文章编号:1005-8451(2004)06-0004-03

关于数据仓库中编码位图索引的研究

周丽萍,黄厚宽
(北京交通大学 计算机与信息技术学院,北京 100044)

摘 要:引入一种新的索引技术—编码位图索引,介绍它的基本思想及其维护方法,并列举了它在数据仓库中的几种典型应用。其中编码位图索引,能较好地提高数据仓库的查询效率。

关键词:数据仓库;简单位图索引;编码位图索引;检索函数
中图分类号:TP311.131 **文献标识码**:A

Study on encoded bitmap indexing used for data warehouse

ZHOU Li-ping, HUANG Hou-kuan
(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract: It was presented a new indexing technique, encoded bitmap indexing, introduced the essential idea and the maintenance of the encoded bitmap indexing, and enumerated its typical applications in data warehouse(DW) environment. The encoded bitmap indexing can enbance the efficiency of retrieving for data warehouse.

Key words: data warehouse; simple bitmap indexing; encoded bitmap indexing; retrieval function

数据仓库是一个面向主题的、稳定的、与时间相关的、集成的、能够更好的支持企业或组织的决策分析处理的环境,它将来自多个源的数据结合起来,为联机分析处理(OLAP)、数据挖掘等面向主题的应用提供集成、综合的数据。复杂的查询类型、海量的数据和很高的读取/更新比率是数据仓库中查询处理的主要特点,这些特点使得用于传统的数据库系统中的查询方法和优化技术并不适合于数据仓库环境。

为了提高数据仓库系统的查询速度,人们已经提出了很多方法,例如,预先计算出某些汇总数据,预定义访问路径,使用某些特殊的索引技术等等。在这里,我们只讨论索引技术。

本文介绍了一种编码位图索引,它是对简单位图索引的扩展,能够有效地改善数据仓库的查询性能。

1 位图索引技术

1.1 简单位图索引的优点和局限性

(1)简单位图索引的时间和空间开销要比B-树少

空间开销:假设一个表T有n个不同元组(称n为表T的基数),那么在属性A(其中A可以有m个不同取值,称m为属性A的基数)上建立简单位图索引所需字节数为$\frac{m \times n}{8}$,而在属性A上建B-树索引所需空间为$\frac{1.44 \times n}{M} \times P$Bit,其中p表示每页的大小,M表示B-树的阶。当m<11.52*p/M时,简单位图索引所需空间要比B-树索引少。也就是说,假设页大小p=4K,阶数M=512,那么当属性A的基数m<93时,建简单位图索引要比B-树节省空间。

时间开销:在属性A上建立B-树索引的时间复杂度为$O(n \times \log_{\frac{M}{2}}(m))+O(n \times \log_2(\frac{p}{4}))$,其中p为页大小,4为元组号所占字节数,式子的前半部分表示从根节点搜索到叶子节点的时间开销,后半部分表示把元组号插入相应的叶子节点的开销。而建立简单位图索引的时间复杂度为$O(n \times m)$,当n足够大而m足够小时,$O(n \times \log_{\frac{M}{2}}(m))+O(n \times \log_2(\frac{p}{4})) > O(n \times m)$,即建B-树索引的时间复杂度要大于简单位图索引。

(2)简单位图索引的协同性要比B-树好

当用户查询中有两个或更多的查询条件,例如A=a AND B=b,对于这种查询,若使用简单位图索引,只需要在相应的位图矢量上进行对应的逻辑操作(本例中为对属性A的值a对应的位图矢量和属性B的值b对应的位图矢量进行AND操作),而分别建立在属性A和属性B上的两个单独的B-树索引并不能如此有效地结合使用,在B-树索引上实现这种组合

收稿日期:2003-10-15
作者简介:周丽萍,在读硕士研究生;黄厚宽,教授。

查询要比简单位图索引复杂得多。

从上面的分析可以看出，当关键字的基数增大时，建立和维护简单位图索引所需的时间和空间也迅速增大。而且，简单位图索引的空间使用率只有 1/m，当 m 增大时，空间利用率降得很低。此外，对于范围查询，当要查询的范围很大时，要用到的位图矢量本身所占的空间也变得很大，在这种情况下，简单位图索引的性能要比 B - 树差。为了克服简单位图索引的这些局限，我们来介绍一种编码位图索引技术。

1.2　编码位图索引的基本思想

假设有一个包括 N 个元组的事实表 SALES，还有一个维表 PRODUCTS，包含大约 12 000 条产品信息，一般说来，若要在 PRODUCTS 维上建简单位图索引，需要 12 000 个 N 位的位图矢量。若使用编码位图索引，只需要 $\lceil \log_2 12000 \rceil = 14$ 个位图矢量和一个映射表。例如，假设表 T 中属性 A 的取值域为{a,b,c}，若使用编码位图索引，只需用 $\lceil \log_2 3 \rceil = 2$ 个位图矢量来在属性 A 上建立索引。如图 1 中所示，采用两个二进制位来对 A 的取值域{a,b,c}进行编码，a 被编码成 00，b 被编码成 01，c 被编码成 10。

简单位图索引　　编码位图索引

表 T		B_a	B_b	B_c	B_1	B_0	映射表	
…	A	1	0	0	0	0	a	00
a		0	1	0	0	1	b	01
b		0	0	1	1	0	c	10
c		0	0	1	0	1		
b		0	1	0	0	0		
a		1	0	0	0	0		
…		…	…	…	…	…		

图 1　编码位图索引的例子

然后，对那些 $A = a$ 的元组，分别在两个位图矢量 B_1 和 B_0 相应的位上置位 0，同样，对 $A = b$，置 B_1=0，B_0=1……依此类推。为了检索数据，给每个属性值定义一个检索布尔函数（简称为检索函数），在这个例子中 $k = \lceil \log_2 |A| \rceil = 2$。如果某个值 v_1 被编码为 b_1b_0（$b_i \in \{0,1\}$，i =0,1），那么值 v_0 的检索函数被定义为 x_1x_0，其中 $x_i = B_i$，如果 b_i=1，否则 $x_i = \overline{B_i}$（B_i 的否）。在上面的例子中，a,b,c 的检索函数分别为 $f_a = \overline{B_1}\ \overline{B_0}$，$f_b = \overline{B_1}\ B_0$，$f_c = B_1\ \overline{B_0}$，例如，要选出 $A = a$ OR $A = b$ 的数据，那只要在 f_a 和 f_b 上执行简单的 OR 操作，即 $f_a + f_b = \overline{B_1}\ \overline{B_0} + \overline{B_1}\ B_0$，显然，该等式可以化简为 $\overline{B_1}$，也就是说，要检索那些满足条件 $A = a$ OR $A = b$ 的元组，只要取位图矢量 B_1 中为 0 的位所标识的那些元组就可以了。

1.3　编码位图索引的维护

当数据仓库中的数据更新时，编码位图索引也要产生相应的修改以反映数据仓库的变化，我们分两种情形来讨论编码位图索引的维护。

1.3.1　取值域不扩充的更新

举上面的例子，表 T 中加入一个属性值 $A = b$ 的元组，只需要在位图矢量 B_1 和 B_0 的末尾加上 $B_1[j] = 0$ 和 $B_0[j] = 1$（其中 j 是新插入的元组在表 T 中的位置）。

1.3.2　取值域有扩充的更新

如果一个属性值 $A = d$ 的元组加入表 T 中，那么现在 A 的取值域扩充为{a,b,c,d}，那么先要判断下面的等式是否成立：

$$\lceil \log_2 A^{(m-1)} \rceil = \lceil \log_2 A^{(m)} \rceil \qquad (1)$$

其中 $|A^{(m-1)}|$ 代表插入前 A 的基数，$|A^{(m)}|$ 代表插入后 A 的基数。

如果等式（1）成立，就像例子中的这种情况，那么只要在映射表中加入映射：

$M^A(d) = 11$，并置 $B_i[j] = M^A(d)[i]$　i=0,1,…,k-1，j 表示新插入元组在表 T 中的位置），如图 2(a) 所示，d 的检索函数 $f_d = B_1\ B_0$。

表 T		B_1	B_0	映射表	
…	A	0	0	a	00
a		0	1	b	01
b		1	0	c	10
c		0	1	d	11
b		0	0		
a		…	…		
…		1	1		
d					

(a)

表 T		B_2	B_1	B_0	映射表	
…	A	0	0	0	a	000
a		0	0	1	b	001
b		0	1	0	c	010
c		0	0	1	d	011
b		0	0	0	e	100
a		…	…	…		
…		0	1	1		
d		1	0	0		
e						

(b)

图 2　取值域有扩充的更新

如果还有另一个元组（其属性值 $A = e$）加入表 T，现在 A 的取值域变为{a,b,c,d,e}，$\lceil \log_2 |A^{(m-1)}| \rceil < \lceil \log_2 |A^{(m)}| \rceil$，其更新后的位图矢量和映射表如图 3（b）所示，

在这种情形下，必须执行如下几步操作：

（1）扩充映射 M^A：$\{A \mid A \in \{a,b,c,d\}\} \rightarrow \{<b_1\ b_0> \mid b_i \in \{0,1\},\ i=0,1\}$ 到 M^A：$\{A \mid A \in \{a,b,c,d,e\}\} \rightarrow \{<b_2\ b_1\ b_0> \mid b_i \in \{0,1\}, i=0,1,2\}$。

(2)增加一个位图矢量B_2到B_A，并置矢量B_2为0。

置B_i[j]= M^A(e)[i]，其中i=0,1,2,j表示新插入元组在表T中的位置。

值e增加布尔函数$f_e=B_2\ \overline{B_1}\ \overline{B_0}$并修改值a，b，c的布尔函数，即在它们原来的布尔函数前加上$\overline{B_2}$，修改后为$f_a=\overline{B_2}\ \overline{B_1}\ \overline{B_0}$，$f_b=\overline{B_2}\ \overline{B_1}\ B_0$，$f_c=\overline{B_2}\ B_1\ \overline{B_0}$，$f_d=\overline{B_2}\ B_1\ B_0$。

1.4 编码位图索引的应用

这一节中，列举了编码位图索引在数据仓库中的几种可能的应用及变更的例子。

1.4.1 层次编码

数据仓库中的数据多以星型模式组织，通常是包括一个(几个)事实表和多个维表，维表中可能存在层次关系。例如，销售数据的维表销售点可能被分为3个层次单元：部门、公司和联盟。假设有12个部门{1,2,3,…,12}，5个公司{a,b,c,d,e}和3个联盟{X,Y,Z}.一个公司分为好几个部门，几个部门组成一个联盟。例如，部门{1,2,3,4}属于公司a，部门{5,6}属于公司b，…，公司{a,b,c}组成联盟X等。

OLAP的许多常用的数据分析和操作，例如，选出联盟Z中所有公司的销售数据等，都是基于维表及维表单元上的选择或查询的。因此，在数据仓库中，同一维表层次中的数据很有可能被同时访问。

层次编码的思想就是建立关于各层次单元上的选择的编码位图索引。在上面的例子中，"公司"和"联盟"这两层的取值域分别是{a,b,c,d,e}和{X,Y,Z}，那么在"公司"或"联盟"上的选择集谓词为P=$\{s_{company=i}\mid i\in\{a,b,c,d,e\}\}\cup\{s_{alliance=j}\mid j\in\{X,Y,Z\}\}$。一种关于P的最优编码方式，对沿着维度单元"公司"或"联盟"的选择进行了优化，例如，对于"联盟=X"，只需要访问一个位图矢量。

这一思想还可以进一步扩展到使用编码位图索引建立分组集索引。分组集索引对应于用户查询中的Group-By从句。由于篇幅有限，就不详细讨论这种情况了。

1.4.2 编码位图索引用做范围索引

编码位图索引的一个可能的变种是将它用于基于范围的索引，举个例子来阐述这一方法的主要思想。

给定一个属性A的取值域为$6\le A<20$，$A\in N$。假设用户预先定义了如下的范围查询"$6\le A<11$"，"$8\le A<12$"，"$11\le A<13$"，"$16\le A<20$"。根据预定义的选择，属性A的取值域应该先划分为6个不相交的部分：{[6,8),[8,11),[11,12),[12,13),[13,16),[16,20)}.然后对这6个区域如图3(a)编码。那么，对于范围查询如"$8\le A<12$"，检索函数为$\overline{B_2}\ \overline{B_1}\ B_0+B_2\ \overline{B_1}\ B_0$，能化简为$\overline{B_1}\ B_0$，所有预定义范围查询的检索函数化简后如图3(b)所示。

[6,8)	000
[8,11)	001
[11,12)	101
[12,13)	100
[13,16)	010
[16,20)	110

(a)范围编码

$6\le A<11$：$\overline{B_2}\ \overline{B_1}$

$8\le A<12$：$\overline{B_1}\ B_0$

$11\le A<13$：$B_2\ \overline{B_1}$

$16\le A<20$：$B_2\ B_1$

(b)检索函数

图3　编码位图索引用做范围索引的例子

1.4.3 混合索引

当索引属性的基数增大时，映射表也变得很大，因此，必须有一种有效的方法来创建和存取映射表。如果数据仓库中的数据在最初装入以后，索引属性的基数不再变化，就用静态哈希表来建立映射表，否则，可以使用可扩展哈希表，可扩展哈希表中的桶能根据实际需要增大或变小，虽然它的平均效率不及静态哈希表，但却优于B-树。

编码位图索引还有一些其它的应用，这里就不一一列举了。

2 结束语

本文首先对简单位图索引做了简要的回顾，阐述了其优缺点，在此基础上引入了一种编码位图索引技术，它既保留了简单位图索引的简单性及协同性等优点，而且克服了其不适用于大基数属性的局限，能较好地提高数据仓库的查询效率。

参考文献：

[1] 王能斌．数据库系统原理[M]．北京：电子工业出版社，2000.

[2] 殷人昆，陶永雷．数据结构[M]．北京：清华大学出版社，2001.

[3] 王　珊．数据仓库与联机分析处理[M]．北京：科学出版社，1998.6.

[4] Chee-Yong Chan and Yannis E. Ioannidis. Bitmap Index Design and Evaluation[C]. In SIGMOD'98，1998.