ELSEVIER

Decision Support Systems 32 (2002) 379–399

Decision Support Systems

www.elsevier.com/locate/dsw

Finding an efficient rewriting of OLAP queries using materialized views in data warehouses

Chang-Sup Park *, Myoung Ho Kim, Yoon-Joon Lee

Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Kusung-dong, Yusung-gu, Taejon, 305-701, South Korea

Received 1 January 2001; received in revised form 1 March 2001; accepted 1 June 2001

Abstract

OLAP queries involve a lot of aggregations on a large amount of data in data warehouses. To process expensive OLAP queries efficiently, we propose a new method to rewrite a given OLAP query using various kinds of materialized views which already exist in data warehouses. We first define the normal forms of OLAP queries and materialized views based on the selection and aggregation granularities, which are derived from the lattice of dimension hierarchies. Conditions for usability of materialized views in rewriting a given query are specified by relationships between the components of their normal forms. We present a rewriting algorithm for OLAP queries that can effectively utilize materialized views having different selection granularities, selection regions, and aggregation granularities together. We also propose an algorithm to find a set of materialized views that results in a rewritten query which can be executed efficiently. We show the effectiveness and performance of the algorithm experimentally.

Keywords: Query rewriting; Materialized view; OLAP; Data warehouse

1. Introduction

Data Warehouses (DWs) integrate information from multiple operational databases and are often used to support On-Line Analytical Processing (OLAP) [4]. A DW tends to have a huge amount of raw data. Queries used in OLAP are much more complex than those used in traditional OLTP applications and have many different features. They generally consist of multi-dimensional grouping and aggregation operations. The large size of DWs and the complexity of OLAP queries considerably increase the execution cost of the queries and have a critical effect on performance and productivity of decision support systems.

For efficient processing of OLAP queries, a commonly used approach is to store the results of frequently issued queries in summary tables, i.e., Materialized Views (MVs), and make use of them to evaluate other queries. This approach involves selecting views to materialize and rewriting queries using MVs. Most of the proposed materialized view selection techniques, such as Refs. [2,10,12,13,15,24], select a set of views that optimize query evaluation cost for a given set of queries under given storage space limitation or MV maintenance cost constraint. To answer queries that are not relevant to those considered in view selection process efficiently, a query rewriting strategies for con-

* Corresponding author.
E-mail addresses: parkcs@dbserver.kaist.ac.kr (C.-S. Park), mhkim@dbserver.kaist.ac.kr (M.H. Kim), yjlee@dbserver.kaist.ac.kr (Y.-J. Lee).

0167-9236/02/$ - see front matter
PII: S0167-9236(01)00123-3

junctive queries and aggregation queries have been proposed in the literature of relational databases, data integration, and data warehouses [3,5,11,16,22,23,26]. Few of them, however, exploited the characteristics of DWs and OLAP queries effectively and utilized existing MVs sufficiently.

In this paper, we propose a new algorithm for rewriting OLAP queries which improves the usability of MVs significantly in contrast to the previous studies. We consider typical OLAP queries that aggregate raw data by the attributes in dimension hierarchies and define a normal form of the queries based on the lattice of dimension hierarchies. Then we present conditions on which an MV can be used in rewriting a given query, which are specified by relationships between the components of their normal forms, and suggest the rewriting algorithm for normal form OLAP queries.

The main contribution of this paper are as follows:

- Aggregate MVs defined by an arbitrary region of selection can be used in our method. That is, we can use the results of grouping and aggregation of the raw data selected by an arbitrary range of values of dimensional attributes.
- We exploit meta-information of DW schema effectively. Dimension hierarchies in DWs are implicity used in constructing OLAP queries. By using such semantic information, we can achieve query rewriting that utilizes various kinds of MVs together. Ad hoc queries that were not considered at view selection process can be also rewritten using MVs by the proposed method.
- For a given OLAP query, there can be many equivalent rewritings using different MVs in various ways. Their execution costs, in general, are different from one another. We propose a heuristic algorithm to find a set of MVs and their query regions that result in an efficient query plan.

The rest of the paper is organized as follows. We first present some examples motivating our studies in Section 2. In Section 3, we define the normal forms of OLAP queries and MVs considered in this paper. In Section 4, we describe our method for rewriting OLAP queries using MVs. We propose an algorithm to select an efficient set of MVs for query rewriting in Section 5 and present experimental results in Section 6. Related work is discussed in Section 7, and we draw conclusions in Section 8.

2. Motivating examples

We present examples to show how OLAP queries can be rewritten using various MVs in DWs.

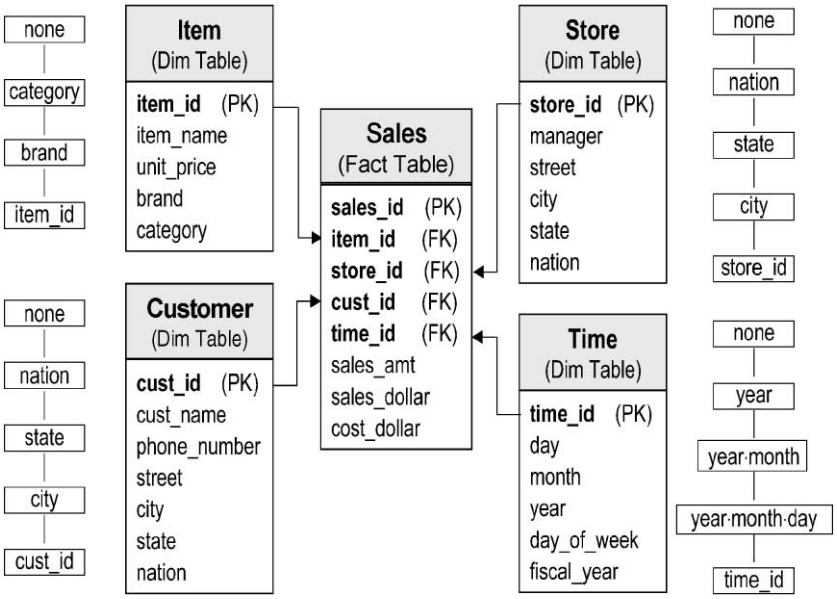

Fig. 1. An example data warehouse schema.

MV_1: total sales of stores from 1997 by state and year

```
SELECT  state, year, SUM(sales_dollar)
        AS sum_dollar1
FROM    Sales, Store, Time
WHERE   Sales.store_id = Store.store_id
        AND Sales.time_id = Time.time_id
        AND Time.year ≥ 1997
GROUP BY state, year
```

MV_2: total sales of the stores in the USA by state and month

```
SELECT  state, year, month,
        SUM(sales_dollar) AS sum_dollar2
FROM    Sales, Store, Time
WHERE   Sales.store_id = Store.store_id
        AND Sales.time_id = Time.time_id
        AND Store.nation = 'USA'
GROUP BY state, year, month
```

MV_3: total sales of the stores in Canada to 1996 by city and month

```
SELECT    city, year, month, SUM(sales_dollar) AS sum_dollar3
FROM      Sales, Store, Time
WHERE     Sales.store_id = Store.store_id AND Sales.time_id = Time.time_id
          AND Store.nation = 'Canada' AND Time.year <= 1996
GROUP BY  city, year, month
```

Fig. 2. Example materialized views.

Example 1. Consider a DW with sales data of a large chain of department stores, whose schema is shown in Fig. 1. The DW consists of a fact table and four-dimension tables and has a dimension hierarchy in each dimension table. We assume that three MVs are available in the DW as shown in Fig. 2.

Q_1:

```
SELECT  state, year, SUM(sales_dollar)
FROM    Sales, Store, Time
WHERE   Sales.store_id = Store.store_id
        AND Sales.time_id = Time.time_id
        AND (Store.nation = 'USA' OR
             Store.nation = 'Canada' )
        AND Time.year ≥ 1996
        AND Time.year ≤ 1999
GROUP BY state, year
```

Q_1':

```
(SELECT state, year, sum_dollar1
 FROM   MV1, (SELECT DISTINCT state
              FROM   Store
              WHERE  nation = 'USA' OR
                     nation = 'Canada') S
 WHERE MV1.state = S.state AND year ≤ 1999)
 UNION
(SELECT state, year, SUM(sum_dollar2)
 FROM   MV2
 WHERE year = 1996
 GROUP BY state, year)
 UNION
(SELECT S.state, year, SUM(sum_dollar3)
 FROM   MV3, (SELECT DISTINCT city, state
              FROM   Store) S
 WHERE MV3.city = S.city AND year = 1996
 GROUP BY S.state, year)
```

Fig. 3. Example query Q_1 and its rewritten query Q_1'.

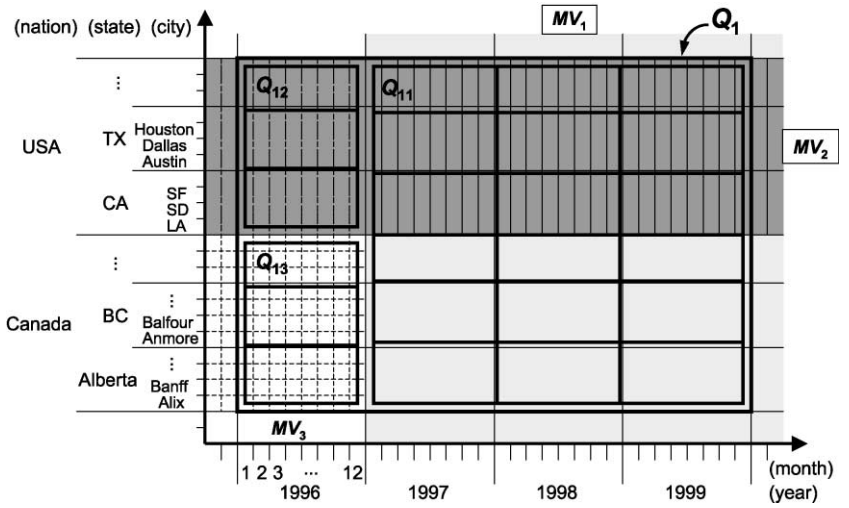

Fig. 4. The aggregate groups of three query blocks in Q_1' .

Consider the OLAP query Q_1 in Fig. 3, which asks for the total sales of the stores in the USA or Canada from 1996 to 1999 by state and year. Q_1 can be rewritten to the query Q_1' in Fig. 3, which uses the three MVs instead of the fact table Sales. Q_1' consists of three query blocks, whose results are combined by union. Each query block contains a different MV and computes a part of aggregate groups of Q_1 as shown in Fig. 4. Specifically, the first query block computes from MV_1 the total sales of the stores in the USA or Canada from 1997 to 1999 by state and year. The second and the third one use MV_2 and MV_3, respectively, to compute the total sales of the stores in the USA and Canada in 1996 by state. Since the three sets of groups are disjoint and the union of them is equal to the set of groups computed by Q_1, we can

Q_2:

```
SELECT  state, SUM(sales_dollar)
FROM    Sales, Store
WHERE   Sales.store_id = Store.store_id
        AND Store.nation = 'Canada'
GROUP BY state
```

Q_2':

```
SELECT state, SUM(psum)
FROM (SELECT state, SUM(sum_dollar1) AS psum
      FROM   MV1,
             (SELECT DISTINCT state
              FROM   Store
              WHERE nation = 'Canada')  S1
      WHERE  MV1.state = S1.state
      GROUP BY state
      UNION ALL
      SELECT  state, SUM(sum_dollar3) AS psum
      FROM    MV3,
              (SELECT DISTINCT city, state
               FROM   Store)  S2
      WHERE   MV3.city = S2.city
      GROUP BY state)
GROUP BY state
```

Fig. 5. Example query Q_2 and its rewritten query Q_2' .

obtain the same result of Q_1 by taking the union of them as in Q_1'.

The next example shows another type of query rewriting.

Example 2. Consider the Q_2 in Fig. 5 over the sales DW in Fig. 1. Q_2 can be rewritten to the equivalent query Q_2' in Fig. 5, which uses MV_1 and MV_3 instead of the fact table Sales.

Q_2 asks for the total sales of the stores in Canada in *all* years by state. However, MV_1 has the total sales of the stores only from 1997, and MV_3 has those only to 1996. That is since MV_1 and MV_3 are the results of aggregations over a part of the raw data contained in each aggregate group of Q_2, neither of them can compute the aggregation of Q_2. However, as shown in Fig. 6, two sets of the raw data selected by MV_1 and MV_3 are disjoint, and the union of them is equal to the set of the raw data for the aggregate groups of Q_2. Thus, the result of Q_2 can be obtained by computing aggregations over MV_1 and MV_3 for all aggregate groups of Q_2 and then aggregating the partial results of the same groups once more, as shown in Q_2'.

The rewritings in these two examples cannot be obtained by other methods proposed earlier. Our method proposed in this paper can achieve these types of rewritings. We will revisit these examples in Section 4 to illustrate how our algorithm can perform the rewritings.

3. OLAP queries and materialized views

3.1. The lattice of dimension hierarchies

We consider DWs that have a star schema [4] consisting of one fact table and d dimension tables like the one in Fig. 1. The fact table has a foreign key to each dimension table and measure attributes on which aggregations are performed. The tuples in the fact table are called the *raw data*. We suppose that we can obtain hierarchical classification information from dimension tables and consider one dimension hierarchy for each dimension table. A dimension hierarchy DH_i of a dimension table DT_i, defined as $DH_I=(L_0^i, L_1^i,\ldots, L_h^i, \text{none})$, is an ordered set of dimension levels. A dimension level L_j^i is a set of attributes from DT_i. We call j of L_j^i its *height*. L_0^i is the lowest level which equals the primary key of DT_i. None denotes the highest level and is an empty set. There exists a functional dependency $L_{j-1}^i \rightarrow L_j^i$ between L_{j-1}^i and L_j^i $(1 \le j \le h)$. Each dimension level is used as a criterion by which raw data are grouped for aggregation. We call the attributes in dimension levels the *dimensional attributes*.

The Cartesian product of all dimension hierarchies, defined as $DH=DH_1 \times DH_2 \times \ldots \times DH_d$, is a class of the ordered sets of dimension levels from all different dimension hierarchies. There exists a partial

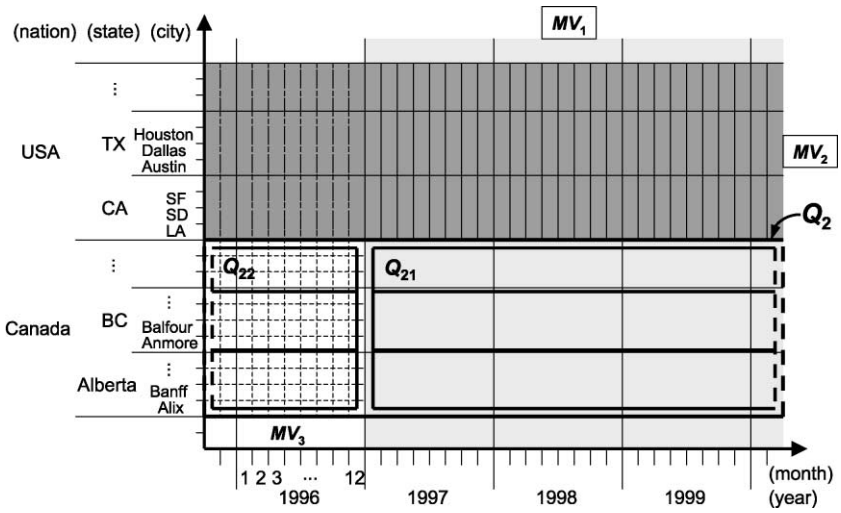

Fig. 6. The aggregate groups of the query blocks in Q_2'.

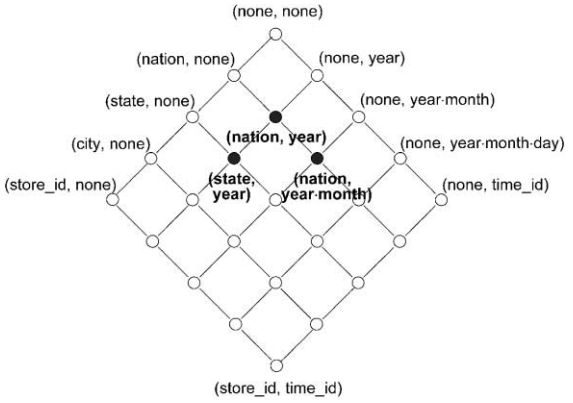

Fig. 7. An example DH lattice.

ordering relation $\leq$ among the elements in *DH*, defined as

$$(L_{l_1}^1, L_{l_2}^2, \ldots, L_{l_d}^d) \leq (L_{m_1}^1, L_{m_2}^2, \ldots, L_{m_d}^d)$$

if and only if $L_{l_i}^i \rightarrow L_{m_i}^i$ or $L_{l_i}^i = L_{m_i}^i$ for all $1 \leq i \leq d$.

Two additional relations $<$ and $<>$ can be derived from $\leq$ as follows.

$(L_{l_1}^1, L_{l_2}^2, \ldots, L_{l_d}^d) < (L_{m_1}^1, L_{m_2}^2, \ldots, L_{m_d}^d)$ if and only if $(L_{l_1}^1, L_{l_2}^2, \ldots, L_{l_d}^d) \leq (L_{m_1}^1, L_{m_2}^2, \ldots, L_{m_d}^d)$ but there exists j $(1 \leq j \leq d)$ such that $L_{l_j}^j \neq L_{m_j}^j$.
$(L_{l_1}^1, L_{l_2}^2, \ldots, L_{l_d}^d) <> (L_{m_1}^1, L_{m_2}^2, \ldots, L_{m_d}^d)$ if and only if neither $(L_{l_1}^1, L_{l_2}^2, \ldots, L_{l_d}^d) \leq (L_{m_1}^1, L_{m_2}^2, \ldots, L_{m_d}^d)$ nor $(L_{l_1}^1, L_{l_2}^2, \ldots, L_{l_d}^d) \geq (L_{m_1}^1, L_{m_2}^2, \ldots, L_{m_d}^d)$.

DH can be represented as a lattice, called the *Dimension Hierarchies (DH) lattice* in this paper.[1] Each node in the lattice means a criterion for selecting or grouping raw data. Fig. 7 shows a DH lattice derived from dimension hierarchies in Store and Time in Fig. 1.[2]

3.2. The normal form of OLAP queries

The OLAP queries we consider in this paper are unnested single-block aggregate queries over the base tables. Queries over MVs can be transformed to those that contain only base tables by unfolding the MVs. The target queries can be characterized by a set of selection attributes, a selection predicate, a set of grouping attributes, a set of projection attributes, a set of aggregate functions, and a condition on the values of aggregate functions and the grouping attributes.

The selection attributes are those contained in the selection predicate of the query. We assume that the set of selection attributes is a union of dimension levels from some different dimension hierarchies.[3] Then, it can be mapped to a node in the DH lattice, i.e., an ordered set of dimension levels, which we call the *Selection Granularity* (SG) of the query. The selection predicate is a Boolean combination of comparison predicates on the selection attributes. A comparison predicate has the form α *op* c, where α is a dimensional attribute, c is a constant, and $op \in \{<, \leq, =, \geq, >\}$. In the selection predicate expressed in a disjunctive normal form, each conjunct can be geometrically represented as a hyper-rectangle in the d-dimensional domain space of dimensional attributes derived from the dimension hierarchies. Thus, the selection predicate can be considered a set of hyper-rectangles, called the *selection region* of the query. The set of grouping attributes is used for grouping raw data. Like the set of selection attributes, it can be mapped to a node in the DH lattice, which we call the *Aggregation Granularity* (AG) of the query. The set of projection attributes is supposed to be identical to the set of grouping attributes.

We propose a normal form of the considered OLAP queries using these components of the queries.

Definition 1. The normal form of an OLAP query Q is defined as

$$Q(SG, R, AG, AGG, HAV),$$

where

- $SG=(S_1, S_2, \ldots, S_d)$ is the selection granularity of Q, where S_i $(1 \leq i \leq d)$ is a dimension level in the dimension hierarchy DH_i.

[1] A similar lattice framework was proposed in Ref. [13], which represents a dependence relation among a set of views.

[2] For simplicity in notation, we denote a dimension level $\{a_1, a_2, \ldots, a_n\}$ by $a_1 \cdot a_2 \cdot \ldots \cdot a_n$.

[3] The method proposed in this paper can be extended to the case where the set of selection attributes includes the attributes in different levels in the same dimension hierarchy.

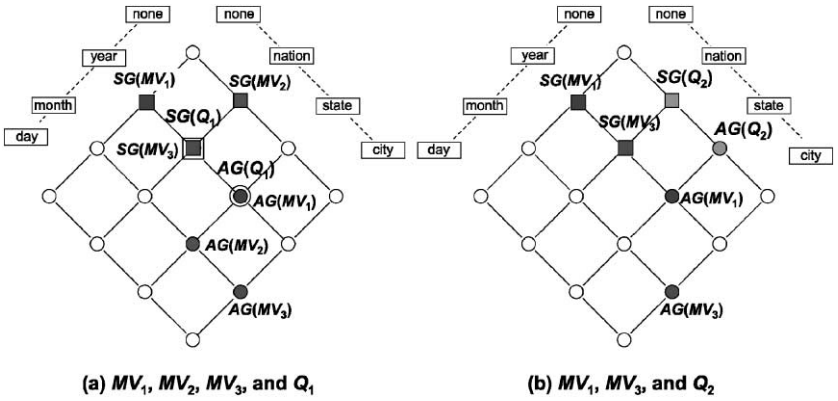

Fig. 8. The SGs and AGs of the queries and MVs in Example 1 and Example 2.

• $R=\{R_i\}$ is the selection region of Q, which is a set of hyper-rectangles. A hyper-rectangle $R_i=(I_{i1}, I_{i2}, \ldots, I_{id})$ is an ordered set of d intervals of the values of the dimension levels obtained from the selection predicate of Q. I_{ij} denotes an interval of a dimension level in DH_j, which can be open, closed, or half-closed. The endpoints of the interval are specified by the concatenation of all values of the higher levels in the dimension hierarchy. We denote unspecified left (right) endpoints of intervals by $-\infty$ $(+\infty)$.

• $AG=(A_1, A_2, \ldots, A_d)$ is the aggregation granularity of Q, where A_i $(1 \le i \le d)$ is a dimension level in the dimension hierarchy DH_i.

• $AGG=\{agg(m) \mid agg \in \{\text{MIN, MAX, SUM, COUNT}\}$ and m is a measure attribute$\}$.[4]

• HAV is a logical formula of comparison predicates on the attributes in AG and the aggregate functions in AGG. It implies the HAVING condition of Q in SQL. We denote an empty condition by *null*.

$SG(Q)$, $R(Q)$, $AG(Q)$, $AGG(Q)$, and $HAV(Q)$ denote SG, R, AG, AGG, and HAV in the normal form of Q, respectively. For simplicity in notation, we also use $SG(Q)$ and $AG(Q)$ to denote the union of the dimension levels in them. $AG(Q, i)$ and $SG(Q, i)$ $(1 \le i \le d)$ denote the dimension levels from DH_i contained in $AG(Q)$ and $SG(Q)$, respectively.

[4] We do not consider AVG in this paper since it can be computed using SUM and COUNT.

In this paper, we consider MVs that store the results of the normal form OLAP queries which satisfy $SG(Q) \ge AG(Q)$ and $HAV(Q)=null$. MVs that do not meet the conditions are not very useful for rewriting other queries. The normal form of a materialized view MV, denoted by $MV(SG, R, AG, AGG)$, is defined in the similar way.

Example 3. The normal forms of the MVs and the queries in Example 1 and Example 2 are as follows.

• $MV_1(SG, R, AG, AGG)=MV_1$ ((none, year, none, none), {$(-\infty, +\infty)$, [1997, $+\infty$), $(-\infty, +\infty)$, $(-\infty, +\infty)$)}, (state, year, none, none), {SUM(sales_dollar)}).

• $MV_2(SG, R, AG, AGG)=MV_2$ ((nation, none, none, none), {(['USA', 'USA'], $(-\infty, +\infty)$, $(-\infty, +\infty)$, $(-\infty, +\infty)$, $(-\infty, +\infty)$)}, (state, year·month, none, none), {SUM(sales_dollar)}).

• $MV_3(SG, R, AG, AGG)=MV_3$ ((nation, year, none, none), {(['Canada', 'Canada'], $(-\infty$, 1996], $(-\infty, +\infty)$, $(-\infty, +\infty)$)}, (city, year·month, none, none), {SUM(sales_dollar)}).

• $Q_1(SG, R, AGG, HAV)=Q_1$ ((nation, year, none, none), {(['USA', 'USA'], [1996, 1999], $(-\infty, +\infty)$, $(-\infty, +\infty)$) (['Canada', 'Canada'], [1996, 1999], $(-\infty, +\infty)$, $(-\infty, +\infty)$)}, (state, year, none, none), {SUM(sales_dollar)}, *null*).

• $Q_2(SG, R, AGG, HAV)=Q_2$ ((nation, none, none, none), {(['Canada', 'Canada'], $(-\infty, +\infty)$, $(-\infty, +\infty)$, $(-\infty, +\infty)$)}, (state, none, none, none), {SUM(sales_dollar)}, *null*).

Fig. 8 shows the SGs and AGs of the MVs and queries in Example 1 and Example 2 on the DH lattice.

We define two operations between selection regions of queries and MVs which are used in Section 4.

Definition 2. Let Q and MV be a query and a materialized view. The region intesection $\cap^*$ between the selection region of Q and that of MV is defined as

$$R(Q) \cap^* R(MV) = \{R_i \cap R_j \mid R_i \in R(Q), R_j \in R(MV)\},$$

where $R_i \cap R_j$ is a set of hyper-rectangle which is the intersection of R_i and R_j. The region differences $-^*$ of the selection region of Q and that of MV is defined as

$$R(Q) -^* R(MV) = \{R_k \mid R_k \in (R_i - R_j), R_j \in R(Q), R_j \in R(MV)\},$$

where $R_i - R_j$ is a set of hyper-rectangles whose union is identical to the difference of R_i and R_j. The intersection and difference of the selections of two MVs are defined in the way.

The intersection and difference of two hyper-rectangles can be obtained by the algorithms proposed in computational geometry (e.g., Ref. [17]). Since the endpoints of hyper-rectangles are represented as a concatenation of all values of the higher levels in dimension hierarchies as defined in Definition 1, the operations can be applied on the queries and MVs having different selection granularities. The granularity of the result is equal to the Greatest Lower Bound (GLB) of the selection granularities of two operands. We say that a tuple in an MV or the fact table is contained in a selection region R if it is selected by the selection predicate for R which is specified on dimensional attributes involved in R.

4. Query rewriting using materialized views

Given a query Q, a query Q' is called a rewriting, or a rewritten query, of Q that uses a materialized view MV if: (a) Q' and Q compute the same result for any given database, and (b) Q' contains MV in the FROM clause of one of its query blocks [23]. If there exists such a rewritten query, we say that MV is *usable* in rewriting Q. In this section, we propose an algorithm for rewriting normal form OLAP queries using different classes of normal form MVs. Rewritten queries are expressed in SQL.

4.1.Candidate materialized views

To rewrite a given OLAP query Q, we have to find MVs that can be used to compute a part of the aggregate groups of Q. We present sufficient conditions on which a materialized view can be used in rewriting Q in the following definition.

Definition 3. An MV is a candidate MV for a query Q if it satisfies the following four conditions: $R(MV) \cap^* R(Q) \neq \phi$, $AG(MV) \leq SG(Q)$, $AG(MV) \leq AG(Q)$, and $AGG(MV) \supseteq AGG(Q)$.

The usability of the candidate MVs can be proved by the following results.

Lemma 1. If $AG(MV) \leq SG(Q)$, then the tuples in MV that are contained in $R(MV) \cap^* R(Q)$ are the result of aggregation over the tuples in the fact table that are contained in $R(MV) \cap^* R(Q)$.

Proof. For a tuple t' in MV, let $G(t')$ be the set of tuples in the table FT that constitute the aggregate group for t'. The granularity of $R(MV) \cap^* R(Q)$ is $GLB(SG(MV), SG(Q))$. If $AG(MV) \leq SG(Q)$, then with the constraint of $AG(MV) \leq SG(MV)$ in the definition of MV, we have

$$AG(FT) \leq AG(MV) \leq GLB(SG(MV), SG(Q)).$$

The relation $\leq$ implies the functional dependencies $AG(FT) \rightarrow AG(MV)$ and $AG(MV) \rightarrow GLB(SG(MV), SG(Q))$, and $AG(FT) \rightarrow GLB(SG(MV), SG(Q))$ is derived from them by transitivity of functional dependencies. Thus,

for all t' in MV contained in $R(MV) \cap^* R(Q)$ and for all t in FT such that $t \in G(t')$, t is also contained in $R(MV) \cap^* R(Q)$. Inversely, for all t in FT that is contained in $R(MV) \cap^* R(Q)$, there exists a tuple t' in MV such that $t \in G(t')$ and t' is also contained in $R(MV) \cap^* R(Q)$. Therefore, the lemma follows. □

Lemma 2. Let S' be a set of tuples from MV and S be the set of tuples from the fact table which constitute the aggregate groups for the tuples in S'. If $AG(MV) \leq AG(Q)$, the result of aggregation over the tuples in S by $AG(Q)$ and an aggregate function $agg(m)$ in $AGG(Q) \cap AGG(MV)$ is equal to the result of aggregation over the tuples in S' by $AG(Q)$ and an aggregate function $agg'(m')$, where agg' is equal to agg if agg is MIN, MAX, or SUM and agg' is SUM if agg is COUNT, and m' is the result attribute for $agg(m)$.

Proof. $AG(FT) \leq AG(Q)$ implies functional dependencies $AG(FT) \rightarrow AG(MV)$ and $AG(MV) \rightarrow AG(Q)$. Consider an equivalent relation R on S such that $(t_1, t_2) \in R$ if and only if t_1 and t_2 have the same value of $AG(MV)$, after being joined with related dimension tables, if necessary. By $AG(FT) \rightarrow AG(MV)$, the aggregate groups for the tuples in S' derive a partition π_1 of S induced by R. Similarly, we consider an equivalent relation R' on S' such that $(t_1', t_2') \in R'$ if and only if t_1' and t_2' have the same value of $AG(Q)$, after being joined with related dimension tables, if necessary. Then, by $AG(MV) \rightarrow AG(Q)$, the aggregate groups for the result S_1'' of aggregation over S' by $AG(Q)$ derive a partition π_2 of S' induced by R'. We have $AG(FT) \rightarrow AG(Q)$ by the transitivity of functional dependencies. Thus, the aggregate groups for the result S_2'' of aggregation over S by $AG(Q)$ derive a partition π_3 of S, and π_1 is a refinement of π_3. Since all the considered aggregate functions, i.e., MIN, MAX, SUM, and COUNT, are distributive functions [9], for all t_2'' in S_2'', there exists t_1'' in S_1'' such that

$$
\begin{aligned}
&agg(\{G(\pi_3, t_2'') \mid \text{a block of } \pi_3 \text{ which is an aggregate group for } t_2'' \text{ in } S_2''\}) \\
&\quad = agg'(\{agg(\{G(\pi_1, t' \mid \text{a block of } \pi_1 \text{ which is an aggregate group for } t' \text{ in } S' \text{ and } G(\pi_1, t') \subseteq G(\pi_3, t_2'')\})\}) \\
&\quad = agg'(\{G(\pi_2, t_1'') \mid \text{a block of } \pi_2 \text{ which is an aggregate group for } t_1'' \text{ in } S_1''\}),
\end{aligned}
$$

where agg' is equal to agg if agg is MIN, MAX, or SUM, and agg' is SUM if agg is COUNT. The converse also holds. Therefore, S_1'' and S_2'' are equal to each other. □

Theorem 1. Given an OLAP query Q, a candidate materialized view MV for Q is usable in rewriting Q.

Proof. $R(MV) \cap^* R(Q) \neq \phi$ means that there exist raw data that satisfy both of the selection predicates of MV and Q. $R(Q)$ can be divided into two disjoint selection regions, $R(MV) \cap^* R(Q)$ and $R(Q) -^* R(MV)$, which are evaluated by a query block containing MV and the fact table FT, respectively. We first consider the query block for MV. By Lemma 1, the set of tuples in MV that are selected by $R(MV) \cap^* R(Q)$ reflects the aggregation over the tuples in the fact table that are contained in $R(MV) \cap^* R(Q)$. By Lemma 2 and $AGG(MV) \supseteq AGG(Q)$, aggregating the selected tuples in MV by $AG(Q)$ gives the same result of the aggregation of Q. Therefore, we can compute the aggregation of Q for the selection region $R(MV) \cap^* R(Q)$ using MV. The aggregation for $R(Q) -^* R(MV)$ can be obtained from FT. □

The detailed algorithm for generating query blocks for MVs and the fact table and integrating them into a rewritten query is given in Section 4.2. For notational convenience, we regard the fact table as a kind of candidate MV for all queries in the rest of the paper and denote the set of the candidate MVs for a query Q by $V(Q)$. Fig. 9 shows the possible aggregation granularities of the candidate MVs for a query Q, which satisfy $AG(MV) \leq SG(Q)$ and $AG(MV) \leq AG(Q)$.

4.2. The query rewriting method

A given OLAP query Q can be rewritten using a set of the candidate MVs in $V(Q)$. Our rewriting method consists of three main steps. [21]

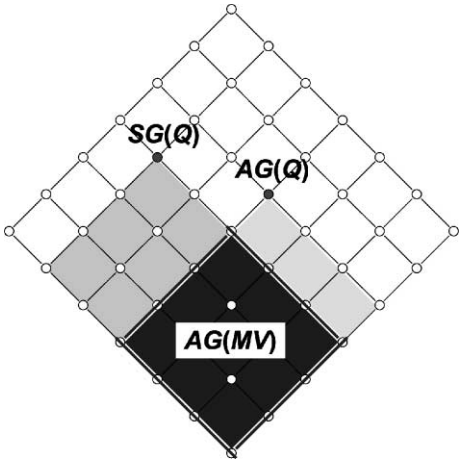

Fig. 9. The area of the AGs of possible candidate MVs for a query Q.

4.2.1. Step 1: selecting materialized views

In the first step, we select MVs from $V(Q)$ that will be actually used in a rewriting of Q and also determine the query regions for the selected MVs. In general, a rewritten query over a materialized view MV_i selects tuples in MV_i satisfying some selection predicate. The selection predicate can be represented as a selection region, which is called the query region for MV_i and denoted by $QR(MV_i)$. It must be subsumed in $R(MV_i) \cap^* R(Q)$. All query regions for the selected MVs must not overlap one another and cover the selection region of Q together. That is, if we let $S(Q)$ be a set of MVs selected for rewriting Q, the query regions for the MVs in $S(Q)$ must satisfy the following conditions:

$$QR(MV_i) -^* (R(MV_i) \cap^* R(Q)) = \phi \text{ for all } MV_1 \in S(Q),\ QR(MV_i) \cap^* QR(MV_j) = \phi \text{ for all } MV_i, MV_j \in S(Q) \text{ such that } MV_i \neq MV_j, \text{ and } R(Q) -^* \bigcup_{MV_i \in S(Q)} QR(MV_i) = \phi.$$

In general, there can be many equivalent rewritings containing a different set of candidate MVs, which differ greatly in execution performance. In Section 5, we will formalize the problem of selecting the candidate MVs and the query regions for them that constitute an efficient rewritten query. We will also provide an effective heuristic algorithm to solve the problem.

4.2.2. Step 2: generating query blocks

For each materialized view MV selected in Step 1, we generate a query block using its query region $QR(MV)$. We first consider the following normal form query over the fact table,

$$Q_{MV}(SG', QR(MV), AG(Q), AGG(Q), HAV(Q)),$$

where SG' in the granularity of $QR(MV)$. The query block for MV must be equivalent to Q_{MV} and defined over MV instead of the fact table. Since all the attributes in MV except for the results of aggregations are included in $AG(MV)$, if an attribute in SG' or $AG(Q)$ is not included in $AG(MV)$, a join operation between MV and the dimension table DT_i containing the attribute is required to perform selection by $QR(MV)$ or grouping by $AG(Q)$. However, if the join attributes are not foreign key referring to DT_i, the join may result in duplication of the same

tuples. In order to prevent it, we first evaluate a duplicate-eliminating (distinct) selection query over DT_i and then perform a join of MV with the result of the query, which will be nested in the query block of MV. A selection predicate on the attributes in DT_i that is subsumed by the selection predicate for $QR(MV)$ can be included in the nested subquery. As a result, the query block for MV is formalized in the following definition.

Definition 4. Given a query Q, a materialized view in MV in $V(Q)$, and a query region $QR(MV)$, the query block for MV, $QB(MV)$, has the form,

SELECT	P, AGG
FROM	R
WHERE	JC, SC
GROUP BY	G
HAVING	HC

where the components are defined as follows:

$$P = \bigcup_{1 \le i \le d} AG(Q, i)$$

$$AGG = \begin{cases} \{agg_i'\ (m')AS\ alias_i \mid m' \text{ is an attribute in } MV \text{ that is the result of } agg_i(m) \text{in } AGG(Q), \\ agg_i' = agg_i \text{ if } agg_i \in \{MIN, MAX, SUM\} \text{ and } agg_i' = SUM \text{ if } agg_i = COUNT\}, \\ \qquad\qquad \text{if } AG(Q) > AG(MV) \\ \{m'\ AS\ alias_i \mid m' \text{ is an attribute in } MV \text{ that is the result of } agg_i(m) \text{ in } AGG(Q)\}, \\ \qquad\qquad \text{if } AG(Q) = AG(MV) \end{cases}$$

$$R = \{MV, DT_i, NB_j) \mid (AG(MV, i) \not\supseteq SG(Q_{MV}, i) \text{ or } AG(MV, i) \not\supseteq AG(Q, i)), AG(MV, i) = L_0^i, (AG(MV, j) \not\supseteq SG(QMV, j) \text{ or } AG(MV, j) \not\supseteq AG(Q, j)), AG(MV, j) \neq L_0^j\}$$

where the nested subquery block NB_j has the form

(SELECT	DISTINCT AG(MV, j)∪AG(Q, j)
FROM	DT_j
WHERE	p(QR(MV), DT_j)) NB_j

where $p(QR(MV), DT_j)$ denotes the selection predicate on the attributes in DT_j which is subsumed by the selection predicate for $QR(MV)$ and excludes sub-predicates subsumed by the predicate for $R(MV)$.

$$\text{JC} = \left(\bigwedge_{DT_i \in R} (MV.AG(MV, i) = DT_i.AG(MV, i)) \right) \wedge \left(\bigwedge_{NB_j \in R} (MV.AG(MV, j) = NB_j.AG(MV, j)) \right)$$

$$\text{SC} = \left(\bigwedge_{AG(MV,\, i) \supseteq SG(Q_{MV},\, i)} p(QR(MV), DT_i) \right) \wedge \left(\bigwedge_{DT_j \in R} p(QR(MV), DT_j) \right)$$

$$\mathrm{G} = \begin{cases} AG(Q), & \text{if } AG(Q) > AG(MV) \\ \phi, & \text{if } AG(Q) = AG(MV) \end{cases}$$

$$\mathrm{HC} = \begin{cases} HAV'(Q), & \text{if } HAV(Q) \neq null \text{ and } SG(MV) \geq AG(Q) \\ \phi, & \text{otherwise} \end{cases}$$

where $HAV'(Q)$ is a logical formula on P and AGG derived from $HAV(Q)$ by replacing $agg_i(m)$ in $AGG(Q)$ with the related $alias_i$ in AGG. The GROUP BY and HAVING clauses are included only when G and HC are not ϕ, respectively.

4.2.3. Step 3: integrating the query blocks

If only one MV is selected in Step 1, the query block generated in Step 2 becomes the final result of rewriting. Otherwise, we obtain a rewritten query by integrating the generated query blocks. The MVs selected in Step 1 can be classified into two categories by a relationship between the SG of an MV and the AG of the given query Q. Fig. 10 shows the possible selection granularities of these two kinds of MVs.

- $S_1 = \{MV_i \mid SG(MV_i) \geq AG(Q)\}$: the MVs in this class can be used to evaluate aggregation for all or a part of the aggregate groups of Q.
- $S_2 = \{MV_j \mid SG(MV_j) < AG(Q) \text{ or } SG(MV_j) <> AG(Q)\}$: each MV in this class may not compute aggregation for an aggregate group of Q, but all MVs in this class can compute it together.

Query blocks for the MVs in S_1 are integrated into a UNION multi-block query using UNION operators. On the other hand, query blocks for the MVs in S_2 are connected by UNION ALL operators and then integrated into a UNION ALL-GROUP BY query. It has a set $AGG'(Q)$ of the aggregate functions from $AGG(Q)$ except for COUNTs which are replaced by SUMs, a GROUP BY clause containing $AG(Q)$, and a HAVING clause with the condition on $AG(Q)$ and $AGG'(Q)$ derived from $HAV(Q)$ which is not *null*. Combining these two query blocks

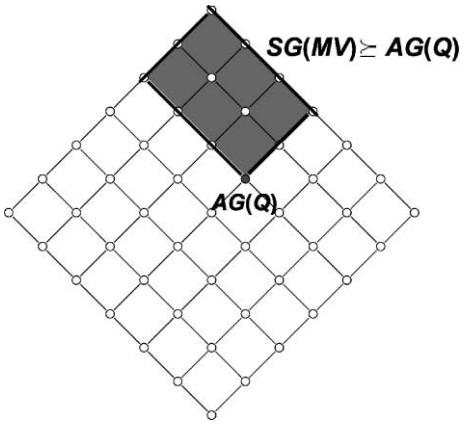

Fig. 10. The area of the SGs of the MVs that can be integrated by UNIONs.

using a UNION operator generates the final rewritten query. Given $S_1=\{MV_{11}, MV_{12},\ldots, MV_{1m}\}$ and $S_2=\{MV_{21}, MV_{22},\ldots, MV_{2n}\}$, the rewritten query has the form

($QB(MV_{11})$ UNION $QB(MV_{12})$ UNION...UNION $QB(MV_{1m})$)	
UNION	
(SELECT	$AG(Q)$, AGG' (Q)
FROM	($QB(MV_{21})$ UNION ALL $QB(MV_{22})$ UNION ALL...UNION ALL $QB(MV_{2n})$)
GROUP BY	$AG(Q)$
HAVING	HAV' (Q))

Example 4. We describe rewriting steps for the query Q_1 in Example 1.
Step 1: By Definition 3, $V(Q_1)=\{MV_1, MV_2, MV_3, \text{Sales}\}$. We select MV_1, MV_2, and MV_3 in the order using the algorithm proposed in Section 5. Their query regions are determined as follows (see Fig. 4).

$$QR(MV_1) = \{([\text{'USA','USA'}], [1997, 1999], (-\infty, +\infty), (-\infty, +\infty)), ([\text{'Canada', 'Canada'}], [1997, 1999], (-\infty, +\infty), (-\infty, +\infty))\}$$

$$QR(MV_2) = \{([\text{'USA', 'USA'}], [1996, 1996], (-\infty, +\infty), (-\infty, +\infty))\}$$

$$QR(MV_3) = \{([\text{'Canada', 'Canada'}], [1996, 1996], (-\infty, +\infty), (-\infty, +\infty))\}$$

Step 2: We will show how the query block for MV_1, $QB(MV_1)$, is generated. The normal from query for $QR(MV_1)$ over the fact table is

$$Q_{MV1}((\text{nation, year, none, none}), \{([\text{'USA', 'USA'}], [1997, 1999], (-\infty, +\infty), (-\infty, +\infty)), ([\text{'Canada', 'Canada'}], [1997, 1999], (-\infty, +\infty), (-\infty, +\infty))\}, (\text{state, year, none, none}), \{\text{SUM}(\text{sales}_{\text{dollar}})\}).$$

The components in $QB(MV_1)$ are determined by Definition 4 as follows. P={state, year} from $AG(Q_1)$ in Example 3. AGG={sum_dollar$_1$} because $AG(MV_1)=AG(Q_1)$ and SUM (sales_dollar) in $AGG(Q_1)$ is named sum_dollar$_1$ in MV_1. Since $AG(MV_1, 1)=\{\text{state}\} \not\supseteq \{\text{nation}\}=SG(Q_{MV1}, 1)$ and $AG(MV_1, 1)=\{\text{state}\} \not\supseteq \{\text{stor-}\{\text{store\_id}\}=L_0^1$, a join of MV_1 with a nested subquery block NB_1 over Store is needed. With $AG(MV_1, 1)=AG(Q_1, 1)=\{\text{state}\}$ and $p(QR(MV_1), \text{Store})=(\text{nation}=\text{'USA' OR nation}=\text{'Canada'})$, NB_1 has the form (SELECT DISTINCT state FROM Store WHERE nation = 'USA' OR nation = 'Canada') NB_1 However, since $AG(MV_1, 2)=SG(Q_{MV1}, 2)=AG(Q_1, 2)=\{\text{year}\}$, join of MV_1 with Time is not necessary. Thus we have R=$\{MV_1, NB_1\}$ and JC=(MV_1.state = NB_1.state). We have SC = $p(QR(MV_1), \text{Time})=(\text{year} \le 1999)$ since the predicate for $QR(MV_1)$ is (year $\ge$ 1997 AND year $\le$ 1999) but the sub-predicate (year $\ge$ 1997) is already included in the predicate for $R(MV_1)$. Since $AG(MV_1)=AG(Q_1)$ and $HAV(Q_1)=null$, GROUP BY and HAVING clauses are not required. Therefore, the query block $QB(MV_1)$ is written as follows.

SELECT	state, year, sum_dollar$_1$
FROM	MV_1, (SELECT DISTINCT state FROM Store WHERE nation = 'USA' OR nation = 'Canada') NB_1
WHERE	MV_1.state = NB_1 state AND year $\le$ 1999

The query blocks for MV_2 and MV_3 can be generated in a similar manner (see Q_1' in Fig. 3).
Step 3: MV_1, MV_2, and MV_3 satisfy $SG(MV_1)>AG(Q_1)$, $SG(MV_2)>AG(Q_1)$, and $SG(MV_3)>AG(Q_1)$, respectively, as shown in Fig. 8a. Thus, the query blocks for MV_1, MV_2, and MV_3 can be integrated into a UNION multi-block query, i.e., Q_1' in Fig. 3.

Example 5. In rewriting the query Q_2 in Example 2 by the proposed method, MV_1 and MV_3 are selected in the first step. Since $SG(MV_1)<>AG(Q_2)$ and $SG(MV_3)<>AG(Q_2)$ hold as shown in Fig. 8b, the query blocks for MV_1 and MV_3 can be integrated into a UNION ALL-GROUP BY query, i.e., Q_2' in Fig. 5.

5. Finding an efficient rewriting

In general, there exist many equivalent rewritings containing a different set of candidate MVs, which have different execution cost. Hence, it is desirable for system performance to choose such MVs and query regions as result in a rewritten query having an efficient execution plan. In this section, we explore the problem of selecting MVs and query regions for query rewriting and propose a heuristic algorithm that delivers an efficient solution within an acceptable time.

5.1. Problem definition

The problem of selecting a set of MVs and query regions to be used in rewriting a query is formalized as an optimization problem in Definition 5.

Definition 5. (the optimal MV set problem) Given a query $Q=(AG, SG, R, AGG, HAV)$, a set $V(Q)$ of the candidate MVs for Q, and a cost function cost (MV, QR), which provides cost for a materialized view MV with a query region QR, the optimal MV set problem is to find an optimal set S of pairs (MV_i, QR_i), where $MV_i \in V(Q)$ and QR_i is a non-empty query region for MV_i, which minimize the total cost of S,

$$\sum_{(MV_i,\, QR_i)\in S} cost(MV_i, QR_i)$$

subject to

$$QR_i - {}^*(R(MV_i) \cap {}^*R(Q)) = \phi \text{ for all } (MV_i, QR_i)\in S,$$

$$QR_i \cap {}^*QR_j = \phi \text{ for all pairs of } QR_i \text{ and } QR_j \text{ in } S \text{ such that } i\neq j, \text{ and}$$

$$R(Q) - {}^* \bigcup_{(MV_i,QR_i)\in S} QR_i = \phi.$$

The following theorem shows the intractability of the optimal MV set problem.

Theorem 2. The optimal MV set problem is NP-hard.

Proof. For simplicity of description, we do not consider the query regions for the selected MVs, which does not increase the complexity of the problem. Then, the decision version of the optimal MV set problem is to determine if there is a set S of $MV_i\in V(Q)$ such that

$$\sum_{MV_i\in S} cost(MV_i)\leq c \text{ and}$$

$$\bigcup_{MV_i\in S} (R(MV_i) \cap {}^*R(Q)) = R(Q),$$

where c is a positive real number. We will show that the minimum set cover decision problem [8], which was known to be NP-complete, can be polynomially transformed to this problem. The minimum set cover decision problem is described as follows: Given a collection $F=\{S_1, S_2,\ldots,S_n\}$ of subsets of a finite set S and a positive integer k, is there a subset F' of F such that $|F'| \leq k$ and $\bigcup_{S'\in F'} S' = S$? Given an instance of the minimum set cover decision problem, we can construct an instance of our problem, such that $V(Q)=\{MV_1, MV_2,\ldots, MV_n\}$ where $MV_i=S_i(1 \leq i \leq n)$, $R(Q) = \bigcup_{MV_i\in V(Q)} R(MV_i)$, $cost(MV_i) = 1(1\leq i\leq n)$, and $c = k$. Then, straightforwardly, the minimum set cover decision problem has a solution F' if and only if the optimal MV set decision problem has a solution S. □

5.2. Cost models

We assume that the execution cost of a rewritten query obtained by our method is the sum of the execution costs of the query blocks in it. For cost-

MV Selection Algorithm

Input: a query $Q = (SG, R, AG, AGG, HAV)$ and a set $V(Q)$ of the candidate MVs for Q

Output: a set S of pairs (MV_i, QR_i)

begin

$S := \phi$; $OPEN := V(Q)$; $QR := R(Q)$;

while $QR \neq \phi$ **do**

Compute the profit of MV_i for QR, $profit(MV_i, QR)$, for each $MV_i \in OPEN$;

Select a materialized view MV_i such that $profit(MV_i, QR)$ is maximal;

if $profit(MV_i, QR) = 0$ **then**

$S := S \cup \{(FT, QR)\}$;

return S;

end if;

$OPEN := OPEN - \{MV_i\}$;

$S := S \cup \{(MV_i, R(MV_i) \cap^* QR)\}$;

$QR := QR -^* R(MV_i)$;

end while;

return S;

end

Fig. 11. Greedy algorithm.

based selection of MVs and query regions, we have to estimate the execution cost of a query block defined over a candidate MV and a query region for the MV. We propose different cost models for an MV and the fact table considering different access methods for them. We extend the linear cost model proposed in Ref. [13] to use the number of disk pages to be retrieved from an MV or the fact table as a measure of the execution cost of a query block over it.

We assume that there is no special index or clustering on the attributes in MVs. That implies we have to retrieve all pages in an MV to evaluate a query block for it, regardless of the query region applied to the MV. Therefore, the execution cost of a query block for a materialized view MV is defined as

$$cost(MV, QR) = N_{MV}$$

where N_{MV} denotes the number of pages in MV.

On the other hand, several types of access methods have been proposed in the literature to support fast access to the fact table in data warehouses [7,20,25]. In particular, bitmap join indices [19] on the dimensional attributes are frequently used for efficient joins between a dimension table and the fact table.

Considering such index structures, we assume the execution cost of a query block over the fact table to be the number of tuples contained in a given query region over the fact table as long as it does not exceed the total number of pages in the fact table. That is, the execution cost is defined as

$$cost(FT, QR) = \min\{k, N_{FT}\}$$

where k is the number of tuples in the fact table selected by the query region QR and N_{FT} is the number of pages in the fact table. Supporting that the values of dimensional attributes are uniformly distributed over the d-dimensional domain space, k can be estimated as

$$k = n_{FT} \cdot \frac{area(QR)}{area(R(FT))}$$

where n_{FT} is the number of tuples in the fact table and $area(R)$ means the total number of different values of

the set of dimensional attributes (i.e., foreign keys) in the fact table contained in the selection region R. In this cost model, we ignore the cost of searching dimension tables or bitmap indices since it is relatively low compared with that of accessing MVs and the fact table.

5.3. A heuristic algorithm

In this section, we introduce a greedy heuristic algorithm to find an effective solution for selecting a set of MVs and query regions quickly. The algorithm, shown in Fig. 11, works in stages. At each stage, it selects an MV from the set of candidate MVs for a given query Q that gives the maximum profit in execution cost for the remaining query region QR', which is defined as

$$QR' = R(Q) - {}^* \bigcup_{MV_i \in S} R(MV_i) = \phi$$

where S is a set of the MVs already selected. Using the cost model in Section 5.2, we employ a profit measure for MVs in $V(Q)$ defined as

$$\begin{aligned} &profit(MV, QR') \\ &\quad = cost(FT, QR') - (cost(FT, QR' - {}^*QR(MV)) \\ &\qquad + cost(MV, QR(MV))) \\ &\quad \cong cost(FT, QR(MV)) - cost(MV, QR(MV)) \\ &\quad = \begin{cases} \min\{n_{FT} \cdot \frac{area(QR(MV))}{area(R(FT))}, N_{FT}\} - N_{MV}, & \text{if } MV \neq FT \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

The query region for an MV is computed by

$$QR(MV) = R(MV) \cap {}^*QR'.$$

At each selection of an MV, query regions and profits are recomputed for all the remaining MVs in $V(Q)$ and then the MV with the largest profit value is opted. The algorithm terminates when the maximum profit is 0, i.e., the fact table has the maximum profit among the remaining MVs, or the remaining query region becomes empty. Time complexity of the algorithm is $O(n^2)$ where n is the cardinality of $V(Q)$.

6. Experimental evaluation

In this section, we evaluate our rewriting method adopting the MV selection algorithm proposed in Section 5 by some experiments. We performed two kinds of evaluations to show the effectiveness and performance of the algorithm. We measured the quality of the solution produced by the algorithm and compared it with the optimal solution, i.e., the set of MVs and their query regions constituting the most efficient query plan. To find the optimal solution, we developed an A^* algorithm which avoids an exhaustive search of the state–space graph by pruning a large part of the search space that do not contain the optimal solution [18]. We also measured and compared the execution time taken by the algorithms.

We implemented the proposed algorithm and conducted experiments on a SUN UltraSPARC-II 450 MHz processor with 256 MB main memory running Solaris 2.7. We made an assumption that the fact table has 1 billion tuples and the values of their foreign key attributes referring to dimension tables are uniformly distributed. We supposed that there are four dimension tables in the DW and five dimension levels in each dimension hierarchy. The fan-outs along the dimension hierarchies were set to 10 in all dimensions. We assumed the size of a tuple in the fact table and MVs to be 128 bytes and used a page size of 8 KB.

We generated 1000 materialized views without any knowledge of query workloads. The SGs and AGs of the MVs were uniformly distributed over all granularities in the DH lattice and $SG \geq AG$ held for all MVs. The selection region of an MV was arbitrarily determined on the SG of the MV under the constraints that 50% of the intervals of dimension levels should be points and its area be larger than 10% of the area of the fact table.

Then, 500 test OLAP queries over the fact table were generated and rewritten by the proposed method. The AGs and SGs of the queries were also uniformly selected like those of MVs. Moreover, to simulate a workload of typical OLAP, we made query distribution such that 30% of the queries are drill-down queries, another 30% are roll-up queries, and remaining 40% are random queries. The drill-down or roll-up queries were restricted to go down or up at most one dimension level along each dimension hierarchy. The selective regions of the queries were generated like those of MVs except that they have no constraint on their area. We assumed that all MVs and queries have only SUM as their aggregate function and all queries have no HAVING condition.

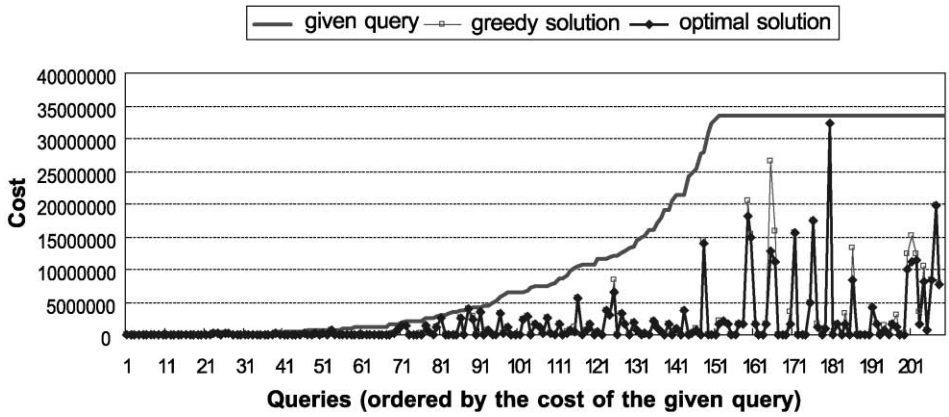

Fig. 12. The estimated execution costs of the given queries and their rewritten queries.

6.1. Quality of the solution

We ran the greedy algorithm as well as the *A** algorithm on each test query to find a greedy solution and an optimal solution for selecting MVs and query regions. Using the cost model proposed in Section 5.2, we estimated the execution costs of the rewritten queries from two solutions. We also computed the cost of evaluating the given query over the fact table. Then we compared the three costs with one another.

Fig. 12 shows the estimated costs of the rewritten queries generated with the solutions of the greedy and *A** algorithm. Only the results for 208 queries for whom there exist a rewritten query having the execution cost lower than that of the original one are presented. The queries are arranged on the *x*-axis in non-decreasing order of their costs over the fact table.

For each test query, we also compared the ratio of the cost of a rewritten query over the cost of the test query, as depicted in Fig. 13. The queries are arranged in non-decreasing order of the cost ratios for their optimal solutions. The result indicates that the average cost of the rewritten queries using the optimal solutions is 16.57%, and the average cost of the rewritten queries using the greedy solutions is 17.26%. Note that if we create MVs using static view selection techniques such as [2,10,12,13,15,24] or adopt dynamic view management schemes [1,14] reflecting the changing query workloads in data warehouses, the expected cost of the rewritten queries will be more reduced.

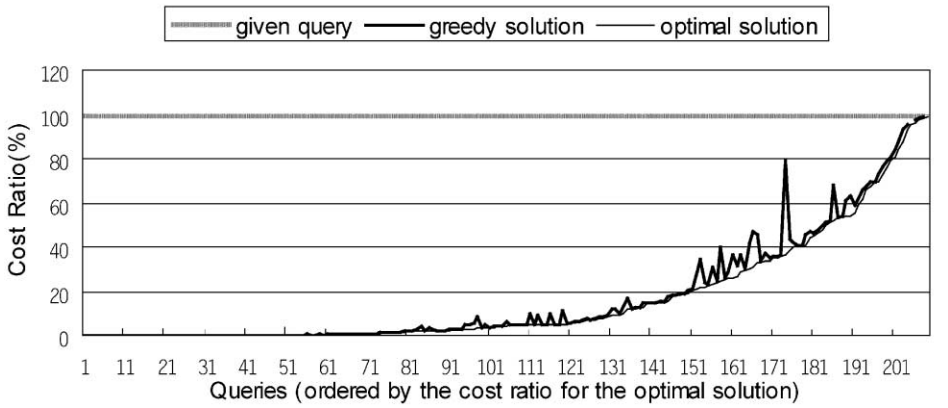

Fig. 13. The ratios of the cost of the optimal and greedy solution to the cost of the given query.

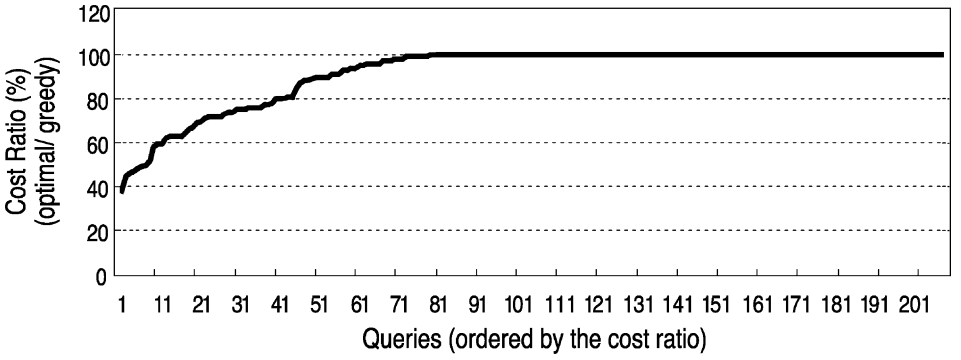

Fig. 14. The ratios of the cost of the optimal solution to the cost of the greedy solution.

Fig. 14 presents the quality of the solution returned by the proposed greedy algorithm, which is measured by a ratio of the cost of the optimal solution to the cost of the greedy solution. We observe that most of the solutions obtained by the proposed algorithm are close to optimal. It yielded optimal solutions for 137 queries, which are about 66% of the test queries for which an efficient rewriting exists. The sub-optimal solution had the quality of around 80% of that of the optimal solution on the average.

6.2. Execution time

Fig. 15 shows the execution time taken by the proposed greedy algorithm to find a solution for each test query, which is compared with that of the *A** algorithm. The results are arranged in non-decreasing order of the execution time of the *A** algorithm. In this experiment, the greedy algorithm took about 1/160 of the execution time of the *A** algorithm on the average. This ratio is expected to be much more reduced as the number of MVs to be searched by the *A** algorithm is increased. We also observe that the greedy algorithm scales up to large number of candidate MVs well in contrast to the *A** algorithm whose execution time increases exponentially.

7. Related work

Several works on answering conjunctive queries using materialized views has been proposed in the literature [3,5,6,11,16,22,23,26]. Levy et al. [16] formalized the problem of finding rewritings of a conjunctive query under set semantics in terms of containment mappings from views to the query. Chaudhuri et al. [5] addressed the problem of optimizing queries using MVs. They proposed a rewriting method for conjunctive SPJ queries and

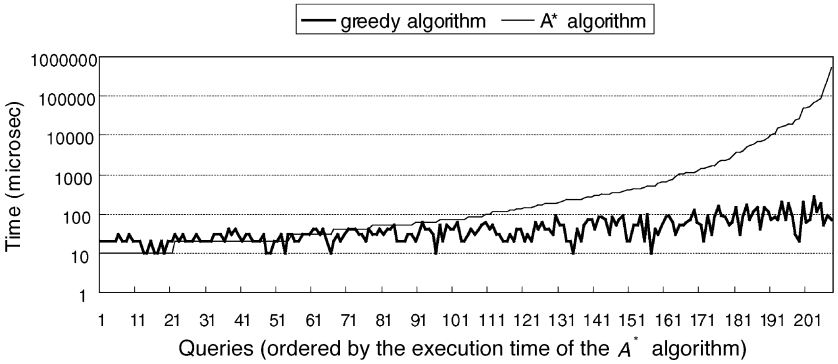

Fig. 15. Execution time of the greedy algorithm and the *A** algorithm.

integrated it into a cost-based query optimization algorithm. However, they did not consider either aggregate queries or aggregate views and their algorithm can generate only single-block queries.

Recently, the problem of rewriting aggregate queries has received much attention. Gupta et al. [11] proposed an algorithm for answering aggregate queries using materialized views in DW environments. Their algorithm performs syntactic transformations on the query tree of a given query using a set of proposed transformation rules. In their algorithm, however, an MV is usable only when a part of the definition of the query can be exactly transformed to the definition of the MV. Hence, the class of usable MVs and the types of rewritings generated by the algorithm are restrictive.

Srivastava et al. [23] proposed several algorithms to rewrite aggregate queries using conjunctive or aggregate MVs. They presented sufficient conditions for an MV to be used in rewriting a query using their methods, which include the following constraints. First, all tables included in the definition of the MV must also appear in the definition of the query. Second, if an attribute in the SELECT or GROUP BY clause of the query is from a table included in the definition of the MV, the SELECT clause of the MV must contain the attribute. In Example 2, However, Time that is contained in all MVs does not appear in the definition of Q_2, and in Example 1, state in the SELECT clause of Q_1 does not contained in the SELECT clause of MV_3. Thus, their algorithms cannot perform the rewritings such as Q_1' and Q_2'. As to the syntactic forms of rewritten queries, Ref. [23] includes the UNION ALL of single-block queries, but it is a kind of the UNION rewriting in our method since each single-block query computes a separate part of the aggregate groups of the given query. Without a detail description of the rewriting algorithm, Albrecht et al. [1] also presented the same UNION rewriting example in the form of a UNION ALL multi-block query in their scheme for dynamic management of multidimensional query results. The UNION ALL-GROUP BY rewriting proposed in this paper cannot be performed by either of them.

In Refs. [3,26], new methods that exploit various MVs to evaluate complex queries in DWs have been suggested. While most previous algorithms demand that there should be a one-to-one mapping or a containment mapping from an MV to the given query, the algorithm proposed in Chang and Lee [3] can utilize the MVs which include relations not referred to in the query. The usability of MVs is determined based on the functional dependencies between grouping attributes in MVs and the query. However, they did not consider selection predicates of the query and MVs, and the algorithm can generate only a single-block aggregate query. Zaharioudakis et al. [26] addressed the rewriting of queries including complex expressions, supergroup aggregation, and nested subqueries. They represent queries and MVs as graphs and suggested matching conditions and compensation rules for equivalence between two subgraphs in a query and an MV for several simple matching patterns. Their algorithm scans the query and MV graphs in a bottom-up fashion, identifying potential pairs of matching subgraphs, performing compensation for the matching patterns, and generating rewritten subqueries including the MV. However, they did not consider the types of rewritings proposed in this paper which use UNION and UNION ALL operators.

Pottinger and Levy [22] proposed an algorithm for rewriting a conjunctive query using conjunctive views in the context of data integration. Our method is different from the work in that it finds the equivalent rewriting of a given query which is optimal or efficient in execution cost while the algorithm in Ref. [22] generates the maximally contained rewriting which obtains as many answers as possible from existing MVs.

In summary, the previous query rewriting approaches have some restrictions. They did not effectively exploit semantic information such as dimension hierarchies in DWs and the characteristics of OLAP queries. Hence, they can perform relatively simple types of rewritings using a limited class of MVs compared with our proposed method. In addition, the problem of finding the cost-optimal rewritten query among many candidate ones has not been investigated enough in the previous work.

8. Conclusions and future work

In this paper, we proposed an approach to rewrite a given OLAP query using MVs existing in data

warehouses. We defined the normal form of typical OLAP queries and MVs based on the lattice structure derived from dimension hierarchies in DWs. We presented conditions for usability of MVs in rewriting OLAP queries and then proposed a rewriting method that can rewrite the normal form OLAP queries using various kinds of MVs together. The algorithm consists of three main steps. In the first step, it selects MVs that will be used in rewriting and determines query regions for them. In the second step, it generates query blocks for the selected MVs using their query regions. The last step integrates the query blocks into a final rewritten query. We use two ways of integration, viz., the UNION integration and the UNION ALL-GROUP BY integration, depending on a relationship between the SGs of MVs and the AG of the query. By exploiting the semantic information available in DWs and the characteristics of OLAP queries, the proposed algorithm utilizes a much broader class of MVs and yields more general types of rewritings than other previous approaches can do. Furthermore, we investigated the problem of finding an optimal set of MVs and their query regions that results in a rewritten query which can be executed efficiently. We presented a heuristic algorithm and showed by experiments that it provides an effective solution in an acceptable time even for a large number of materialized views.

Acknowledgements

This work was supported in part by BK21 Educational–Industrial collaboration fund and the Ministry of Information and Communication of Korea (“Support Project of University Foundation Research < 2000>” supervised by IITA).

References

[1] J. Albrecht, A. Bauer, O. Deyerling, H. Gunze, W. Hummer, W. Lehner, L. Schlesinger, Management of Multidimensional Aggregates for Efficient Online Analytical Processing, Proceedings of International Database Engineering and Applications Symposium, 1999, pp. 156–164.

[2] E. Baralis, S. Paraboschi, E. Teniente, Materialized View Selection in a Multidimensional Database, Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, pp. 318–329.

[3] J. Chang, S. Lee, Query Reformulation Using Materialized Views in Data Warehouse Environment, Proceedings of the First ACM International Workshop on Data Warehousing and OLAP, 1998, pp. 54–59.

[4] S. Chaudhuri, U. Dayal, An overview of data warehousing and OLAP technology, SIGMOD Record 26 (1) (1997) 65–74.

[5] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, Optimizing Queries with Materialized Views, Proceedings of 11th IEEE International Conference on the Data Engineering, 1995, pp. 190–200.

[6] C.M. Chen, N. Roussopoulos, The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching, Proceedings of the 4th International Conference on Extending Database Technology, 1994, pp. 323–336.

[7] M. Ester, J. Kohlhammer, H.-P. Kriegel, The DC-Tree: A Fully Dynamic Index Structure for Data Warehouses, Proceedings of the 16th IEEE International Conference on Data Engineering, 2000, pp. 379–388.

[8] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W.H. Freeman, San Francisco, CA, 1979.

[9] J. Gray, A. Bosworth, A. Laymen, H. Pirahesh, Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals, Proceedings of the 12th IEEE International Conference on Data Engineering, 1996, pp. 152–159.

[10] H. Gupta, Selection of Views to Materialize in a Data Warehouse, Proceedings of the International Conference on Database Theory, 1997, pp. 98–112.

[11] A. Gupta, V. Harinarayan, D. Quass, Aggregate-Query Processing in Data Warehousing Environments, Proceedings of the 21st International Conference on Very Large Data Bases, 1995, pp. 358–369.

[12] H. Gupta, I.S. Mumick, Selection of Views to Materialize Under a Maintenance Cost Constraint, Proceedings of the International Conference on Database Theory, 1999, pp. 453–470.

[13] V. Harinarayan, A. Rajaraman, J.D. Ullman, Implementing Data Cube Efficiently, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996, pp. 205–216.

[14] Y. Kortidis, N. Roussopoulos, DynaMat: A Dynamic View Management System for Data Warehouses, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999, pp. 371–382.

[15] W.J. Labio, D. Quass, B. Adelberg, Physical Database Design for Data Warehouses, Proceedings of the 13th IEEE International Conference on Data Engineering, 1997, pp. 277–288.

[16] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava, Answering Queries Using Views, Proceedings of the ACM Symposium on Principles of Database Systems, 1995, pp. 95–104.

[17] A. Margalit, G.D. Knott, An algorithm for computing the union, intersection or difference of two polygons, Computers and Graphics 13 (2) (1989) 167–183.

[18] N.J. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[19] P. O’Neil, G. Graefe, Multi-table joins through bitmapped join indices, SIGMOD Record 24 (3) (1995) 8–11.

[20] P. O'Neil, D. Quass, Improved Query Performance with Variant indexes, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1997, pp. 38–49.
[21] C.-S. Park, M.H. Kim, Y.-J. Lee, Rewriting OLAP Queries Using Materialized Views and Dimension Hierarchies in Data Warehouses, Proceedings of the 17th IEEE International Conference on Data Engineering, 2001, pp. 515–523.
[22] R. Pottinger, A. Levy, A Scalable Algorithm for Answering Queries Using Views, Proceedings of the 26th International Conference on Very Large Data Bases, 2000, pp. 484–495.
[23] D. Srivastava, S. Dar, H.V. Jagadish, A.Y. Levy, Answering Queries with Aggregation Using Views, Proceedings of the 22nd International Conference on Very Large Data Bases, 1996, pp. 318–329.
[24] D. Theodoratos, T. Sellis, Data Warehouse Configuration, Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, pp. 318–329.
[25] M.-C. Wu, A.P. Buchmann, Encoded Bitmap Indexing for Data Warehouses, Proceedings of the 14th IEEE International Conference on Data Engineering, 1998, pp. 220–230.
[26] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, M. Urata, Answering Complex SQL Queries Using Automatic Summary Tables, Proceedings of 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 105–116.

Chang-Sup Park received his BS and MS degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Teajon, Korea, in 1995 and 1997, respectively. He is currently a PhD student in the Department of Computer Science at KAIST. His research interests include OLAP, data warehouses, multi-dimensional indexing, and semi-structured data management. He is a student member of the ACM.

Myoung Ho Kim received his BS and MS degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1982 and 1984, respectively, and his PhD 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 professor. His research interests include OLAP, data mining, information retrieval, mobile computing and distributed processing. He is a member of the ACM and the IEEE Computer Society.

Yoon-Joon Lee received his BS degree in Computer Science and Statistics from Seoul National University, Seoul, Korea, in 1977, his MS degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, in 1979, and his PhD degree in Computer Science from INPG-ENSIMAG, France, in 1983. In 1984, he joined the faculty of the Department of Computer Science at KAIST, Taejon, Korea, where currently he is a professor. His research interests include data warehouses, OLAP, WWW, multimedia information systems, and database systems. He is a member of the ACM and the IEEE Computer Society.