函数依赖对商覆盖立方体生成效率的影响

彭湘凯[1],陈富强[1,2]
PENG Xiang-kai[1],CHEN Fu-qiang[1,2]

1.广东技术师范学院,广州 510665
2.华南理工大学 计算机科学与工程学院,广州 510641
1.Guangdong Polytechnic Normal University,Guangzhou 510665,China
2.School of Computer Science & Engineering,South China University of Technology,Guangzhou 510641,China
E-mail:paulpeng@163.net

PENG Xiang-kai,CHEN Fu-qiang.Computing cover quotient cube considering functional dependency in base tables. Computer Engineering and Applications,2009,45(34):134-135.

Abstract: This paper proposes a new algorithm DDFS to compute cover quotient cube.Functional dependency between base attributes are considered to improve the efficiency of computing.When the base table is partitioned with a particular attribute order,the attribute functionally depended is considered with a priority.For the attributes pair where there is a functional dependency,the cost of finding the congeneric functional dependency is retrenched.Experimental result on the dataset weather shows that the computing time associated with DDFS is only 76% of that with DFS.There is a conclusion that DDFS can shorten the time of computing cover quotient cube,compared with DFS.

Key words: data warehouse;cover quotient cube;functional dependency;Deep First Search(DFS)

摘 要:提出一种新的商覆盖立方体生成算法 DDFS。指出在基本表维属性之间可能存在函数依赖;分析了这种函数依赖对 DFS 算法的影响;按照决定子在前的原则调整了 DFS 算法对基本表进行水平分割时所依据的维属性的顺序;对于存在函数依赖的维属性对,节省了判断是否存在类函数依赖的操作。采用 weather 数据集进行的实验结果表明,采用 DDFS 计算商覆盖立方体所需时间仅为采用 DFS 算法时的 76%。得出 DDFS 较之 DFS 可以缩短商覆盖立方体生成时间。

关键词:数据仓库;商覆盖立方体;函数依赖;深度优先探索
DOI:10.3778/j.issn.1002-8331.2009.34.041 **文章编号**:1002-8331(2009)34-0134-02 **文献标识码**:A **中图分类号**:TP311

1 概述

数据立方体(Data Cube)是基于相应的基本表而生成的,而在 OLAP 系统中的基本表通常含有大量的元组,因此生成数据立方体通常需要耗费相当长的时间。为了提高生成的效率,研究者们提出了多种算法,其中 BUC 算法[1]的效率较高。Lakshmanan L.等提出了用 DFS 算法生成商覆盖立方体(Cover Quotient Cube)[2-3]。商覆盖立方体是一种压缩后的数据立方体,它是其原数据立方体的一个子集。一个数据立方体可以被划分为多个互不相交的类(class),每个类是一个或多个格(cell)的集合,同一个类中所有的格具有相同的基本元组集,因而对任意聚集函数具有相等的聚集值。在同一个类中具有最少 All 值的格被称为该类的上界,一个数据立方体中所有类的上界所形成的集合就是该数据立方体的商覆盖立方体。

研究目标是提出一种新的商覆盖立方体生成算法,这种算法综合考虑了基本表维属性之间的函数依赖,能在 DFS 算法的基础上进一步缩短生成所需的时间。

2 基本表维属性之间的函数依赖

在数据仓库中,数据的存储多为星形模式或雪花模式。在星形模式下,一个基本表与多个维表共同保存相关的数据,基本表中的一部分维属性是维表的外键,维表中以相应的属性为主键并包含其他的属性以共同保存相关的信息。理论上,当基本表和维表的结构较为规范化时,数据冗余较小,需要的存储空间较小,但在实际工作中,人们在数据仓库中存储数据时通常会对基本表的结构进行反规范化,即降低基本表结构的规范化程度,允许其中存在一定的数据冗余,这种冗余具体表现为在基本表模式中包括了维表中的一些非主键的属性。通过这种反规范化可以减少回答 OLAP 查询时所需连接基本表和维表的操作,从而缩短响应时间。

在对基本表进行计算生成相应的数据立方体时,也通常需要对基本表的结构进行反规范化。在 OLAP 查询中通常会包含聚集函数,而聚集函数的计算相当费时,为了避免在每次响应查询时都去计算相应的聚集函数,在数据仓库中常常采用数据

基金项目:广东省自然科学基金(the Natural Science Foundation of Guangdong Province of China under Grant No.8151063301000012)。
作者简介:彭湘凯(1967-),男,副教授,主要研究领域为数据库技术、网络技术;陈富强(1974-),男,副教授,博士生,主要研究领域为数据库技术。
收稿日期:2009-06-25 **修回日期**:2009-08-03

立方体来存储物化后的聚集函数查询结果。数据立方体是通过对基本表进行多次 Group by 操作而得到的，参与 Group by 操作的字段可能存在于基本表中，也可能存在于维表中，这时就需要对经过了反规范化的基本表进行计算得出相应的数据立方体。

对基本表进行如上所述的反规范化处理后，基本表的各个维属性之间常会存在函数依赖。给出一个简单的例子，表 1 所示的基本表是一个关于销售信息的二维表，其模式为 $TS(P,S,D,A)$，其中 P 表示产品，S 表示商店，D 表示日期，A 表示销售金额。P、S、D 三个维属性各有相应的维表存储相关细节信息，其中存储 S 相关信息的维表如表 2 所示，其模式为 $ATTS(sid,scity,sprovince)$，其中 sid 为主键，表示商店的编码，$scity$ 表示商店所在的城市，$sprovince$ 表示商店所在的省。

表 1　基本表 *TS*

P	*S*	*D*	*A*
P1	01	20010101	30
P2	01	20010223	60
P3	03	20020205	20
P3	02	20020109	10
P4	04	20010206	50

表 2　维表 *ATTS*

sid	*scity*	*sprovince*
01	广州	广东
02	深圳	广东
03	石家庄	河北
04	保定	河北

以 $SUM(\ )$ 为聚集函数，得到表 TS 的数据立方体 $CS(P,S,D,SUMA)$。假设用户要查询所有商品在广东省内的所有销售记录的合计金额，那么在 CS 中是无法直接得到数据的，因为在 CS 中并未存储省的信息，这时用户需要将 CS 中所有商店编号为 01 或 02 的记录汇总以后，才能得到查询结果，这就需要与 $ATTS$ 表相连接。为避免这种连接操作延长查询响应时间，将表 1 与表 2 进行连接操作，得到基本表 $NTS(P,sid,D,sprovince,A)$，并基于 NTS 的所有维属性聚集而得到其商覆盖立方体 CQ_{NTS}。注意到在维表 $ATTS$ 中，sid 是主键，$sprovince$ 函数依赖于 sid，这种函数依赖也存在于 NTS 和 CQ_{NTS} 中。因此，NTS 中在 sid 上具有相同值的所有元组，在 $sprovince$ 上也就具有相同的值。如果在 NTS 中还包含有各维表的其他相应字段，则基本表中将存在更多的两个维属性的对，每对中的两个维属性之间存在函数依赖。

接下来将分析 BUC 算法和 DFS 算法，并探讨当基本表中的维属性之间存在上述函数依赖时，可以对 DFS 算法做出何种改进。

3　对 BUC 算法和 DFS 算法的分析

BUC 算法通过逐层将基本表划分成一个一个的子集并对之分别进行计算而得出相应的数据立方体。在计算时，先分别对参与聚集操作的每个维属性进行 Group by 操作，在进行 Group by 操作的过程中，依据每个维属性中的各个不同的维值将把基本表划分成一个一个的子集，BUC 算法分别在这些子集中进一步对更多维属性进行 Group by 操作并得出相应的结果。BUC 算法可以被用于生成数据立方体和商覆盖立方体。DFS 算法则在 BUC 算法的基础上进一步省略了部分 Group By 操作，而以更少的时间得到商覆盖数据立方体。

在 DFS 算法中需要遍历多个 $\theta_{X=v}(R)$ 这样的基本表子集，以判断其中的元组在非 X 维属性上的值是否相等，这个判断的操作比较耗时。研究中发现，利用基本表维属性之间的函数依赖，可以省去部分上述的判断操作，从而缩短生成商覆盖立方体的时间。

4　概念与定理

设有关系 $R(U,F)$，X、Y 是 U 的子集，$X\cap Y=\phi$，C_R 是 R 的数据立方体，CQ_R 是 R 的商覆盖立方体。

定义 1 若对于在 X 上的某个属性值 v，在 R 的子集 $\theta_{X=v}(R)$ 中的所有元组在 Y 上均具有相同的属性值，则称 X 关于 v 与 Y 之间存在类函数依赖(congeneric functional dependency，记作 $Xv\triangle Y$。

类函数依赖与函数依赖的不同之处在于：$Xv\triangle Y$ 只要求在属于当前关系的在 X 上具有某个属性值的子集中不存在使得 $t1[X]=t2[X]$且 $t1[Y]\neq t2[Y]$的两个元组，而 $X\rightarrow Y$ 则要求在 R 的任意可能的元组中对任何 X 的属性值均不可能存在使得 $t1[X]=t2[X]$且 $t1[Y]\neq t2[Y]$的两个元组。当 $Xv\triangle Y$ 时，在 C_R 中的单元格$(\cdots,v,\cdots)$与$(\cdots,v,\cdots,Y\cdots)$显然是由基本表的相同子集计算而得，从而对于任何聚集函数，在相同度量维上具有相同的函数值。

定理 1 若 $X\rightarrow Y$，则对于任意在 X 上的属性值 v，均有 $Xv\triangle Y$。

证明 因为 $X\rightarrow Y$，所以对任意两个元组 $t1\in R$，$t2\in R$，若 $t1[X]=t2[X]=v$，则必有 $t1[Y]=t2[Y]$。由定义 1 可知 $Xv\triangle Y$。

由定理 1 可知，若 $X\rightarrow Y$，在 DFS 算法中，可以省去依次对 X 上的每一个属性值判断 $Xv\triangle Y$ 是否成立的操作。

5　改进 DFS 算法

如上所述，当在基本表中各维属性之间存在函数依赖时，可以在 DFS 算法中省略一些步骤。DFS 算法先是按照一定的顺序，依据各个维属性对基本表进行水平分割，接下来对每一个分割所得的子集运行 $DFS(\ )$函数。$DFS(\ )$函数中的第二个步骤是找出 c 的上界 UB_c。设 c 中非 All 值的维属性的集合为 X，UB_c 中非 All 值的维属性的集合为 Z，算法 DFS 需要对 X 和 $Y\in Z-X$ 的属性值进行比较，判断 $X_v\triangle Y$ 是否成立。由定理 1，如果 $X\rightarrow Y$，则可以省略这个比较的操作。

据此，对 DFS 算法的做了三处修改，一是用一个线性表来保存基本表中维属性之间的函数依赖；二是对于每对 $X\rightarrow Y$，决定子 X 总是先于 Y 被用于对基本表进行水平分割；三是当在 DFS 函数中需要比较两个维属性的值时，算法先到上述线性表中查找，如果查找的结果表明这两个维属性之间存在函数依赖，则跳过比较维属性值的操作。将改进后的算法称为 DDFS 算法，描述如下：

Algorithm DDFS.[Computing Cover Quotient Cubes]

Input：Base table B，Line table F

//F stores the attributes pairs where there is a functional dependency.

Output：Cover Quotient Cube of B.

Steps：

1.Alternate the attributes pairs in B to ensure the depended attribute is before the corresponding attribute.

2.b=(*,...,*)；

3.call NEWDFS(b,B,0)；

4.output upper bounds；

Function NEWDFS(c,Bc,k,pID)

//c is a cell and Bc is the corresponding partition of the base table；

(下转 151 页)