约束数据库中的索引*

王宇君　田增平　曲云尧　施伯乐

(复旦大学计算机系　上海　200433)

摘要　本文研究了约束数据库中的索引技术,提出了存储区间常数刺穿集的数据结构 S 树和 S* 树.在刺穿集的最大长度受到限制的条件下,S* 树存储效率是最优的.与 M 树相比,S 树和 S* 树有一个明显的改进:可以支持删除操作.

关键词　约束数据库,索引,B+树,动态区间管理.

中图法分类号　TP391

约束数据库是 Kanellakis 等在文献[1]中提出来的一种新的数据模型.由于它能对时空数据进行统一的表示,且有较好的数学背景,因此近年来受到了数据库研究者的广泛关注.但总的说来对其研究还很不完善.如约束数据库的物理组织,目前仅限于讨论索引的数据结构及算法,主要的研究成果有文献[2]中提出的 Meta_block Tree(简称 M 树).在不考虑插入删除时,M 树的存储和查询效率是最优的,即有类似关系数据库中 B+树的性能.但 M 树数据结构复杂,而且不支持删除操作.

我们考虑直接存储刺穿集来提高算法性能,关于刺穿集下面将会介绍.其它约束数据库中的基本概念,参见文献[1,3].我们提出了 S* 树的数据结构,基本思想是存储区间常数的刺穿集,但限制刺穿集最大深度,且减少所存刺穿集的个数.以期针对实际中使用的约束的特点,获得较好的存储效率.同时 S* 树的有关查询、插入及删除的算法性能也较好.

本文第 1 节介绍约束数据库中的索引的特点;第 2 节介绍 S 树;第 3 节介绍 S* 树;最后是结论.

1　约束数据库中索引的特点

设 R 为变量集 X 上的约束关系,D 为论域,$x \in X$,R 在 x 上的 1 维索引有如下特点:

(1) 该索引是对一组区间在辅存的管理.其中每个区间对应 R 中一个约束元组.它是该约束元组在 x 上的投影.由于约束 k 元组在 D^k 空间中对应的点集是一个凸集,因此它在 x 上的投影是一个区间.若投影结果不是闭区间,约定增加端点的值将其扩充为闭区间.

---

* 本文研究得到国家自然科学基金资助.作者王宇君,1970 年生,博士,主要研究领域为约束数据库.田增平,1968 年生,博士,主要研究领域为多媒体数据库.曲云尧,1961 年生,博士,副教授,主要研究领域为面向对象数据库.施伯乐,1935 年生,教授,博士导师,主要研究领域为数据库理论及应用.

本文通讯联系人:王宇君,上海 200433,复旦大学计算机系

本文 1997-01-22 收到修改稿

(2) 约束数据库在变量 x 上的1维范围查询,指对给定的变量 x 上的待查询范围$[L,U]$,查询满足$(L\leqslant x\leqslant U)$的区间的集合.

(3) 满足查询的区间共有4种可能,如图1(a),其中1、2两种情形是容易找到的.因为此时左端点的值都在 L 和 U 之间,易知只需对所有的区间按它们的左端点的值依次排列,通过使用B+树即可做到.其中3、4两种情形我们称为刺穿查询.对它的处理是整个查询能否有效完成的关键.如图1(b)所示,区间的4种情形对应4个区域,其中刺穿查询对应左下角为(L,L)、标记为3,4的对角区域,因此区间管理可看成是关系数据库中2维范围搜索的特例.

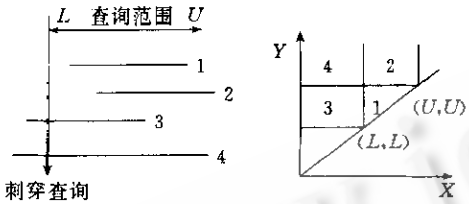

图1

(4) 插入或删除一个约束元组,需要在索引中相应插入或删除一个区间.

定义 1. 约束关系 R 的索引区间集是 R 对应的3元组的集合$\{[i,L,U]\}$,其中$[i,L,U]$含义为标识为 i 的约束元组对应的待索引的区间为$[L,U]$.

定义 2. 设 R 的索引区间集为 T,约束关系 R 的区间常数集为$\{U|[i,L,U]\in T\}\cup\{L|[i,L,U]\in T\}$.

定义 3. 设 R 的索引区间集为 T,常数 c 的起始集为$\{[i,L,U]|[i,L,U]\in T$ 且 $L=c\}$.

定义 4. 设 R 的索引区间集为 T,常数 c 的刺穿集为$\{[i,L,U]|[i,L,U]\in T$ 且 $L<c\leqslant U\}$.

2 S树

2.1 S树的数据结构

S树可分为3部分.① 一棵B+树,存储关系R的区间常数集.叶子上的每个数据项对应关系 R 中一个区间常数;② B+树叶子上的每个区间常数对应的一个线性表,记录该常数的起始集;③ 每个区间常数对应的一个线性表,记录该常数的刺穿集.

为考虑S树的存储性能,我们定义重叠度 k 的概念.直观地说,重叠度是所有区间常数的刺穿集的平均大小.

定义 5. 设关系 R 的区间常数集大小为 n_a,又设每个区间常数对应的刺穿集为 S_i,$1\leqslant i\leqslant n_a$,$S_i$ 的大小为 $|S_i|$,则重叠度 k 为:$k=(\Sigma_{1\leqslant i\leqslant n_a}|S_i|)/n_a$.

S 树的三部分空间开销分别为 $O(n_a/B)$,$O(n/B)$及 $O((n_a*k)/B)$.在重叠度不超过某个常数时,存储的效率为 $O(n/B)$.然而重叠度最坏时可能与数据库的大小的增长成正比,这时 S 树的存储效率为 $O(n_a*n/B)$.

2.2 S树的1维范围搜索

对给定的待查询范围区间$[L,U]$,它对前述 $B+$ 树使用约 $\log_B(n_a)$ 次辅存访问,便可找到满足如下条件的 L_a:L_a 为大于等于 L 的最小的常数.显然若 $L_a\leqslant U$,则 L_a 的刺穿集都满足条件.从 L_a 至 U 的每个常数对应的起始集也都满足条件.而其它则不满足条件.对刺穿集和起始集的访问的全体正好构成 t 个查询输出结果,访问辅存 $O(t/B)$ 次,故总的访问次

数最坏时为 $O(\log_B n_a+t/B)$次.且系数接近 1.

2.3 S 树的区间插入

设待插入区间为$[i,L,U]$.先在 B+树中搜索 L.若找不到,则在该 B+树中插入数据项 L,并初始化 L 的起始集为空,刺穿集为 B+树叶子中的 L 的下一数据项对应的刺穿集的复制.类似地,若 B+树中找不到数据项 U,则插入 U 并作初始化.在 L 的起始集中增加$[i,L,U]$.并从 L 的下一数据项开始,依次考虑每个数据项 Z 直至 $Z>U$,在 Z 的刺穿集中增加$[i,L,U]$.

2.4 S 树的区间删除

设待删除区间为$[i,L,U]$.在 B+树中搜索 L,若找不到则返回.否则在 L 的起始集中搜索区间$[i,L,U]$,若找不到则返回.否则在 L 的起始集中删除区间$[i,L,U]$,并从 L 的下一个数据项开始,对每个小于等于 U 的数据项 Z,在 Z 的刺穿集中删除区间$[i,L,U]$.若 L(或 U)的起始集和刺穿集都为空,则在 B+树中删去 L(或 U).

3 S 树的改进 S* 树

分析实际中使用的约束集对应的区间,可以看到,对大部分区间来说,和它相交的区间集的大小远远小于数据库的大小.基于这样的事实,我们提出 S 树的改进 S* 树.

3.1 S* 树的数据结构

S* 树对 S 树进行了 3 方面的改进:①限制了刺穿集的最大长度 ω;②将起始集与存储常数的 B+树合二为一;③每个 B+树的叶子结点只有第 1 个区间左端点的刺穿集才得以存储.这样存储效率是最优的.插入、删除性能也较 S 树更好.付出的代价是,最坏情况时查询效率下降.

S* 树可分为两部分.第 1 部分是一棵 B+树,存储所有区间.第 2 部分是该 B+树每个叶子结点的第 1 个区间的左端点对应的一个线性表,它记录该常数值对应的刺穿集.我们称该常数为该叶子结点的刺穿常数,称该刺穿集为该叶子结点的刺穿集.

由于并入了起始集,在 S* 树的 B+树中,用来作为查找条件的值仅有一个常数是不够的,区间的两个端点的值都需参加比较.我们定义区间的先后次序比较规则:先比较左端点的值,若相等再比较右端点的值.由于限制了刺穿集的最大长度 ω,故约定:刺穿集的区间也根据上述比较规则按先后次序排列.超过 ω 后的区间在刺穿集中不再保存.

3.2 S* 树的 1 维范围查询

算法 1. Range_Search

输入:待查询范围区间$[L,U]$及 S* 树

输出:与$[L,U]$相交非空的区间的集合

步骤 1. 在 B+树中查找区间$[L,U]$直至叶子结点 N,N 的刺穿常数为 L_a,N 的刺穿集大小为 s;

步骤 2. 对 N 的刺穿集中的每一区间$[j,L_j,U_j]$,若 $L\leqslant U_j$,则输出该数据项;

步骤 3. 若 $s=\omega$,设 L_a 的刺穿集最后一项为$[m,L_m,U_m]$,则搜索$[m,L_m,U_m]$,并从$[m,L_m,U_m]$的下一数据项开始.依次考虑$[j,L_j,U_j]$直至到达 N,若 $L\leqslant U_j$,则输出该数据项;

步骤 4. 从 N 的第 1 个数据项开始,依次考虑$[j,L_j,U_j]$,若 $L_j>U$,则返回,否则,若 $L_j\leqslant L\leqslant U_j$,则输出该数据项.

对给定的待查询范围区间$[L,U]$,步骤 1 需约 $\log_B n$ 次辅存访问.若 N 的刺穿集大小不到 ω,则对刺穿集和起始集的访问的全体正好构成 t 个查询输出结果,访问辅存 $O(t/B)$次,故最坏时总的访问次数为 $O(\log_B n+t/B)$次.且系数接近 1.因而性能非常好.若 L_a 的刺穿

集大小为 ω,则需多执行步骤 3,这时算法的最坏性能为 $O(n/B)$.

3.3 S*树的区间插入

算法 2. Insert_Interval

输入:待插入的区间$[i,L,U]$及 S* 树

输出:修改后的 S* 树

步骤 1. 在 B+树中搜索$[i,L,U]$,若找不到,则在该 B+树中插入数据项$[i,L,U]$;

步骤 2. 若$[i,L,U]$插入后没有新叶子结点产生,则转至步骤 5;

步骤 3. 否则设结点 N 分裂为新的叶子结点 N_1 及 N_2,设 N_2 的首项为$[k,X,Y]$,则 N_2 的刺穿常数为 X. 通过访问结点 N_1 及其刺穿集,初始设置 N_2 的刺穿集;

步骤 4. 若 N_1 的刺穿集大小为 ω,最后项为$[m,L_m,U_m]$,而 N_2 的刺穿集大小不到 ω,则搜索$[m,L_m,U_m]$,并从$[m,L_m,U_m]$的下一数据项开始,依次考虑$[j,L_j,U_j]$直至$[k,X,Y]$或 N_2 的刺穿集大小为 ω,若 $U_j \geqslant X$,则将该数据项加至 N_2 的刺穿集;

步骤 5. 从$[i,L,U]$所在叶子结点的下一结点开始,对每个刺穿常数 Z 小于等于 U 的叶子结点,将$[i,L,U]$插入 Z 的刺穿集,这是有序表的插入.

3.4 S*树的区间删除

算法 3. Delete_Interval

输入:待删除的区间$[i,L,U]$及 S* 树

输出:修改后的 S* 树

步骤 1. 树的 B+树中搜索区间$[i,L,U]$,若找不到,则返回;

步骤 2. 否则删除区间$[i,L,U]$,设原$[i,L,U]$的下一数据项为$[k,X,Y]$;

步骤 3. 若删除区间后导致了叶子结点的合并,则删去一个相应的刺穿集;

步骤 4. 从区间$[k,X,Y]$的下一个叶子结点开始,依次考虑每个叶子结点 N,若其刺穿常数 $Z>Y$,则返回;否则在 N 的刺穿集中删除$[i,L,U]$,并执行步骤 5;

步骤 5. 若删除前 N 的刺穿集是满的,设刺穿集的最后项为$[m,L_m,U_m]$,则搜索$[m,L_m,U_m]$,并从$[m,L_m,U_m]$的下一项开始,依次考虑$[j,L_j,U_j]$直至结点 N,若 $Z \leqslant U_j$,则将该数据项加至 N 的刺穿集,并返回步骤 4.

在步骤 5 中,访问的$[j,L_j,U_j]$一定出现在步骤 4 中已访问过的叶子结点中. 通过优化,可进一步减少 I/O 次数.

3.5 刺穿集深度的动态适应

当重叠度 k 接近或超过 ω 时,对大多数查询来说,都必须往回搜索,这样查询效率将下降,为使 S* 树能适应这种变化,可通过适当修改上述算法使之能动态修改参数 ω. 由于 ω 一般为内外存交换单位 B 的整数倍,因此可通过对 ω 增加或减少 B 来实现. 同时,不同的叶子结点的刺穿集的最大长度也可以不同. 这样可最大限度地平衡时间和空间开销. 这也是 S* 树的灵活性的一种体现.

4 结 论

约束数据模型是关系数据模型的推广,它允许把约束的合取式作为约束元组存储,使得无限信息能得以精确描述,特别适合空间数据、时态数据等的表示. 索引技术的研究使得相应查询的求值可得到更为一般的优化. 现有的研究主要有 M 树[2],它分析了约束数据库中索引的特点,并提出了 M 树的数据结构,但不支持删除. 本文从另一个角度提出了 S 树、S* 树的概念. 它们支持删除. S* 树在重叠率 k 不超过某常数时,表现出良好的性能. 若允许对刺穿集的深度作动态调整,则在重叠率变化时,表现出对时间和空间开销的良好动态平衡.

未来的工作有:结合约束数据库的实现及在 GIS 等中的应用,对 S* 树作进一步的改进.

参考文献

1 Kanellakis Paris C. Kuper Gabriel M. Revesz Peter Z. Constraint query languages. JCSS 51. 1995.

2 Kanellakis Paris C, Ramaswamy Sridhar. Vengroff Darren *et al.* Indexing for data models with constraints and

classes. PODS 93.

3 王宇君. 约束数据库：模式设计，代数查询语言及索引[博士论文]. 复旦大学，1996.

INDEXING FOR CONSTRAINT DATABASES

WANG Yujun　TIAN Zengping　QU Yunyao　SHI Baile

(*Department of Computer Science　Fudan University　Shanghai*　200433)

Abstract　In this paper, the indexing in constraint databases is considered. Meta-block tree is improved and a data structure S* tree is presented. It stores the stabbing sets for constants that appears in the intervals. If the maximum length of each stabbing set is limited, the space used in S* tree is optimal. Compared with M tree, a significant improvement of S* is that it can support delete operation.

Key words　Constraint databases, index, B+tree, dynamic interval management.

Class number　TP391