计算机科学1998Vol. 25№. 6

关系数据库系统中的视图

On Views in Relational Database Systems

谭红星 周龙骧

（中国科学院数学研究所 北京100080）

（河南大学计算机学院 开封475001）

摘 要 Views play important roles in relational database systems, while there are still some problems to solve. In this paper we will review these problems on two aspects: views and materialized views. Problems of views mainly include update propagation or translation, i. e., how to translate updates on views into those on base relations. Materialized views came into existence along with the appearance of some new kinds of application on the database systems, such as distributed databases, mobile databases and data warehouses. Problems of materialized views mainly include selection, maintenance of materialized views and query optimization using materialized views.

关键词 Views, Materiallzed Views

一、概述

数据库系统的一个重要特征是它提供了数据独立性，这可分为物理独立性和逻辑独立性。按照ANSI/SPARC框架，数据库系统的体系结构为三级模式，即存储模式、概念模式和外模式。物理独立性通过在概念模式和存储模式间建立映射实现，逻辑独立性则通过在概念模式和外模式间建立映射实现。关系数据库中外模式是通过视图实现的。

常规视图由关系上的检索操作定义，这里的关系可以是基表也可以是已定义的视图。视图本身并不保存数据，当用户访问视图数据时，系统通过视图定义将对视图的访问映射为对基表的访问。

由于视图担负外模式的职能，就应允许用户在其上执行常规的数据库操作，如检索、更新等。视图上的检索操作容易实现，这无非是将其与视图的定义——基表或其它视图上的检索操作——相联结。而更新操作的实现则要困难得多，本文将在第二节描述更新视图所面临的问题以及人们提出的解决方案。

客观地说，视图更新问题虽未得到完善的解决，但却并没有妨碍关系数据库系统日益成熟并被广泛地应用，这种现状导致传统视图中存在的问题渐渐被人们忽略。然而随着新应用的出现，原有数据库系统的结构不再能满足需要，视图又重新得到了重视，这些新的应用包括分布式数据库、移动数据库和数据仓库等。人们在这类应用中重新拾起视图是为了减少访问基表的代价，而让视图起到数据缓冲的作用，这时的视图不像传统视图那样仅保存其定义，还需要保存从基表中检索到的数据，故称为实视图，其问题也不再仅仅是更新传播，还有如下几个方面：

·如何根据需要选择实视图；

·如何维护实视图，即如果基表的内容发生变化，如何将这些变化反映到实视图中来；

·如何有效地利用实视图，特别是利用实视图进行查询优化。

本文通过一个例子介绍传统视图的更新问题以及讨论实视图的有关问题，最后提出了一些有待解决的问题。

二、用例

设数据库中有两个基表：Employee(EName, Dept, Sal)和Management(Dept, Mgr)，它们分别表示公司员工和部门的管理情况，Employee三个属性的含义依次是姓名、所属部门和工资；Management的属性Mgr指本部门的经理。这两个关系的当前内容如下：

Employee

EName	Dept	Sal
Joe	Art	2,000
Fred	Music	3,000
Gorge	Music	2,500

Management

Dept	Mgr
Art	Sally
Music	Ira

在这些基表上定义了如下视图：

EDM = $\Pi_{EName,Dept,Mgr}$ (Employee ⋈ Management)；

EM = $\Pi_{EName,Sal.,Mgr}$ (Employee ⋈ Management)；

Emp=Employee；

Mng=Management；

ExeEmp=$\Pi_{EName,Dept}(\sigma_{Sal>2000}$(Employee))；

我们在这些关系(包括基表和视图)上准备执行如下更新操作：

U1：replace Mgr with "Sally" in EM where EName="Fred"；

U2：replace Mgr with "Sally" in EDM where EName="Fred"；

U3：delete from Employee where Sal>1500；

三、传统视图的更新传播

由于传统视图不保存数据，施加在其上的更新操作最终都要转换成基表上的更新操作，这个转换过程称为更新传播或更新翻译。由更新后的数据库可得到新的视图，我们希望新视图为更新直接施加在旧视图上所得，如图1所示，在该图中，V 表示视图定义，U 表示施加在 V 上的更新操作，T 表示对 U 的翻译，T(U)则是施加在 DB 上的更新操作，由于 T(U)可能涉及到 DB 中的多个表，故 T(U)定义为一组更新操作的集合。

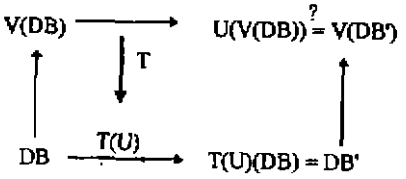

图1 视图更新转换示意图

更新传播的主要问题是二义性和副作用。二义性是指 T 的选择不止一种，如 T'和 T"均可使得 U(V(DB))=V(DB')。但其中只有一个符合实际情况。副作用则指式 U(V(DB))=V(DB')得不到满足，T(U)执行后，V(DB)中被更新的元组不是过多就是过少。

例1：考察 U1，其翻译至少有两种选择：

T'(U1)：{replace Dept with "Art" where EName="Fred"in Employee}

T"(U1)：{replace Mgr with "Sally"where Dept ="Music" in Management}

即其一是在部门间调动雇员，其二是更换部门经理，这就是更新传播的二义性，从应用逻辑的角度来看，仅有一种符合实际情况。

例2：考察 U2，合理的选择是：

T(U2)：{replace Mgr with "Edo"where Dept= "Music"in Management}

注意这时 EDM 的变化：

Before T(U2)

EName	Dept	Mgr
Joe	Art	Sally
* Fred	Music	Ira
* * Gorge	Music	Ira

After T(U2)

EName	Dept	Mgr
Joe	Art	Sally
Fred	Music	Edo
Gorge	Music	Edo

* 应被更新的元组，* * 被额外更新的元组

T(U2)导致了 EDM 中额外的更新，这就是更新传播的副作用。

一般认为，视图更新问题产生的原因是视图定义中缺乏足够的语义信息用于完成对视图的更新，人们最初定义视图的目的仅在于查询、信息隐藏以及访问权限控制等方面，而后来则要它承担外模式的重担，使之难以胜任。因此，要解决该问题应从挖掘视图的语义信息入手，使之尽量能被更新。较为突出的工作有：

1. 探索可更新视图应具备的条件[4]。一般认为，更新翻译的正确性标准应为图1中的 U(V(DB)=V(DB')。

在此标准下，我们需要讨论正确翻译的条件，即判断哪类视图的哪类更新可以被正确地翻译。这可从两方面入手，即外延条件和语法条件。所谓外延条件就是基表内容所应满足的条件，又称作语义条件；

语法条件指数据库和视图的定义，即基表和基表之间、视图和基表之间以及实体完整性、函数依赖等应满足的条件。

2. 利用视图的补视图[2]。Bancihon 和 Spytaros 给我们提供了一个全新的角度来考察视图的语义问题。通过视图，用户可检索数据库中的部分数据，其它数据则对用户隐蔽，这种效果同样可以用在更新上：用户仅能更新他通过视图可以"看到"的数据，其他的数据则不应受到更新的影响，即更新翻译的准则是保持该视图外的数据不变。

设欲更新的视图为 V，V 外的数据可通过另一视图 V′得到，那么，通过 V 和 V′可以获得整个数据库的数据，称 V 和 V′互为补视图。翻译 V 上的更新操作 U 时，应使得 T(U)保持 V′不变。如例1中，Emp 是 EM 的补视图，U1要保持 Emp 不变，可选择 T″。

可以看出一个视图可以有多个补视图，如 Mng 也是 EM 的补视图，若要保持 Mng 不变，则应选择 T′。因此选择不同的补视图，就是选择了不同的翻译算法。当一个视图的补视图 V′确定后，就可判断一个更新操作 U 是否可被正确地翻译：如果可以找到 T 使得 T(U)保持 V′不变，则 U 可翻译，否则不可。

3. 在定义视图时定义更新翻译的方法[9]。更新翻译问题，仅仅通过理论推导难以得到完善的解决，二义性问题有着复杂的实际背景，那些用 Join 运算定义的视图，更新时很容易会出现副作用，人们在应用逻辑中能够直觉地感到这一点。因此更新问题较为自然的解决途径是由数据库系统和用户协作：由系统分析问题产生的原因，列举出所有可能的解决方案，再由用户根据实际情况在这些方案间选择。如例1中二义性问题的解决严重依赖于具体的应用，系统能够分析出可能的解决方案，但具体怎么做还要由用户决定。

四、实视图

分布式数据库、移动数据库需要访问异地数据，这种访问的代价一般比较高，如在通讯上，特别是在移动数据库中，异地的数据未必随时可用。为减少这种代价，人们将本地经常用到的异地数据暂时存放在本地，这些数据即为异地数据的实视图。在异构数据库集成、OLAP 等应用中，由于源数据库存取权限、综合数据生成速度等方面的限制，人们也常构造实视图。特别是数据仓库，其本身就是一个实视图的集合。

但在实视图的应用中又出现了许多新问题，本节将介绍这些问题以及相应的解决方案。

4.1 实视图的选择

在设计数据仓库时，一个重要的问题是决定数据仓库要保存那些实视图。具体地说，已知该数据仓库要完成的查询，如何选择一批实视图，使得这些查询总的响应时间和实现图总的维护代价两方面综合起来达到最优。在对查询效率要求较高的系统中，也会面临实视图的选择问题，即如何选择一批实视图，使得系统的查询效率有显著的提高。

选择实视图集合时，首先要解决的一个问题是这些实视图应从何处挑选，即其选择的范围是什么。通常的做法是由给定的查询集合求出它们重复使用的子表达式，由这些子表达式定义的视图作为实视图的候选者，也可将查询本身作为候选的实视图。

决定一个实视图集合是否最优的因素包括[10]：

(a)实视图可利用的存储空间的大小 S；(b)查询 q 的使用频率 f_q；(c)基表 r 的更新频率 f_r；(d)实现查询 q 时访问实视图 v 的代价 $C_q^v(v)$；(e)基表 r 更新时维护实视图 v 的代价 $C_m^r(v)$等。

使用实视图 v 的代价为：$C_{total}(v)=\sum_{q\in Q} f_q C_q^v(v)+\sum_{r\in R} f_r C_m^r(v)$，其中 Q 表示给定查询的集合，R 表示 Q 所涉及到的基表的集合。

实视图选择的任务就是选择视图的集合 M 使得其使用代价 $C_{total}=\sum_{v\in M} C_{total}(v)$达到最小，这样的算法一般都是启发式的[10]。

4.2 实视图的维护

当基表的内容发生变化时，若这些变化影响到在其上定义的实视图，则对这些实视图也要进行相应的更新以保持与基表一致。最原始的方法是一旦更新基表，就重新构造整个实视图，这样做的缺点是如果基表更新频繁，系统的效率将非常低。于是有效地维护实视图的问题就摆在了人们面前。

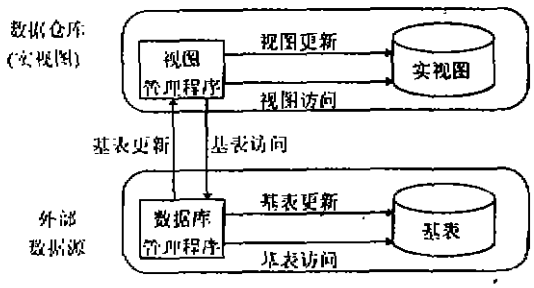

图2 视图维护机制示意图

一般地，实视图维护机制如图2所示[8]。如果有基表上的更新操作 U_r，基表本身并不负责对在其上定义的视图进行更新，而是由数据库管理程序将 U_r 通知给视图管理程序，由视图管理程序负责实视图的更新。在接收到 U_r 后，视图管理程序的反应一般为[7]：

1. 检测 U_r 是否对实视图有影响，若无，这些更新是无关更新，可不予理睬，否则

2. 检测仅利用实视图和 U_r 两方面的信息可否对视图进行维护，若可以，U_r 是可自维护更新，不需访问基表，视图进行自维护，否则

3. 通过访问基表维护实视图。

可以看出，实视图的维护包括三个方面的工作：

·无关更新检测；

·可自维护的更新检测和自维护实施；

·维护实施。

实际上前两个问题很接近，无关更新检测和可自维护更新检测都是对视图状态、数据库状态和更新操作是否满足某些特定条件的判定，实视图自维护的实施也较易实现。因此我们主要讨论上述的两个检测和维护实施问题，为叙述方便，我们将这两类更新简记为 IAU（Irrelevent and Autonomously Computable Updates）。

4.2.1 IAU 检测 与视图更新可翻译性的检测类似，IAU 的检测也分为两种情况，按照[2]中的术语称为语法检测和外延检测，也有人称之为编译时检测和运行时检测[6]，总之前者仅利用数据库模式、视图定义和更新操作三方面的信息，而后者除此之外还需要访问数据库和实视图中的数据。

无关更新的语法检测[1]的主要问题是判断一个含有变量的逻辑条件是否可满足，即是否存在一个或一组值，用它(们)替换条件中的变量可以使得该条件取真值。该条件是由实视图定义中的选择条件加上更新操作中的选择条件形成的，由于二者都用于选择基表中的某些元组，它们联合选出的元组就有着双重身份，首先这些元组被用来形成实视图中的元组，其次是这些元组要被更新；因此结合后的条件可用来判定实视图对基表更新操作的反应。若该条件可能为真，更新操作就可能会影响到实视图，反之，即该条件不成立，那么更新操作便与实视图无关。例如，设实视图 V 的定义条件为 ζ，其基表之一为 R，考虑 R 上的两类更新操作 Insert 和 Delete：

更新操作 U_r	U_r 与 V 无关的充要条件
Insert t into R	$\zeta[t]$[1]为假
Delete from R for ζ_D	$\zeta\&\zeta_D$ 为假

可自维护更新的语法检测是检测可否不访问基表 R 便可在 V 中完成相应的更新。如果 V 仅建立在 R 之上，这总是可以的。否则，即 V 建立在多个基表之上，如果 U_r 是插入操作，则由于 U_r 中缺乏 R 外其它基表的信息使得 V 不能完成相应的更新；如果 U_r 是删除操作，则视 ζ_D 可否在 V 中判别而定。具体地说：

考察 V 中列的集合 A_v 和条件 ζ_D 所涉及到的列的集合 A_d，如果 $A_d \subseteq A_v$，则 ζ_D 可在 V 中进行判断；否则可以考虑利用条件 ζ：设 $A_c = A_d - A_v$，如果 ζ 成立时，ζ_D 的值便不再与 A_c 有关，那么 U_r 仍然是可自维护的更新。

例如 U3，这里 $A_c = \{Sal\}$，$\zeta = Sal > 2000$，如果当 ζ 为真时，无论 Sal 取何值，条件 $\zeta_D = Sal > 1500$ 总成立，因此 U3可以在实视图 ExeEmp 中执行。

IAU 的语法检测由于不访问关系中的数据，所以效率很高；其缺点是可能会将一些 IAU 漏过。以上表中删除操作 U_r 的无关性检测为例，当 $\zeta\&\zeta_D$ 为假时，U_r 一定与 V 无关，而 $\zeta\&\zeta_D$ 不能确定为假（因为其中含有变量），而可能为真时，U_r 未必会影响到 V，这要视基表中被删除的元组是否已被用于构成 V 中的元组而定。这种检测需要访问关系中的数据，被称作外延检测。与语法检测相比，外延检测由于需要访问基表或实视图中的数据，其效率相对要低，但考虑到访问代价一般要比更新代价小，外延检测也有研究的必要。

4.2.2 维护的实施 实视图的维护一般采取渐进式策略[5]，即当基表变化时，不是重新构造实视图，而是根据实视图、基表和 U_r 三方面的信息计算出对实视图的更新操作 U_v，将 U_v 施加在实视图上。

前已述及，数据仓库是实视图的集合，实视图的维护对它而言也就格外重要。由于在数据仓库中实视图和基表相对独立，与普通实视图维护相比，数据仓库中实视图的维护有如下特点：

1. 基表更新和对基表查询的不同步导致实视图更新异常[11]。

2. 数据源中的基表有部分或全部不能被数据仓库访问[5]。这时的实视图维护称为实视图的自维护。

1) $\zeta[t]$是将 t 代入 ζ 所得。

3. 实视图更新时可以访问其他实视图[6]。

4.3 实视图的利用

在一般的数据库系统中增加了实视图以后，一些对基表或普通视图的查询就可以通过实视图来完成，这将有效地提高查询速度，其主要原因是：

·实视图的规模一般要比基表小；

·有些实视图的数据是对基表数据的加工处理，这种加工可被给定的查询利用。

因此利用实视图进行查询优化受到了人们的重视[3]，主要的工作表现在两个方面：

(1)查询改写：在给定的查询中寻找可以由实视图替换的子表达式，然后用实视图替换成为新的查询。

(2)与传统方案相比较：将传统优化方案与利用视图的优化方案相比较，最终求得最佳优化方案。

五、总结

关系模型由于其理论上的严谨而获得了一致的好评，它得到普及推广，根本原因仍在于它解决实际问题的能力。虽然由于视图问题使之在适合 ANSI/SPARC 框架时显得有些欠缺，关系系统的应用并未因之受到太大影响，因而修补这些欠缺的努力也渐渐冷却下来。

实际需求是科学理论研究的原动力，实视图问题正是随着新型数据库应用而出现的，其研究现状是人们正在致力于解决所面临的实际问题，目前的局面是一方面一些特定的问题尚未解决，另一方面人们更期待着通用解决方案的出现。具体地说，实视图研究需要解决的问题可大致分为如下三种：

一、使已有的方案更为通用。许多方案是在简化的模型上，或者是针对特殊的应用提出来的，为适应更广泛的应用，这些方案还有待进一步的扩充，如：

· 大多数无关更新和自维护更新的检测，以及实视图维护策略的研究都是针对 SPJ 视图进行的，而实视图，特别是在数据仓库中，多用于存放汇总数据，定义这类视图的操作便不再仅限于选择、投影和联结这三种，还包括了交、并以及聚集等更为复杂的运算，如何有效地维护这类视图是一个较为迫切的问题。

· 可利用信息对实视图维护的影响，这些可利用信息不仅仅包括基表的定义，还应包括数据库模式定义中更为广泛的内容，如实体完整性、关联完整性等，甚至还应包括数据仓库的模式及其内容。

二、提高系统的效率。

三、实视图的实现。现有的数据库系统是一个成熟而复杂的系统，添加实视图后，在具体的实现方法上有许多问题有待研究，其中表现最为突出的是实视图维护与事务的关系，如：

· 实视图维护是事务的一部分还是与之相互独立？

· 维护发生在事务提交前还是提交后？发生在执行前还是执行后？

· 一个事务中同时发生的多个更新对维护的影响等等，

参 考 文 献

[1] J. A. Blakeley, et al., Updating Derived Relations: Detecting Irrelevent and Autonomously Computable Updates, ACM Trans. on Database Systems, 14(3), 1989

[2] F. Bancihon, et al., Update Semantics of Relational Views, ACM Trans. on Database Systems, 6(4), 1981

[3] S. Chaudhuri, et al., Optimizing Queries with Materialized View, In Proc. ICDE, 1995

[4] U. Dayal., et al., On the Correct Translation of Update Operations on Relational Views, ACM Trans. on Database Systems, 8(3), 1982

[5] A. Gupta et al., Maintenance of Materialized Views: Problems, Techniques, and Applications, IEEE Engineering Bulletins, Special Issue on Materialized Views & Data Warehousing, 18(2), 1995

[6] A. Gupta, et al., Maintaining Views Incrementally, SIGMOD 1993

[7] J. V. Harrison, et al., Incremental View Maintenance, In Proc. of the 5th Australia Database Conference, 1994

[8] N. Huyn, Multiple-View Self-Maintenance in Data Warehousing Environments, In Proc. of the 23rd VLDB Conference, 1997

[9] A. M. Keller, The Role of Semantics in Translating View Updates, Computer, January, 1986

[10] J. Yang, et al., Algorithms for Materialized View Design in Data Warehousing Environment, Same to[8]

[11] Y. Zhuge, et al., View Maintenance in a Warehousing Environment, SIGMOD, San Jose, CA, May, 1995