有效的近似数据方体维护算法

李翠平 王 珊
(中国人民大学信息学院 北京 100871))

摘 要 尽管利用预计算可以提高OLAP的查询效率,但是,由于存储空间的限制,预计算整个数据方体是不现实的。最近提出的综合数据方体通过将数据单元进行等价划分的方法解决了这个问题。然而,当数据源发生改变的时候,要对这样的数据方体进行维护是很困难的,即使只有一条元组发生了变化,所有的聚集值都必须重新计算,代价非常高。实际上,在有些应用环境中,人们更关注查询响应的速度,在查询结果的精度上可以放低一些要求。本文提出了如何对近似的综合数据方体进行增量维护的方法。实验证明,这些方法是非常有效的。

关键词 近似综合数据方体,增量维护,联机分析处理

Efficient Approximate Update to Data Cube

LI Cui-Ping WANG Shan
(Information School, Renmin University of China, Beijing 100872)

Abstract It is often not feasible to compute a complete data cube due to the storage requirement. Recently proposed quotient cube addresses this issue through a partitioning method that groups cube cells into equivalence partitions. However, when the data source is updated, the aggregate values need to be recomputed even after one tuple is inserted or deleted. To keep the aggregate values to be always exact can prohibitively expensive in terms of time and/or storage space in a data warehouse environment. In many applications, it is sufficient to generate fast, approximate instead of full precise answers to queries. In this paper, we propose and examine techniques at the maintenance of an approximate quotient cube. Efficient algorithms are proposed and their effectiveness at storage and maintenance is investigated. A systematic performance study is conducted on different kind of data sets, which demonstrates our algorithms are efficient and scalable over large databases.

Keywords Approximate quotient cube, Incremental maintenance, Online analytical proccssing

1 前言

数据方体预计算操作可以看作是对已有的GROUP-BY操作符的一种扩充,它对多个维的不同组合进行聚集计算。例如由如下查询表达的一个数据方体格结构共包含22个不同的数据单元:

```
SELECT A, B, C, SUM(M)
FROM R
CUBE BY A, B, C
```

数据方体的体积通常都比较大。例如,不考虑层次的话,一个10维(每个维有1000个不同的值)的数据方体就包含1001^{10}个数据单元。考虑层次的话会更多。如何有效地预计算一个数据方体成为数据库研究领域的一个重要课题[1~4],但由于存储空间的限制,要预计算一个完整的数据方体不太现实。最近,人们提出了一些方法来对数据方体的体积进行缩减,例如文[5~7]。综合数据方体通过将数据单元进行等价划分的方法解决了这个问题。作者提出了有效的综合数据方体构建算法。但是,当数据源发生改变的时候,所有的数据单元的值都必须重新计算,代价非常昂贵。考虑到有一些应用环境对查询精度的要求可以放低一些,我们提出了近似综合数据方体的概念,并提出了有效的增量维护算法。

所提出的近似综合数据方体可以在更大程度上缩减数据方体的体积,从而减少数据方体的计算代价和存储代价,但是并不丢失语义信息。

本文第2节介绍相关工作,第3节给出本文的背景资料和问题定义,第4节提出了近似综合数据方体增量维护算法。第5节给出了一些实验结果,最后对本文作了一个总结。

2 相关工作

到现在为止已经提出了很多数据方体预计算的方法,例如PipeSort, Overlap, PartitionCube, MemoryCube, ArrayCube和BUC。受存储空间的限制,不可能对整个数据方体进行预计算。最近提出了一些通过缩减数据方体体积从而来减少计算代价和存储代价的方法。Wang等提出了浓缩数据方体的概念[6],Lakshmanan等提出了综合数据方体的概念[5],Dwarf[7]通过去掉前缀和后缀冗余来缩减数据方体的体积。这三种缩减方法都能在某种程度上减小数据方体的体积,其中综合数据方体最为有效。本文在综合数据方体的基础上,提出了近似综合数据方体的概念,利用计算近似聚集值的方法,进一步缩减了数据方体的体积。传统的近似数据方体计算方法主要是借用统计学上的一些方法和模型,例如wavelet,采样,直方图等。但这些方法无法保证近似结果的误差范围。所采用的统计模型也比较复杂,不容易被用户所理解。

当数据源发生改变的时候,近似综合数据方体中的等价类可能会改变,要对其进行增量维护非常困难。本文的贡献

主要在于:(1)提出了近似综合数据方体的概念,其误差在一个可以控制的范围之内。(2)提出了有效的对近似综合数据方体进行增量维护的算法。实验结果表明该算法非常有效。

3 背景知识和问题定义

数据仓库由一个事实表和若干个维表组成。事实表由一个或多个维 $D_1,\cdots,D_n$ 和一个度量值组成。我们用符号 dom(D_i)来表示维 D_i 的值域。

事实表 R 中的一条元组 t 可以表示成 $t=(tid, dvalue, m)$,其中 tid 表示元组号,$dvalue\in dom(D_1)(dom(D_2)\times\cdots\times dom(D_n)$是元组 t 的维值集合。M 是元组 t 的度量值。我们用 $t.tid$, $t.dvalue$, 和 $t.m$ 来代表元组 t 的每一个分量。

例如,$t=$(4, a3b2c1, 10),表示 $t.tid$ 是 4, $t.dvalue$ 是 a3b2c1, $t.m$ 是 10。

如果元组 t 的 $t.dvalue$ 值在任何一个维上都和数据单元 c 的值相等,则说元组 t 和数据单元 c 相匹配。如果元组 t 和等价类 C 中的每一个数据单元都相匹配,则说元组 t 和等价类 C 相匹配。

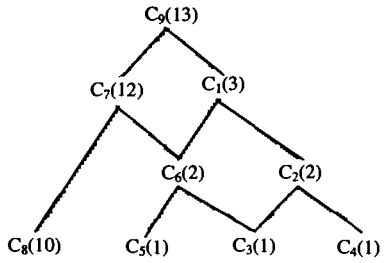

图 1 综合数据方体

定义(等价类) 某个数据单元集合 C 如果满足下列条件,则称它是一个等价类:

1) 给定任意两个数据单元 c, c',满足 $c\leqslant c'$ 且 c, c' 同时属于 C,则任意一个介于它们之间的数据单元 c'' 也属于 C。

2) C 是和相同的元组集合相匹配的最大数据单元集合。

一个等价类可以表示成 $C=(upp, m)$,其中 upp 表示类的上届,m 是聚集值。我们分别用 $C.upp$ 和 $C.m$ 来代表等价类 C 的两个部分。例如,图中的等价类 $C2$ 可以表示成 $C2=$(a1b1,9)。

一个综合数据方体由多个等价类组成。例如,图 1 所示是一个综合数据方体格结构的例子。正如在前言部分所介绍的,要计算所有的等价类来支持精确查询代价非常昂贵。本文提出了近似综合数据方体的概念,用户可以给定一个能够容忍的最大误差范围,在此基础上生成一个体积较小的数据方体,从而提高查询效率和存储效率。

例 1 假设数据方体的格结构如图 1 所示,用户指定的误差范围是 3。图 1 中的等价类的聚集值可以进似算出。比如,等价类 $C2$ 的聚集值在[agg(C1)-3, agg(C1)]之间,即在[0, 3]之间。图 1 对应的近似综合数据方体如图 2 所示。

最简单的计算近似综合数据方体的方法可以分为两个步骤。首先,使用文[5]中的深度优先算法确定出所有的等价类。其次,利用文[5]中的类合并方法将那些具有相似的聚集值的类合并在一起。由于在每次进行合并的时候都需要比较,该方法效率很低。而且当数据源发生改变的时候,所有的聚集值都需要重新计算。

问题定义 给定一个事实表,一个聚集函数和一个用户指定的误差范围,本文要解决的问题就是如何计算以及增量维护一个近似综合数据方体。

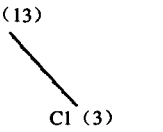

图 2 近似综合数据方体

4 近似综合数据方体的快速维护算法

考虑往事实表中插入一条元组的情况。假设 Φ 是现有的综合数据方体的等价类集合,t 是新插入的元组,Φ' 是新的等价类集合。T 可以从如下几个方面来影响一个现有的等价类。(1)使某个等价类的聚集值发生变化。(2)使某个现有的等价类分裂成两个等价类。(3)不做任何影响。我们综合了这几种情况,将所有的等价类分成三类,分别是:值改变的等价类,新类产生器,哑类。提出了两个有效的操作符:分裂操作和相交操作。

分裂操作是最耗时间的操作,我们希望尽量少做。如前所述,在有些情况下,人们并不要求非常精确的答案,查询结果只要能够保证在某个可以接受的误差范围之内就可以了。因此,如果对某个等价类所做的改变没有超出该误差范围的话,就可以暂时不分裂。

考虑往事实表 R 中插入一条元组 $t1$ 时的情况。假设有一个等价类 C 可能需要分裂,根据用户提供的误差范围,我们可以做如下考虑:

- 如果 $t1.m\geqslant\delta$, 则 C 需要分裂成两个类。
- 如果 $t1.m<\delta$, 则 C 不需要分裂。但为了累计效果,需要把此次修改记录下来。所以我们仍然需要生成一个隐含的新等价类。每一个新类产成器都有一个数据结构 plist 用来记录由它所生成的隐含的新等价类。

现在假设又来了第二个元组 $t2$。当一个新类产生器 C 生成一个隐含的新等价类 Cn 时,我们需要去查一下 $C.plist$,如果 Cn 已经存在,则需要将这两次修改累加在一起。累加完后,如果 $Cn.m$-$C.m$ 超过误差范围 δ,则 Cn 变成一个真正的新类,旧类 C 需要分裂。否则的话,什么都不用做。如果 Cn 在 plist 中不存在的话,则需要将其插入到列表中。

现在我们给出对现有的近似综合数据方体进行增量维护的算法 Approximate_Inc。其中的 Approximate_split 函数表示如何对旧类进行分裂。

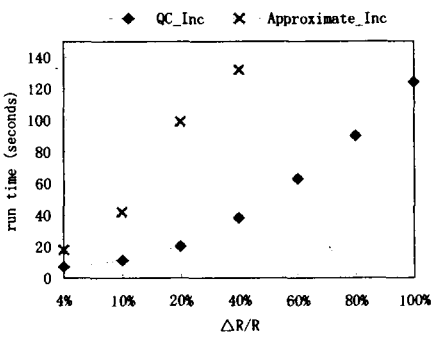

图 3 算法的有效性比较

算法 1 (算法 Approximate_Inc)

输入:旧类集合 Φ, 新元组 t, 误差范围 δ.

```
输出：新类集合 Φ′
1. 创建一个虚类 VC=(DV,{DV},0)，并将其插入 Φ
2. 将所有的类根据它的上届所包含的维的个数分成 B[0]…B[n+1]
   个桶，VC 在 B[n+1]中
3. B′[i]=O(i=0…n)
4. for i=0 to n+1 do
5.    for each class C in B[i] do
6.       if t.dvalue matches C.upp then
7.          C.m=Agg(t.m, C.m) and add C to B′[i]
8.       else
9.          MaxMatch = C.upp∩t.dvalue
10.         let k =|MaxMatch|
11.         if (k==0) then continue
12.         if →∃Z∈B′[k] s.t. Z.upp=MaxMatch
            then Approximate_Split(i,t,C,δ)
13. 输出 B′[i]中的所有类(i=0…n)
Function Approximate_Split(i,t,C,δ)
14. if t.m≥δ
15.  将 C 分裂成 Cn 和 C′
16.  add Cn to B′[k] and C′ to B′[i]
17.  if ∃Ct∈C.plist s.t. Ct.upp=Cn.upp then
18.     Cn.m=Agg(Ct.m+t.m),delete Ct
19.  else 产生一个隐含的新等价类 Cp
20.  if ∃Ct∈C.plist s.t. Ct.upp=Cp.upp then
21.     Ct.m=Agg(Ct.m+t.m)
22.     if Ct.m-C.m≥δ then
23.        split C into Cn and C′
24.        Cn.m=Ct.m
25.        add Cn to B′[k] and C′ to B′[i],
26.        delete Ct
27.     else insert Cp to C.plist
```

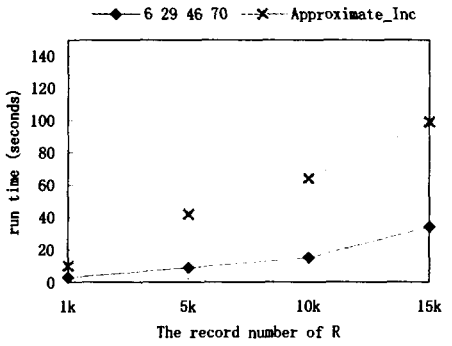

图 4　算法的可伸缩性比较

算法 Approximate_Inc 的第一行产生了一个虚类，第二行将所有的类分到 $B[0],\cdots,B[n+1]$ 个桶中。桶 $B[i]$ 包含所有满足条件 $|C.upp_\alpha|=i$ 的那些等价类。另一组桶 $B'[0],\cdots,B'[n]$ 用来存放 Φ′被修改或新生成的类（line 3）。

主循环（4～12 行）按照 $B[0],\cdots,B[n+1]$ 的顺序在每个桶内进行循环。对于桶 $B[i]$ 的每一个等价类 C，Approximate_Inc 首先看它是否是一个值改变的等价类，如果是，则对其进行修改。

如果不是值改变的等价类的话，则看它是否是一个新类产生器（9～12 行）。如果 C 是一个新类产生器，则将其进行分裂。最后输出所有的等价类。

5　性能分析

我们在不同的数据集合上做了大量的实验来验证上面给出的增量维护算法的性能和效率。实验所使用的机器是 Pentium4，内存 256 兆，操作系统是 WindowXP。采用的编程语言是 VC 6.0。

我们对近似综合数据方体增量维护算法 Approximate_Inc 进行了测试。为了进行性能上的比较，我们采用文[10]中的思想实现了算法 QC_Inc，它用于维护精确的综合数据方体。我们先生成了一个具有 10000 条元组的综合数据方体 R。然后分别采用上述两种方法来对其进行维护。我们将更新率（update ratio）从 4%变到 100%。更新率 k%表示 $|\triangle T|=(k\%)*|R|$ 的元组被插入。图 3 显示了两种算法的运行时间。可以看出，当 ΔR 比较小的时候，算法 Approximate_Inc 和 QC_Inc 都表现很好，随着 ΔR 的增加，QC_Inc 的运行时间迅速增长，而 Approximate_Inc 却增长比较缓慢。原因是当 ΔR 增加的时候，由 ΔR 所产生的等价类迅速增加，从而导致 QC_Inc 的性能迅速下降。

在另外一个实验里面，我们测试了这两种算法的伸缩性。将更新率固定在 20%，将总的元组个数 R 从 1k 增加到 15k。图 4 显示了实验的结果。可以看出，算法 Approximate_Inc 的伸缩性比算法 QC_Inc 要好。

结论　本文提出了近似综合数据方体的概念。在这种方体中，度量值可以在用户指定的一个误差范围内浮动，导致产生的等价类个数比较少，从而可以在很大程度上减少数据方体的体积，提高了查询效率和存储效率。我们同时提出了有效的对该种数据方体进行维护的算法：Approximate_Inc。实验证明该方法非常有效。

参 考 文 献

1 Agarwal S, et al. On the computation of multidimensional aggregates. In: Proc. of the 22st Int'l Conf. on Very Large Databases (VLDB), 1996

2 Zhao Y, et al. An array-based algorithm for simultaneous multidimensional aggregates. In: Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, 1997

3 Gray J, et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-tab, and Sub-Total. In: Proc. of 1996 Int'l Conf. on Data Engineering (ICDE), 1996

4 Harinarayan V, et al. Implementing data cubes efficiently, In: Proc. Of ACM-SIGMOD Int'l Conf. on Management of Data, 1996

5 Lakshmanan L V S, et al. Quotient Cube: How to Summarize the Semantics of a Data Cube. In Proc. of the 28st Int'l Conf. on Very Large Databases (VLDB), 2002

6 Wang W, et al. Condensed Cube: An Effective Approach to Reducing Data Cube Size. In: Proc. of 2002 Int'l Conf. on Data Engineering (ICDE), 2002

7 Sismanis Y, et al. Dwarf: Shrinking the Petacube. In: Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, 2002

8 Barbara D, et al. Quasi-cubes: Exploiting approximation in multidimensional databases. SIGMOD Record, 1997, 26: 12～17