View Maintenance Using Conditional Tables*

Hua Shu

Institut für Informatik
Universität Hannover
Lange Laube 22, D-30159 Hannover, Germany
hs@informatik.uni-hannover.de

Abstract. This paper presents a new approach to maintaining materialized views without accessing the underlying base relations. Views can be made self-maintainable using additional data together with the views. For instance, one can replicate auxiliary views of the base relations at the site where the views are materialized to ensure self-maintenance of the views. However, the previous approaches often lead to the replication of the entire base relations, which is not acceptable from the data protection point of view. We propose to represent the base data using tables with *variables* and to materialize auxiliary views of such tables in form of *conditional tables*. Modeling updates of the base data as changes of the assignment to the variables, we can compute the updated views by evaluating the conditional tables with respect to the new assignment of the variables. Our approach avoids the replication of the base data and allows active self-maintenance of views triggered by identified updates of the base data.

1 Introduction

Views are versions of data that are restructured and possibly restricted images of a database. Materialized views are physical copies of views that are stored and maintained. The *view maintenance problem* is about how to efficiently update a view that is materialized in response to updates of the base relations. Suppose that there is a view defined by a relational algebra expression q over a database schema. A *state* of the view with respect to a set I of base relations is denoted by $q(I)$. Consider an update operation μ against I. Let I' denote the updated state of the base relations. The simplest way to update the view is to compute $q(I')$ from scratch. Sometimes it is more efficient to incrementally maintain the view, i.e. to compute only the changes in the view [6].

We are particularly concerned with the situations where the access to the base data may be slow, expensive or even periodically unavailable, and it is desirable to be able to incrementally maintain views without additional queries over the base data. Views that can be maintained only based on information about the

* This work was supported by Swedish Research Council for Engineering Sciences (TFR) under grant 282-95-966.

www.manaraa.com

views and the updates against the base data, without additional queries over the base data, are said to be *self-maintainable* [4].

Self-maintainability of views has been recognized as one of the main optimization problems in, for instance, data warehouses. At a sufficiently abstract level, a data warehouse can be seen as a collection of materialized views over base data residing at external information sources. An important task of warehouse management is to perform materialized view maintenance [10, 12]. In order to integrate the change of the base data into the warehouse, it may be needed to fetch additional data from the external sources. Issuing such queries to the sources may lead to a processing delay. The queries can be expensive or may not be permitted at all for security reasons. Thus it is often required to minimize the additional queries to the external sources.

Previous studies have mainly been focus on identifying the class of self-maintainable views. It has been shown that only a very restricted subclass of SPJ views is self-maintainable [11, 3, 5, 6, 10, 12]. Views involving joins, for instance, are generally not self-maintainable in response to insertions into a component relation, and are self-maintainable in response to deletions and modifications only under certain conditions [3, 4].

Suppose that views are stored at sites different from where the base relations are stored. A site where base relations are stored is called base-relation site. A site where views are stored is called view site. A simple way to achieve self-maintenance of views is to replicate (a subset of) the base relations at view site. However, the cost of replicating large base relations may become prohibitive. Moreover, the replication may not be acceptable for applications where the very purpose of defining views is *data protection*, i.e. to limit the access to the entire base relations.

Another approach is to materialize *auxiliary views* of the base relations at view sites [7, 10]. In [7], views are made self-maintainable by storing, at the view sites, the results of pushing down selections and projections to the base relations. The cost of storing the results is usually lower than the cost of replicating the base relations in their entirety. For example, consider a view defined by $\pi_{r_1,r_3,s_1,s_2}(\sigma_{E_1}R \bowtie_{r_2=s_1} \sigma_{E_2}S)$, where r_1, r_2, r_3 are some attributes of R, s_1, s_2 are some attributes of S and E_1, E_2 are selection conditions. The idea is to materialize, at view site, the auxiliary views $\pi_{r_1,r_2,r_3}\sigma_{E_1}R$ and $\pi_{s_1,s_2}\sigma_{E_2}S$ of the base relations R and S. These views are smaller than the base relations. Based on the materialized, auxiliary views, the view $\pi_{r_1,r_3,s_1,s_2}(\sigma_{E_1}R \bowtie_{r_2=s_1} \sigma_{E_2}S)$ can be computed without access to the base relations. However, when no selections and projections can be pushed down to the base relations, this approach degenerates into the replication of the entire base relations.

The updates of the base relations that are relevant to a view are sometimes known to be restricted to certain part of the base relations. For instance, in a personnel database, it may be that only the records of the temporarily employed or guests can be removed from the database, and only the fields of salaries are frequently modified. It is then desirable to have some *active* mechanism of view self-maintenance triggered by such *expected* updates.

www.manaraa.com

This paper explores an alternative approach to providing auxiliary data which enable the self-maintenance of materialized views. The general idea of our approach is to represent the base data using tables with *variables* and to materialize auxiliary views of such tables in form of *conditional tables* of Imielinski and Lipski [8]. Modeling updates of the base data in terms of changes of the assignment of the variables, we can compute the updated views by evaluating the conditional tables with respect to the new assignment of the variables. It can be proven that all views defined by relational algebra expressions taken together with the materialized conditional tables are self-maintainable (based on information about the updates of the base data sent to the view site). This approach avoids the replication of the entire base relations and allows active self-maintenance of views triggered by identified updates of the base relations.

This paper is organized as follows. First, we present the notion of conditional tables introduced in [8] with extensions based on the three-valued logic. Then we illustrate our approach using an example. In Section 4, we describe the ideas of self-maintaining views with respect to deletions against underlying base data. In Section 5, we extend the approach to cover the cases with respect to deletions *and* insertions. In Section 6, we discuss how to handle modifications as atomic operations. Finally, we conclude the paper with a summary and some discussion.

2 Conditional Tables

A table is a relation with constants and variables. A conditional table is an extension of a table with one more column containing logical formulas attached with the tuples of the relation. Table 1 shows an example of conditional tables, where x, y, u, v, t and s are variables. The logical formulas are represented under a column labeled *con*. *con* can be seen as a special attribute.

Definition 1. A *condition* is an expression built up by means of Boolean connectives $\neg, \wedge, \vee$ and $\Rightarrow$ from atoms **true**, **false**, and equality atoms of the forms $x = y$ and $x = c$, where x and y are variables and c is a constant. $x \neq y$ $(x \neq c)$ is the abbreviation for $\neg(x = y)$ (resp. $\neg(x = c)$).

Definition 2. A *conditional table* (or *c-table* for short) is a pair $T_c = (T, \phi)$, where

- T is a table,
- ϕ is a mapping over T that associates a condition $\phi(t)$ with each tuple t of T. $\phi(t)$ is called a local condition. A *c-tuple* is a tuple t of T together with local condition $\phi(t)$, denoted $(t, \phi(t))$.

When dependency constraints are concerned, an additional kind of conditions, called *global conditions*, needs to be introduced, as proposed in [2]. For simplicity, we do not consider dependency constraints and therefore ignore the global conditions of conditional tables.

www.manaraa.com

Table 1. A conditional table (c-table)

A	C	*con*
t	c	$x = u$
s	c	$y = u$
t	c'	$x = v$
s	c'	$y = v$

To interpret information in a c-table is to map it to relation instances by assigning values to the involved variables. Previously, the interpretation has been done based on two-valued logic. In this section, we define a three-valued interpretation of c-tables.

Let $\mathbf{V}$ be a finite set of *variables* and each variable x in $\mathbf{V}$ have an associated domain, denoted $D(x)$. A *valuation* over $\mathbf{V}$ is a mapping $\mathbf{v}$ which maps each variable in $\mathbf{V}$ to a value in the associated domain. Such a valuation is expressed in form of $\{x_1/a_1, x_2/a_2, ..., x_n/a_n\}$, where $x_1, ..., x_n$ is a listing of $\mathbf{V}$ and $a_i = \mathbf{v}(x_i) \in D(x_i)$ for each $i \in (1, n)$.

We assume that there is a special value η, called *non-existing value*, included in the domain of each variable. Intuitively, the assignment of the non-existing value to variable x means that the value of x simply does not exist. It is introduced to model the updates (deletions, insertions and modifications). It is also assumed that the domain of each attribute does not contain the non-existing value.

The evaluation of conditions is based on the *strong Kleene logic* [9]. We denote the evaluation of a condition C with respect to a valuation $\mathbf{v}$ by $V^{\mathbf{v}}(C)$. With respect to valuation $\mathbf{v}$, C can be satisfied (denoted $V^{\mathbf{v}}(C) = T$), falsified (denoted $V^{\mathbf{v}}(C) = F$) or *undefined* (denoted $V^{\mathbf{v}}(C) = U$). The evaluation of a condition is defined recursively as follows:

1. **true** is always evaluated T and **false** always F.
2. For an equality atom $x = y$, $V^{\mathbf{v}}(x = y) = U$ if $\mathbf{v}(x) = \eta$ or $\mathbf{v}(y) = \eta$. $V^{\mathbf{v}}(x = y) = T$ if $\mathbf{v}(x) \neq \eta$, $\mathbf{v}(y) \neq \eta$ and $\mathbf{v}(x) = \mathbf{v}(y)$. Otherwise, $V^{\mathbf{v}}(x = y) = F$.
3. $V^{\mathbf{v}}(\neg p) = \neg V^{\mathbf{v}}(p)$, $V^{\mathbf{v}}(p \wedge q) = V^{\mathbf{v}}(p) \wedge V^{\mathbf{v}}(q)$ and $V^{\mathbf{v}}(p \vee q) = V^{\mathbf{v}}(p) \vee V^{\mathbf{v}}(q)$, where the connectives $\neg, \wedge$ and $\vee$ on the right-hand sides of the above equations are defined by the well-known truth tables of the strong Kleene logic shown in Table 2.

We say that two conditions C_1 and C_2 are *equivalent* iff $V^{\mathbf{v}}(C_1) = V^{\mathbf{v}}(C_2)$ for any valuation $\mathbf{v}$. In c-tables, conditions equivalent to **true** are simply omitted. If all local conditions in a c-table are **true**, then the *con* column can be omitted.

Now we define the result of evaluating c-table $T_c = (T, \phi)$ with respect to valuation $\mathbf{v}$, denoted $\mathbf{v}(T_c)$, as follows:

$$\mathbf{v}(T_c) = \{\mathbf{v}(t) \mid t \in T, V^{\mathbf{v}}(\phi(t)) = T \text{ and } \mathbf{v}(x) \neq \eta \text{ for any } x \in t\}, \quad (1)$$

www.manaraa.com

Table 2. The truth tables of the connectives $\neg, \wedge$ and $\vee$

p	$\neg p$
T	F
F	T
U	U

$\wedge$	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

$\vee$	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

where $\mathbf{v}(t)$ is the result of evaluating each value in tuple t with respect to $\mathbf{v}$. So $\mathbf{v}(T_c)$ contains all such $\mathbf{v}(t)$ that $t \in T$, $\mathbf{v}$ satisfies the local condition $\phi(t)$ and $\mathbf{v}(t)$ does not contain the non-existing value.

We call $\mathbf{v}(t)$ the *instance* of the c-tuple $(t, \phi(t))$ with respect to $\mathbf{v}$ and c-tuple $(t, \phi(t))$ the *abstraction* of $\mathbf{v}(t)$. $\mathbf{v}(T_c)$ is called the *instance* of T_c with respect to $\mathbf{v}$.

Given a database I_c with a number of c-tables, the instance of I_c with respect to $\mathbf{v}$ is defined as: $\mathbf{v}(I_c) = \{\mathbf{v}(T_c) \mid T_c \in I_c\}$.

For example, the instance of the c-table shown in Figure 1 with respect to the valuation $\{x/a, y/a, u/b, v/a, t/d, s/d\}$ is the relation with one tuple (d, c'). Tuple (d, c') is the instance of c-tuples $(t, c', x = v)$ and $(s, c', y = v)$. In other words, both c-tuples $(t, c', x = v)$ and $(s, c', y = v)$ are the abstractions of tuple (d, c').

The set of relation instances represented by c-table $T_c = (T, \phi)$ is defined as follows:

$$rep(T_c) = \{\mathbf{v}(T_c) \mid \text{for any valuation } \mathbf{v}\}. \tag{2}$$

For a database $I_c = (T_1, ..., T_n)$, $rep(I_c) = rep(T_1) \times ... \times rep(T_n)$.

For two c-tables T_1 and T_2, they are said to be *rep-equivalent* if $rep(T_1) = rep(T_2)$. Given a c-table T_c, if we (a) replace each of the conditions in T_c by an equivalent one, (b) delete all $t \in T_c$ such that $\phi(t)$ is equivalent to **false**, and (c) replace some $t_1, ..., t_k \in T_c$ such that $t_1[X] = ... = t_k[X]$, where X is the set of attributes in T_c, by a single tuple t such that $t[X] = t_1[X]$ and $\phi(t) = \vee_{i=1}^{k} \phi(t_i)$, then the resulting c-table will be rep-equivalent to T_c [8]. If the resulting c-table does not contain different c-tuples agreeing on all the attributes of T_c, then it is said to be *normalized* [8]. The result of normalizing a c-table T_c is denoted by $(T_c)^0$. Basically, the normalization process eliminates redundant c-tuples and unite c-tuples without loss of useful information.

Suppose that c-tables T and W are given, which are defined on sets X and Z of attributes, respectively. The definitions of some relational operators on c-tables are as follows (see [8] for the definitions of the other operators):

- The *projection* of T on a set Y of attributes ($Y \subseteq X$) is defined by

$$\pi_Y(T) = \{t[Y \cup \{con\}] \mid t \in T\}^0.$$

- The *natural join* of T and W is defined by

$$T \bowtie W = \{t \bowtie w \mid t \in T \wedge w \in W\}^0,$$

www.manaraa.com

where $t \bowtie w$ is the c-tuple on $X \cup Z$ such that

$$(t \bowtie w)(A) = \begin{cases} t(A) & \text{if } A \in X \\ w(A) & \text{if } A \in Z - X \end{cases}$$

$$(t \bowtie w)(con) = t(con) \wedge w(con) \wedge \bigwedge_{A \in X \cap Z} (t(A) = w(A)).$$

– Given selection condition E, the *selection* of T based on E is defined by

$$\sigma_E(T) = \{\sigma_E(t) \mid t \in T\}^0,$$

where $\sigma_E(t)$ is the c-tuple on X with

$$\sigma_E(t)[X] = t[X],$$

$$\sigma_E(t)(con) = t(con) \wedge E(t).$$

$E(t)$ is the result of substituting $t(A)$ for A in E, for every $A \in X$.

The above definitions of the operations on a single c-table generalize straightforward to the definitions of the operations on a set of c-tables.

Let $q(I_c)$ be a view of I_c defined by relational algebra expression q. It has been proven (Theorem 7.1 in [8]) that

$$\mathbf{v}(q(I_c)) = q(\mathbf{v}(I_c)) \tag{3}$$

holds for any valuation $\mathbf{v}$ of the variables in I_c. Based on (3), it can be proved that the answers to all relational queries on a database of c-tables can be represented using c-tables [8]. Note that although the non-existing value is not considered in [8], the above result still applies because it does not affect the definitions of the relational operators.

Let $\sqsubset$ be a binary relation on the set $\{T, F, U\}$ satisfying

$$U \sqsubset T, U \sqsubset F. \tag{4}$$

$\sqsubseteq$ stands for "$\sqsubset$ or $=$". The ordering $\sqsubseteq$ is reflexive and transitive. The structure $\langle\{T, F, U\}, \sqsubseteq\rangle$ is called the *approximation lattice* [1]. We extend the relation $\sqsubseteq$ to define an ordering over the domain of a variable. For any variable x, $\sqsubseteq$ is a binary relation over $D(x)$ such that for any value a,

$$\eta \sqsubseteq a. \tag{5}$$

$\eta \sqsubset a$ stands for "$\eta \sqsubseteq a$ and $\eta \neq a$". Further, we extend the relation $\sqsubseteq$ to define the *degree-of-definedness* ordering between valuations. For any valuations $\mathbf{v}$ and $\mathbf{v}'$,

$$\mathbf{v} \sqsubseteq \mathbf{v}' \text{ iff } \mathbf{v}(x) \sqsubseteq \mathbf{v}'(x) \text{ for each variable } x. \tag{6}$$

$\mathbf{v} \sqsubset \mathbf{v}'$ means that $\mathbf{v} \sqsubseteq \mathbf{v}'$ and $\mathbf{v} \neq \mathbf{v}'$. $\mathbf{v} \sqsubseteq \mathbf{v}'$ implies that whenever $\mathbf{v}'$ and $\mathbf{v}$ assign different values to the same variable, it must be that $\mathbf{v}'$ assigns η to the variable.

www.manaraa.com

A propositional language is said to be *persistent* if for any valuations $\mathbf{v}$ and $\mathbf{v}'$, and formula C in the language, if $\mathbf{v} \sqsubseteq \mathbf{v}'$, then $V^{\mathbf{v}}(C) \sqsubseteq V^{\mathbf{v}'}(C)$. The strong Kleene logic is the strongest extension of the classical two-valued logic satisfying the persistence condition [9]. Thus, for any valuations $\mathbf{v}$ and $\mathbf{v}'$, and condition C, if $\mathbf{v} \sqsubseteq \mathbf{v}'$, then $V^{\mathbf{v}}(C) \sqsubseteq V^{\mathbf{v}'}(C)$. The following proposition follows directly from the property.

Proposition 3. *For any valuations* $\mathbf{v}$ *and* $\mathbf{v}'$*, and c-table* T_c*, if* $\mathbf{v} \sqsubseteq \mathbf{v}'$*, then* $\mathbf{v}(T_c) \subseteq \mathbf{v}'(T_c)$.

3 An Illustrative Example

In this section, we illustrate our approach by example. We discuss only self-maintenance of views in response to deletions and insertions against the base relations. Consider database (T_1, T_2) at base-relation site and materialized view $\pi_B(T_1 \bowtie_{B=C} T_2)$ at view site, as shown in Table 3.

To make the view self-maintainable with respect to the deletions against the base relations, we carry out the following steps:

1. At the base-relation site, we generate tables by replacing the values of certain attributes in the base relations using variables, namely those attributes that participate in the definition of the view. For each base relation, it is only necessary to identify the values of one such a attribute. The result of this process, (T_1^v, T_2^v), is called a *D-version* of database (T_1, T_2) and shown in Table 4. The values of attribute B, for instance, are identified by three variables y_1, y_2 and y_3, respectively. Let the values of the variables with respect to T_1 and T_2 be kept in a valuation $\mathbf{v} = \{y_1/b, y_2/b, y_3/e, z_1/a, z_2/e\}$.

2. At the view site, we store an additional relation called the *conditioned version* of the materialized view (or the *conditioned view* for short), shown in Table 5. The conditioned view is a *subset* of the C-table $\pi_B(T_1^v \bowtie_{B=C} T_2^v)$ shown in Table 6. The c-tables are normalized. The relation between the materialized view and the conditioned view is that the conditioned view *contains and only contains* the *abstractions* of all the tuples in the materialized view; the tuples in the materialized view are *instances* of the c-tuples in the conditioned view with respect to the valuation $\mathbf{v}$. For instance, the tuple (e) in the materialized view is the instance of c-tuple $(y_3, (y_3 = z_1) \vee (y_3 = z_2))$ with respect to the valuation $\mathbf{v}$.

Note that if we store the D-version of the database and the valuation $\mathbf{v}$, the database can be computed by $T_1 = \mathbf{v}(T_1^v)$ and $T_2 = \mathbf{v}(T_2^v)$. Similarly, if we store the conditioned view and the valuation $\mathbf{v}$, the current state of the view can be computed by evaluating the conditioned view using the valuation $\mathbf{v}$. Thus the new tables at the base-relation and the view sites enable us to compute the base relations and the view whenever needed.

Now let us see how the materialized view can be maintained with respect to deletions against the base relations. Suppose that the tuple (d, e) in the base

www.manaraa.com

relation T_1 is deleted. The purpose of introducing variables to the base relations is to make it possible to model each deletion operation against the base relations as a change of the valuation of the variables. The deletion of tuple (d, e) against T_1 can be captured as the change of the value for variable y_3 in T_1^v to the non-existing value. Thus a change of the valuation of the variables y_3 to the non-existing value is made. The new valuation is $\mathbf{v}_1 = \{y_1/b, y_2/b, y_3/\eta, z_1/a, z_2/e\}$.

Suppose that we send the information about the update against the base relations to the view site by sending the new valuation (or the difference between the old and the new ones if the old valuation is already stored at the view site). Based on the relation between the materialized view and the conditioned view, it is easy to see that the tuples to be deleted from the materialized view can be obtained by evaluating the conditioned view with respect to the new valuation $\mathbf{v}_1$. Since y_3 is assigned the non-existing value, the condition $(y_3 = z_1) \vee (y_3 = z_2)$ can no longer be satisfied. The tuple (e) should be deleted from the materialized view. The evaluation can be done without additional queries over the base relations. Correspondingly, the c-tuple $(y_3, (y_3 = z_1) \vee (y_3 = z_2))$ is deleted from the conditioned view. This way, the view taken together with the conditioned view is self-maintained with respect to the deletion.

Table 3. Base data and view

T_1

A	B
a	b
c	b
d	e

T_2

C	D
a	d
e	a

Base Relations

$\pi_B(T_1 \bowtie_{B=C} T_2)$

B
e

View

Table 4. D-version

T_1^v

A	B
a	y_1
c	y_2
d	y_3

T_2^v

C	D
z_1	d
z_2	a

Table 5. Conditioned view

B	*con*
y_3	$(y_3 = z_1) \vee (y_3 = z_2)$

Table 6. $\pi_B(T_1^v \bowtie_{B=C} T_2^v)$

B	*con*
y_1	$(y_1 = z_1) \vee (y_1 = z_2)$
y_2	$(y_2 = z_1) \vee (y_2 = z_2)$
y_3	$(y_3 = z_1) \vee (y_3 = z_2)$

Now we consider how the view can be made self-maintainable with respect to insertions against the base relations. Suppose that insertions are expected against both of the base relations T_1 and T_2. We carry out the following steps in addition to the previous ones:

www.manaraa.com

3 At the base-relation site, we extend the D-version (T_1^v, T_2^v) of the database by adding free tuples containing only variables. Let the free tuples be $t_1^v = (n_1, n_2)$ and $t_2^v = (n_3, n_4)$. The resulting database, $(T_1^v \cup \{t_1^v\}, T_2^v \cup \{t_2^v\})$ (shown in Table 7), is called the *DI-version* of (T_1, T_2). Let the values of the variables with respect to T_1 and T_2 still be kept in the valuation $\mathbf{v}$.

4 At the view site, in addition to the conditioned view, we store another relation, called the *insert set* of the view. The insert set, shown in Table 8, is defined as the difference between $\pi_B((T_1^v \cup \{t_1^v\}) \bowtie_{B=C} (T_2^v \cup \{t_2^v\}))$ and $\pi_B(T_1^v \bowtie_{B=C} T_2^v)$, which is equivalent to $\pi_B(\{t_1^v\} \bowtie_{B=C} T_2^v \cup T_1^v \bowtie_{B=C} \{t_2^v\} \cup \{t_1^v\} \bowtie_{B=C} \{t_2^v\})$.

Suppose that tuple (b, a) is inserted into T_1. The purpose of defining the DI-version of the database is to introduce variables in such a way that it is possible to model insertions against the base relations in terms of changes of the valuation of the variables in the free tuples[2]. The insertion of tuple (b, a) into T_1 can be modeled as assigning b to variable n_1, a to variable n_2 and η to n_3 and n_4, giving rise to the valuation $\mathbf{v}_i = \{y_1/b, y_2/b, y_3/e, z_1/a, z_2/e, n_1/b, n_2/a, n_3/\eta, n_4/\eta\}$.

Suppose that $\mathbf{v}_i$ is sent to the view site as information about the update against the base relation. Now the set of new tuples to be inserted into the materialized view can be computed as the instance of the insert set with respect to the valuation $\mathbf{v}_i$. The outcome of the computation is the tuple (a), which is the instance of the c-tuple $(n_2, n_2 = z_1)$ in the insert set with respect to $\mathbf{v} \cup \mathbf{v}_i$. The updated view is shown in Table 9.

Corresponding to a tuple inserted into a base relation, a new tuple needs to be inserted into the DI-version of the base relation. According to the definition of a DI-version of the database, all the values of attribute B in T_1 are to be identified by variables. Thus a new variable needs to be introduced to identify the value of attribute B (which is a) in the tuple (b, a) inserted into T_1. Let us assume that y_4 be the new variable. The new tuple to be inserted into T_1^v is (b, y_4). The information about the mapping from the value of attribute B in the inserted tuple to variable y_4 should also be sent to the view site.

The conditioned view and the insert set can be incrementally self-maintained in response to both deletions and insertions against the base relations. We have already shown how the conditioned view is maintained with respect to deletions against the base relations. In response to insertions into the base relations, new c-tuples have to be inserted into the conditioned view when new tuples are inserted into the materialized view. The insertion of (a) into the materialized view implies that $(n_2, n_2 = z_1)$ should be inserted into the conditioned view. Note that n_2 is in fact a place holder for the new variable y_4. The new c-tuple inserted into the conditioned view should be $(y_4, y_4 = z_1)$. The resulting relations are shown in Table 9. We leave the detail of updating the insert set to Section 5. Here we only mention the results. In response to the deletion of tuple (d, e) from T_1, the tuple $(y_3, y_3 = n_3)$ should be deleted from the insert set. The updated

[2] Deletions can be modeled as before; in addition, all the variables in the free tuples are given the value η.

www.manaraa.com

insert set is shown in Table 10. In response to an insertion, new tuples should be inserted into the insert set. For the insertion of (b, a) into T_1, tuple $(y_4, y_4 = n_3)$ is inserted into the insert set. The updated insert set is shown in Table 11.

Table 7. DI-version $(T_1^v \cup \{t_1^v\}, T_2^v \cup \{t_2^v\})$ of (T_1, T_2)

$T_1^v \cup \{t_1^v\}$

A	B
a	y_1
c	y_2
d	y_3
n_1	n_2

(t_1^v)

$T_2^v \cup \{t_2^v\}$

C	D
z_1	d
z_2	a
n_3	n_4

(t_2^v)

Table 8. The insert set

B	*con*
n_2	$n_2 = z_1$
n_2	$n_2 = z_2$
n_2	$n_2 = n_3$
y_1	$y_1 = n_3$
y_2	$y_2 = n_3$
y_3	$y_3 = n_3$

Table 9. The updated view and its conditioned version

The updated view

B
e
a

Conditioned version of the updated view

B	*con*
y_3	$(y_3 = z_1) \vee (y_3 = z_2)$
y_4	$y_4 = z_1$

4 Self-Maintenance of Views w.r.t. Deletions

In this section, we describe the self-maintenance of views in response to deletions against the base relations in more rigorous terms. We consider views defined by relational algebra expressions in this and the following sections.

Let us review some notions adopted from [5]. A view can be defined using a relational expression, which can be transformed into an equivalent *select-from-where* expression. An attribute A is said to be *distinguished* in a view defined by a *select-from-where* expression if attribute A appears in the set of attributes specified by the *select* clause. An attribute A is said to be *exposed* in a view defined by a *select-from-where* expression if A is used in the selection condition specified by the *where* clause. The union of the distinguished and exposed attributes is called the *extended attribute set* of the view. Essentially, only changes of the attributes in the extended attribute set of a view possibly affect the state of the view. For example, consider relation $R(A, B, C)$ and $R'(C)$ and a view defined by $\pi_A(R \bowtie R')$. Attributes A and C are in the extended attribute set of

www.manaraa.com

Table 10. The updated insert set in response to deletion of (d, e) from T_1

B	*con*
n_2	$n_2 = z_1$
n_2	$n_2 = z_2$
n_2	$n_2 = n_3$
y_1	$y_1 = n_3$
y_2	$y_2 = n_3$

Table 11. The updated insert set in response to insertion of (b, a) into T_1

B	*con*
n_2	$n_2 = z_1$
n_2	$n_2 = z_2$
n_2	$n_2 = n_3$
y_1	$y_1 = n_3$
y_2	$y_2 = n_3$
y_3	$y_3 = n_3$
y_4	$y_4 = n_3$

the view; attribute A is distinguished, while C is exposed in the join. Attribute B is said to be irrelevant to the view, as it is not mentioned at all in the view definition.

For each base relation $T \in I$ defined on a set X of attributes and a view q defined over I, if some tuples are expected to be deleted from T, then we choose one (and at most one) *exposed* attribute $A \in X$ of the view, replace the values of A in the relation T with *distinct* variables and put on the side the valuation of the variables. The resulting set of tables after variables are introduced in this way is called a *D-version* of the database I with respect to q. For a D-version of the given database, let $\mathbf{v}$ be the valuation that maps the introduced variables to the replaced values. Then $\mathbf{v}(I_d) = I$ and $\mathbf{v}(q(I_d)) = q(\mathbf{v}(I_d)) = q(I)$. $\mathbf{v}$ is called the valuation of the variables in I_d with respect to I.

Consider the database (T_1, T_2) and the view $\pi_B(T_1 \bowtie_{B=C} T_2)$ shown in Table 3. Attributes B and C are exposed in the view. To generate a D-version of the database, we introduce variables to replace values of the exposed attribute B in T_1 and C in T_2. The resulting tables are T_1^v and T_2^v shown in Table 4. The valuation of the variables is defined in $\mathbf{v}$.

As mentioned earlier, the purpose of generating a D-version of a database with respect to a view is to make it possible to model each deletion operation against the base relations as a change of the valuation of the variables.

Lemma 4. *Let I_d be a D-version of database I with respect to a view of I and the valuation of the variables in I_d with respect to I be $\mathbf{v}$. For the deletion of a tuple t from I, there is a valuation $\mathbf{v}'$ such that $\mathbf{v}'(I_d) = I - \{t\}$, and $\mathbf{v}' \sqsubseteq \mathbf{v}$.*

Proof. Let t be any tuple in a base relation in I and t_c be the corresponding c-tuple in I_d. Then t_d must be the result of replacing the value of an exposed attribute in t with a variable x. Let $\mathbf{v}'$ be such a valuation that $\mathbf{v}'(x) = \eta$ and for all other variables y, $\mathbf{v}'(y) = \mathbf{v}(y)$. Then $\mathbf{v}'(I_d) = I - \{t\}$ and $\mathbf{v}' \sqsubseteq \mathbf{v}$. This concludes the proof.

Definition 5. Let I_d be a D-version of database I with respect to view q and $\mathbf{v}$ denote the valuation of the variables in I_d with respect to I. The *conditioned*

www.manaraa.com

version of view $q(I)$ with respect to I_d, denoted T_d, is defined as:

$$T_d = \{(t, \phi(t))) \mid (t, \phi(t)) \in q(I_d), V^{\mathbf{v}}(\phi(t)) = T \text{ and } \mathbf{v}(x) \neq \eta \text{ for any } x \in t\}.$$

By definition, $T_d \subseteq q(I_d)$ and $\mathbf{v}(T_d) = \mathbf{v}(q(I_d))$. Since $\mathbf{v}$ is the valuation of the variables in I_d with respect to I, $q(I) = \mathbf{v}(q(I_d))$ must hold. Because $\mathbf{v}(T_d) = \mathbf{v}(q(I_d))$, it must be that $q(I) = \mathbf{v}(T_d)$. That is to say, the view $q(I)$ is the instance of the conditioned version T_d with respect to $\mathbf{v}$.

Before stating the theorem about the self-maintainability of the views in response to deletions, we prove another lemma.

Lemma 6. *Let I_d be a D-version of database I with respect to view q, T_d be the conditioned version of the view $q(I)$ with respect to I_d, and the valuation of the variables in I_d with respect to I be $\mathbf{v}$, i.e. $\mathbf{v}(I_d) = I$. Then*

$$\mathbf{v}'(T_d) = \mathbf{v}'(q(I_d)) \tag{7}$$

holds for any valuation $\mathbf{v}'$ of the variables in I_d such that $\mathbf{v}' \sqsubseteq \mathbf{v}$.

Proof. By definition of T_d, $T_d \subseteq q(I_d)$. For any valuation $\mathbf{v}' \sqsubseteq \mathbf{v}$, according to Proposition 3, $\mathbf{v}'(q(I_d) - T_d) \subseteq \mathbf{v}(q(I_d) - T_d)$ holds. Again, by definition of T_d, $\mathbf{v}(T_d) = \mathbf{v}(q(I_d))$, i.e. $\mathbf{v}(q(I_d) - T_d) = \emptyset$. On the other hand, since $T_d \subseteq q(I_d)$, it must be that $\emptyset \subseteq \mathbf{v}'(q(I_d) - T_d)$. That is, $\emptyset \subseteq \mathbf{v}'(q(I_d) - T_d) \subseteq \mathbf{v}(q(I_d) - T_d) = \emptyset$. It follows immediately that $\mathbf{v}'(q(I_d) - T_d) = \emptyset$. Consequently, $\mathbf{v}'(q(I_d)) = \mathbf{v}'(T_d \cup (q(I_d) - T_d)) = \mathbf{v}'(T_d) \cup \mathbf{v}'(q(I_d) - T_d) = \mathbf{v}'(T_d)$. This concludes the proof.

Now we are ready to prove that in response to deletions of the base relations, materialized views defined by relational algebra expressions are self-maintainable by means of the conditioned versions of the views. Basically, the updated state of a view can be computed by evaluating the conditioned version of the view.

Theorem 7. *Let I_d be a D-version of database I in response to a view q, $\mathbf{v}$ be the valuation of the variables in I_d with respect to I, and T_d be the conditioned version of the view with respect to I_d. The view, taken together with T_d, is self-maintainable in response to deletions against I.*

Proof. First, we prove that for the deletion of a tuple t from I, $q(I - \{t\}) = \mathbf{v}'(T_d)$ for some valuation $\mathbf{v}' \sqsubseteq \mathbf{v}$. According to Lemma 4, there must be some valuation $\mathbf{v}'$ of the variables in I_c such that $\mathbf{v}'(I_d) = I - \{t\}$ and $\mathbf{v}' \sqsubseteq \mathbf{v}$. According to (3), it follows that $\mathbf{v}'(q(I_d)) = q(\mathbf{v}'(I_d)) = q(I - \{t\})$. That is to say, the updated view $q(I - \{t\})$ is equivalent to $\mathbf{v}'(q(I_d))$. Let T_d be the conditioned version of the view with respect to I_d. According to Lemma 6, $\mathbf{v}'(q(I_d)) = \mathbf{v}'(T_d)$. Thus we have $\mathbf{v}'(T_d) = q(I - \{t\})$. Note that $q(I - \{t\})$ is the updated state of the view. Since T_d is materialized, according to the above theorem, we can compute the update view without access to the base relations. The conditioned version T_d can be updated by deleting all the c-tuples where the local conditions are not satisfied by $\mathbf{v}'$. We can conclude that the view, taken together with the conditioned version T_d of the view with respect to I_d, is self-maintainable in response to deletions against I. This concludes the proof.

www.manaraa.com

Let us review relations T_1, T_2 and the view defied by $\pi_B(T_1 \bowtie_{B=C} T_2)$ shown in Table 3. Suppose that tuple (c, b) is deleted from T_1. This deletion can be modeled as the modification of the valuation of y_2 from b (according to $\mathbf{v}$) to the non-existing value η. The new valuation of the variables is $\mathbf{v}_2 = \{y_1/b, y_2/\eta, y_3/e, z_1/b, z_2/e\}$. The result of evaluating the c-table shown in Table 5 using $\mathbf{v}_2$ is the updated state of the view. In this case, the updated state of the view is the same as the old state of the view. So the deletion has no impact on the state of the view.

5 Self-Maintenance of Views w.r.t. Deletions and Insertions

In this section, we describe the self-maintenance of views defined by relational algebra expressions in response to deletions and insertions against the base relations.

For a database I and view q on I, if both deletions and insertions are expected, then we first generate a D-version of I with respect to view q, denoted I_d. Note that there is a one-to-one mapping from the relations in I to the relations in I_d. For each relation T in I, if some tuples are expected to be inserted, then we add a free tuple with only distinct variables to the corresponding relation of T in I_d. The variables in the free tuple are distinct from all those already used. The resulting database I_{di} is called the *DI-version* of the database I with respect to q.

The purpose of defining the DI-version of the database is to introduce variables in such a way that it is possible to model not only deletions, but also insertions against the base relations as changes of the valuation of the variables. Deletions can be modeled by changing the assignments of some variables to the non-existing value, as indicated in the proof of Lemma 4. Insertions can be modeled by assigning values to the variables in the free tuples. Recall the example database (T_1, T_2) shown in Table 3 and the DI-version of the database shown in Table 7. The insertion of tuple (b, a) into T_1 can be modeled by assigning b to variable n_1 and a to variable n_2. Thus we have the following lemma. The proof is trivial and is omitted.

Lemma 8. *Let I_d be a D-version of database I with respect to a view of I and I_{di} be the corresponding DI-version of database I. Let the valuation of the variables in I_d with respect to I be $\mathbf{v}$. For the insertion of a tuple t into I, there is a valuation $\mathbf{v}_i$ such that $\mathbf{v}_i(I_{di}) = I \cup \{t\}$, and $\mathbf{v}_i(I_d) = \mathbf{v}(I_d) = I$.*

Now we define the notion of insert set.

Definition 9. Let I_d be a D-version of database I with respect to view q and I_{di} be the DI-version of database I with respect to the same view. $q(I_{di}) - q(I_d)$ is called the *insert set* of the view.

www.manaraa.com

The insert set of the view is the auxiliary data to be materialized in order to self-maintain the view.

Now we shall prove that a view is self-maintainable in response to deletions and insertions based on the conditioned version of the view and the insert set of the view. The assumption is that some information about the updates of the data at the base-relation site is sent to the view site.

Theorem 10. *A view q of a database I, taken together with the conditioned version of the view and the insert set of the view, is self-maintainable in response to deletions and insertions against I.*

Proof. We need to prove that 1) the view can be self-maintained in response to deletions and insertions against the base relations based on the conditioned version of the view and the insert set of the view, and 2) both the conditioned version of the view and the insert set of the view are self-maintainable.

We start with 1). For deletions, the proof is similar to that of Theorem 7. The updated state of the view can be computed by an evaluation of the conditioned version of the view. Now we consider insertions. Let I_d be a D-version of I with respect to the view q and I_{di} be a DI-version of I. Then $q(I_{di}) - q(I_d)$ is the insert set of the view. Suppose that t is inserted into I. According to Lemma 8, there is a valuation $\mathbf{v}_i$ such that $\mathbf{v}_i(I_{di}) = I \cup \{t\}$, and $\mathbf{v}_i(I_d) = \mathbf{v}(I_d) = I$. Thus

$$\mathbf{v}_i(q(I_{di}) - q(I_d)) = q(\mathbf{v}_i(I_{di})) - q(\mathbf{v}_i(I_d)) = q(I \cup \{t\}) - q(I).$$

That is to say, the set of tuples to be inserted into the view is the result of evaluating the insert set by $\mathbf{v}_i$. Both the insert set and the valuation $\mathbf{v}_i$ are available independent of the base relations. We can conclude that the view is self-maintainable in response to insertions against the base relations.

Now we consider 2). In the previous section, we have described how the conditioned view can be maintained in response to deletions against the base relations. In response to insertions, new c-tuples may be inserted into the conditioned view. Let I_d denote the old D-version, I_{di} denote the old DI-version and I'_d the new D-version after the insertions. I'_d can be obtained by first replacing the free tuples in I_{di} with the inserted tuples, and then replacing the values in the inserted tuples with new variables (in order to satisfy the requirement for the D-version). Let $\mathbf{r}_1$ be such a mapping from the variables in the free tuples in I_{di} to the new variables in I'_d that $I'_d = \mathbf{r}_1(I_{di})$ and $\mathbf{r}_1(I_d) = I_d$. Assume that $\mathbf{r}_1$ is sent to the view site as information about the insertions into the base relations. For a view q, $q(I'_d) - q(I_d) = q(\mathbf{r}_1(I_{di})) - q(I_d) = \mathbf{r}_1(q(I_{di}) - q(I_d))$, where $q(I_{di}) - q(I_d)$ is the insert set before updates. Let T_{ci} be the set of c-tuples in the old insert set where the associated local conditions are satisfied by $\mathbf{v}_i$. Then according to the definition of the conditioned view, the set of c-tuples to be inserted into the conditioned view is $\mathbf{r}_1(T_{ci})$, which can be computed based on the available information at the view site.

Next we show how to maintain the insert set in response to deletions and insertions against the base relations. For simplicity, we consider only a database with two relations T_1 and T_2, and a view $\pi_X(\sigma_E(T_1 \bowtie T_2))$, where X is a subset

www.manaraa.com

of attributes of T_1 and T_2, and E is a selection condition. The cases with multiple relations can be proved easily by induction. Let ΔT_1 and ΔT_2 denote the sets of tuples inserted into T_1 and T_2. Assume that (T_1^v, T_2^v) is a D-version of (T_1, T_2). Let $\mathbf{r}_2$ denote the mapping from the values in ΔT_1 and ΔT_2 to the variables in the updated D-version. Assume that $\mathbf{r}_2$, ΔT_1 and ΔT_2 are sent to the view site as the information about the insertions against the base relations. The old insert set before the insertions is:

$$S_{old} = \pi_X(\sigma_E((T_1^v \cup \{t_1^v\}) \bowtie (T_2^v \cup \{t_2^v\})) - \pi_X(\sigma_E(T_1^v \bowtie T_2^v))) = \\ \pi_X(\sigma_E(\{t_1^v\} \bowtie T_2^v \cup T_1^v \bowtie \{t_2^v\} \cup \{t_1^v\} \bowtie \{t_2^v\})).$$

The new insert set after the insertions is:

$$S_{new} = \pi_X(\sigma_E((T_1^v \cup \mathbf{r}_2(\Delta T_1) \cup \{t_1^v\}) \bowtie (T_2^v \cup \mathbf{r}_2(\Delta T_2) \cup \{t_2^v\})) - \\ \pi_X(\sigma_E((T_1^v \cup \mathbf{r}_2(\Delta T_1)) \bowtie (T_2^v \cup \mathbf{r}_2(\Delta T_2)))) = \\ \pi_X(\sigma_E(\{t_1^v\} \bowtie (T_2^v \cup \mathbf{r}_2(\Delta T_2)) \cup (T_1^v \cup \mathbf{r}_2(\Delta T_1)) \bowtie \{t_2^v\} \cup \{t_1^v\} \bowtie \{t_2^v\})).$$

Then the set of new tuples to be inserted into the insert set, which is the difference between the new insert set and the old insert set, is:

$$S_{new} - S_{old} = \pi_X(\sigma_E(\{t_1^v\} \bowtie \mathbf{r}_2(\Delta T_2) \cup \mathbf{r}_2(\Delta T_1) \bowtie \{t_2^v\})).$$

Since t_1^v and t_2^v are available at the view site, the above set can be computed without access to the base relations. When deletions against the base relations are considered, it can be derived that the above set is precisely the set of tuples to be deleted from the insert set. That is to say, the insert set can be self-maintained in response to deletions and insertions. This concludes the proof of the theorem.

6 Modifications as Atomic Operations

If modifications are handled as deletions followed by insertions, then Theorem 10 is sufficient to guarantee that the views can be self-maintained in response to modifications against the base data. In this section, we discuss hoe to handle modifications as atomic operations.

Suppose that we are given a database I of base relations. For each base relation, if there are some values that are expected to be modified, then we replace the values with *distinct* variables and put on the side the valuation of the variables. Recall that only modifications of the values of the attributes in the attribute set of a view are relevant to the view. So we only need to introduce variables to identify values of the attributes in the extended attribute set. Let I_m denote the resulting set of tables after variables are introduced as described above. I_m is called a *M-version* of the database I with respect to the view. Let $\mathbf{v}$ denote the valuation of the variables in I_m with respect to I. Then I is the instance of I_m with respect to $\mathbf{v}$, i.e. $\mathbf{v}(I_m) = I$.

The M-version of the example database (T_1, T_2) with respect to view $\pi_B(T_1 \bowtie_{B=C} T_2)$, shown in Figure 3, coincides with the D-version of the database with respect

www.manaraa.com

to the view. Generally, $rep(I_d) \subseteq rep(I_m)$. Thus the M-version of a database can replace the D-version of the database.

The purpose of generating the M-version of a database with respect to a view is to make it possible to model each modification operation against the base relations as a change of the valuation of the variables.

Lemma 11. *Let I_m be a M-version of database I with respect to a view of I and the valuation of the variables in I_m with respect to I be $\mathbf{v}$. For the modification of a tuple t in I to t', there is a valuation $\mathbf{v}'$ such that $\mathbf{v}'(I_m) = (I - \{t\}) \cup \{t'\}$.*

Proof. Let t_c be the corresponding c-tuple to t in I_m. Then t_c must be the result of replacing the values of attributes expected to be modified in t with variables. Let $\mathbf{v}'$ be such a valuation that $\mathbf{v}'(t_c) = t'$ and for the any other c-tuple $t'_c \in I_m$, $\mathbf{v}'(t'_m) = \mathbf{v}(t'_c)$. Then it is easy to see that $\mathbf{v}'(I_m) = (I - \{t\}) \cup \{t'\}$.

To model insertions against the base relations, we can add free tuples to I_m and generate a *MI-version* of the database.

As before, we could define a subset T_m of $q(I_m)$ as the *conditioned version* of view $q(I)$ with respect to I_m by $T_m = \{(t, \phi(t)) \mid (t, \phi(t)) \in q(I_m), V^{\mathbf{v}}(\phi(t)) = T$ and $\mathbf{v}(x) \neq \eta$ for any $x \in t\}$. Now the question is whether we can prove something similar to Theorem 7, namely that a view q, taken together with the conditioned version T_m, is self-maintainable in response to modifications against I. Unfortunately, it is not possible because, different from Lemma 4, Lemma 11 cannot guarantee $\mathbf{v}' \sqsubseteq \mathbf{v}$, and consequently not $\mathbf{v}'(T_m) = \mathbf{v}'(q(I_m))$ either. Thus instead of materializing the conditioned version T_m of the view, we have to materialize $q(I_m)$ for the self-maintenance of the view. For instance, consider the modification of tuple (e, a) in T_2 (in the example database) to tuple (b, a). This amounts to change the valuation of variable z_2 from e to b. Evaluating the conditioned version of the view using the new valuation of the variables leads to an empty state, while the state of the updated view contains one tuple (b). On the other hand, evaluating the auxiliary view $\pi_B(T_1^v \bowtie_{B=C} T_2^v)$ shown in Table 6 results exactly in the state of the updated view.

We prove the self-maintenance of a view taken together with the M-version. Due to space limit, we consider only modifications.

Theorem 12. *Let I_m be a M-version of database I with respect to a view q. The view, taken together with the auxiliary view $q(I_m)$, is self-maintainable in response to modifications against I.*

Proof. Consider the modification of a tuple t from some base relation in I to tuple t'. According to Lemma 11, there must be some valuation $\mathbf{v}'$ of the variables in I_c such that

$$\mathbf{v}'(I_m) = (I - \{t\}) \cup \{t'\}. \tag{8}$$

It follows from (8) and the property (3) that

$$\mathbf{v}'(q(I_m)) = q(\mathbf{v}'(I_m)) = q((I - \{t\}) \cup \{t'\}). \tag{9}$$

www.manaraa.com

That is to say, the updated view $q((I - \{t\}) \cup \{t'\})$ is equivalent to $\mathbf{v}'(q(I_m))$. Since $q(I_m)$ is materialized, we can compute the updated view without access to the base relations. The auxiliary view definition remains unchanged in response to the modification. This way, we achieve the goal of making the view self-maintainable in response to modifications of the identified values.

7 Conclusion and Discussion

We have presented a new approach to maintaining materialized views without accessing the underlying base relations. The basic idea is to to represent the base data using tables with *variables* and to materialize auxiliary data in form of conditional tables.

In order to make a materialized view self-maintainable in response to deletions against the base data, a D-version of the base data is generated at the base-relation site and a conditioned version of the view (which is a subset of the view over the D-version) is materialized at the view site. A deletion against the base data can be modeled as changes of the assignment to the variables in the D-version. The updated view can be computed by evaluating the conditioned version of the view with respect to the new assignment of the variables.

In order to make a materialized view self-maintainable in response to deletions and insertions against the base data, a DI-version of the base data is generated at the base-relation site. An insert set is materialized at the view site, in addition to the conditioned version of the view. An insertion into the base data can be modeled as changes of the assignment to the variables in the DI-version. The set of new tuples to be inserted into the materialized view can be computed by evaluating the insert set with respect to the new assignment of the variables.

Modifications against the base data can either be handled as deletions followed by insertions, or as atomic operations. In the latter case, a M-version of the base data is generated at the base-relation site and an auxiliary view of the M-version is materialized at the view site. A modification against the base data can be modeled as changes of the assignment to the variables in the M-version. The updated view can be computed by evaluating the auxiliary view with respect to the new assignment of the variables. We have also shown that the auxiliary data materialized at the view site are self-maintainable with respect to the updates against the base data.

This approach has two main advantages over the previous ones. First of all, it avoids the replication of the entire base relations. Secondly, It allows active self-maintenance of views triggered by expected updates of the base relations (by means of the introduction of variables).

Those advantages come at the price of storing and maintaining auxiliary data at the view site. The size of the auxiliary data varies with the complexity of the materialized view. The size of the conditioned view is linear to the size of the materialized view. Thus materializing the conditioned view does not lead to significant increase of the space complexity. When a view is defined upon a join of m base relations with the maximal size of n tuples, the size of the c-tables

www.manaraa.com

representing the insert set and the auxiliary view $q(I_m)$ (where I_m is the M-version of the database) can be exponential to the number of joined base relations in the materialized view. That implies, this approach is less expensive when handling deletions against the base data, and handling modifications as atomic operations using the current approach may need more space for materializing the auxiliary data than handling them as deletions followed by insertions.

Acknowledgments Thanks are due to Prof. Udo Lipeck and the anonymous referees for many helpful comments.

References

1. S. Blamey. Partial logic. In *Handbook of Philosophical Logic*, volume III, pages 1–70. D. Reidel, 1986.
2. G. Grahne. *The problem of incomplete information in relational databases.* Springer-Verlag, 1991.
3. A. Gupta, H. V. Jagadish, and I. S. Mumick. Data integration using self-maintainable views. Technical Report 223880-941101-32, AT&T Bell Laboratories, November 1994.
4. A. Gupta, H. V. Jagadish, and I. S. Mumick. Data integration using self-maintainable views. In P. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Advances in Database Technology – EDBT'96, 5th International Conference on Extending Database Technology*, LNCS 1057, pages 140–144, Berlin, 1996. Springer-Verlag.
5. Ahishs Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering*, 18(2):3–18, June 1995.
6. Ashish Gupta, Inderpal Singh Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In Peter Buneman and Jajodia Sushil, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, number 2 in SIGMOD Record, pages 157–166, New York, 1993. ACM Press.
7. R. Hull and G. Zhou. A framework of supporting data integration using the materialized and virtual approaches. In H.V. Jagadish, T.H. Merrett, and I.S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, June 4-6, 1996*, number 2 in SIGMOD Record, pages 481–492, New York, June 1996. ACM Press.
8. Tomasz Imielinski and Witold Jr. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, October 1984.
9. S. Kleene. *Introduction to Metamathematics.* Van Nostrand, Princeton, 1952.
10. D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Fourth International Conference on Parallel and Distributed Information Systems, Dec. 18-20, 1996, Miami Beach*, pages 158–169, Brüssel, 1996. IEEE Computer Society Press. http://www.research.att.com/ mumick/projects/matViews.html.
11. F. W. Tompa and J. A. Blakeley. Maintaining materialized views without accessing base data. *Information Systems*, 13(4):393–406, 1988.
12. Jennifer Widom. Research problems in data warehousing. In N. Pissinou, A. Silberschatz, E. K. Park, and K. Makki, editors, *Proc. of the 4th Int'l Conference on Information and Knowledge Management (CIKM'95).* ACM Press, November 1995. http://pub-db.stanford.edu/publist.html.

www.manaraa.com