面向对象数据库中“联系-选择”复合操作的优化*

吴胜利 王能斌

(中国科学院软件研究所 北京 100080) (东南大学计算机科学与工程系 南京 210096)

摘要 在面向对象数据库中,“联系-选择”复合操作的功能类似于嵌套关系数据库中多个嵌套关系的连接,是查询优化时需要重点考虑的问题.本文首先证明满足一定条件限制的“联系-选择”复合操作可在多项式时间内得到最佳查询方案,然后给出了支持双向联系时“联系-选择”复合操作的一些优化算法.模拟实验表明,文中所提出的优化算法性能平稳,效果很好.

关键词 面向对象数据库,查询优化,“联系-选择”复合操作.

中图法分类号 TP311.13

在面向对象数据库中,“联系-选择”复合操作 AS(associate-select)与关系模型中多个关系的连接功能类似,执行代价大,优化困难(需指数级时间才能找到最佳查询方案),且两者均应用频繁,故需要重点探讨.

在文献[1]中,我们采用相同的原理,对“联系-选择-投影”和“联系-选择-连接-投影”两种复合操作进行了讨论,并给出了实现算法.算法思想较简单,易于实现,但对“联系-选择-投影”操作而言,存在一些不足之处:① “联系-选择-投影”比“联系-选择-连接-投影”更特殊,存在效果更佳的实现算法;② 谓词的费时程度要由用户说明,不够客观.因此值得进一步探讨.

本文首先证明满足一定条件限制的 AS 查询,可在多项式时间内得到最佳查询方案,然后再改变某些条件讨论.最后对文中所提出的算法给出模拟实验的结果.文中的一些术语及约定同文献[1].投影操作已在文献[1]中讨论,本文从略,只涉及“联系-选择”,即 AS 操作.

1 树型查询与最佳查询方案

本文限于讨论非递归查询,此时查询结构呈树型.

定义 1. 查询树 T 是一棵树,具有零个或零个以上结点.对于树中任一结点 R_i,其上有 3 种参数,谓词选择率 $S(R_i)$、对象个数 $N(R_i)$和单位代价 $C(R_i)$,$0<S(R_i)\leqslant 1$,$N(R_i)>0$,C

---

* 作者吴胜利,1963 年生,博士,副研究员,主要研究领域为数据库与信息系统.王能斌,1929 年生,教授,博士导师,主要研究领域为数据库与信息系统.

本文通讯联系人:吴胜利,北京 100080,中国科学院软件研究所

本文 1997-02-19 收到修改稿

$(R_i)>0$. 此外,整个查询树还具有参数 N,表示查询涉及的对象总个数,N 为正整数.

对上述定义中的几个参数略作说明. 对 T 的查询可用如下的联系代数表达式[2~4]表示

$$\sigma_m(\ldots\sigma_2(\sigma_1(R_1)[P_1]*R_2)[P_2]*\ldots*R_m)[P_m] \tag{1}$$

$N(R_i)$ 为结点 R_i 所对应的对象数,$S(R_i)$ 为 R_i 中所对应的谓词 P_i 的选择率,$C(R_i)$ 为对 R_i 中一个对象进行相应操作的平均单位代价. 对于 T 中根结点只进行选择操作;对于 T 中其它结点,先进行联系操作再进行选择操作. 另外,显然有 $N=N(R_1)$.

定义 2. 对查询树 T 进行一次有效查询,亦即遍历 T 中所有的结点,并且需满足下列限制:在访问 T 中任一结点 R_i 之前,R_i 的祖先结点均已被访问.

定义 3. 对于查询树 T 的每一个有效查询,均具有相应的查询代价. 设查询树 T 具有 m 个结点,有效查询对该查询树的遍历次序是 $R_1,R_2,\ldots,R_m$,则相应的查询代价为

$$C(T)=N*(C(R_1)+S(R_1)*C(R_2)+\ldots+S(R_1)*\ldots*S(R_{i-1})*C(R_i)$$
$$+\ldots+S(R_1)*\ldots*S(R_{m-1})*C(R_m)) \tag{2}$$

(2)式作为"联系-选择"复合操作的代价是准确的[4]. 由于对于一特定的树 T 而言,N 是不变的,为方便起见,下面将(2)式中的 N 去掉,这对于讨论如何找出最佳查询方案没有影响. 现在问题是,对于一特定的查询树,如何找出一有效查询,使其具有最小的查询代价.

一般而言,上述代价函数 C 可递归定义为:

$C(\wedge)=0$,当查询树 T 不含任何结点;

$C(T)=C(R_1)$,当查询树 T 仅含一个结点 R_1;

$C(B_1B_2)=C(B_1)+S(B_1)*C(B_2)$,当对 T 的有效查询树包含 B_1 和 B_2 两个非空结点序列. 其中非空结点序列 B 的选择率用下面的方法计算:设 B 包含结点 $R_1,R_2,\ldots,R_m$,则 $S(B)=S(R_1)*S(R_2)*\ldots*S(R_m)$.

容易验证,上述代价函数 C 的定义是一致的,亦即如果有 $B_1B_2=B_1'B_2'$,则 $C(B_1B_2)=C(B_1'B_2')$. 此外,由

$$\begin{aligned}C(B_1B_2)-C(B_2B_1)&=C(B_1)+S(B_1)*C(B_2)-C(B_2)-S(B_2)*C(B_1)\\&=C(B_2)*[S(B_1)-1]-C(B_1)*[S(B_2)-1]\\&=C(B_1)*C(B_2)*[(S(B_1)-1)/C(B_1)-(S(B_2)-1)/C(B_2)]\end{aligned}$$

可得 $C(B_1B_2)\leqslant C(B_2B_1)$,当且仅当 $r(B_1)\leqslant r(B_2)$,这里 $r(B)$ 称作 B 的秩(B 不为空,即 $B\neq\wedge$). $r(B)$ 的定义为

$$r(B)=(S(B)-1)/C(B) \tag{3}$$

下面讨论如何得到查询树 T 的最佳运算次序. 先看 3 种最简单的情形:

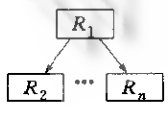

图1 情形2中树结构

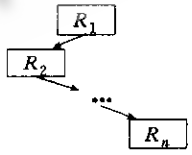

图2 情形3中树结构

(1)若树 T 只包含一个结点 R_1,则最佳运算次序即为 R_1;

(2)若树 T 包含结点 $R_1,R_2,\ldots,R_n(n\geqslant 2)$,$R_1$ 为根,$R_2,\ldots,R_n$ 为 R_1 的子女,如图 1 所示,则对 $R_2,\ldots,R_n$ 按秩从小到大排序,结果为 $R_2',\ldots,R_n'$. 则最佳运算次序为 $R_1',R_2',\ldots,R_n'$;

(3)若树 T 包含结点 $R_1,R_2,\ldots,R_n(n\geqslant 2)$,$R_1$ 是根,R_2 是 R_1 的子女,$\ldots$,R_n 是 R_{n-1} 的子女,如图 2 所示,则最佳运算次序为 $R_1,R_2,\ldots,R_n$.

上述第 2 种情形所得最佳运算次序中,R_1 的秩可能大于 $R_2',\ldots,R_n'$ 中任一个的秩. 第

3 种情形中 $R_1,R_2,\ldots,R_n$ 中任一个排在前面的 $R_i(1\leqslant i\leqslant n-1)$ 的秩都可能大于后面的 R_j $(i<j\leqslant n)$ 的秩. 这时我们引入结点序列 $R_1,R_2,\ldots,R_n$ 规范化的概念.

定义 4. 对结点序列 $R_1,R_2,\ldots,R_n$，如果对任意的 $R_i(1\leqslant i\leqslant n-1)$ 和 $R_j(i\leqslant j\leqslant n)$，有 $r(R_i)\leqslant r(R_j)$，则称 $R_1,R_2,\ldots,R_n$ 为规范化结点序列.

对结点序列 $R_1,R_2,\ldots,R_n$ 的规范化由下述过程实现：

```
procedure NORMALIZE(T)
//T 为结点序列 R1,R2,...,Rn(n≥2)
  while T 中有相邻的结点 Ri,Ri+1且 r(Ri)>r(Ri+1) do
    将 Ri,Ri+1合并成 Ri', C(Ri')=C(Ri)+S(Ri)*C(Ri+1),S(Ri')=S(Ri)*S(Ri+1);
  end of while;
end procedure.
```

对结点序列 T 调用该过程得到一个新的结点序列 T'：$R_1',R_2',\ldots,R_m'(m\leqslant n)$. 只要在 NORMALIZE 过程中记住结点合并前后的情况，则可得到 T' 与 T 的对应关系.

引入规范化概念后，再看一种简单情况：

(4) 如果 T 以 R_1 为根，$R_2,\ldots,R_n(n>2)$ 均是 R_1 的子女，$R_2,\ldots,R_n$ 为根所对应的子树(不妨设为 $T_2,\ldots,T_n$)均符合上述第 3 种情形，并且与 $T_2,\ldots,T_n$ 相对应的最佳结点序列均是规范化的. 此时树结构如图 3 所示，则 T 的最佳运算次序用下面方法取得：

(a) R_1 总是排在首位；

(b) 将子树 $R_2,\ldots,R_n$ 所对应的结点序列分别放进 $n-1$ 个有序集合 $S_2,\ldots,S_n$ 中；

(c) 按 $n-1$ 路合并排序之法对 $S_2,\ldots,S_n$ 中的结点排序.

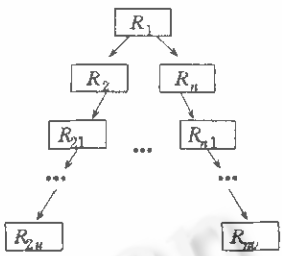

图3　情形4中树结构

对于任意复杂的查询树 T，我们从叶结点开始处理，逐步上升，则所遇到的子树必为上述 4 种简单情况之一，得到相应结点序列的最佳运算次序，然后对之进行规范化处理，直至根结点为止. 可用算法 1 描述.

算法 1. 确定查询树 T 的最佳运算次序

输入：查询树 T 与 T 中每一个结点 $R_i(1\leqslant i\leqslant m)$ 相对应的 $S(R_i)$ 和 $C(R_i)$

输出：R 的最佳运算次序($R_1,R_2,\ldots,R_m$ 的一种排序)

步骤：

(1) 按广度优先遍历查询树 T，将其中出度不小于 2 的内结点放入栈中；

(2) 反复从栈 F 中弹出结点 R_i 做(3)，直至栈空；

(3) 调用过程 NORMALIZE 规范化 R_i 的每一个子树所对应的最佳结点序列，然后根据秩的大小将它们合并成一个新的结点序列.

(4) 根据记载的合并情况将合并所形成的结点还原为原来的结点.

一棵有 m 个结点的树，其内结点不超过 $m/2$ 个，对于每一个内结点，其处理过程不超过 $O(m)$，故算法最坏性能不超过 $O(m^2)$，其平均性能为 $O(m\log m)$.

在以上的讨论中，对查询树中任一个结点而言，其联系与选择操作捆绑在一起，不能分开，这对于选择谓词费时甚微的情形是非常合适的. 具体作法就是在用联系操作生成每一类对象时，随即执行选择谓词决定取舍. 但是如果选择谓词非常费时，再这样处理就不行了，例如对图 4 所示的查询树 T，N 表示对象总的个数，$S(B)$ 表示 B 结点中谓词选择率，$M\_C(B)$ 和 $P\_C(B)$ 分别表示 B 中每个对象执行联系操作和选择操作的代价，其余类推. 按照捆绑式方法，只有一种运算次序，其代价为

$$C(T)=1000\times(10-1)+1000\times(100+1)+1000\times0.8\times2=113600$$

如果按下面(4)式：

$$\sigma_1(\sigma_2(\sigma_3(A*B*C)[P_3])[P_2])[P_1] \tag{4}$$

执行，则执行代价为

$$C(R)=1000+1000+2000+0.1\times1000\times100+0.1\times0.8\times1000\times10=14800$$

两种方案执行代价相差8倍之多，所以有必要将每个结点中的选择与联系操作分开，采取更灵活的优化措施. 一种可行办法是将每个结点中的两种操作分开，各成一个结点，使联系操作为选择操作的父结点. 对于这样生成的查询树我们仍可以用前述方法得到最佳计算次序.

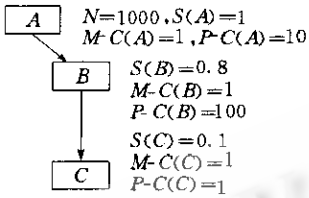

图4 查询树例子

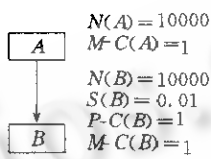

图5 查询树例子

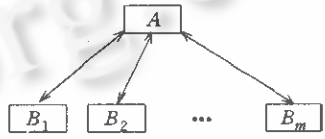

图6 查询树例子

2 支持双向联系时寻找最佳查询方案的启发式算法

上面的结果虽然很有用，但查询树必须满足较严格的条件限制. 在实际应用中，可能需要放宽某些限制条件，这时寻找最优查询方案就变成非常复杂的问题. 本节讨论引入双向联系后给该问题带来的影响. 为此假定在本节中，除非特别声明，所有的联系都是双向的.

双向联系可由对象聚集、索引和反向属性等引起，其重要性已在文献[1]中讨论过，此时联系操作具有可交换性.

先看一个例子，对于如图5所示的查询树，A结点无选择操作，A，B两类中的对象具有一一对应关系，若按文献[1]中的结论，则最优查询方案为$\sigma_B(A*B)[F_B]$，其代价为

$$C(T)=10000\times(1+1+1)=30000$$

若先在B结点上做选择操作，然后将之与A作联系操作，亦即作$\sigma_B(B)[F_B]*A$，其代价为*

$$C'(T)=10000\times2+100=20100$$

由此显见，先对B结点做选择操作更好一些. 后一个方案比前一个方案优越之处在于它不对无用对象做联系操作，如用前一种方案做例中的查询时，99%的用联系操作生成的对象被随后的选择操作抛弃了. 所以，若要寻求最优查询方案，则必须打破第1节规定的父结点先于子结点的优先级限制.

下面说明在上述条件下寻找最优查询方案的难度是NP-完全的. 首先，该问题可被不确定算法在多项式时间内完成显然成立的，亦即该问题肯定是一个NP问题. 为证明该问题是NP-完全的，我们要证明能在多项式时间内可将该问题归约为另一NP-完全问题的输入. 考虑如图6所示的查询树，假设对任一$1\leqslant i\leqslant m$，均有$N(B_i)\times S(B_i)<N$(N为查询所

* 按该方式尚需对结果进行反转操作，因代价较小，故此处忽略不计. 下面第3节的讨论亦同此.

涉及的对象总数)成立,A 与任一 B_i 的对象间具有一一对应关系.显然,在查询的中间过程中,对任一 $B_i(1\leqslant i\leqslant m)$,若 $N(B_i)\times S(B_i)<N$ 成立(此时 N 是查询中间结果中含有的对象个数),则先对 B_i 做选择操作较有利.

在查询的第 1 步,要确定这些结点的执行次序,再者,对确定要执行的结点 $B_i(1\leqslant i\leqslant m)$ 还面临着先做选择还是先做联系的问题.为简化讨论,假定 $M\_C(B_1)=M\_C(B_2)=\ldots=M\_C(B_m)=0$,$N(B_1)=N(B_2)=\ldots=N(B_m)=N/10$,$P\_C(B_1)=P\_C(B_2)=\ldots=P\_C(B_m)=1$,则最佳方案是选定 k 个结点,对这些结点先做选择后做联系,再对其余的结点按秩从小到大进行"联系-选择"操作.该问题等价于从 m 个结点中选出 k 个结点,不妨设为 $B_1,B_2,\ldots,B_k$,当满足

$$S(B_1)\times S(B_2)\times\ldots\times S(B_k)\geqslant 0.1 \tag{5}$$

时,使 $C(T)$ 最小.(5)式可改写成

$$\log(1/S(B_1))+\log(1/S(B_2))+\ldots+\log(1/S(B_k))\leqslant 1 \tag{6}$$

$C(T)$ 取最小值可转换成另一个等价条件,设该查询树的满足优先级限制的最优查询方案的查询代价为 $O\_C(T)$,则 $C(T)$ 最小即为 $O\_C(T)-C(T)$ 最大.最优方案的改进之处反映在对 $B_1,B_2,\ldots,B_k$ 共 k 个结点的处理上,则

$$O\_C(T)-C(T)=N\times(1-S(B_1))+N\times S(B_1)\times(1-S(B_2))+\ldots+N\times S(B_1)\times\ldots\times S(B_{k-1})\times(1-S(B_k)) \tag{7}$$

要取最大值.这恰好是一个 NP-完全的背包问题(见文献[5]中第 79 页).由于我们讨论的例子经过许多简化,故一般情形的难度应当不低于此.

上面已说明当支持双向联系时从查询树中找出最佳查询方案是 NP-完全问题.当树中结点个数很少时,或许可采用穷举搜索法找出最佳方案.但当结点个数较多时,穷举搜索法便不再适用,而可用启发式算法、随机搜索或前两者的结合.本节讨论支持双向联系时的寻找最佳查询方案的启发式算法,而随机搜索、随机搜索与启发式算法相结合的方法可参阅文献[4].

在讨论之前先引入概念"结点反演"和"可反演性".

定义 5. 对于查询树 T 中一结点 A,先作联系操作,再作选择操作,则称对结点 A 进行正演;若对 A 先作选择后作联系操作,则称对 A 进行反演.

定义 6. 对于查询树 T 中一结点 A(不为根)可反演,是指在某一状态下,对 A 进行反演比进行正演代价要低,则称 A 结点在该状态下可反演.

例如在图 7 中,设对象个数 $N=100$,B 中对象数 $N(B)=50$,谓词选择率 $S(B)=0.8$,C 中对象数 $N(C)=60$,谓词选择率 $S(C)=0.5$,A 与 B,A 与 C 中对象均为一对一联系.当 A 未和 C 进行联系操作前,因 $S(B)\times N(B)<N$,$S(C)\times N(C)<N$,所以 B 和 C 均是可反演的;但当 A 和 C 进行联系操作并在 C 上进行过选择操作后,对象个数 $N=60\times 0.5=30$,$S(B)\times N(B)>N$,则 B 不再具有可反演性.因此,我们说一结点具有可反演性是针对某个特定状态而言的.

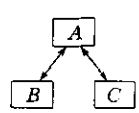

图7 查询树例子

回忆在第 1 节中,处理过程对结点的处理是自下而上进行的,而寻找可反演结点这种过程是自上而下的.因此,要把这两种过程综合起来考虑有一定难度.一种解决办法是先采用

算法1得到一组具有优先级限制的最佳结点序列,然后再在该结点序列中寻找可反演结点进行反演.可用算法2描述:

算法2. 在支持双向联系的查询树中寻找最优查询方案(1)

输入:算法1的输出

输出:最优查询方案

方法:按原次序,检查结点序列中除根结点外的所有结点,如果发现可反演结点,则反演之.

该算法对所有结点进行一次扫描即可,故算法复杂性为$O(m)$.在考虑费时谓词时,将联系和选择分为两个结点的情形,算法2亦需作相应的变动.此时可用算法3来描述:

算法3. 在支持双向联系的查询树中寻找最优查询方案(2)

输入:考虑费时谓词时算法1的输出

输出:最优查询方案

方法:按原次序检查结点序列中除根结点外的所有联系结点,若一联系结点A后紧跟对A的选择操作,则对该两结点合并考虑,判断其是否可反演,若可,则反演之.

有两个理由使我们相信算法2和3是好的,并且实现效率很高.第1,在某一状态下,采用算法2找到下一个结点这件事本身具有全局影响,它影响这之后的每一步操作的代价,而反演一个结点只具有局部影响,只要我们决定了需要进行操作的下一个结点,正演或反演该结点只对这个结点的操作代价产生影响,而与其后的操作代价无关.第2,采用算法1寻找下一个结点所依据的准则中包含谓词选择率,当谓词选择率低时,该结点被选中的概率就大,而这个因素也同样适用于选择反演结点.因此这两者在一定程度上是一致的.

算法2与3不难推广应用于一般情形,如查询树中不仅包含有双向边,还有单向边,只需在寻找可反演结点时考虑到这一点就可以了,甚至无需对算法2、3作字面上的更动.

3 算法2和算法3的有效性检验

实验依赖的查询树与查询代价计算方法如第1节所描述.查询树具有的结点数在10^+~20^+之间*,分成3个区间,具体数目是随机产生的.树中每个非叶结点最多可具有的子结点数分为4和8两种情况,但对一棵树中任一特定结点,其子结点数亦是随机产生的.根据结点数与最多可允许的子结点数两种因素组合可得6种情形.大部分结点其单位代价为1,对全部对象的选择率(即各个结点上的选择率之积)在0.1~5%之间,初始时全部对象数为10 000.

第1组和第2组实验分别检验算法2和3的有效性.对第1组实验,10~30%的结点其单位代价在1~2之间.对第2组实验,10~30%的结点其单位代价在1~20之间.在这两组实验中,都采用了穷举搜索算法找出最佳查询方案的代价,然后与算法2和3的结果相比较.两组实验均运行了120个例子,具体结果为:

算法2的总有效率=最佳查询方案的平均代价/算法2的平均代价=88.6%

算法3的总有效率=最佳查询方案的平均代价/算法3的平均代价=82.7%

在最极端的情形,算法2达到的最佳有效率为100%,最差的有效率为75.3%,算法3

* 生成查询树的具体作法是:假设查询树具有的结点数为10^+,最多可具有的子结点数为4.首先生成树根结点,为之增添m个子结点(m为0~4范围内的随机数).然后重复下列过程:判断已生成的树是否已有10个结点,若有,则结束;否则,从所有的叶结点集合中任意挑选一个,为它添上n个子结点(n为0~4范围内的随机数).

达到的最佳有效率为 94.2%，最差的有效率为 69.9%，这表明算法 2 和 3 性能平稳. 另外，算法 3 的表现比算法 2 稍逊色一些，这主要是因为，如前所述，两组实验所用的查询树初始条件相近，只是算法 3 所用的含有一些费时谓词，这时，算法 3 将联系操作和选择操作分开，形成更多的结点，使查询树更复杂，所以算法 3 所得到的结果要差一些.

参考文献

1 吴胜利，王能斌. 面向对象数据库的查询优化. 软件学报，1997，8(2)：153～160.

2 Stanley Y W Su, Guo Mingsen, Lam H. Association algebra: a mathematical foundation for object-oriented databases. IEEE Transactions on Knowledge and Data Engineering, 1993, 5(5): 775～798.

3 吴胜利，王能斌. 面向对象数据库中基于有向图的联系代数. 计算机学报，1997，20(1)：58～67.

4 吴胜利. 面向对象数据库系统的查询优化[博士论文]. 南京：东南大学，1996.

5 Garey M R, Johnson D S. 计算机和难解性-NP 完全性理论导引. 北京：科学出版社，1987.

OPTIMIZATION OF “ASSOCIATE-SELECT” COMPOUND OPERATION IN OBJECT-ORIENTED DATABASE SYSTEMS

WU Shengli

(*Institute of Software The Chinese Academy of Sciences Beijing* 100080)

WANG Nengbin

(*Department of Computer Science and Engineering Southeast University Nanjing* 210096)

Abstract “Associate-Select” compound operation, which is similar in functionality to join operation of multi-nested relations in nested relational databases, is a key issue of query optimization in object-oriented database systems. This paper proves that the optimum solution for treelike “Associate-Select” queries with some restrictions can be obtained with polynomial complexity, then presents some algorithms to deal with the situation when birelationship exists in complex objects (for example by defining inverse attribute et al.). A simulation experiment is performed to demonstrate the steadiness and effectiveness of the algorithms proposed in the paper.

Key words Object-oriented database, query optimization, “associate-select” compound operation.

Class number TP311.13