集合索引结构及其联接操作*

汪 卫[+], 谢闽峰, 陶 春, 施伯乐

(复旦大学 计算机与信息技术系,上海 200433)

Index Structure for Set data and Its Join Operation

WANG Wei[+], XIE Min-Feng, TAO Chun, SHI Bai-Le

(Department of Computing and Information Technology, Fudan University, Shanghai 200433, China)

+ Corresponding author: Phn: +86-21-65643501, Fax: +86-21-65643502, E-mail: weiwang1@fudan.edu.cn

Received 2003-05-26; Accepted 2003-09-26

Wang W, Xie MF, Tao C, Shi BL. Index structure for set data and its join operation. *Journal of Software*, 2004,15(11):1661~1670.

http://www.jos.org.cn/1000-9825/15/1661.htm

Abstract: Set type is an important data type in object-oriented database system and object-relational database system. An index structure of set type Set_struc is presented in this paper. In Set_struc all sets are organized as a tree, and the sets with common prefix are merged. So the size of the index will be decreased for the data set with a large number of repeated data and frequent patterns. Based on Set_struc, an algorithm of join operation with Set_struc is presented. Its performance is better than other methods such as PSJ (partition based join).

Key words: index of set; Set_struc; join operation

摘 要: 集合类型是面向对象数据库和对象-关系数据库中的一种重要的数据类型.提出了集合类型数据的一种索引结构 Set_struc,并提出了基于 Set_struc 的集合联接算法.Set_struc 通过合并集合数据的公共前缀组织数据.这种方法可以减少重复数据和重复模式的存储空间,并通过基于树的联接算法提高集合数据上的联接操作的性能.其性能优于现有的算法,如 PSJ(partition based join).

关键词: 集合索引;Set_struc;联接操作

中图法分类号: TP311 **文献标识码**: A

集合类型是一种很常见的数据类型.现实世界中的很多关系均可以用集合类型描述,如一对父母有若干个孩子,一篇文章有若干个关键词等.同时,实体建模中的很多关系也是用集合类型描述的,如实体间 Part-of 的关系等.为此,数据库界一直在研究这种关系的存储和表示方法,在关系数据库模式设计中将其作为嵌套关系进行处理或将其分解成多个关系进行存储[1].面向对象数据库系统和对象-关系数据库系统出现以后,集合类型被直

---

* Supported by the National Natural Science Foundation of China under Grant Nos.69933010, 60303008 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA4Z3430 (国家高技术研究发展计划(863))

作者简介: **汪卫**(1970－),男,山东济南人,博士,教授,主要研究领域为数据库技术,数据挖掘,XML 数据管理;**谢闽峰**(1980－),男,硕士生,主要研究领域为 XML 数据管理;**陶春**(1974－),男,博士生,主要研究领域为数据库,XML 数据管理,数据挖掘;**施伯乐**(1935－),男,教授,博士生导师,主要数据库系统及其应用,数据仓库与数据挖掘,数字图书馆,安全数据库.

© 中国科学院软件研究所 http://www.jos.org.cn

接加入到数据模型和查询语言中[2~4],并在这方面进行了比较深入的研究.由于集合类型结构同原子类型有很大的差别,原来基于原子类型的查询和索引技术对集合类型的数据都不适用了,目前还没有成熟的、用于提高集合类型数据查询性能的索引结构.所以,目前在数据库应用中很少使用集合类型.

随着计算机应用的深入,越来越多的应用(如 XML 数据处理和生物信息学)提出了对大规模的集合数据进行有效管理和查询的要求.目前,XML 数据应用日益广泛,将 XML 数据存放在关系数据库中是一种常用的方法.而在 XML 数据中有大量的集合数据.在 XML 文本中,形如〈! ELEMENT person (name,children*,…)〉的元素定义是非常普遍的.在这个定义中,children 就是一个集合数据.目前对这种数据还没有很好的方法,一般都是将其存成单独的表[5].这种方法对于一些略微复杂的结构将生成大量的表,在查询时引入大量的联接操作,从而影响了查询的性能.

与基于常规数据的数据库查询操作相同,在集合数据中联接操作的计算量是非常大的.这不仅体现在 I/O 的计算量上,还体现在集合操作中,需要大量的内存计算代价.集合数据上的联接操作主要包括两种:一种是集合相等关系,另一种是包含关系.

本文针对集合数据提出了一种索引结构 Set_struc.这种结构是利用集合数据中的结构信息和集合数据的次序信息建立的一棵树型结构,同时还借助了反向文件的思想定义了反向链表.本文还基于 Set_struc 提出了集合数据包含联接操作的实现算法.它具有较高的执行效率,同时避免了基于 Signature 方法中计算量较大的验证阶段.

本文第 1 节介绍集合数据查询技术的相关工作.第 2 节详细介绍集合数据索引结构:Set_struc 的组织结构.第 3 节对基于 Set_struc 的几种基本操作即选择、插入和删除的实现方法进行简单的描述.第 4 节着重介绍基于 Set_struc 的集合数据联接操作的实现算法.第 5 节分析基于 Set_struc 的联接操作的计算复杂性以及与相关方法的比较,并对相关的实验结果进行分析和比较.最后对全文进行总结.

1 相关工作

由于集合数据具有特殊的结构和广泛的应用背景,对集合数据的查询操作方面的研究近年来引起了数据库研究人员的重视.文献[6]讨论了基于传统的 Signature 和 Invert_file(反向文件)的方法,研究了基于 B_树的 Invert list, Sequential signature file, Signature tree 和 Extendible Signature Hashing 这 4 种集合索引结构,并对选择操作的实现方法进行了研究和比较.文献[7]提出了一种基于树的集合索引结构 R_D tree,R_D tree 以 R_树的思想为基础,将内容相近集合数据存储在一起.文献[8,9]基于 Signature 的方法对集合数据的联接操作进行了研究,提出了 PSJ(partition based join)算法.这种方法将联接操作分成 3 步:第 1 步,将两个表中所有的集合数据转换成 Signature,并将这些 Signature 通过散列函数分成若干个分区;第 2 步,对两个表的对应的分区进行联接,生成可能的结果对;第 3 步,再次访问数据库,对可能的结果对逐个进行验证,得到最终正确的结果.这种方法的问题在于:

1. 由于不同元素的 Signature 的组合可能产生相同的值,所以这种方法需要一个确认阶段来判断结果的正确性.而确认阶段需要再次访问数据库,这部分开销在整个开销中占了很大的部分[8].

2. 如果生成的 Signature 无法整个放在内存中,则在生成分区的阶段需要做大量的对硬盘的写操作.

一般在数据库中会出现大量的重复模式,例如,在关于数据库的论文的关键词列表中经常会同时出现“查询语言”和“查询代数”.前面提到的方法均没有考虑在数据库中重复出现的模式.

文献[10,11]对基于 Signature 的方法进行了进一步的研究,对各种情况下的集合数据的联接方法和代价进行了研究.

2 Set_struc 索引结构

在介绍集合索引结构之前,首先介绍几个常用的概念.

定义 1. 对集合 $S=\{a_1,a_2,\ldots,a_n\}$,S 的规范化表示 $S=\langle b_1,b_2,\ldots,b_n\rangle$,其中 $b_1,b_2,\ldots,b_n$ 是 S 中的元素,且对所有的 $i(n>i\geq 1)$,有 $b_{i+1}\geq b_i$.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 2. 对集合 $S1=\{a_1,a_2,...,a_n\}$和 $S2=\{b_1,b_2,...,b_n\}$,若它们的规范化表示为 $S1=\langle a_1',a_2',...,a_n'\rangle,S2=\langle b_1',b_2',...,b_n'\rangle$,若称 $S1>S2$,则存在 i,对所有的 $j<i$,有 $a_j'=b_j'$,且 $a_i'<b_i'$.

例如,集合$\{a,c,b\}$的规范化表示为$\langle a,b,c\rangle$,同时对集合$\{a,c,b\}$和$\{e,c,a,f\}$,有$\{a,c,b\}<\{e,c,a,f\}$.

2.1 集合数据的树型表示

我们可以通过合并集合数据中公共前缀将集合数据集组织成一棵树的结构,例如对集合值的集合:$SET=\{s1:\{a,b,c\},s2:\{a,c,e\}, s3:\{a,b,d\},s4:\{a,b,e,f\},s5:\{b,c\},s6:\{a,b\},s7:\{a,b,e\}\}$,我们可以将它们组织成如图 1 所示的结构.

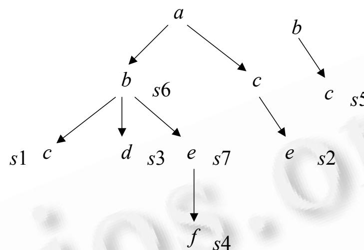

Fig.1 Demonstration of Set-tree

图 1 Set-tree 示意图

在这棵树中每个结点包含一个集合的元素,作为该结点的值,树中的每个结点都对应一个集合值,集合中的元素由从根到这个结点的路径上的结点上的值构成,这棵树构造的方法是首先将集合中元素按从小到大的次序进行排序,将相同的前序数据合并在一起.例如,在图 1 中,$s6,s3,s7,s4$ 具有公共的元素$\{a,b\}$.它们对应的结点构成一棵树.我们称这种结构为 Prefix 树.每个结点的子结点的次序是以它的值的次序为准的.本文基于集合数据的这种组织方式建立集合数据的索引结构 Set_struc.

2.2 Set_struc的物理结构

整个 Set_struc 由 3 部分构成:一部分是 prefix 树的森林;一部分是树上结点构成的反向链表;另一部分是数据域,它记录了树上的结点所对应的数据库中的记录位置.

2.2.1 prefix 树和数据域

从图 1 中我们可以看到,具有相同最小元素的集合可以构成一棵树,这棵树称为 prefix 树.所有的 prefix 树构成一个森林.森林的整体结构和图 1 是相似的.prefix 树中的每个结点由 5 个域构成:

- 元素:代表该结点对应的元素.
- 标号:记录了该结点在树中的前序位置.
- 数据域指针:指向数据域中该结点对应的集合数据的 id 的位置.如果该指针为空,则该结点为空结点.
- 子指针:指向第 1 个子结点.
- 兄弟指针:指向下一个兄弟结点.

下面看一个 Set_struc 的例子.假设集合数据集为 $s1:\{a,b,c\},s2:\{a,c,e\},s3:\{a,b,d\},s4:\{a,b,e,f\},s5:\{b,c\},s6:\{a,b\},s7:\{a,b,e\},s8:\{a,b,d\}$,假设 ai 代表集合值 si 对应标识符,则这个数据集对应的 Set_struc 为:在图 2 中结点 1、结点 7 和结点 9 是空结点,表示这个结点代表的集合数据没有在数据库中出现.结点 4 代表集合$\{a,b,d\}$.由于数据库中有两条记录值为$\{a,b,d\}$,所以在数据域中有两条记录与之相对应,并排列在一起,而结点 4 的指针指向最前面一个结点,这样通过结点 4 就可以找到所有值为$\{a,b,d\}$的集合.

在 Prefix_树中,结点的标号有一个非常有用的特点,即定理 1.

定理 1. 对于 Prefix_树中的结点 a(其标号为 a')和 a 的相邻的右兄弟结点 b(其标号为 b'),c(其标号为 c')是 a 的后代结点,当且仅当 $b'>c'>a'$.

定理 1 可以很容易地从树的前序遍历的性质中得到.

© 中国科学院软件研究所 http://www.jos.org.cn

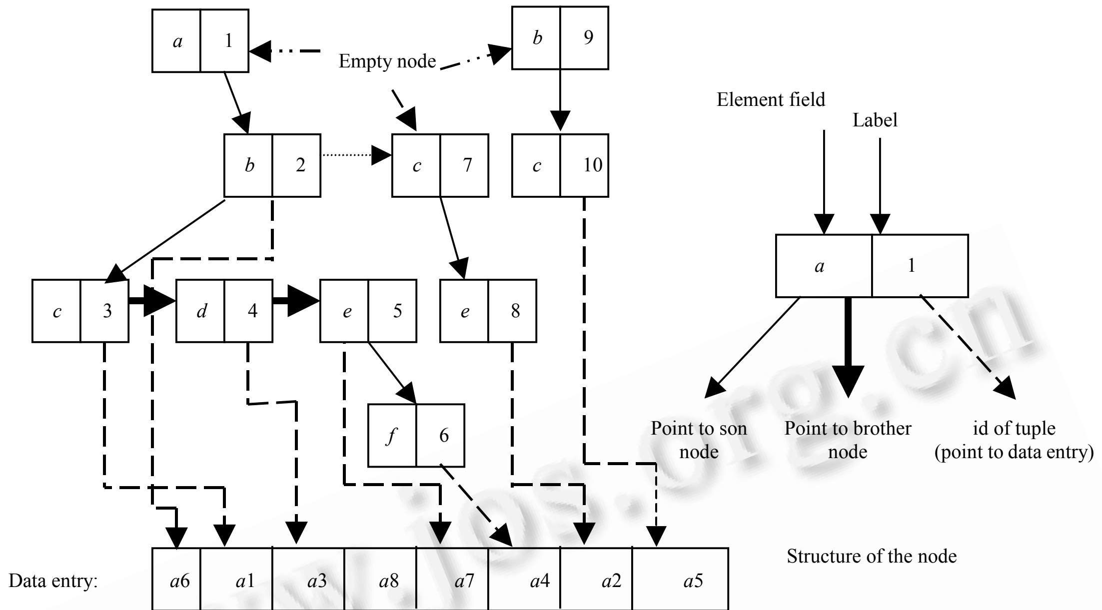

Fig.2 Prefix_tree and the data field
图 2 Prefix_树和数据域部分

2.2.2 反向链表

Set_struc 的另一部分是一个反向链表,该链表记录了集合数据集中的每个元素在 prefix 树中所出现的位置.

例如,对图 2 中的 prefix 树对应的链表为

a:1
b:2,9
c:3,7,10
d:4
e:5,8
f:6

反向链表在选择和联接操作中将起到重要的作用.

3 联接操作

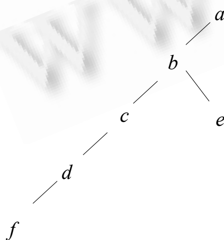

Fig.3 Demonstration of related node
图 3 结点相关性示意图

联接操作是数据库中的主要操作之一,同时也是数据库管理系统中最耗费代价的操作,所以联接操作一直是数据库界研究的重点,并产生了很多方法,如基于块的嵌套循环的方法、基于索引的嵌套循环的方法、Merge_Sort 方法[12]等,并在如何利用索引提高查询性能方面进行了深入的研究.这些方法对常规数据具有很好的效果.下面我们将给出基于 Set_struc 的集合数据包含联接算法.

在算法描述之前,我们先给出结点的相关性定义.

定义 3. Prefix_树中结点 m 相关于结点 n,当且仅当 n 的父结点是 m 的祖先结点.

例如,图 3 中的集合{a,b,e}对应的结点相关于集合{a,b,c,d,f}对应的结点.结点的相关性有一个非常重要的性质.

© 中国科学院软件研究所 http://www.jos.org.cn

定理 2. Prefix_树中的两个结点 m 和 n,若结点 m 相关于结点 n,且结点 m 和结点 n 对应的元素是相同的,则结点 m 所对应的集合是结点 n 所对应的集合的超集.

定理 2 的证明可以很容易地从 Prefix_树的定义得到.

下面讨论联接算法.设将对 R 和 S 两张表做联接操作,联接条件是 $R.A \subseteq S.B$,若 $R.A$ 和 $S.B$ 有它们相对应的 Set_struc.算法的主要过程是:对 $R.A$ 对应的每个 prefix_struc(假设根结点的元素为 r),与 $S.B$ 中所有元素为 r 的结点的子树进行联接,这些子树可以通过 r 元素的反向链表得到.这样,整个联接过程变成了若干个树的联接.树的联接过程(Tree_Join 函数)就是针对 $R.A$ 对应的树的每个结点 m,在 $S.B$ 对应的树中寻找对应的集合包含 m 所对应的集合的结点,这些结点与 m 和 m 的祖先结点对应集合满足包含关系,可以作为结果输出.为此,在算法中使用了一个候选串,该串记录了 R 中从根结点到当前结点的路径上所有的结点的集合值.由于 R 中每一结点 n 对应的集合均包含于其祖先结点对应的集合,所以若 n 对应的集合与 S 中的某个集合具有包含关系,则 n 的祖先结点对应的集合也应与 S 中的集合形成包含关系.由于 m 需要同 $S.B$ 中的多个结点进行树联接,所以在算法中设立了联接串记录了还需要考虑的联接结点对.算法 1 对这个过程进行了描述.

算法 1. 联接算法.

输入:两个 Set_struc.

输出:具有包含关系的集合标号对.

程序:

```
建立空候选串和联接串.
For R 中的每棵 Prefix_树 rst(设其根结点为 p, p 上的元素是 a)
{
  将 rst 导入内存
  for 对 S 中 a 的反向链表中的每个结点 n;   //找到 S 中所有以 a 为根结点的子树;
        {
      将 Set_struc 中的以 n 为根结点的子树 sst(设其根结点为 q)导入内存;
        Tree_join(p,q,NULL)
        }
}
Tree_Join 函数的算法是:
Tree_Join(A,B,Path)   //树联接操作
{
将 B 代表的集合的标识 id_B,与候选串中的各个集合标识 id_A 合并成(id_A,id_B),放入结果集中   //生成当前结点对应的联接结果
For R 中每个与 A 相关的结点 n,且 n 的元素为 B.element
      将(n,B,path+n)加入 n 的联接串
      /*由定理 2 可知,若 p 对应的集合是 n 对应的集合的超集,则也应该放入联接结果中*/
For   (联接串中的所有和 A 相关的结点对(A,C,path1))
      {
      根据 Path1 调整候选串中的结点;
        Tree_Join(A,C,path1)   //联接候选串中的子树对
      }
将 A 加入候选串.
pointA=A 的第 1 个儿子;
If (pointA<>NULL)
{
```

```
    for (A 的每个儿子 pointA)    //对 p 和 q 的子结点进行遍历,找到相包含的结点对
    {
        若 pointA 包含的元素为 e,从关系 S 中 e 的反向链表中找到所有属于 B 的子孙的结点 l
        for(每个 l)
        {
           将从 B 到 l 路径上的所有结点对应的集合数据放入候选串
              Tree_Join(pointA,l,path+pointA 包含的元素)
              /*对子结点递归调用树联接*/
        }
    }
  }
  else
        取出 B 的所有子孙结点,将它们的标识 idB 与候选串中的各结点代表的集合数据的标识 idA 合并成
        (idA,idB),放入查询结果中.
  }
```

下面通过一个例子看一下两个 Set_struc 联接的过程.假设图 4 是 R 和 S 两个关系对应的 Set_struc,其中关系 R 包含两棵 Prefix_树,分别以 a 和 b 为根结点,而关系 S 只有一棵 Prefix_树.

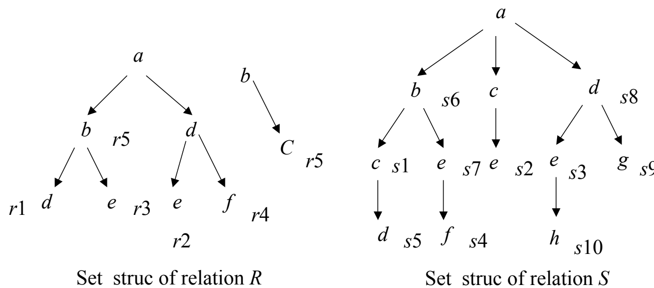

Fig.4 An example

图 4 一个例子

算法执行的过程如下:首先执行两个以 a 为根结点的 Prefix_树联接操作.树联接的过程从两个根结点开始,两个结点均为“a”,然后访问到两个值为“b”的结点,返回结果“$\langle r5,s6\rangle$”,然后访问关系 R 中的结点“d”,关系 S 遍历到结点“c”,由于 $d>c$,关系 S 继续向下遍历,到结点“d”,返回结果$\langle r5,s1\rangle$,$\langle r5,s5\rangle$,$\langle r1,s5\rangle$在第 2 层 b 和 d 是公共结点,所以可以对它们对应的子树进行联接操作.当处理到 R 表中的$\{a,b,d\}$对应的结点和 S 表中$\{a,b,c,d\}$对应的结点时需要在 R 中寻找相关结点,这次找到的是$\{a,d\}$对应的结点.这样,$\{a,d\}$和$\{a,b,c,d\}$对应的结点放入候选串,然后依次向后找,直到遍历两棵树.然后再从候选串中取出候选对继续执行.当以 a 为结点的树联接完以后,联接 R 中以 b 为根结点的子树,通过反向链表找到$\{a,b\}$对应的结点为根结点的子树,联接结果是将$\{r5,s1\}$和$\{r5,s2\}$放入结果集中.

4 性能分析

下面来分析基于 Set_struc 的联接算法的性能.

4.1 比较计算量分析

在进行分析之前,我们首先定义一些符号.

M 和 N 代表 R 和 S 两个关系表中的元组数.P 代表 R 和 S 中分区的数量.Set_Size_S 代表 S 中集合数据的长度,Set_Size_R 代表 R 中集合数据的长度.$N1_S$ 代表 S 中 signature 中设置成 1 的位数.$NODENUM_R$ 是 R 对应的

关系的 Prefix_树中结点的个数,*NODENUM_S* 是 *S* 对应的关系的 Prefix_树中结点的个数.

首先分析基于 Signature 的方法的比较的次数.通过文献[8]中的实验可以发现,大约 90%的开销花费在试探阶段和确认阶段,其中花费在试探阶段的代价为 $o(M*N*Set_Size_S*N1_S)/(P)$.确认阶段的比较次数为:数据库的长度+预测失误的数据量+结果的长度.

下面分析基于 I/O 的复杂性.如果内存的可用量是有限的,则基于 Signature 的方法将为 *R* 和 *S* 中的每个元组生成相应的 signature 并存放在硬盘上,*S* 中的每个 signature 要向硬盘写 *rs* 次(*rs* 是 *S* 中 signature 中 1 的个数),因为它属于 *m* 个分区,这样,总的写操作的次数是 $M+N*rs$(在[*kr*]中,*rs* 接近于 *S* 的集合数据的长度,如果分区的数量达到一定的程度).读操作的次数是 $2M+N*(rs+1)$.

定理 3. 算法 1 的计算的最大复杂性为 $o((NODENUM_R*NODENUM_S)*((2^{SET_SIZE_R/2})/C_d^{TR/2}))*(SET_SIZE_R*d)*(\log_B{}^{NODENUM_S/d}))+o(t)$.

证明:Prefix_树的一个结点进行联接的结点有如下特征之一:

(1) *A* 为空值.

(2) 它们的前缀又包含关系.

对于第 1 种情况,其每步计算均产生一个查询结果.所以,其复杂性为 $o(t_1)$,其中 t_1 为这一步产生的结果的数量.

对于第 2 种情况,满足第 2 个特征的结点数在整个树中占的比例平均为 $o(2^{SET_SIZE_R/2})/C_d^{TR/2}$,其原因是对一个长度为 *SET_SIZE_R*/2 的集合,其不同的选择是 $C_d^{TR/2}$(不失一般性,可假设 Set_struc 中的结点对应的集合平均为 *SET_SIZE_R*/2),而对其中某个特定的长度为 *SET_SIZE_R*/2 的集合而言,其包含的集合的个数为 $2^{SET_SIZE_R/2}$.找到相包含的结点后的代价主要包括以下部分:

1. 返回结果,这部分的代价之和为 $o(t_2)$,t_2 为这一步产生的查询结果的数量;

2. 寻找相关节点,而找到相关结点的代价为 $o(SET_SIZE_R*d)$;

3. 找到下一个满足包含条件的结点对,这部分的工作是通过查找反向链表实现的,如果反向链表是以 *B*_树的形式管理的,则其代价为 $o(\log_B{}^{NODENUM_S/d})$.

从算法的过程可以看到,算法最终结果的数量 *t* 为 $t1+t2$,其原因为联接操作的结果只在这两步中产生.

所以,最终的代价为 $o((SET_SIZE_R*d)*NODENUM_S*((2^{SET_SIZE_R/2})/C_d^{TR/2})*NODENUM_R*o(\log_B{}^{NODENUM_S/d})+o(t)=(NODENUM_R*NODENUM_S)*((2^{SET_SIZE_R/2})/C_d^{TR/2}))*(SET_SIZE_R*d)*(\log_B{}^{NODENUM_S/d}))+o(t)$. □

定理 4. 在一棵 Prefix_树中,$NODENUM_R>M*SET_SIZE_R$.

定理 4 是很好理解的,对于一个集合数据集,$M*SET_SIZE_R$ 为数据集中所有的元素的个数.Prefix_树中的结点小于 $M*SET_SIZE_R$ 的原因有两个:

1. 在 Prefix_树中,数据集中公共的前缀被合并了.例如,在图 2 中,$a1=\{a,b,c\}$和 $a7=\{a,b,e\}$两个集合中的 6 个元素被合并成 4 个元素.

2. 在 Prefix_树中,数据集中的重复元组只处理一次.例如,在图 2 中,$a3=\{a,b,d\}$和 $a8=\{a,b,d\}$两个具有相同值的集合中的 6 个元素只对应于树中的 3 个元素.

在一般的数据集中,这种重复元组或重复前缀是大量出现的,所以,使用 prefix-树将大大提高存储的效率.

由此可见,$M*N*SET_SIZE_R*Set_Size_S$ 一定小于 $NODENUM_R*NODENUM_S$.同时,根据 PSJ 的原理,$(T_S*N1_S)/(P)$将远大于$((2^{SET_SIZE_R/2})/C_d^{\mathrm{Set_Size_R}/2}))*(SET_SIZE_R*d)*(\log_B{}^{NODENUM_S/d})$.所以,从计算复杂性上看,基于 Set_struc 的方法的计算量比 PSJ 的方法中的试探阶段的计算量要小,基于 Set_struc 的方法中没有确认阶段,所以基于 Set_struc 的方法的性能要优于 PSJ 的方法.

从 I/O 上看,我们的方法是一种针对各种查询操作均有效的索引结构,所以是事先生成的,在联接操作中没有写操作,而其他方法很多是针对联接操作设计的,需要临时生成,所以在内存空间不足以容纳所有数据的时候需要执行写操作,这同样会造成性能的下降.

4.2 实验分析

4.2.1 实验条件

我们用 IBM 的数据生成器[13]生成了一批事务数据集,然后将相应的事务集转换成集合数据.我们的实验在奔腾 4 1.7G,256M 的 PC 机上进行,用 VC 6.0 上实现了我们的算法.为了比较,我们还做了两个实验:

(1) 在同等条件下实现了 PSJ 的算法[8].

(2) 根据文献[8]的描述,我们基于 IBM 的 UDB 7.0 实现了基于关系数据库的集合数据的联接方法.假设联接的两个关系为 $R_S(i,b)$和 $S_S(j,d)$,i 和 j 为每个集合的标识符,b 和 d 是相应的集合数据中的值.标识相同的行构成一个集合.整个查询分成 3 步实现.第 1 步是 count query:

Insert into $R_Stmp(I,count)$
Select $R_S.i,count(*)$
From R_S
Broup by $R_S.i$

第 2 步是 Candidate query:

Insert into $R_SS_Stmp(i,j,count)$
Select $R_S.i$, $S_S.j,count(*)$
From R_S, S_S
Where $R_S.b=S_S.d$
Broup by $R_S.i$, $S_S.j$

第 3 步是 Verify query:

Select $R_SS_Stmp.i$, $R_SS_Stmp.j$
From R_SS_Stmp, R_stmp
Where $R_SS_Stmp.i=R_Stmp.i$ and $R_SS_Stmp.count=R_Stmp.count$

4.2.2 无重复数据的联接操作性能分析

首先我们分析在没有重复的情况下,3 种联接算法的计算量.我们就集合平均长度为 5 的数据集进行了测试,不失一般性,我们就两个相同关系的联接操作进行了测试,结果如图 5 所示.从图 5 中可以看到,在小数据量的情况下,PSJ 和基于 Set_struc 的方法(标为 treejoin)的性能与基于数据库的方法(标为 db2)相比要好很多.这有两个原因:

(1) 现有的关系数据库管理系统中包含很多功能,因此加重了基于数据库的方法其系统的负载.

(2) 基于数据库的方法需要做两遍联接操作,所以系统开销很大.

从图 6 中可以看到,在大数据量的情况下,基于 Set_struc 的方法比基于 PSJ 的方法性能要好.其原因在前面已经进行了分析.

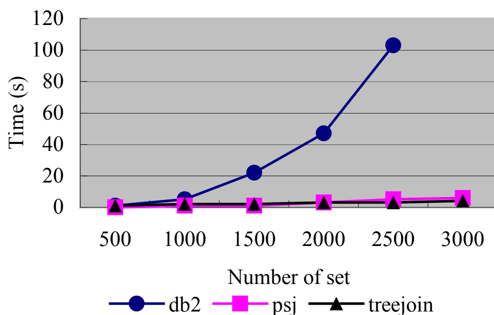

Fig.5 Cost of Join operation on small data set
图 5 小数据量情况下的联接代价

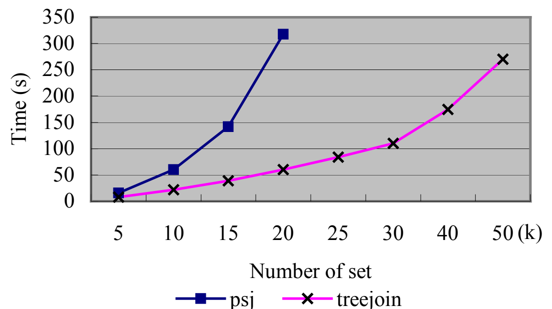

Fig.6 Cost of Join operation on large data set
图 6 大数据量情况下的联接代价

4.2.3 重复数据的联接操作性能分析

下面在有重复数据的情况下,对两种算法进行比较.我们取 5 000 个集合数据,不断改变其中数据重复遍数,进行实验.结果如图 7 所示,可见,在有重复数据的情况下,基于 Set_struc 的方法性能要占优.其原因是,PSJ 的计算量与数据库中集合数据的数据量成正比,所以,随着重复遍数的增加,联接操作的计算量并没有明显的改变.但是,基于 Set_struc 的方法的计算量与 Prefix_树中的结点数相关,而在有大量重复数据的情况下,Prefix_树中结点数将大大减少.所以,随着重复遍数的增加,基于 Set_struc 的方法的执行时间逐渐减小.

4.2.4 存储空间分析

由于在索引中要存储 Prefix_struc 和相应的反向链表,所以在存储空间上有一定的代价,但是当数据出现重复时,存储空间会节省很多.我们对平均长度为 5 的无重复集合数据库进行了测试,结果如图 8 所示.有趣的是,随着数据量的增加,索引文件的长度和原始文件的比例逐渐下降,如图 9 所示.其原因在于,随着数据量的增加,频繁的前缀模式逐渐增多,因此减少了数据的存储量.当然,随着数据库中重复数据的增加,Prefix_struc 中的结点数下降很快,因此索引的长度会有较大的下降.我们以平均长度为 5 的 50 000 条集合数据为例,从图 10 中可以看到,随着重复数据的增加,索引的长度有较大幅度的减少.当每条数据平均重复 2 遍的时候,其长度已经与原始文件的长度相差不大,当大于 2 遍的时候,其长度将小于原始文件的长度.

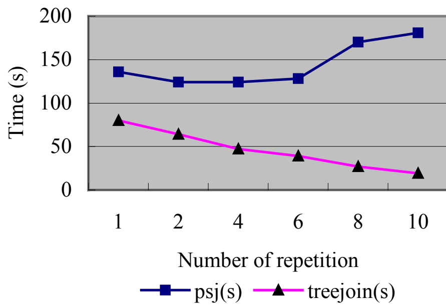

Fig.7 Cost of Join operation on repeated data
图 7 重复数据的联接代价

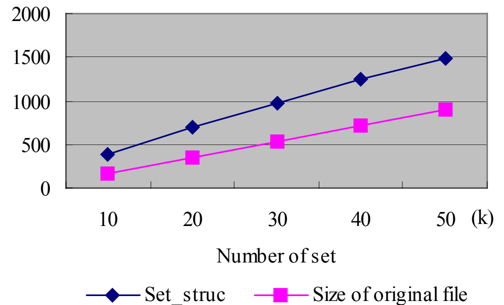

Fig.8 Comparison of the size of the file
图 8 文件长度比较

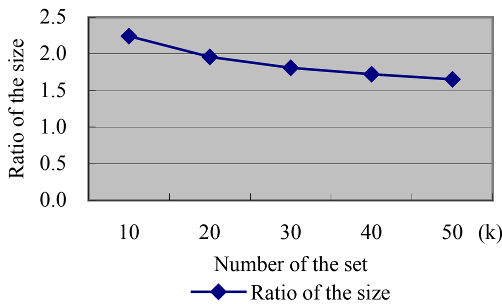

Fig.9 Ratio change of data file and index file
图 9 数据文件和索引文件比例变化

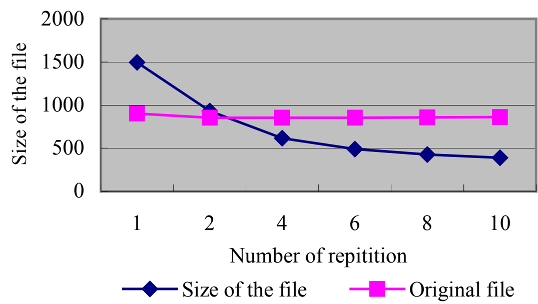

Fig.10 Index file size on repeated data
图 10 重复数据对文件长度的影响

由此可见,基于 Set_struc 的方法在存储空间略有损失的情况下,可取的联接算法有大幅度的提高.

5 总 结

本文提出了一种集合数据的索引结构.我们的方法提供了一个集合数据的索引结构 Set_struc.

- 这种结构对集合数据库进行了合理的组织,可以有效地管理数据集中的重复模式和重复数据.
- 基于这种结构的联接操作具有较好的性能.
- 可以提高选择、修改操作的性能.

今后,我们将继续研究基于 Set_struc 的其他查询操作的实现方法和基于块的 Set_struc 的访问方法.

References:

[1] Shi BL, Ding BK, Wang W. Database System Tutorial. 2nd ed., Beijing: Higher Education Press, 2003 (in Chinese).

[2] Beeri C. New data models and language—The challenge. In: Proc. of the 11th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. New York: ACM Press, 1992. 351~362.

[3] Cattell RGG. The Object Database Standard: ODMG-93. Morgan Kaufmann, 1993.

[4] Tannen V. Languages for collection Types. In: Proc. of the 13th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. New York: ACM Press, 1993. 150~154.

[5] Shanmugasundaram J, Tufte K, Zhang C, He G, DeWitt DJ, Naughton JF. Relational databases for querying XML documents: Limitations and opportunities. In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, eds. Proc. of the 25th Int'l Conf. on Very Large Data Bases. Morgan Kaufmann, 1999. 302~314.

[6] Helmer S, Moerkotte G. A study of four index structure for set-valued attributes of low cardinality. Reihe Informatik 2, University of Mannheim. 2003.

[7] Hellerstein JM, Pfeffer A. The RD-Tree: An index structure for sets. Technical Report, No.1252, University of Wisconsin, Madison, 1997.

[8] Ramasamy K, Patel JM, Naughton JF, Kaushik R. Set containment joins: The good, the bad and the ugly. In: Abbadi AE, Brodie ML, Chakravarthy S, Dayal U, Kamel N, Schlageter G, Whang KY, eds. Proc. of the 26th Int'l Conf. on Very Large Data Bases. Morgan Kaufmann, 2000. 351~362.

[9] Helmer S, Moerkotte G. Evaluation of main memory join algorithms for joins with subset join predicates. In: Jarke M, Carey MJ, Dittrich KR, Lochovsky FH, Loucopoulos P, Jeusfeld MA, eds. Proc. of the 23rd Int'l Conf. on Very Large Data Bases. Morgan Kaufmann, 1997. 386~395.

[10] Mamoulis N. Efficient processing of joins on set-valued attributes. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2003. 157~168.

[11] Melnik S, Garcia-Molina H. Adaptive algorithms for set containment joins. ACM Trans. on Database Systems, 2003,28(2):1~38.

[12] Garcia-Molina H, Ullman JD, Widom J. Database System Implementation. Prentice Hall, 2000.

[13] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: Bocca JB, Jarke M, Zaniolo C, eds. Proc. of the 20th Int'l Conf. on Very Large Data Bases. Morgan Kaufmann, 1994. 487~499.

附中文参考文献:

[1] 施伯乐,丁宝康,汪卫.数据库系统教程.第 2 版,北京:高等教育出版社,2003.