次读取外地表记录时都要进行请求、应答等操作,大量时间消耗在网络通信上;另外,它没有充分利用其它服务器并行进行部分查询工作.

算法中的步骤(2)是实现联邦查询处理的另一种比较好的方案.在进行连接之前,先把外地表分离出来,并且把相应条件分离出来,组成 SQL 子查询语句,发送给相应的服务器,然后进行其它工作,同时等待结果,一旦接收到结果,

即把该结果作为本地表一样使用.这样就可以在本地主机和外地服务器同时进行连接运算,提高了系统的效率.具体的联邦查询过程如图5所示.

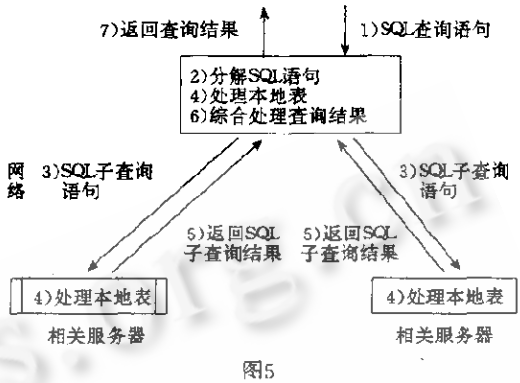

图5

为了提高联邦查询处理的效率,达到查询优化的目的,算法中的步骤(2)还可以进一步改进.由于联邦查询处理涉及若干成员数据库系统,我们遵循系统与系统交互时解决效率问题的3个主要原则:①系统间通信次数尽可能地少;②每次通信传输的数据量尽可能地少;③系统每次通信的执行代价尽可能地小.为了减少与成员数据库系统的通信次数,可以对各个成员数据库系统构造针对该成员的最大子查询,即把原来的几个相关的、涉及同一成员的子查询合并成一个大子查询,结果可能是原来几个查询结果的连接,但该连接由成员数据库系统完成.这样,不但可以减少通信次数,而且减少了中间结果的传递,同时提高了机与机间的并行性.一个用于减少每次通信传输的数据量的方法是采用相关子句间的优化技术(跨事务缓冲),即充分利用上一次操作的结果.但这需要一定的缓冲区来保存上一次的操作信息和结果信息.假设有两个查询操作 Q_1 和 Q_2,Q_1 先于 Q_2 发生,则进行 Q_2 时,首先判断能否利用 Q_1 的结果.可能有如下几种情况:①Q_1 包含 Q_2,则 Q_2 不必再通过网络发生;②Q_2 包含 Q_1,则可进行差优化,在 Q_2 的条件中去掉 Q_1 的条件;③Q_1 与 Q_2 相交,我们同样可以对其进行差优化,在 Q_2 的条件中去掉 Q_1 与 Q_2 相交的那一部分;④Q_1 与 Q_2 无关,这时直接把 Q_2 发送给相应的成员处理.通信的执行代价则与具体的成员数据库系统有关,在此不作讨论.

4 结束语

中山大学软件研究所开发的岭南联邦数据库系统 LNFDBS 是 CLIENT/SERVER 体系结构的联邦数据库系统.目前,我们已经实现以 ORACLE,FOXPRO,LNDBMS(我们自行开发的面向数据采集的数据库管理系统)作为 LNFDBS 中的3个节点,而且我们正在不断地增加新的联邦节点,如 SQL SERVER,INFORMIX 等.

我们的查询优化算法实现于 LNFDBS 上.LNFDBS 采用的是 CLIENT/SERVER 体系结构,其主体进程运行 DEC VAX 机上,SERVER 一端的进程可以通过传递消息来申请主进程的服务.查询优化算法是 LNFDBS 主进程的一个组成部分.目前,我们采用上述算法,已经实现任意个表的任意条件查询,并且已经实现与 ORACLE,FOXPRO 的联邦查询,还进行了并行性、嵌套查询等方面的研究.

参考文献

1 Jefferey D Ullman. Principles of Database Systems. New York: Computer Science Press, 1982

2 肖永桥.异构多数据库互操作性——联邦数据库方法研究[硕士论文].中山大学,1995
(Xiao Yong-qiao. Heterogeneous multi-database interactivity——federated database survey[M. S. thesis]. Zhongshan University, 1995)

3 肖永桥,姚卿达.联邦数据库系统的研究.计算机科学,1994,21(6):55~57
(Xiao Yong-qiao, Yao Qingda. On federated database system. Computer Science, 1994,21(6):55~57)

4 姚卿达,肖永桥,陈晓蕾.一个高效的面向数据采集的数据库管理系统.软件学报,1996,7(增刊):254~260
(Yao Qingda, Xiao Yong-qiao, Chen Xiao-heng. A data collection oriented DBMS with high performance. Journal of Software, 1996,7(supplement):254~260)

An Algorithm of Database Query Optimization Adapted in Federal Database System——LNFDBS

YAO Qing-da HE Xin HUANG Xiao-chun LI Zhi-fang

(*Lingnan College Zhongshan University Guangzhou* 510275)

Abstract Query is the core function of a Database Management System. A powerful and effective query algorithm is the key of a successful database system. In the paper, an algorithm of database query optimization is introduced and is implemented on a federal database system——LNFDBS(Lingnan federal database system). The adaptation to the federal database system is also discussed.

Key words Query optimization, join, project, restriction, federal database system.