数据仓库中实体化视图的重计算代价最小化问题的研究

谷 岩 郭 庆
(广州大学 广州510091)

摘 要 数据仓库中实体化视图的重计算问题实际上就是由视图的结构发生变化而引起的。对基本关系的每个可能的Schema模式变化,必须对视图重计算而得到新视图,这种重计算过程是需要付出代价的,为了使视图的重计算代价最小化,不应该对新视图中的所有数据都重新计算一遍,而应该通过一定的算法保留旧视图中的数据,只通过重计算而获取新数据,这样就可以使视图的重计算代价最小化。

关键词 数据仓库,实体化视图,重计算,Schema模式,视图维护

Study on How to Make the Cost of Materialized View Recomputation Be Minimized in Data Warehouse

GU Yan GUO Qing
(Guangzhong University,Guangzhou 510091)

Abstract Materialized view recomputation in data warehouse may result from changes of view's structure. For schema changes of basic relation,view recomputation has to be done. But it will pay out the cost. For making the cost of materialized view recomputation be minimized,we don't need re-computed all data in view. The best way is that the data in old view are reserved and the new data are re-computed by some arithmetic.

Keywords Data warehouses,Materialized view,Recomputation,Schema,View maintenance

1 引言

数据仓库的主要功能是为OLAP提供支持,OLAP在统计查询时将涉及大量的数据,每次获取数据时都要进行数据的投影、连接、汇总等预处理,由于该过程非常耗时,因此OLAP的查询效率非常低,而OLAP又要求其查询被快速地响应。为了解决该矛盾,数据仓库的解决方案就是创建实体化视图(Materialized View)。实体化视图是数据仓库针对OLAP可能的查询对原始数据进行投影、连接、汇总等预处理而建立的,它与数据库的"视图"概念不同之处在于:它不是虚拟的,而是已经过计算的、含有大量数据的、并存储在数据仓库中的表。因此通过数据的预计算,OLAP基本上不再对源数据进行复杂处理,而只需在实体化视图上进行一些简单的计算就可以完成复杂的查询,最终达到提高OLAP查询数据效率的目的。

实现实体化视图是提高系统响应时间的一个关键技术和有效的解决方案,但在具体的实现过程中面临着新的问题,即实体化视图的一致性维护问题。在一般情况下,实体化视图的结构和数据具有相对的稳定性,但源数据是相对不稳定的,其结构和数据是不断发生变化的,当这些源数据发生变化时,如果其变化不能及时传播到实体化视图中,以保持实体化视图与源数据的一致性,就会降低实体化视图中数据的新鲜度,从而影响OLAP查询结果的真实性和有效性。关于视图中数据的维护问题,已提出许多解决方案,最著名的方法有增量视图维护方法和版本链控制方法,但这些方法都未涉及视图结构的维护问题。为此,本文将对实体化视图由于结构的变化而进行重计算的代价最小化问题进行研究,并提出相应的实现算法。

2 实体化视图的生成及生成机制

2.1 生成实体化视图的语法结构

从广义上讲,数据仓库中的数据都可以看作是以实体化视图的方式来存储的,视图在数据仓库中是作为基表存在的。根据SPJ(Selection-Projection-Join)原理,设 $A_1,A_2,\cdots,A_n$ 和 $B_1,B_2,\cdots,B_p$ 是关系 $R_1,R_2,\cdots,R_m$ 中的属性,生成视图的语法结构如下:

```
Create View V As
Select A1,A2,…,An
From R1,R2,…,Rm
Where C1 and C2… and Ck
```

在这里V是生成的实体化视图,$A_1,A_2,\cdots,A_n$ 是被列出的属性,$B_1,B_2,\cdots,B_p$ 是不被列出的属性,$C_i(i\in[1,k])$ 是一个选择条件或一个连接条件。

2.2 实体化视图的生成机制

数据仓库能够集成来自不同数据源和不同数据结构的数据。由于源数据的这种异构性,实体化视图是不能从源数据直接通过计算而生成,它必须采用多级视图生成机制来实现。多级视图生成机制的基本原理是:通过局部视图和中间视图作为过渡性视图,有步骤地根据源数据导出实体化视图。具体步骤是:(1)对源数据进行翻译或结构转换,形成局部视图,局部视图具有共同的数据结构形式;(2)对局部视图消除语义上的冲突,进行连接、投影,形成中间视图,并作为建立实体化视图的数据基础;(3)根据系统的实际主题、维度及粒度模型对中间视图的数据进行概括汇总,形成实体化视图。图1表示了实体化视图的生成过程。

谷 岩 副教授,硕士,主要研究方向为管理信息系统、数据库技术及Internet/Intranet的研究与应用开发。

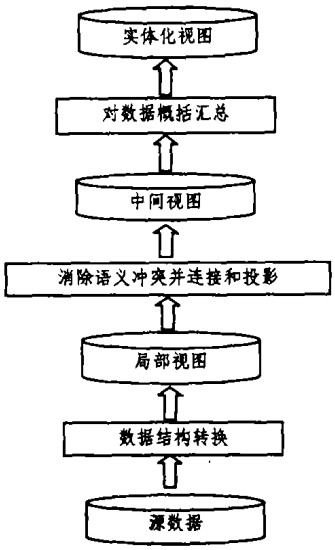

图1 实体化视图的生成机制

3 一个需要使用的数据库实例

本文以"学生选课"数据库作为实例，以方便后面的讨论。该数据库包括5个表，即学生基本信息表(Student)、课程基本信息表(Subject)、学生选课关系表(Enrolment)、学位基本信息表(Degree)及学院或系基本信息表(School)，有下划线的属性是关系的关键字，这些表的 Schema 模式如图2所示。

Student (Stud_ID, Stud_fname, Stud_Lname, Stud_Address, Degree_Code)
Subject(Subj_Code, Subj_Name, Subj_Units)
Enrolment(Stud_ID, Subj_Code, Subj_Grade)
Degree (Degree_Code, Degree_Name, Degree_Abrev, School_Code)
School(School_Code, School_Name, School_Location)

图2 "学生选课"数据库中表结构

4 实体化视图基本结构的重计算

实体化视图基本结构的重计算问题实际上就是由视图的结构发生变化而引起的。对基本关系的每个可能的 Schema 模式变化，我们必须重新生成新视图，然后代替旧视图，该过程我们称之为视图的重计算(recomputation)。这种重计算过程是需要付出代价的，但我们希望重计算所付出的代价最小化。经过研究我们发现，通过重计算所得到的新视图，一部分数据来源于旧视图，另一部分不包含在旧视图中的新数据则是通过重计算而得到的。因此，为了使视图重计算的代价最小化，我们不应该对新视图中的所有数据都重新计算一遍，而应该通过一定的算法保留旧视图中的数据，只通过重计算而获取新数据，这样就可以使重计算的代价最小化。

基于基本关系的实体化视图基本结构的变化主要包括：① 增加、删除或修改属性；② 删除关系。本文将围绕这两类变化进行讨论和研究。

4.1 视图中增加新属性

当一个新的属性被添加到源关系时，很自然就要在与此相关的视图中增加新属性，新属性对实体化视图的影响及代价将通过下面的实例来说明。

实例 根据上节给出的数据库实例，我们需要产生一个视图，该视图将提供每个学生的基本信息、选修的课程及成绩。产生该视图的一般定义语句如下：

```
Create View Info_Stud As
Select   ST.Stud_ID,  ST.Stud_Fname,  ST.Stud_Lname,
         ST.Stud_Adress,SU.Subj_Name,SU.Subj_Grade
From Student ST,Subject SU,Enrolment E
Where  ST.Stud_ID = E.Stud_ID and E.Subj_Code =
       SU.Subj_Code;
```

现在我们需要在 Student 中增加一个新属性 Stud_E-mail，并且希望在 query 中列出，如果仍用一般的视图生成方法，那么就必须通过重计算来获取新视图，生成语句如下：

```
Create View Info_Stud As
Select   ST.Stud_ID,  ST.Stud_Fname,  ST.Stud_Lname,
         ST.Stud_Adress, ST.Stud_Email, SU.Subj_Name,
         SU.Subj_Grade
From Student ST,Subject SU,Enrolment E
Where  ST.Stud_ID = E.Stud_ID and E.Subj_Code =
       SU.Subj_Code;
```

为了使重计算的代价最小化，我们将利用旧视图，并采用与要增加属性的关系进行自然连接的方式生成新视图。生成语句如下：

```
Create View V' As
Select V.Stud_ID, V.Stud_Fname, V.Stud_Lname, V.Stud_
       Adress,ST.Stud_Email,V.Subj_Name,V.Subj_Grade
From Info_Stud V,Student ST
Where ST.Stud_ID= V.Stud_ID;
Drop Info_Stud;
Rename V' As Info_Stud;
```

从上述实例我们可以看到，如果采用重新生成新视图的方式，必然要对3个关系进行连接计算；而如果采用上述算法，视图无论来自于多少个关系，它只需对旧视图和要增加属性的关系这两个表进行连接操作。因此我们可以说采用这种方式对视图重计算的代价应该是最小化的。生成新视图 V 的算法如下(假使属性 A_i 将被添加到 R_1 关系中，A_j 是被列出的并处于 R_1 关系中的一个属性)：

```
Create View V' As
(Select V.A1,V.A2,…,V.An,R1.Ai
 From V,R1
 Where V.Aj=R1.Aj);
Drop View V;
Rename V' As V;
```

4.2 视图中删除一个属性

从源关系中删除一个属性而引起视图中一个属性被删除，我们可以根据以下三种情形来讨论：①属性涉及 SELECT 子句；②属性涉及 WHERE 子句；③同时涉及 SELECT 和 WHERE 子句。

(1)涉及 SELECT 子句

从源关系中删除一个属性，如果仅仅影响到视图的 SELECT 子句，我们可以在旧视图的基础上将该属性删除，然后生成一个新视图。

实例 假使我们已生成一个视图，该视图将提供所有选择课程代码为 cp1200或 cp1500并成绩高于60分的所有学生的基本信息以及课程的基本信息，视图的生成语句如下：

```
Create View Info_Stud_Subj As
Select   ST.Stud_ID,  ST.Stud_Fname,  ST.Stud_Lname,
         ST.Stud_Adress,SU.Subj_Name,SU.Subj_Units
From Student ST,Subject SU,Enrolment E
Where (E.Subj_Code= 'cp1200' or E.Subj_Code= 'cp1500')
      and E.Subj_Grade)=60 and ST.Stud_ID=E. Stud_ID
      and SU.Subj_Code=E.Subj_Code;
```

现在我们假使从关系 Student 中删除 Stud_Adress 属性，由于它已存在于旧视图 Info_Stud_Subj 中，因此我们可以用下列方法对视图进行重计算：

```
Create View New_View As
Select Stud_ID,Stud_Fname,Stud_Lname,Subj_Name,Subj_
       Units
From Info_Stud_Subj;
Drop View Info_Stud_Subj;
Rename New_View As Info_Stud_Subj;
```

从上述方法我们可以看到，当从旧视图中删除一个属性时，所生成的新视图不要根据源关系重计算，而是根据旧视图来重计算新视图。由于新视图是从实体化视图中读数据，它所

付出的代价应该是最小的，因为实体化视图一定是小于源关系的。重计算视图的算法如下(假使 A_1是从旧视图 V 中要删除的属性)：

```
Create View V' As
(Select A2,A3,…,An
 From V);
Drop View V;
Rename V' As V;
```

(2)仅涉及 WHERE 子句

如果删除的属性将影响到 WHERE 子句，实际上就是将涉及该属性的条件去掉，那么在生成新视图时，除了保留旧视图的数据外，还要将被上述条件已抛弃的元组找回并插入到新视图中。

实例在上节所生成的 Info_Stud_Subj 视图中，如果要删除 Enrolment 关系的 Subj_Grade 属性，我们可以利用下面的方法来生成新视图：

```
Create View New_View As
(Select *
 From Info_Stud_Subj)
   Union
( Select  ST.Stud_ID, ST.Stud_Fname, ST.Stud_Lname,
          ST.Stud_Adress,SU.Subj_Name,SU.Subj_Units
      From Student ST,Subject SU,Enrolment E
      Where (E.Subj_Code = 'cp1200' or E.Subj_Code =
             'cp1500') and
             Not(E.Subj_Grade>=60)and ST.Stud_ID=
             E. Stud_ID and
             SU.Subj_Code=E. Subj_Code);
```

从上述生成过程我们可以看到，新视图的数据实际上分成两部分：第1部分是原视图的数据，它不需要重新计算；第2部分是新数据，它需要重新计算。这种方法所付出的代价应该是最小的，因为它省去了重新计算已经存在于实体化视图中的元组的时间。在此情况下重计算视图的算法如下(假使删除的属性包含在条件 C1中)：

```
Create View V'As
(Select *
 From V)
   Union
(Select A1,A2,…,An
 From R1,R2,…,Rm
 Where (not C1)and C2 and … Ck);
```

(3)同时涉及 SELECT 和 WHERE 子句

如果删除的属性将同时影响到 SELECT 和 WHERE 子句，那么生成新视图的过程被分成两部分：第1部分就是从旧视图中删除属性；第2部分仍然是找回被要删除的条件已抛弃的元组并插入到新视图中。

实例 如果在生成新视图 Info_Stud_Subj_1时，query 要列出的属性包括：Stud_ID，Stud_Fname，Stud_Lname，Stud_Adress，Subj_Name，Subj_Units，Subj_Grade，其生成语句如下：

```
Create View Info_Stud_Subj_1 As
Select  ST.Stud_ID, ST.Stud_Fname, ST.Stud_Lname,
        ST.Stud_Adress,SU.Subj_Name,
SU.Subj_Units,E.Subj_Grade
From Student ST,Subject SU,Enrolment E
Where (E.Subj_Code= 'cp1200' or E.Subj_Code= 'cp1500')
      and E.Subj_Grade)=60 and ST.Stud_ID=E. Stud_ID
      and SU.Subj_Code=E.Subj_Code;
```

如果我们要从 Enrolment 关系中删除 Subj_Grade 属性，那么生成新视图的方法如下：

```
Create View New_View As
( Select  ST.Stud_ID, ST.Stud_Fname, ST.Stud_Lname,
          ST.Stud_Adress,SU.Subj_Name,SU.Subj_Units
From Info_Stud_Subj_1)
Union
( Select  ST.Stud_ID, ST.Stud_Fname, ST.Stud_Lname,
          ST.Stud_Adress,SU.Subj_Name,SU.Subj_Units
From Student ST,Subject SU,Enrolment E
Where (E.Subj_Code = 'cp1200' or E.Subj_Code = 'cp1500')
      and Not (E.Subj_Grade) = 60) and ST.Stud_ID =
      E.Stud_ID and SU.Subj_Code=E.Subj_Code);
```

第1部分是从原视图的数据中重计算，它所付出的代价应该是最小的，因为它重计算的数据存在于实体视图中而不是源关系中；而第2部分是新数据，它需要重新计算。此情况下重计算视图的算法如下(假使 A_1是要从旧视图中删除的属性，C_1是包含了 A_1的条件)：

```
Create View V' As
(Select A2,A3,…,An
 From V)
   Union
(Select A2,A3,…,An
 From R1,R2,…,Rm
 Where (not C1)and C2 and … Ck);
```

4.3 删除一个与视图有关的基本关系

在这节我们将讨论删除一个与视图有关的源关系。如果受影响的视图只是来源于一个关系，那么只需删除这个关系后再直接删除这个视图，否则，如果被删除的这个关系出现在 SELECT 和/或 WHERE 子句，那么就要相应地调整新视图。从视图疑问中删除一个关系，实际上就是将与对应的连接条件不匹配的新元组插入到视图中(假使被删除的关系在调整时间内仍然有效)。

实例 视图 Join_View 是由下面的语句定义的：

```
Create View Join_view
As Select R.A,S.C
From R,S,T
Where R.A=S.B and S.C=T.C;
```

我们假使关系 T 已经从数据源中删除，那么这将导致连接条件 $S.C=T.C$ 被删除。为了调整的目的，我们应该将与连接条件 $R.A=S.B$ 匹配但与条件 $S.C=T.C$ 不匹配的元组插入到新视图中。因此为了删除关系 T 而重计算视图的语句是：

```
Create View New_View As
(Select R.A,S.C
 From Join_View)
   Union
(Select R.A,S.C
 From R,S
 Where not(S.C=T.C)and (R.A=S.B));
```

本视图疑问的第1个子疑问包含在视图 V 中，因此它不需要存取源数据，只有第2个子疑问要从基本关系中重计算，并且所得到的元组将被插入到实体化视图中。虽然调整和重计算需要存取源关系，但调整过程中将节省已经存在于视图 V 中的元组插入到新视图的时间。

结束语 本文所提出的使实体化视图由于结构的变化而进行重计算的代价最小化的方法应该对数据仓库的维护具有十分重要的意义，因为它不仅能对实体化视图的结构发生变化进行及时地更新和维护，同时可以做到视图维护的代价最小、维护的效率最高。当然在具体的维护过程中，所面临的问题可能会比文中所提出的实例要复杂得多，所要考虑的因素也要多得多，但其维护的基本思路不会变，那就是：要充分利用旧视图，避免旧视图中的数据重计算。

参 考 文 献

1 陈京民，等. 数据仓库与数据挖掘技术[M]. 北京：电子工业出版社，2002

2 王珊，等. 数据仓库技术与联机分析处理[M]. 北京：科学出版社，1998

3 Monhania M. Avoiding re-computation views adaptation in data warehouse[J]. In：Proc. of 8th intl. database workshop，Springer，Hong Kong，1997. 151～165

4 Gupta H. Selection of views to materialized in a data Warehouse [J]. In：Proc. of the Intl. conf. on database theory，Delphi，Greece，1997. 98～112

5 Zhuge Yue，Garcia-Molina H. Consistency Algorithms for Multi-Source Warehouse View Maintenance. http://www-db. stanford. edu/warehouse.

6 Agrawal D，Abbadi E. Efficient. view maintenance at data warehouse[J]. In：Proc. of SIGMOD，1997. 417～427