Reducing the cost of accessing relations in incremental view maintenance

Ki Yong Lee a,⁎, Jin Hyun Son b, Myoung Ho Kim a

a Department of Electrical Engineering and Computer Science, KAIST, 373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, South Korea
b Department of Computer Science and Engineering, Hanyang University, 1271 Sa-I dong, Ansan, Kyunggi-Do, 425-791, South Korea

Received 23 August 2005; received in revised form 6 November 2006; accepted 12 November 2006
Available online 11 January 2007

Abstract

In the data warehouse environment, the concept of a materialized view is common and important for efficient support of OLAP query processing. Materialized views are generally derived from several relations. These materialized views need to be updated when source relations change. Since the propagation of updates to the views may impose a significant overhead, it is essential to update the warehouse views efficiently. Though various view maintenance strategies have been discussed in the past, optimizations on the total accesses to relations have not been sufficiently investigated.

In this paper we propose an efficient incremental view maintenance method called optimal delta evaluation that can minimize the total accesses to relations. We first present the delta evaluation expression and a delta evaluation tree which are core concepts of the method. Then, a dynamic programming algorithm that can find the optimal delta evaluation tree is proposed. We also present various experimental results that show the usefulness and efficiency of our proposed method.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Materialized view; View maintenance; Incremental maintenance

1. Introduction

The concept of a data warehouses has been used to provide analysts and managers with strategic information about the key figures of the underlying business. Data warehouses periodically extract and store the data needed for analytical purposes from remote information sources. Most queries on data warehouses are related to statistics involving aggregates rather than specific data content. They need to be processed in highly efficient manner to facilitate on-line analytical processing (OLAP). These kinds of queries generally include joins of many relations, which incur considerable query processing costs. In this regard, materialized views in a data warehouse should be effective in speeding up the OLAP queries, and are increasingly supported by many commercial systems.

The main objective of a materialized view is to improve query performance. When base relations in a data warehouse are updated due to the changes of remote information sources, the materialized views defined by the base relations must also be updated. Since the amount of data reflected to data warehouses and queries calling for up-to-date information have been increasing, an efficient view maintenance strategy
has been an important issue in data warehouse applications.

We can reflect changes in base relations to materialized views by either recomputation or incremental maintenance. Here, incrementally maintaining a materialized view denotes the propagation of only its changes. Since the amounts of changes are much smaller than the sizes of base relations and views in general, computing only the changes of a view is usually much cheaper than recomputing from scratch. Thus, many possible methods that allow incremental view maintenance have been proposed in the past [5,3,8,19,15,7].

### 1.1. Related work

Various kinds of materialized views can be formed over base relations, e.g., select views, project views, select-project-join (SPJ) views, aggregation views, and so on. [5,3,6] proposed formal expressions through which we can incrementally maintain SPJ views. The incremental view maintenance expressions that can additionally support aggregation views were proposed in Refs. [8,19,15,7]. However, they did not mention how to select an efficient maintenance expression from many possible expressions that can be applied to a view. Recently, the incremental maintenance of views with more complex operators, such as non-distributive aggregate functions [17], top-κ queries [25], pivots and unpivots [1], was proposed.

In this paper, we will focus on SPJ materialized views that have been one of the most widely used materialized views in data warehouse applications. There have been a few methods for incremental maintenance of an SPJ view. For an n-way join view \( V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \), [5] proposed the following maintenance expression to compute the change of \( V \), which is denoted by \( \Delta V \).

\[
\Delta V = (\Delta R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) \cup (R_1 \bowtie \Delta R_2 \bowtie \ldots \bowtie R_n) \\
\cup \cdots (\Delta R_1 \bowtie \Delta R_2 \bowtie \ldots \bowtie \Delta R_n).
\]  

(1)

Here, \( \Delta R_i \) denotes the change of base relation \( R_i \). To update view \( V \), it is sufficient to compute \( \Delta V \) and propagate it to \( V \). The relation that denotes the change of some relation such as \( \Delta R_i \) is called a delta relation. Let each join expression, i.e., a subexpression consisting of only joins, be called a join term, or simply a term. Then, expression (1) consists of \( (2^\alpha - 1) \) terms (i.e., \( \Delta R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n, R_1 \bowtie \Delta R_2 \bowtie \ldots \bowtie R_n, \ldots, \Delta R_1 \bowtie \Delta R_2 \bowtie \ldots \bowtie \Delta R_n \)). For example, \( \Delta V \) of a 3-way join view \( V = R_1 \bowtie R_2 \bowtie R_3 \) can be obtained by the following expression consisting of seven terms:

\[
\Delta V = (\Delta R_1 \bowtie R_2 \bowtie R_3) \cup (R_1 \bowtie \Delta R_2 \bowtie R_3) \\
\cup (\Delta R_1 \bowtie R_2 \bowtie \Delta R_3) \cup (\Delta R_1 \bowtie \Delta R_2 \bowtie R_3) \\
\cup (\Delta R_1 \bowtie \Delta R_2 \bowtie \Delta R_3).
\]

Since expression (1) requires too many terms in computing the change of a view, a new maintenance expression consisting of only \( n \)-terms has been proposed in Ref. [8] as follows.

\[
\Delta V = (\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie \cdots \bowtie R_n) \\
\cup (R_1 \bowtie \Delta R_2 \bowtie R_3 \bowtie \cdots \bowtie R_n) \cup \cdots \\
\cup (R_1 \bowtie R_2 \bowtie \cdots \bowtie \Delta R_{n-1} \bowtie \Delta R_n).
\]  

(2)

Here, \( R'_1 \) represents \( R_1 \cup \Delta R_i \). Expression (2) is equivalent to expression (1), but has fewer terms. Roughly speaking, each term in expression (2) represents the change of \( V \) due to the change of a certain base relation. For example, the first term \( \Delta R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \) represents the change of \( V \) due to \( \Delta R_1 \), and the second term \( R'_1 \bowtie \Delta R_2 \bowtie \ldots \bowtie R_n \) represents the change of \( V \) due to \( \Delta R_2 \), after \( R_1 \) has been updated. In general, the term \( R'_1 \bowtie \ldots \bowtie \Delta R_{i-1} \bowtie R_i \bowtie \Delta R_{i+1} \bowtie \ldots \bowtie R_n \) represents the change of \( V \) due to \( \Delta R_i \) after \( R_1, R_2, \ldots, R_{i-1} \) have been updated.

Besides expression (1) and expression (2), there can be many other expressions that can be applied to compute the change of an SPJ view. In Ref. [12], however, it has shown that expression (2) can build the most efficient maintenance strategy so far.

How to maintain views correctly using the previous maintenance expressions when multiple data sources are distributed and their changes occur concurrently was discussed in Refs. [27,26,2]. Incremental maintenance of join views in a parallel RDBMS was described in Ref. [13]. Ref. [14] proposed a maintenance algorithm that exploits common subexpressions among view maintenance expressions. Ref. [10] proposed a new incremental view maintenance strategy that selectively propagates the change of some base relations while keeping batching others. However, all these works are based on expression (1) or expression (2) to compute the change of SPJ views.

### 1.2. Motivation

For the performance of an incremental maintenance expression, the total amount of accesses to relations is very important. As we showed before, an incremental maintenance expression is composed of base relations and their delta relations in certain ways. Because the size
of a base relation is much larger than that of its changes in general, the cost of evaluating incremental maintenance expression is mainly affected by the number of accesses to base relations.

Consider a view $V$ defined over $n$ base relations, i.e., $V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$. Let us assume that each of the terms in a maintenance expression is evaluated independently. In expression (1), there are $(2^n - 1)$ terms and each base relation is included in exactly $(2^n - 1)$ terms. Thus, each base relation has to be accessed at least $(2^n - 1)$ times to evaluate expression (2). On the other hand, expression (2) consists of $n$ terms and each base relation is included in $(n - 1)$ terms. Hence, each base relation is accessed only $(n - 1)$ times to evaluate expression (2). This expression has been known to be the most efficient among all expressions developed until now [13]. However, we can further reduce the number of accesses to base relations. The following example motivates our approach.

For a view $V = R_1 \bowtie R_2 \bowtie R_3$, one of the expressions that compute $\Delta V$ by expression (2) is as follows.

$$\Delta V = (\Delta R_1 \bowtie R_2 \bowtie R_3) \cup (R'_1 \bowtie \Delta R_2 \bowtie R_3) \cup (R'_1 \bowtie R'_2 \bowtie \Delta R_3).$$

(3)

Consider the following expression that is equivalent to the above expression.

$$\Delta V = ((\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2)) \bowtie R_3 \cup (R'_1 \bowtie R'_2 \bowtie \Delta R_3).$$

(4)

Here, $(\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2)$ is evaluated first, and then its result is joined with $R_3$. Hence, base relation $R_3$ is accessed only once in expression (4) while $R_3$ is accessed two times in expression (3). Thus, if the size of $R_3$ is much larger than those of $R_1$ and $R_2$, we can reduce the cost of computing $\Delta V$ considerably. Though the transformation in this case is quite simple, the optimal expression in the general case is not very straightforward. In this paper we propose an efficient incremental view maintenance method that guarantees high performance view maintenance.

We will assume throughout this paper that the join operation $W$ has precedence over the union operation $\cup$. Thus, we will sometimes omit the parenthesis in the expression if there is no ambiguity. That is, expression (4) can be described as $\Delta V' = (\Delta R_1 \bowtie R_1 \cup R'_1 \bowtie \Delta R_2) \bowtie R_3 \cup R'_1 \bowtie R'_2 \bowtie \Delta R_3$.

The remainder of the paper is organized as follows. In Section 2, we present our materialized view model, and then its cost model for performance comparison of various view maintenance strategies. Our view maintenance method called optimal delta evaluation is proposed in Section 3. Analysis of the proposed method is described in Section 4. We present an extension of the method to multiple views in Section 5. Section 6 gives results of performance experiments. Finally, we conclude our work in Section 7.

2. Preliminaries

2.1. View definition model

The derived data from remote information sources is stored in a data warehouse as fact tables, dimension tables or summary tables [11]. These tables can be seen as materialized views defined over certain base relations. When base relations are changed, the changes need to be propagated to these views properly to keep them up-to-date. In most data warehouse applications, source changes are gathered and propagated to the views in large batches for efficiency. The data warehouse is unavailable to users during the batch window. Hence, the study of the efficient view maintenance mechanisms is being pursued vigorously.

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 select-project-join (SPJ) expression. SPJ views as a general form of a view definition can cover most of materialized views in a data warehouse environment. Moreover, an SPJ view can be easily extended to accommodate a view with aggregations using generalized projections [19].

An SPJ view $V$ over $n$ base relations $R_1, R_2, \ldots, R_n$, is defined as follows:

$$V = \Pi_{C}(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n)$$

where $L$ is a list of projection attributes and $C$ is a selection condition.

For a base relation $R$, the changes of $R$ are denoted by $\Delta R$. We keep additional count information in each tuple of $R$ and $\Delta R$ as in Ref. [8]. The count value for a tuple of $R$ represents the number of distinct derivations of the tuple. The positive count value for a tuple of $\Delta R$ represents the number of newly inserted copies of the tuple into $R$, while the negative count value means the number of deleted copies of the tuple from $R$. An update is modeled as a delete followed by an insert. Some more details can be found in Ref. [8]. $R'$ represents the relation $R$ to which $\Delta R$ is propagated, i.e., $R' = R \cup \Delta R$. The formal definition of the union operator with the count value can be found in Refs. [5,8]. Based on this framework, our method can be applied to inserts, deletes, and updates of base tables. If the change of a
view $V$ is computed by $\Delta V = (\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2)$, then an incremental maintenance of $V$ can be expressed as follows:

$$V' = V \cup (\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2).$$

Note that our method can also be applied to aggregate SPJ views. For an SPJ view $V$, an aggregate SPJ view can be expressed as $\Pi_L(V)$, where $\Pi$ is the generalized projection and $L$ is the projected attributes that include aggregate functions. If aggregate functions are distributive (e.g., COUNT, SUM, AVG), the change of the aggregate SPJ view $\Pi_L(V)$ can be written as $\Delta(\Pi_L(V)) = \Pi_L(\Delta V)$ where $\Delta V$ is the change of $V$ [19]. Thus, our method can be applied to compute $\Delta V$ in $\Pi_L(\Delta V)$.

2.2. Cost model

In this paper we adopt the linear work metric developed in Ref. [12] 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 [12]. In the linear work metric, the cost or processing a maintenance expression is the sum of the costs or processing each term of the expression when each term is assumed to be evaluated independently, 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 $E$. If the change of view $V = R_1 \bowtie R_2 \bowtie R_3$ is computed by expression (3), then the cost of computing $\Delta V$ can be obtained as follows:

$$\text{Cost}(\Delta V) = \text{Cost}(\Delta R_1 \bowtie R_2 \bowtie R_3) + \text{Cost}(R'_1 \bowtie \Delta R_2 \bowtie R_3) + \text{Cost}(\Delta R'_1 \bowtie R'_2 \bowtie \Delta R_2)$$

$$= c \cdot (|\Delta R_1| + |R_2| + |R_3|) + c \cdot (|R'_1| + |\Delta R_2| + |R_3|) + c \cdot (|\Delta R'_1| + |R'_2| + |\Delta R_2|).$$

Here, $c$ is a constant and $|R|$ is the size of relation $R$. If $\Delta V$ is computed by expression (4), Cost($\Delta V$) is as follows:

$$\text{Cost}(\Delta V) = \text{Cost}((\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2)) \bowtie R_3) + \text{Cost}(R'_1 \bowtie R'_2 \bowtie \Delta R_3)$$

$$= \text{Cost}((\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2)) + \text{Cost}(R'_1 \bowtie R'_2 \bowtie \Delta R_3)$$

$$= c \cdot (|\Delta R_1| + |R_2| + |R'_1| + |\Delta R_2|) + c \cdot (|\Delta R'_1| + |R'_2| + |\Delta R_3|).$$

Recall that $(\Delta R_1 \bowtie R_2) \cup (R'_1 \bowtie \Delta R_2)$ is evaluated first, and then its result is joined with $R_3$ in expression (4).

3. Optimal delta evaluation method

We propose a new view maintenance method called the optimal delta evaluation method which can minimize the cost of computing the change of a view. $\Delta V$ can be computed in many ways, each of which depends on the expression for $\Delta V$. Thus, we need to find the optimal maintenance expression to minimize the cost of computing $\Delta V$. For example, from expression (5) and expression (6), we can notice that the costs of computing $\Delta V$ become different depending on the expressions for $\Delta V$.

As we mentioned before, expression (2) has been known to be the most efficient maintenance expression described so far [12]. However, we can find out other equivalent maintenance expressions that further reduce accesses to base relations. The basic idea is based on recursive partitioning which is described below.

3.1. Recursive partitioning

Consider a view $V = R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$. According to expression (2), $\Delta V$ can be computed as follows.

$$\Delta V = (\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4) \cup (R'_1 \bowtie \Delta R_2 \bowtie R_3 \bowtie R_4) \cup (R'_1 \bowtie R'_2 \bowtie \Delta R_3 \bowtie R_4) \cup (R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie \Delta R_4).$$

(7)

By factoring out $R_3 \bowtie R_4$ from the first and second terms and $R'_1 \bowtie R'_2$ from the third and fourth terms, expression (7) can be rewritten such that

$$\Delta V = ((\Delta R_1 \bowtie R_2 \bowtie R'_1 \bowtie \Delta R_2) \bowtie R_3 \bowtie R_4) \cup (R'_1 \bowtie R'_2 \bowtie (\Delta R_3 \bowtie R_4 \cup R'_3 \bowtie \Delta R_4)).$$

(8)
Note that the number of accesses to each of $R_1$, $R_2$, $R_3$, and $R_4$ is reduced to 2 in expression (8) while it is 3 in expression (7).

Now, let us introduce the notation $\Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k)$ to denote the change of $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k$. That is, $\Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k)$ is the difference between $R'_1 \bowtie R'_2 \bowtie \ldots \bowtie R'_k$ and $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k$. Then, expression (8) can be rewritten as

$$\Delta V = (\Delta(R_1 \bowtie R_2) \bowtie R_3 \bowtie R_4) \cup (R'_1 \bowtie R'_2 \bowtie \Delta(R_3 \bowtie R_4)),$$

where $\Delta(R_1 \bowtie R_2) = (\Delta R_1 \bowtie R_2 \bowtie R_3) \cup (R'_1 \bowtie \Delta R_2 \bowtie R_3) \cup (R'_1 \bowtie R'_2 \bowtie \Delta R_3)$. Here, $\Delta(R_1 \bowtie R_2)$ and $\Delta(R_3 \bowtie R_4)$ are computed first before other remaining joins to compute $\Delta V$. Expression (9) implies the following evaluation steps:

(i) The base relations are grouped into \{R_1, R_2\} and \{R_3, R_4\}.

(ii) For each group, the change of the join of base relations in that group is computed.

(iii) Then, the overall change of $V$, i.e., $\Delta V$ is computed by using the results of (ii).

Expression (7) can be rewritten in a number of different ways. For example,

$$\Delta V = (\Delta(R_1 \bowtie R_2 \bowtie R_3) \bowtie R_4) \cup (R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie \Delta(R_4)),$$

where $\Delta(R_1 \bowtie R_2 \bowtie R_3) = (\Delta R_1 \bowtie R_2 \bowtie R_3) \cup (R'_1 \bowtie \Delta R_2 \bowtie R_3) \cup (R'_1 \bowtie R'_2 \bowtie \Delta R_3)$. Here, $\Delta(R_1 \bowtie R_2 \bowtie R_3)$ is computed first before other remaining joins. In this case, the base relations are grouped into \{R_1, R_2, R_3\} and \{R_4\}, which results in only one access to $R_4$. Note that expression (9) and expression (10) show two different partitions of \{R_1, R_2, R_3, R_4\}, i.e., \{\{R_1, R_2\}, \{R_3, R_4\}\} and \{\{R_1, R_2, R_3\}, \{R_4\}\}. Moreover, the evaluation costs of expression (9) and expression (10) should be different.

Now we generalize this concept. Suppose $V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$. Let $\{P_1, P_2, \ldots P_m\}$ be a partition of \{R_1, R_2, \ldots R_n\} such that $\bigcup_{i=1}^m P_i = \{R_1, R_2, \ldots R_n\}$, $P_i \cap P_j = \emptyset \ (1 \leq i \neq j \leq m)$, and $P_i \neq \emptyset \ (1 \leq i \leq m)$. Then $\Delta V$ can be represented as follows:

$$\Delta V = (\Delta(\bowtie P_1) \bowtie (\bowtie P_2) \bowtie \ldots \bowtie (\bowtie P_m)) \cup ((\bowtie P'_1) \bowtie \Delta(\bowtie P_2) \bowtie \ldots \bowtie (\bowtie P_m)) \cup \ldots \cup ((\bowtie P'_1) \bowtie (\bowtie P'_2) \bowtie \ldots \bowtie \Delta(\bowtie P_m)),$$

where $\bowtie P_i$ and $\bowtie P'_i$ denote $R_i \bowtie R_i \bowtie \ldots \bowtie R_u$ and $R'_i \bowtie R'_i \bowtie \ldots \bowtie R'_u$ respectively for $P_i = \{R_u, R_t, \ldots R_a\}$. Expression (11) partitions the set of base relations \{R_1, R_2, \ldots, R_n\} into $m$ groups $P_1, P_2, \ldots, P_m$, and the join of relations in each group is treated as if it were a base relation. It computes $\Delta(\bowtie P_1)$, $\Delta(\bowtie P_2), \ldots \Delta(\bowtie P_m)$ first, and then finally computes the overall change of $V$ by using the results of $\Delta(\bowtie P_1)$, $\Delta(\bowtie P_2), \ldots \Delta(\bowtie P_m)$.

Now, computation of $\Delta(\bowtie P_i)(1 \leq i \leq m)$ can proceed by applying the same strategy recursively. In other words, computing $\Delta(\bowtie P_i)$ can be performed by partitioning $P_i$ into multiple groups. Partitioning is recursively performed until each group in a partition has only one base relation. We call expression (11) a delta evaluation expression. Here, it is assumed that the expression is recursively expanded until $\Delta(\bowtie P_j)(1 \leq j \leq m)$ is resolved into $\Delta R_j$ for some $j (1 \leq j \leq n)$. It is easy to see that the delta evaluation expression computes the changes of a view correctly.

Depending on the choices of $P_1, P_2, \ldots, P_m$, there can be many delta evaluation expressions for a view. For example, $P_1 = \{R_1, R_2\}$, $P_2 = \{R_3\}$ for a view $V = R_1 \bowtie R_2 \bowtie R_3$ produce a delta evaluation expression $\Delta V = (\Delta(R_1 \bowtie R_2) \bowtie R_3) \cup (R'_1 \bowtie R'_2 \bowtie \Delta(R_3))$, while $P_1 = \{R_1\}$, $P_2 = \{R_2\}$, $P_3 = \{R_3\}$ produce $\Delta V = (\Delta R_1) \bowtie R_2 \bowtie R_3) \cup (R'_1 \bowtie \Delta R_2) \bowtie R_3 \cup (R'_1 \bowtie R'_2 \bowtie \Delta R_3)$. Although all these expressions are equivalent, i.e., compute the change of the view correctly, their evaluation costs differ. Thus, we need to find the optimal delta evaluation expression among these expressions to minimize the maintenance cost. An incremental view maintenance method that uses the optimal delta evaluation expression to compute the change of a view is called the optimal delta evaluation method. Shortly, we will present an algorithm that finds the optimal delta evaluation expression for a view.
3.2. Delta evaluation trees

A delta evaluation expression can be represented as a tree, which we call a delta evaluation tree. Consider a delta evaluation expression for a view \( V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \). Then the root node of the delta evaluation tree representing that expression is \( \Delta V \), i.e., \( \Delta (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) \). The changes of each base relation, i.e., \( \Delta R_1, \Delta R_2, \ldots, \Delta R_n \), become the leaf nodes of the tree. Each nonleaf node corresponds to \( \Delta (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_a) \) for some \( \{R_a, R_b, \ldots, R_n\} \subseteq \{R_1, R_2, \ldots, R_n\} \). If \( \{R_a, R_b, \ldots, R_n\} \) is partitioned into \( P_1, P_2, \ldots, P_m \) in the corresponding delta evaluation expression for this tree, then \( \Delta (\bowtie P_1) \), \( \Delta (\bowtie P_2), \ldots, \Delta (\bowtie P_m) \) become the children of the node \( \Delta (R_1 \bowtie \ldots \bowtie R_n) \) with \( \Delta (\bowtie P_i) (1 \leq i \leq m) \) being the \( i \)-th child from the left. For example, Fig. 1 shows two possible delta evaluation trees for \( V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6 \). (The numbers in the upper left corner of each node in Fig. 1 will be described in Section 4.) Each tree in Fig. 1 represents the following delta evaluation expression:

\[
\begin{align*}
(a) \Delta (R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6) &= \left(\Delta (R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6) \cup (R_1' \bowtie R_2' \bowtie R_3' \bowtie R_4' \bowtie R_5' \bowtie \Delta (R_6))\right)
\end{align*}
\]

\[
\begin{align*}
\Delta (R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5) &= \left(\Delta (R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5) \cup (R_1' \bowtie R_2' \bowtie R_3' \bowtie \Delta (R_4 \bowtie R_5))\right)
\end{align*}
\]

\[
\begin{align*}
\Delta (R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4) &= \left(\Delta (R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4) \cup (R_1' \bowtie \Delta (R_3 \bowtie R_4))\right)
\end{align*}
\]

\[
\begin{align*}
\Delta (R_1 \bowtie R_2) &= \Delta (R_1) \cup (R_1' \bowtie \Delta R_2)
\end{align*}
\]

As you can notice, the expression represented by Fig. 1-(b) is the same as expression (2). In fact, when \( P_1 = \{R_1\}, P_2 = \{R_2\}, \ldots, P_n = \{R_n\} \), expression (11) is resolved into expression (2). That is, the delta evaluation expressions cover all the expressions that have the form of expression (2). As we mentioned before, there can be many possible delta evaluation trees for a given view. We will discuss how to find the optimal delta evaluation tree among them in the next subsection.

3.3. Optimal delta evaluation tree

In this paper, the optimal delta evaluation tree means one whose corresponding delta evaluation expression requires the minimal evaluation cost with respect to the cost model described in Section 2.2. We now present a dynamic programming algorithm that can find out the optimal delta evaluation tree for a given view.

For a given SPJ view, there can be many possible delta evaluation trees. As we can see in expression (11), different partitions \( \{P_1, P_2, \ldots, P_m\} \) of \( \{R_1, R_2, \ldots, R_n\} \) result in different delta evaluation trees. The number of different partitions
\{P_1, P_2, \ldots, P_m\} of \{R_1, R_2, \ldots, R_n\} is \(B(n)\), which is the \(n\)th Bell number [16]. For a large value \(n\), \(B(n)\) is approximately \(n^n\) [9]. Thus, there can be a large number of delta evaluation trees for a given view. The optimal delta evaluation tree is the one with the minimal evaluation cost among them.

We can obtain the evaluation cost of a delta evaluation tree by applying the linear work metric described in Section 2.2 to the delta evaluation expression represented by the tree. From expression (11) and the definition of the linear work metric, the cost of a delta evaluation expression can be computed recursively by the following equation. Here, we assume that the constant \(c\) is equal to 1 in the linear work metric, for simplicity.

\[
\text{Cost}(\Delta R_i) = 0
\]

\[
\text{Cost}(\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)) = \sum_{i=1}^{m} \text{Cost}(\Delta(R_i)) + (|P_i'| + |\Delta(P_i)|) + \cdots + |P_m|
\]

\[
+ \cdots + (|P_1' \bowtie |P_2'| \bowtie \cdots \bowtie |P_m'|),
\]

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 \(P_i = \{R_s, R_t, \ldots, R_u\}\). Note that we assume \(\text{Cost}(\Delta R_i) = 0\) because \(\Delta R_i (1 \leq i \leq n)\) are given, which means no need for computing it. The optimal delta evaluation expression is the one that minimizes expression (12).

To find an optimal solution that minimizes expression (12), we use the dynamic programming, which is commonly applied to optimization problems (Fig. 2). We use a bottom-up approach. That is, we first construct optimal delta evaluation tree for each base relation \(R_i\). Then we construct optimal delta evaluation trees for \(R_i \bowtie R_j \bowtie R_k \bowtie R_l \bowtie R_n\) using the results of the first step. In the next step we construct optimal delta evaluation trees for \(R_i \bowtie R_j \bowtie R_k \bowtie R_l \bowtie R_m\) using the

---

**Procedure FindOptimalDeltaEvaluationTree**

```plaintext
/* Initialize */

For (i = 1 to n) do

- Construct the optimal delta evaluation tree for \(R_i\), i.e., \(\Delta R_i\).

/* Main body */

For (i = 2 to n) do

For (each \(P \subseteq \{R_1, R_2, \ldots, R_n\}\) (such that the cardinality of \(P\) is \(i\)) do

- Find the partition \(\{P_1, P_2, \ldots, P_m\}\) of \(P\) that minimizes expression (12).

- Construct the optimal delta evaluation tree for \(\bowtie P\) using optimal delta evaluation trees for \(\bowtie P_1, \bowtie P_2, \ldots, \bowtie P_m\).

(Note that the optimal delta evaluation tree for \(\bowtie P_i\) \((1 \leq i \leq m)\)

is already obtained.)

- Compute \(|P_i|, |P_i'|, |\Delta(\bowtie P)|\) using \(|P_i|, |P_i'|, |\Delta(\bowtie P_i)|\) \((1 \leq i \leq m)\).

End For

End For

/* As a result, we can obtain an optimal delta evaluation tree for the input view \(V\) */

End Procedure
```

Fig. 2. Dynamic programming algorithm.
results of the previous steps. In this way, we can eventually construct the optimal delta evaluation tree for \( R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \).

To construct the optimal delta evaluation tree for \( R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \), we first find the partition \( \{ P_1, P_2, \ldots, P_m \} \) of \( \{ R_1, R_2, \ldots, R_n \} \) that minimizes expression (12). Then the optimal delta evaluation tree for \( R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \) is constructed as follows. We make \( \Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) \) the new root node, and the optimal delta evaluation tree obtained for \( (\bowtie P_i)(1 \leq i \leq m) \) becomes the \( i \)th child from the left. Note that by the Principle of Optimality any subtree of an optimal delta evaluation tree must be also an optimal delta evaluation tree.

When we wish to apply the bottom-up based dynamic programming to expression (12), we need to compute \(|\{R_1, R_2, \ldots, R_n\}|, |\{R_1, R_2, \ldots, R_n\}|, |\{R_1, R_2, \ldots, R_n\}|, |\{R_1, R_2, \ldots, R_n\}|, |\{R_1, R_2, \ldots, R_n\}|, |\{R_1, R_2, \ldots, R_n\}|, \ldots \) and then we have \(|R_1 \bowtie R_2| = JS(R_1, R_2) \times |R_1| \times |R_2| \ldots \). We can easily extend this to any number of relations as follows [24]:

\[
|R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n| = \left( \prod_{1 \leq j, \sum_{i=1}^{j} i < j} JS(R_1, R_j) \right) \times |R_1| \times |R_2| \times \cdots \times |R_n|.
\]

For simplicity, let us assume that \(|(\Delta R \bowtie S) \cup (R' \bowtie \Delta S)|\) is equal to \(|\Delta R \bowtie S| + |R' \bowtie \Delta S|\) [4]. Then we have

\[
|\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)| = |\Delta(\bowtie P_1) \bowtie (\bowtie P_2) \bowtie \cdots \bowtie (\bowtie P_m)| + |(\bowtie P'_1) \bowtie \Delta(\bowtie P_2) \bowtie \cdots \bowtie (\bowtie P_m)| + \cdots \n+ |(\bowtie P'_1) \bowtie (\bowtie P'_2) \bowtie \cdots \bowtie \Delta(\bowtie P_m)|
\]

\[
= \prod_{1 \leq j, \sum_{i=1}^{j} i < j} JS(R_1, R_j) \times |\Delta(\bowtie P_1)| \times \prod_{R_k \in P_2} |R_k| \times \cdots \times \prod_{R_k \in P_m} |R_k| + \prod_{R_k \in P'_1} |R'_k| \times |\Delta(\bowtie P_2)| \times \cdots \times \prod_{R_k \in P'_2} |R'_k| \times \cdots \times |\Delta(\bowtie P_m)|.
\]

From the above equation, we can obtain \(|\Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n)|\).

Based on the above observations, our dynamic programming algorithm named as FindOptimalDeltaEvaluationTree is presented in Fig. 4. FindOptimalDeltaEvaluationTree finds out the optimal delta evaluation tree for a given view based on the bottom-up approach. The algorithm first constructs the optimal delta evaluation tree for each base relation, which is unique and trivial as you expect. Then, it iteratively constructs the optimal delta evaluation tree for the join of each subset of \( \{ R_1, R_2, \ldots, R_n \} \) from size 2 to \( n \) in stages. When we try to construct an optimal delta evaluation tree for the join of a subset of size \( i \), the algorithm uses optimal delta evaluation trees already obtained for joins of subsets of size 1 to \((i-1)\). Eventually, the algorithm finds out the optimal delta evaluation tree for a view defined over \( n \) base relations. For example, Fig. 3 shows how FindOptimalDeltaEvaluationTree finds the optimal delta evaluation tree for a view \( V' = R_1 \bowtie R_2 \bowtie R_3 \). Each tree in Fig. 3 represents the optimal delta evaluation tree for the join of a certain subset of \( \{ R_1, R_2, R_3 \} \).

4. Analysis of the proposed method

First, let us consider the time complexity of FindOptimalDeltaEvaluationTree. Given a view \( V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \), the algorithm considers all possible subsets of \( \{ R_1, R_2, \ldots, R_n \} \) to find out the optimal delta evaluation tree for the join of each subset of \( \{ R_1, R_2, \ldots, R_n \} \). Also, for each subset \( \{ R_v, R_v, \ldots, R_v \} \subseteq \{ R_1, R_2, \ldots, R_n \} \), the algorithm traces all possible partitions of \( \{ R_v, R_v, \ldots, R_v \} \) to construct the optimal delta evaluation tree for \( R_v \bowtie R_v \bowtie \ldots \bowtie R_v \). The number of all possible subsets of \( \{ R_1, R_2, \ldots, R_n \} \) is \( \sum_{i=1}^{n} C_i \), where \( C_i \) is the number of combinations of size \( i \) from \( n \) distinct objects, and the number of all possible partitions of a set of size \( i \) is \( B(i) \). Therefore, the worst time complexity of the algorithm is approximately \( O\left(\sum_{i=1}^{n} (C_i \times B(i))\right) \). However, in practice, most views in data warehouses are defined over less than 10 base relations. For example, in TPC-R, which is a standard benchmark for decision support systems, each of the queries is defined over at most 7 base relations.

\[1\] For the union operation, the size of the result is not easy to determine. One simple assumption is that the size of the result is the sum of the sizes of the arguments.
For $n \leq 7$, $B(n)$ is less than 900. Therefore, the algorithm can be used properly in finding out optimal delta evaluation trees for most data warehouse applications.

Now, let us compare our optimal delta evaluation method with previous incremental maintenance methods. As we mentioned in Section 1.1, expression (2) has been known to be the most efficient maintenance expression among others proposed so far. Hence, we compare our method with a method that uses expression (2). We will compare the number of accesses to base relations in both methods, and then analyze the processing cost of both methods under the linear work metric.

As we mentioned in Section 1.2, every base relation has to be accessed $(n-1)$ times to evaluate expression (2). On the contrary, the number of accesses to each base relation may vary between 1 and $(n-1)$ in our method.

As we mentioned in Section 1.2, every base relation has to be accessed $(n-1)$ times to evaluate expression (2). On the contrary, the number of accesses to each base relation may vary between 1 and $(n-1)$ in our method.

**Theorem 4.1.** For any delta evaluation expression of a view $V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$, the number of accesses to base relation $R_i$ (or $R'_i$) appears always include $\Delta(R_s \bowtie R_i \bowtie \ldots \bowtie R_u)$ where $R_i \notin \{R_s, R_u, \ldots, R_u\}$. Thus, the number of accesses to $R_i$ can not exceed the number of all possible $\Delta(R_s \bowtie R_i \bowtie \ldots \bowtie R_u)$, where $R_i \notin \{R_s, R_u, \ldots, R_u\}$ and $\{R_s, R_1, \ldots, R_u\} \subset \{R_1, R_2, \ldots, R_n\}$. Because any two $\{R_s, R_1, \ldots, R_u\}$ of such $\Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_u)$ are disjoint with each other, the number of all possible $\Delta(R_s \bowtie R_i \bowtie \ldots \bowtie R_u)$ is at most $|\{R_1, R_2, \ldots, R_n\} - \{R_i\}| = (n-1)$. Hence, this theorem may follow.

Thus, the number of accesses to each base relation in the delta evaluation expression is always less than or equal to that in expression (2).

As we have seen, while all base relation need to be accessed $(n-1)$ times equally in expression (2), the number of accesses to each base relation can vary in our method. This makes our method more flexible and advantageous than previous methods. Suppose that $R_6$ is much larger than the other base relations in Fig. 1. If we use a delta evaluation tree in Fig. 1-(a) rather than Fig. 1-(b), then we can reduce the cost of accessing the base relations considerably by reducing the number of accesses to $R_6$ from 5 to 1. Note that even though the number of accesses to $R_6$ is reduced to 1, the number of accesses to the other base relations, i.e., $R_1, R_2, R_3, R_4, R_5$, is still less than 5.
when \( \Delta \)

Consider a view method that uses expression (2) with respect to the intermediate results. This allows us to maintain more than one view. The following example motivates our approach for sharing overall cost of accessing relations even more. In this section, we present an algorithm that reuses common results generated during maintaining some views can be reused to maintain other views, we may significantly reduce the overall maintenance cost. Recall that, in expression (11), our method computes \( \Delta V \) first to compute the overall change of \( V \). This means that \( R_3 \) is accessed twice in expression (13), while \( R_3 \) is accessed only once but the temporary relation \( \Delta (R_1 \bowtie R_2) \) is accessed additionally in expression (13). Because we have assumed that \( |R_1| \ll |R_3| \) and \( |R_2| \ll |R_3| \), we can expect \( |\Delta (R_1 \bowtie R_2)| \ll |R_3| \) in general. Thus, the cost of expression (14) is less than the cost of expression (13) in most cases. Therefore, we can conclude that our method can find out more efficient maintenance expressions than expression (2) with respect to the linear cost metric.

As we mentioned before, the major limitation of expression (2) is that every base relation must be accessed \((n-1)\) times equally regardless of its size. Compared with that, the optimal delta evaluation method can save the cost considerably by reducing the cost of accessing such large relations, e.g., \( R_3 \) in expression (13).

5. Extension to 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 expression (11), our method computes \( \Delta (\bowtie P_1), \Delta (\bowtie P_2), \ldots, \Delta (\bowtie P_m) \) first to compute the overall change of \( V \). Thus, if we can reuse \( \Delta (\bowtie P_1) \) to maintain other views, we may reduce the overall cost of accessing relations even more. In this section, we present an algorithm that reuses common intermediate results to maintain more than one view. The following example motivates our approach for sharing intermediate results.

Consider \( V_1 = A \bowtie B \bowtie C \) and \( V_2 = B \bowtie C \bowtie D \). Fig. 4-(a) shows the optimal delta evaluation trees for \( V_1 \) and \( V_2 \) when \( \Delta V_1 \) and \( \Delta V_2 \) 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. Fig. 4-(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 $V = \{V_1, V_2, \ldots, V_n\}$ be a set of views to be maintained, and let $\text{MinCost}(\Delta V)$ denote the cost of the optimal delta evaluation tree for $V_i$. If we do not reuse any intermediate results, the total cost $\text{TotalCost}$ of computing $\Delta V_1, \Delta V_2, \ldots, \Delta V_n$ and is as follows.

\[
\text{TotalCost}(V) = \sum_{V_i \in V} \text{MinCost}(\Delta V_i).
\]  

(15)

Let $S$ denote a set of the intermediate results that are decided to be materialized and reused. For example, $S = \{\}$ in Fig. 4-(a) and $S = \{\Delta(B \bowtie C)\}$ in Fig. 4-(b). Let $\text{MinCost}(\Delta V_i | S)$ 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 $\text{MinCost}(\Delta V_i | S) \leq \text{MinCost}(\Delta V_i)$. Given a set $S$, the total cost $\text{TotalCost}(V|S)$ of computing $\Delta V_1, \Delta V_2, \ldots, \Delta V_n$ and using $S$ is as follows.

\[
\text{TotalCost}(V|S) = \sum_{S \in S} \text{MinCost}(S_i | S - \{S_i\}) + \sum_{V_i \in V} \text{MinCost}(\Delta V_i | S).
\]  

(16)

The first term corresponds to the cost of computing $S$. Note that $\text{MinCost}(S_i | S - \{S_i\})$ is used because $S_i$ should not be in $S$ when we compute $S_i$. The second term corresponds to the cost of computing $\Delta V_1, \Delta V_2, \ldots, \Delta V_n$ given that the expressions in $S$ are computed and materialized. To obtain $\text{MinCost}(\Delta V_i)$, we can use the $\text{FindOptimalDeltaEvaluationTree}$ procedure in Section 3.3. However, for $\text{MinCost}(\Delta V_i | S)$, we need to extend expression (12) in Section 3.3 to reflect $S$. Expression (12) is extended as follows:

\[
\text{Cost}(\Delta R_i | S) = 0 \quad \text{if} \quad \Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m) \in S
\]

\[
\text{Cost}(\Delta R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m | S) = \sum_{i=1}^{m} \text{Cost}(\Delta(\bowtie P_i) | S) + (|\Delta(\bowtie P_1)| + |P_2| + \cdots + |P_m|)
\]

\[
+ (|P_1| + |\Delta(\bowtie P_2)| + \cdots + |P_m|) + \cdots + (|P_1| + |P_2|)
\]

\[
+ \cdots + |\Delta(\bowtie P_m)|), \quad \text{otherwise}
\]

(17)

where $\text{Cost}(\Delta V_i | S)$ denotes the cost of processing expression $\Delta V$ given that expressions $S$ are materialized. Note that if we have $\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m)$ in $S$, $\text{Cost}(\Delta(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m) | S) = 0$. Using expression (17), we can obtain $\text{MinCost}(\Delta V_i | S)$ 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 $V = \{V_1, V_2, \ldots, V_n\}$, find a set of intermediate results $S$ to be materialized and reused, and the optimal delta evaluation trees for $V_1, V_2, \ldots, V_n$ that use $S$ such that $\text{TotalCost}(V|S)$ 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 [21,18,22,20]. 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. However, Ref. [21] 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 reused in a greedy manner. Our
The approach is similar to that of Ref. [20]. 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 intermediate results 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 $\text{TotalCost}(V|S)$, one of the simplest methods is to iterate over each subset of $C$ and select a subset of $C$ that minimizes $\text{TotalCost}(V|S)$. 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. Fig. 5 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. Experiments

We show in this section the results of various performance experiments among several view maintenance methods. The performance of each view maintenance method was measured by the time taken for updating a view by that method. In the experiments, we used the TPC-R benchmark 256MB RAM was used as the data warehouse environment. We defined two views, i.e., $V_1$ and $V_2$, each of which was defined as follows:

$V_1 = \text{CUSTOMER} \bowtie \text{ORDERS} \bowtie \text{LINEITEM}$
$\bowtie \text{SUPPLIER} \bowtie \text{NATION} \bowtie \text{REGION}$

$V_2 = \text{PART} \bowtie \text{SUPPLIER} \bowtie \text{LINEITEM} \bowtie \text{PARTSUPP}$
$\bowtie \text{ORDER} \bowtie \text{NATION}$.

---

**Procedure GreedyMultipleViews**

```plaintext
// Initialize */
C = \{Δ(R_1 \bowtie R_2 \bowtie ... \bowtie R_n) \mid \text{R}_1 \bowtie \text{R}_2 \bowtie ... \bowtie \text{R}_n \text{ is a common subexpression among two or more views in } V\}
S = \ø

// Main loop */
While (C ≠ \ø) do
    Choose $c \in C$ which minimizes $\text{TotalCost}(V|S \cup \{c\})$
    If ($\text{TotalCost}(V|S \cup \{c\}) < \text{TotalCost}(V|S)$) do
        $S = S \cup \{c\}; \ C = C - \{c\}$
    Else
        $C = \ø$
    End If
End While

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

Fig. 5. A greedy algorithm for multiple views.
The definitions of $V_1$ and $V_2$ are based on the TPC-R query $Q_5$ and $Q_9$ respectively, and all the base relations were created and populated using the TPC-R schema.

In the experiments, we compared our optimal delta evaluation method with the recomputation method and the incremental maintenance method proposed in Ref. [12]. The method proposed in Ref. [12] uses expression (2) to maintain a view. As we mentioned in Section 1.1, Ref. [12] has showed that expression (2) is the best strategy among existing incremental strategies. Let us call expression (2) and the method in Ref. [12] the $n$-term expression and the $n$-term method respectively. The $n$-term method uses an optimal $n$-term expression to maintain a view. According to Ref. [12], an optimal $n$-term expression is the one that propagates the changes of base relations in increasing order of the size of the deltas of the base relations, i.e., $|R'_i| - |R_i|$. In the experiments, we used optimal $n$-term expressions for the $n$-term method.

The time taken for executing each view maintenance method for $V_1$ and $V_2$ is shown in Fig. 6-(a) and (b) respectively. 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 [15,12]. Fig. 6 shows that our optimal delta evaluation method outperforms the other methods in our whole experiments, i.e., when changes are no greater than 20%.

The time reported in Fig. 6 includes optimization time. For the $n$-term method, optimization time means the time taken for finding an optimal $n$-term expression for a view. Similarly, for our method, optimization time means the time taken for finding an optimal delta evaluation tree for a view. However, for both methods, these optimizations took less than 2 s in our environment, which is negligible.

Table 1 shows the relative sizes of the base relations used in the experiments and the number of accesses to each base relation in the $n$-term method and our delta evaluation method for $V_1$ and $V_2$. From Table 1, we can see that the number of accesses to relatively large base relations by inserting new tuples to the base relations.

Fig. 6. The performance evaluation by varying the size of changes.

Fig. 7. The performance evaluation by scaling the size of base relations.
relations, e.g., LINEITEM, ORDERS, was reduced in our method. Consequently, our method becomes more efficient than the $n$-term method as we can see in Fig. 6.

According to the cost model specified in the paper, the size of base relations may considerably affect the performance of view maintenance strategies. Fig. 7 shows the performance results of the three methods when we scaled the size of the base relations from 100% to 500%. Again in this case, our proposed method outperforms the other methods for both $V_1$ and $V_2$. Hence, we can confirm that our delta evaluation method is more efficient than existing ones in maintaining materialized views.

So far, we have considered the performance of maintaining a single view. The performance result of maintaining multiple views is shown in Fig. 8. Here, our method that allows sharing of intermediate results (i.e., ours (sharing)) is compared with the one that does not allow sharing (i.e., ours (no sharing)). The $n$-term method is also compared in the experiment. Along with $V_1$ and $V_2$, we also used the following four views in the experiment, all of which are based on the TPC-R queries.

$$\begin{align*}
V_3 & = \text{PART} \otimes \text{SUPPLIER} \otimes \text{PARTSSUP} \otimes \text{NATION} \otimes \text{REGION} \\
V_4 & = \text{CUSTOMER} \otimes \text{ORDERS} \otimes \text{LINEITEM} \\
V_5 & = \text{NATION} \otimes \text{LINEITEM} \otimes \text{ORDERS} \otimes \text{SUPPLIER} \\
V_6 & = \text{NATION} \otimes \text{CUSTOMER} \otimes \text{ORDERS} \otimes \text{LINEITEM}.
\end{align*}$$

The time reported in Fig. 8 is the optimization time plus the total time for updating all the six views. i.e., $V_1$, $V_2$, ..., $V_6$ and our optimal delta evaluation method performs better than the $n$-term method in this case also. As expected our extension to multiple views that share common intermediate results outperforms all the other methods.

### 7. Conclusion

Data warehouses store a large amount of summarized data to support decision making process. These summarized data can be seen as materialized views defined over some data sources. When data sources change, these 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. We presented the optimal delta evaluation method that can maintain materialized views efficiently in the data warehouse environment.

The concept of the delta evaluation tree with consideration of the size of each base relation makes it possible to minimize the cost of maintaining views. As a result, the experimental results show the efficiency of the proposed method compared with previous methods. Moreover, the intermediate results of delta evaluation trees can be shared among views. We have also developed an extension such that intermediate results of delta evaluation trees can be reused when multiple views are updated. We showed through experiment that the proposed extension gives even more benefits than the method that maintains each view separately.

### References


Ki Yong Lee is a senior engineer at Samsung Electronics Co., Korea. He received his B.S. and M.S. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1998 and 2000, respectively, and his Ph.D. degree in Computer Science from KAIST in 2006. His research interests include database systems, data warehousing, OLAP and embedded software.

Jin Hyun Son is an assistant professor of the Department of Computer Science and Engineering at Hanyang University, Korea, from 2001. He was a postdoctoral researcher in the Division of Computer Science at Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea. He received his B.S. degree in Computer Science from Sogang University, Seoul, Korea, in 1996, and his M.S. and Ph.D. degrees in Computer Science from KAIST in 1998 and 2001, respectively. His research interests include database systems, distributed processing, business process management and embedded software.

Myoung Ho Kim received his B.S. and M.S. degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1982 and 1984, respectively, and his Ph.D. degree in Computer Science from Michigan State University, East Lansing, MI, in 1989. In 1989 he joined the faculty of the Department of Computer Science at KAIST, Taejon, Korea, where currently he is a full professor. His research interests include database systems, data stream processing, sensor networks, mobile computing, OLAP, XML, information retrieval, workflow and distributed processing. He is a member of the ACM and IEEE Computer Society.