Rewriting OLAP Queries Using Materialized Views and Dimension Hierarchies 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, Taejon, Korea
{parkcs,mhkim,yjlee}@dbserver.kaist.ac.kr

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 for rewriting a given OLAP query using various kinds of materialized aggregate views which already exist in data warehouses. We first define the normal forms of OLAP queries and materialized views based on the lattice of dimension hierarchies, the semantic information in data warehouses. Conditions for usability of a materialized view 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 effectively utilizes existing materialized views. The proposed algorithm can make use of materialized views having different selection granularities, selection regions, and aggregation granularities together to generate an efficient rewritten query.

1. Introduction

Recently, decision support systems based on Data Warehouses (DWs) are widely used and researches on On-Line Analytical Processing (OLAP) have been actively made. DWs over which OLAP queries are evaluated tend 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 severely 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 pre-compute the results of frequently issued OLAP queries, store them in summary tables, i.e., Materialized Views (MVs), and evaluate incoming queries using the MVs. This approach requires techniques to select views to materialize and rewrite given queries using MVs. There have been several query rewriting strategies proposed in the literature [11, 7, 3, 5, 10, 1, 12]. Most of them, however, have not much considered the characteristics of DWs and OLAP queries and hence cannot utilize the existing MVs sufficiently.

In this paper, we propose a new algorithm for rewriting OLAP queries using MVs which improves the usability of MVs significantly in contrast to the earlier approaches. We define a normal form of typical OLAP queries based on the lattice of dimension hierarchies and 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.

The main contributions of the paper are as follows. First, 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. Second, we exploit meta-information of DW schema effectively. Dimension hierarchies in dimension tables are implicitly used in constructing OLAP queries and MVs. By using such semantic information in DWs, query rewriting that utilizes various kinds of MVs together can be achieved. Third, for a given OLAP query, there can be many equivalent rewritings using different MVs in various ways, whose execution costs are different from one another. We propose a greedy heuristic to select good MVs and their query regions for generating an efficient rewritten query.

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. Related work is discussed in Section 5, and we draw conclusions in Section 6.
2. Motivating Examples

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

**Example 1** Consider a DW with sales data of a large chain of department stores, whose schema is shown in Figure 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, namely, $MV_1$ which summarizes the total sales of stores from 1997 by state and year, $MV_2$ which has the total sales of the stores in the USA by state and month, and $MV_3$ which contains the total sales of the stores in Canada to 1996 by city and month.

$MV_1$: SELECT state, year, SUM(sales_dollar) AS sum_dollar,
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

$MV_2$: SELECT state, year, month, SUM(sales_dollar) AS sum_dollar,
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

$MV_3$: SELECT city, year, month, SUM(sales_dollar) AS sum_dollar,
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

We consider the following OLAP query $Q_1$, which asks for the total sales of the stores in the USA or Canada from 1996 to 1999 by state and year.

$Q_1$: 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

$Q_1$ can be rewritten to the following query $Q'_1$, which uses the three materialized views instead of the fact table Sales.

$Q'_1$: (SELECT state, year, sum_dollar_1
FROM $MV_1$, (SELECT DISTINCT state
FROM Store
WHERE nation = 'USA' OR nation = 'Canada') S
UNION
(SELECT state, year, SUM(sum_dollar_2)
FROM $MV_2$
WHERE year = 1996
GROUP BY state, year)
UNION
(SELECT S.state, year, SUM(sum_dollar_3)
FROM $MV_3$, (SELECT DISTINCT city, state
FROM Store) S
WHERE $MV_3$.city = S.city AND year = 1996
GROUP BY S.state, year)

$Q'_1$ has three query blocks, whose results are combined by union. Each query block contains a different MV and computes a part of the aggregate groups of $Q_1$ as shown in Figure 2. 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 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 following query $Q_2$ over the sales DW in Example 1.

$Q_2$: SELECT state, SUM(sales_dollar)
FROM Sales, Store
WHERE Sales.store_id = Store.store_id
AND Store.nation = 'Canada'
GROUP BY state

$Q_2$ requests the sales of department stores in Canada, which are not materialized in the DW of Example 1. However, we can rewrite $Q_2$ using $MV_3$ that computes the sales of department stores in Canada to 1996 by city and month. A possible rewrite of $Q_2$ is the following query $Q'_2$.

$Q'_2$: (SELECT S.state, SUM(S.sum_dollar)
FROM Store S
WHERE S.nation = 'Canada'
GROUP BY S.state)

$Q'_2$ has three query blocks, whose results are combined by union. Each query block contains a different MV. The first one computes the sales of department stores in Canada to 1996 by city as shown in Figure 2. Specifically, the first query block computes from $MV_3$ the total sales of department stores in Canada from 1997 to 1999 by state and month. The second and the third one use $MV_2$ and $MV_3$ respectively to compute the total sales of department 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 $Q_2$, we can obtain the same result of $Q_2$ by taking the union of them as in $Q'_2$. □

The next example shows another type of query rewriting.
$Q_2$ can be rewritten to the equivalent query $Q'_2$, which uses $MV_1$ and $MV_3$ instead of the fact table $Sales$.

$Q'_2$: SELECT state, SUM(psum)
FROM (SELECT state, SUM(sum_dollar_1) AS psum
FROM $MV_1$,
(SELECT DISTINCT state
FROM Store
WHERE nation = 'Canada') $S_1$
WHERE $MV_1$.state = $S_1$.state
GROUP BY state
UNION ALL
SELECT state, SUM(sum_dollar_3) AS psum
FROM $MV_3$,
(SELECT DISTINCT city, state
FROM Store) $S_2$
WHERE $MV_3$.city = $S_2$.city
GROUP BY state

GROUP BY state

$Q_2$ 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 Figure 3, 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 the 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. We will revisit these examples in Section 4 to illustrate how our algorithm can achieve the rewritings.

3. OLAP Queries and Materialized Views

3.1. The Lattice of Dimension Hierarchies

We consider DWs that have a star schema [2] consisting of one fact table and $d$ dimension tables like the one in Figure 1. The fact table has foreign keys to dimension tables 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.

A dimension hierarchy $DH_i$ of a dimension table $DT_i$, defined as $DH_i = (L^i_0, L^i_1, \ldots, L^i_h, \text{none})$, is an ordered set of dimension levels. A dimension level $L^i_j$ is a set of attributes from $DT_i$. We call $j$ of $L^i_j$ its height. $L^i_0$ is the lowest level, which equals the primary key of $DT_i$. none denotes the highest level and is an empty set. There is a functional dependency $L^i_{j+1} \rightarrow L^i_j$ between $L^i_{j+1}$ and $L^i_j$ ($1 \leq j \leq h$). 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, $DH = DH_1 \times DH_2 \times \cdots \times DH_d$, is a class of the ordered sets of dimension levels from all different dimension hierarchies. There exists the partial ordering relation $\preceq$ among the elements in $DH$, defined as

$$(L^1_{i_1}, L^2_{i_2}, \ldots, L^d_{i_d}) \preceq (L^1_{m_1}, L^2_{m_2}, \ldots, L^d_{m_d})$$

if and only if $L^1_{i_1} \rightarrow L^1_{m_1}$ or $L^1_{i_1} = L^1_{m_1}$, for all $1 \leq i \leq d$.

Two additional relations can be derived from $\preceq$ as follows.

$$(L^1_{i_1}, L^2_{i_2}, \ldots, L^d_{i_d}) \prec (L^1_{m_1}, L^2_{m_2}, \ldots, L^d_{m_d})$$

if and only if $L^1_{i_1} \rightarrow L^1_{m_1}$, for all $1 \leq i \leq d$ but there exists $j (1 \leq j \leq d)$ such that $L^j_{i_j} \neq L^j_{m_j}$.

$$(L^1_{i_1}, L^2_{i_2}, \ldots, L^d_{i_d}) \ll (L^1_{m_1}, L^2_{m_2}, \ldots, L^d_{m_d})$$

if and only if neither $(L^1_{i_1}, L^2_{i_2}, \ldots, L^d_{i_d}) \preceq (L^1_{m_1}, L^2_{m_2}, \ldots, L^d_{m_d})$ nor $(L^1_{i_1}, L^2_{i_2}, \ldots, L^d_{i_d}) \prec (L^1_{m_1}, L^2_{m_2}, \ldots, L^d_{m_d})$. 

Figure 2. The aggregate groups of three query blocks in $Q'_1$

Figure 3. The aggregate groups of two query blocks in $Q'_2$
3.2. The Normal Form of OLAP Queries

The OLAP queries we consider in this paper are un-nested single-block aggregate queries over the base tables. Queries over MVs can be transformed to those over the 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 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. 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 \( \alpha \text{ op } c \), where \( \alpha \) is a dimensional attribute, \( c \) is a constant, and \( \text{op} \in \{<,\leq,=,\geq,>\} \).

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 \( DH_i \).
- \( R = \{R_i\} \) is the selection region of \( Q \), which is a set of hyper-rectangles. A hyper-rectangle \( R_i = (l_i, U_i) \) 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 \( A_i \) (\( 1 \leq i \leq d \)) is a dimension level in the dimension hierarchy \( DH_i \).
- \( AGG = \{\text{agg}(m) | \text{agg} \in \{\text{MIN, MAX, SUM, COUNT}\} \text{ and } m \text{ is a measure attribute}\} \)\(^2\).
- \( HAV \) is a logical formula of comparison predicates on the attributes in \( AG \) and the aggregate functions in \( AGG \). It represents the HAVING condition of \( Q \) expressed in SQL. We denote an empty condition by \( \text{null} \).

\( SG(Q), R(Q), AG(Q), AGG(Q), \) 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 \( SG(Q,i) \) (\( 1 \leq i \leq d \)) denote the dimension levels from \( DH_i \) contained in \( AG(Q) \) and \( SG(Q) \), respectively.

A class of queries frequently posed in OLAP includes drill-downs, roll-ups, and slice&dices [2]. They can be expressed in the proposed normal forms and classified by relationships between their selection and aggregation granularities. For instance, if we let \( Q_n \) be a drill-down from \( Q_o \), it satisfies \( SG(Q_n) \supseteq AG(Q_n) \) and \( AG(Q_n) \supseteq AG(Q_o) \).

In this paper, we consider only MVs that store the results of the normal form OLAP queries satisfying \( SG \supseteq AG \) and \( HAV = \text{null} \) since 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.

---

1A similar lattice framework was proposed in [6], which represents a dependence relation among a set of views.

2Since AVG can be computed using SUM and COUNT, we do not consider it in this paper.
Example 3 The normal forms of the MVs and the query in Example 1 are as follows.
\[ MV_1(SG, R, AG, AGG) = MV_1((\text{null}, \text{year}, \text{none}, \text{none}), \{(-\infty, +\infty), [1997, +\infty), (-\infty, +\infty), (-\infty, +\infty)\}), \]
\[ (\text{state}, \text{year}, \text{none}, \text{none}), \{\text{SUM(sales, dollar)}\}) \]
\[ MV_2(SG, R, AG, AGG) = MV_2((\text{null}, \text{year}, \text{none}, \text{none}), \{(["USA", "USA"], (-\infty, +\infty), (-\infty, +\infty), (-\infty, +\infty)), \}
\[ (\text{state}, \text{year-month}, \text{none}, \text{none}), \{\text{SUM(sales, dollar)}\}) \]
\[ MV_3(SG, R, AG, AGG) = MV_3((\text{null}, \text{year}, \text{none}, \text{none}), \{(["Canada", "Canada"], [1996, 1999], (-\infty, +\infty), (-\infty, +\infty)), \}
\[ (\text{city}, \text{year-month}, \text{none}, \text{none}), \{\text{SUM(sales, dollar)}\}) \]
\[ Q_1(SG, R, AG, AGG, HAV) = \]
\[ Q_1((\text{null}, \text{year}, \text{none}, \text{none}), \{(["USA", "USA"], [1996, 1999], (-\infty, +\infty), (-\infty, +\infty)), \}
\[ (["Canada", "Canada"], [1996, 1999], (-\infty, +\infty), (-\infty, +\infty)), \}
\[ (\text{state}, \text{year}, \text{none}, \text{none}), \{\text{SUM(sales, dollar)}\}, \text{null} \)

Figure 5 shows the SGs and AGs of the MVs and queries in Example 1 and Example 2 on the DH lattice.

We define two set operations which are used in Section 4.

Definition 2 Let \( Q \) and \( MV \) be a query and a materialized view. The region intersection \( \cap^\star \) between the selection region of \( Q \) and that of \( MV \) is defined as
\[ R(Q) \cap^\star R(MV) = \{ R_i \cap R_j | R_i \in R(Q), R_j \in R(MV) \} \]
where \( R_i \cap R_j \) is the hyper-rectangle which is the intersection of \( R_i \) and \( R_j \). The region difference \( \neg^\star \) of the selection region of \( Q \) and that of \( MV \) is defined as
\[ R(Q) \neg^\star R(MV) = \{ R_k | R_k \notin (R_i \cap R_j), R_k \in R(Q), R_j \in R(MV) \} \]
The intersection and difference of the selection regions of two MVs can be defined in the same way.

The intersection and difference of two hyper-rectangles can be obtained by algorithms proposed in computational geometry (e.g., [8]). The granularity of \( R(Q) \cap^\star R(MV) \) and \( R(Q) \neg^\star R(MV) \) is the Greatest Lower Bound (GLB) of \( SG(Q) \) and \( SG(MV) \) in the DH lattice. 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. Rewriting OLAP Queries Using MVs

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 [10]. 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.
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 \( AG(MV) \subseteq AG(Q) \), the result of aggregation over the tuples in \( S \) by \( AG(Q) \) and an aggregate function \( agg(m) \) in \( AGG(Q) \) is equal to the result of aggregation over the tuples in \( S' \) by \( AG(G) \) and an aggregate function \( agg'(m') \), where \( agg' \) is equal to \( agg \) if \( agg \) is \( \text{MIN}, \text{MAX}, \text{or SUM} \) and \(agg' \) is \( \text{SUM} \) if \( agg \) is \( \text{COUNT} \), and \( m' \) is the result attribute for \( agg(m) \).

Using the above lemmas, we obtain the following theorem on the MVs in \( V(Q) \). The proof is shown in [9].

**Theorem 1** Given an OLAP query \( Q \), a materialized view \( MV \) in \( V(Q) \) is usable in rewriting \( Q \).

For notational convenience, we regard the fact table as a kind of MV included in \( V(Q) \). Figure 6 shows the area of the AGs of possible candidate MVs for \( Q \) on the DH lattice.

### 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.

#### 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 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 \( S \) be a set of MVs selected for rewriting \( Q \), the query regions for the MVs in \( S \) must satisfy the following conditions:

\[
QR(MV_i) \cap^* (R(MV_i) \cap R(Q)) = \phi \quad \text{for all } MV_i \in S, \\
QR(MV_i) \cap^* QR(MV_j) = \phi \quad \text{for all } MV_i, MV_j \in S \text{ such that } MV_i \neq MV_j, \\
R(Q) \cap^* \bigcup_{MV_i \in S} QR(MV_i) = \phi
\]

In general, there can be many equivalent rewritings containing a different set of candidate MVs, which differ greatly in execution performance. Hence it is desirable to choose such MVs and query regions for them as constitute a rewritten query that can be executed efficiently. Unfortunately, the problem of finding a cost-optimal set of MVs and query regions for rewriting a query is NP-hard [9]. To obtain an efficient solution within an acceptable time, we propose a greedy heuristic algorithm which works in stages. At each stage, it selects a materialized view from \( V(Q) \) that gives the maximum profit in execution cost for the remaining query region \( QR' \) which is defined as

\[
QR' = R(Q) \cap^* \bigcup_{MV_j \in S'} R(MV_j)
\]

where \( S' \) is a set of the MVs already selected. We employ a profit measure for MVs in \( V(Q) \) defined as

\[
\text{profit}(MV, QR') = \\
\text{cost}(FT, QR') - (\text{cost}(FT, QR' - QR(MV)) + \text{cost}(MV, QR(MV))) \\
\simeq \text{cost}(FT, QR(MV)) - \text{cost}(MV, QR(MV))
\]

where \( \text{cost}(MV, R) \) is the execution cost of the query over \( MV \) that selects tuples contained in \( R \). The query region for \( MV \) 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 picked. The algorithm terminates when the maximum profit becomes less than or equal to 0, or all the MVs in \( V(Q) \) have been selected, or the remaining query region becomes empty. Time complexity of the algorithm is \( O(n^2) \) where \( n \) is the cardinality of \( V(Q) \). The pseudo-code of the described algorithm is shown in [9].

#### 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,

\[
Q_{MV}(SGQ, QR(MV), AG(Q), AGG(Q), HAV(Q)),
\]

---

3The cost can be estimated using an appropriate cost model, such as the linear cost model proposed in [6].
where $SG^j$ is the granularity of $QR(MV)$. The query block for $MV$ must be equivalent to $Q_M$ 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 $SG^j$ 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 a 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 for $MV$. 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 $MV$, and a query region $QR(MV)$, the query block for $MV$, $QB(MV)$, has the form,

\[
\begin{align*}
\text{SELECT} & \quad P, AGG \\
\text{FROM} & \quad R \\
\text{WHERE} & \quad JC, SC \\
\text{GROUP BY} & \quad G \\
\text{HAVING} & \quad HC
\end{align*}
\]

where the components are defined as follows:

\[
P = \bigcup_{1 \leq i \leq d} AG(Q, i)
\]

\[
AGG = \begin{cases} 
\{ag_{ij}(m')|as_{ij}a_{ij}m' \text{ is an attribute in } MV \text{ that is the result of } ag_{ij}(m) \text{ in } AG(Q), \\
\text{and } ag_{ij} = agg_{ij} \text{ if } ag_{ij} \in \{\text{MIN, MAX, SUM} \} \\
\text{and } ag_{ij} = SUM \text{ if } ag_{ij} = \text{COUNT} \}
\end{cases}
\]

\[
AGG = \begin{cases} 
\{ag_{ij}(m')|as_{ij}a_{ij}m' \text{ is an attribute in } MV \text{ that is the result of } ag_{ij}(m) \text{ in } AG(Q) \}
\end{cases}
\]

\[
R = \{MV, DT_i, NB_j|AG(MV, i) \nless \less SG(Q_{MV}, i) \text{ or } \}
\]

\[
\begin{cases} 
AG(MV, i) \nless AG(Q, i), AG(MV, i) = L_0, \\
AG(MV, j) \nless SG(Q_{MV}, j) \text{ or } \\
AG(MV, j) \nless AG(Q, j), AG(MV, j) \neq L_0
\end{cases}
\]

where the nested subquery block $NB_j$ has the form

\[
(\text{SELECT DISTINCT } AG(MV, j) \cup AG(Q, j) \\
\text{FROM } DT_j \\
\text{WHERE } p(QR(MV), DT_j) ) \quad NB_j
\]

where $p(QR(MV), DT_j)$ denotes the selection predicate on the attributes in $DT_j$ which is subsumed by the selection predicate for $QR(MV)$ with the sub-predicates identical to those of the predicate for $R(MV)$ excluded.

\[
J_C = ( \bigwedge_{DT_i \in R} (MV, AG(MV, i) = DT_i, AG(MV, i))) \land \\
( \bigwedge_{NB_j \in R} (MV, AG(MV, j) = NB_j, AG(MV, j)))
\]

\[
S_C = ( \bigwedge_{AG(MV, i) \nless SG(Q_{MV}, i)} p(QR(MV), DT_j)) \land \\
( \bigwedge_{DT_j \in R} p(QR(MV), DT_j))
\]

\[
G = \begin{cases} 
AG(Q) & \text{if } AG(Q) \prec AG(MV) \\
\phi & \text{if } AG(Q) = AG(MV)
\end{cases}
\]

\[
HC = \begin{cases} 
HA^V(Q) & \text{if } HA^V(Q) \neq \text{null and } \\
SG(MV) \prec AG(Q) & \text{if } \phi
\end{cases}
\]

where $HA^V(Q)$ is a logical formula on $P$ and $AGG$ derived from $HA^V(Q)$. The GROUP BY and HAVING clauses are included only when $G$ and $HC$ are not $\phi$, 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 $Q$.

- $S_1 = \{MV_i|SG(MV_i) \succeq AG(Q)\}$: The MVs in this class can be used to evaluate aggregation for all or a part of the aggregate groups of $Q$.
- $S_2 = \{MV_i|SG(MV_i) \prec AG(Q) \text{ or } SG(MV_i) \prec AG(Q)\}$: 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 which has a set of the aggregate functions $AGG^j(Q)$ 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 $AGG^j(Q)$ derived from $HA^V(Q)$ which is not null. Combining these two query blocks using a UNION operator generates the final rewritten query. Given $S_1 = \{MV_{i1}, MV_{i2}, \ldots, MV_{in}\}$ and $S_2 = \{MV_{j1}, MV_{j2}, \ldots, MV_{jn}\}$, the rewritten query has the form
( QB(MV1) UNION QB(MV2) UNION ... 
UNION QB(MVm) )
UNION 
(SELECT AG(Q), AGG(Q) 
FROM ( QB(MV21) UNION ALL QB(MV22) 
UNION ALL ... UNION ALL QB(MV2n) )
GROUP BY AG(Q)
HAVING HAV'(Q) )

Example 4 We describe rewriting steps for the query Q1 in Example 1.
Step 1: By Definition 3, V(Q1) = {MV1, MV2, MV3, Sales}. Using the proposed greedy algorithms, MV1, MV2, and MV3 are selected in the order. Their query regions are determined as follows (see Figure 2).
Q(R(MV1)) = {('USA', 'USA'), [1997, 1999], (-∞, +∞), (-∞, +∞)), ('Canada', 'Canada'), [1997, 1999], (-∞, +∞), (-∞, +∞))}
Q(R(MV2)) = {('USA', 'USA'), [1996, 1996], (-∞, +∞), (-∞, +∞))}
Q(R(MV3)) = {('Canada', 'Canada'), [1996, 1996], (-∞, +∞), (-∞, +∞))}

Step 2: We will show how the query block for MV1, QB(MV1), is generated. The normal form query for Q(R(MV1)) over the fact table is
Q(MV1)((nation, year, none, none), ('USA', 'USA', [1997, 1999], (-∞, +∞), (-∞, +∞)), ('Canada', 'Canada'), [1997, 1999], (-∞, +∞), (-∞, +∞)), (state, year, none, none), (SUM(sales_dollar))}
The components of QB(MV1) are determined by Definition 4 as follows. P = {state, year} from AG(Q1) in Example 3. AGG = {sum_dollars} because AG(MV1) = AG(Q1) and SUM(sales_dollars) in AGG(Q1) is named sum_dollars in MV1. Since AG(MV1, 1) = {state} \∋ \{nation\} = SG(MV1, 1) and AG(MV1, 1) = {state} \∋ \{store_id\} = LQ, a join of MV1 with a nested subquery block NB1 over Store is needed. With AG(MV1, 1) = AG(Q1, 1) = {state} and p(Q(R(MV1)), Store) = (nation = 'USA' OR nation = 'Canada'), NB1 has the form
SELECT DISTINCT state 
FROM Store 
WHERE nation = 'USA' OR nation = 'Canada'

However, since AG(MV1, 2) = SG(Q(MV1), 2) = AG(Q1, 2) = {year}, join of MV1 with Time is not necessary. Thus we have R = {MV1, NB2} and JC = (MV1.state = NB2.state). We have SC = p(Q(R(MV1)), Time) = (year ≤ 1999) since the predicate for Q(R(MV1)) is (year ≥ 1997 AND year < 1999) but (year ≥ 1997) is already included in the predicate for R(MV1). Since AG(MV1) = AG(Q1) and HAV(Q1) = null, GROUP BY and HAVING clauses are not required. Thus, QB(MV1) is written as follows.

SELECT state, year, SUM(dollar) 
FROM MV1, (SELECT DISTINCT state 
FROM Store 
WHERE nation = 'USA' OR nation = 'Canada') NB1 
WHERE MV1.state = NB1.state AND year ≤ 1999

The query blocks for MV2 and MV3 can be generated in a similar manner (see Q2 in Example 1).

Step 3: MV1, MV2, and MV3 satisfy SG(MV1) ⊃ AG(Q1), SG(MV2) ⊃ AG(Q1), and SG(MV3) ⊃ AG(Q1) as shown in Figure 5(a). Therefore, the query blocks for MV1, MV2, and MV3 can be integrated into a UNION multi-block query, i.e., Q1 in Example 1.

Example 5 In rewriting the query Q2 in Example 2 by the proposed method, MV1 and MV3 are selected in the first step. Since SG(MV1) ⊃ AG(Q2) and SG(MV3) ⊃ AG(Q2) hold as shown in Figure 5(b), the query blocks for MV1 and MV3 can be integrated into a UNION ALL–GROUP BY query, i.e., Q2 in Example 2.

5. Related Work

Several work on answering conjunctive queries using MVs has been proposed in the literature [11, 4, 7, 3]. [7] formalized the problem of finding rewritings of a conjunctive query under set semantics in terms of containment mappings from views to the query. [3] addressed the problem of optimizing queries using MVs. They proposed a rewriting method for conjunctive SPJ queries and 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 [1, 5, 10, 12]. [5] 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 those of the MV. Hence, the class of usable MVs and the types of rewritings generated by the algorithm are restrictive.

[10] 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: (a) All tables included in the definition of the MV must also appear in the definition of the query. (b) 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.

Proceedings of the 17th International Conference on Data Engineering (ICDE '01)  
1063-6382/01 $10.00 © 2001 IEEE

IEEE Computer Society
Since the MVs and queries in Example 1 and Example 2 satisfy neither of them, the algorithms proposed in [10] cannot perform the rewritings such as \( Q'_1 \) and \( Q'_2 \).

In [1, 12], new methods that exploit various MVs to evaluate complex queries in DW environments 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 [1] 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. [12] addressed the rewriting of queries including complex expressions, supergroup aggregations, and nested subqueries. They represented 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, performing compensation for the identified matching patterns and generating rewritten subqueries including the MV.

In summary, all the previous approaches have some restrictions. First, they did not effectively exploit meta-information such as dimension hierarchies in DWs and the characteristics of OLAP queries, so they can use only a limited class of MVs in rewriting queries. Second, they perform relatively simple types of rewritings compared with our proposed method which can generate the UNION ALL–GROUP BY rewriting. Last, little attention has been given to the problem of finding an efficient rewritten query among many candidates, considered in this paper.

6. Conclusions and Future Work

In this paper, we proposed a new approach to rewrite a given OLAP query using MVs existing in data warehouses. We defined the normal forms of OLAP queries and MVs based on the DH lattice derived from dimension hierarchies in DWs. We presented conditions for usability of MVs in rewriting OLAP queries and proposed a rewriting algorithm consisting 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 by using 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 meta-information of DWs and the characteristics of OLAP queries, our algorithm can rewrite a typical class of OLAP queries using various kinds of MVs together. It utilizes a much broader class of MVs and yields more general types of rewritings than other previous approaches can do. We also proposed a greedy heuristic method for selecting MVs and their query regions that results in a rewritten query which can be executed efficiently.

We are currently developing several heuristic methods for selecting an optimal or near-optimal set of MVs that are used in rewriting a query. Our future plans include extending the proposed rewriting method to deal with more general and complex OLAP queries and integrating the method with the process of query optimization.

References