封闭数据立方体技术研究*

李盛恩[1+], 王 珊[2]

[1](山东建筑工程学院 计算机科学与技术系,山东 济南 250014)

[2](中国人民大学 信息学院,北京 100872)

Research on Closed Data Cube Technology

LI Sheng-En[1+], WANG Shan[2]

[1](Department of Computer Science and Technology, Shandong Institute of Architecture and Engineering, Ji'nan 250014, China)

[2](School of Information, Renmin University, Beijing 100872, China)

+ Corresponding author: E-mail: lsn@sdai.edu.cn, http://www.sdai.edu.cn

Received 2003-05-23; Accepted 2003-09-26

Li SE, Wang S. Research on closed data cube technology. *Journal of Software*, 2004,15(8):1165~1171.
http://www.jos.org.cn/1000-9825/15/1165.htm

Abstract: There is a lot of redundant information in a data cube. Removing redundancy from a data cube can not only reduce the storage space but also accelerate the computation. Tuples of a data cube can be divided into closed-tuples and non-closed tuples. For any non-closed tuple, there exists a closed-tuple, and both are aggregated from the same set of tuples in a base table and have the same aggregated value. By removing all non-closed tuples, a data cube can be translated to a closed data cube. The algorithm of computing a closed data cube is given, answering a query and maintaining the closed data cube incrementally. The results of experiments are also presented by using both the synthetic and real-world data sets. The experimental results show that the closed data cube technique is effective.

Key words: data warehouse; on-line analytical processing; data cube; incrementally maintenance; compression

摘 要: 数据立方体中有很多冗余信息,去除这些冗余信息不但可以节约存储空间,还可以加快计算速度.数据立方体中的元组可以划分为封闭元组和非封闭元组.对任何一个非封闭元组,一定存在一个封闭元组,它们都是从基本表的同一组元组中经过聚集运算得到的,因而具有相同的聚集函数值.去掉数据立方体中所有的非封闭元组就产生了一个封闭数据立方体.提出了封闭数据立方体的生成算法、查询算法和增量维护算法,并使用合成数据和实际数据做了一些实验.实验结果表明,封闭数据立方体技术是有效的.

关键词: 数据仓库;联机分析处理;数据立方体;增量维护;压缩

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60273017 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA4Z3420 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2001CCA03003 (国家重点基础研究发展规划(973))

作者简介: 李盛恩(1963－),男,山东蓬莱人,博士,副教授,主要研究领域为数据仓库,联机分析,数据挖掘;王珊(1944－),女,教授,博士生导师,主要研究领域为数据仓库,联机分析,数据挖掘,Web 数据管理,网格计算.

数据立方体(data cube)[1]是联机分析中的一个非常重要的操作符,一个 n 维数据立方体就是 2^n 个 group-by 的并.文献[1~4]研究了一般的计算方法.其主要思想是通过共享输入、共享排序结果来减少读磁盘的次数、字节数和计算时间.文献[5,6]考虑通过减少实例化视图的个数来减少数据立方体的体积,从而减少写磁盘的时间.Iceberg[7]对进入数据立方体的元组施加了限制条件,减少了数据立方体的元组个数.Iceberg 以及选择实例化视图的方法只是计算了数据立方体中的部分元组,当需要查询那些没有被计算出的元组时,需要实时计算.最近,研究工作者提出了 Condensed Cube[8],Quotient Cube[9]和 Dwarf[10],其基本思想是通过共享元组来压缩数据立方体的体积,而且不需要实时计算.

通过对 Quotient Cube 的分析我们发现,可以只保留每个类的上界而去掉所有的下界,这样基本不影响查询性能,又进一步节省了存储空间,更重要的是,可以用关系作为存储结构,直接用于主流的关系系统.我们提出了封闭数据立方体的概念,给出了生成算法,讨论了查询处理和增量维护两个关键问题,并给出了相应的算法.实验结果表明,我们的思想是可行的、有效的.图 1 给出了一个 Quotient Cube 和封闭数据立方体(closed data cube)的示例,图中用*代表文献[1]提出的特殊值 All.

Base table

A	B	C	M
0	0	0	10
0	0	1	30
0	1	1	20
0	1	2	60
0	2	3	40

Quotient cube

Upper bound	Lower bound
(0, 0, 0, 10)	(*, *, 0, 10)
(0, 0, 1, 30)	(*, 0, 1, 30)
(0, 1, 1, 20)	(*, 1, 1, 20)
(0, 1, 2, 60)	(*, *, 2, 60)
(0, 2, 3, 40)	(*, 2, *, 40), (*, *, 3, 40)
(0, 0, *, 40)	(*, 0, *, 40)
(0, 1, *, 80)	(*, 1, *, 80)
(0, *, 1, 50)	(*, *, 1, 50)
(0, *, *, 160)	(*, *, *, 160)

Closed data cube

A	B	C	M
0	0	0	10
0	0	1	30
0	1	1	20
0	1	2	60
0	2	3	40
0	0	*	40
0	1	*	80
0	*	1	50
0	*	*	160

Fig.1 Quotient cube and closed data cube
图 1 Quotient cube 和 closed data cube

1 基本概念和生成算法

假设关系 R 的模式是 $R(A_1,A_2,...,A_n,M)$,其中 A_i 作为维属性,$1\le i\le n$,M 作为度量属性,$A_1A_2...A_n$ 是 R 的关键字.用 C 表示由 R 产生的数据立方体.

定义 1. $t_1\in C,t_2\in C,\forall A_i,1\le i\le n$,如果满足以下条件,则称 t_2 覆盖 t_1 或 t_1 被 t_2 覆盖:

- 如果 $t_2(A_i)\neq$All,则 $t_1(A_i)=t_2(A_i)$;
- 如果 $t_2(A_i)=$All,则 $t_1(A_i)$可以取任意值.

例如,元组(*,0,*,40)覆盖元组(*,0,1,30),(*,0,0,10),(0,0,*,40),(0,0,1,30),(0,0,0,10),(*,0,*,40).

定义 2. 元组 t 的基本元组集 $BTS(t')=\{t'|t'\in R,t\in C,t$ 覆盖 $t'\}$.

例如,BTS((0,1,*,80))={(0,1,1,20),(0,1,2,60)},BTS((0,1,1,20))={(0,1,1,20)}.

由定义 1 和定义 2 可知,如果 t_2 覆盖 t_1,则 $BTS(t_1)\subseteq BTS(t_2)$.

定义 3. 对于元组 $t_1\in C$,如果不存在元组 $t_2\in C,t_2\neq t_1,t_1$ 覆盖 t_2,并且 $BTS(t_1)=BTS(t_2)$,则称元组 t_1 为封闭元组.

例如,元组(*,*,*,160)不是封闭元组,因为它覆盖元组(0,*,*,160),并且它们的 BTS 相等.元组(0,*,*,160)是封闭元组.

因为 $A_1A_2...A_n$ 是 R 的关键字,对任意的 $t\in R,BTS(t)=\{t\}$,即 t 不会覆盖除了 t 以外的其他元组,所以 R 中的所有元组都是封闭元组.

定义 4. 封闭数据立方体 $C_c=\{t|t\in C,t$ 是封闭元组$\}$.

引理 1. $t_1\in C$,如果存在一个维 $A_i,t_1(A_i)=$All,并且 $BTS(t_1)$中的所有元组在 A_i 上有相同的值 $c,c\neq$All,则 t_1 不是一个封闭元组.

证明:略. □

引理 2. 设 t_1 是一个非封闭元组,t_2 是一个封闭元组,t_1 覆盖 t_2,并且在所有被 t_1 覆盖的封闭元组中,t_2 中出现 All 的次数最多,则 $BTS(t_1)=BTS(t_2)$.

证明:略. □

引理 2 说明,尽管在封闭数据立方体中不存储非封闭元组,但是对任意的聚集函数,我们都可以从某个封闭元组中得到非封闭元组的聚集函数值.

例如,元组(*,0,*,40)覆盖封闭元组(0,0,*,40),(0,0,1,30)和(0,0,0,10),因为元组(0,0,*,40)中出现 All 的次数最多,所以 *BTS*(*,0,*,40)=*BTS*(0,0,*,40),二者的聚集函数值相等.

下面我们给出计算封闭数据立方体的算法 ClosedCube.其基本思想是产生数据立方体的所有元组,对每一个元组,根据引理 1 判断它是否是封闭元组,如果是封闭元组则输出它,否则不输出.

变量 *gtuple* 表示数据立方体中的一个元组,*bts* 是 *gtuple* 的基本元组集,*dim* 是维的编号,*numDims* 是维的个数.当第 1 次调用时,*gtuple*=(All,All,…,All),*bts*=*R*,*dim*=0.在第 2~8 行,按照引理 1 构造一个封闭元组,存放在变量 *closedTuple* 中.如果 *gtuple* 是一个封闭元组,即 *gtuple* 等于 *closedTuple*,则进行聚集计算并输出 *gtuple* 和聚集函数值(第 9~12 行).处理完一个元组以后,继续去产生其他元组(第 13~20 行),其过程类似于算法 BUC[7].第 1 行是一种优化措施.因为|*bts*|=1,如果 *gtuple* 中没有出现 All,则 *gtuple* 是 *R* 中的一个元组,一定是一个封闭元组,但是没有必要输出它,因为它已经在 *R* 中了.如果 *gtuple* 的某个维属性的值等于 All,由于它只覆盖了 *R* 中的一个元组,由引理 1,*gtuple* 不是封闭元组,不需要输出它.另外,由 *gtuple* 产生的其他元组也不是封闭元组.

定理 1. 算法 ClosedCube 输出了所有的封闭元组一次且仅有一次.

证明:略. □

```
Algorithm ClosedCube(bts,dim)
1.  if |bts|==1 then return
2.  for (d=0;d<numDims;d++) do
3.      if bts[d] has unique value then
4.          closedTuple[d]=bts[d]
5.      else
6.          closedTuple[d]=All;
7.      end if
8.  end for
9.  if gtuple==closedTuple then
10.     v=aggregate(bts);
11.     write closedTuple and v;
12. end if
13. for (d=dim; d<numDims; d++) do
14.     c=Partition(bts,d,P);
15.     for (i=0; i<c; i++) do
16.         gtuple[d]=P_i[d];
17.         ClosedCube(P_i,d+1);
18.     end for
19.     gtuple[d]=All;
20. end for
```

2 查询处理

在 Quotient Cube 中查找一个元组的过程可以分解为两步:第 1 步,利用某种存取方法(例如索引)从外存上找到满足上界条件的类,并把这些类一一读到内存;第 2 步,在内存中利用下界作最后的判断.假设读一个类到内存需要一次 I/O 操作,有 *n* 个满足条件的类,并且元组属于每个类的概率相同,则平均查找代价为

$$Cost_{QC}=n/2 \quad (1)$$

在封闭数据立方体中查找元组时,因为封闭元组就是 Quotient Cube 中的上界,因此可以用与 Quotient Cube 相同的方法找到满足条件的封闭元组.因为在封闭数据立方体中没有存储下界,由引理 2,我们只有把 n 个封闭元组全部读入内存,经过检查后才能作出正确的判断,因此平均查找代价为

$$Cost_{CC}=n \tag{2}$$

数据立方体的所有 cuboid 形成了一个格[2],一个具有 n 个维的数据立方体有 2^n 个 cuboid,分别属于 n+1 个层,第 i 层由包含 i 个维的 cuboid 组成.图 2 是一个具有 3 个维的数据立方体的格.有了格的概念后,由引理 2,当查找元组 q 的聚集函数值时,我们应该按照由低层到高层的次序查找,如果在某一层找到了被 q 覆盖的封闭元组,就不必再到更高层去查找.但是,同一层有多个 cuboid,每个 cuboid 中的元组有相同个数的维的值等于 All,那么是否要对同一层的 cuboid 都检查一遍呢?引理 3 回答了这个问题.

引理 3. 如果按照第 i,i+1,…,n+1 层的次序查找被 t 覆盖的封闭元组,并且如果在第 $j(j\geq i)$层找到,就不到第 j+1 层继续查找,则第 $k(k\geq i)$层最多只有一个封闭元组被 t 覆盖.

证明:略. □

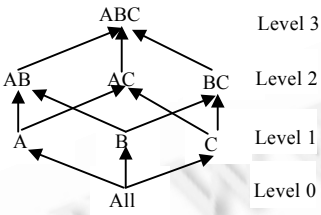

Fig.2 Data cube lattice
图 2 数据立方体格

根据引理 3,在检查 n 个满足条件的封闭元组时,不必把它们全部读入内存,可以按照一定的次序一个一个地读,当找到一个满足条件的封闭元组时就停止检查,因此平均查找代价与 Quotient Cube 相同.

根据引理 2 和引理 3,我们设计了查找算法 HQ.在算法中,用 $SV(c)$表示视图 c 以及它的祖先集合,$SV(c)$中的视图按照它所在层的层号从小到大排列.算法在每个 cuboid 中查找是否有被 q 覆盖的封闭元组(第 1~4 行),如果找到了一个,则取出它的聚集函数值作为结果返回(第 3 行),如果在所有的 cuboid 中都没有找到,则返回一个空值(第 5 行).

Algorithm HQ(q,$SV(c)$)

1. for each view $v\in SV(c)$ do
2. Scan v to find a tuple t, which covered by q
3. if found then return $t(M)$
4. end for
5. return null

对于范围查询,我们提出了一个逐步分解、分层查找的算法.假设范围查询 rq 在维 $A_1,A_2,...,A_i$ 上的查询条件是用枚举形式表示的一个范围,其他维上的查询条件是一个具体值,即 $rq=(\{a_{11},a_{12},...,a_{1m}\},\{a_{21},a_{22},...,a_{2n}\},...,\{a_{i1},a_{i2},...,a_{iz}\},a_{i+1},...,a_n)$.限于篇幅,我们仅用下面的例子来说明算法的思想,省略了算法的具体描述.

例如,针对表 1 中的封闭数据立方体有一个查询 $rq=(\{0,1\},\{1,2\},*)$,我们首先在 cuboid AB 中查找满足条件的元组 t,$t(A)\in\{0,1\}$,$t(B)\in\{1,2\}$,结果是(0,1,*,80),然后分解 rq,得到两个查询(0,2,*)和(1,{1,2},*).到下一层的 cuboid ABC 中查找满足查询(0,2,*)的元组,结果是(0,2,3,40),满足查询(1,{1,2},*)的元组为空.最终返回的查询结果是{(0,1,*,80),(0,2,3,40)}.这个查询可以分解为 4 个点查询,而用我们的方法,查询(1,{1,2},*)根本不可能被满足,因此没有必要对它再分解,只向系统提交 3 个查询,减少了 1 个.

3 增量维护

当基本表中的元组发生变化时,需要对封闭数据立方体进行增量维护[1].引理 4 给出了增量维护的基本原理,即子集中的封闭元组一定是超集中的封闭元组.

引理 4. 假设基本表 R_1 和 R_2 具有相同的模式($A_1,A_2,...,A_n,M$),其中 $A_i(1\leq i\leq n)$是维属性,M 是度量属性,$R_1\subseteq R_2$,由 R_1 和 R_2 生成的数据立方体分别是 C_1 和 C_2.如果 t_1 是 C_1 中的封闭元组,则 C_2 中存在一个封闭元组 t_2,$t_1(A_i)=t_2(A_i)$,$1\leq i\leq n$.

证明:略. □

算法 MPLUS 首先使用 ClosedCube 计算新增加的元组集合ΔR 所生成的封闭数据立方体 $C_{\Delta R}$,由引理 4,$C_{\Delta R}$

中的元组一定属于 $R\cup\Delta R$ 生成的封闭数据立方体.然后使用算法 BUC[7]计算出由ΔR 构成的数据立方体中的所有元组集合 C(第 1 行).$C_P=C-C_{\Delta R}$ 为非封闭元组,对任意 $t\in C_P$,由引理 1,一定存在一个属性 A_i,$t(A_i)$=All,$BTS_{\Delta R}(t)$中的所有元组在属性 A_i 上有相同的值 c,$c\neq$All,但是 $BTS_{\Delta R}(t)\subseteq BTS_{\Delta R+R}(t)$,所以 t 在由 $R\cup\Delta R$ 生成的数据立方体中有可能是封闭元组,第 3~7 行找出所有这样的元组并存放到 C_n 中.最后把 $C_{\Delta R}$ 和 C_n 中的元组安装到封闭数据立方体 cube 中.

算法 MMINUS 是针对删除一个元组而设计的增量维护算法.其核心是第 1 步,删除 t 以后可能影响到的元组一定存在于由 R_1 生成的数据立方体中.算法的其他部分很容易理解,在此不再赘述.

Algorithm MPLUS(cube,ΔR)

1. $C_{\Delta R}=ClosedCube(\Delta R,0)$, $C=BUC(\Delta R,0)$
2. $C_n=\varnothing$, $C_P=C-C\Delta R$
3. for each tuple t in C_P do
4. gather $BTS(t)$ from $R\cup\Delta R$
5. $C_{tmp}=ClosedCube(BTS(t),0)$
6. if t exists in C_{tmp}, then $C_n=C_n\cup\{t\}$
7. end for
8. $UpdateOrInsert(cube,C_{\Delta R})$, $Insert(cube,C_n)$

Algorithm $MMINUS(cube,t)$

1. $R_1=\sigma_F(R)$, $F=t(A_1)\vee t(A_2)\vee\ldots\vee t(A_n)$
2. $C_1=ClosedCube(R_1,0)$
3. $R_2=R_1-\{t\}$
4. $C_2=ClosedCube(R_2,0)$
5. $C_d=C_1-C_2$
6. $Delete(cube,C_d)$, $Update(cube,C_2)$

4 实验结果与分析

为了验证封闭数据立方体的有效性,我们使用合成数据和实际数据做了一组实验,在所有的实验中,聚集函数为 *SUM*,计算结果存放在文本文件中.由于文献[9]没有涉及到查询处理和增量维护,我们只比较了二者在生成速度上的差异,并比较了封闭数据立方体和完整数据立方体的查询响应时间.我们尽最大的努力实现了文献[9]中的 DFS 算法,与原算法不同的是只输出上界而没有输出下界,这样更有利.

实验 1 验证了数据稀疏对算法的影响.我们生成了符合均匀分布的 4 个数据集,每个数据集有 6 个维,每个维的基数(不同维值的个数)为 100,元组个数分别是 1000,10000,100000,1000000.结果如图 3 所示.随着元组个数的增加,算法产生的类的个数也将增加,因此执行时间也有所增加.因为 DFS 会产生重复的类,需要通过外排序合并这些类,随着类个数的增加,DFS 用于合并的时间大幅度地增加.

实验 2 比较了数据偏斜对算法的影响.我们产生了 7 个符合 Zipf 分布的数据集,每个数据集有 6 个维,每个维的基数为 100,元组个数为 1 000 000,zipf 因子从 0 到 3.0,因子越大,数据偏斜程度越大.实验结果如图 4 所示.数据越偏斜,某些维值出现的频率越高,每个维出现的不同维值的个数越小,因此,类的个数也将变少.算法 ClosedCube 的执行时间波动不大,而 DFS 受数据偏斜的影响比较大.

实验 3 使用了一个实际数据集 *weather*[12].*weather* 数据集被多个算法[4,7~10]所采用,它有 1 015 367 个元组(大约 27.1MB),9 个维.我们通过投影的方法从 *weather* 数据集中产生了 8 个数据集,它们的维数是 2,3,…,9,即前 2 个,3 个,…,9 个维的投影,元组个数、每个维的基数都相同,实验结果如图 5 所示.随着维数的增加,类的个数增加,两个算法的运行时间都增加,但是 DFS 增加的幅度比较大.

实验 4 比较了封闭数据立方体和完整数据立方体在查询性能上的差异.我们使用实验 3 中具有 3 个、5 个和 7 个维的数据集做了实验.用算法 BUC[7]在每个数据集上计算了一个完整数据立方体,每个 cuboid 存储在一

个文件中,3 个数据集占用的磁盘存储空间分别是 8.97,93.1 和 984M.然后,用 ClosedCube 算法产生了 3 个封闭数据立方体,同样,每个 cuboid 存储在一个文件中,占用的磁盘存储空间分别是 4.71M,20.3M 和 83.6M.

针对每个数据集,我们分别随机地产生了 1 000 个点查询,然后用算法 HQ 来回答查询,没有采用任何索引,而是用顺序查找的方法,但是基本表常驻内存.1 000 个查询的总的响应时间如图 6 所示.在理论上,封闭数据立方体的平均查询响应时间应该比完整数据立方体要慢.在 3 个维和 5 个维的情况下,完整数据立方体的查询响应时间比封闭数据立方体慢一些,前者分别是后者的 6.14 倍和 1.46 倍,主要原因是 cuboid 个数比较少(8 个和 32 个),每个 cuboid 中的元组数又比完整数据立方体中的 cuboid 少.另外,基本表常驻内存也有利于改善封闭数据立方体的查询性能.在 7 个维的情况下,封闭数据立方体的查询响应时间要慢一些,是完整数据立方体的 1.16 倍,因为要在很多的 cuboid 中查找满足条件的元组.

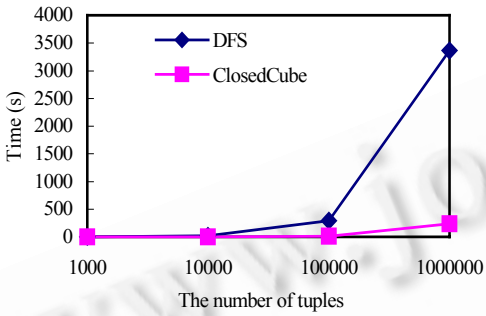

Fig.3 The effect of sparse
图 3 数据稀疏的影响

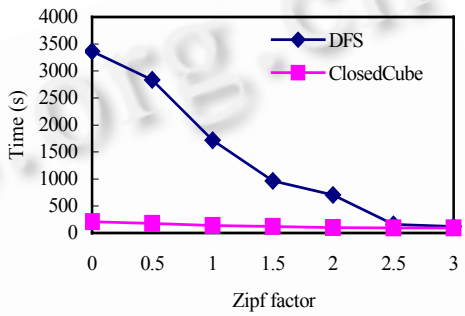

Fig.4 The effect of skew
图 4 数据偏斜的影响

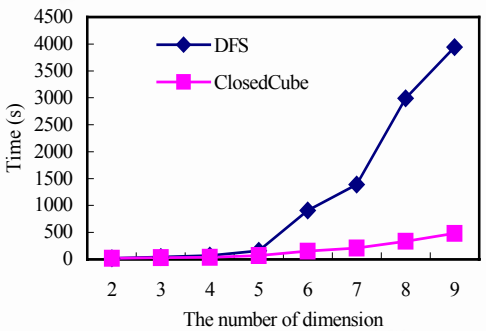

Fig.5 The weather data set
图 5 天气数据集

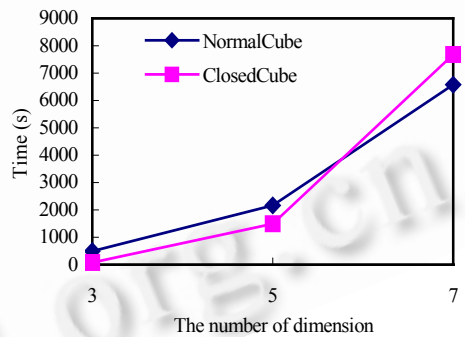

Fig.6 Query response time
图 6 查询响应时间

5 相关工作

我们的工作与文献[1~10]有关,与文献[8~10]密切相关.下面重点说明本文与这 3 篇文献的关系.

Dwarf 对数据立方体实行了前缀和后缀压缩,其前缀压缩是一项传统的技术,后缀压缩是封闭数据立方体的特例.我们也可以利用它的前缀压缩技术进一步缩小封闭数据立方体的体积,但是,为了实现前缀压缩,Dwarf 采用了特殊的链表结构,使得它不能直接用于关系系统.

封闭数据立方体实际上是去除下界的 Quotient Cube,但是占用的存储空间更少,可以直接应用于关系系统,使用关系系统中成熟的索引技术.理论分析表明,封闭数据立方体和 Quotient Cube 的查询效率基本相同,实验结果表明,生成封闭数据立方体的速度更快.另外,我们提出了增量维护算法和层次查询处理的技术,并通过实验研究了封闭数据立方体与普通数据立方体在查询响应时间上的差异,而文献[9]没有涉及这些问题.需要指出的是,使用文献[9]的 connection partition 原理可以产生体积比封闭数据立方体更小的 Quotient Cube,但是目前只能用于像 sum,count 等这类具有反单调性的函数,不具有通用性.

Condensed Cube 的基本原理与 Quotient Cube 相同,生成算法速度很慢,可以用 ClosedCube 算法快速地计算 Condensed Cube.因为如果数据立方体中的一个元组不被基本表中的一个元组所封闭,就必须计算并存储到 Condensed Cube 中,所以,只要修改 ClosedCube 算法第 9 行的判断条件即可.

6 结 语

本文提出了封闭数据立方体的概念,并给出了生成算法、增量维护算法和查询处理方法.理论分析和实验结果表明,封闭数据立方体大大减少了数据立方体的体积,在查询响应时间上的延迟也处于可以接受的范围,而且封闭数据立方体可以直接用于关系系统.我们今后的主要工作是考虑当采用其他物理存储结构(如文献[13]所述)时,如何发展新的查询处理技术,进一步加快查询响应时间.另外,如何在封闭数据立方体中保留一些冗余信息,适当地牺牲压缩率,使得查询响应时间基本与完整数据立方体相同,也是我们感兴趣的一个研究方向.

References:

[1] Gray J, Bosworth A, Layman A, Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In: Su SYW, ed. Proc. of the 12th Int'l Conf. on Data Engineering. New Orleans: IEEE Computer Society, 1996. 152~159.

[2] Agarwal S, Agrawal R, Deshpande PM, Gupta A, Naughton JF, Ramarkrishman R, Sarawagi S. On the computation of multidimensional aggregates. In: Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL, eds. Proc. of the 22nd Int'l Conf. on Very Large Data Bases. Mumbai: Morgan Kaufmann, 1996. 506~521.

[3] Zhao Y, Deshpande PM, Naughton JF. An array-based algorithm for simultaneous multidimensional. In: Peckham J, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Tucson: ACM Press, 1997. 159~170.

[4] Ross KA, Srivastava D. Fast computation of sparse datacubes. 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. Athens: Morgan Kaufmann, 1997. 116~125.

[5] Harinarayan V, Rajaraman A, Ullman JD. Implementing data cubes efficiently. In: Jagadish HV, Mumick IS, eds. Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data. Montreal: ACM Press, 1996. 205~216.

[6] Shukla A, Deshpande PM, Naughton JF. Materialized view selection for multidimensional datasets. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th Int'l Conf. on Very Large Data Base. New York: Morgan Kaufmann, 1998. 488~499.

[7] Beyer K, Ramakrishnan R. Bottom-Up computation of sparse and iceberg CUBEs. In: Delis A, Faloutsos C, Ghandeharizadeh S, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Philadelphia: ACM Press, 1999. 359~370.

[8] Wang W, Lu HJ, Feng JL, Yu JX. Condensed cube: An effective approach to reducing data cube size. In: Proc. of the 18th Int'l Conf. on Data Engineering. San Jose: IEEE Computer Society, 2002. 155~165.

[9] Lakshmanan LVS, Pei J, Han JW. Quotient cube: How to summarize the semantics of a data cube. In: Bressan S, Chaudhri AB, Lee ML, Yu JX, Lacroix Z, eds. Proc. of the 23rd Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann, 2002. 778~789.

[10] Sismanis Y, Deligiannakis A, Roussopoulos N, Kotidis Y. Dwarf: Shrinking the PetaCube. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM Press, 2002. 464~475.

[11] Mumick IS, Quass D, Mumick BS. Maintenance of data cubes and summary tables in a warehouse. In: Peckham J, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Tucson: ACM Press, 1997. 100~111.

[12] Hahn C, Warren S, London J. Edited synoptic cloud reports from ships and land stations over the globe. 1996. http://cdiac.esd.ornl.gov/cdiac/ndps/ndp026b.html

[13] Feng JH, Jiang XD, Zhou LZ. An improved multi-dimensional storage structure for data warehousing. Journal of Software, 2002,13(8):1423~1429 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1423.pdf

附中文参考文献:

[13] 冯建华,蒋旭东,周立柱.用于数据仓储的一种改进的多维存储结构.软件学报,2002,13(8):1423~1429.http://www.jos.org.cn/1000-9825/13/1423.pdf