

文章编号:1001-9081(2007)06-1397-03

## 基于源视图增量的在线实化视图自维护

刘 海<sup>1,2</sup>, 汤 庸<sup>2</sup>, 陈启买<sup>1</sup>

(1. 华南师范大学 计算机工程系, 广东 广州 510631; 2. 中山大学 计算机科学系, 广东 广州 510275)

(namelh@gmail.com)

**摘 要:**借鉴传统的基于基表变化的数据仓库维护方法 Strobe, 提出一种基于源视图增量的在线实化视图自维护方法, 使实化视图的状态保持与底层数据源的一致性。这种方法不仅保持数据仓库数据的一致性, 而且还能够加快实化视图维护的速度, 减少底层信息源与数据仓库之间的网络通信负担。

**关键词:**源视图; 视图自维护; 数据仓库

**中图分类号:** TP311.13 **文献标识码:** A

## Online self-maintenance of materialized view based on source view increment

LIU Hai<sup>1,2</sup>, TANG Yong<sup>2</sup>, CHEN Qi-mai<sup>1</sup>

(1. Department of Computer Engineering, South China Normal University,  
Guangzhou Guangdong 510631, China;

2. Department of Computer Science, Sun Yat-sen University, Guangzhou Guangdong 510275, China)

**Abstract:** View maintenance is an important field in the research of dataware house. This paper got idea from the strobe algorithm and provided an online view self-maintenance method based on source view's increment to keep the materialized view consistent with the data sources. This method does not require querying back to data source; it can accelerate view maintenance and decrease the burden of communication between dataware house and data sources.

**Key words:** source view; view self-maintenance; data warehouse

### 0 引言

随着数据通信和互联网技术的迅速发展,人们经常通过建立数据仓库来实现高效地从远程分布、自治、而且通常是异质的多个信息源进行数据集成,以便为人们进行信息查询和决策支持提供综合的数据源。为了提高数据仓库的响应速度,通过选取一些视图进行实化,而当底层信息源发生变化后,采用实化视图的维护方法,将底层信息源的变化及时反映在数据仓库中,以确保决策查询结果的正确性。

实化视图维护是数据仓库中一个重要的研究领域,本文在斯坦福大学 WHIPS 提出的数据仓库体系结构基础上,改变传统的基于基表变化来进行实化视图维护的方法,将数据仓库的维护建立在底层信息源的源视图的变化之上,在多数据源数据集成环境下,以 PSJ 视图增量自维护为背景,在数据仓库中保留源视图副本的前提下,提出了基于源视图的事务方式维护算法,以使数据仓库中的数据保持与底层信息源的一致性。

### 1 相关的研究工作

关于数据仓库维护,国内外的研究具有代表性的成果是斯坦福大学 DB Group 工作组在 WHIPS<sup>[1,2,4]</sup> 项目中做出的。他们提出了数据仓库维护的体系结构,将数据仓库维护的体系结构分为集成器与监视器两大部件,监视器负责分析和报送数据源基表内容的变化,集成器在基表内容变化的基础上

运行相容性控制算法来保证数据仓库数据的一致性。对于数据仓库的维护,研究人员根据仓库的维护时机,提出了立即维护方式、延迟维护方式和周期维护方式<sup>[7,9,10]</sup>。立即维护方式是在基表更新事务尾部加入视图维护操作,这种方式虽然保证视图的实时更新,但严重增加了数据源更新事务的负担。当视图维护操作包含远程数据访问时,这种负担难以接受。它适用于小的且底层数据源变化缓慢的数据仓库。延迟维护方式是在基表更新事务完成之后对视图进行维护操作。虽然降低了更新的实时性,但它也降低了更新事务的负担,延迟维护方式一般用于数据仓库环境下的视图维护。周期维护方式则指定更新的时间间隔,周期性的更新视图。这是一种折衷的策略,既考虑到系统资源,又兼顾了对用户提供服务的性能。根据视图的维护方法可以分为重新计算视图方法和补偿查询方法。典型算法包括单数据源环境实化视图相容维护的 ECA 算法<sup>[9,10]</sup>、多数据源环境实化视图相容性维护的 Strobe 算法<sup>[7]</sup> 和自维护方法<sup>[11,12]</sup>。针对实化视图是涉及的基表是否在同一数据库中,人们提出了单数据源实化视图的维护和多数据源的维护。通过这些算法,使得数据仓库中实化视图的内容和底层信息源的内容保持某种程序的一致性,研究人员根据数据仓库环境中数据一致性的程度,将实化视图维护的一致性分为不一致性、收敛一致性、弱一致性、强一致性和完全一致性等五个不同的层次<sup>[5]</sup>,并对具体算法达到的一致性层次做了具体的分析。

### 2 基于源视图增量的数据仓库在线维护

收稿日期:2006-12-08;修订日期:2007-03-05 基金项目:广东省科学技术基金项目(04105503, 05200302)

作者简介:刘海(1974-),男,湖南张家界人,讲师,博士研究生,主要研究方向:数据库与数据仓库、协同软件; 汤庸(1964-),男,湖南张家界人,教授,博士,主要研究方向:数据库与协同软件; 陈启买(1968-),男,湖南人,副教授,硕士,主要研究方向:数据库与数据挖掘。

### 2.1 一种改进的数据仓库维护体系结构

WHIPS 项目组提出的数据仓库体系结构中,数据源的监视器负责监视基表信息的变化,无论这种基表的变化是否会引起实化视图内容的变化,都将送到集成器,这样一来就可能造成一些不必要的网络与计算开销。我们提出一种基于源视图的变化的策略(见图1虚线所示),只有在数据源的基表变化的内容引起了相应的源视图的内容发生变化以后,才将源视图的变化送给集成器。在引入了源视图后,数据仓库的维护体系,变成了基表-源视图-实化视图,监视器待基表内容发生了变化以后,首先要检测是否会引起源视图的内容的变化,若源视图的内容无变化,则不需要进行实化视图的相应的维护,只有基表内容变化引起源视图内容的变化以后,才将源视图的变化送往数据仓库去进行实化视图的维护。

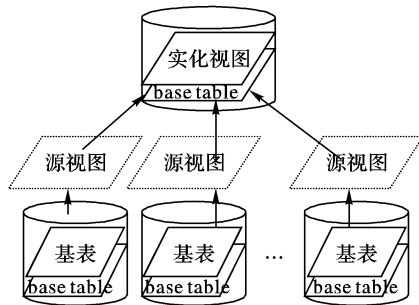


图1 改进的数据仓库维护体系结构

### 2.2 视图维护的相关定义和数据结构

#### 2.2.1 相关的术语定义

**定义1** 实化视图 MV (Materialized View): 实化视图是定义在数据源基本关系之上的, 一个用 PSJ 表达式表示的 SQL 表达式, 即:

$$MV = \pi_{proj} \sigma_{select-cond} R_1 \bowtie R_2 \bowtie \dots \bowtie R_n \quad (1)$$

**定义2** 源视图 SV (Source View): 源视图是由位于一个独立数据源的基本关系组成的, 用 PSJ 表达式表示的 SQL 表达式, 即:

$$SV_i = \pi_{proj} \sigma_{select-cond} R_1 \bowtie \dots \bowtie R_i, 1 < i < n \quad (2)$$

$R_i$  属于同一个数据源。

**定义3** 基于源视图的实化视图重写 VRW (View Rewriting): 利用实化视图至源视图的分解算法, 可以将实化视图分解成为一个由若干源视图组成的 PSJ 表达式, 即:

$$MV = \pi_{proj} \sigma_{select-cond} SV_1 \bowtie SV_2 \bowtie \dots \bowtie SV_n \quad (3)$$

本文集中于基于源视图变化的实化视图的维护, 对于实化视图至源视图的分解算法及其证明, 参见文献6。当底层数据源的某个基本关系发生变化后, 计算基本关系变化所引起的源视图的变化, 只有当某个源视图的内容发生变化  $\Delta SV_i (1 \leq i \leq n)$  后, 实化视图的内容才将发生变化。

**定义4** 基于源视图增量的实化视图的增量  $\Delta MV$

$$\Delta MV = \pi_{proj} \sigma_{select-cond} SV_1 \bowtie SV_2 \bowtie \dots \bowtie \Delta SV_i \bowtie \dots \bowtie SV_n \quad (4)$$

#### 2.2.2 事务维护算法中涉及的重要数据结构

为了提高响应的速度, 临时在数据仓库中保留有各个源视图的副本, 因此计算  $\Delta MV$  仅需要在本地做连接运算即可。为了更好地描述事务方式下视图维护算法, 首先介绍视图维护中需要的数据结构和方法。

1) 实化视图更新文件队列 updateActionList

updateActionList 记录随着源视图的变化需要对实化视图进行的更新操作(包括插入和删除)和操作值, 当且仅当执行

updateActionList 记录的更新操作后, 能够保证实化视图和数据源的状态一致, 系统将 updateActionList 中的所有更新操作作为一个事务提交。

2) 事务增量文件队列 IncrementFileList

IncrementFileList 中记录了所有数据仓库端收到的但还未被处理事务增量文件。

3) 方法 *excuteMVDelta*( $Q_i$ )

它是以非阻塞的方式执行的, 也就是说, 在该函数的执行过程中, 集成器可能又会收到新的更新消息, 它的作用是负责计算由于源视图  $\Delta SV_i$  发生变化, 对实化视图所产生的增量  $\Delta MV$ ,  $\Delta MV$  包含插入的元组, 也可能包含有删除的元组。

4) 方法 *deleteTuples*( $V_i, \Delta V_i$ )

从视图  $V_i$  中删除和  $\Delta V_i$  中包含的元组。

5) 方法 *insertTuples*( $V_i, \Delta V_i$ )

从视图  $V_i$  中插入和  $\Delta V_i$  中包含的元组。

6)  $MV(\Delta SV_i)$

表示在实化视图表达式中, 用元组  $\Delta SV_i$  代替  $MV$  中所在的源视图后所得到的关系表达式。

7) UQS 为正在集成端进行增量计算但又没有得到结果的查询对象队列。

8) *Pending*( $Q_i$ ) 队列为查询对象  $Q_i$  正在进行时, 发生的交叠更新。

#### 2.3 基于源视图事务增量的维护算法具体描述

假定在同一个信息源上发出的任意两条消息, 其到达集成器的顺序与它们发出的顺序是一致的。如果不一致, 我们通过对源视图的事务增量文件添加计数器号来实现解决事务增量文件乱序问题。

**定义4**  $\Delta SV$  事务  $T$  所引起的源视图更新列表, 它包含着由事务  $T$  对源视图的插入分量和删除分量。  $IU(T) \subseteq UL(T)$  是事务  $T$  对源视图的插入分量。

##### 2.3.1 基于源视图增量的自维护算法

每个信息源端:

一旦检测某个事务操作对于本地的源视图产生增量, 监视器立即向数据仓库报告, 并将增量文件  $\Delta SV$  传送给集成器端。

在数据仓库端:

1) 初始化将视图维护目标对象中的 updateLinkedList 和 UQS 队列为空。其中 updateLinkedList 为实化视图更新文件队列。UQS 为正在集成端进行增量计算但又没有得到结果的查询对象队列。

2) 当收到某个源视图的增量  $\Delta Sv_i$  后,

- 对于  $\Delta Sv_i$ , 将每个元组依次更新到对应的源视图中。

- 对于任意  $Q_j \in UQS$ , 将  $\Delta Sv_i$  加入 *pending*( $Q_j$ )。

- 将 *deleteTuples*( $MV, \Delta Sv_i$ ) 加入 updateLinkedList。

3) 令  $Q_i = MV(\Delta SV_i)$ , 计算  $A_i = excuteMVDelta(Q_i)$ , 并置 *pending*( $Q_i$ ) 为空。

4) 将 *Insert*( $MV, A_i$ ) 插入到 updateLinkedList 的最后。

- 对于任意的  $U \in pending(Q_i)$ , 计算 *deleteTuples*( $A_i, U$ )。

- 将 *Insert*( $MV, A_i$ ) 加入 updateLinkedList。

5) 当 UQS 为空时, 将 updateLinkedList 队列中的所有内容作为一个原子事务更新到实化视图里去, 并重新设置 updateLinkedList 为空。

## 2.4 应用实例

假设存在一个基于源视图定义的实化视图  $MV = SV_1 \bowtie SV_2 \bowtie SV_3$ ,  $SV_1, SV_2, SV_3$  是分别位于数据源  $x, y, z$  的三个源视图,其中  $SV_1 = r_1 \bowtie r_2$ , 是位于信息源  $x$  的源视图,它定义在基表关系  $r_1$  和  $r_2$  之上,初始化时,各个源视图的内容和基表的内容分别如表1和表2所示。

表1 数据源  $x$  的基本关系表

r1:	A	E	r2:	E	C
	1	1		1	2

表2 数据仓库中最初的源视图副本

sv1:	A	B	sv2:	B	C	sv3:	C	D
	1	2		-	-		3	4

首先,根据实化视图的定义, $MV$ 的内容为空,假定在数据源  $x$  发生一个更新  $U = insert(r_1, [1, 4])$ ,很明显这次更新不会引起源视图内容的变化,因此这次更新不必送往集成器进行实化视图的维护。

其次,假定由于基表的变化,先后引起了源视图变化  $\Delta SV_2 = insert(SV_2, [2, 3])$  and  $\Delta SV_1 = delete(SV_1, [1, 2])$ ,此时需要将源视图变化的内容送至数据仓库进行维护,集成器维护过程如下:

1) 队列 UpdateLinkedList 为空,集成器收到来自于数据源  $y$  的源视图增量  $\Delta SV_2$ ,并利用  $insertTuples(SV_2, \Delta SV_2)$  更新源视图  $SV_2$ 。

2) 为了进行视图的维护,集成器端产生一个查询  $Q_1 = ExcuteMVDelta(\Delta SV_2)$ ,并将  $InsertTuples(MV, Q_1)$  插入到 updateLinkedList。

3) 在查询  $Q_1$  执行的过程中,集成器收到了来自于数据源  $x$  的源视图增量  $\Delta SV_1$ ,它首先将  $\Delta SV_1$  加入到队列  $pending(Q_1)$ ,利用  $DeleteTuples(SV_1, \Delta SV_1)$  更新源视图  $SV_1$ ,然后将  $DeleteTuples(MV, \Delta SV_1)$  插入到 UpdateLinkedList,至此队列 UpdateLinkedList 的内容为  $\langle InsertTuples(MV, Q_1), DeleteTuples(MV, \Delta SV_1) \rangle$ 。

4) 当集成器计算完查询  $Q_1$ ,结果为  $[1, 2, 3, 4]$ ,由于队列  $pending(Q_1)$  不为空,但是  $UQS$  为空,所以集成器将更新队列的全部内容作为一个事务全部更新到实化视图中去,结果实化视图的内容仍然为空,得到正确的维护结果。

## 2.5 基于视图实化的视图自维护算法优点

基于源视图自维护算法,可以使数据仓库中的实化视图达到一致性的状态,详细证明见文献[8],在此我们只做一些定性的分析。由于我们的体系结构中,底层信息源的监视器过滤掉那些对集成端源视图无关的基表增量,这在一定程度上减少了实化视图维护的负担,将一部分负担转移到各个信息源。集成器的维护对象从每个数据源上的多表简化为一个或者多个源视图,集成计算逻辑得到简化。从整个系统来说,通信的负担减少了,维护性能得到了提高,从而可以减少数据仓库监视器与集成器之间的网络开销,提高维护的效率。从算法本身来看,基于源视图自维护算法和 T - Strobe 算法对比,1)由于在数据仓库端保留有各个源视图的副本,因此在执行  $excuteMVDelta(Q_i)$  方法时不需要回访查询,因此减少了集成器方和监视代理方的通信负担,同时加快了在线维护速

度,可以适用于要求快速响应的集成应用环境。2)由于这种算法是基于源视图的算法,减少了连接运算的次数,提高了集成器的运算速度。

## 3 结语

借助于基于源视图的在线维护方法,使数据仓库中实化视图保持与底层信息源的一致性,同时又可以减少了信息源与数据仓库之间的信息传递的次数,提高了维护的效率,减轻了网络的负担。下一步的工作是实化视图分解为源视图的算法的进一步完善,以及在数据仓库中引入视图定义中有聚集函数的数据仓库维护方法的实现。

## 参考文献:

- [1] HAMMER J, GARCIA-MOLINA H, WIDOM J, et al. The Stanford Data Warehousing Project[J]. In IEEE Data Engineering Bulletin, 1995, 18(2): 41 - 48.
- [2] WIENER JL, GUPTA H, LABIO WJ, et al. A System Prototype for Warehouse View Maintenance[A]. Workshop on Materialized Views [Z]. 1996. 26 - 33.
- [3] GUPTA A, MUMICK I. Maintenance of Materialized Views: Problems, Techniques, and Applications[J]. IEEE Quarterly Bulletin on Data Engineering, 1995, 18(2) : 3 - 18.
- [4] LABIO WJ, YUE ZG, WIENER JL, et al. The WHIPS Prototype for Data Warehouse Creation and Maintenance[A]. Proc of the ACM SIGMOD Conf [C]. Tucson, Arizona, 1997. 557 - 559.
- [5] ZHUGE Y. Incremental Maintenance of Consistent Data Warehouse [D]. June 1999.
- [6] YUE ZG, MOLINA HG, HAMMER J, et al. View maintenance in a warehousing environment[A]. In: Proceedings of ACM SIGMOD Conference [C]. San Jose, USA, 1995. 316 - 327.
- [7] YUE ZG, GARCIA-MOLINA H, WIENER JL. The Strobe Algorithms for Multi-Source Warehouse Consistency[Z]. PDIS'96, Miami Beach, Florida, December 1996.
- [8] 刘海. 数据仓库中在线集成器的研究与实现[D]. 广州: 华南师范大学, 2004.
- [9] YUE ZG, GARCIA-MOLINA H, WIENER JL. Consistency Algorithms for Multi-Source Warehouse View Maintenance[J]. Distributed and Parallel Databases Journal (DAPD), Kluwer, July 1997.
- [10] YUE ZG, WIENER JL, GARCIA-MOLINA H. Multiple View Consistency for Data Warehousing[A]. ICDE'97[C]. Birmingham, UK, April 1997.
- [11] TOMPA FW, BLAKELEY JA. Maintaining Materialized Views without Accessing Base Data[P]. In Information System, 1988, 13(4): 393 - 406.
- [12] HUYN N. Efficient View Self-Maintenance[A]. In Proc. Workshop on Materialized-Views: Techniques and Applications[C]. Montreal, Canada, 1996. 17 - 25.
- [13] QUASS D, GUPTA A, MUMICK I, et al. Making Views Self-Maintenance for Data Warehousing[A]. In PDIS, Miami Beach, Florida, Dec. 1996. 158 - 169.
- [14] HUYN N. Exploiting Dependencies to Enhance View Self-Maintainability[R]. Technical Report, Stanford University, 1997.
- [15] HUYN N. Efficient Self-Maintenance of Materialized Views[R]. Technical Report, Stanford University, 1996.
- [16] QUASS D, WIDOM J. On-Line Warehouse View Maintenance[J]. In Proc. ACM SIGMOD Intl. Conf. on Management of Data[C]. 1997, 26(2): 393 - 404.