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 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 \( R_1 \ast R_2 \ast R_3 \), \( V \) is the relation shown in Fig. 1 (b). Some tuples of \( V \) are produced by several tuples of \( R_1 \ast R_2 \ast R_3 \), for example \((a_2, b_2, c_2, d_2, e_2)\) by \((a_2, b_2, c_2, d_2, e_2), (a_3, b_3, c_2, d_2, e_2), \) 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 \( V \).

<table>
<thead>
<tr>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
<th>( V )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( B )</td>
<td>( B )</td>
<td>( C )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( b_2 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( b_2 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>( b_3 )</td>
<td>( b_3 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_4 )</td>
<td>( b_4 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_4 )</td>
<td>( b_4 )</td>
<td>( c_3 )</td>
</tr>
</tbody>
</table>

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.

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 \( R_1 \ast R_2 \) denote 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 \( C \), \( C_1 \land C_2 \land \cdots \land C_m \) for \( C = \{C_1, C_2, \ldots, C_m\} \). Although join is assumed to be natural join for simplicity in this paper, it can be easily extended to \( \theta \)-join with a slight modification. \( t[Y] \) is the \( Y \) value of tuple \( t \).

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

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.
Definition 1 Let a tuple \( t \) be deleted from \( R_1 \). The candidate relation and the existing relation of the update for view \( V = f(R) \) is \( V_C = f(\{\{t\}, R_2, \ldots, R_n\}) \) and \( V_E = f(\{R_1 - \{t\}, R_2, \ldots, R_n\}) \), 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 \( f(R) = \sigma_C \pi_X \sigma_Y (R_1 \ast R_2 \ast \ldots \ast R_n) \) 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 \( tv \) is a tuple of \( V - V_C \). There must be a tuple of \( R_1 \) other than \( t \) which derives \( tv \) because \( V_C \) is a set of tuples derived from \( t \); \( tv \) is derived from \( R - \{t\} \), that is \( tv \in V_E \). 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

![Diagram showing the relations V, V_E, V_T, and V_C]

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

<table>
<thead>
<tr>
<th>V_C</th>
<th>A</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_2</td>
<td>c_2</td>
<td>e_1</td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>c_2</td>
<td>e_2</td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>c_2</td>
<td>e_4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V_E</th>
<th>A</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>c_1</td>
<td>e_1</td>
<td></td>
</tr>
<tr>
<td>a_1</td>
<td>c_1</td>
<td>e_4</td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>c_2</td>
<td>e_2</td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>c_3</td>
<td>e_3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V_T</th>
<th>A</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_2</td>
<td>c_2</td>
<td>e_1</td>
<td></td>
</tr>
</tbody>
</table>

**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 $V_T$. \( \Box \)

**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 $V_T$, $(V'_C - V'_E) - V_T = (V'_C - V_C) - V'_E$. $V'_C \subseteq V_C$ and $V_T = V_C - V_E$ show that $V'_C - V'_E$ is a subset of $V_C \cap V_E$, which is equal to $V_F$. That is $(V'_C - V_T) - V'_E$ 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 $V_E (= V_E)$ 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 $V(X_V) = f(R)$, $t$ be a tuple of $R_1$, and $Y_1$ be $X_1 \cap X_V$.

1. $V'_C = \{t\}$, $V'_E = \sigma_{Y_1 \rightarrow t}(R_1 - \{t\})$
2. For $i = 2$ to $n$, $V'_C = V'_C - t \ast R_1$, $V'_E = V'_E - r_i$
3. $V_T = \sigma_{C \cap X_V} V'_C - \sigma_{C \cap X_V} V'_E$ \( \Box \)

**Theorem 1** The resulting relation $V_T$ of algorithm *NAIVE* is the target relation of the update which deletes $t$ from $R_1$. \( \Box \)
Figure 4: Property of the target relation

Proof: \( \sigma_{G \pi_Y V_C} V_C \) is the candidate relation \( V_C^a \) because \( V_C^a = \{ t \} * R_2 * \cdots * R_n \). \( \sigma_{G \pi_Y V_C} V_C^a = \sigma_{G \pi_Y V_C} (\sigma_{Y_1 \pi_{Y_1}} (R_1 \setminus \{ t \}) * R_2 * \cdots * R_n) = \sigma_{Y_1 \pi_{Y_1}} (R_1 \setminus \{ t \}) * R_2 * \cdots * R_n \subseteq V_E. \) Every \( Y_1 \) values of tuples of \( V_C \) is \( t[Y_1] \) because \( V_C = \sigma_{G \pi_Y V_C} (\{ t \} * R_2 * \cdots * R_n) \). Since \( V_F \subseteq V_G, Y_1 \) values of tuples of \( V_F \) is also \( t[Y_1]. \) \( V_F \subseteq \sigma_{G \pi_Y V_C} V_C^a \) because \( \sigma_{G \pi_Y V_C} V_E^a = \sigma_{Y_1 \pi_{Y_1}} V_E \) and \( V_F \subseteq V_E. \) Thus \( V_F \subseteq \pi_{X_Y V_C} V_E^a \subseteq V_E. \) Proposition 2 shows such relation \( V_T = \sigma_{G \pi_Y V_C} V_C^a - \pi_{X_Y V_C} V_E^a \) is the target relation.

Q.E.D.

Example 2  Fig. 5 shows the process to generate \( \sigma_{G \pi_Y V_C} V_C^a = \pi_{A C B C D} V_C^3 \) and \( \sigma_{G \pi_Y V_C} V_E^a = \pi_{A C B C D} V_E^3 \) by algorithm NAIVE.

![Figure 5: Delete tuple \((a_2, b_2)\) from \(R_1\) with algorithm NAIVE](image)

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.
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 $V_T \subseteq V'_E \subseteq V_E$ 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 $V'_E$ because there are cases in which some tuples in $V'_E$ that do not agree with tuples in $V'_C$ on the attributes in $X_U$ do not derive any tuples in $V_T$. For example $(a_2, b_4, c_3, d_3)$ of $V'_E$ does not derive the tuples in $\pi_{X_U} V'_C$ that intersect with tuples in $\pi_{\text{AC}} V'_E$ in Fig. 5. RIE is the algorithm to reduce such tuples in the same way as NAIVE reduces $V'_E$.

Algorithm 2 RIE (Reducing intermediate results of the Existing relation)

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

1. $V'_C = \{t\}$, $V'_E = \sigma_{Y_1 \in \{t\}}(R_1 - \{t\})$

2. For $i = 2$ to $n$, $V'_C = V'_{C}^{i-1} \ast R_i$, $V'_E = \sigma_{Y_1 \in \pi_{Y_1} V'_C} (V'_{E}^{i-1} \ast R_i)$

3. $V_T = \sigma_{\pi_{X_U} V'_C} - \sigma_{\pi_{X_U} V'_E}$

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 $\pi_{X_U} V'_E$ becomes $\sigma_{Y_1 \in \{t\}} \sigma_{Y_3 \in \pi_{Y_3} V'_E} \cdots \sigma_{Y_n \in \pi_{Y_n} V'_E}$.

Example 3 Fig. 6 shows the process to generate $V'_C = \pi_{\text{AC}} V'_E$ and $V'_E = \pi_{\text{AC}} V'_E$ by algorithm RIE.

There exist tuples with the same $Y_2$ value $(a_2, c_2)$ in $V'_C$ and $V'_E$ in Fig. 5. Since the tuples derived from $(a_2, b_2, c_2, d_2)$ of $V'_E$ and $(a_2, b_3, c_2, d_2)$ of $V'_E$ are the same tuples $(a_2, c_2, c_2)$ and $(a_2, c_2, e_4)$ of $V_C$ and $V_E$, we can get $V_T = \sigma_{\pi_{X_U} V'_C} - \sigma_{\pi_{X_U} V'_E}$ even if $(a_2, b_2, c_2, d_2)$ is deleted from $V'_C$. 

Figure 6: Delete tuple $(a_2, b_2)$ from $R_1$ with algorithm RIE
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 \((a_2, b_2, c_2, d_2)\) or \((a_2, b_4, c_2, d_2)\).

Note that this reduction cannot be applied to \(V_E^2\); 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 \(V_C^2\) agrees with a tuple \(t_C^j\) of \(V_C^2\) on the attributes in \(X_V\) and on the join attributes of later joins. \(t_C^i\) is still a candidate if \(t_C^j\) does not derive any tuple of \(V_E^2\) 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 \(t_C^j\) on the attributes which appear in selection condition, too, if we delete \(t_C^i\) from \(V_C^2\).

Algorithm \(RIC\) shows this reduction of the intermediate relations. Let \(J(R_i, R_j)\) be the join attributes of the join \(R_i \bowtie R_j\). If \(R_i\) or \(R_j\) is not defined, \(J\) is the empty set. Let \(\text{attr}(c)\) be the set of attributes which appear in selection condition \(c\), and \(\text{attr}(C)\) be \(\cup_{c \in C} \text{attr}(c)\) 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 \(f(R)\), \(t_i\) be a tuple of \(R_i\), \(X_V\) be \(U_{i=1}^{n} X_i \cap X_V\), and \(Z_i\) be \(Y_i \cup U_{k=i+1}^{n} J(R_k, R_{k+1}) \cup \text{attr}(C)\).

1. \(V_C^0 = \{\}\, V_E^1 = \sigma_{Y_i = E_i(R_i)}(R_i - \{\})\)
2. For \(i = 2\) to \(n, V_E^i = V_E^{i-1} \bowtie R_i, V_C^i = \sigma_{Z_i \cap \text{attr}(C)}(V_E^{i-1} \bowtie R_i)\)
3. \(V_T = \sigma_{C \cap \text{attr}(C)} V_C^n\)

**Lemma 2** The resulting relation \(V_T\) of algorithm \(RIC\)'s the target relation of the update which delete \(t\) from \(R_i\).

**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 \(t_C^i = \pi_{X_V}(\pi_{X_V}(t_C^i \bowtie \cdots \bowtie t_n)[R_{i-j}, i < j \leq n])\), \(t_E^i\) is joinable with \(t_{i+1} \bowtie \cdots \bowtie t_n\) 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 \(t_C^i \in V_C^i \bowtie \pi_{X_V}(t_C^i[Z_i] = t_E^i[Z_i])\), there exists a tuple \(t_E^i \in V_E^i\) such that \(t_C^i[X_V] = t_E^i[X_V]\). If \(t_C^i\) satisfies \(C, t_E^i\) also satisfies \(C\) because of \(\pi_{\text{attr}(C)} = \pi_{\text{attr}(C)}\). \(t_C^i[X_V]\) is a tuple of \(V_T\) because \(t_C^i[X_V]\) is a tuple of \(V_C\). If \(t_C^i\) does not satisfy \(C, t_E^i[X_V]\) 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, \(\sigma_{C \cap \text{attr}(C)} V_C^n\) is \(\sigma_{C \cap \text{attr}(C)} \pi_{X_V} \sigma_{X_V \cap \text{attr}(C)} V_E^{n-1} \bowtie R_n\) - \(\sigma_{C \cap \text{attr}(C)} V_C^n\), which is a set of tuples in a subset of \(V_C\) but not in \(\sigma_{C \cap \text{attr}(C)} V_C^n\). Therefore \(\sigma_{C \cap \text{attr}(C)} V_C^n\) is a superset of the target relation and does not include any tuple of \(V_E^n\), 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 \(V_C\) which will derive only tuples in \(V_T\) but not tuples in \(V_T\) and stops the derivation from such tuples. It often occurs that \(V_C^2\) becomes the empty set, that is \(V_T\) is empty, which allows to quit the materialized view update without accessing \(R_k\) (\(i < k \leq n\)).

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
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 $V(X_Y)$ be defined by $f(R)$, $t$ be a tuple of $R_1$, $Y_i$ be $\bigcup_{j=1}^{i} X_i \cap X_Y$, and $Z_i$ be $Y_i \cup \bigcup_{k=1}^{i} \left( \bigcup_{k=1}^{n} \{R_k, R_{k+1}\} \bigcup \text{attr}(C) \right)$.

1. $V_E^0 = \{t\}$, $V_E^i = \sigma_{Y_i \in Y_Y}(R_1 - \{t\})$
2. For $i = 2$ to $n$, $V_E^i = V_E^{i-1} \ast R_i$, $V_C^i = \sigma_{Z_i \in Z_Y} V_C^{i-1} \ast R_i$, $V_C^i = \sigma_{Y_i \in Y_Y} V_C^i V_E^i$
3. $V_T = \sigma_{C \in X_Y} V_C^n$

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 $Y_i \in \pi_X V_C^i$ in $i$-th step will not effect the reduction of $V_E^k$ ($i < k \leq n$) 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 $\sigma_{C \in X_Y} V_C^n$ in RIC is the target relation, $\sigma_{C \in X_Y} V_C^n$ 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 $X_Y$ 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] = $
Figure 8: Delete tuple \((a_2, b_2)\) from \(R_1\) with algorithm \textsc{Reduce}

\[
\begin{array}{ccc}
V_C^1 & V_C^2 & \pi_{ACE} V_C^3 \\
A & B & A & C & E \\
a_2 & b_2 & a_2 & b_2 & a_2 \\
b_2 & & c_2 & d_2 & c_2 \\
h_3 & & d_2 & e_2 & d_2 \\
h_3 & & e_2 & & c_2 \\
\end{array}
\]

\[
\begin{array}{ccc}
V_E^1 & V_E^2 & \pi_{ACE} V_E^3 \\
A & B & A & C & D & E \\
a_2 & b_3 & a_2 & b_2 & a_2 & a_2 \\
b_3 & c_2 & b_4 & c_2 & b_4 & b_2 \\
h_3 & c_2 & h_3 & c_2 & h_3 & c_2 \\
h_3 & c_2 & h_3 & c_2 & h_3 & c_2 \\
\end{array}
\]

By Theorem 3, \(V_b \ast R_{i+1} \ast \ldots \ast R_n\) becomes the target relation if \(X_V\) includes the key of \(R_4\). 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 \(V_T\) 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 \(R_1\), \(t^i_\text{C}[Y_i \cup X_V]\) 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 \(\sigma_{X_1 \cup X_V = t[X_1 \cup X_V]} V\), that is, \(\sigma_{X_1 \cup X_V = t[X_1 \cup X_V]} V\) 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 \textsc{Naive}, \textsc{Rie}, \textsc{Ric}, and \textsc{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 \(t^i_\text{C}\) of \(V_C^i\), if there is a tuple \(t^i_\text{E}\) of \(V_E^i\) such that \(t^i_\text{C}[Y_i \cup X_V] = t^i_\text{E}[Y_i \cup X_V]\), \(t^i_\text{C}\) does not derive any tuple of \(V_T\) because \(t^i_\text{E}\) derives the same tuple as \(t^i_\text{C}\) derives. If there is no tuple \(t^i_\text{E}\) of \(V_E^i\) such that \(t^i_\text{C}[Y_i \cup X_V] = t^i_\text{E}[Y_i \cup X_V]\), the tuples derived from \(t^i_\text{C}\) is tuples of \(V_T\) because the tuples derived from tuples of \(V_E^i\) do not agree with them on \(Y_i \cup X_V\). That is, the tuples of \(V_C\) whose \(Y_i \cup X_V\) values are in \(V_C^i[Y_i \cup X_V] - V_E^i[Y_i \cup X_V]\) are tuples of \(V_T\). Every tuple of \(V - V_C\) has a different value of \(Y_i \cup X_V\) from the tuples of \(V_T\) because \(Y_i \cup X_V\) is a superset of the key of \(V\). Therefor the set of the tuples of \(V\) whose \(Y_i \cup X_V\) values are in \(V_C^i[Y_i \cup X_V] - V_E^i[Y_i \cup X_V]\) 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 \( \{ c | c \in C \cup \bigcup_{j=1}^{i-1} C_j, \text{attr}(c) \subseteq \bigcup_{j=1}^{i} X_j \} \), 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: \( V_0^C = \sigma_{C_1}(\{t\}), V_0^B = \sigma_{C_1}(\sigma_{Y_{V_1}}(R_1 - \{t\})) \)
2. the \( i \)-th step (\( 2 \leq i \leq n \)): \( V_E^{i-1} \ast R_i \) and \( V_C^{i-1} \ast R_i \) are replaced with \( \sigma_{C_i}(V_E^{i-1} \ast R_i) \) and \( \sigma_{C_i}(V_C^{i-1} \ast R_i) \), respectively.
3. \( V_T: \pi_{X_Y} V_C^a - \pi_{X_Y} V_B^a \) in \textit{NAIVE} and \textit{RIE}, \( \pi_{X_Y} V_C^a \) in \textit{RIC} and \textit{REDUCE}

Tuples \( (a_2, b_3, c_2, d_2) \) and \( (a_2, b_4, c_2, d_2) \) of \( V_2^B \) in Fig. 8 derive the same tuples of \( \pi_{ACE} V_2^B \).

Also there are cases that some tuples of \( V_2^B \) derive the same tuple of \( V_C^a \).

It is enough to keep only one tuple of the tuples of \( V_C^a \) and \( V_C^b \) deriving the same set of tuples of \( \sigma_{C_1} \pi_{X_Y} V_C^a \) and \( \sigma_{C_1} \pi_{X_Y} V_B^a \), 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 \( \bigcup_{j=1}^{i} X_j \cap X_Y \), they can have different values of other than the attributes. If the values of the join attributes of \( R_k \ast R_{k+1}(k \leq i) \) are different from each other, we have to keep them. For example, the two tuples of \( V_2^B \) in Fig. 5 derive the different sets of tuples of \( \pi_{ACE} V_2^B \), which have the same value \( (a_2, c_2, e_2) \) on \( AC = \bigcup_{j=1}^{2} X_j \cap ACE \), \{\( (a_2, c_2, e_1), (a_2, c_2, e_4) \)\} and \{\( (a_2, c_2, c_2), (a_2, c_2, e_4) \)\}, respectively. Furthermore attributes used by selection conditions must be kept, too.

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

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

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

### Modifying the reducing algorithms for projection

1. the first step: \( V_0^C = \pi_{Z_1}(\{t\}), V_0^B = \pi_{Z_1}(\sigma_{Y_{V_1}}(R_1 - \{t\})) \)
2. the \( i \)-th step (\( 2 \leq i \leq n \)): \( V_E^{i-1} \ast R_i \) and \( V_C^{i-1} \ast R_i \) are replaced with \( \pi_{Z_i}(\pi_{E^{i-1}} \ast R_i) \) and \( \pi_{Z_i}(\pi_{C^{i-1}} \ast R_i) \), respectively.
3. \( V_T: \sigma_{C} V_C^a - \sigma_{C} V_B^a \) in \textit{NAIVE} and \textit{RIE}, \( \sigma_{C} V_C^a \) in \textit{RIC} and \textit{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 \( \text{attr}(C_j)(1 \leq j \leq i) \) but not in \( \text{attr}(C_k)(i < k \leq n) \) are no longer needed. \( Z_i \) is \( Y_i \cup \bigcup_{k=i+1}^{n} J(R_k, R_{k+1}) \cup \bigcup_{k=i+1}^{n} \text{attr}(C_k) \) in this case, and the algorithms are modified as follows.

### Modifying the reducing algorithms for selection and projection

1. the first step: \( V_0^C = \pi_{Z_1} \sigma_{C_1}(\{t\}), V_0^B = \pi_{Z_1} \sigma_{C_1}(\sigma_{Y_{V_1}}(R_1 - \{t\})) \)
2. the \( i \)-th step (\( 2 \leq i \leq n \)): \( V_E^{i-1} \ast R_i \) and \( V_C^{i-1} \ast R_i \) are replaced with \( \pi_{Z_i} \sigma_{C_i}(\pi_{E^{i-1}} \ast R_i) \) and \( \pi_{Z_i} \sigma_{C_i}(\pi_{C^{i-1}} \ast R_i) \), respectively.
3. \( V_T: \sigma_{C} V_C^a - \sigma_{C} V_B^a \) in \textit{NAIVE} and \textit{RIE}, \( \sigma_{C} V_C^a \) in \textit{RIC} and \textit{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 $f(R)$, $t$ be a tuple of $R_1$, $V_i$ be $U_{i-1}X_j \cap X_V$, and $Z_i$ be $Y_i \cup \bigcup_{k=i}^{n+1} \{R_k, R_{k+1}\} \cup \bigcup_{k=i}^{n+1} \text{attr}(C_k)$.

1. $V_C^1 = \pi_{Z_1} \sigma_{C_1}(t)$, $V_E^1 = \pi_{Z_1} \sigma_{C_1}(V_{E'}^{i-1} \ast R_i)$, $V_C^2 = \pi_{Z_1} \sigma_{C_2} \pi_{Z_2} \sigma_{Y_1 = \pi_{Y_1}(R_1 - \{t\})}$
2. For $i = 2$ to $n$, $V_C^i = \pi_{Z_1} \sigma_{C_i} (V_E^{i-1} \ast R_i)$, $V_E^i = \pi_{Z_1} \sigma_{C_i} \pi_{Z_2} \sigma_{Y_1 = \pi_{Y_1}(R_1 - \{t\})}$, $V_C^i = \pi_{Z_1} \sigma_{C_i} \pi_{Z_2} \sigma_{Y_1 = \pi_{Y_1}(R_1 - \{t\})}$
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 $\sigma_{C_i}$ 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 $V_{E'}^{i-1} \ast R_i$ and $V_{C'}^{i-1} \ast R_i$ which do not satisfy $C_i$ do not derive any tuple of $V_E$ and $V_C$, respectively.

Projection $\pi_{Z_2}$ in $i$-th step does not affect the later selection because $V_E^i$ and $V_C^i$ keep the attributes which appear in $C_k$ or $Y_k$ ($i < k \leq n$).

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.

\begin{figure}[h]
\centering
\begin{tabular}{c|c}
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
</tbody>
</table>
\end{tabular}
\quad
\begin{tabular}{c|c|c}
<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
<td>$d_1$</td>
</tr>
</tbody>
</table>
\end{tabular}
\quad
\begin{tabular}{c|c|c|c}
<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
<td>$e_1$</td>
</tr>
</tbody>
</table>
\end{tabular}
\quad
\begin{tabular}{c|c|c|c|c|c}
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_4$</td>
</tr>
</tbody>
</table>
\end{tabular}
\quad
\begin{tabular}{c|c|c|c|c}
<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>
\end{tabular}
\quad
\begin{tabular}{c|c|c|c|c|c}
<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>
\end{tabular}
\caption{Delete tuple $(a_2, b_2)$ from $R_1$ with algorithm $REDUCE^+$}
\end{figure}

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 $V_G = \sigma_{G \pi_X(y)}(\{t\} \ast R_2 \ast \cdots \ast R_n)$ 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 $V = f(R)$ 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](image)

**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\textsuperscript{+} 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 \( V_R = V \cup V_T = V \cup V_C \).

By Lemma 3, we can get \( V_R \) without \( V'_C \) as \( V \cup V'_C \). This property is quite different from tuple deletion. Lemma 3 allows to cut the computation of \( V'_C \) at some step in each algorithm. That is, the algorithms can stop at \( i \)-th step and then \( V'_C = \sigma_{C \pi_X} V_C \ast R_{i+1} \ast \cdots \ast R_n \) is added to \( V \). Since algorithms RIE, RIC, and REDUCE\textsuperscript{+} 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 \( Y = \pi_Y T \) in Step 1 to \( Y_1 \in \pi_Y T \) 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** \( V_{C_I} = f((S_1, R_2, \cdots, R_n)) \) 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 \( (V - V_{S_D}) \cup V_{T_I} = \sigma_{C \pi_X} ((R_1 - S_D) \cup S_I) \ast R_2 * \cdots * R_n \).

**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 $V_{TI}$ and $V_{TD}$, 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 $f(R)$ 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 $V_T$ and $V_{TD}$ by algorithms NAIVE, RIE, RIC, or REDUCE.
2. Let $V_R$ be $(V - V_{TD}) \cup V_T$.

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

Proof: $V_T$ and $V_{TD}$ in algorithm MU1 are the target relations of insertion and deletion, respectively. Lemma 4 shows $V_R = (V - V_{TD}) \cup V_T$ in algorithm MU1 is the view after the update of R1.

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 $V_R = (V - V_{TD}) \cup V_T$ by Lemma 4, $(V \cup V_T) - V_{TD}$ cannot be $V_R$ because there can be a tuple $t$ in $V_R$ such that $t \in V_T$ and $t \notin V_{TD}$. Such $t$ is derived from both $S_I$ and $S_D$ and is not in $(V \cup V_T) - V_{TD}$. 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 $V_T$ as a part of the existing relation of the deletion.

Algorithm 7 MU2 (Multiple Updates 2)
Let a view $V(X_V)$ be defined by $f(R)$, $S_I$ and $S_D$ be sets of tuples inserted to and deleted from $R_1$ ($S_I \cap S_D = \emptyset$), respectively, $Y_i$ be $\cup_j X_j \cap X_V$, and $Z_i$ be $Y_i \cup \cup_{j=1}^{n} I(R_i, R_{i+1}) \cup attr(C)$.
1. $V_{C_I} = S_I$, $V_{E_I} = \sigma_{Y_i \in \sigma_{Z_i}} S_I R_1$, $V_{C_D} = S_D$, $V_{E_D} = \sigma_{Y_i \in \sigma_{Z_i}} S_D R_1$
2. For $i=2$ to $n$, $V_{E_I} = V_{E_i} \ast R_i$, $V_{C_I} = \sigma_{Z_i \in \sigma_{Z_i}} (V_{E_i} \ast R_i)$, $V_{E_i} = \sigma_{Y_i \in \sigma_{Y_i}} V_{C_I}$, $V_{E_D} = V_{E_i} \ast R_i$, $V_{C_D} = \sigma_{Z_i \in \sigma_{Z_i}} (V_{E_D} \ast R_i)$, $V_{E_D} = \sigma_{Y_i \in \sigma_{Y_i}} V_{C_D}$
3. $V_{TI} = \sigma_{\ast_{X_V} X_V V_{C_I} \ast_{C_I} \ast_{D}}$, $V_{TD} = \sigma_{\ast_{X_V} X_V V_{C_D} \ast_{C_D} \ast_{D}}$

Theorem 8 For $V_{TI}$ and $V_{TD}$ of algorithm MU2, $(V_{TI} \cup V_{TD})$ is the resulting relation of the update, insertion of $S_I$ to $R_1$ and deletion of $S_D$ from $R_1$.

Proof: $V_{TI}$ is the target relation of the insertion because the process to produce $V_T$ is the same as REDUCE. Obviously $V_{TD} \subseteq V_{TD}$. Let $T_D$ be $V_{TD} - V_{TD}$. The tuples of $T_D$ are caused by the selection $\sigma_{Z_i \in \sigma_{Z_i}} V_{C_I}$ to reduce $V_{CD}$ 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 $(V - V_{TD}) \cup V_{TI}$ is equal to $(V - V_{TD}) \cup V_{TI}$, which is the resulting relation $V_R$.

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


