Star Cube——一种高效的数据立方体实现方法

李盛恩[1] 王 珊[2]
[1](山东建筑工程学院计算机系 济南 250014)
[2](中国人民大学信息学院 北京 100872)
(lsn@sdai.edu.cn)

摘 要 一个具有 n 个维的数据立方体有 2^n 个视图,视图越多,用于维护数据立方体的时间也就越长. 通过将维分成划分维和非划分维,数据立方体可以转换成star cube. star cube由一个综合表和那些仅包含划分维的视图组成. star cube使用前缀共享和元组共享技术不仅减少了所需的存储空间,还大大减少了计算和维护时间. 在把一个分片限制在一个I/O单位的条件下,star cube的查询响应时间与数据立方体基本相同. 实验结果也表明,star cube是一种在时空两方面均有效的数据立方体实现技术.

关键词 数据仓库;联机分析;数据立方体

中图法分类号 TP311.13

Star Cube —An Approach to Implementing Data Cube Efficiently

LI Sheng-En[1] and WANG Shan[2]
[1](*Department of Computer, Shandong Institute of Architecture and Engineering, Jinan* 250014)
[2](*Information School, Renmin University of China, Beijing* 100872)

Abstract A data cube of n dimensions has 2^n views. The more the views, the more the maintenance time of data cube. By dividing dimensions of a data cube into partition dimensions and no-partition dimensions, a data cube can be transformed to a star cube, which includes a summary table and views consisting of partition dimensions. Star cube uses prefix and tuple sharing technology to reduce the storage requirement, computation time, and maintenance time. By confining the size of a fragment to one I/O unit, the query response time of star cube is almost the same as data cube. The results of experiments also show that star cube is a promising way to implement data cube.

Key words warehouse; OLAP; data cube

1 引 言

决策支持系统需要在超大规模的数据仓库上执行复杂的查询,而查询响应时间必须很快以满足交互环境的需要. 为了解决这个问题,除了传统的查询优化和索引技术以外,人们又引入了数据立方体[1]预计算技术,一个 n 维的数据立方体就是 2^n 个group-by的并. 在实际应用中,数据立方体中的元组个数往往是基本表的几百倍或几千倍,要占用GB甚至是PB级的存储空间,花费很长的计算和维护时间. 由于这类系统一般需要24小时不间断的运行,可用于维护的时间很短,这就要求减少实例化视图的个数.

文献[1~4]研究了计算整个数据立方体的方法,主要的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]考虑了如何减少实例化视图的个数,既把存储空间限

收稿日期:2003-01-23;修回日期:2003-07-07
基金项目:国家"八六三"高技术研究发展计划基金项目(2002AA4Z3420);国家"九七三"重点基础研究发展规化基金项目(2001CCA03003)
© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

制在一个给定的范围内又使得平均查询响应时间最快. IceBerg[7] 对进入数据立方体的元组施加限制条件,从而减少数据立方体的元组个数. 由于实际数据一般是稀疏的关联的[8], 文献[9~11]采用共享元组的方法大大减少了数据立方体的体积. 理论分析表明,查询响应时间将变慢,需要新的查询优化技术,目前仍处于研究中. 文献[12]利用了数据压缩技术来减少数据立方体的体积. 文献[13]提出近似计算的方法. 除文献[5,6]以外,其他的方法虽然减少了数据立方体的体积,加快了数据立方体的计算,但是本质上仍然是计算完整的数据立方体,实例化视图的个数并没有减少.

在数据立方体中,有些维构成的多维空间是密实的,这些维被称为划分维,而另一些维构成的多维空间是稀疏的,称它们为非划分维[14]. 例如,在一个用于分析销售情况的数据立方体中,时间、地区和商品为维,销售量为度量,时间维构成的多维空间是密实的,因为一般情况下,每个地区在每个月都会销售商品,某种商品在每个月都会有销售量. 而地区和商品的组合可能是稀疏的,因为某些商品限定在特定的地区销售.

根据划分维和非划分维的概念,可以在组成数据立方体的视图集合 V 上定义一个等价关系 R,如果视图 v_1 与 v_2 有相同的划分维则 $v_1 R v_2$. 显然 R 是一个等价关系,可以由 R 导出 V 的一个划分. 例如,分别用 R, P 和 T 表示地区、商品和时间维,用 S 表示销售量,T 为非划分维,R 和 P 是划分维,销售数据立方体的 8 个视图可以划分为:{$RPTS$, RPS},{RTS, RS},{PTS, PS}和{TS, S}. 对每一个等价类,如果只实例化包含所有非划分维的视图,即 $RPTS$, RTS, PTS 和 TS,则可以减少实例化视图的个数. 因为需要实例化的视图都包含了非划分维和度量,可以把它们组织成一个 star cube:每种划分维的组合构成了一个实例化视图(materialized view, MV),非划分维和度量形成了综合表(summary table, ST),综合表和视图通过外键联接. 图 1 给出了销售数据立方体的 star cube 表示形式.

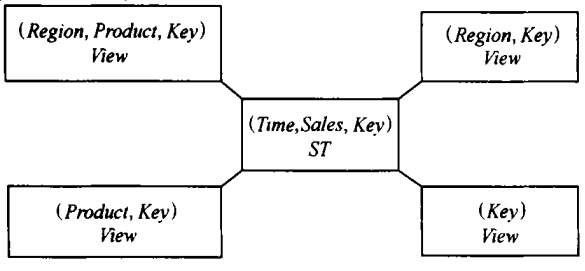

图 1 star cube

本文是这样组织的:第 2 节叙述了 star cube 的概念;第 3 节给出了一个基于流水线的 star cube 生成算法;第 4 节讨论了 star cube 的查询和增量维护问题;第 5 节给出了实验结果与分析;第 6 节总结了全文.

2 基本概念

我们将数据立方体中的维分为划分维和非划分维,可以利用语义信息或者用抽样的方法确定将一个维作为划分维还是非划分维,基本标准是划分维构成的一个维值关联了多个非划分维构成的维值. 在后面的叙述中,用 PD 和 ND 表示划分维和非划分维. 本节首先给出一些基本定义.

我们用商品销售作为全文的例子,表模式为 $Sale(T, R, P, S)$,T, R, P 和 S 分别表示时间、地区、商品和销售量,R 和 P 为划分维,T 为非划分维,表 $Sale$ 的一个实例如表 1.

表 1 基本表

T	R	P	S
t_1	r_1	p_1	10
t_2	r_1	p_1	20
t_1	r_2	p_2	10
t_2	r_2	p_2	50
t_1	r_2	p_3	10

定义 1. 从基本表 $R(PD, ND, M)$ 导出的综合表 ST 具有模式 $(ND, f(M), K)$,f 是聚集函数. 对任何的 $DS \subseteq PD$, $T_1 \subseteq R$, T_1 中的所有元组在 DS 上有相同的值 ds,则存在 $T_2 \subseteq ST$, $T_2(ND) = T_1(ND)$, T_2 中的所有元组在 K 上有相同的值 $h(ds)$,h 是一个函数,$h: PD \rightarrow N$,并且对任何 $t \in T_2$, $t(f(M)) = f(t_1(M), t_2(M), \cdots, t_k(M))$, $t_i \in T_1$, $t_i(ND) = t(ND)$.

定义 2. ST 中在 K 上有相同值的元组构成了一个分片.

定义 3. 实例化视图 V 具有模式 (DS, K), $DS \subseteq PD$, $V(DS) = R(DS)$, $K = h(DS)$, h 是一个函数,$h: PD \rightarrow N$.

定义 4. 一个 star cube 是从一个具有 n 个维的数据立方体中导出的,它有 m 个划分维,$n-m$ 个非划分维,划分维构成了 2^m 个视图,非划分维构成

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

了一个综合表.

定理 **1**. 如果star cube S 是从数据立方体D 导出的, 则 S 包含D 的任何一个元组.

证明. D 的视图可以分为 4 类: 包含划分维和非划分维; 只包含划分维; 只包含非划分维; 既不包含划分维又不包含非划分维. 我们对每一类视图分别给出证明.

(1) 对第 1 类视图 $V(DS, NS, f(M))$, $DS \subseteq PD$, $NS \subseteq ND$, f 是聚集函数, 对任何一个元组 t_1, $t_1 \in V$, 由定义 3, 存在 S 的视图 $V_S(DS, K)$, $t_2 \in V_S$, $t_1(DS) = t_2(DS) = ds$, 由定义 1, ST 有一个分片 C, $C(K) = t_2(K)$, C 是D 中所有在 DS 上值为 ds 的元组在ND 上的投影、聚集的结果. 如果 $NS = ND$, 则存在 $t_3 \in C$, $t_1 = \pi_{DS, ND}(t_2 \triangleright\triangleleft t_3)$, 否则, 对 C 中那些在NS 上有相同值的元组做聚集运算, 生成元组 t_4, $t_1 = \pi_{DS, NS}(t_2 \triangleright\triangleleft t_4)$.

(2) 对第 2 类视图 $V(DS, f(M))$, $DS \subseteq PD$, f 是聚集函数, 对任何一个元组 t_1, $t_1 \in V$, 由定义 3, 存在 S 的视图 $V_S(DS, K)$, $t_2 \in V_S$, $t_1(DS) = t_2(DS)$, 由定义 1, ST 有一个分片 C, $C(K) = t_2(K)$, 对 C 中的所有元组做聚集运算生成元组 t_3, $t_1 = \pi_{DS}(t_2 \triangleright\triangleleft t_3)$.

(3) 对第 3 类视图 $V(NS, f(M))$, $NS \subseteq ND$, f 是聚集函数, 对任何一个元组 t_1, $t_1 \in V$, 由定义 3, 存在 S 的视图 $V_S(K)$, $V_S(K)$有惟一的元组 t_2, 由定义 1, ST 有一个分片 C, $C(K) = t_2(K)$. 如果 $NS = ND$, 则存在 $t_3 \in C$, $t_1 = \pi_{ND}(t_2 \triangleright\triangleleft t_3)$, 否则, 对 C 中那些在NS 上有相同值的元组做聚集运算, 生成元组 t_4, $t_1 = \pi_{NS}(t_2 \triangleright\triangleleft t_4)$.

(4) 对第 4 类视图 $V(f(M))$, f 是聚集函数, V 中只有一个元组 t_1. 由定义 3, 存在 S 的视图 $V_S(K)$, $V_S(K)$有惟一的元组 t_2, 由定义 1, ST 有一个分片 C, $C(K) = t_2(K)$, 对 C 中的所有元组做聚集运算生成元组 t_3, $t_1 = \pi_{f(M)}(t_2 \triangleright\triangleleft t_3)$.

从star cube 的定义可以推导出 star cube 的两个特性:

(1) 假设数据立方体的维数为 n, 非划分维的个数为 m, 则数据立方体中的视图的个数是 star cube 中的视图个数的 2^m 倍.

(2) star cube 的视图中的元组个数要小于数据立方体中有相同维属性并包含所有非划分维的视图中的元组个数.

3 生成算法 *PBSC*

从上面的定义来看, star cube 是一个由划分维构成的变异数据立方体, 每个元组是一个分片, 所以生成一个 star cube 实际上是一个数据立方体计算过程. 根据 star cube 的特点, 我们设计了算法 *PBSC* (pipeline based star cube), 由 3 部分组成: *PBSC*(R, PD, ND), *Propagate*(B, P) 和 *Flash*(P), 其主要伪代码如下:

算法 **1**. PBSC 算法.

```
Input: base table, PD and ND;
Output: Star Cube;
① generate pipeline
② for each pipeline P do
③   find the sort order S;
④   sort base table on S;
⑤   put first block to node N0 of P;
⑥   for each block B in base table do
⑦     if π_DT(N0)(DVT(B)) = DVT(N0) then
⑧       aggregate B with N0;
⑨     else
⑩       Propagate(B, P);
⑪     end if
⑫   end for
⑬   flash(P);
⑭ end for
```

算法 **2**. Propagate 算法.

```
Input B, P;
① for each node Ni of P do
②   output DVT(Ni) to MV;
③   if HT(Ni) is not copied then
④     output HT(Ni) to ST;
⑤   end if
⑥   if π_DT(Ni+1)(DVT(Ni)) = DVT(Ni+1)
        or HT(Ni+1) = {} then
⑦     aggregate/put HT(Ni) with/to
        HT(Ni+1); k = i;
⑧     break
⑨   end if
⑩ end for
⑪ for each node Ni of P, 0< i ≤k do
⑫   Ni = Ni-1;
⑬ end do
```

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

⑭put B to N_0.

PBSC 是主要函数, 我们首先使用文献[4]中的方法生成由 *PD* 组成的流水线[行①], 每条流水线有若干个结点, 每个结点上记录了划分维表 *DT*, 划分维维值表 *DVT*, Hash 表 *HT* 等信息, 每个 Hash 表的大小等于非划分维基数之乘积. 对于表 1 的例子, 将产生两条流水线: $RP \rightarrow P \rightarrow All$ 和 R, 第 1 条流水线有 3 个结点, 分别为 N_0, N_1 和 N_2, 每个结点的划分维表为 *RP*, *P* 和空集, Hash 表的大小为 2, 因为时间维 *T* 的基数为 2.

生成流水线后, 处理所有的流水线[行②~⑭]. 对每一条流水线, 首先确定它的排序次序 *S*[行③], 然后对基本表按 *S* 排序[行④], 在 *S* 上有相同值的元组组成了一个块(block). 上述两条流水线的排序次序分别是 *PR* 和 *R*, 表 1 按 *PR* 排序后没有发生变化, *PR* 排序后形成了 3 个块.

将第 1 块传送到流水线的头结点 N_0 中[行⑤]后, 开始处理所有其他的块[行⑥~⑭]. 对每个块 B, 如果 $DVT(B)$ 在 N_0 的维上投影后的结果与头结点 N_0 中的划分维维值相同, 则将 B 与 $HT(N_0)$ 做聚集运算, 否则, 调用函数 *propagate*, 该函数是标准的流水线执行过程, 行③~⑤完成元组共享, 达到压缩综合表的目的.

当处理完所有的块后, 调用函数 *flash*(*P*)[行⑬]输出 *P* 中各结点中的数据, 其执行过程类似于 *propagate*.

例 **1**. 生成表 1 的 star cube.

首先, 执行流水线 $RP \rightarrow P \rightarrow All$. 流水线的排序次序为 *PR*, 将表 1 中的基本表按 *PR* 排序后, 产生了块 B_0, B_1 和 B_2, $DVT(B_0)=p_1r_1$, $DVT(B_1)=p_2r_2$, $DVT(B_2)=p_3r_2$. 将 B_0 复制到 N_0, $DVT(N_0)=p_1r_1$, $HT(N_0)=\{10, 20\}$. N_1 和 N_2 中的内容为空, 即 $DVT(N_1)=\{\}$, $DVT(N_2)=\{\}$, $HT(N_1)=HT(N_2)=\{\}$.

读入块 B_1, 因为 $DVT(B_1)\neq DVT(N_0)$, 执行函数 *propagate*. 生成关键字 $Key=1$, 输出 $DVT(N_0)$, 即元组 $(r_1, p_1, 1)$ 到视图 *RP*, 输出 $HT(N_0)$, 即 $(t_1, 1, 10)$ 和 $(t_2, 1, 20)$ 到综合表 *ST*[行②~⑤]. 因为 $HT(N_1)=\{\}$, 将 N_0 复制到 N_1, 结果是 $DVT(N_1)=p_1$, $HT(N_1)=\{10, 20\}$[行⑥~⑨]. 因为 $k=0$, 所以跳过了行⑪~⑬ 最后把 B_1 复制到 N_0, $DVT(N_0)=p_2r_2$, $HT(N_0)=\{10, 50\}$[行⑭].

读入块 B_2, 因为 $DVT(B_2)\neq DVT(N_0)$, 执行函数 *propagate*. 生成关键字 $Key=2$, 输出 $(r_2, p_2, 2)$ 到 *RP*, 输出 $(t_1, 2, 10)$ 和 $(t_2, 2, 50)$ 到 *ST*. 因为 $\pi_{DT(N_1)}(DVT(N_0))=p_2$, $DVT(N_1)=p_1$, 二者不相等, 输出 $(*, p_1, 1)$ 到视图 *P*, 因为 $HT(N_1)$ 是 $HT(N_0)$ 的复制结果, 所以不执行语句 4, 达到了共享元组的目的. 因为 $HT(N_2)$ 是空集, 复制 N_1 到 N_2, $DVT(N_2)=\{\}$, $HT(N_2)=\{10, 20\}$. 执行语句⑪~⑭后, $DVT(N_0)=p_3r_2$, $HT(N_0)=\{10\}$, $DVT(N_1)=p_2$, $HT(N_1)=\{10, 50\}$.

最后, 执行函数 *flash*(*P*). 处理 N_0, 输出 $(r_2, p_3, 3)$ 到 *RP*, 输出 $(t_1, 3, 10)$ 到 *ST*. 因为 $\pi_{DT(N_1)}(DVT(N_0))\neq DVT(N_1)$, 输出 $(*, p_2, 2)$ 到 *P*. 因为从 N_1 中去掉维 *P* 后 $DVT(N_1)$ 是空集, 所以 $\pi_{DT(N_2)}(DVT(N_1))=DVT(N_2)$, 聚集的结果是 $HT(N_2)=\{20, 70\}$. 复制 N_0 到 N_1, $DVT(N_1)=p_3$, $HT(N_1)=\{10\}$; 处理 N_1, 输出 $(*, p_3, 3)$ 到 *P*. 把 $HT(N_1)$ 聚集到 $HT(N_2)$ 后, $HT(N_2)=\{30, 70\}$; 处理 N_2, 输出 $(t_1, 4, 30)$ 和 $(t_2, 4, 70)$ 到 *ST*.

其次, 执行流水线 *R*. 流水线的排序次序为 *R*, 将表 1 中的基本表按 *R* 排序后, 产生了块 B_0 和 B_1, $DVT(B_0)=r_1$, $DVT(B_1)=r_2$. 将 B_0 送入 N_0, $DVT(N_0)=r_1$, $HT(N_0)=\{10, 20\}$.

读入块 B_1, 因为 $DVT(B_1)\neq DVT(N_0)$, 执行函数 *propagate*. 输出 $(r_1, *, 5)$ 到 *R*, 输出 $(t_1, 5, 10)$ 和 $(t_2, 5, 20)$ 到 *ST*. 将 B_1 存放到 N_0, $DVT(N_0)=r_2$, $HT(N_0)=\{20, 50\}$. 最后, 执行函数 *flash*(*P*), 输出 $(r_2, *, 6)$ 到 *R*, $\{t_1, 6, 20\}$ 和 $\{t_2, 6, 50\}$ 到 *ST*.

最后生成的 star cube 如表 2 所示, 综合表中每个分片的元组存放在相临的物理页面. 为了方便比较, 表 3 给出了由表 1 生成的数据立方体. 可以发现, star cube 视图的个数、视图的体积要比 data cube 的少.

4 查询处理和增量维护

在 star cube 中回答一个查询需要经过 3 个步骤: 首先取出查询中值不等于 *All* 的划分维集合 *D*, 找到维属性等于 *D* 的视图; 在视图中找到满足条件的元组, 取出 *K* 值; 根据 *K* 值到综合表中读出一个分片, 取出一个元组或进行聚集计算.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

表2 star cube

Views				Summary Table		
View	*R*	*P*	*K*	*T*	*K*	*Sum(S)*
RP	r_1	p_1	1	t_1	1	10
	r_2	p_2	2	t_2	1	20
	r_2	p_3	3	t_1	2	10
R	r_1	*	5	t_2	2	50
	r_2	*	6	t_1	3	10
P	*	p_1	1	t_1	4	30
	*	p_2	2	t_2	4	70
	*	p_3	3	t_1	5	10
All	*	*	4	t_2	5	20
				t_1	6	20
				t_2	6	50

表3 data cube

View	*T*	*R*	*P*	*Sum(S)*
RPT	t_1	r_1	p_1	10
	t_2	r_1	p_1	20
	t_1	r_2	p_2	10
	t_2	r_2	p_2	50
	t_1	r_2	p_3	10
RP	*	r_1	p_1	30
	*	r_2	p_2	60
	*	r_2	p_3	10
P	*	*	p_1	30
	*	*	p_2	60
	*	*	p_3	10
R	*	r_1	*	30
	*	r_2	*	70
RT	t_1	r_1	*	10
	t_2	r_1	*	20
	t_1	r_2	*	20
	t_2	r_2	*	50
PT	t_1	*	p_1	10
	t_2	*	p_1	20
	t_1	*	p_2	10
	t_2	*	p_2	50
	t_1	*	p_3	10
T	t_1	*	*	30
	t_2	*	*	70
All	*	*	*	100

例2. 回答查询, Q_1: 商品 p_1 在地区 r_1 的销售量. Q_2: 商品 p_1 在时间 t_1 的销售量. Q_3: 时间 t_1 的销售量.

对查询 Q_1, 首先在视图 RP 中找到元组$(r_1, p_1, 1)$, 到综合表中读出 $K = 1$ 的元组$(t_1, 1, 10)$, $(t_2, 1, 20)$, 做聚集运算, 结果为 30.

在视图 P 中查找 $P = p_1$ 的元组, 结果为$(*, p_1, 1)$, 然后从综合表中读出 $K = 1$ 的元组$(t_1, 1, 10)$, $(t_2, 1, 20)$, 找出 $T = t_1$ 的元组, Q_2 的结果为 10.

由于 Q_3 中不包含任何划分维, 因此, 读出视图 All 的惟一的分片, 有两个元组$(t_1, 4, 30)$, $(t_2, 4, 70)$, 找出 $T = t_1$ 的元组, Q_3 的结果为 30.

定理2. 在 star cube 中查找一个元组的平均 I/O次数为 $s + k$. 其中, s 是在视图中查找一个元组的平均 I/O 次数, k 是读一个分片的平均 I/O 次数.

证明 由查询处理过程可以得证.

因为查询响应时间基本由 I/O 次数决定, 所以可以近似地用 I/O 次数来表示查询性能. 假设在数据立方体的一个视图中查找一个元组的平均 I/O 次数为 z, 由 star cube 的性质 2 可以知道 $s \leqslant z$, 如果通过适当的选择非划分维, 使得一个分片的大小为一个 I/O 单位, 即 $k = 1$, 这时 star cube 的查询性能与数据立方体基本一样.

因为 star cube 是一个变异的数据立方体, 我们可以采用任何一种可以用于数据立方体的增量维护算法, 限于篇幅的限制, 不再详细论述这个问题, 感兴趣的读者, 可以参考文献[15]. star cube 中的视图个数大大小于数据立方体中视图的个数, 因此, 维护时间也会大大减少.

5 实验结果和分析

为了验证 star cube 的有效性, 我们使用实际数据集 weather[8] 进行了实验. 实验环境是在一台 Intel Pentium III 667Hz, 256MB 内存, 运行 Windows 2000 Professional 的 PC 机上执行的, 我们用 Microsoft Visual C++ 6.0 实现了所有的算法, 聚集函数为 Sum, 计算结果存放在文本文件中.

weather 数据集被多个算法所采用[4,7,9~11], 它有 1015367 个元组(大约 27.1MB), 9 个维: *station-id*(7037), *longitude*(352), *solar-altitude*(179), *latitude*(152), *present-weather*(101), *day*(30), *weather-change-code*(10), *hour*(8) 和 *brightness*(2), 括号中的数字是基数. 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

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

数, 每个维的基数都相同.

实验 1 比较了 star cube 和一般数据立方体的计算时间和占用的存储空间. 对每一个数据集, 我们用算法 *BUC*[7] 产生了一个完整的数据立方体, 用算法 *PBSC* 生成一个 star cube, 综合表中同一个分片的元组连续存放, 8 个数据集的非划分维集合分别是{2}, {3}, {4}, {5}, {6}, {6, 7}, {7, 8} 和{7, 8, 9}, 以保证每个分片的大小不超过一个 I/O 单位, 并且记录下计算时间和存储空间, 结果请见图 2 和图 3. 可见 star cube 占用的存储空间比一般的数据立方体要少, 随着维数的增加, 这种差异变得越明显, 这样就减少了写磁盘的次数, 因此, 计算 star cube 需要的计算时间也少.

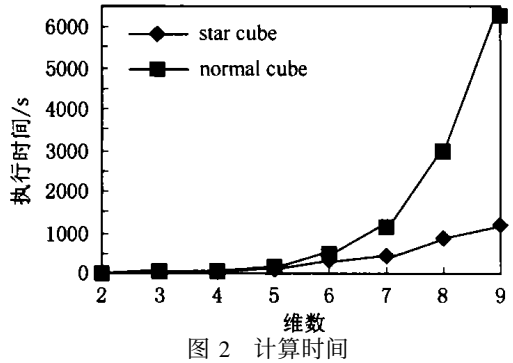

图 2 计算时间

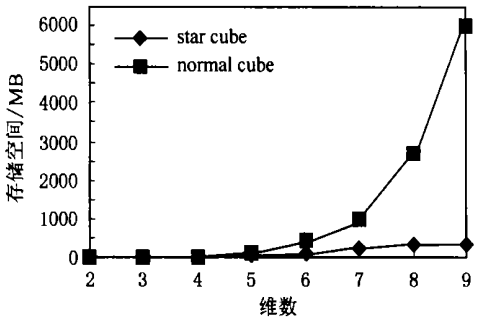

图 3 存储空间

图 4 是 star cube 中元组个数(视图和综合表) 与一般数据立方体中元组个数之比, 图 5 是 star cube 中元组个数(仅视图) 与一般数据立方体中元组个数之比, 从中可以看出, star cube 的元组个数要少于一般数据立方体的元组个数, 因此 star cube 占用的存储空间要少. 图 6 给出了由于元组共享, 综合表所节省的存储空间, 最多可以节省 19% 的存储空间.

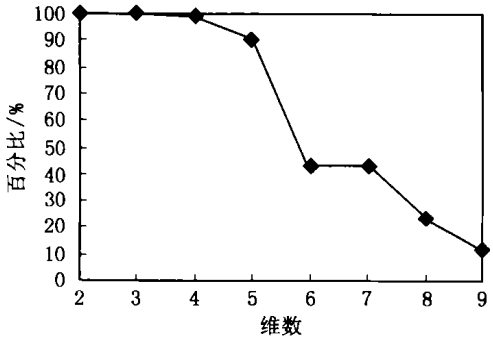

图 4 star cube 与数据立方体元组数之比

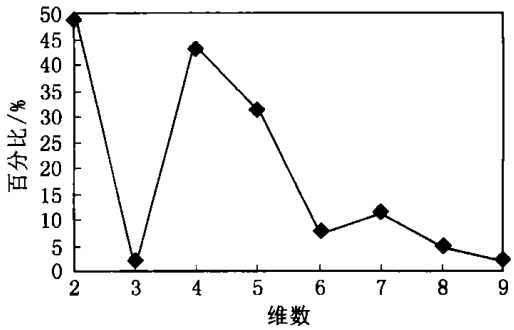

图 5 star cube 视图中的元组数与数据立方体之比

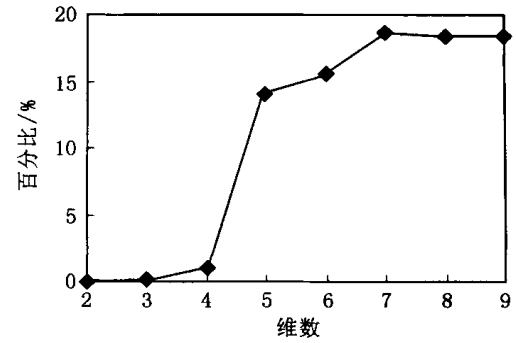

图 6 综合表的压缩率

实验 2 比较了 star cube 和数据立方体在查询性能上的差异. 我们使用实验 1 中具有 3、5 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的数据立方体, 每个视图存储在一个文件中. 然后用 *PBSC* 算法产生了 3 个 star cube, 同样, 每个视图存储在一个文件中. 针对每个数据集我们分别随机地产生了 1000 个点查询(point query). 然后用第 4 节描述的方法来回答查询, 没有采用任何索引, 而是用顺序查找的方法. 1000 个查询的响应时间请见图 7, star cube 的查询响应时间却比一般的数据立方体要快. 从图 5 中可以看出 star cube 视图中元组的个数与数据立方体的元组个数之比, 最多的为 50%, 最少的为 2%, 假设在数据立方体中查找一个元组需要读一次磁盘, 那么在 star cube 视图中查找一个元组只需要读 0.5 到 0.02 次磁盘. 对于 3 个维的数据集, star cube 视图中有 14427 个元组, 数据立方体中有 650482 个元组. 在 star cube 中查找一个元组, 首先在视图中查找, 假设读磁盘的次数为 s, 然后到综合表中读出分片, 需要 1 次 I/O, 总的 I/O 次数为 $s+1$, 由于 star cube 中元组个数很少, 实际上已经被操作系统缓冲, 因此, $s=$

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

0, 回答一个查询只需要 1 次 I/O, 而在数据立方体回答一个查询至少要读 1 次磁盘, 这就是本实验中 star cube 的查询响应时间比数据立方体快的原因.

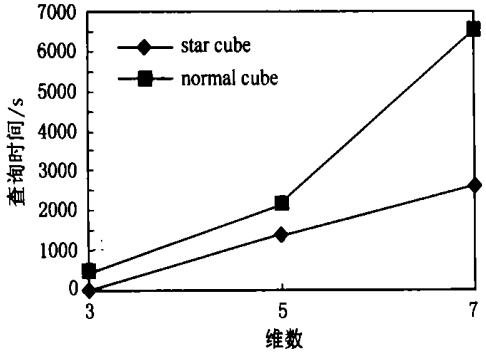

图 7 查询响应时间

6 结束语

本文提出了 star cube 的概念, 并给出了计算和查询处理方法. star cube 的视图个数由划分维个数决定, 由于对任何数据集, 至少可以用一个维作为非划分维, 因此, star cube 的视图个数最多是数据立方体的一半, 减少了维护时间. 当限制一个分片的大小为一个 I/O 单位时, 理论分析表明 star cube 的查询性能与数据立方体基本相同. 实验结果表明, star cube 是一种高效的数据立方体实现方法. 我们今后的主要工作是利用 Data Mining 技术来发现用哪些维作为非划分维, 如何组织流水线, 以便充分发挥前缀共享和元组共享的作用.

参 考 文 献

1 J Gray, A Bosworth, A Layman *et al*. Datacube: A relational aggregation operator generalizing groupby, cross-tab, and sub-total. In: Proc of the 12th Int'l Conf on Data Engineering. New Orleans: IEEE Computer Society Press, 1996. 152~ 159

2 S Agarwal, R Agrawal, P Deshpande *et al*. On the computation of multidimensional aggregates. In: Proc of the 22nd Int'l Conf on Very Large Data Bases. Mumbai, India: Morgan Kaufmann, 1996. 506~ 521

3 Y Zhao, P Deshpande, J F Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In: Proc of ACM SIGMOD Int'l Conf on Management of Data. Tucson, Arizona: ACM Press, 1997. 159~ 170

4 K A Ross, D Srivastava. Fast computation of sparse data cubes. In: Proc of the 23rd Int'l Conf on Very Large Data Bases. Athens, Greece: Morgan Kaufmann, 1997. 116~ 185

5 V Harinarayan, A Rajaraman, J D Ullman. Implementing data cubes efficiently. In: Proc of the ACM SIGMOD Int'l Conf on Management of Data. Le Centre Sheraton, Montreal: ACM Press, 1996. 205~ 227

6 A Shukla, P M Deshpande, J F Naughton. Materialized view selection for multidimensional datasets. In: Proc of the 24th Int'l Conf on Very Large Data Bases. New York City: Morgan Kaufmann, 1998. 488~ 499

7 K S Beyer, R Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In: Proc of the ACM SIGMOD Int'l Conf on Management of Data. Philadelphia, Pennsylvania: ACM Press, 1999. 359~ 370

8 C Hahn, S Warren, J London. Edited synoptic cloud reports from ships and land stations over the globe. 2002. http://cdiac.esd.ornl.gov/cdiac/ndps/ndp026b.html

9 Wei Wang, Hongjun Lu, Jianlin Feng *et al*. Condensed cube: An effective approach to reducing data cube size. In: Proc of the 18th Int'l Conf on Data Engineering. San Jose, California: IEEE Computer Society Press, 2002. 155~ 165

10 L V S Lakshmanan, Jian Pei, Jiawei Han. Quotient cube: How to summarize the semantics of a data cube. In: Proc of the 28th Int'l Conf on Very Large Data Bases. Hong Kong: Morgan Kaufmann, 2002. 766~ 777

11 Y Sismanis, A Deligiannakis, N Roussopoulos *et al*. Dwarf: Shrinking the PetaCube. In: Proc of the ACM SIGMOD Int'l Conf on Management of Data. Madison, Wisconsin: ACM Press, 2002. 464~ 475

12 J Li, D Rotem, J Srivastava. Aggregation algorithms for very large compressed data warehouses. In: Proc of the 25th Int'l Conf on Very Large Data Bases. Edinburgh, Scotland: Morgan Kaufmann, 1999. 651~ 662

13 D Barbara, M Sullivan. Quasi-cubes: Exploiting approximations in multidimensional databases. SIGMOD Record, 1997, 26(3): 12~ 19

14 G Colliat. OLAP, relational, and multidimensional database systems. SIGMOD Record, 1996, 25(3): 64~ 69

15 I S Mumick, D Quass, B S Mumick. Maintenance of data cubes and summary tables in a warehouse. In: Proc of the ACM SIGMOD Int'l Conf on Management of Data. Tucson, Arizona: ACM Press, 1997. 100~ 110

李盛恩 男, 1963 年生, 博士, 副教授, 主要研究方向为数据仓库、联机分析和数据挖掘.

王 珊 女, 1944 年生, 教授, 博士生导师, 主要研究方向为数据仓库、联机分析、数据挖掘和 XML (suang@public.bta.net.cn).

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net