# 一个有效处理高价选择谓词的查询优化算法

阳国贵,吴泉源
(国防科技大学计算机学院,长沙 410073)

**摘 要**: 文中讨论了对象关系数据库查询优化中所面临的新问题之一,即如何对查询中所包含的高价选择谓词进行优化. 在经典的关系优化方法中,选择操作的代价较低,对选择操作均采用下推的方式,即让选择操作靠近选择关系,以图缩小参加后续连接运算的关系的规模(元组数). 但在对象关系数据库中,由于选择操作本身就十分费时,因此,选择操作下推的前提条件已不成立,需要对此进行专门研究. 在介绍了已有解决方法并分析其主要弊端后,文中提出了一个新的查询优化算法 Predicate-Rolling-Up.

**关键词**: 查询优化; 连接算法; 高价谓词; 对象关系数据库
**中图分类号**: TP392 **文献标识码**: A **文章编号**: 0372-2112 (2001) 02-0182-04

# A New Optimization Algorithm for Queries with Expensive Selections

YANG Guo-gui, WU Quan-yuan
*(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)*

Abstract: In this paper, one of the new problems encountered in the query optimization of ORDB (object relational data base), that is, how to optimize the expensive predicates contained in the query, is discussed. In traditional query optimizers, selections have been handled by "pushdown" rules under the assumption that selection can be processed with little time or no time, and selections are processed immediately after the scan of the relation, so as to diminish the size of relations to be joined. In ORDB, however, complex methods can be embedded in selections, thus selections may take significant amount of time, and the assumption for "pushdown" is no longer sound, so attention must be paid on this new problem. Based on the analysis of some solutions, a new algorithm Predicate-Rolling-Up is presented.

Key words: query optimization; join algorithm; expensive predicate; ORDB

## 1 引言

在关系数据库中,关系的查询优化技术侧重于连接方法和连接次序的确定,而对各种限制条件通常采用"选择条件下推"的启发式规则加以处理,以图尽早减少参加连接运算的关系表的大小,这样的启发式规则在选择操作的处理代价很低(甚至零代价)的情况下是十分合适的,然而,在对象关系数据库、面向对象数据库中就必须慎重使用了,这是由于查询限制条件和连接谓词中可能包含用户自定义的计算昂贵的函数(Expensive Function),这样对关系表的限定操作将变得和连接运算一样十分费时,甚至比连接运算更费时,这样原有的"下推"式策略不再有效. 通常把含有高价函数的选择谓词称为高价限定谓词(Expensive Restricted Predicate)简称高价谓词.

如查询 select $e$.name, $f$.name from emp $e$, emp $f$ where $e$.salary = $f$.salary and redness ($e$.picture) < 0.1 and redness($f$.picture) < 0.1

由于 redness 函数的执行代价较大,盲目使用选择操作"下推"将使 redness 函数的执行次数较多,可能不如把它放到连接操作之后(连接操作的结果集较小).

Michael Stonebraker[1] 在 postgres 的多层存贮结构(multi-level store)下,首先提出了高价谓词优化问题,当时针对的主要应用背景是 Project Sequoia,它主要管理 GIS(Geographic Information System)数据,在对该类数据进行处理中,需要处理许多含有高价函数的查询语句. Joseph M. Hellerstein 和 Michael Stonebraker 提出了所谓谓词迁移(Predicate Migration)算法[2]来确定高价选择谓词在查询规划中的适当位置,随后又对该算法进行了一些改进[3,4]. 但是,谓词迁移算法由于计算的不精确难于获得最优的执行规划,并且由于"不可剪枝的子查询"(unprunable subplan)的采用,使得它所基于的动态规划方法易于陷入穷举,不仅增加了所需保存的中间规划的数目,而且降低了算法的性能. 为了克服上述不足,Surajit Chaudhuri 提出了一种新的优化方法[5],该方法对高价选择谓词在执行规划中所处位置的各种情况都进行搜索、比较,从中挑选出最优的执行规划,但该方法的不足在于它扩大了优化方法的搜索空间. 为此,本文将提出一种新的能处理高价谓词的优化方法,它是

收稿日期:2000-03-01;修回日期:2000-10-08

一种基于动态规划的两阶段优化方法,它利用 System R 中的动态规划优化算法为连接运算产生连接次序,然后,根据高价选择谓词的 Rank 值,把它们放置到执行规划中的适当位置.该方法的开销小,且能产生满意的执行规划.

本文将在第二部分讨论单表上高价谓词的优化处理问题,第三部分具体介绍基于动态规划和 Rank 值的 Predicate_Rolling_Up 算法,最后是性能对比及展望.

## 2　单表上高价选择谓词的优化处理

关系 $R$ 上的选择操作 $\sigma p(R)$,其选择谓词的合取形式为 $p = e1 \wedge e2 \wedge \cdots \wedge en$(有时也称 $ei$ 为布尔因子),对关系 $R$ 中的每个元组,完成 $e1, e2, \cdots, en$ 求值的执行时间分别为 $t1, t2, \cdots, tn$,而 $R$ 中的元组满足布尔因子 $e1, e2, \cdots, en$ 的可能性分别为 $p1, p2, \cdots, pn$(或称 $e1, e2, \cdots, en$ 的选择率),这样,按从左到右依次计算 $ei$ 时,执行 $\sigma p(R)$ 所需的时间为:

$$T = t_1 \times R + t_2 \times p_1 \times R + \cdots t_n \times \prod_{i=1}^{n-1} P_i \times R$$

为此,优化的任务就是要寻找一种合适的求值序列(对应到 $e1, e2, \cdots, en$ 的一种置换)使 $T$ 为最少.该优化问题可以采用高价选择谓词的 Rank(秩)值概念来加以解决,Rank 最先由 Michael Z. Hanani[6]提出,它刻划了选择谓词的选择率和计算选择谓词的代价两方面的性质.

**定义 1**　选择谓词的 Rank 值 $Rank = (p - 1)/t$,其中,$t$ 为该谓词对每个元组进行求值所需的处理代价(时间),$p$ 为该选择谓词的选择率.

**定理 1**　对于关系上的选择操作 $\sigma p(R)$,$p = e_1 \wedge e_2 \wedge \cdots \wedge e_n$ 而言,分别计算每个布尔因子的 Rank 值,根据这些 Rank 值的升序,可获得相应布尔因子的一个排列次序 $S$,当布尔因子 $e_1, e_2, \cdots, e_n$ 的求值次序按 $S$ 中所确定的次序进行时,$T$ 将最少.

证明:限于篇幅,具体证明略.

该定理的直观解释是,对执行时间相同的两因子而言,谁的选择率低,谁就先执行,这样显然可以减少参与后续操作的元组数目;同样,对选择率(小于 1 时)相同的两因子而言,谁所需的执行时间短,谁就先执行.

## 3　基于动态规划和 Rank 值的 Predicate_Rolling_Up 优化算法

动态规划优化算法是关系查询优化中一个非常出色的好算法,能为连接运算确定连接次序和为连接操作选择具体的执行算法,另一方面,在第二部分已经指出,基于 Rank 值概念,可以非常好地解决单表上选择运算的执行次序.因此,能否把连接运算也看成选择运算,来优化选择操作在执行规划中的位置呢?但由于连接运算为二元运算,若把它看成连接关系笛卡尔积上的选择运算的话,则连接运算的代价应当是该笛卡尔积的线性函数,但正如 Hellerstein 指出的那样,连接运算的代价可用连接关系元组个数的线性函数来表达[3,4],而不是其笛卡尔积的线性函数.为此,需要把连接运算的 Rank 值分为两个,分别对应到两个连接关系,即内外关系.并对连接运算的选择率进行同样的变化,即对内外关系分别求出选择率,这样,有了内外关系上的选择率和代价公式,就可以分别求出连接运算对内外关系的 Rank 值,依据其对内外关系的 Rank 值,就可以解决内外关系上的选择谓词是否应当放到连接运算之后的问题了.

**定义 2**　对关系 $R$ 与关系 $S$ 的连接运算而言,连接运算的选择率 $Sj$ = 连接结果的元组数/[(关系 $R$ 的元组数) ×(关系 $S$ 的元组数)].

**定义 3**　对关系 $R$ 与关系 $S$ 的连接运算而言,连接运算对关系 $R$ 的选择率 $Sr = Sj \times |S|$,对关系 $S$ 的选择率 $Ss = Sj \times |R|$.其中,$|S|$ 和 $|R|$ 分别表示关系 $S$ 和 $R$ 的元组数.

值得特别指出的是,定义 3 中对连接运算的内外关系上选择率的定义隐含高价选择操作的代价是元组个数的线性函数,即将选择操作作用到每一个元组之上,而不是采用函数值索引或函数值缓冲(Function Caching)技术来降低高价选择谓词的操作次数.因为若采用函数值缓冲就可消除重复求值,如 $P$ 为 $f(R.a) > 1$,则仅需对不同的 $R.a$ 进行 $f$ 函数的运算即可,而不需对具有相同 $R.a$ 值的不同元组进行重复求值.但采用函数值缓冲将带来其他系统开销,甚至得不偿失.因此,定义 3 是合理的.

这样,当为连接运算分别求出其对内外关系的 Rank 值 Rank_1 和 Rank_2 后,就可将 Rank_1 与内关系上的选择操作的 Rank 值进行比较,使 Rank 值大于 Rank_1 的那些选择操作位于该连接运算之后,对外关系上的选择操作也可按同样的方式进行.但是,连接运算的次序是由动态规划优化算法确定的,且连接运算的次序与各连接运算的 Rank 值的升序可能是不一致的,如对连接运算 $\sigma p(A) \times_1 B \times_2 C$ 而言,若动态规划优化算法产生的连接次序为 $(\sigma p(A) \times_1 B) \times_2 C$,且选择操作 $\sigma p$ 的 Rank 值小于连接操作 $\times_1$ 对外关系的 Rank 值,但大于连续操作 $\times_2$ 对外关系的 Rank 值,此时,对部分执行规划 $\sigma p(A) \times_1 B$ 而言,选择操作应该先执行,但对整个执行规划而言,由于有连接操作 $\times_2$ 的存在,选择操作也许应当最后执行(使得整个执行规划的代价更低),因此,根据什么原则把高价选择操作安排到执行规划中的合适位置,就是一个需要考虑全局才能解决好的问题.由于连接运算具有结合律性质,因此,当把不是按 Rank 值的升序排列的多个连接运算看成一个复合结点,并为这个复合结点计算 Rank 值,使复合结点扮演与普通连接结点一样的角色时,就可为选择操作在整个执行规划中找到一个合适的位置,从而获得一个更好的执行规划.

**定义 4**　左深树(Left Deep Tree)中相邻两连接运算 J1 和 J2 复合后,其复合结点的 Rank 值 $Rank(J1J2)$ 为:

$$Rank(J1J2) = \frac{selectivity(J1J2) - 1}{cost(J1J2)} = \frac{selectivity(J1) * selectivity(J2) - 1}{cost(J1) + selectivity(J1) * cost(J2)}$$

上述定义和计算公式的合理性在于连接运算 $J1$ 和 $J2$ 是相邻的,且 $J1$ 必先于 $J2$ 执行.

**定理 2**　连接运算的复合 Rank 值计算满足结合律,即 $Rank((J1J2)J3) = Rank(J1(J2J3))$.

证明:限于篇幅,具体证明略.该定理表明,复合连接运算

$(J1J2J3)$的 Rank 值可从$(J1J2)$和 $J3$ 或由 $J1$ 和$(J2J3)$得到.

**定理 3** 对左深树中相邻两连接运算的复合 Rank 值 Rank$(J1J2)$而言,当 Rank$(J1)$ > Rank$(J2)$时,Rank$(J2)$ < Rank$(J1J2)$ < Rank$(J1)$.

证明:限于篇幅,具体证明略.

该定理表明,对相邻的连接运算(从叶往根方向)$J1$ 和 $J2$ 而言,若 $J1$ 的 Rank 值较大,有可能阻止 $J1$ 下的选择操作上移时,$(J1J2)$将小于 $J1$ 的 Rank 值,使得某些选择操作可以上移了,从而在更大的子执行规划内来考虑选择操作的位置,达到优化的目的. 如对前面提到的$(\sigma p(A) \times_1 B) \times_2 C$,若 Rank$(p)$ > Rank$(\times_1 \times_2)$时,将使选择操作上移,得到更好的执行规划 $\sigma p((A \times_1 B) \times_2 C)$.

有了上述准备之后,就好理解基于动态规划和 Rank 值的 Predicate_Rolling_Up 优化算法了,它由两阶段组成,第一阶段,利用 System $R$ 中的动态规划算法来确定连接算法和连接次序,在此基础上,计算连接运算对连接关系的 Rank 值和关系上高价选择操作的 Rank 值,依据定理 1 来确定这些选择运算在执行规划中的合适位置. 当执行规划中的连接运算次序与对应的 Rank 值的次序(升序)不一致时,将依据连接运算的可结合性,采用将两个或多个连接运算看成一个复合连接运算的办法,依据定义 3 计算出复合连接运算的 Rank 值. 由定理 3 可知,复合 Rank 值将小于其中较大的那一个,因此,通过上述方法,可以使得规划中的连接运算(或复合连接运算)的次序与其对应的 Rank 值的次序(升序)一致,从而为高价选择操作确定合适的位置.

限于篇幅,这里不对 System $R$ 中的动态规划优化算法[7]进行介绍了. 但需要特别指出的是,为避免谓词迁移算法中由于采用"不可剪枝子查询"(unprunable subplan)而可能出现的穷举问题,Predicate_Rolling_Up 优化算法将加进一条重要的启发式规则,即在左深树形成过程中的每个阶段,如果基关系上的选择操作(按 Rank 值排序)的 Rank 值大于其上的连接操作的 Rank 值时,就把选择操作上移,使得动态规划优化算法选出的解为一满意解.

下面主要介绍算法的第二阶段,即如何解决高价选择操作在执行规划中的位置问题. 前面对解决这一问题的思路进行了讨论,基本思想是把连接运算看成复合连接运算,使最后的连接运算(或复合连接运算)的次序与 Rank 值的升序一致. 算法采用的数据结构和执行过程可用下图加以说明:

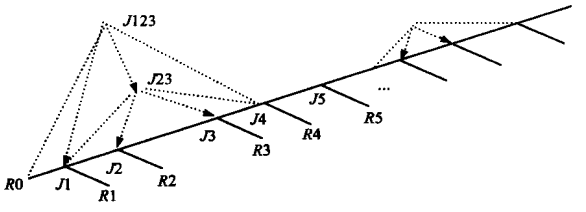


图 1 数据结构与执行过程示例

图中 $Rx$ 表示某个连接关系,而 $Jx$ 为某个连接运算. 并且区分两类连接结点,一类为基结点,即第一阶段生成的规划树中的结点,另一类为第二阶段中形成的一些临时性复合结点,结点类型由 Type 标识. 每个结点中有指向左右孩子的指针 Left_Child 和 Right_Child,在基结点中,有指向父结点的指针 Base_Parent 和指向复合结点的指针 Composite_Parent,而在复合结点中,为指向左右结点的 Left_Order 和 Right_Order 指针.

当 Rank$(J2)$ > Rank$(J3)$时,由于 $J2$ 的出现,阻止了那些其 Rank 值小于 Rank$(J2)$但大于 Rank$(J3)$的高价选择操作的上移,为此,把 $J2$ 和 $J3$ 看成一个复合连接操作,形成新结点 $J23$,计算并保存该结点的复合计算代价、复合选择率及 Rank 值. 在 $J23$ 中,有指向 $J1$、$J4$ 的指针(指向 $J1$ 的为左序指针 Left_Order,指向 $J4$ 的为右序指针 Right_Order),也有指向 $J2$、$J3$ 的指针(指向 $J2$ 的为左孩子指针 Left_Child,指向 $J3$ 的为右孩子指针 Right_Child). 而 $J2$ 中既有指向原有父结点 $J3$ 的指针 Base_Parent,也有指向新结点 $J23$ 的指针 Composite_Parent. 形成复合结点后,先与该结点的左结点进行比较,假定 Rank$(J23)$ < Rank$(J1)$,此时,需要按上述过程形成新的复合结点 $J123$. 上述过程直至复合结点的前一个结点的 Rank 值小于该复合结点的 Rank 值为止. 当遇到左序结点为 $R0$ 时,复合结点将与其右结点进行比较,直至复合结点的 Rank 值小于右边结点的 Rank 值为止. 该过程将由算法中的 form_composite_node() 子过程完成.

随后,将考查当前结点的 Right_Order 指向的下一结点,重复上述过程,直到所有连接结点(或复合结点)均按 Rank 值的升序排列为止. 连接运算(或复合连接运算)按 Rank 值升序排序后,根据定理 1,高价选择操作谓词在执行规划中的位置也就容易确定了.

下面给出算法描述,其中算法的输入为连接关系集合 $R1, \cdots Rn$,各关系上的高价谓词集合 Predicates,算法的输出为优化的执行规划树. 全局变量 Root 指向规划树最上层的那个连接结点,Tail 指向规划树中最低的那个连接结点,current 指向正待处理的那个结点,start_point 指向 for 循环中下一起始结点.

```
predicate_rolling_up_algorithm(R1, …, Rn, Predicates)
{ Root = DP_algorithm(R1, …, Rn, Predicates);
   current = start_point = Tail;
   current_R = start_point -> left_child;
   if (expensive_predicates(current_R) > 0){
      while current ≠end do{form_composite_node(current);
                  current = get_next_of(current);   }
      sort_place_predicates(current_R,predicates);
      }/ *end of if  */
   for(i = 1; i <= N; i++)
      {current_R = start_point -> right_child;
   current = start_point;
      current.rank = right_rank_of(current);
      if (expensive_predicates(current_R) > 0){
         form_composite_node(current);
```

```
        current = get_next_of(current);
        while(is_convergence_point(current) == FALSE) do{
                                form_composite_node(current);
                               current = get_next_of(current);}
        attach_convergence_point();
        sort_place_predicates(current_R.predicates);}/ *end of
if */
    start_point = start_point -> Base_Parent;}/ *end of for */
}/ *end of Predicate_Rolling_Up */
form_composite_node(current)
{next = get_next_of(current);
  if (rank(current) > rank(next)) then
  {new_p = form_a_new_node(current,next);
       insert_in_proper_position(new_p);
       if (new_p.left_Order ≠current_R) then current = get_
pre_of(new_p)
     else current = new_p;
  next = get_next_of (current);
  if (rank(current) > rank(next)) then form_composite_node(cur-
rent);}
}
```

算法中,对 $R0$ 上的高价谓词进行了单独处理,因为对 $R0$ 而言,在与它的父结点 $J1$ 进行比较时,$J1$ 要计算它对外关系的 Rank 值,但处理其他关系时,与它们的父结点进行比较时,父结点的 Rank 值应为对内关系的 Rank 值,如对 $R1$ 而言,$J1$ 中要用到的将是它对内关系的 Rank 值.但不管对哪种情况,随后的所有其它连接运算都使用其对外关系的 Rank 值.

算法中的 get_pre_of()以及 get_next_of()分别为取当前结点的前序(Left_Order)和后序(Right_Order)结点.值得指出的是,在 for 循环中,初始点的不同,将有可能形成一部分不同的复合结点,但在某结点 $Jk$ 之后,却是相同的,称这样的结点 $Jk$ 为汇合点结点,为此,在 form_composite_node()之后,取当前结点的下一结点,如该结点的 Composite_Parent 为空,则可断定该结点即为汇合点.确定为汇合点后,只需把上次所形成的后续次序链添加到本次所形成的次序链,及早结束本次 for 循环,从而改进算法性能.

## 4　性能对比及展望

当连接关系个数为 $N$ 时,算法调用 form_composite_node 一次所形成的复合结点的个数最多为 $N-1$ 个,而算法中 For 循环所导致的内层调用次数为 $N$,所以,本算法中用于处理高价选择谓词的计算复杂性的上界为 $N$ 的平方阶.而调用动态执行规划算法的计算复杂性与 System $R$ 中的相同.

当 Predicate_Rolling_Up 算法与处理高价选择谓词的 PushDown、PullUp 和 Predicate Migration 算法进行对比分析时,可以看到:PushDown 能对单表进行优化处理,但有多表连接时,性能很差.PullUp 在关系表比较小,且选择谓词相当复杂时,性能较好;反之,当选择谓词代价较低时,性能表现较差.Predicate Migration 和 Predicate_Rolling_Up 算法通常优于 PushDown 和 PullUp 算法.而 Predicate_Rolling_Up 中由于避免了穷举,并利用了汇合点机制,性能将好于 Predicate Migration.需要指出的是,不管是谓词迁移算法、Predicate_Rolling_Up 算法,还是基于代价的其他优化算法,其产生的执行规划的优劣均与选择率、选择操作的执行代价、连接算法的执行代价等因素有关,为此,在实际系统中,如何准确估计高价选择谓词的选择率和执行代价也就显得十分重要.

本文讨论的是对高价选择谓词进行优化处理的一般办法,在某些情况下,也可将其转化为一般的选择谓词进行处理.如对选择谓词 $P(R.c)\ \vartheta C$ 而言,$P$ 是用户定义的复杂函数,为算术比较运算符($<$、$\leq$、$>$、$\geq$、$=$、$\neq$),$C$ 为常量,若存在建立在 $P(R.c)$ 上的函数索引时,就可把 $P(R.c)\ \vartheta C$ 看成一般的选择谓词.有关函数索引的建立、维护与管理与一般的索引类似,通过索引查找的办法找到满足选择条件的元组比对每个元组进行测试来确定是否满足选择条件的做法可能更可取.当然,与一般索引一样,索引的维护会带来其他开销.除了考虑高价选择谓词与连接运算间的关系外,事实上,还可以对其他运算中包含高价谓词的情况进行研究,如对关系的差、并和交运算等,这里给出如下的启发式规则:

$$\sigma_A(R) \cap S = \sigma_A(R \cap S) \qquad \sigma_A(R) - S = \sigma_A(R - S)$$

$$\sigma_A(R) \cup \sigma_A(S) = \sigma_A(R \cup S)$$

当 $A$ 为高价选择谓词时,上述转换公式应当是有效的,直观上讲,它们减少了高价选择谓词求值的次数.

**参考文献**:


[1] Michael Stonebraker. Managing Persistent Objects in a Multi-Level Store [M],ACM SIGMOD,June 1991.

[2] Joseph M. Hellerstein. Michael Stonebraker. Predicate Migration: Optimizing Queries with Expensive Predicates [M]. ACM SIGMOD 5/1993:267 - 276.

[3] Joseph M. Hellerstein. Practical Predicate Placement [M]. ACM SIGMOD 5/1994:325 - 335.

[4] Joseph M. Hellerstein. Optimization techniques for queries with expensive methods [J],1998,TODS(2):113 - 157.

[5] Surajit Chaudhuri,et al. Optimization of queries with user-defined predicates [A],22nd VLDB [C],1996.

[6] Michael Z. Hanani. An Optimal evaluation of boolean expressions in an online query system [J]. Commun. May 1977,ACM 20,5:344 - 347.

[7] P. G. Selinger,et al. Access Path Selection in a Relational Database Management System [M]. ACM SIGMOD,June 1979.


**作者简介**:


**阳国贵**　1985 年毕业于中南矿冶学院计算机专业,获学士学位,1988 年毕业于中国人民大学信息系,获硕士学位.后到国防科技大学工作,从事数据库技术的研究与教学,先后发表学术论文三十余篇,主编或参与出版数据库方面的著作三部.主要研究兴趣包括面向对象数据库、并行数据库、对象关系数据库等.