Optimizing the Incremental Maintenance of Multiple Join Views *

Ki Yong Lee
Korea Advanced Institute of Science and Technology
373-1, Guseong-Dong, Yuseong-Gu
Daejeon, South Korea
kylee@dbserver.kaist.ac.kr

Myoung Ho Kim
Korea Advanced Institute of Science and Technology
373-1, Guseong-Dong, Yuseong-Gu
Daejeon, South Korea
mhkim@dbserver.kaist.ac.kr

ABSTRACT

Materialized views are nowadays commonly used in the data warehouse environment. Materialized views need to be updated when data sources change. Since the update of the views may impose a significant overhead, it is essential to update the views efficiently. Though there has been much work on efficient maintenance of a single view, maintenance of multiple views has not been sufficiently investigated.

In this paper we propose an efficient incremental maintenance of multiple join views. In our previous work[6], we proposed the *delta propagation strategy* that computes the change of a join view in a recursive manner. We extend the delta propagation strategy to multiple views. The recursive property of the strategy makes it possible to share common intermediate results among views effectively. We first define the multiple view maintenance problem, then a heuristic algorithm that finds a global maintenance plan for the given views is proposed. We also present experimental result that shows the efficiency of the proposed method.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Data Warehouse and repository*

General Terms

Performance

Keywords

Incremental View Maintenance, Multiple View Maintenance

1. INTRODUCTION

Materialized views are nowadays commonly used in the data warehouse environment to support on-line analytical processing (OLAP). Materialized views are generally defined over several base relations. When base relations are changed, the materialized views defined by the base relations must also be updated. Since queries calling for up-to-date information has been increasing, an efficient view maintenance strategy has been an important issue in data warehouse application.

Materialized views can be maintained by either recomputation or incremental maintenance. Incremental maintenance computes and propagates only the change of the views. For example, consider a materialized view $V = A \bowtie B$. If the changes to A and B are given as ΔA and ΔB respectively, the change of V, i.e., ΔV, can be computed by $\Delta V = (\Delta A \bowtie B) \cup (A \bowtie \Delta B) \cup (\Delta A \bowtie \Delta B)$, or $\Delta V = (\Delta A \bowtie B) \cup (A' \bowtie \Delta B)$, where $A' = A \cup \Delta A$. When the amounts of changes are much smaller than the sizes of base relations and views, incremental maintenance is usually much cheaper than recomputation. Thus, many possible methods that allow incremental view maintenance have been proposed in the past[1][2][3][4][5][6][7][8].

In general, there can be more than one view in the data warehouse. These views often share some common expressions among them. If the intermediate results generated during maintaining some views can be reused to maintain other views, we may significantly reduce the overall maintenance cost. Suppose that we have two materialized views $V_1 = A \bowtie B \bowtie C$ and $V_2 = B \bowtie C \bowtie D$, and the changes to A, B, C, and D are given as ΔA, ΔB, ΔC, and ΔD respectively. Note that the expression $B \bowtie C$ is shared between V_1 and V_2. In our previous work[6], we proposed an efficient incremental view maintenance method that computes the change of a view in a recursive manner. The proposed method is called the *delta propagation strategy*. If we apply the delta propagation strategy, ΔV_1 and ΔV_2 can be computed by the following expressions.

$$\begin{aligned} \Delta V_1 &= (\Delta A \bowtie B \bowtie C) \cup (A' \bowtie \Delta(B \bowtie C)) \\ \Delta V_2 &= (\Delta(B \bowtie C) \bowtie D) \cup (B' \bowtie C' \bowtie \Delta D) \end{aligned}$$

Here, $A' = A \cup \Delta A$, $B' = B \cup \Delta B$, and $C' = C \cup \Delta C$. $\Delta(B \bowtie C)$ represents the change of $B \bowtie C$, i.e., $\Delta(B \bowtie C) = (\Delta B \bowtie C) \cup (B' \bowtie \Delta C)$. In the above example, the change of a common subexpression of V_1 and V_2, i.e., $\Delta(B \bowtie C)$, is computed first before ΔV_1 and ΔV_2 are computed. We can now see that $\Delta(B \bowtie C)$ is shared between ΔV_1 and ΔV_2. If we reuse $\Delta(B \bowtie C)$ in computing ΔV_1 and ΔV_2, we

*This work was supported in part by grant No. (R01-2003-000-10627-0) from the Basic Research Program of the Korea Science & Engineering Foundation.

DOLAP'05, November 4–5, 2005, Bremen, Germany.

may reduce the overall cost of maintaining V_1 and V_2. Note that this is possible because the delta propagation strategy computes ΔV_1 and ΔV_2 in a recursive way.

In this paper, we address the problem of optimizing the incremental maintenance of a set of materialized views. We showed in [6] that the delta propagation strategy can maintain a view efficiently by computing the change of the view recursively. We extend the delta propagation strategy to multiple views in this paper. We define the multiple view maintenance problem, and present a heuristic algorithm that selects the intermediate results to be reused in a greedy manner.

The remainder of the paper is organized as follows. Related work is outlined in Section 2. In Section 3, we present our materialized view model, and it cost model for performance comparison of various view maintenance expressions. Our previous work for a single view[6] is described briefly in Section 4. Section 5 proposes our incremental maintenance method for multiple views. Section 6 gives results of performance experiment. Finally, we conclude our work in Section 7.

2. RELATED WORK

There has been much work on incremental maintenance of various kinds of materialized views [1][10][12][13][14][15]. In this paper, we will focus on join views that have been considered the most widely used in data warehouse applications. There have been a few methods for incremental maintenance of a join view. For an n-way join view $V = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, [1][4] proposed the following maintenance expression to compute ΔV.

$$\begin{aligned} \Delta V &= (\Delta R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n) \\ &\quad \cup (R_1 \bowtie \Delta R_2 \bowtie \cdots \bowtie R_n) \\ &\quad \cup \cdots \\ &\quad \cup (\Delta R_1 \bowtie \Delta R_2 \bowtie \cdots \bowtie \Delta R_n) \end{aligned} \tag{1}$$

Expression (1) consists of $(2^n - 1)$ terms (i.e., $\Delta R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, $R_1 \bowtie \Delta R_2 \bowtie \cdots \bowtie R_n$, $\cdots$, $\Delta R_1 \bowtie \Delta R_2 \bowtie \cdots \bowtie \Delta R_n$). Since expression (1) requires too many terms, the following maintenance expression consisting of only n terms has been proposed in [2][5].

$$\begin{aligned} \Delta V &= (\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie \cdots \bowtie R_n) \\ &\quad \cup (R_1^{'} \bowtie \Delta R_2 \bowtie R_3 \bowtie \cdots \bowtie R_n) \\ &\quad \cup \cdots \\ &\quad \cup (R_1^{'} \bowtie R_2^{'} \bowtie \cdots \bowtie R_{n-1}^{'} \bowtie \Delta R_n) \end{aligned} \tag{2}$$

Here, $R_i^{'}$ represents $R_i \cup \Delta R_i$. Expression (2) is equivalent to expression (1), but has only n terms. Roughly speaking, the term $R_1^{'} \bowtie \cdots \bowtie R_{i-1}^{'} \bowtie \Delta R_i \bowtie R_{i+1} \bowtie \cdots \bowtie R_n$ represents the change of V due to ΔR_i after $R_1, R_2, \cdots, R_{i-1}$ have been updated. Because of its simplicity, expression (2) has been widely used to compute the change of a join view.

However, even in expression (2), each base relation R_i (or $R_i^{'}$) appears in $(n-1)$ terms repeatedy. Each base relation, therefore, needs to be accesses $(n-1)$ times to evaluate expression (2). Hence, we proposed a new incremental maintenance method for a join view to further reduce the number of accesses to base relations in [6]. Our proposed method in [6] is called the *delta propagation strategy*. The delta propagation strategy computes the change of a view in a recursive manner. For example, consider $V = R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$. One of the maintenance expressions used by the delta propagation strategy for V is as follows.

$$\begin{aligned} \Delta V &= (\Delta(R_1 \bowtie R_2) \bowtie R_3 \bowtie R_4) \\ &\quad \cup (R_1^{'} \bowtie R_2^{'} \bowtie \Delta(R_3 \bowtie R_4)), \end{aligned} \tag{3}$$

where $\Delta(R_1 \bowtie R_2) = (\Delta R_1 \bowtie R_2) \cup (R_1^{'} \bowtie \Delta R_2)$ and $\Delta(R_3 \bowtie R_4) = (\Delta R_3 \bowtie R_4) \cup (R_3^{'} \bowtie \Delta R_4)$. That is, the delta propagation strategy partitions $\{R_1, R_2, R_3, R_4\}$ into $\{R_1, R_2\}$ and $\{R_3, R_4\}$, then it computes $\Delta(R_1 \bowtie R_2)$ and $\Delta(R_3 \bowtie R_4)$ first before it computes ΔV. Note that compared with expression (2), the number of accesses to R_1, R_2, R_3, and R_4 is reduced to 2 respectively in expression (3). Besides expression (3), various expressions can be generated according to how $\{R_1, R_2, R_3, R_4\}$ is partitioned. More detail on the delta propagation strategy will be described in Section 4.

Maintenance of multiple views is closely related to multiple query optimization[17][18]. However, applying multiple query optimization techniques directly to multiple view maintenance would be very expensive, since incremental maintenance expressions are generally quite complex. To our best knowledge, [7] is the only work that has addressed the problem of optimizing the maintenance of multiple views. They applied their previous work on multi-query optimization [16] to exploit common subexpressions among view maintenance expressions. However, their work is based on expression (2), which requires more number of accesses to base relations than the expressions considered by the delta propagation strategy.

3. PRELIMINARIES

3.1 View Definition Model

A materialized view can be variously defined over base relations or other materialized views or both. In this paper we consider materialized views defined by the join of base relations. Join views can be easily extended to select-project-join views, which cover most of materialized views in the data warehouse environment. Moreover, a join view can be easily extended to accommodate a view with aggregations using generalized projections[10].

A join view V over n base relations $R_1, R_2, \cdots, R_n$ is defined as follows:

$$V = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$$

For a base relation R, the change of R are denoted by ΔR. $R^{'}$ represents the relation R to which ΔR is propagated, i.e., $R^{'} = R \cup \Delta R$. The formal description of ΔR and the union operator can be found in [2] and [9].

3.2 Cost Model

In this paper we adopt the linear work metric developed in [5] as a cost model to compare view maintenance expressions. Although the linear work metric is relatively simple, the cost of processing complex maintenance expressions can be effectively estimated[5]. In the linear work metric, the cost of processing a maintenance expression is the sum of the costs of processing each term of the expression, and the cost of processing a term is proportional to the sum of the sizes of the operands of the term. Let $Cost(E)$ be the cost

of processing the expression E, then the cost of computing $\Delta V = (\Delta R_1 \bowtie R_2) \cup (R_1^{'} \bowtie \Delta R_2)$ can be obtained as follows:

$$\begin{aligned} Cost(\Delta V) &= Cost((\Delta R_1 \bowtie R_2) \cup (R_1^{'} \bowtie \Delta R_2)) \\ &= Cost(\Delta R_1 \bowtie R_2) + Cost(R_1^{'} \bowtie \Delta R_2) \\ &= c \cdot (|\Delta R_1| + |R_2|) + c \cdot (|R_1^{'}| + |\Delta R_2|) \end{aligned}$$

Here, c is a constant and $|R|$ is the size of relation R.

4. MAINTAINING A SINGLE VIEW

In this section, we briefly describe the delta propagation strategy proposed in [6].

4.1 Delta Propagation Strategy

The delta propagation strategy is an incremental view maintenance method that uses the *delta expression* to compute the change of a join view. The delta expression is defined as follows.

DEFINITION 4.1. *Consider a view $V = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$. Let $\{P_1, P_2, ..., P_m\}$ be a partition of $\{R_1, R_2, ..., R_n\}$ such that $\bigcup_{i=1}^{m} P_i = \{R_1, R_2, ..., R_n\}$, $P_i \cap P_j = \phi$ $(1 \le i \ne j \le m)$, and $P_i \ne \phi$ $(1 \le i \le m)$. Then the delta expression for V representing ΔV is defined as follows:*

$$\begin{aligned} \Delta V = {} & (\Delta(\bowtie P_1) \bowtie (\bowtie P_2) \bowtie \cdots \bowtie (\bowtie P_m)) \\ & \cup ((\bowtie P_1^{'}) \bowtie \Delta(\bowtie P_2) \bowtie \cdots \bowtie (\bowtie P_m)) \\ & \cup \cdots \\ & \cup ((\bowtie P_1^{'}) \bowtie (\bowtie P_2^{'}) \bowtie \cdots \bowtie \Delta(\bowtie P_m)), \end{aligned}$$

where $\bowtie P_i$ and $\bowtie P_i^{'}$ denote $R_s \bowtie R_t \bowtie \cdots \bowtie R_u$ and $R_s^{'} \bowtie R_t^{'} \bowtie \cdots \bowtie R_u^{'}$ respectively for $P_i = \{R_s, R_t, \cdots, R_u\}$. □

In this paper, the changes of join views are computed by the delta expression. The delta expression computes $\Delta(\bowtie P_1), \Delta(\bowtie P_2), ..., \Delta(\bowtie P_m)$ first, and then finally computes the overall change of V by using the results of $\Delta(\bowtie P_1), \Delta(\bowtie P_2), ..., \Delta(\bowtie P_m)$. Computation of $\Delta(\bowtie P_i)$ $(1 \le i \le m)$ can proceed by applying the same strategy recursively. That is, the expression is recursively expanded until $\Delta(\bowtie P_i)$ $(1 \le i \le m)$ is resolved into ΔR_j for some j $(1 \le j \le n)$. Depending on the choices of $P_1, P_2, \cdots, P_m$, there can be many delta expressions for a view. Figure 1 shows some possible delta expressions for $V = R_1 \bowtie R_2 \bowtie R_3$ according to the choices of $P_1, P_2, \cdots, P_m$.

A delta expression can be represented as a tree, which we call a *delta evaluation tree*. The root node of a delta evaluation tree for a view V is ΔV. According to Definition 4.1, $\Delta(\bowtie P_1), \Delta(\bowtie P_2), ..., \Delta(\bowtie P_m)$ become the children of the node ΔV with $\Delta(\bowtie P_i)$ $(1 \le i \le m)$ being the ith child from the left. $\Delta R_1, \Delta R_2, ..., \Delta R_n$ become the leaf nodes of the tree. Figure 2 shows the delta evaluation trees representing the delta expressions in Figure 1.

4.2 Optimal Plan for a Single View

There can be many possible delta propagation trees for a join view. The optimal delta propagation tree means one whose corresponding delta expression has the minimal cost with respect to the cost model described in Section 2.2. In this section, we describe briefly how to find the optimal delta propagation tree.

(a) $P_1 = \{R_1\},\ P_2 = \{R_2\},\ P_3 = \{R_3\}$:
$\Delta V = (\Delta R_1 \bowtie R_2 \bowtie R_3) \cup (R_1^{'} \bowtie \Delta R_2 \bowtie R_3) \cup (R_1^{'} \bowtie R_2^{'} \bowtie \Delta R_3)$

(b) $P_1 = \{R_1, R_2\},\ P_2 = \{R_3\}$:
$\Delta V = (\Delta(R_1 \bowtie R_2) \bowtie R_3) \cup (R_1^{'} \bowtie R_2^{'} \bowtie \Delta R_3)$

(c) $P_1 = \{R_1\},\ P_2 = \{R_2, R_3\}$:
$\Delta V = (\Delta R_1 \bowtie R_2 \bowtie R_3) \cup (R_1^{'} \bowtie \Delta(R_2 \bowtie \Delta R_3))$

(d) $P_1 = \{R_1, R_3\},\ P_2 = \{R_2\}$:
$\Delta V = (\Delta(R_1 \bowtie R_3) \bowtie R_2) \cup (R_1^{'} \bowtie R_3^{'} \bowtie \Delta R_2)$

Figure 1: Some possible delta expressions for $V = R_1 \bowtie R_2 \bowtie R_3$

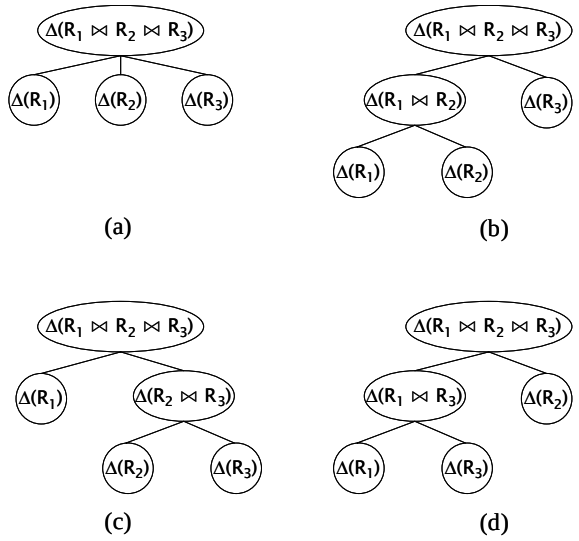

Figure 2: The delta evaluation trees representing the delta expressions in Figure 1

The cost of a delta propagation tree can be obtained by applying the cost model in Section 2.2 to the delta expression represented by the tree as follows.

$$\begin{aligned} & Cost(\Delta R_i) = 0 \\ & Cost(\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)) = \\ & \qquad \sum_{i=1}^{m} Cost(\Delta(\bowtie P_i)) \\ & \qquad + \ (|\Delta(\bowtie P_1)| + |P_2| + \cdots + |P_m|) \\ & \qquad + \ (|P_1^{'}| + |\Delta(\bowtie P_2)| + \cdots + |P_m|) \\ & \qquad + \ \cdots \\ & \qquad + \ (|P_1^{'}| + |P_2^{'}| + \cdots + |\Delta(\bowtie P_m)|), \end{aligned} \tag{4}$$

where $|P_i|$, $|P_i^{'}|$, and $|\Delta(\bowtie P_i)|$ denote $|R_s| + |R_t| + \cdots + |R_u|$, $|R_s^{'}| + |R_t^{'}| + \cdots + |R_u^{'}|$, and $|\Delta(R_s \bowtie R_t \bowtie \cdots \bowtie R_u)|$ respectively for $P_i = \{R_s, R_t, \cdots, R_u\}$. Here, we assume that the

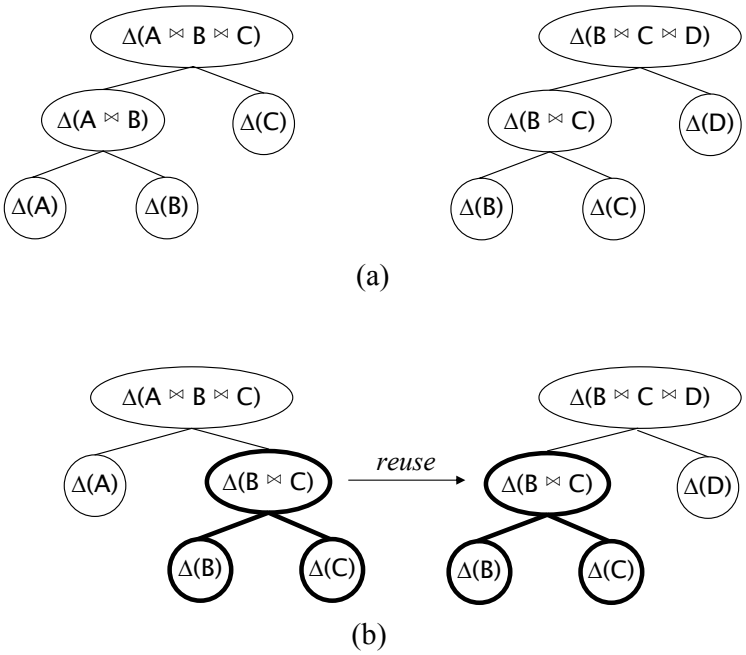

Figure 3: Reusing common intermediate results

constant c is equal to 1 in the linear work metric, for simplicity. The optimal delta evaluation expression is the one that minimizes expression (4). To find the optimal solution of expression (4), we developed a dynamic programming algorithm in [6]. See [6] for more detail on the proposed algorithm. Using the proposed algorithm in [6], we can find the optimal delta propagation tree for a singe join view.

5. MAINTAINING MULTIPLE VIEWS

So far, we have described a strategy for maintaining a single view. However, in general, there can be more than one view in the data warehouse. These views often share some common expressions among them. If the intermediate results generated during maintaining some views can be reused to maintain other views, we may significantly reduce the overall maintenance cost. Recall that, in Definition 4.1, the delta expression computes $\Delta(\bowtie P_1), \Delta(\bowtie P_2), ..., \Delta(\bowtie P_m)$ first to compute the overall change of V. Thus, if we can reuse $\Delta(\bowtie P_i)$ to maintain other views, we may reduce the overall cost of maintaining views even more. In this section, we present an algorithm that reuses common intermediate results to maintain more than one view. The following is an example of sharing intermediate results.

Consider $V_1 = A \bowtie B \bowtie C$ and $V_2 = B \bowtie C \bowtie D$. Figure 3-(a) shows the optimal delta evaluation trees for V_1 and V_2 when ΔV_1 and ΔV_2 are computed separately. These are the *locally* optimal trees for V_1 and V_2. However, if we reuse $\Delta(B \bowtie C)$, it may lead to the *globally* optimal plan. Figure 3-(b) shows the *globally* optimal trees for V_1 and V_2 that share $\Delta(B \bowtie C)$. Note that reusing common results may not always lead to a globally optimal strategy. If $|\Delta(B \bowtie C)|$ is considerably larger than $|\Delta(A \bowtie B)|$, reusing $\Delta(B \bowtie C)$ may not be a good plan. Thus, we need to decide what intermediate results should be materialized and reused. We will call this problem the *multiple view maintenance problem.*

5.1 Problem Formulation

Let $V = \{V_1, V_2, ..., V_n\}$ be a set of views to be maintained, and let $MinCost(\Delta V_i)$ denote the cost of the optimal delta evaluation tree for V_i. If we do not reuse any intermediate results, the total cost $TotalCost(V)$ of computing $\Delta V_1, \Delta V_2, ..., \Delta V_n$ is as follows:

$$TotalCost(V) = \sum_{V_i \in V} MinCost(\Delta V_i)$$

Let S denote a set of the intermediate results that are decided to be materialized and reused. For example, $S = \{\}$ in Figure 3-(a) and $S = \{\Delta(B \bowtie C)\}$ in Figure 3-(b). Let $MinCost(\Delta V_i|S)$ denote the cost of the optimal delta evaluation tree for V_i given that the expressions in S are already computed and materialized. It is clear that $MinCost(\Delta V_i|S) \leq MinCost(\Delta V_i)$. Given a set S, the total cost $TotalCost(V|S)$ of computing $\Delta V_1, \Delta V_2, ..., \Delta V_n$ using S is as follows:

$$\begin{aligned} TotalCost(V|S) &= \sum_{S_i \in S} MinCost(S_i|S - \{S_i\}) \\ &\quad + \sum_{V_i \in V} MinCost(\Delta V_i|S) \end{aligned}$$

The first term corresponds to the cost of computing S. Note that $MinCost(S_i|S - \{S_i\})$ is used because S_i should not be in S when we compute S_i. The second term corresponds to the cost of computing $\Delta V_1, \Delta V_2, ..., \Delta V_n$ given that the expressions in S are computed and materialized. To obtain $MinCost(\Delta V_i)$, we can use expression (4) in Section 4.2. However, for $MinCost(\Delta V_i|S)$, we need to extend expres-

```
Procedure GreedyMultipleViews(input: V = {V_1, V_2, ..., V_n})
  /* Initialize */
  C = {Δ(R_s ⋈ R_t ⋈ ... ⋈ R_u) | R_s ⋈ R_t ⋈ ... ⋈ R_u appears in more than two views in V}
  S = ∅

  /* Main loop */
  While (C ≠ ∅) do
    Choose c ∈ C which minimizes TotalCost(V|S ∪ {c})
    If (TotalCost(V|S ∪ {c}) < TotalCost(V|S)) do
      S = S ∪ {c}; C = C − {c}
    Else
      C = ∅
    End If
  End While

  /* Results */
  Construct the optimal delta evaluation trees for V_1, V_2, ..., V_n using S.
End Procedure
```

Figure 4: An greedy algorithm for maintaining multiple views

sion (4) to reflect S. Expression (4) is extended as follows:

$$
\begin{aligned}
&Cost(\Delta R_i|S) = 0\\
&Cost(\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)|S) = 0,\\
&\qquad\qquad if\ \Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n) \in S\\
&Cost(\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)|S) = \qquad (5)\\
&\qquad \sum_{i=1}^{m} Cost(\Delta(\bowtie P_i)|S)\\
&\qquad + \ (|\Delta(\bowtie P_1)| + |P_2| + \cdots + |P_m|)\\
&\qquad + \ (|P_1^{'}| + |\Delta(\bowtie P_2)| + \cdots + |P_m|)\\
&\qquad + \ \cdots\\
&\qquad + \ (|P_1^{'}| + |P_2^{'}| + \cdots + |\Delta(\bowtie P_m)|),\\
&\qquad\qquad otherwise,
\end{aligned}
$$

where $Cost(\Delta V_i|S)$ denotes the cost of processing expression ΔV given that expressions in S are materialized. Note that if we have $\Delta(R_1 \bowtie R_2 \bowtie ... \bowtie R_n)$ in S, $Cost(\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)|S) = 0$. Using expression (5), we can obtain $MinCost(\Delta V_i|S)$ and the corresponding optimal delta evaluation tree for V_i that uses S. Now, we can define the multiple view maintenance problem as follows: *Given $V = \{V_1, V_2, ..., V_n\}$, find a set of intermediate results S to be materialized and reused, and the optimal delta evaluation trees for $V_1, V_2, ..., V_n$ that use S such that $TotalCost(V|S)$ is minimized.*

5.2 The Greedy Algorithm

The multiple view maintenance problem we have defined has the same form as the traditional multiple query optimization problem[17][18]. The aim of multiple query optimization is to exploit common subexpressions to reduce overall evaluation cost. A delta evaluation tree for a view in our problem corresponds to an access plan for a query in the multiple query optimization problem. Similarly, common nodes among delta evaluation trees correspond to common subexpressions among input queries. Note that instead of original incremental maintenance expressions, which can be quite complex, we consider delta evaluation trees in our problem. However, [17] showed that the multiple query optimization problem is NP-hard. For this reason, most work on the multiple query optimization has concentrated on heuristic algorithms. In this paper, we develop a heuristic algorithm that selects a set S to be materialized in a greedy manner. Our approach is similar to that of [16]. After S is selected, the algorithm constructs the optimal delta evaluation tree for each view using S.

In our heuristic algorithm, we first construct a set C that contains all possible subexpressions that can be reused among views. It is clear that S must be a subset of C, i.e., $S \subseteq C$. To find the optimal set S that minimizes $TotalCost(V|S)$, one of the simplest methods is to iterate over each subset of C and select a subset of C that minimizes $TotalCost(V|S)$. However, the number of subsets of C is exponential in the size of C. Therefore the exhaustive algorithm that iterates over each subset of C is impractical. Figure 4 shows a heuristic algorithm that selects S in a greedy manner. The algorithm iteratively picks shared expressions to materialize. At each iteration, the expression that is most beneficial to the total cost if it is materialized is chosen to be added to S. We present the performance result of our algorithm for multiple views in Section 6.

6. EXPERIMENT

In this section, we show the result of performance experiment among several view maintenance methods. The performance of each view maintenance method was measured by the time taken for updating views by that method. In

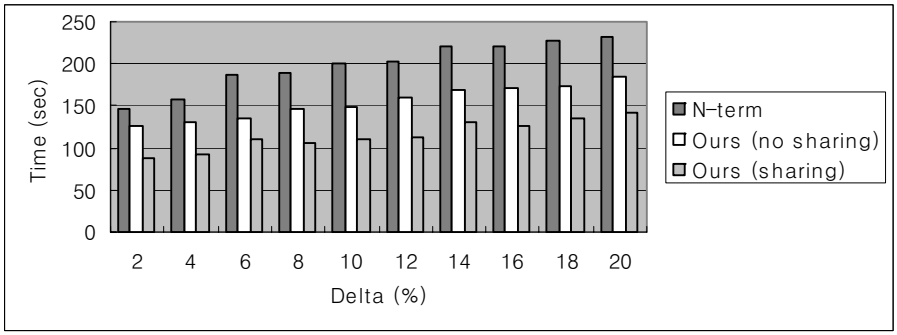

Figure 5: The performance evaluation of updating multiple views

the experiment, we used the TPC-R benchmark schema and data[19]. An Oracle8*i* database system running on a Sun Ultra-60 with 256MB RAM was used as the data warehouse environment. We defined six views, i.e., V_1, V_2, V_3, V_4, V_5, and V_6, each of which was defined as follows:

$$
\begin{aligned}
V_1 &= PART \bowtie SUPPLIER \bowtie PARTSUPP \bowtie \\
&\quad NATION \bowtie REGION \\
V_2 &= CUSTOMER \bowtie ORDERS \bowtie LINEITEM \\
V_3 &= CUSTOMER \bowtie ORDERS \bowtie LINEITEM \bowtie \\
&\quad SUPPLIER \bowtie NATION \bowtie REGION \\
V_4 &= PART \bowtie SUPPLIER \bowtie LINEITEM \bowtie \\
&\quad PARTSUPP \bowtie ORDERS \bowtie NATION \\
V_5 &= NATION \bowtie CUSTOMER \bowtie ORDERS \bowtie \\
&\quad LINEITEM \\
V_6 &= NATION \bowtie LINEITEM \bowtie ORDERS \bowtie \\
&\quad SUPPLIER
\end{aligned}
$$

The definitions of V_1, V_2, V_3, V_4, V_5, and V_6 are based on the TPC-R query Q_2, Q_3, Q_5, Q_9, Q_{10}, and Q_{21} respectively. All the base relations were created and populated using the TPC-R schema.

In the experiment, we compared our incremental maintenance method for multiple views (i.e., *Ours (sharing)*) with our previous method that does not allow sharing of intermediate results. (i.e., *Ours (no sharing)*) Also, a view maintenance method that uses expression (2) was compared in the experiment. (i.e., *N-term*)

The time taken for executing each view maintenance method is shown in Figure 5. In the experiment, we made changes to the base relations by inserting new tuples to the base relations. We varied the size of the changes to the base relations from 2% to 20% of their original size, which is a typically used range of changes in the experiments of incremental view maintenance methods[11][5]. Figure 5 shows that our incremental maintenance method for multiple views outperforms the other methods in the whole range.

The time reported in Figure 5 is the optimization time plus the total time for updating all the six views. For our method, optimization time means the time taken for finding delta evaluation trees for the six views. However, these optimizations took just a few seconds in the experiment, which is quite a short time in our environment. Hence, we can confirm that our proposed method can be effectively used to maintain multiple join views.

7. CONCLUSIONS

Materialized views have been widely used in data warehouses to support OLAP queries. When data sources change, materialized views need to be updated to reflect the changes of data sources. Since the updates of views may impose a significant overhead on the warehouse, it is very important to update the warehouse views efficiently. There are commonly multiple views in the warehouse. We presented an efficient incremental maintenance method for multiple join views that reuses common intermediate results among views. The concept of the delta evaluation tree with consideration of sharing intermediate results makes it possible to maintain multiple join views efficiently. We showed through experiment that the proposed method gives more benefits than the method that maintains each view separately.

8. REFERENCES

[1] J. A. Blakeley, P. A. Larson, F. W. Tompa, Efficiently Updating Materialized Views, In Proceedings of ACM SIGMOD Conference, pp. 61-71, 1986.

[2] A. Gupta, I. S. Mumick, V. S. Subrahmanian, Maintaining views incrementally, In Proceedings of ACM SIGMOD Conference, pp. 157-166, 1993.

[3] A. Gupta, I. S. Mumick, Maintenance of materialized views: problems, techniques, and applications, IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2):3-18, 1995.

[4] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, H. Trickey, Algorithms for deferred view maintenance, In Proceedings of ACM SIGMOD Conference, pp. 469-492, 1996.

[5] W. J. Labio, R. Yerneni, H. Garcia-Molina, Shrinking the Warehouse Update Window, In Proceedings of ACM SIGMOD Conference, pp. 383-394, 1999.

[6] K. Y. Lee, J. H. Son, M. H. Kim, Efficient Incremental View Maintenance in Data Warehouses, In Proceedings of ACM CIKM Conference, pp. 349-356, 2001.

[7] H. Mistry, R. Roy, S. Sudarshan, K. Ramamritham, Materialized View Selection and Maintenance Using Multi-Query Optimization, In Proceedings of ACM SIGMOD Conference, 2001.

[8] Hao He, Junyi Xie, Jun Yang, Hai Yu, Asymmetric Batch Incremental View Maintenance, In Proceedings of ICDE, 2005.

[9] T. Griffin, L. Libkin, Incremental maintenance of views with duplicates, In Proceedings of ACM SIGMOD Conference, pp. 328-339, 1995.

[10] D. Quass, Maintenance expressions for views with aggregation, In Workshop on Materialized Views: Techniques and Applications, pp. 110-118, 1996.

[11] I. S. Mumick, D. Quass, B. S. Mumick, Maintenance of data cubes and summary tables in a warehouse, In Proceedings of ACM SIGMOD Conference, pp. 100-111, 1997.

[12] H. Gupta, I.S. Mumick, Incremental maintenance of aggregate and outerjoin expressions, Technical Report, Stanford University, 1999.

[13] T. Palpanas, R. Sidle, R. Cochrane, H. Pirahesh, Incremental maintenance for non-Distributive aggregate functions, In Proceedings of VLDB, 2002.

[14] K. Yi, H. Yu, J. Yang, G. Xia, Y. Chen, Efficient Maintenance of Materialized Top-k Views, In Proceedings of ICDE, 2003.

[15] S. Chen, E. A. Rundensteiner, GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views, In Proceedings of ICDE, 2005.

[16] P. Roy, S. Seshadri, S. Sudarshan, S. Bhobe, Efficient and Extensible Algorithms for Multi Query Optimization, In Proceedings of ACM SIGMOD Conference, pp. 249-260, 2000.

[17] T. K. Sellis, Multiple Query Optimization, ACM Transactions on Database Systems, 13(1):23-52, 1988.

[18] K. Shim, T. Sellis, D. Nau, Improvements on a Heuristic Algorithm for Multi-query Optimization, Data and Knowledge Engineering, 12:197-222, 1994.

[19] TPC Committee, Transaction Processing Council, http://www.tpc.org/