利用区间约束优化包含多个用户函数的查询*

杨 波, 洪晓光, 王海洋

(山东大学 计算机科学系,山东 济南 250100)

E-mail: yangbo12@263.net

http://www.cs.sdu.edu.cn

摘要: 如何高效地处理说明性查询语言中嵌入的用户自定义函数,是查询优化的一个重要内容.以往的研究成果不能处理一条语句中的多个用户函数,并且难以实现.提出了分3个阶段进行优化的方案,能够对用户定义的多个函数进行处理.首先,把用户定义的函数以区间约束的形式等价地表述出来;然后,通过对区间约束进行分层筛选,去掉冗余;最后,选择最佳的执行策略.该方案易于实现,效率较高,特别是在用户定义的函数本身隐含多个表的连接条件时,更能取得明显的优化效果.

关键词: 约束数据库;查询优化;用户定义函数;区间约束;连接

中图法分类号: TP311 **文献标识码**: A

现代数据库的设计越来越注重查询语言的描述能力.一个典型的例子是在扩展关系数据库和面向对象的数据库系统中允许用户定义面向应用的函数,并且可以把它们加入查询谓词中.用户定义函数的引入,使数据库操纵语言(data manipulation language,简称DML)得到了很大的扩充.但是,由于用户函数的时间复杂性对于优化器来说是未知的,所以多数系统采取先对已知复杂性的查询谓词进行优化,然后再将结果施以用户函数的方式.这种方式通常不是一种理想的选择,尤其是在存在连接运算而用户函数又相对简单的情况下,更显得缺乏效率.这一点在文献[1]中得到了证实.

针对这一缺陷,有很多文章提出了改进方法.按照优化思想的分类,目前可以看出这类问题解决方案的3个发展方向.第1种方案称为"谓词迁移"[2],是采用代数形式来描述用户定义的函数.另一种方案是"物化函数",就是在数据库中保存函数调用的结果[3].

我们采取的是第3种方案,称为"重写"方法.与一般的重写方法不同,我们首先把用户定义的函数以区间约束的形式表述出来.它虽然结构简单,但有很强的扩展能力.其次,我们通过引入谓词之间"覆盖"的概念,对函数进行了分层处理.然后,我们建立面向约束的索引,使求值策略进一步得到优化.这3种方案的对照关系见表1.

由表1可以看出,如果采用"谓词迁移"的方案,不仅在构造和实现代数模型方面还有待研究,而且在现实的数据库系统中对代数表达形式也缺乏有效的支持."物化函数"方案的思想十分直接,但运行时开销太大.另外,对函数本身也有一定的限制."重写"方案便于实现,优化效率比较高,虽然区间约束形式简单,但对于可以用分段函数表示的数据(如空间数据)则有很强的表示能力.

* **收稿日期**: 2000-02-15; **修改日期**: 2000-05-12

基金项目: 山东省自然科学基金资助项目(Q97G01158);霍英东青年科学家基金资助项目(71065)

作者简介: 杨波(1975-),男,山东济南人,硕士生,主要研究领域为数据库及应用;洪晓光(1964-),男,山东济南人,博士生,副教授,主要研究领域为数据库应用;王海洋(1965-),男,山东文登人,博士,教授,博士生导师,主要研究领域为数据库及应用,软件工程,CSCW.

Table 1 Comparison of current plans
表 1 现有解决方案的比较

Predicate migration①	Function materialization②	Rewriting③
◆Construct describing model④	◆Construct the materialization table of user defined functions⑦	◆Rewrite user-defined functions⑩
◆Partly abandon the idea of encapsulation⑤	◆Maintain the materialization table dynamically⑧	◆Describe functions equi-valently in the form of interval constraints⑪
◆Represent user-defined functions as algebra⑥	◆Convert function invocation into manipulation of table⑨	◆Eliminate the redundancy and create index⑫

①谓词迁移,②物化函数,③重写,④构造表示模型,⑤一定程度上忽略封装特性,⑥以代数形式描述用户定义的函数,⑦建立用户定义函数的物化表,⑧动态维护物化表,⑨把函数调用转化为对物化表的操作,⑩改写用户定义的函数,⑪以区间约束的形式等价地描述出来,⑫去掉冗余,建立索引.

1 基本概念

为了讨论方便,我们约定一个 Datalog 查询为如下形式:

$$Q(x_1,\ldots,x_k):-R_1,\ldots,R_m,\Phi_1(.),\ldots,\Phi_n(.).$$

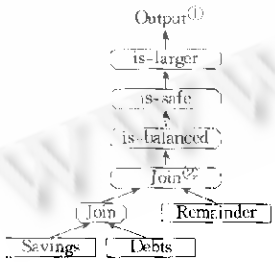
其中 $Q(x_1,\ldots,x_k)$表示查询返回的属性;$R_i(1\leqslant i\leqslant m)$表示查询所要操作的基本表;$\Phi_j(.)(1\leqslant j\leqslant n)$表示以表 $R_1,\ldots,R_m$ 的属性作为变元的查询谓词;$\Phi_1(.),\ldots,\Phi_n(.)$之间是合取的关系.

例 1:在银行系统中,每一个用户都有存款、贷款、余额这 3 项内容.要求用户满足等式"贷款+余额=存款",存款和贷款是正值,余额可以是负值.满足等式的称为账目平衡.如果余额大于存款,称为安全贷款.

我们可以用以下的语句来表示查询账目平衡、存款大于贷款的情形,而且是安全贷款的用户.

$$Q(Account):\ savings(Account,S),debts(Account,D),remainder(Account,R),$$
$$is\_balanced(S,D,R),is\_safe(R,D),is\_larger(S,D)$$

在本例中,$is_balanced(S,D,R)$可以等价地描述为 $D+R=S$,而 $is_safe(R,D)$和 $is_larger(S,D)$可分别用 $R>D$ 和 $S>D$ 替代.然而从优化器的角度来看,这 3 个函数都是封装的"黑盒",它们的计算复杂性是未知的.因此,优化器将会首先对 savings,debts 和 remainder 连接,然后进行选择,它的执行方案如图 1 所示.



①输出,②连接.

Fig. 1 The common optimization plan
图1 通常的优化方案

定义 1(区间约束). 区间约束是形如 $P(a_1,\ldots,a_L)\ \theta\ 0$ 的一个范式,其中 $L\geqslant 1$,$P(.)$是 $a_1,\ldots,a_L$ 的一次多项式,$\theta\in\{<,=,>,\neq,\leqslant,\geqslant\}$.

由于约束实际上是以数据库属性作为变元的一阶谓词逻辑,我们引入如下假定:

假定 1. 设 T 是查询语句中的约束的集合,则 T 中不包含永远为真或永远为假的约束.

可以看出,上述假定是合理的,并不会影响我们讨论问题的一般性.

定义 2(函数谓词).

如果数据库中共有 m 个表 $R_1,\ldots,R_m$,且 $A(R_i)$表示表 R_i 的属性集合,则 $A=\bigcup_{i=1}^{m}A(R_i)$表示所有属性.$C$ 为数据库查询语句中所涉及的参量集.

对于一个外部函数 $\Phi(.)$,如果它的参数是 A 和 C 的子集,且 $\Phi(.)$不会带来任何副作用,则称

$\Phi(.)$为函数谓词或Φ谓词.

定义 3. Ψ谓词

$\Phi(a_1,\ldots,a_L,C)$是一个函数谓词,令 A 是$\{a_1,\ldots,a_L\}$的子集,A 表示 $a_1,\ldots,a_L$ 中所有在$\Phi(.)$中允当约束变元的属性集合.$\overline{A}$ 是 A 的补集,它是所有自由变元的集合.令 ξ_A^i 表示 A 中属性的若干约束的合取.

$\oint_A$ 是一个映射,对于 A 和C的某一个初始值$(A^*,\overline{A}^*,C^*)$,无论 $\overline{A}^*$ 如何取值,都满足以下条件:所有满足 $\Phi(a_1,\ldots,a_L,C)$的元素都满足 η_A,其中 η_A 是 0 个或若干个 $\xi_A^i(i\leqslant D_A)$的析取,反之亦然.由此,我们有如下定义:

$\Phi(.)$是一个 Ψ 谓词$\Leftrightarrow$对任何$(A^*,\overline{A}^*,C^*)$都有 $\oint_A$ 存在,使得 $\Phi(.)$至少满足一个 ξ_A^i.这里,$\oint_A$ 称为 $\oint$-映射,D_A 是 $\oint_A$ 的维度.

定义 4. 对于 Ψ 谓词 Ψ_A 和 Ψ_B,如果在 $R_1\times R_2\times\ldots\times R_n$ 中所有满足 Ψ_A 的元素,也必定满足 Ψ_B,则称 Ψ_A 覆盖 Ψ_B.

2 优化思想

本文提出的优化方法主要分 3 个阶段完成.第 1 阶段,改写用户定义的函数为区间约束的形式;第 2 阶段,利用区间约束的表达形式对函数进行分类;第 3 阶段,利用面向约束的索引求得优化结果.这种方案扩大了查询语句的描述能力,可以对多个用户定义的函数进行处理;特别是在用户定义的函数本身隐含多个表的连接条件时,更能充分体现分层和索引的作用,取得明显的优化效果.

(1) 重写用户定义的函数

对用户定义的函数进行重写,是为了充分挖掘函数的语义特性,并以区间约束的形式表述出来.在这个过程中,我们要引入两类等价关系.一类等价关系是用户定义的函数与改写后的区间约束表达式在语义上的等价性;另一类是由用户定义的函数所描述的各属性之间的等价性.比如,对例 1 中所定义的函数 is_balanced,is_safe 和 is_larger 可以作如下变换:

$$is\_balanced(S,D,R)\Leftrightarrow\Phi_{-}(S,D+R),$$

$$is\_safe(R,D)\Leftrightarrow\Phi_{>}(R,D),$$

$$is\_larger(S,D)\Leftrightarrow\Phi_{>}(S,D).$$

考虑到用户定义的等价关系 $\Phi_{-}(S,D+R)$,$is_larger(S,D)$可进一步重写为 $\Phi_{>}(R,0)$.

(2) 分类优化

假定 Ψ 谓词(析取范式)Ψ_A 覆盖 Ψ_B,则它的等价描述是,Ψ_A 的可满足集合是 Ψ_B 的可满足集合的子集,也可以等价地表示为$\neg\Psi_A\vee\Psi_B$ 是重言式.基于这一事实,我们用下面的填表法进行覆盖的判定:

令 $\Psi_A=\xi_A^1\vee\xi_A^2\vee\ldots\vee\xi_A^m(m\geqslant 1)$,$\xi_A^i$ 是合取项为约束的合取式;

$\Psi_B=\xi_B^1\vee\xi_B^2\vee\ldots\vee\xi_B^n(n\geqslant 1)$.

由于$\neg\Psi_A$ 和$\neg\Psi_B$ 都可以等价地重写为析取范式的形式,且有如下假定:

$$\neg\Psi_A=\eta_A^1\vee\eta_A^2\vee\ldots\vee\eta_A^k(k\geqslant 1),$$

$$\neg\Psi_B=\eta_B^1\vee\eta_B^2\vee\ldots\vee\eta_B^L(L\geqslant 1),$$

则我们定义判定覆盖的表格共有 $n+L$ 行 $m+k$ 列,第 1 行~n 行依次标记为 Ψ_B 的 n 个合取式 $\xi_B^1\xi_B^2\ldots\xi_B^n$,第 $n+1$~$n+L$ 行依次标记为$\neg\Psi_B$ 的 L 个合取式 $\eta_B^1\eta_B^2\ldots\eta_B^L$.

第 1~k 列依次标记为$\neg\Psi_A$ 的合取式 $\eta_A^1\eta_A^2\ldots\eta_A^k$,第 $k+1$~$k+m$ 列依次标记为 Ψ_A 的合取式 $\xi_A^1\xi_A^2\ldots\xi_A^m$,见表 2.

Table 2 The table to identify 'cover'

表 2 "覆盖"判定表

	η_A^1	η_A^2	...	η_A^k	ξ_A^1	ξ_A^2	...	ξ_A^{m-1}	ξ_A^m
η_B^1	1	1	...	1	1	1	...	0	0
η_B^2	1	1	...	1	1	0	...	0	0
...									
η_B^L	1	1	...	1	1	0	...	0	0
ξ_B^1	0	0	...	0	1	1	...	1	1
ξ_B^2	0	0	...	0	1	1	...	1	1
...									
ξ_B^n	0	0	...	0	1	1	...	1	1

我们令表的列标记集合为 $M_c=\{m_1,m_2,\ldots,m_p\}$,$p=m+k$.表的行标记集合为 $M_L=\{L_1,L_2,\ldots,L_q\}$,$q=n+L$.同时,我们用 $Conj(m_i)$和 $Conj(L_j)$表示 m_i 和 L_j 所代表的合取式.表的填写规则是,对 m_i 和 L_j,在表的第 j 行第 i 列中填写逻辑谓词$\exists\ a_1\exists\ a_2\ldots\exists\ a_L(Conj(m_i)\wedge Conj(L_j))$.也就是说,如果 $Conj(m_i)\wedge Conj(L_j)$不是否定式,则填写"1",否则填写"0".

定理 1. Ψ 谓词 Ψ_A 覆盖 Ψ_B 的充要条件是:

1) 所有列标记为 ξ_A,行标记为 η_B 的表格内容为"0".

2) 所有列标记为 ξ_A,行标记为 ξ_B 的表格内容为"1".

3) 所有列标记为 η_A,行标记为 η_B 的表格内容为"1".

证明:

① 对所有列标记为 ξ_A^i,行标记为 η_B^j 的表格,由于 $Conj(m_i)\wedge Conj(L_j)$表示的是 $\Psi_A\wedge\neg\Psi_B$,也就是$\neg(\neg\Psi_A\vee\Psi_B)$,则$\neg\Psi_A\vee\Psi_B$ 是重言式$\Leftrightarrow Conj(m_i)\wedge Conj(L_j)$是否定式.

② 对所有列标记为 ξ_A^i,行标记为 ξ_B^j 的表格,由于 $Conj(m_i)\wedge Conj(L_j)$表示的是 $\Psi_A\wedge\Psi_B$,又由于假定 1 中已经假定不存在永远为假的约束,所以满足 $Conj(m_i)\wedge Conj(L_j)$的元素一定存在.

③ 对所有列标记为 η_A^i,行标记为 η_B^j 的表格,由于 $Conj(m_i)\wedge Conj(L_j)$表示的是$\neg\Psi_A\wedge\neg\Psi_B$,又由于在假定 1 中已经假定不存在永远为真的约束,所以满足 $Conj(m_i)\wedge Conj(L_j)$的元素一定存在.

综上可得,定理 1 成立. □

根据定理 1,我们可以把查询语句改写成所有未被其他谓词覆盖的 Ψ 谓词所表示的形式,在语义上与未改写的查询语句是等价的,但改写后的结果不包含冗余的函数.

(3) 建立索引

为了与用户定义的函数中隐含的语义信息有效地结合,我们可以充分地利用 $\oint$ 映射描述用户定义的函数中显式地给出的等价关系.例如,我们引入:

$$\oint_{\text{balanced}}:R,D\longrightarrow\xi_{(R,D)},$$

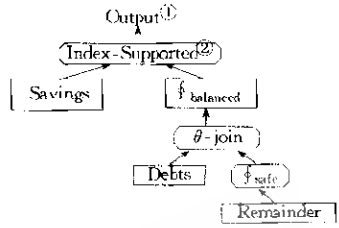
$$\oint_{\text{safe}}:R\longrightarrow\xi_{(R)},$$

其中 $\xi_{(R,D)}$定义为 $S=R\mid D$,$\xi_{(R)}$定义为 $R>D$. 满足 $R>0$,$D>0$. 相应地,is_balanced,is_safe 也要作如下替换:

$$is\_balanced(S,D,R)\Leftrightarrow\Phi\_(S,\oint_{balanced}(R,D)),$$

$$is\_safe(R,D)\Leftrightarrow\oint_{safe}(R).$$

由于 $\oint_{balanced}$和 $\oint_{safe}$的引入,查询谓词的结构变得复杂了. 同时,优化器也有了更大的选择空间以挑选最佳的执行策略. 一种可能的执行方案如图 2 所示.



①输出,②索引支持.

Fig. 2 The three stage optimization plan
图2 三阶段优化方案

3 问题讨论

在以上的讨论中,我们对支持用户定义的函数的查询优化问题采取了分 3 个阶段加以解决的方法. 目前,在基于重写的方案中,还没有处理多个用户定义的函数的方法. 我们的研究结果可以满足这方面的要求. 在优化性能方面,我们的方法较之其他方案,也有很大的改进. 以 V. Gaede[1]提出的 Φ-谓词方案为例,对例 1 中的查询语句,两种方案的执行策略的比较见表 3.

Table 3 The performance comparison of $\oint$-predicate plan and three-stage optimization plan

表 3 使用 $\oint$-谓词方案与三阶段方案的性能比较

Φ-Predicate plan①	Three-Stage optimization②
Do not construct index on table remainder③	Construct index on table remainder④
Join debts and remainder without condition⑤	θ-join⑥
Select records that satisfy function is_safe⑦	No selection needed⑧
Select records that satisfy function is_larger⑨	No selection needed

①Φ-谓词方案,②三阶段优化,③不对表 remainder 建索引,④对表 remainder 建索引,⑤对表 debts 和 remainder 作无条件连接,⑥θ-连接,⑦选择满足 is_safe 的记录,⑧无需选择,⑨选择满足 is_larger 的记录.

利用区间约束实现支持用户定义的函数的查询优化,是我们进行的一个初步尝试. 优化方法本身还有很多问题需要进一步探讨. 其中包括:

(1) 如何找到一种高效的重写工具,能够方便地把查询谓词改写为析取范式的形式.

(2) 对区间约束深入研究,寻找在现实的数据库系统中的实现途径.

(3) 在多维数据库、数据仓库和 Web 数据库中的实现形式和优化方法.

我们将就以上问题展开讨论,继续进行研究.

References:

[1] Gaede, V., Gunther, O. Constraint-Based query optimization and processing. In: Kuper, G. M., Wallace, M., eds. Constraint Databases and Applications. Friedrichshafen, Germany: ESPRIT WG CONTESSA Workshop, 1995. 84~101.

[2] Hellerstein, J. M. Practical predicate replacement. In: Snodgrass, R. T., Winslett, M., eds. Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. Minneapolis, Minnesota: ACM Press, 1994. 325~335.

[3] Kemper, A. Kilger, C. Moerkotte, G. Function materialization in object bases. In: Clifford, J., King, R., eds. Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data. Denver, Colorado: ACM Press, 1991. 258~268.

Optimization of Multiple User-Defined Functions in a Query Using Interval Constraints*

YANG Bo, HONG Xiao-guang, WANG Hai-yang

(Department of Computer Science, Shandong University, Ji'nan 250100, China)

E-mail: yangbo12@263.net

http://www.cs.sdu.edu.cn

Abstract: How to process user-defined functions incorporated in declarative query languages efficiently is an important aspect of query optimization. The problem of several user-defined functions in a query clause hasn't been solved in the former researches. There isn't a proposal in these researches that can be implemented easily. In this paper, a 3-stage optimization plan is put forward, which has the potency of processing several user-defined functions in a query clause: firstly, rewrite user-defined functions equivalently in the form of interval constraints; secondly, stratify the constraints and eliminate the redundant ones; finally, select the optimal execution strategy. This plan has the virtue of easy implementation and higher efficiency. Especially when the user-defined functions imply join conditions of several tables, this plan can get an obvious optimization result.

Key words: constraint database; query optimization; user-defined function; interval constraints; join

* Received February 15, 2000; accepted May 12, 2000

Supported by the Natural Science Foundation of Shandong Province of China under Grant No. Q97G01158; the Huo Ying-dong Young Scientists Foundation under Grant No. 71065