1996

Reducing Intermediate Results for Incremental Updates of Materialized Views

Tetsuya Furukawa

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Report Number:
96-037

REDUCING INTERMEDIATE RESULTS FOR INCREMENTAL UPDATES OF MATERIALIZED VIEWS

Tetsuya Furukawa
Ahmed K. Elmagarmid

CSD TR-96-037
June 1996

Reducing Intermediate Results for Incremental Updates of Materialized Views

Tetsuya Furukawa†‡ and Ahmed K. Elmagarmid†

† Dept. Computer Science, Purdue University
West Lafayette, IN 47907, USA
{furukawa, ake}@cs.purdue.edu
‡ Dept. Economic Engineering, Kyushu University
Fukuoka 812-81, Japan
furukawa@en.kyushu-u.ac.jp

Abstract

To increase retrieval query processing efficiency, views are sometimes materialized in database systems. Since materialized views have to be consistent with their original data, they are updated according to updates of the original data. Materialized view update, however, becomes complicated when the view definition includes projection because one view data corresponds to several original data. This paper shows how to reduce the intermediate results to increase query processing efficiency when a materialized view defined with projection is updated. We need three steps to update materialized views, to get which view data corresponds with the updated original data, to check whether it indeed have to be updated, and to update the view data. The algorithms proposed in this paper reduce the intermediate results by applying the second step as early as possible during the first step, as well as allow to finish the second step in an early stage of the first step. The paper also gives algorithms for multiple updates.

1 Introduction

In database systems, query processing performance is one of the most important aspects of database efficiency. To increase query processing efficiency, query optimization methods reduce the amount of data to be processed by reducing intermediate results. It is particularly important to reduce intermediate results in distributed environment such as workstations connected by networks because communication costs depend on the intermediate results. In this paper we discuss how to reduce intermediate results to update materialized views incrementally.

Data in a database consists of original data and view definitions on the data which are used to realize user friendly interface. In order to improve retrieval query processing efficiency, views are sometimes materialized or cached to eliminate the view construction process. Materialized views are copies of data derived from original data. Derived data and materialized views are used extensively in object-oriented database systems that support engineering or software environments [7] [8]. These databases consist of original data and materialized views. The same concept also appears in view cache [14] and warehousing environment [17].

When original data is updated, the materialized views may also be required to change to keep database consistency. Since it costs a great deal to rebuild the materialized views, incremental re-computation methods are proposed for relational databases [16]. One of the performance problems occurs when views are updated. There are several approaches: immediate updates [3] [6], deferred updates [5] [15], and periodically or on-demand updates [1] [11]. These methods are based on relational expressions showing which tuples are inserted to or deleted from the materialized views.

Materialized view updates become complicated when the view expressions contain projection [2] [3] [4]. Suppose that a tuple is inserted to or deleted from a base relation, original data in a relational database. In general, the corresponding tuples of view relations are inserted or deleted according to the update of the base relation. There can be, however, view relations which do not change because the view tuples may be generated by other tuple connections rather than the connections including the updated tuple. If a view tuple is derived from other tuples than the updated tuple t of the base relation, the materialized view relation does not change when t is updated. Methods to maintain materialized views with projection in their definitions are proposed as counting algorithms [3] [4] and super-key approach [9] [10].

Let base relations be R_1, R_2, and R_3 as shown in Fig. 1 (a). If view relation V is defined as projection on ACE of these relations' join $R_1 * R_2 * R_3$, V is the relation shown in Fig. 1 (b). Some tuples of V are produced by several tuples of $R_1 * R_2 * R_3$, for example (a_2, c_2, e_2) by $(a_2, b_2, c_2, d_2, e_2)$, $(a_2, b_3, c_2, d_2, e_2)$, and $(a_2, b_4, c_2, d_2, e_2)$. This fact causes a problem when a base relation is updated. When tuple (a_2, b_2) of R_1 is deleted, only $(a_2, b_2, c_2, d_2, e_2)$ disappears in these tuples. Since the rest two tuples still exist, (a_2, c_2, e_2) is not deleted from V.

R_1

A	B
a_1	b_1
a_2	b_2
a_2	b_3
a_2	b_4

R_2

B	C	D
b_1	c_1	d_1
b_2	c_2	d_1
b_2	c_2	d_2
b_3	c_2	d_2
b_4	c_2	d_2
b_4	c_3	d_3

R_3

D	E
d_1	e_1
d_1	e_4
d_2	e_2
d_2	e_4
d_3	e_3

(a)

V

A	C	E
a_1	c_1	e_1
a_1	c_1	e_4
a_2	c_2	e_1
a_2	c_2	e_2
a_2	c_2	e_4
a_2	c_3	e_3

(b)

Figure 1: Base relations and a view

We need the following three steps to update materialized views incrementally when a base relation is updated.

1. Get new generated or disappeared tuple connections in base relations.
2. Check whether there are other tuple connections producing the same view tuples as the view tuples of the tuple connections in Step 1.
3. Update the view if Step 2 is false.

This paper gives algorithms to get the tuples actually inserted into or deleted from the materialized views whose view expressions include projection. We discuss such problem with relational databases, where the original data is base relations and views are defined by relational expressions on the base relations. The algorithms are based on combining Steps 1 and 2. For example, when the deleted tuple (a_2, b_2) of R_1 is joined with tuple (b_2, c_2, d_2) of R_2 in Step 1, we can find that the tuples in V derived from (a_2, b_2, c_2, d_2) are also derived from other tuples such as (a_2, b_3, c_2, d_2), which still exists after the deletion. There are cases in which Step 2 finishes in an early stage of Step 1, that is, the problem caused by projection disappears when the algorithms are used.

The counting algorithms stores the multiplicity of tuple duplicates [3] [4]. In the algorithms, Step 1 counts how many times view tuples derived from the deleted tuple, and subtracts the number from the multiplicity. In Step 2, it is checked whether the stored number is zero or not. If the multiplicity of a view tuple is zero, the tuple is deleted. While the counting algorithms always count the multiplicity of the view tuples derived from a modified tuple, the reducing algorithms proposed in this paper can finish the view update at the time that it is found that the multiplicity of view tuples does not become zero.

The reducing algorithms have the following advantages.

- Algorithm independence: Each algorithm is independent of the others. We can adopt an arbitrary combination of the algorithms at each join independently. Also the algorithms are independent from classical query optimization methods such as selection and projection as early as possible, and we can hence combine both of the reducing algorithms and the query optimization methods together.
- Data model independence: Although this paper adopts the relational data model for the discussion, the algorithms can be used in other data models such as object-oriented data model.
- Access independence: Each base relation is accessed only once independently. The algorithms consequently do not require any extra costs even if they are used in autonomous distributed databases such as multidatabases.

This paper is organized as follows. Section 2 gives basic concepts for this paper and shows a simple algorithm to update a materialized view incrementally, which is the base of the reducing algorithms proposed in this paper. In Section 3, basic ideas to reduce intermediate results are shown for deletion of a tuple, which are put together into one algorithm *REDUCE*. In Section 4, desirable properties of the reducing algorithms are discussed and classical query optimization methods are applied to the algorithms. Insertion of a tuple is discussed in Section 5. It is shown that every algorithm for deletion of a tuple given in Section 3 also works for insertion of a tuple. Section 6 gives algorithms for multiple updates which reflects several insertion and deletion of tuples to a materialized view in a time. Section 7 is the conclusions.

2 Incremental Approach to Updating Materialized View

A relation $R(X)$ is a set of tuples on attributes X. $\sigma_C R$, $\pi_Y R$, and $R_1 * R_2$ denotes selections of R by condition C, projection of R on attributes $Y (Y \subseteq X)$, and join of R_1 and R_2, respectively. If C is a set of conditions, the selection condition is conjunction of the elements of C, $c_1 \wedge c_2 \wedge \cdots \wedge c_m$ for $C = \{c_1, c_2, \cdots, c_m\}$. Although join is assumed to be natural join for simplicity in this paper, it can be easily extended to θ-join with a slight modification. $t[Y]$ is the Y value of tuple t.

A database is $(\mathbf{R}, \mathbf{V})$, where $\mathbf{R}$ is a set of relations $\{R_1, R_2, \cdots, R_n\}$ and $\mathbf{V}$ is a set of materialized view relations $\{V_1, V_2, \cdots, V_m\}$. Each view relation $V_i(X_{V_i})$ is defined as $f_i(\mathbf{R}_i)$, where f_i is a relational expression on a subset $\mathbf{R}_i$ of $\mathbf{R}$. View V_i is consistent if $V_i = f_i(\mathbf{R}_i)$. Database $(\mathbf{R}, \mathbf{V})$ is consistent if all views of $\mathbf{V}$ are consistent. If we can treat each materialized view individually, $\mathbf{R}_i$, V_i, and f_i are denoted by $\mathbf{R}$, V, and f, respectively, in the rest of this paper. V is defined as selection by condition C and projection on X_V of $R_1 * R_2 * \cdots * R_n$, that is $f(\mathbf{R}) = \sigma_C \pi_{X_V} R_1 * R_2 * \cdots * R_n$.

There are three types of updates, deletion of a tuple t from a relation R_i, insertion of a tuple t to a relation R_i, and modification of values of a tuple t in a relation R_i. First, we discuss deletion of a tuple. Insertion of a tuple is handled as the same way as shown in Section 5. Modification of values of a tuple from t_1 to t_2 is treated as deletion of t_1 and insertion of t_2 as shown in Section 6.

If a relation R_i is updated, the update have to be reflected to V to keep the database consistency. Since join operations are commutative, we assume R_1 is the relation to be updated without a loss of generality. For a tuple t (or a set of tuple S) of R_1, we say a tuple t_V of V is derived from t (S) if t_V is in $f(\{\{t\}, R_2, \cdots, R_n\})$ ($f(\{S, R_2, \cdots, R_n\})$, respectively). Suppose a tuple t is deleted from R_1. Some of the tuples derived from t may not be deleted from V because there may be the same tuples as the tuples derived from t in the tuples derived from $R_1 - \{t\}$. The tuples derived from t are the candidates to be deleted from the view relation and the tuples derived from $R_1 - \{t\}$ still exist after the deletion of t.

Definition 1 Let a tuple t be deleted from R_1. The candidate relation and the existing relation of the update for view $V = f(\mathbf{R})$ is $V_C = f(\{\{t\}, R_2, \cdots, R_n\})$ and $V_E = f(\{R_1 - \{t\}, R_2, \cdots, R_n\})$, respectively. V_C is the set of tuples of V which are derived from t and V_E is the set of tuples of V which are derived from $R_1 - \{t\}$. □

Note that V_E is the same view relation as the resulting view relation V_R of the deletion of t from R_1 because R_1 becomes $R_1 - \{t\}$ after the deletion. V, V_R, V_C, and V_E have the property as shown in Proposition 1.

Proposition 1 Let V be a view relation defined by $f(\mathbf{R}) = \sigma_C \pi_{X_V}(R_1 * R_2 * \cdots * R_n)$ and t be a tuple deleted from R_1. The resulting view relation V_R of the deletion is $V - (V_C - V_E)$. □

Proof: Suppose t_V is a tuple of $V - V_C$. There must be a tuple of R_1 other than t which derives t_V because V_C is a set of tuples derived from t; t_V is derived from $R - \{t\}$, that is $t_V \in V_E$. Thus $V - V_C \subseteq V_E$. Since $V - (V_C - V_E) = (V - V_C) \cup V_E$ and $V - V_C \subseteq V_E$, $V - (V_C - V_E)$ is equal to V_E, which is the same relation as V_R. *Q.E.D.*

While V_C is the relation of candidate tuples to be deleted from V, V_E may include some tuples of V_C. If $V_C \cap V_E \neq \emptyset$, there is at least one tuple of R_1 other than t which derives tuples in V_C. Proposition 1 shows that only the tuples in $V_C - V_E$ are deleted from V. If a view expression does not include projection, $V_C \cap V_E$ is the empty set. We can get V_R by deleting V_C from V in this case.

Definition 2 Let a tuple t be deleted from R_1. The target relation of the update is $V_T = V_C - V_E$ and the failure relation of the update is $V_F = V_C \cap V_E$. □

The target relation is the difference between V and V_R, which is the relation of the tuples deleted from the view V. The failure relation is the relation of the tuples which are derived from t but not deleted from the view V because they are also derived from other tuples of R_1 than t. Fig. 2 shows the relationship among these relations.

Example 1 Suppose tuple (a_2, b_2) is deleted from R_1 in the database shown in Fig. 1. The tuples of V derived from (a_2, b_2) may have to be deleted. V_C in Fig. 3 is the relation of such candidate tuples, the candidate relation. There are, however, tuples of R_1 other than (a_2, b_2) which derive some of the tuples of V_C. V_E in Fig. 3 is the existing relation whose tuples are derived from $R_1 - \{(a_2, b_2)\}$, the resulting relation V_R of the deletion in this example. Since (a_2, c_2, e_2) and (a_2, c_2, e_4) of V_C are also tuples of V_E, only (a_2, c_2, e_1), the tuple of the target relation V_T, is deleted from V. □

We could get V_T by difference between V_C and V_E according to the definition of V_T. It is not, however, a good method because we have to rebuild the resulting view relation V_R as V_E. We do not have to get the exact V_C and V_E to get V_T.

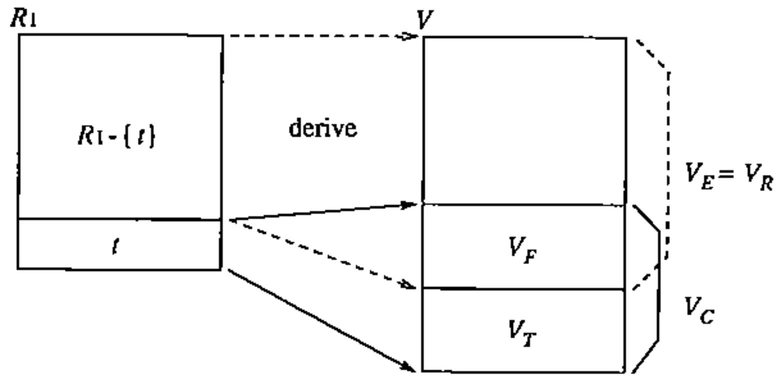

Figure 2: Candidate, existing, target, and failure relations

V_C

A	C	E
a_2	c_2	e_1
a_2	c_2	e_2
a_2	c_2	e_4

V_E

A	C	E
a_1	c_1	e_1
a_1	c_1	e_4
a_2	c_2	e_2
a_2	c_2	e_4
a_2	c_3	e_3

V_T

A	C	E
a_2	c_2	e_1

Figure 3: Delete tuple (a_2, b_2) from V

Proposition 2 Let V'_C and V'_E be such relations that $V_T \subseteq V'_C \subseteq V_C$ and $V_F \subseteq V'_E \subseteq V_E$, respectively. Then $V'_C - V'_E$ is the target relation V_T. □

Proof: $V'_C - V'_E \supseteq V_C - V_E$ because $V'_C \supseteq V_C$ and $V'_E \subseteq V_E$. Since V_T is $V_C - V_E$, $V'_C - V'_E$ is a superset of V_T. Then we get the difference of $(V'_C - V'_E)$ and V_T, $(V'_C - V'_E) - V_T = (V'_C - V_T) - V'_E$. $V'_C \subseteq V_C$ and $V_T = V_C - V_E$ show that $V'_C - V_T$ is a subset of $V_C \cap V_E$, which is equal to V_F. That is $(V'_C - V_T) - V'_E \subseteq V_F - V'_E$. $V_F - V'_E$ is empty because V'_E is a superset of V_F. Thus $V'_C - V'_E$ is a subset of V_T. $V'_C - V'_E = V_T$ because $V'_C - V'_E \supseteq V_T$ and $V'_C - V'_E \subseteq V_T$. Figure 4 illustrates this proof. *Q.E.D.*

The reducing algorithms given in this paper are based on Proposition 2. The algorithms show how to reduce V'_C and V'_E.

In Fig. 3 the tuples in $V_R(= V_E)$ derived from (a_1, b_1) of R_1, (a_1, c_1, e_1) and (a_1, c_1, e_4) are not in V_C because the value of attribute A, which is one of the attributes of the view relation, is different from (a_2, b_2). Such tuples can be reduced as algorithm *NAIVE* when V'_E is generated.

Algorithm 1 *NAIVE*

Let a view be $V(X_V) = f(\mathbf{R})$, t be a tuple of R_1, and Y_1 be $X_1 \cap X_V$.

1. $V^1_C = \{t\}$, $V^1_E = \sigma_{Y_1 = t[Y_1]}(R_1 - \{t\})$
2. For $i = 2$ to n, $V^i_C = V^{i-1}_C * R_i$, $V^i_E = V^{i-1}_E * R_i$
3. $V_T = \sigma_C \pi_{X_V} V^n_C - \sigma_C \pi_{X_V} V^n_E$ □

Theorem 1 The resulting relation V_T of algorithm *NAIVE* is the target relation of the update which deletes t from R_1. □

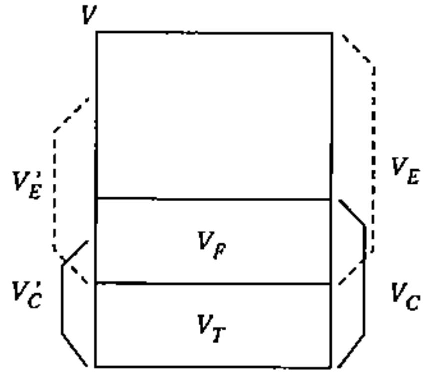

Figure 4: Property of the target relation

Proof: $\sigma_C\pi_{X_V}V_C$ is the candidate relation V_C^n because V_C^n is $\{t\} * R_2 * \cdots * R_n$. $\sigma_C\pi_{X_V}V_E^n = \sigma_C\pi_{X_V}(\sigma_{Y_1=t[Y_1]}(R_1-\{t\})*R_2*\cdots*R_n) = \sigma_{Y_1=t[Y_1]}(\sigma_C\pi_{X_V}(R_1-\{t\})*R_2*\cdots*R_n) = \sigma_{Y_1=t[Y_1]}V_E \subseteq V_E$. Every Y_1 values of tuples of V_C is $t[Y_1]$ because $V_C = \sigma_C\pi_{X_V}(\{t\}*R_2*\cdots*R_n)$. Since $V_F \subseteq V_C$, Y_1 values of tuples of V_F is also $t[Y_1]$. $V_F \subseteq \sigma_C\pi_{X_V}V_E^n$ because $\sigma_C\pi_{X_V}V_E^n = \sigma_{Y_1=t[Y_1]}V_E$ and $V_F \subseteq V_E$. Thus $V_F \subseteq \pi_{X_V}V_E^n \subseteq V_E$. Proposition 2 shows such relation $V_T = \sigma_C\pi_{X_V}V_C^n - \pi_{X_V}V_E^n$ is the target relation. *Q.E.D.*

Example 2 Fig. 5 shows the process to generate $V'_C = \pi_{ACE}V_C^3$ and $V'_E = \pi_{ACE}V_E^3$ by algorithm *NAIVE*. □

V_C^1

A	B
a_2	b_2

V_C^2

A	B	C	D
a_2	b_2	c_2	d_1
a_2	b_2	c_2	d_2

V_C^3

A	B	C	D	E
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_1	e_4
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_2	e_4

$\pi_{ACE}V_C^3$

A	C	E
a_2	c_2	e_1
a_2	c_2	e_2
a_2	c_2	e_4

V_E^1

A	B
a_2	b_3
a_2	b_4

V_E^2

A	B	C	D
a_2	b_3	c_2	d_2
a_2	b_4	c_2	d_2
a_2	b_4	c_3	d_3

V_E^3

A	B	C	D	E
a_2	b_3	c_2	d_2	e_2
a_2	b_3	c_2	d_2	e_4
a_2	b_4	c_2	d_2	e_2
a_2	b_4	c_2	d_2	e_4
a_2	b_4	c_3	d_3	e_3

$\pi_{ACE}V_E^3$

A	C	E
a_2	c_2	e_2
a_2	c_2	e_4
a_2	c_3	e_3

Figure 5: Delete tuple (a_2, b_2) from R_1 with algorithm *NAIVE*

Let N_1 be the number of tuples of R_1 and n_1 be the number of such tuples of R_1 that agree with t on Y_1. Intermediate results of *NAIVE* is about n_1/N_1 as comparing with rebuilding the view because the size of $V_C^1 \cup V_E^1$ is n_1/N_1 of $R_1 - \{t\}$. *NAIVE* shows that incrementally updating a materialized view is much more efficient than rebuilding the view even if the view definition includes projection.

V_C^1

A	B
a_2	b_2

V_C^2

A	B	C	D
a_2	b_2	c_2	d_1
a_2	b_2	c_2	d_2

V_C^3

A	B	C	D	E
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_1	e_4
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_2	e_4

$\pi_{ACE}V_C^3$

A	C	E
a_2	c_2	e_1
a_2	c_2	e_2
a_2	c_2	e_4

V_E^1

A	B
a_2	b_3
a_2	b_4

V_E^2

A	B	C	D
a_2	b_3	c_2	d_2
a_2	b_4	c_2	d_2

V_E^3

A	B	C	D	E
a_2	b_3	c_2	d_2	e_2
a_2	b_3	c_2	d_2	e_4
a_2	b_4	c_2	d_2	e_2
a_2	b_4	c_2	d_2	e_4

$\pi_{ACE}V_E^3$

A	C	E
a_2	c_2	e_2
a_2	c_2	e_4

Figure 6: Delete tuple (a_2, b_2) from R_1 with algorithm *RIE*

3 Reducing the Intermediate Relations

We do not have to create V_E and V_C to get V_T. It is enough to get V_E' and V_C' such that $V_F \subseteq V_E' \subseteq V_E$ and $V_T \subseteq V_C' \subseteq V_C$, respectively, as shown in Proposition 2. In this section, basic ideas to reduce V_E and V_C are shown as algorithms, and they are put together to algorithm *REDUCE*. Although we can apply some selection and projection of the view definition during the early stage of the joins, the selection and the projection are applied after the joins in this section for the simplicity. Such optimization is discussed in Section 4.

In Step 2 of algorithm *NAIVE*, we can reduce V_E^i because there are cases in which some tuples in V_E^i that do not agree with tuples in V_C^i on the attributes in X_V do not derive any tuples in V_F. For example (a_2, b_4, c_3, d_3) of V_E^2 does not derive the tuples in $\pi_{X_V}V_E^3$ that intersect with tuples in $\pi_{ACE}V_C^3$ in Fig. 5. *RIE* is the algorithm to reduce such tuples in the same way as *NAIVE* reduces V_E^1.

Algorithm 2 *RIE* (Reducing Intermediate results of the Existing relation)

Let a view $V(X_V)$ be defined by $f(\mathbf{R})$, t be a tuple of R_1, and Y_i be $\cup_{j=1}^{i} X_j \cap X_V$.

1. $V_C^1 = \{t\}$, $V_E^1 = \sigma_{Y_1=t[Y_1]}(R_1 - \{t\})$
2. For $i = 2$ to n, $V_C^i = V_C^{i-1} * R_i$, $V_E^i = \sigma_{Y_i \in \pi_{Y_i}V_C^i}(V_E^{i-1} * R_i)$
3. $V_T = \sigma_C \pi_{X_V} V_C^n - \sigma_C \pi_{X_V} V_E^n$ □

Lemma 1 The resulting relation V_T of algorithm *RIE* is the target relation of the update which deletes t from R_1. □

Proof: Same as the proof of Theorem 1 except that $\pi_{X_V}V_E^n$ becomes $\sigma_C\sigma_{Y_1=t[Y_1]}\ \sigma_{Y_2\in\pi_{Y_2}V_C^2}\cdots$ $\sigma_{Y_n\in\pi_{Y_n}V_C^n}V_E$. *Q.E.D.*

Example 3 Fig. 6 shows the process to generate $V_C' = \pi_{ACE}V_C^3$ and $V_E' = \pi_{ACE}V_E^3$ by algorithm *RIE*. □

There exist tuples with the same Y_2 value (a_2, c_2) in V_C^2 and V_E^2 in Fig. 5. Since the tuples derived from (a_2, b_2, c_2, d_2) of V_C^2 and (a_2, b_3, c_2, d_2) of V_E^2 are the same tuples (a_2, c_2, e_2) and (a_2, c_2, e_4) of V_C and V_E, we can get $V_T = \sigma_C\pi_{X_V}V_C^3 - \sigma_C\pi_{X_V}V_E^3$ even if (a_2, b_2, c_2, d_2) is deleted from V_C^2.

Although the AC value of (a_2, b_2, c_2, d_1) of V_C^2 is also (a_2, c_2), it cannot be deleted. It has the different value d_1 from the D value of the tuples of V_E^2 which have (a_2, c_2) on AC. Since (a_2, b_2, c_2, d_1) is joined with different tuples of R_3, the tuples derived from (a_2, b_2, c_2, d_2) may not derived from (a_2, b_3, c_2, d_2) or (a_2, b_4, c_2, d_2).

Note that this reduction cannot be applied to V_E^i; we cannot delete (a_2, b_3, c_2, d_2) from V_E^2 because it may produce the same tuples as tuples produced by other tuples of V_C^2 than (a_2, b_2, c_2, d_2), (a_2, c_2, e_4) in this example.

When the view definition has selection, we have to consider another problem. Suppose a tuple t_C^i of V_C^i agrees with a tuple t_E^i of V_E^i on the attributes in X_V and on the join attributes of later joins. t_C^i is still a candidate if t_E^i does not derive any tuple of V_E^n satisfying the selection. For example, if the view definition of the running example has selection $B = b_2$, (a_2, b_2, c_2, d_2) of V_C^2 is still a candidate because (a_2, b_3, c_2, d_2) of V_E^2 does not derive any tuple of the view relation. Thus t_C^i must agree with t_E^i on the attributes which appear in selection condition, too, if we delete t_C^i from V_C^i.

Algorithm *RIC* shows this reduction of the intermediate relations. Let $J(R_i, R_j)$ be the join attributes of the join $R_i * R_j$. If R_i or R_j is not defined, J is the empty set. Let $attr(c)$ be the set of attributes which appear in selection condition c, and $attr(C)$ be $\cup_{c \in C} attr(c)$ for a set of selection conditions C.

Algorithm 3 *RIC* (Reducing Intermediate results of the Candidate relation)
Let a view $V(X_V)$ be defined by $f(\mathbf{R})$, t be a tuple of R_1, Y_i be $\cup_{j=1}^{i} X_i \cap X_V$, and Z_i be $Y_i \cup \cup_{k=i}^{n} J(R_k, R_{k+1}) \cup attr(C)$.

1. $V_C^1 = \{t\}$, $V_E^1 = \sigma_{Y_1 = t[Y_1]}(R_1 - \{t\})$
2. For $i = 2$ to n, $V_E^i = V_E^{i-1} * R_i$, $V_C^i = \sigma_{Z_i \notin \pi_{Z_i} V_E^i}(V_C^{i-1} * R_i)$
3. $V_T = \sigma_C \pi_{X_V} V_C^n$ □

Lemma 2 The resulting relation V_T of algorithm *RIC* is the target relation of the update which delete t from R_1. □

Proof: Let $t_C^i \in V_C^i$, $t_E^i \in V_E^i$ such that $t_C^i[Z_i] = t_E^i[Z_i]$ in algorithm *NAIVE*. If $t_C^n = \pi_{X_V}(t_C^i * t_{i+1} * \cdots * t_n)(t_j \in R_j, i < j \le n)$, t_E^i is joinable with $t_{i+1} * \cdots * t_n$ because $t_C^i[Z_i] = t_E^i[Z_i]$ leads $t_C^i[J(V_C^i, R_{i+1})] = t_E^i[J(V_E^i, R_{i+1})]$. Thus for every $t_C^n \in V_C^n(t_C^n[Y_i] = t_C^i[Y_i])$, there exists a tuple $t_E^n \in V_E^n$ such that $t_E^n[X_V] = t_C^n[X_V]$. If t_C^n satisfies C, t_E^n also satisfies C because of $t_C^n[attr(C)] = t_E^n[attr(C)]$. $t_C^n[X_V]$ is a tuple of V_F because $t_E^n[X_V]$ is a tuple of V_E. If t_C^n does not satisfy C, $t_C^n[X_V]$ is not a tuple of the view relation. In both of the cases, t_C^i does not derive any tuple of the target relation, that is V_T of *RIC* is a superset of the target relation. On the other hand, $\sigma_C \pi_{X_V} V_C^n$ is $\sigma_C \pi_{X_V} \sigma_{X_V \notin \pi_{Z_n} V_E^n}(V_C^{n-1} * R_n) = \sigma_C \pi_{X_V}(V_C^{n-1} * R_n) - \sigma_C \pi_{X_V} V_E^n$, which is a set of tuples in a subset of V_C but not in $\sigma_C \pi_{X_V} V_E^n = V_E$. Therefore $\sigma_C \pi_{X_V} V_C^n$ is a superset of the target relation and does not include any tuple of V_E, that is, the target relation. *Q.E.D.*

Example 4 If we use algorithm *RIC* to get the target relation for the running example, the intermediate results of the algorithm are as shown in Fig. 7. □

The reduction of *RIC* is one of the advantages of the reducing algorithms. The counting algorithms [3] [4] have to produce V_C in order to get the multiplicity of the tuples. *RIC* finds some of the tuples of V_C^i which will derive only tuples in V_F but not tuples in V_T, and stops the derivation from such tuples. It often occurs that V_C^i becomes the empty set, that is V_T is empty, which allows to quit the materialized view update without accessing R_k $(i < k \le n)$.

The algorithms *RIE* and *RIC* shows the basic ideas of how to reduce the intermediate results of *NAIVE*. These ideas can be used for each i-th join in Step 2 individually, and they can be also

V_C^1

A	B
a_2	b_2

V_C^2

A	B	C	D
a_2	b_2	c_2	d_1

V_C^3

A	B	C	D	E
a_2	b_2	c_2	d_1	e_1

$\pi_{ACE}V_C^3$

A	C	E
a_2	c_2	e_1

V_E^1

A	B
a_2	b_3
a_2	b_4

V_E^2

A	B	C	D
a_2	b_3	c_2	d_2
a_2	b_4	c_2	d_2
a_2	b_4	c_3	d_3

V_E^3

A	B	C	D	E
a_2	b_3	c_2	d_2	e_2
a_2	b_3	c_2	d_2	e_4
a_2	b_4	c_2	d_2	e_2
a_2	b_4	c_2	d_2	e_4
a_2	b_4	c_3	d_3	e_3

$\pi_{ACE}V_E^3$

A	C	E
a_2	c_2	e_2
a_2	c_2	e_4
a_2	c_3	e_3

Figure 7: Delete tuple (a_2, b_2) from R_1 with algorithm *RIC*

combined in i-th join. Algorithm *REDUCE* is such algorithm that adopts both of the reducing algorithms. V_E^i can be further reduced in *REDUCE* because tuples of V_E^i for reduced tuples of V_C^i need not be kept no longer.

Algorithm 4 *REDUCE*

Let a view $V(X_V)$ be defined by $f(\mathbf{R})$, t be a tuple of R_1, Y_i be $\cup_{j=1}^{i} X_i \cap X_V$, and Z_i be $Y_i \cup \cup_{k=i}^{n} J(R_k, R_{k+1}) \cup attr(C)$.

1. $V_C^1 = \{t\}$, $V_E^1 = \sigma_{Y_1 = t[Y_1]}(R_1 - \{t\})$
2. For $i = 2$ to n, $V_E^{i\prime} = V_E^{i-1} * R_i$, $V_C^i = \sigma_{Z_i \notin \pi_{Z_i} V_E^{i\prime}}(V_C^{i-1} * R_i)$, $V_E^i = \sigma_{Y_i \in \pi_{Y_i} V_C^i} V_E^{i\prime}$
3. $V_T = \sigma_C \pi_{X_V} V_C^n$ □

Theorem 2 The resulting relation V_T of algorithm *REDUCE* is the target relation of the update which deletes t from R_1. □

Proof: The reduced tuples of V_E^i by $Y_i \in \pi_{Y_i} V_C^i$ in i-th step will not effect the reduction of V_C^k $(i < k \leq n)$ because no tuple derived by the reduced tuples agree with the tuples of V_C^k on Y_i which is a subset of Z_k. Therefore V_C in *REDUCE* is the same relation as V_C in *RIC*. Since Lemma 2 shows that $\sigma_C \pi_{X_V} V_C^n$ in *RIC* is the target relation, $\sigma_C \pi_{X_V} V_C^n$ in *REDUCE* is also the target relation. *Q.E.D.*

Example 5 Fig. 8 is the intermediate results to get the target relation by algorithm *REDUCE*. □

4 Properties of the Reducing Algorithms

When *RIE* and *RIC* are adopted together like *REDUCE*, we can use desirable properties which are shown as Theorems 3 and 4.

Theorem 3 If X_V is a super set of the key of R_i, $V_E^i = \emptyset$ in *REDUCE*. □

Proof: Suppose V_E^i is not empty, that is, some tuple t_E^i is in V_E^i. There must be a tuple t_C^i in V_C^i such that $t_E^i[Y_i] = t_C^i[Y_i]$ because of the selection condition of *RIE*. To derive these tuples, there must be tuples t_E^{i-1} in V_E^{i-1}, t_C^{i-1} in V_C^{i-1}, and t_E and t_C in R_i such that $t_E^{i-1}[Y_{i-1}] =$

V_C^1

A	B
a_2	b_2

V_C^2

A	B	C	D
a_2	b_2	c_2	d_1

V_C^3

A	B	C	D	E
a_2	b_2	c_2	d_1	e_1

$\pi_{ACE}V_C^3$

A	C	E
a_2	c_2	e_1

V_E^1

A	B
a_2	b_3
a_2	b_4

V_E^2

A	B	C	D
a_2	b_3	c_2	d_2
a_2	b_4	c_2	d_2

V_E^3

A	B	C	D	E
a_2	b_3	c_2	d_2	e_2
a_2	b_4	c_2	d_2	e_2
a_2	b_3	c_2	d_2	e_4
a_2	b_4	c_2	d_2	e_4

$\pi_{ACE}V_E^3$

A	C	E
a_2	c_2	e_2
a_2	c_2	e_4

Figure 8: Delete tuple (a_2, b_2) from R_1 with algorithm *REDUCE*

$t_E^i[Y_{i-1}]$, $t_C^{i-1}[Y_{i-1}] = t_C^i[Y_{i-1}]$, $t_E^{i-1}[J(R_{i-1}, R_i)] = t_E[J(R_{i-1}, R_i)]$, and $t_C^{i-1}[J(R_{i-1}, R_i)] = t_C[J(R_{i-1}, R_i)]$. $t_C^{i-1}[J(R_{i-1}, R_i)] \neq t_E[J(R_{i-1}, R_i)]$ because *RIC* deletes such tuples from V_C^{i-1} that their Z_{i-1} values do not appear in V_E^{i-1}. Thus t_E and t_C are different tuples. Since $X_i \cap X_V$ includes the key of R_i, $t_E[X_i \cap X_V] \neq t_C[X_i \cap X_V]$, which conflicts with $t_E^i[Y_i] = t_C^i[Y_i]$. Therefor t_E^i cannot exist in V_E^i, that is, $V_E^i = \emptyset$. *Q.E.D.*

By Theorem 3, $V_C^i * R_{i+1} * \cdots * R_n$ becomes the target relation if X_V includes the key of R_i. The cost to produce the tuples of V_T is proper one, while the costs to keep tuples of V_E^i and tuples of V_C^i which derive tuples of V_F are overhead. If V_E^i is empty, there is no overhead after i-th step. This theorem will help to decide an optimal join order. It is not a special case that a view include the key attributes of some base relation.

If X_V is a superset of the key of R_1, t is the only one tuple that can derive tuples of V whose $X_1 \cap X_V$ value is $t[X_1 \cap X_V]$. We can get V_T as $\sigma_{X_1 \cap X_V = t[X_1 \cap X_V]}V$, that is, $\sigma_{X_1 \cap X_V \neq t[X_1 \cap X_V]}V$ is the view relation after the update. We need not access R_i $(2 \leq i \leq n)$ in this case.

Theorem 3 is a property of the reducing algorithms when X_V is a superset of the key of a base relation. If a set of attributes of base relations is a superset of the key of the view relation, the reducing algorithms have a property shown in Theorem 4.

Theorem 4 If $Y_i = \cup_{j=1}^{i} X_j \cap X_V$ is a superset of the key of the view relation and $C = \emptyset$, $\sigma_{Y_i \cap X_V \in (V_C^i[Y_i \cap X_V] - V_E^i[Y_i \cap X_V])}V$ is the target relation V_T in *NAIVE*, *RIE*, *RIC*, and *REDUCE*. □

Proof: Since $C = \emptyset$, every tuple of V_C^i and V_E^i derives some tuples of V. If two tuples of V_C^i and V_E^i agree with each other on $Y_i \cap X_V$, they derive the same tuple of V because $Y_i \cap X_V$ is a superset of the key of V. For a tuple t_C^i of V_C^i, if there is a tuple t_E^i of V_E^i such that $t_C^i[Y_i \cap X_V] = t_E^i[Y_i \cap X_V]$, t_C^i does not derive any tuple of V_T because t_E^i derives the same tuple as t_C^i derives. If there is no tuple t_E^i of V_E^i such that $t_C^i[Y_i \cap X_V] = t_E^i[Y_i \cap X_V]$, the tuples derived from t_C^i is tuples of V_T because the tuples derived from tuples of V_E^i do not agree with them on $Y_i \cap X_V$. That is, the tuples of V_C whose $Y_i \cap X_V$ values are in $V_C^i[Y_i \cap X_V] - V_E^i[Y_i \cap X_V]$ are tuples of V_T. Every tuple of $V - V_C$ has a different value of $Y_i \cap X_V$ from the tuples of V_T because $Y_i \cap X_V$ is a superset of the key of V. Therefor the set of the tuples of V whose $Y_i \cap X_V$ values are in $V_C^i[Y_i \cap X_V] - V_E^i[Y_i \cap X_V]$ is V_T. *Q.E.D.*

Theorem 4 allows to stop the reducing algorithms at i-th step. We need not access R_k $(i < k \leq n)$ for any instance of the database if the assumption of the theorem is satisfied.

The reducing algorithms allow to adopt classical query optimization methods, selection and projection as early as possible. We discuss how the query optimization methods are applied to the reducing algorithms.

There are several query optimization methods to increase performance by applying selection as early as possible. There is also a materialized view update algorithm based on this method [13], which does not treat the projection problem. The reducing algorithms can adopt this optimization with slight modification.

Let C_i be the set of the selection conditions $\{c|c \in C \cup_{j=1}^{i-1} C_j, attr(c) \subseteq \cup_{j=1}^{i} X_j\}$, which become newly applicable in i-th step. The reducing algorithms are modified as follows.

Modifying the reducing algorithms for selection

1. the first step: $V_C^1 = \sigma_{C_1}\{t\}, V_E^1 = \sigma_{C_1}\sigma_{Y=t[Y_1]}(R_1 - \{t\})$
2. the i-th step ($2 \leq i \leq n$): $V_E^{i-1} * R_i$ and $V_C^{i-1} * R_i$ are replaced with $\sigma_{C_i}(V_E^{i-1} * R_1)$ and $\sigma_{C_i}(V_C^{i-1} * R_i)$, respectively.
3. V_T: $\pi_{X_V} V_C^n - \pi_{X_V} V_E^n$ in *NAIVE* and *RIE*, $\pi_{X_V} V_C^n$ in *RIC* and *REDUCE* □

Tuples (a_2, b_3, c_2, d_2) and (a_2, b_4, c_2, d_2) of V_E^2 in Fig. 8 derive the same tuples of $\pi_{ACE} V_E^3$. Also there are cases that some tuples of V_C^i derive the same tuple of V_C. It is enough to keep only one tuple of the tuples of V_C^i and V_E^i deriving the same set of tuples of $\sigma_C \pi_{X_V} V_C^n$ and $\sigma_C \pi_{X_V} V_E^n$, respectively. The optimization by projection as early as possible can reduce such redundancy as well as reduces the number of attributes.

Note that even if two tuples have the same values of attributes $\cup_{j=1}^{i} X_j \cap X_V$, they can have different values of other than the attributes. If the values of the join attributes of $R_k * R_{k+1} (k \leq i)$ are different from each other, we have to keep them. For example, the two tuples of V_C^2 in Fig. 5 derive the different sets of tuples of $\pi_{ACE} V_C^3$, which have the same value (a_2, c_2) on $AC = \cup_{j=1}^{2} X_i \cap ACE$, $\{(a_2, c_2, e_1), (a_2, c_2, e_4)\}$ and $\{(a_2, c_2, e_2), (a_2, c_2, e_4)\}$, respectively. Furthermore attributes used by selection conditions must be kept, too.

The attributes which have to be kept in i-th step are

- $Y_i = \cup_{j=1}^{i} X_j \cap X_V$: attributes in X_V,
- $\cup_{k=i}^{n} J(R_k, R_{k+1})$: join attributes of k-th step ($k > i$), and
- $attr(C)$: attributes which appear in selection conditions.

Let Z_i be $Y_i \cup \cup_{k=i}^{n} J(R_k, R_{k+1}) \cup attr(C)$. The reducing algorithms are modified as follows.

Modifying the reducing algorithms for projection

1. the first step: $V_C^1 = \pi_{Z_1}\{t\}, V_E^1 = \pi_{Z_1}\sigma_{Y=t[Y_1]}(R_1 - \{t\})$
2. the i-th step ($2 \leq i \leq n$): $V_E^{i-1} * R_i$ and $V_C^{i-1} * R_i$ are replaced with $\pi_{Z_i}(V_E^{i-1} * R_1)$ and $\pi_{Z_i}(V_C^{i-1} * R_i)$, respectively.
3. V_T: $\sigma_C V_C^n - \sigma_C V_E^n$ in *NAIVE* and *RIE*, $\sigma_C V_C^n$ in *RIC* and *REDUCE* □

When we use both of the optimization methods, selection and projection as early as possible, Z_i should be changed to less attributes because attributes which appear in $attr(C_j)(1 \leq j \leq i)$ but not in $attr(C_k)(i < k \leq n)$ are no longer needed. Z_i is $Y_i \cup \cup_{k=i}^{n} J(R_k, R_{k+1}) \cup \cup_{k=i+1}^{n} attr(C_k)$ in this case, and the algorithms are modified as follows.

Modifying the reducing algorithms for selection and projection

1. the first step: $V_C^1 = \pi_{Z_1}\sigma_{C_1}\{t\}, V_E^1 = \pi_{Z_1}\sigma_{C_1}\sigma_{Y=t[Y_1]}(R_1 - \{t\})$
2. the i-th step ($2 \leq i \leq n$): $V_E^{i-1} * R_i$ and $V_C^{i-1} * R_i$ are replaced with $\pi_{Z_i}\sigma_{C_i}(V_E^{i-1} * R_1)$ and $\pi_{Z_i}\sigma_{C_i}(V_C^{i-1} * R_i)$, respectively.
3. V_T: $\sigma_C V_C^n - \sigma_C V_E^n$ in *NAIVE* and *RIE*, $\sigma_C V_C^n$ in *RIC* and *REDUCE*

□

Algorithm *REDUCE*$^+$ is such reducing algorithm that is the result of applying both of the optimization methods to *REDUCE*.

Algorithm 5 *REDUCE*$^+$ (Reducing algorithm with query optimization methods)

Let a view $V(X_V)$ be defined by $f(\mathbf{R})$, t be a tuple of R_1, Y_i be $\cup_{j=1}^{i} X_j \cap X_V$, and Z_i be $Y_i \cup \cup_{k=i}^{n} J(R_k, R_{k+1}) \cup \cup_{k=i+1}^{n} attr(C_k)$.

1. $V_C^1 = \pi_{Z_1} \sigma_{C_1} \{t\}$, $V_E^1 = \pi_{Z_1} \sigma_{C_1} \sigma_{Y_1 = t[Y_1]} (R_1 - \{t\})$
2. For $i = 2$ to n, $V_E^{i'} = \pi_{Z_i} \sigma_{C_i} (V_E^{i-1} * R_i)$, $V_C^i = \pi_{Z_i} \sigma_{C_i} \sigma_{Z_i \notin \pi_{Z_i} V_E^{i'}} (V_C^{i-1} * R_i)$, $V_E^i = \sigma_{Y_i \in \pi_{Y_i} V_C^i} V_E^{i'}$
3. $V_T = V_C^n$

□

Theorem 5 The resulting relations V_T of the resulting algorithms of modifying *RIE*, *RIC*, and *REDUCE* for selection, projection, or both of them are the target relation of the update which deletes t from R_1. □

Proof: Even if the selection σ_{C_i} is applied in i-th step, V_E and V_C are the same relation as C_i is applied in the last step, because the tuples of $V_E^{i-1} * R_i$ and $V_C^{i-1} * R_i$ which do not satisfy C_i do not derive any tuple of V_E and V_C, respectively.

Projection π_{Z_i} in i-th step does not effect the later selection because V_E^i and V_C^i keep the attributes which appear in C_k or Y_k $(i < k \leq n)$. *Q.E.D.*

Example 6 Fig. 9 shows the process to get the target relation of the running example by algorithm *REDUCE*$^+$. Although the view expression of the example does not contain selection we can observe the effect of query optimization by projection. □

V_C^1

A	B
a_2	b_2

V_C^2

A	C	D
a_2	c_2	d_1

V_C^3

A	C	E
a_2	c_2	e_1

V_E^1

A	B
a_2	b_3
a_2	b_4

V_E^2

A	C	D
a_2	c_2	d_2

V_E^3

A	C	E
a_2	c_2	e_2
a_2	c_2	e_4

Figure 9: Delete tuple (a_2, b_2) from R_1 with algorithm *REDUCE*$^+$

Theorems 3 and 4 also valid in the reducing algorithms after the query optimization methods are applied. Note that $C = \emptyset$ in the assumption of Theorem 4 can be weakened to $C_k = \emptyset$ $(i < k \leq n)$ because the theorem requires only that there is no selection to be applied after i-th step.

5 Insertion of a Tuple

In Sections 3 and 4 the algorithms for deletion of a tuple from a base relation are shown. In this section, it is shown that the algorithms also work for insertion of a tuple to a base relation without any change.

Suppose a tuple t is inserted to R_1 ($t \notin R_1$). Although the tuples of relation $V_C = \sigma_C \pi_{X_V}(\{t\} * R_2 * \cdots * R_n)$ are candidates to be inserted to V, the same tuples may already exist in V. Thus, we define the candidate relation and the existing relation of a case of tuple insertion.

Definition 3 Let a tuple t be inserted to R_1. The candidate relation and the existing relation of the update for a view $V = f(\mathrm{R})$ are $V_C = f(\{\{t\}, R_2, \cdots, R_n\})$ and $V_E = f(\{R_1, R_2, \cdots, R_n\})$, respectively. □

Note that V_E is the current view relation V. Since $R_1 - \{t\} = R_1$, the definitions of V_C and V_E for tuple deletion in Definition 1 can be regarded as the definition for tuple insertion.

As the same way as deletion of a tuple, we define the target relation and the failure relation for insertion of a tuple.

Definition 4 Let a tuple t be inserted to R_1. The target relation of the update is $V_T = V_C - V_E$ and the failure relation of the update is $V_F = V_C \cap V_E$. Fig. 10 shows the relationship among these relations. □

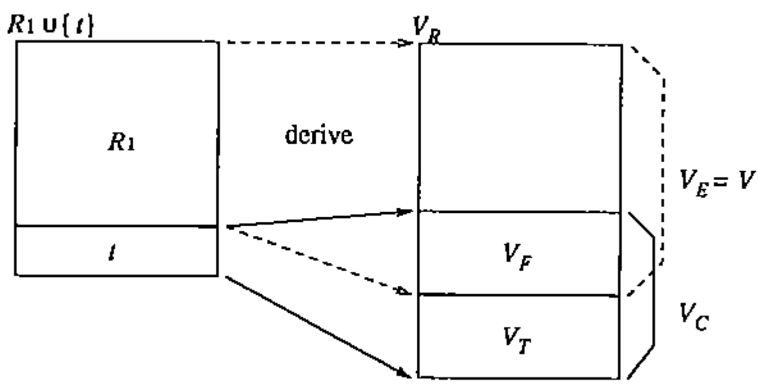

Figure 10: Candidate, existing, target, and failure relations for insertion

Proposition 3 Let V'_C and V'_E be such relations that $V_T \subseteq V'_C \subseteq V_C$ and $V_F \subseteq V'_E \subseteq V_E$, respectively. Then the target relation of insertion is $V'_C - V'_E$. □

Proof: Same as the proof of Proposition 1. Fig. 4 also illustrates the proof of this proposition. *Q.E.D.*

The candidate, existing, target, and failure relations of tuple insertion have the same relationship as ones of tuple deletion as shown in Proposition 3. In the same way of tuple deletion, algorithms *NAIVE, RIE, RIC,* and *REDUCE* work to get the target relation in the case of tuple insertion.

Theorem 6 Algorithms *NAIVE, RIE, RIC,* and *REDUCE* output the target relation of the update which insert a tuple t to R_1. □

Proof: Proposition 3 shows that the candidate, existing, target, and failure relations for tuple insertion have the same property as those of tuple deletion. The proof is the same as the proofs of Theorem 1, Lemmas 1 and 2, and Theorem 2. *Q.E.D.*

As shown in Theorem 6, the algorithms for tuple deletion can be used for tuple insertion. The properties for tuple deletion shown in Section 4 are also valid for insertion of a tuple. *REDUCE+* works for tuple insertion and Theorems 3 and 4 are also held in the case of tuple insertion. The proofs of these are the same as the proofs for tuple deletion.

Tuple insertion, however, has a different property from tuple deletion, which is formally described as Lemma 3.

Lemma 3 Let V'_C be such relation that $V_T \subseteq V'_C \subseteq V_C$. $V \cup V'_C$ is the resulting relation V_R of the insertion of a tuple. □

Proof: The resulting relation V_R is $V \cup V_T$. Since $V_T = V_C - V_E$ by Definition 4 and $V_E = V$ for tuple insertion, $V_R = V \cup (V_C - V_E) = V \cup (V_C - V) = V \cup V_C$. For V'_C such that $V_T \subseteq V'_C \subseteq V_C$, $V_R = V \cup V'_C$ because $V_R = V \cup V_T = V \cup V_C$. *Q.E.D.*

By Lemma 3, we can get V_R without V'_E as $V \cup V'_C$. This property is quite different from tuple deletion. Lemma 3 allows to cut the computation of V'_E at some step in each algorithm. That is, the algorithms can stop at i-th step and then $V'_C = \sigma_C \pi_{X_V} V^i_C * R_{i+1} * \cdots * R_n$ is added to V. Since algorithms *RIE*, *RIC*, and *REDUCE* contain operations to reduce the intermediate results, there are cases that it is more efficient to cut the reducing process after some i-th step.

6 Multiple Updates

It is sometimes more efficient to update the view for several updates of a base relation than to update the view for each update of a base relation. This strategy is proposed as deferred updates in [5] [15]. In this section algorithms to handle such multiple updates is proposed. Since modification of values can be transformed to multiple updates, deletion of a tuple and insertion of a tuple, the algorithms work for tuple modification.

If the all updates are either insertions or deletions, we can get the reducing algorithms by replacing inserted or deleted tuple t with a set of tuples S. The algorithms consequently work for insertion or deletion of multiple tuples with changes in the selection condition $Y_1 = t[Y_1]$ in Step 1 to $Y_1 \in \pi_{Y_1} T$ in each algorithms.

Then we treat insertion of tuples and deletion of tuples together. Let S_I and S_D ($S_I \cap S_D = \emptyset$) be sets of tuples which are inserted to and deleted from R_1, respectively.

Definition 5 $V_{C_I} = f(\{S_I, R_2, \cdots, R_n\})$ and $V_{C_D} = f(\{S_D, R_2, \cdots, R_n\})$ are the candidate relations of the insertion and the deletion, respectively. The existing relations of the insertion and the deletion are $V_{E_I} = f(\{R_1 - S_D, R_2, \cdots, R_n\})$ and $V_{E_D} = f(\{R_1 - S_D, R_2, \cdots, R_n\})$, respectively. The target and the failure relations of the insertion are $V_{T_I} = V_{C_I} - V_{E_I}$ and $V_{F_I} = V_{C_I} \cap V_{E_I}$, and the target and the failure relations of the deletion are $V_{T_D} = V_{C_D} - V_{E_D}$ and $V_{F_D} = V_{C_D} \cap V_{E_D}$, respectively. □

Lemma 4 Let S_I and S_D be a set of tuples inserted to and deleted from R_1, respectively. $(V - V_{S_D}) \cup V_{T_I}$ is the resulting view relation V_R of the updates which insert S_I to R_1 and delete S_D from R_1. That is $(V - V_{S_D}) \cup V_{T_I} = \sigma_C \pi_{X_V}((R_1 - S_D) \cup S_I) * R_2 * \cdots * R_n$. □

Proof: Suppose the database after deletion of S_D from R_1. The base relations are $R'_1 = R_1 - S_D, R_2, \cdots, R_n$ and the view is $V' = V - V_{S_D}$. Then insert S_I to R'_1. The candidate relation of this insertion is $V'_{C_I} = f(\{S_I, R_2, \cdots, R_n\})$, which is equal to V_{C_I}. The view after the insertion is $V'' = V' \cup V'_{C_I}$ by Lemma 3. Thus $V'' = (V - V_{S_D}) \cup V_{C_I}$. V'' is the view of the database whose base relations are $(R_1 - S_D) \cup S_I, R_2, \cdots, R_n$. $(R_1 - S_D) \cup S_I$ is the relation after the updates described in the lemma because $S_D \cap S_I = \emptyset$. *Q.E.D.*

Lemma 4 shows that we can update the view relation using V_{T_I} and V_{T_D}, which we can get by *NAIVE, RIE, RIC*, or *REDUCE*.

Algorithm 6 *MU1* (Multiple Updates 1)

Let a view $V(X_V)$ be defined by $f(\mathbf{R})$ and S_I and S_D be sets of tuples inserted to and deleted from R_1 ($S_I \cap S_D = \emptyset$), respectively.

1. Get V_{T_I} and V_{T_D} by algorithms *NAIVE, RIE, RIC*, or *REDUCE*.
2. Let V_R be $(V - V_{T_D}) \cup V_{T_I}$. □

Theorem 7 The resulting relation V_R of algorithm *MU1* is the view relation after the updates. □

Proof: V_{T_I} and V_{T_D} in algorithm *MU1* are the target relations of insertion and deletion, respectively. Lemma 4 shows $V_R = (V - V_{T_D}) \cup V_{T_I}$ in algorithm *MU1* is the view after the update of R_1. *Q.E.D.*

The advantage of the multiple updates is not only to reduce the times of materialized view updates but also to reduce the intermediate results. While $V_R = (V - V_{T_D}) \cup V_{T_I}$ by Lemma 4, $(V \cup V_{T_I}) - V_{T_D}$ cannot be V_R because there can be a tuple t in V_R such that $t \in V_{T_I}$ and $t \in V_{T_D}$. Such t is derived from both S_I and S_D and is not in $(V \cup V_{T_I}) - V_{T_D}$. This fact indicates that we can consider the newly generated tuples by insertion of S_I as tuples of the existing relation of the deletion. The intermediate results is reduced further by treating V_{T_I} as a part of the existing relation of the deletion. In the selection condition to reduce V_C^i in i-th step $\sigma_{Z_i \notin \pi_{Z_i} V_E^{i'}}$ of *RIC* and *REDUCE*, $V_E^{i'}$ becomes $V_E^{i'} \cup V_C^i$ where $V_E^{i'}$ and V_C^i should be $V_{E_D}^{i'}$ and $V_{C_I}^i$, respectively, by the definition for multiple updates.

MU2 is such the algorithm that applied this reduction to the multiple updates version of *REDUCE*.

Algorithm 7 *MU2* (Multiple Updates 2)

Let a view $V(X_V)$ be defined by $f(\mathbf{R})$, S_I and S_D be sets of tuples inserted to and deleted from R_1 ($S_I \cap S_D = \emptyset$), respectively, Y_i be $\cup_{j=1}^{i} X_j \cap X_V$, and Z_i be $Y_i \cup \cup_{k=i}^{n} J(R_i, R_{i+1}) \cap attr(C)$.

1. $V_{C_I}^1 = S_I$, $V_{E_I}^1 = \sigma_{Y_1 \in \pi_{Y_1} S_I} R_1$, $V_{C_D}^1 = S_D$, $V_{E_D}^1 = \sigma_{Y_1 \in \pi_{Y_1} S_D} R_1$
2. For i=2 to n, $V_{E_I}^{i'} = V_{E_I}^{i-1} * R_i$, $V_{C_I}^i = \sigma_{Z_i \notin \pi_{Z_i} V_{E_I}^{i'}} (V_{C_I}^{i-1} * R_i)$, $V_{E_I}^i = \sigma_{Y_i \in \pi_{Y_i} V_{C_I}^i} V_{E_I}^{i'}$,
   $V_{E_D}^{i'} = V_{E_D}^{i-1} * R_i$, $V_{C_D}^i = \sigma_{Z_i \notin \pi_{Z_i} V_{E_D}^{i'} \cup V_{C_I}^i} (V_{C_D}^{i-1} * R_i)$, $V_{E_D}^i = \sigma_{Y_i \in \pi_{Y_i} V_{C_D}^i} V_{E_D}^{i'}$
3. $V_{T_I} = \sigma_C \pi_{X_V} V_{C_I}^n$, $V'_{T_D} = \sigma_C \pi_{X_V} V_{C_D}^n$ □

Theorem 8 For V_{T_I} and V'_{T_D} of algorithm *MU2*, $(V'_{T_D}) \cup V_{T_I}$ is the resulting relation of the update, insertion of S_I to R_1 and deletion of S_D from R_1. □

Proof: V_{T_I} is the target relation of the insertion because the process to produce V_{T_I} is the same as *REDUCE*. Obviously $V'_{T_D} \subseteq V_{T_D}$. Let T_D be $V_{T_D} - V'_{T_D}$. The tuples of T_D are caused by the selection $\sigma_{Z_i \notin \pi_{Z_i} V_{C_I}^i}$ to reduce $V_{C_D}^i$ in Step 2. In the same way as the proof of Lemma 2, it can be shown that the tuples of T_D are in V_{C_I}. Thus $(V - V'_{T_D}) \cup V_{T_I}$ is equal to $(V - V_{T_D}) \cup V_{T_I}$, which is the resulting relation V_R. *Q.E.D.*

The reducing algorithms for multiple updates *MU1* and *MU2* can be also combined with the query optimization methods by the same way as the algorithms for single update shown in Section 4. The properties of the single update algorithms, Theorems 3 and 4, are still held by the multiple update algorithms.

7 Conclusions

We have shown the incremental recomputation algorithms to update materialized views. The algorithms that handle the cases that view expressions include projection, where it is required to check if the candidate tuples are actually inserted to or deleted from the view relations, are there. One of the advantages of the reducing algorithms is to quit producing the tuples other than the tuples in the target relation. It is often found in early stage of the update process that candidate tuples at i-th step do not derive any tuples of the target relation which allows to quit producing view tuples from the candidate tuples, while the counting algorithms always produce all view tuples derived from the candidate tuples.

The algorithms are effective for views in distributed databases as well because the communication cost depends on the size of the intermediate relations. They are also applicable to multidatabases because each step of them can be closed in an autonomous component.

The ideas to reduce intermediate results were given as *RIE* and *RIC*. We need not apply these reducing methods in every step like *REDUCE*. If the efficiency of the reduction in i-th step is not expected to be well, it may be better to skip the reduction by *RIE* or *RIC*, because there is trade off between the size of data to join and the cost of selection operations for the reduction.

The order of the joins is also related to the performance. The properties of the reducing algorithms shown in Section 4 will be a great help to decide the order of the joins.

References

[1] Adiba, M. E. and Lindsay, B. G., "Database Snapshots," *Proc. Int. Conf. on Very Large Data Bases*, pp. 88–91, Oct. 1980.

[2] Blakeley, J. A., Coburn, N., and Larson, P., "Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates," *ACM Trans. on Database Syst.*, Vol. 14, No. 3, pp. 369–400, Sept. 1989.

[3] Blakeley, J. A., Larson, P., and Tompa, F. W., "Efficiently Updating Materialized Views," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 61–71, May 1986.

[4] Gupta, A., Mumick, I. S., and Subrahmanian, V. S., "Maintaing Views Incrementally," *Proc. ACM SIGMOD Int. Contf. on Management of Data*, pp. 157–166, 1993.

[5] Hanson, E. N., "A Performance Analysis of View Materialization Strategies," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 440–453, May 1987.

[6] Horowitz, S. and Teitelbaum, T., "Generating Editing Environment Based on Relations and Attributes," *ACM Trans. Programming Lang. Syst.*, Vol. 8, Oct. 1986.

[7] Hudson, S. and King, R., "The Cactis Project: Database Support for Software Environments," *IEEE Trans. Software Eng.*, Vol. 14, pp. 709–719, June 1988.

[8] Katz, R. and Chang, E., "Managing Change in CAD Databases," *Proc. Int. Conf. on Very Large Data Bases*, pp. 455–462, 1987.

[9] Konomi, S. and Furukawa, T., "Updating Duplicate Values in Distributed Multidatabase Systems," *Proc. IEEE Int. Workshop on Interoperability in Multidatabase Syst.*, pp. 243–246, April 1991.

[10] Konomi, S., Furukawa, T., and Kambayashi, Y., "Super-key Classes for Updating Materialized Derived Classes in Object Bases," *Proc. Int. Conf. on Deductive and Object-Oriented Databases, Lecture Notes in Computer Sciences*, Springer-Verlag, Vol. 760, pp. 310–326, Dec. 1993.

[11] Lindsay, B., Haas, L., Mohan, C., Pirahesh, H., and Wilms, P., "A Snapshot Differential Refresh Algorithm," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 53–60, May 1986.

[12] Liu, K. W. and Spooner, D., "Object-Oriented Database Views for Supporting Multidisciplinary Concurrent Engineering," *Proc. IEEE Computer Software and Applications Conf.*, pp. 19–25, Nov. 1993.

[13] Qian, X. and Wiederhold, G., "Incremental Recomputation of Active Relational Expressions," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 3, No. 3, pp. 337–341, Sept. 1991.

[14] Roussopoulos, N., "An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis," *ACM Trans. on Database Syst.*, Vol. 16, No. 3, pp. 535–563, Sept. 1991.

[15] Segev, A. and Park, J., "Updating Distributed Materialized Views," *IEEE Trans. Knowledge and Data Eng.*, Vol. 1, No. 2, pp. 173–184, June 1989.

[16] Stonebreaker, M., "Implementation of Integrity Constraints and Views by Query Modification," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 65–78, 1975.

[17] Zhuge, G., Garcia-Morina, H., Hammer, J., and Widom, J., "View Maintenace in a Warehousing Environment," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 316-327, May 1995.