ISSN 1000-1239/CN 11-1777/TP
49(10): 2106-2117, 2012

基于视图树的实视图动态选择

林子雨[1] 邹 权[1] 林 琛[1] 赖永炫[2] 郑 炜[1]
[1](厦门大学计算机科学系 福建厦门 361005)
[2](厦门大学软件学院 福建厦门 361005)
(ziyulin@xmu.edu.cn)

View-Tree-Based Dynamic View Selection

Lin Ziyu[1], Zou Quan[1], Lin Chen[1], Lai Yongxuan[2], and Zheng Wei[1]
[1](*Department of Computer Science, Xiamen University, Xiamen, Fujian* 361005)
[2](*School of Software, Xiamen University, Xiamen, Fujian* 361005)

Abstract User-oriented materialized views are able to greatly improve OLAP query performance for users. However, the available methods for cache management are not able to deal with the issue of dynamic view selection, since they do not take into account the data access pattern of OLAP queries of specific users. In this paper, the concepts of view path and view tree are proposed to organize the views. Also, a method called reverse path growing is proposed to quickly compute view path for a newly-arrived query, so as to greatly reduce query response time. Furthermore, an effective view replacement method based on reserved view path is designed to better deal with the issue of dynamic adjusting of view tree. Extensive experiments show that the proposed method can achieve better performance than those previous ones.

Key words materialized view selection; data warehouse; OLAP; multi-dimensional data; cache

摘 要 为用户缓存实视图可以有效提高其OLAP查询的性能.但是,已有的缓存管理策略由于没有考虑用户在进行OLAP分析时的数据访问特性,在处理实视图动态选择问题时无法获得好的性能.提出了视图路径和视图树的概念,并以视图树作为客户端缓存中的实视图组织方式.提出了"逆路径增长法"来快速计算新到达查询的视图路径,提高了查询的响应速度.对于视图树的动态调整问题,以"保留路径"为参照,设计了合理有效的视图替换策略.实验证明,该方法能够比已有的动态选择方法取得更好的性能.

关键词 实视图选择;数据仓库;联机分析处理;多维数据;缓存

中图法分类号 TP311

在实时主动数据仓库环境中,为了支持实时联机分析处理(on-line analytical processing,OLAP)查询,系统就必须保证查询的快速响应.但是,针对历史数据的OLAP查询通常是需要一些汇总数据,有时为了得到一个简单的汇总数值,甚至需要对整个数据仓库的基本表进行扫描,这需要花费大量的处理时间,很难保证查询的实时性.一种有效改善查询响应时间的方法就是采用实视图(materialized view)[1]技术.但是,出于空间的限制,数据仓库的多维数据实视图不可能包含所有级别的数据,只能根

收稿日期:2011-09-14;**修回日期**:2012-04-23
基金项目:厦门大学基础创新科研基金项目(中央高校基本科研业务费专项资金项目)(2011121049);国家自然科学基金项目(61001013,61102136);福建省自然科学基金项目(2011J05156,2011J05158)

据一定的策略有选择性地实化一部分多维数据,这就是“多维数据实视图选择[1]”问题(通常简称为“视图选择问题”).

针对视图选择问题,研究人员已经提出许多方法,这些方法主要分为两大类:静态视图选择方法[2-8]和动态视图选择方法[9-15].如果把数据仓库应用环境分成3个层次结构:基本数据层(数据仓库)、中间层(数据缓存)和应用层(客户端),那么,静态视图选择发生在基本数据层,已有的动态视图选择发生在中间层,比如临时数据缓存,面向的对象是多个用户的大量当前查询;而目前在中间层还没有针对具体某个用户的视图选择.但是,随着实时主动数据仓库的不断发展和成熟,为了给执行高级战术决策的用户提供更好的实时OLAP查询性能,在中间层为特定用户开辟缓存并存储一些与具体应用相关的实视图显得日益重要,它可以作为另外两种类型的实视图的有益补充,进一步提高数据仓库的实时查询响应性能.这里,我们把前两种视图称为“公共视图”,把用户缓存中的视图称为“用户视图”.

对于用户缓存而言,它通常只包含单个用户的查询数据.而对于一个典型的用户而言,他的查询通常具有访问的局部性[9,12],比如,一个用户在某个会话期间可能多次访问某个数据,而对于用多维数据格表示的用户查询而言,格中具有父子关系或者兄弟关系的节点很可能会被一起访问.另外,用户的查询通常还具有可导航性[16].在OLAP系统中,如果用户在前一个分析结果的基础上继续得到下一个分析结果,那么OLAP查询负载就具有可导航性.例如,大型零售商沃尔玛的某个销售分析师正在对华东市场销售数据进行分析,期望发现一些感兴趣的趋势.这位分析师可能先在$(Product,City)$这个层次查看上海市的销售数据,这时,他可能发现沃尔玛在该市的销售业绩没有达到预期,于是,他会继续下钻到粒度更细的层次$(Product,Store)$,来查看上海市某个具体的沃尔玛连锁店A的销售数据,从而判断是哪家连锁店的销售出了问题.如果没有发现异常,他会又回到$(Product,City)$层次,继续查看其他省份的销售数据.文献[16]对某个具体应用的实验分析结果显示,典型的OLAP会话都具有一定的长度,只有1%的会话只包含单个查询,并且,有一部分会话包含的查询数量达到100个以上,更加容易发现查询的访问局部性和可导航性.因此,根据用户查询的访问局部性和可导航性,就可以在用户缓存中设计相应的视图选择算法,对实视图集合进行动态调整.

用户缓存的视图选择:假设当前的用户缓存实视图集合为M,新到达的查询为q,用户的访问具有局部性和可导航性,实视图的动态选择问题就是采用一定的视图替换策略,用与查询q相关的视图来替换M中一部分与q最不相关的视图.

以前的动态方法[9-12]采用的缓存替换策略大都是基于统计数据的,比如查询结果尺寸、查询访问频率以及查询更新频率等等.这些方法适合于服务器层的缓存,目的在于提高对多个用户的查询响应速度.由于服务器层同时考虑多个用户的大量查询,这些查询样本具有较强的可统计性,比较容易从中获得有价值的统计信息,所以这些动态方法可以取得好的性能.但是,在用户缓存中,缓存的对象是单个用户在分析期间的一系列查询,这时,以前的动态方法就不能获得好的性能,因为少量的查询样本的可统计性比较弱,比如在一次用户会话(OLAP分析开始到结束的过程)期间,对用户查询使用某个视图的频率进行统计是没有多大意义的[11].而已有的缓存管理方法[17]由于没有专门考虑用户在进行OLAP分析时的数据访问特性,因此在处理实视图动态选择问题时不能获得好的性能.

本文主要研究基于用户缓存实视图的实时OLAP查询性能优化,通过在用户缓存存储用户查询的中间结果,加快OLAP查询响应时间.本文具体贡献包括以下几个方面:

1) 设计了有效的视图组织方式:视图树作为视图的组织方式,从而支持对缓存中的视图的有效管理,并为设计视图选择算法奠定基础;

2) 提出了高效的视图选择算法:在给定的视图组织方式——视图树的基础上,设计了高性能的视图选择算法,可以根据用户查询过程的变化,快速高效地对实视图集合进行动态调整;

3) 设计了用户视图与公共视图的有效互补机制:使得用户缓存的视图成为公共视图的有效补充,而不是简单地冗余存储一个视图的多个副本;用户缓存视图的内容应该随着公共视图的变化而动态变化,公共视图的内容也受到用户视图内容的影响;

4) 实验结果证明了算法的性能:本文进行了大量的实验,证明本文提出的方法能够比已有动态选择方法取得更好的性能.

1 视图树

本节首先介绍与多维数据相关的基本概念,包括多维数据格、多维数据上的查询、完整聚集视图和

部分聚集视图等等;然后给出视图路径和视图树的定义.

1.1 基本概念

定义 1. 多维数据格[1]. 多维数据格$\langle M,\leqslant\rangle$是由多个数据节点 T 构成的,其中 $T=(a_1,a_2,\cdots,a_n)$,M 是多个节点 T 的集合,每个 a_i 代表第 i 维上的一个级别,并且有:

1) 对于两个节点 $T_1=(a_1,a_2,\cdots,a_n)$和 $T_2=(b_1,b_2,\cdots,b_n)$,$T_1\leqslant T_2$(即 T_1 依赖于 T_2),当且仅当对于所有的 i 都有 $a_i\leqslant b_i$,即 T_2 在各维上的级别均低于或等于 T_1 在相应维上的级别,并且利用 T_2 可以计算得到 T_1;

2) 对于任意两个节点 T_1 和 T_2,如果 $T_1\leqslant T_2$,则 T_1 是 T_2 的子节点,用 $T_1=C(T_2)$表示;T_2 是 T_1 的父节点,用 $T_2=P(T_1)$表示;

3) 格中基本元表示为 $\boldsymbol{DB}=(c_1,c_2,\cdots,c_n)$,其中,$c_i$ 表示第 i 维上最低的一个级别;通常假定 $\boldsymbol{DB}$ 的数据是已知的,可利用它计算格中任意节点的数据.

例 1. 多维数据模式包含两个维,即 *Product* 和 *Location*,每个维又具有各自的层次结构. *Product* 维的层次结构为 $ProductId\rightarrow Category\rightarrow All$; *Location* 维的层次结构为 $StoreId\rightarrow Area\rightarrow All$. 该多维数据格如图 1 所示,其中 p,c,a,s 分别表示 *ProductId*,*Category*,*StoreId*,*Area*,根据多维数据格的定义,基本元 $\boldsymbol{DB}=(p,s)$.

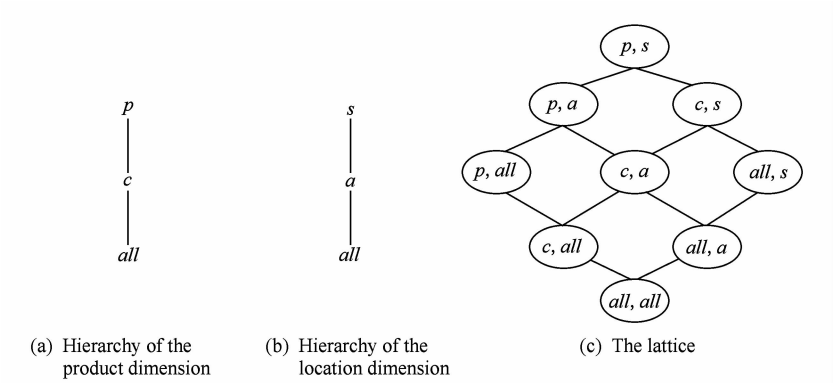


Fig. 1 An example lattice.

图 1 一个多维数据格的例子

定义 2. 多维数据上的查询[10]. 多维数据集合 *MD* 上的查询 q 是对 *MD* 的数据格中某一数据节点的切片或切块,可以将 q 表示成由 d 个二元组组成的 d 元组:$\{(l_1,R_1),(l_2,R_2),\cdots,(l_d,R_d)\}$,其中 d 为 M 的维数,l_i 表示维 d_i 的某一级别,R_i 表示在 l_i 上的选择范围;若 R_i 是 $dom(l_i)$,即取值范围没有限制,可以将(l_i,R_i)记为 l_i;当 R_i 被表示为$\langle r_{i1},r_{i2}\rangle$时,表示在$\langle r_{i1},r_{i2}\rangle$范围上的一个切块,而当 R_i 被表示为$\{r_{i1},r_{i2},\cdots,r_{in}\}$时,则表示在$\langle r_{i1},r_{in}\rangle$范围上的 n 个切片. 若 $l_i=all$,则维 i 的二元组可以不出现在查询中.

定义 3. 完整聚集视图. 在定义 2 中,当对于所有的 i 或者 $l_i=all$ 成立,或者 $R_i=\langle r_{i1},r_{i2}\rangle=dom(l_i)$ 成立,则此时的查询被称为完整聚集视图.

定义 4. 部分聚集视图. 在定义 2 中,当至少存在一个 i 使得 $l_i\neq all$ 且 $R_i=\langle r_{i1},r_{i2}\rangle\neq dom(l_i)$时,则此时的查询被称为部分聚集视图.

例 2. 我们以一个销售应用的多维数据为例,对查询的表示方法以及完整聚集视图和部分聚集视图的概念进行说明. 该多维数据模式包括 3 个维:时间、产品和商店,各维的层次结构如图 2 所示. 我们定义如下查询:

$q_1=\{$(产品 *Id*),(区域,{西北}),(年,{2004, 2006})$\}$,表示“各个产品在西北区域内分别在 2004 年和 2006 年的销售量”;

$q_2=\{$(产品 *Id*),(区域),(年)$\}$,表示“各个产品在各个区域内分别在 2004 年、2005 年和 2006 年的销售量”.

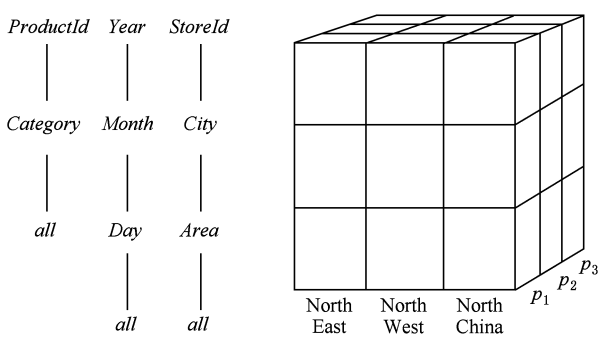


Fig. 2 An example multidimensional data set.

图 2 一个多维数据集例子

很显然,根据定义 3 和定义 4,q_1 属于部分聚集视图,q_2 属于完整聚集视图.容易看出,例 1 所示的多维数据格中的数据节点属于完整聚集视图.

1.2 视图路径

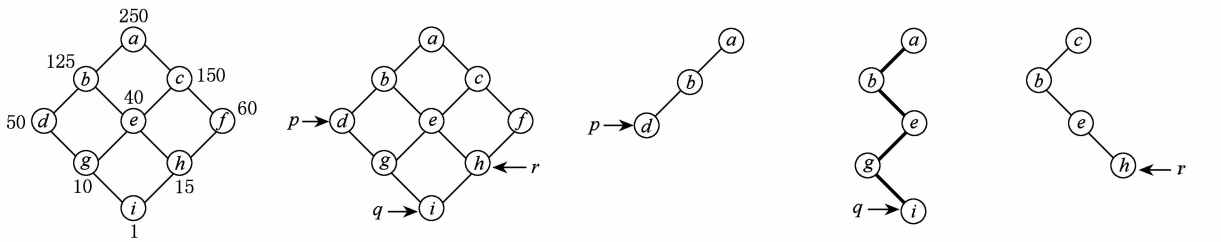
基于上面的定义,我们就可以为基于某个基本元 ***DB*** 的所有完整聚集视图建立一个多维数据格 M.如果查询 q 属于完整聚集视图,那么,q 就可以用 M 中的某个节点直接得到回答;如果查询 q 属于部分聚集视图,那么,q 可以用 M 中的某个节点 N 的切片或切块来回答,这里,我们称节点 N 为查询 q 的宿主节点.

采用多维数据格来组织用户的查询所涉及的视图,用户就可以沿着格中的边依次访问各个节点.当用户访问某个节点 u 时,如果在格中不存在从节点 u 到另一个节点 v 的边,那么用户就不会访问节点 v.在这种情况下,下钻(drill-down)操作实际上就是多维数据格中从较高级别到较低级别的一条路径,而上卷(roll-up)操作则相反.格结构可以告诉我们以什么样的顺序实化视图,以及如何使用已有的实化视图来实化其他视图,这样可以避免访问细节数据,从而降低系统开销.

定义 5. 视图路径.对于给定的查询 q,它的视图路径是多维数据格中,以已经被实化的某个节点为起点、以查询 q 的宿主节点 v 为终点的一条路径 L_q,路径上的各个节点满足查询间的依赖关系和代价最小原则.另外,我们把 L_q 的起点和终点互换后得到的路径 N_q 称为查询 q 的视图逆路径.

查询 q 的视图路径包含了可以满足查询 q 的代价最小的实化视图的序列.根据该序列我们可以知道,在当前的实视图集合的基础上,为了满足查询 q 还需要哪些视图以及这些视图被实化的顺序.

例 3. 我们为例 1 中的数据格中每个节点分配一个唯一的标识符,并根据视图尺寸估算方法得到每个节点的空间代价,结果如图 3(a)所示.现在假设有 3 个查询 p,q,r,它们的宿主节点分别是 d,i,h,并且假设每个查询执行时实视图集合都为空,则它们对应的视图路径分别是 $a\rightarrow b\rightarrow d$,$a\rightarrow b\rightarrow e\rightarrow g\rightarrow i$ 和 $a\rightarrow b\rightarrow e\rightarrow h$(如图 3(c)(d)(e)所示):



(a) The lattice with costs (b) Queries on the lattice (c) View path for query p (d) View path for query q (e) View path for query r

Fig. 3 An example of materialized view path.

图 3 一个视图路径的例子

目前,文献[18]是视图尺寸估算方面比较有代表性的研究,该文的作者 Shukla 等人在视图研究方面积累了丰富的经验,曾经提出了 PBS 视图选择算法[19].Shukla 等人在文献[18]中提出的针对多维数据立方体中视图的尺寸估算方法包括基于采样的方法、基于数学估算的方法和基于概率统计的方法,后来的一些视图尺寸估算方面的研究也都采用了类似的方法.从研究人员的大量实验结果来看,基于概率统计的方法其准确度要比另外两种方法高.

1.3 视图树

多个视图路径的并集可以得到一棵视图树.假设与 n 个查询 $q_1,q_2,\cdots,q_n$ 对应的视图路径分别是 $L_1,L_2,\cdots,L_n$,则由这 n 个查询生成的视图树 T 是这 n 个视图路径的并集,即 $T=L_1\cup L_2\cup\cdots\cup L_n$.视图树的形式化定义如下:

定义 6. 视图树.一棵视图树 $T=(V,E)$,其中 V 是顶点的集合,E 是边的集合,并且满足如下条件:

1) 对于任意节点 $v\in V$,$R(v)$表示节点 v 对应的关系,并且 $R(v)$是实视图集合中的元素;

2) 对于任意节点 $v\in V$,$R(v)$是基于某个基本元 ***DB*** 的多维数据格 M 中的节点;

3) 对于任意两个节点 u 和 v,如果 $R(u)\leqslant R(v)$,则 u 是 v 的子节点,用 $u=C(v)$表示;v 是 u 的父节点,用 $v=P(u)$表示;

4) 对于任意节点 $v\in V$,$A(v)$表示 $R(v)$包含的元组数目;

5) 对于任意节点 $v\in V$,$P(v)$ 表示 v 的父节点,$P^*(v)=P(v)\cup\{\bigcup_{v'\in P(v)}P^*(v')\}$表示 v 的祖先节点;

6) 对于任意节点 $v \in V$,$C(v)$ 表示 v 的子节点,$C^*(v) = C(v) \bigcup \{ \bigcup_{v' \in P(v)} C^*(v') \}$ 表示 v 的后代节点.

2 基于视图树的实视图动态选择

在基于视图树的实视图动态选择方法中,除了维护一个实视图集合以外,还要维护该实视图集合对应的视图树.视图树的根节点对应的视图是存储在数据仓库中的基本元 ***DB***,除此以外,视图树中的每个节点都与实视图集合中的一个视图相对应,对视图树中节点的每个操作都会反映到实视图集合中.比如,在视图树中增加一个节点,就相应地为实视图集合增加这个节点所对应的视图;删除操作亦是如此.因此,实视图的动态选择过程和视图树的动态调整过程是等价的,所以,我们这里只讨论如何进行视图树的动态调整.

下面的内容我们先介绍视图树的动态调整过程描述和算法,然后详细讨论视图树的动态调整过程所涉及的细节问题,包括视图路径的计算、视图树的节点增加和删除等等.

2.1 视图树的动态调整

用户的分析查询总是从一个数据粒度较粗的汇总报表开始,然后逐渐向下钻取分析更细粒度的数据,这中间还伴随着上卷和下钻操作的交替进行.在用户执行第 1 个查询 q_0 来获得汇总报表时,系统为查询 q_0 返回结果的同时,也相应地生成了第 1 条视图路径,该路径反映了从基本元 ***DB*** 生成查询 q_0 的代价最小(本文只考虑查询代价,而不考虑视图更新代价)的视图实化过程,也是构成一棵视图树的初始元素.

随着新的查询的不断到达,视图树会不断地生长,最终,树中节点甚至可以覆盖多维数据格中的所有节点.在实视图空间足够大的情况下,视图树上的所有视图都可以被实化.当存在实视图空间限制时,就只能实化树中的一部分视图,这意味着需要根据新到达的查询不断地对树进行动态调整.

算法 1 显示了视图树的动态调整过程.当新的用户查询 q 到达时,我们需要计算得到查询 q 对应的视图路径(具体计算方法在 2.2 节讨论),如果视图路径上的任意节点 v 不在视图树 T 中,那么就需要把 v 添加到视图树 T 中;这时,如果实视图空间不能容纳所有这些新增加的视图,那么,就需要在视图树中,寻找一个与当前查询 q 相关性最小的节点 u,把 u 从 T 中删除(相应地,节点 u 对应的视图也会被从实视图集合中删除),从而释放占用的空间,并把 v 加入 T(相应地,v 对应的视图会被加入到实视图集合中).这个过程一直重复,直到查询 q 对应的视图路径上所有节点都被加入到 T 中.该过程所涉及的视图树中节点的增加与删除等细节问题将在下面进行讨论.

算法 1. 视图树的动态调整.

输入:当前的视图树 W、实视图空间的尺寸 S、新到达的查询 q;

输出:调整后的视图树 W.

```
begin
  L_q ← CalculateMaterializationPath(W,q);
  /* 计算查询 q 的视图路径 */
  S_W ← SizeSum(W); /* 计算 W 中的所有视图节点的尺寸之和 */
  for each u∈L_q do
    if u∉W then
      while S_W + |u| > S do
          DeleteNode(W,v); /* 从 W 中删除某个节点 v */
          S_W ← S_W − |v|;
      AddNode(W,u); /* 把 u 加入实视图树 W 中 */
      S_W ← S_W + |u|;
  return W;
end
```

2.2 视图路径计算方法

视图树中的第 1 条视图路径的计算只与第 1 个查询 q_0 相关,因为,在此之前的实视图树不存在(即实视图集合为空).假设查询 q_0 的宿主节点是 v_0,那么,第 1 条视图路径就是多维数据格中从基本元 ***DB*** 到 v_0 的一条代价最小的路径.多维数据格是一种图结构,因此,我们可以使用著名的 Floyd 算法[19]来计算得到从基本元 ***DB*** 到 v_0 的视图路径;该方法无论图中有多少条边都能把处理时间限制在 $O(n^3)$,其中 n 表示格中节点的数目.

在第 1 个查询以后到达的查询 q 的视图路径的计算,则与当前的视图树 W 和新到达的查询 q 相关.一种计算视图路径的朴素方法是,先分别计算树中每个节点 v_i 到查询 q 的宿主节点的代价最小的路径 L_i,然后选择代价最小的 L_i 作为查询 q 的视图路径.

但是,朴素方法的计算代价比较大,这里,我们采用计算代价更小的"逆路径增长法"来求解查询 q 对应的视图路径(如算法 2 所示).在逆路径增长法

中,以查询 q 的宿主节点 v 作为逆路径 N_q 的起点,然后,根据多维数据格中的节点依赖关系,逐渐把 N_q 向数据格的上方增长.我们把逆路径存放在一个队列 S 中,然后从队列 S 中取出一条逆路径 N_i,并以 N_i 为基础,继续延长逆路径,并且把新增加的逆路径放入队列 S.如果 N_i 的当前终点在格中有 m 个父节点,N_i 的当前路径长度为 n,那么把 N_i 延长到这些父节点,就会得到 m 条长度为 $n+1$ 的逆路径$\{N_j \mid j=1,2,\cdots,m\}$.一旦逆路径到达视图树 W 中的某个节点,就结束增长,如果这时得到的逆路径 N 优于当前的最优逆路径 N_{best},则把 N 作为新的 N_{best}.实化视图逆路径的终点肯定在视图树中,因为视图树 W 中永远包含数据格的基本元 ***DB***,而基本元 ***DB*** 是实化视图逆路径可能到达的最远的节点.

队列 S 中的逆路径数量并不是无限增大的,我们采用启发算法可以很快丢弃大量不符合条件的逆路径.启发算法的依据是,如果采用逆路径 N 来回答查询 q 的代价 $Cost(N,q)$已经超过当前最优逆路径 N_{best},那么就丢弃 N,因为继续增长 N 只会使 $Cost(N,q)$更大,不可能获得比 N_{best}更优的逆路径.最后,当队列 S 为空时,就可以得到查询 q 的实化视图逆路径 N_{best},由 N_{best}就可以反转得到查询 q 的视图路径.

算法 2. 视图路径的计算.

输入:当前的视图树 W、新到达的查询 q;

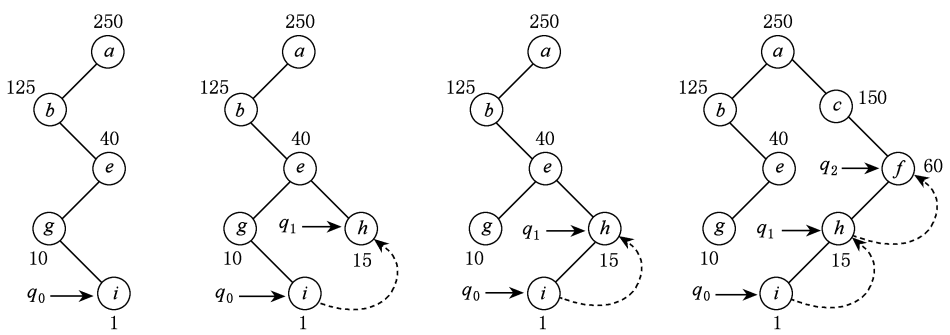
输出:视图路径 L_q.

```
begin
  N_q←{v}; /* v 是查询 q 的宿主节点,作为逆路径 N_q 的起点 */
  S←S∪{N_q}; /* 队列 S 用来存放所有以 v 为起点的逆路径 */
  N_best←{DB}; /* N_best 表示用来回答查询 q 的代价最小逆路径,初试值为基本元 DB */
  C_best←Size(DB); /* C_best 表示生成查询 q 的最小代价,初试值为基本元 DB 的元组数目 */
  while S≠∅ do
    N←S.get(); /* 从队列 S 中取出一个元素 */
    if Cost(N,q)>C_best then /* Cost(N,q)表示通过逆路径 N 得到查询 q 的代价 */
      Release(N); /* N 不可能继续生长为代价最小逆路径,因此被删除 */
    else /* 继续对逆路径进行延伸 */
      if N 没有到达 W 中的任何节点 then
        t←EndNode(N); /* t 是逆路径 N 的当前终点 */
        for each u∈P(t) do /* P(t)表示 t 的父节点 */
          N_1←N∪{u}; /* 把逆路径 N 延长到节点 u */
          S←S.put(N_1); /* 把逆路径 N_1 放入队列 S */
      else /* N 到达 W 中的某个节点 */
          N_best←N; C_best←Cost(N,q);
          Release(N);
  L_q←Rotate(N_best); /* 由逆路径得到路径 */
  return L;
end
```

2.3 视图树的节点增加

在计算得到视图路径 L_q 以后,需要把 L_q 上不在视图树 W 上的节点增加到 W 中.这里,我们以一个实例介绍视图树的节点增加过程,并不给出具体算法.

例 4. 图 4 显示了一棵视图树的生长过程.带有



(a) View tree1 (b) View tree2 (c) View tree3 (d) View tree4

Fig. 4 The growth of materialized view tree.

图 4 视图树的生长过程

节点空间代价的多维数据格 M 如图 3(a)所示.现在假设第 1 个查询 q_0 的宿主节点是 M 中的节点 i,根据 Floyd 算法可以得到第 1 条视图路径 $a\rightarrow b\rightarrow e\rightarrow g\rightarrow i$,并由 a,b,c,g,i 形成最初的视图树 W(如图 4(a)所示).然后,用户进行下钻操作(图 4 中带箭头的虚线表示用户的下钻路径),执行查询 q_1,q_1 的宿主节点是 h,根据 2.2 小节介绍的实化路径计算方法——逆路径增长法,我们可以计算得到查询 q_1 对应的视图路径为 $e\rightarrow h$,由于 h 不在 W 中,因此,把 h 增加到 W 中,得到的结果如图 4(b)所示.这时,我们发现,在 W 中,g 和 i 之间存在一条边,而 h 和 i 之间不存在边,但是,h 却在用户的下钻路径上,而 g 不在下钻路径上,因此,我们把 g 和 i 之间的边删除,增加一条以 h 和 i 为顶点的边,这样做的原因是为了维护当前的保留路径(保留路径的定义和重要作用在 2.4 小节论述).

2.4 视图树的节点删除

定义 7. 保留路径对于给定的一棵视图树 W,树中的保留路径 L_B 是满足如下条件的一条路径:1)根节点(基本元 ***DB***)为起点;2)以用户的第 1 个查询 q_0 的宿主节点 v_0 为终点;3)中途经过用户的当前查询 q_i 的宿主节点 v_i.

例 5. 在图 4 的视图树中,基本元 ***DB*** 是节点 a,查询 q_0,q_1,q_2 对应的宿主节点分别是 i,h,f,则图 4(a)中的保留路径是 $a\rightarrow b\rightarrow e\rightarrow g\rightarrow i$,图 4(c)中的保留路径是 $a\rightarrow b\rightarrow e\rightarrow h\rightarrow i$,图 4(d)中的保留路径是 $a\rightarrow c\rightarrow f\rightarrow h\rightarrow i$.

保留路径是视图树执行节点删除操作时的重要参考信息,该路径上除了包含从用户当前查询 q_i 到基本元 ***DB*** 的视图路径以外,还包含在从用户当前查询 q_i 到第 1 个查询 q_0 的上卷路径.按照用户的分析习惯,这条上卷路径也是可能性比较大的用户分析查询执行路径.

定义 8. 保留节点集假设当前查询 q 的宿主节点是 v,$P(v)$ 表示多维数据格中 v 的父节点,$C(v)$ 表示多维数据格中 v 的子节点,L_B 表示视图树中的当前保留路径,则当前的保留节点集 I 定义如下:

$$I=\{p\mid p\in P(v)\wedge p\in W\}\cup$$
$$\{c\mid c\in C(v)\wedge c\in W\}\cup\{u\mid u\in LB\}.$$

保留节点集中包含了用户的下一个操作(可能是下钻或上卷)可能访问的视图树中的节点,以及当前保留路径上的节点,其中,$P(v)$ 是用户下钻操作可能访问的节点集合,$C(v)$ 是用户上卷操作可能访问的节点集合.

假设实视图空间可以容纳当前保留路径 L_B 上的节点,I 是当前的保留节点集,则视图树 W 中的节点删除步骤如下.

步骤 1:找到树 W 中不属于 I 的叶子节点集合 Y_1,逐个删除 Y_1 中的节点,直到获得足够空间;如果删除了所有 Y_1 中的节点仍然无法获得足够空间,则继续执行步骤 2;

步骤 2:找到树 W 中不属于 L_b 的叶子节点集合 Y_2,逐个删除 Y_1 中的节点,直到获得足够空间;因为已经假设实视图空间可以容纳当前保留路径 L_B 上的节点,所以这个节点删除过程一定可以停止.

我们把上述的视图树的节点删除策略(即视图替换策略)称为"保留路径法".

3 用户视图与公共视图的有效互补机制

下面介绍用户视图和公共视图的有效互补机制,这里我们采用了"虚视图"机制.如图 5 所示,我们在全局视图窗口(全局缓存)内设置了公共视图窗口(公共缓存)和用户视图窗口(用户缓存),公共视图窗口面向所有的查询,用户视图窗口则专门为某个用户提供服务,图中用空心圆表示虚视图,它会引用真实存在的实视图(实心圆表示).对于某个特定的视图窗口 W_1,当某个视图 V 被视图选择算法选中时,为了避免重复存储同一个视图,它需要检查其他视图窗口内是否已经存在该视图 V,如果不存在,则在 W 内存储 V 的实视图,如果在另一个视图窗口 W_2 存在视图 V 对应的实视图 V_{W2},那么,在视图窗口 W_1 内,就不需要再次存储视图 V 对应的实视图,只需要存储一个指向 V_{W2} 的引用.这种视图互补机制促进了多个用户之间的视图共享,提高了有限的缓存资源的利用率.

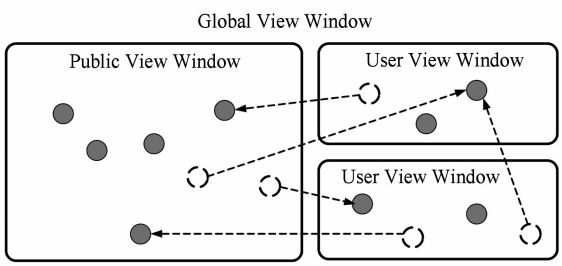


Fig. 5 The supplementing mechanism between user view and public view.

图 5 用户视图与公共视图的有效互补机制

4 性能分析

基于视图树的实视图动态选择比已有的方法具有更好的性能,这里主要从两个方面来讨论,即视图粒度和视图替换策略.

4.1 视图的粒度

视图集合中视图的粒度会影响到查询的性能.如果粒度太细,一方面,固定的实视图空间内可以容纳更多的属于部分聚集视图的多维数据碎片,但这同时也意味着,在实视图集合中寻找可以用来回答某个查询的视图需要耗费更多的时间;另一方面,相对于粗粒度视图而言,细粒度视图能够回答的查询的数量很有限.我们这里举一个实例进行说明.

例 6. 假设采用例 1 中的多维数据集,实视图集合中存储了包括 p_1,p_2,p_3 在内的多个属于部分聚集视图的多维数据碎片,其中,$p_1=\{(ProductId,\{1,1000\}),(StoreId,\{125\})\}$;$p_2=\{(ProductId,\langle 1,1000\rangle),(StoreId,\{225\})\}$;$p_3=\{(ProductId,\{50\}),(StoreId,\{50\})\}$.假设查询为 $q=\{(ProductId,\{50\}),(StoreId,\langle 1,250\rangle)\}$.如图 6 所示,即使使用 p_1,p_2,p_3 这 3 个视图的组合,也无法得到查询 q 所需要的全部数据.

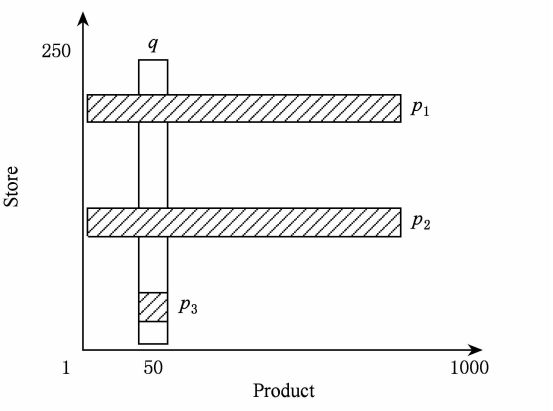


Fig. 6 Answering a query by materialized views.

图 6 使用实视图回答查询

鉴于以上情况,我们的方法把完整聚集视图作为实化对象,如果查询 q 需要使用部分聚集视图,则可以通过其宿主节点得到回答.虽然,完整聚集视图要比部分聚集视图的空间代价更大,但是,它能回答更多查询的优势可以弥补空间代价的劣势.而我们的实验结果也证明了实化完整聚集视图所带来的良好性能.另一方面,视图路径方法可以以最小的代价快速得到查询所需要的数据,提高了查询响应速度.另外,与服务器端缓存相比,客户端缓存所存储的视图的分布范围比较小,因为单个用户的查询具有局部性和可导航性.而服务器端缓存所存储的视图的分布范围则比较广,因为,相比单个用户而言,多个用户的多个查询所访问的数据可能分布在更广泛的数据空间内.因此,数据分布的集中性这个优势也会弥补完整聚集视图空间代价大的劣势,因为客户端缓存只需要存储分布比较集中的一部分数据.

4.2 视图替换策略

目前有 4 种性能比较优越的视图替换策略,包括:1)替换最近最少使用的视图(LRU);2)替换使用频率最小的视图(LFU);3)替换小尺寸视图(SFF);4)替换预期惩罚率最小的视图(SPF)[21].替换预期惩罚率$=(freq(q)\times c(q))/size(q)$,其中,$freq(q)$表示查询 q 的频率,$c(q)$表示查询 q 的执行代价,$size(q)$表示查询 q 的实际尺寸.在同时考虑多个用户的大量查询时,这些方法具有良好的性能.但是,研究的对象是单个用户的一系列查询时,上述方法在性能上就不如本文提出的视图替换方法.

在研究单一用户的一系列查询时,用户所访问的数据具有局部性和可导航性,而其可统计性特性则比较弱,比如,在一个用户会话过程中,对用户查询使用某个视图的频率进行统计是没有多大意义的.缺少有意义的视图统计数据,则上述的第 1)2)和 4)种方法的性能会大打折扣.至于第 3)种方法,也不能取得好的效果,比如,在图 4(d)中,就视图大小而言,节点 i 和 h 显然比 g,e,b 小,但是,i 和 h 在当前的保留路径上,而 g,e,b 则不在保留路径上,按照“替换小尺寸视图“的策略,则会把 i 和 h 优先从实视图集合中删除,而实际上,根据我们前面已经论述的保留路径的重要性,这种替换方法是不可取的.因此,在研究单一用户的一系列查询时,本文提出的“保留路径法”性能上更优越.

5 实验设计与结果

本节内容介绍实验设计与结果,目的在于说明,在用户缓存中,采用基于视图树的实视图动态选择,可以比已有的其他方法取得更好的性能.

5.1 实验设计

实验的硬件环境是:1 台 HP Proliant DL585 服务器作为数据仓库服务器,服务器的配置为 4 颗 AMD 皓龙 2.4 GHz CPU,32 GB 内存和 1.20 TB 硬盘;1 台普通 PC 作为客户端,执行 OLAP 分析查询,

客户端 PC 的配置为 AMD Athlon 64 b 3200+2.2 GHz CPU,1 GB 内存和 80 GB 硬盘.实验的软件环境为服务器安装 Windows Server 2003 操作系统和 ORACLE 10g 数据库管理系统;客户端安装 Window XP 操作系统.

在数据仓库服务器部分,数据库的多维数据模式包括 4 个维 D_0,D_1,D_2 和 D_3,它们分别有 4,3,4,3 个级别.多维数据的基本元 ***DB*** 共有 50 万行,并且假设多维数据的分布是均匀的,根据多维数据集的尺寸估计方法[18],估算得到以 ***DB*** 为基本元的多维数据格 M 的大小为 1500 万行.在客户端部分,我们手工设计了每次用户分析过程执行的查询.

5.2 实验 1:视图替换策略性能的比较

本实验的目的在于比较不同的视图替换策略的性能差异.我们采用 *CSR*(cost saving ratio)[20] 作为性能比较标准,具体定义为

$$CSR = \left(\sum_i (d_i - c_i)\right) \Big/ \left(\sum_j d_j\right),$$

其中,c_i 表示使用缓存时查询 q_i 的执行代价,d_i 表示不使用缓存时查询 q_i 的执行代价,$\{q_i \mid i=1,2,\cdots\}$ 是可以用缓存数据回答的查询的集合,$\{q_j \mid j=1,2,\cdots\}$是所有查询的集合.用户在一次 OLAP 分析过程(或称为一个会话)中会执行一系列查询$\{q_j \mid j=1,2,3,\cdots\}$,在每次会话结束以后都可以计算得到这次会话对应的 *CSR*.我们共设计了 10 次不同的会话,并计算得到这些会话对应的 *CSR* 的平均值.实视图空间(缓存)的大小为多维数据格 M 的 20%.

我们把本文提出的保留路径法 KPM 与其他 4 种方法作了实验对比,分别是替换最少使用的视图 LRU、替换使用频率最小的视图 LFU、替换小尺寸视图 SFF 和替换预期惩罚率最小的视图 SPF[20].图 7 显示了实验的结果,从中可以看出,KPM 的 *CSR* 值最高,达到 56%,SPF 的性能稍差一些,其 *CSR* 值为 32%,而 LFU 和 SFF 的 *CSR* 值分别只有 7%和 8%,也就是说,这两种方法只取得了很小的性能改进,如果把缓存的维护代价也考虑进来,二者的收益就更小.这主要是由于在一个用户会话过程中,对用户查询使用某个视图的频率进行统计是没有多大意义的.缺少有意义的视图统计数据,LFU 和 SFF 自然就无法取得好的性能.

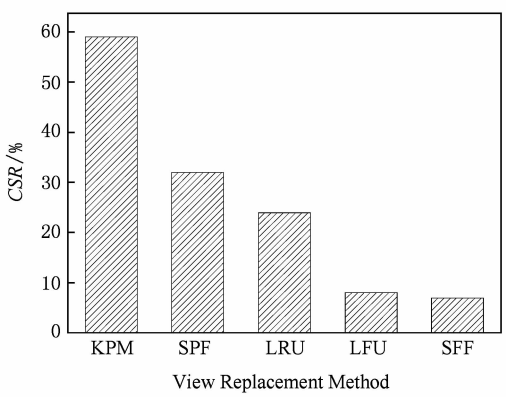


Fig. 7 Performance of different view replacement policy.

图 7 不同视图替换策略的性能

5.3 实验 2:实视图空间大小对视图替换策略性能的影响

本实验的目的在于比较不同的视图替换策略的性能受实视图空间(缓存)大小的影响.实验中,对于同一个用户会话,我们把实视图空间与多维数据格 M 的尺寸的比例从 5%逐渐增加到 20%,分别计算得到在不同空间比例时的 *CSR* 值.我们对 5 个用户会话进行了同样的操作,然后计算对应各个空间比例的 *CSR* 值的平均值.图 8 显示了变化实视图空间大小时不同视图替换策略的性能变化情况.从图 8 可以看出,LFU 和 SFF 的 *CSR* 值基本变化不大,在实视图空间从从 5%逐渐增加到 20%时,二者的 *CSR* 值只是分别从 4.0%和 3.2%增加到 8.0%和 7.0%.而 SPF 和 LRU 的 *CSR* 值有了一定的增长,分别从 8.7%和 6.6%增加到 32%和 24%.KPM 的性能最好,*CSR* 值从 10%增加到 56%.因此,实视图空间大小对 KPM,SPF 和 LRU 具有比较大的影响,空间的增加能带来显著的收益,而对 LFU 和 SFF 的影响则很小,主要原因在于单个用户在一个会话期间的数据访问具有局部性和可导航性.

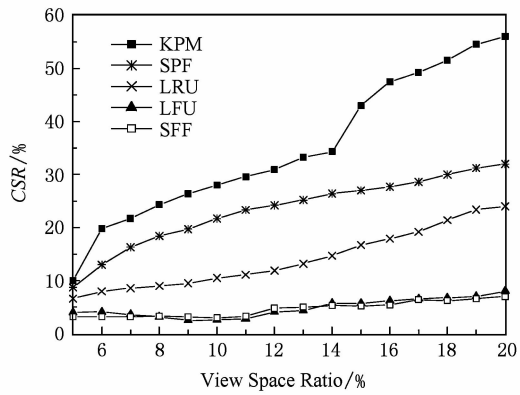


Fig. 8 Performance change of different view replacement policy when varying view space size.

图 8 变化实视图空间大小时不同视图替换策略性能变化

5.4 实验 3:逆路径增长法的性能

本实验的目的在于说明采用逆路径增长法计算视图路径时相对于朴素方法的性能改进.实验中,我

们把实视图空间与多维数据格 M 的尺寸比例从5%逐渐增加到20%,通过改变实视图空间的大小可以改变视图树中节点的数目,从而影响视图路径计算工作量.此外,我们还设计了两个不同的多维数据格,目的是为了改变格中节点的数目.我们设计了5次用户会话,针对每次用户会话,我们计算时间比 $TR=t_1/t_2$,其中,t_1 表示朴素方法的时间代价,t_2 表示逆路径增长法的时间代价;然后计算得到5次用户会话的 TR 的平均值.图9显示了实验的结果,从中可以看出,随着实视图空间的增加而导致的视图树中节点数目的增加,两种方法的时间比 TR 的值逐渐增大.由于 M_1 比 M_2 的节点数目多,因此,性能改善程度也更加明显.

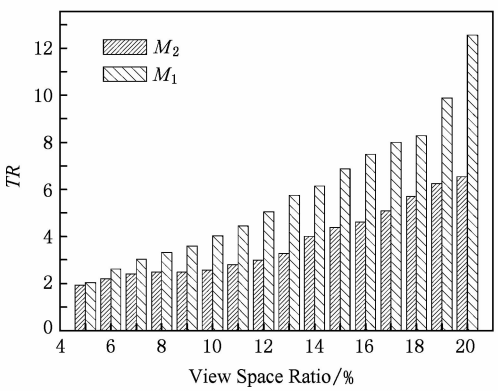


Fig. 9 Time ratio when varying the size of view space.

图9 时间比随着实视图空间大小的变化

6 相关工作

视图选择是数据仓库领域的一个热点研究问题,目前已有不少相关的研究成果[21].在静态视图选择方法中,系统会在维护时间窗口内,根据查询的统计数据把那些比较频繁发生的查询进行实化,从而提高以后到达的查询响应速度,在下一个维护时间窗口到来之前这些实视图不发生变化[12].目前已有许多关于静态视图选择的研究,其中比较典型的几种方法有基于多维数据格[1,22]、基于MVPP[3,5]、基于AND-OR图[6]和基于随机优化[7]的方法.

实视图选择的静态方法确实改善了数据仓库的总体查询性能,但是,它和决策支持分析的动态特性是不相符的.尤其对于即席查询,专家为了在数据仓库中找到一些感兴趣的趋势,他们提出的查询通常是难以预测的.另外,数据仓库中的数据和查询的特征都是随着时间而变化的,静态选择方法得到的结果可能很快就过期.鉴于静态视图选择方法存在的缺点,为了满足用户变化的查询需求,我们有时需要动态的选择方法.

目前,研究人员已经提出不少动态选择方法.Deshpande等人于1998年提出了一个基于块文件的方法[9]来解决查询缓存问题,它把多维查询均匀地分成多个块,并对这些块进行缓存.但是,基于块文件的方法其视图选择和替换策略以及具体实施都比较复杂.Kotidis等人提出的Dynamat方法[20]采用了基于需求的抓取策略,该方法固定保存一定粒度的实化视图来满足被称为MRQ(multidimensional range queries)类型的查询需要.实验结果证明,DynaMat的性能优于最优的静态视图选择算法.但是,这种方法在进行视图选择之前没有考虑用户的访问模式信息.Choi等人提出了基于谓词的动态视图管理方法——PREDICATE[12],可以充分利用关系数据库系统的能力,同时也去掉针对多维数据碎片[20]的各种约束,使得它们可以回答更多的查询.但是,PREDICATE方法没有给出具体的代价评估方法,也没有详细的视图替换策略.Sapia于2000年提出了基于缓存预测的方法PROMISE[16],对某个实际应用系统中的260个会话(共包含3 150个查询)进行了两个月的跟踪,得到的分析结果显示,缓存预抓取策略是可行的.但是,在实际应用中,可能的查询数量通常比较大,使用PROMISE方法预测下一个阶段是哪个查询非常耗费时间,而且,它需要单个实视图的粒度足够小,从而才能捕捉到查询间的细微的差别.文献[11]的研究发现,在一个特定的时间内,用户经常访问某个特定的区域.因此,在这些查询之间就存在语义相关性,这种信息就可以用来重写一个查询或者缓存一个查询来回答更多的查询.但是,不同查询之间语义关系的确定以及基于语义相关性的查询重写都比较复杂.文献[14]提出采用基于蚁群优化的算法来生成实视图.针对视图选择问题的其他动态选择方法还包括文献[10,15]等等.

7 结束语

多维数据实视图选择是数据仓库研究领域的一个重要问题,它对于提高数据仓库总体性能具有重要意义.目前的方法大都适用于服务器端,比如在静态视图选择方法中,实视图集合被存放到数据仓库中,而在动态视图选择方法中,实视图集合则被存放到服务器端的缓存中.这些方法都需要基于来自多个用户的大量查询样本的统计数据,其目的在于提高

多个用户查询的平均响应时间.而基于用户的缓存则是针对某个具体用户的,为之设计的缓存替换策略需要考虑单个用户会话的特点.本文提出的基于视图树的实视图动态选择充分利用了用户在会话期间的数据访问局部性和可导航性,取得了较好的性能."逆路径增长法"支持快速的视图路径计算,提高了查询的响应速度;"保留路径法"则实现了更合理的视图替换.实验证明,我们的方法比已有动态选择方法能够取得更好的性能.

在未来的工作中,我们将继续研究针对用户缓存的实视图动态选择,并考虑引入人工智能领域的模型预测研究成果,实现对用户下阶段查询的主动预测,提前实化即将被使用的视图,从而更好地响应用户查询.

参 考 文 献

[1] Harinarayan V, Rajaraman A, Ullman J D. Implementing data cubes efficiently [C] //Proc of ACM SIGMOD'96. New York: ACM, 1996: 205-216

[2] Baralis E, Paraboshi S, Teniente E. Materialized view selection in a multidimensional database [C] //Proc of the 23rd Int Conf on Very Large Data Bases. San Fransisco: Morgan Kaufmann, 1997: 156-165

[3] Yang J, Karlapalem K, Li Q. Algorithms for materialized view design in data warehousing environment [C] //Proc of VLDB'97. New York: Morgan Kaufmann, 1997: 136-145

[4] Kalnis P, Mamoulis N, Papadias D. View selection using randomized search [J]. Data and Knowledge Engineering, 2002, 42(1): 89-111

[5] Yousri N A R, Ahmed K M, El-Makky N M. Algorithms for selecting materialized views in a data warehouse [C] // Proc of AICCSA'05. Piscataway, NJ: IEEE, 2005: 27

[6] Gupta H, Mumick I S. Selection of views to materialize in a data warehouse [J]. IEEE Trans on Knowledge and Data Engineering, 2005, 33(1): 24-43

[7] Wang Ziqiang, Zhang Dexian. Optimal genetic view selection algorithm under space constraint [J]. Int Journal of Information Technology, 2005, 11(5): 44-51

[8] Xue Yongsheng, Lin Ziyu, Duan Jiangjiao, et al. Dynamic selection of materialized views of multi-dimensional data with multi-users and multi-windows method [J]. Journal of Computer Research and Development, 2004, 41(10): 1703-1711 (in Chinese)
(薛永生, 林子雨, 段江娇, 等. 用多用户多窗口处理多维视图动态选择[J]. 计算机研究与发展, 2004, 41(10): 1703-1711)

[9] Deshpande P, Ramaswamy K, Shukla A, et al. Caching multidimensional queries using chunk [C] //Proc of ACM SIGMOD'98. New York: ACM, 1998: 259-270

[10] Tan Hongxing, Zhou Longxiang. Dynamic selection of materialized views of multi-dimensional data [J]. Journal of Software, 2002, 13(6): 1090-1096 (in Chinese)
(谭红星, 周龙骧. 多维数据实视图的动态选择[J]. 软件学报, 2002, 13(6): 1090-1096)

[11] Yao Qingsong, An Aijun. Using user access patterns for semantic query caching [G] //LNCS 2736: Proc of the Int Conf on Database and Expert System Applications. Berlin: Springer, 2003: 37-746

[12] Choi C, Yu J, Lu Hongjun. Dynamic materialized view management based on predicates [G] //LNCS 2642: Proc of the 5th Asia-Pacific Web Conference on Web Technologies and Applications. Berlin: Springer, 2003: 583-594

[13] Lin Ziyu, Yang Dongqing, Song Guojie, et al. Materialized views selection of multi-dimensional data in real-time active data warehouses [J]. Journal of Software, 2008, 19(2): 301-313 (in Chinese)
(林子雨, 杨冬青, 宋国杰, 等. 实时主动数据仓库中多维数据实视图的选择[J]. 软件学报, 2008, 19(2): 301-313)

[14] Drias H. Generating materialized views using ant based approaches and information retrieval technologies [C] //Proc of CIDM'11. New York: IEEE, 2011: 276-282

[15] Zhou Lijuan, Geng Haijun, Xu Mingsheng. An improved algorithm for materialized view selection [J]. Journal of Computers, 2011, 6(1): 130-138

[16] Sapia C. PROMISE: Predicting query behavior to enable predictive caching strategies for OLAP Systems [G] //LNCS 1874: Proc of the 2nd Int Conf on Data Warehousing and Knowledge Discovery Conference. Berlin: Springer, 2000: 224-233

[17] Dar S, Franklin M J, Jonsson B T, et al. Semantic data caching and replacement [C] //Proc of VLDB'96. San Francisco: Morgan Kaufmann, 1996: 330-341

[18] Shukla A, Deshpande P M, Naughton J F, et al. Storage estimation for multidimensional aggregates in the presence of hierarchies [C] //Proc of VLDB'96. San Francisco: Morgan Kaufmann, 1996: 522-531

[19] Shaffer C A. A Practical Introduction to Data Structures and Algorithm Analysis [M]. 2nd ed. East Rutherford, New Jersey: Pearson Education, 2001

[20] Kotidis Y, Roussopoulos N. DynaMat: A dynamic view management system for data warehouses [C] //Proc of ACM SIGMOD'99. New York: ACM, 1999: 371-382

[21] Lin Ziyu, Yang Dongqing, Wang Tengjiao, et al. Research on materialized view selection [J]. Journal of Software, 2009, 20(2): 193-213 (in Chinese)
(林子雨, 杨冬青, 王腾蛟, 等. 实视图选择研究[J]. 软件学报, 2009, 20(2): 193-213)

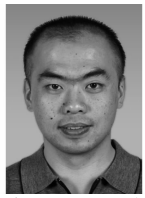
[22] Shukla A, Deshpande P, Naughton J F. Materialized view selection for multidimensional datasets [C] //Proc of VLDB'98. San Francisco: Morgan Kaufmann, 1998: 488-499

Lin Ziyu, born in 1978. Received his PhD degree in computer software and theory from Peking University in 2009. Now he is assistant professor and master's supervisor at Xiamen University. Member of China Computer Federation. His main research interests include data warehouse, OLAP, data mining, etc.

Zou Quan, born in 1982. Reveived his PhD degree in artificial intelligence and information processing from Harbin Institute of Technology in 2009. Now he is assistant professor and master's supervisor at Xiamen University. His main research interests include massive data mining and bioinformatics (zouquan@xmu.edu.cn).

Lin Chen, born in 1982. Received the PhD degree in computer software and theory from Fudan University in 2010. Now she is assistant professor and master's supervisor at Xiamen University. Member of China Computer Federation. Her main research interests include retrieving, mining and organizing unstructured and semistructured data (chenlin@xmu.edu.cn).

Lai Yongxuan, born in 1981. Received his PhD degree in applied technology of computer science from Renmin University of China in 2009. Now he is assistant professor and master's supervisor at Xiamen University. Member of China Computer Federation. His main research interests include database, data management on sensor network, opportunistic network, etc (laiyx@xmu.edu.cn)



Zheng Wei, born in 1979. Received his PhD degree in computer science from University of Manchester in 2010. Now he is assistant professor at Xiamen University. His main research interests include distributed computing, workflow scheduling, algorithms, etc.