Materialized View Selection in a Multidimensional Database

Elena Baralis
Politecnico di Torino
baralis@polito.it

Stefano Paraboschi
Politecnico di Milano
parabosc@elet.polimi.it

Ernest Teniente
Universitat Politècnica de Catalunya
teniente@lsi.upc.es

Abstract

A multidimensional database is a data repository that supports the efficient execution of complex business decision queries. Query response can be significantly improved by storing an appropriate set of materialized views. These views are selected from the multidimensional lattice whose elements represent the solution space of the problem.

Several techniques have been proposed in the past to perform the selection of materialized views for databases with a reduced number of dimensions. When the number and complexity of dimensions increase, the proposed techniques do not scale well.

The technique we are proposing reduces the solution space by considering only the relevant elements of the multidimensional lattice. An additional statistical analysis allows a further reduction of the solution space.

1 Introduction

A multidimensional database (MDDB) is a data repository that provides an integrated environment for decision support queries that require complex aggregations on huge amounts of historical data. An MDDB is a relational data warehouse, in which the information is organized following the so-called star-model [Kim96].

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

Its basic structure may be represented with the simple entity-relationship diagram depicted in Figure 1, in which all the D_i entities represent the dimensions of the MDDB, while the connecting relationship F is the fact table.

Each *dimension table* D_i contains all the information that is specific only to the dimension itself, while the *fact table* F correlates all dimensions and contains information on the attributes of interest for the intersection of all the dimensions. A new operator, the data-cube operator [GBLP96], has been proposed to perform the computation, on a single relation (the fact table), of one or more aggregate functions for all possible combinations of grouping attributes (which are the elements of the data-cube).

Since the computation of any of the elements of the cube is rather time-consuming, it may be precomputed to guarantee a satisfactory query response time to the user. On the other side, the materialization of the complete cube may be unfeasible, both because of its size and of the time required to update it when the fact table is updated. Hence, several techniques [Gup97, GHRU97, HRU96] have been proposed to select an appropriate subset of elements (which are indeed views on the fact table) to materialize.

The proposed algorithms work very well for medium size databases, but do not seem to scale well for the increased complexity of actual operational MDDB's. Indeed, as shown in the practical example of Section 1.1, in addition to the fact table, operational MDDB's may have several dimensions, each of which is characterized by a considerable number of attributes, most of which may be relevant for grouping computation as well. Thus, the presence of dimensions exponentially increases the number of elements in the cube.

If a set of user-specified relevant queries is available, exploiting this information may yield a significant reduction of the solution space. We observe that the number of representative queries is extremely small with respect to the total number of elements of the

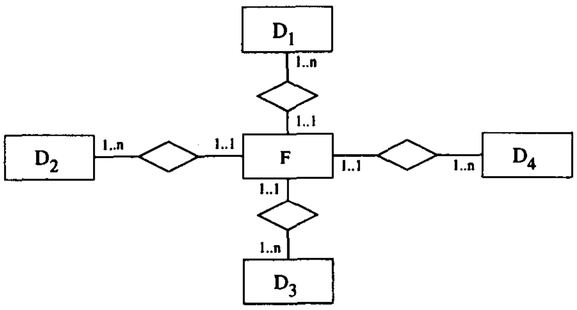

Figure 1: Entity-Relationship representation of an MDDB

complete data-cube. Then, the indication of the relevant queries is exploited to drive the selection of the candidate views, i.e., the views that, if materialized, may yield a reduction of the total cost. The number of candidate views may be further reduced by means of a heuristic based on the estimation of the size of the candidate views. Candidate views are discarded when their aggregation granularity is too big, because their materialization would not yield a substantial improvement in query response time with respect to using a higher level view.

The following section presents a practical example of an MDDB, while Section 1.2 discusses related work. In Section 2 a formal model of a multidimensional database is given, and its relation with the data cube model is discussed. Section 3 formally introduces the problem. Section 4 describes the technique to select views and an algorithm to perform the selection. Furthermore, the statistical technique to improve the selection's efficiency is presented, together with experimental results. Section 5 draws conclusions.

1.1 A Practical Example

Consider as a practical example, taken from [Kim96], the MDDB for a large grocery store chain, characterized by a large number of stores, each of which is a supermarket selling a wide variety of different products (e.g., grocery, frozen foods, bakery, etc.). The MDDB stores information on each sale in each store by day, also considering the promotions under which each product is sold. We can identify the following dimensions:

- *Product,* which can be characterized by more than 50 different attributes.
- *Store,* which characterizes each point of sale. The store dimension may contain more than 20 attributes.
- *Time,* which provides the appropriate detail to allow accurate analysis of the MDDB data. The time dimension may have more than 15 attributes.
- *Promotion,* which describes the characteristics of product promotions. Overall the promotion dimension is characterized by at least 10 attributes.

The fact table provides the sales information on which the actual financial analysis is performed. It includes the identifiers of all the dimensions and several attributes describing sales revenues (e.g., in terms of number of units sold). In this paper we consider a subset of the attributes of each dimension as relevant attributes for grouping computations: we assume 15 attributes for dimensions *Product* and *Store*, 9 attributes for *Time*, and 11 attributes for *Promotion.*

1.2 Related Work

Multidimensional data processing for relational data warehouses has raised considerable interest both in the scientific community [GBLP96, Gup97, GHRU97, HRU96, Wid95, ZGHW95] and in the industrial community, where several products have appeared.

The algorithms presented in this paper are closest to the work in [Gup97, GHRU97, HRU96]. In particular, [HRU96] considers an MDDB including only the fact table and proposes a greedy algorithm for the selection of an appropriate subset of the views of the complete data-cube to materialize. Work in [GHRU97] extends the previous results to the selection of both materialized views and indexes. Both works do not consider the cost of maintaining the materialized views in the model. A more general query and update model is proposed in [Gup97], where a theoretical framework for the view-selection problem is presented. In this context, a general algorithm and several heuristics are proposed. A detailed comparison of our work with [Gup97, GHRU97, HRU96] is performed in the relevant sections of the paper.

[RSS96] first gave a formal description of the multiple view maintenance problem. They present a framework for improving query performances by storing an additional set of materialized views and consider several heuristics for optimization. The cost model they propose, which includes both query and maintenance costs, uses an estimate of the number of disk accesses, by making hypotheses on the physical design of the database, while we selected a more abstract metric.

2 Multidimensional Database Model

Definition 2.1 *A* **Multidimensional Database** *is a collection of relations* $D_1, \ldots, D_n, F$*, where*

- *Each* D_i *is a* **dimension table,** *i.e., a relation characterized by an identifier* d_i *that uniquely identifies each tuple (*d_i *is the primary key of* D_i*).*

- *F is a* **fact table**, *i.e., a relation connecting all tables $D_1, \ldots, D_n$; the identifier of F is given by the foreign keys $d_1, \ldots, d_n$ of all the dimension tables it connects; the schema of F contains a set of additional attributes V (representing the values on which the aggregate functions are applied).*

The dimension tables may contain hierarchies.

Definition 2.2 *Let D be a dimension table with identifier d. An* **attribute hierarchy** *on D is a set of functional dependencies $FD_D = \{fd_0, fd_1, \ldots, fd_n\}$, where each fd_i is characterized by two sets of attributes $A_i^l \subset \mathrm{Attr}(D)$ and $A_i^r \subseteq \mathrm{Attr}(D)$ (respectively called left side and right side of the dependency); the dependency is represented as $fd_i : A_i^l \rightarrow A_i^r$.*

Each functional dependency fd_i is a constraint on the content of the dimension table D: for each tuple pair $t_1, t_2 \in D, t_1[A_i^l] = t_2[A_i^l] \Rightarrow t_1[A_i^r] = t_2[A_i^r]$. A dependency fd_0 with $A_0^l = \{d\}$ and $A_0^r = \{Attr(D) - d\}$ will always be present in FD_D. Functional dependencies must be acyclic, i.e., the graph obtained by drawing an arc from a_x to a_y, if $\exists fd_i \in FD_D \mid a_x \in A_i^l \land a_y \in A_i^r$, must be acyclic.

Definition 2.3 *An attribute hierarchy is* **tree-like** *if disregarding all transitive dependencies (i.e., dependencies that can be deduced from other dependencies), all the attributes appear at most once in the right side of a functional dependency and the left side always contains a single attribute. The transitive closure of a tree-like hierarchy is a* **tree-like complete** *(FD^{TC}) attribute hierarchy.*

In this paper we consider only tree-like attribute hierarchies. In the algorithms we will often use complete hierarchies, in order to simplify the algorithm description.

Example 2.4 *Consider the* Store *dimension of the MDDB described in Section 1.1, with key s and restricted to a set of attributes $\{z, c, s_t, n\}$ representing respectively zip code, county, state, and number of sale clerks. The dimension has the following tree-like attribute hierarchy: $\{fd_0 : s \rightarrow \{z, c, s_t, n\}, fd_1 : z \rightarrow c, fd_2 : c \rightarrow s_t\}$.*

Definition 2.5 *An* **MDDB attribute hierarchy** *FD_{DB} is the union of the attribute hierarchies FD_{D_j} of all the dimensions D_j appearing in the MDDB.*

Business analysis queries usually require the computation of aggregate functions on data grouped on an appropriate set of attributes.

Example 2.6 *Consider again the multidimensional database of our example, MDDB = {Product, Store, Time, Promotion, F}, where each dimension table has its corresponding attributes and with fact table $F = \{p, s, d, r, f\}$, having p, s, d (time dimension is represented with the granularity of day) and r as foreign keys of the dimension tables and f representing the amount of sales. The following queries can be requested on the MDDB:*

- *q_1 = total sales per product*
- *q_2 = total sales per product and store*
- *q_3 = total sales per product and day*
- *q_4 = total sales per product, store and day*

The queries we consider are select-join-groupby queries, with some restrictions on the allowed selection and join predicates. In particular, selection predicates are simple comparison predicates between a dimension attribute and a constant value (i.e., are of the type *attr* `<operator>` *const-value*). These predicates select a "slice" of data on which aggregates are computed. We envision a framework where the user provides a set of queries representative of the queries that the system must answer. For select queries we assume that only the attribute used is important and not the constants used in the comparison (which will generally change from query to query). To simplify the framework, we derive from a selection condition on an attribute a request for aggregating on that attribute. Consider a query q that returns the total sales of a particular store grouped by products. Query q can be answered by accessing the result of a query that returns the sales grouped by product and store, and selecting from them only the tuples relative to the specified store. In this way we can focus on queries computing aggregates. These considerations lead to the restriction that all the attributes appearing in a selection predicate must also appear as grouping attributes.

Join operations may be performed only between the fact table and any of its dimensions. Allowed predicates are equality predicates between a dimension's identifier and the corresponding fact table foreign key, while join predicates involving non-key attributes are disallowed.

We consider the standard SQL notion of group-by and aggregate function (considered functions are `count`, `sum`, `avg`, `min`, `max`). Grouping attributes may be drawn both from dimension and fact table attributes.

Definition 2.7 *Given a query q we define the query as* **characterized** *by the set of its group-by attributes A. We may represent the query as q^A.*

2.1 Data Cube

A fundamental characteristic of MDDB queries is that it is often possible to reuse the results of queries to answer other queries. In Example 2.6, we can use the result of query q_2 to answer query q_1, adding the sales across all stores to get the result.

The reuse of queries is strictly related to a new operator, the data-cube [GBLP96]. This operator, receiving as input a table T, a set of aggregating attributes A and a function f, computes the union of the results of the queries evaluating f, having as grouping attributes all possible combinations of attributes in A.

Definition 2.8 *Given an MDDB* $= \{D_1, \ldots, D_n, F\}$, *the data-cube lattice* **Cube-lattice** *of MDDB is the lattice of the set of all possible grouping queries that can be defined on the foreign keys of* F. *This lattice is characterized by the following elements:*

- *an ordering relation defined as the comparison between the sets of grouping attributes (i.e.,* $q^{A_1} \preceq q^{A_2} \leftrightarrow A_1 \subseteq A_2$*);*
- meet *operator as union of the grouping attributes;*
- join *operator as intersection of the grouping attributes;*
- *The query grouping on all the foreign keys as* top *element;*
- *The query computing the aggregate function on all the tuples of* F *as* bottom *element (empty set of grouping attributes).*

Previous work on the selection of views to materialize has concentrated on the computation of all the elements of the data cube. We instead consider a sample of representative queries which identify the elements that are really needed by the users, because only some of the queries that can be defined on the fact table will be generally requested on an MDDB. Requested queries are associated to a subset of the views of the data-cube lattice of MDDB.

Example 2.9 *Consider again the MDDB of Example 2.6. Figure 2 represents the data-cube lattice derived from the fact table* F *and shows the elements to which a query is associated.*

2.2 The Multidimensional Lattice

The presence of dimensions makes the problem more complex. The first aspect is the increase in the number of potential grouping attributes, which exponentially increases the number of elements of the lattice. The second aspect is the presence of hierarchies, which

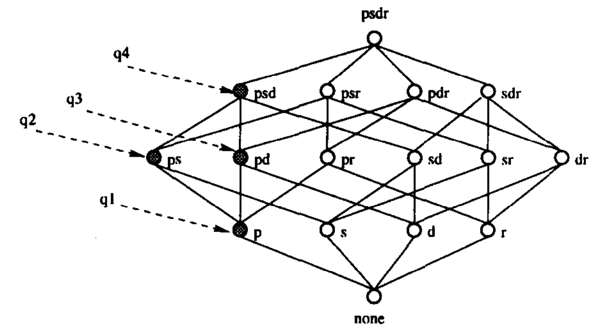

Figure 2: Data-cube lattice with associated queries

permit to remove some elements from the lattice. In fact, consider a query grouping on a dimension key d_i and also on an attribute a_j of the same dimension D_i. Since there exists a functional dependency from d_i to a_j, a query grouping on $\{d_i, a_j\}$ must produce the same result of the query grouping on $\{d_i\}$. This observation is the basis for the following generalization.

Definition 2.10 *Let* $q_x^{A_x}$ *and* $q_y^{A_y}$ *be two queries and* FD_{DB} *the MDDB attribute hierarchy. The operator* **ancestor** *(represented by the symbol* $\oplus$*) is defined by the following algorithm:*

Algorithm 2.11 *Ancestor of two queries.*

operator $\oplus$: $q_x^{A_x} \oplus q_y^{A_y} \Rightarrow q_z^{A_z}$;

$A_z := A_x \cup A_y$;
for each $fd_i \in FD_{DB}^{TC}$
  for each $a_j \in A_i^r$
    if $(\{a_i^l\} \cup a_j) \subseteq A_z$
      $A_z := A_z - a_j$;
return q^{A_z};

Algorithm 2.11 operates by building the union of the attributes characterizing the queries and eliminating all the elements for which there exists a functional dependency in FD_{DB}.

The result of applying the operator $\oplus$ to queries q_x and q_y is the "smallest" query that contains all the information necessary for answering q_x as well as q_y. If applied in a reflexive way (i.e., $q_x^{A_x} \oplus q_x^{A_x}$), it eliminates all redundant attributes.

Example 2.12 *Consider the hierarchy on the* Store *dimension of Example 2.4. The queries* $q^{\{n\}}$, $q^{\{c\}}$, *and* $q^{\{s_t\}}$ *can for example be computed from* $q^{\{c,n\}} = q^{\{n\}} \oplus q^{\{c\}} \oplus q^{\{s_t\}}$.

Definition 2.13 *The operator* **descendent** *(represented by symbol* $\ominus$*) is defined by the following algorithm:*

Algorithm 2.14 *Descendent of two queries.*

operator $\ominus$: $q_x^{A_x} \ominus q_y^{A_y} \Rightarrow q_z^{A_z}$;

$\quad$ *for each* $fd_i \in FD_{DB}^{TC}$
$\quad\quad$ *if* $\{a_i^l\} \subseteq A_x$
$\quad\quad\quad$ $A_x := A_x \cup A_i^r$;
$\quad\quad$ *if* $\{a_i^l\} \subseteq A_y$
$\quad\quad\quad$ $A_y := A_y \cup A_i^r$;
$\quad$ $A_z := A_x \cap A_y$;
$\quad$ *return* $q_z^{A_z} \oplus q_z^{A_z}$;

Algorithm 2.14 first extends the arguments A_x and A_y with all the attributes that can possibly be derived from them. It then considers their intersection A_z and finally removes from it the right sides of dependencies whose left side is contained in A_z.

The descendent operator computes the "greatest" among the set of attributes characterizing the queries that can be computed by both q_x and q_y. This operator is relevant only for the MD-lattice definition.

Definition 2.15 *Let $\{D_1, \ldots, D_n, F\}$ be a multidimensional database and FD_{DB} the MDDB attribute hierarchy. Consider the set of queries characterized by all the combinations among the attributes of $\{D_1, \ldots, D_n, F\}$, except the combinations that contain attributes on both sides of a functional dependency $fd_i \in FD_{DB}$. This set of queries identifies a lattice where:*

- *The ordering relation is given by the following definition: $q^{A_1} \preceq q^{A_2} \leftrightarrow (A_1 \ominus A_2) = A_1 \leftrightarrow (A_1 \oplus A_2) = A_2$;*
- *$\oplus$ is the* meet *operation;*
- *$\ominus$ is the* join *operation;*
- *The query grouping on all the foreign keys of F, $\{d_1, \ldots, d_n\}$, is the* top *element;*
- *The query computing the aggregate function on all the tuples of F is the* bottom *element (empty set of grouping attributes).*

We call this lattice the **MD-lattice** *(Multidimensional lattice).*

Comparing Definitions 2.8 and 2.15, it is easy to observe that without hierarchies the MD-lattice is equivalent to the Cube-lattice built on all the schema attributes (instead of the foreign keys only).

Example 2.16 *Consider the* Store *dimension of Example 2.12. The MD-lattice for an MDDB where* Store *is the only dimension is represented in Figure 3.*

An MD-lattice defines all possible ways of computing queries that can be defined on an MDDB in terms

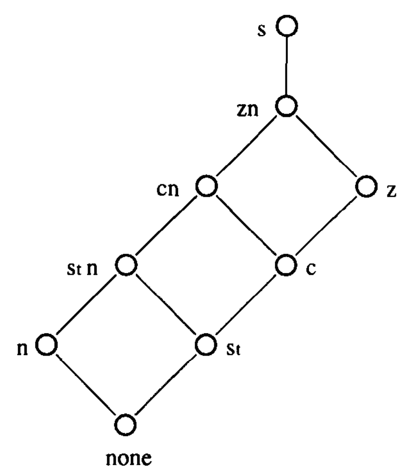

Figure 3: MD-lattice of the *Store* dimension

of other queries defined also on MDDB. In fact, if a query $q_i \preceq q_j$ then q_i can be answered using q_j. This property can be generalized.

Definition 2.17 *Let q_i and q_j be two queries on an MDDB. Then, the* **least upper bound** *(l.u.b.) of q_i and q_j in the MD-lattice is the query $q_i \oplus q_j$, a query (the most specialized) which can be used to answer both q_i and q_j.*

Hierarchies on cube lattices have already been introduced in [HRU96] and also in [SDNR96]. With respect to their description, we provide a more formal and more general treatment. Particularly, we explicitly permit the evaluation of queries combining attributes not directly related by functional dependencies (like the pair of attributes (z, n) in Figure 3).

We now distinguish between queries and views. In the following we will use the term query to refer to the representative queries provided by the user, while we will use the term view to refer to the elements of the lattices. When a query q computes the content of a view v of a lattice, we say that the query q is associated to view v. We also say that a view is characterized by a set of attributes A if A is the set of grouping attributes of v, represented as v^A.

Definition 2.18 *A view v_i, or equivalently its associated query q_i, is said to* **depend** *on view v_j, if q_i can be answered using as only inputs the content of v_j together with any appropriate dimension D_i.*

There is a strict relationship between dependence of a query on a view and the ordering relation on an MD-lattice. In fact, if we consider a view v_i depending on view v_j, it must happen that $v_i \preceq v_j$.

An important relationship exists between the ordering of the elements in the lattice given by relation $\preceq$ and the cardinalities (i.e., number of tuples) of the views. In fact, for each pair of views $v_i, v_j \in$ *MD-lattice*, if $v_j \preceq v_i$, v_j can be computed by v_i with a further grouping. Since grouping can only reduce the number of tuples, it follows that $|v_j| \leq |v_i|$, where $|v|$ represents the cardinality of v.

2.3 Determining the number of views of an MD-lattice

The number of views of an MD-lattice depends on the number of attributes of the dimensions of the MDDB and on the number and structure of the attribute hierarchies.

In the absence of hierarchies on the dimensional tables, the number of views of the MD-lattice is given by the following formula:

$$n_{total} = \prod_i (2^{n_i} + 1)$$

where i is the number of dimensions of MDDB and n_i is the number of non-key attributes of each dimension D_i. In the presence of hierarchies, the number of views in the local lattices is reduced and the above formula for n_{total} constitutes an upper bound. We only provide as an example the determination of the number of elements for the MD-lattice of the MDDB described in Section 1.1.

Example 2.19 *Consider the MDDB of Section 1.1. The number of views of the lattices that can be built on every dimension, considering hierarchies, are:* n_{Product} *= 12,289;* n_{Store} *= 8,193;* n_{Time} *= 129;* $n_{\text{Promotion}}$ *= 1025. The total number of views of the resulting MD-lattice is then the product of all these values, obtaining* |MD-lattice|$= 1.3313 \cdot 10^{13}$.

3 The MDmat problem

The fundamental problem we want to solve is to find the set of view to materialize that maximizes the performances of an MDDB in answering a given set of representative queries. The trade-off consists in choosing a set of materializations able to speed up query response time without requiring too much work to keep the materializations current with respect to the modifications on the tables of the MDDB.

Definition 3.1 *Given an MDDB $\mathcal{DB}$, a set of queries $\mathcal{Q}$, and a set of frequency values $\mathcal{F}$ of queries in $\mathcal{Q}$ and updates on the tables of $\mathcal{DB}$, the* **MDmat-problem** *(Multidimensional Database Materialization problem) is represented by $\Theta(\mathcal{DB}, \mathcal{Q}, \mathcal{F})$. A solution to the problem $\Theta(\mathcal{DB}, \mathcal{Q}, \mathcal{F})$ is a set of views of the MD-lattice $\mathcal{M}$ (which can contain views that are not associated to queries in $\mathcal{Q}$). A trivial solution, $\mathcal{M} = \emptyset$, is always possible, which represents the situation where no additional materialization is available and all the queries must be answered directly by the fact table (the root of the hierarchy).*

To identify the optimal $\mathcal{M}$, we must define a cost function. The cost function is composed of two parts: the *query cost* and the *update cost*.

Definition 3.2 *Let $\mathcal{F}$ be a set of frequencies f_{q_i}, each associated to a query $q_i \in \mathcal{Q}$, representing the frequency with which query q_i is asked. Let $c_{q_i}(\mathcal{M})$ be the cost to compute q_i from the set of materializations $\mathcal{M}$ (discussed in Section 3.1 below). Then, the* **total query cost** *$\mathcal{C}_Q(\mathcal{Q}, \mathcal{M}, \mathcal{F})$ is given by:*

$$\mathcal{C}_Q(\mathcal{Q}, \mathcal{M}, \mathcal{F}) = \sum_{q_i \in \mathcal{Q}} f_{q_i} \cdot c_{q_i}(\mathcal{M})$$

Definition 3.3 *Consider the same set of materialized views $\mathcal{M}$. Let f_{m_i} be the frequency with which the materialized view $m_i \in \mathcal{M}$ is modified and $c_u(m_i)$ its update cost (update frequency and cost are further discussed in Section 3.2). Then, the* **total update cost** *$\mathcal{C}_M(\mathcal{M}, \mathcal{F})$ is given by:*

$$\mathcal{C}_M(\mathcal{M}, \mathcal{F}) = \sum_{m_i \in \mathcal{M}} f_{m_i} \cdot c_u(m_i)$$

Definition 3.4 *Given an MDmat-problem described by $\Theta(\mathcal{DB}, \mathcal{Q}, \mathcal{F})$, the* **cost of a solution** *$\mathcal{M}$ is the sum of query and update costs:*

$$\mathcal{C}(\mathcal{Q}, \mathcal{M}, \mathcal{F}) = \mathcal{C}_Q(\mathcal{Q}, \mathcal{M}, \mathcal{F}) + \mathcal{C}_M(\mathcal{M}, \mathcal{F})$$

3.1 Query Cost

We do not specify completely a cost model. The techniques we describe are applicable to a wide choice of cost models, from simple to complex ones. Very few restrictions have to be imposed on the cost formulas to permit the adoption of our results.

The function $c_{q_i}(\mathcal{M})$ returns the cost of computing query q_i given a set of materializations $\mathcal{M}$. We make two hypotheses about the query cost function: that each query cost depends on a unique element in $\mathcal{M}$, and that the cost is monotonic with the size of the materialization on which the query depends.

Definition 3.5 *A query cost function $c_{q_i}(\mathcal{M})$ is* **restrictible** *if it is always equal to the least among the values obtained by considering $c_{q_i}(m_j)$, for all the $m_j \in \mathcal{M}$ on which query q_i depends.*

Every query of the type introduced in Section 2 is restrictible (having always the fact table in the set of materializations).

Definition 3.6 *Given a query q_i, a restrictible query cost function c_{q_i}, and a set of materializations $\mathcal{M}$, the materialization m_j such that $c_{q_i}(\mathcal{M}) = c_{q_i}(m_j)$ is the* **least expensive materialization** *(for query q_i among the elements of $\mathcal{M}$).*

Definition 3.7 *A query cost function c_{q_i} (MD-lattice) is* **monotonic** *if for all $v_j, v_k \in$MD-lattice on which q_i depends, $|v_j| < |v_k| \rightarrow c_{q_i}(v_j) \le c_{q_i}(v_k)$, where $|v|$ represents the cardinality of v and the arrow denotes logical implication. From the observation on the cardinalities of the views in the lattice, a monotonic function also guarantees that $v_j \preceq v_k \rightarrow c_{q_i}(v_j) \le c_{q_i}(v_k)$.*

The simple cost model introduced in [HRU96] is for example both restrictible and monotonic: the cost of answering a query q is set equal to the number of tuples read to return the answer. An extension of this model permits to consider the availability of indexes to accelerate the execution of queries, as described in [GHRU97]. Sophisticated cost functions can be designed which adequately model the system and still offer restrictibility and monotonicity.

3.2 Update Cost

We consider the insertion of tuples into either the fact table or the dimension tables as the prevailing type of modification in an MDDB. We assume that all performed insertions do not violate the referential integrity constraint between the dimension tables and the fact table.

We simplify our cost model assuming a unique update frequency f_u valid for all the materializations, where f_u represents the frequency of insertions into the fact table. The update cost function of Definition 3.3 becomes:

$$C_{\mathcal{M}} = f_u \cdot \sum_{m_j \in \mathcal{M}} c_u(m_j)$$

Definition 3.8 *An update cost function c_u is* **monotonic** *if for all $m_j, m_k \in \mathcal{M}$, $|m_j| < |m_k| \rightarrow c_u(m_j) \le c_u(m_k)$, where $|m_j|$ represents the cardinality of m_j.*

We require a monotonic update cost function.

4 Identification of Candidate Views

Compared to the number of nodes that form the MD-lattice, the number of representative queries is extremely small. This considerable sparsity of queries among the views of an MD-lattice suggests that only some of these views are relevant when deciding how to minimize the total cost. The idea of our reduction technique is to consider only those views of an MD-lattice that, when materialized, can provide some contribution to reduce the total cost. We call them candidate views.

Definition 4.1 *A view v_i belonging to an MD-lattice is a* **candidate view** *if one of the following two conditions holds:*

- *View v_i is associated to some query q_i.*
- *There exist two candidate views v_j and v_k, and v_i is the least upper bound (l.u.b.) of v_j and v_k.*

Let v_i be a candidate view of an MD-lattice. Then, choosing v_i for materialization may provide some benefit when looking for the solution that reduces the total cost. There are in fact two cases:

1. v_i has an associated query q_i.

   It is trivial to show that the materialization of a view associated to a query can help the computation of the query. Starting from the definition of the cost of a solution, we obtain the following formula, which identifies the query frequency f_{q_i} that makes convenient the materialization of view v_i when a set of views $\mathcal{M}$ is already materialized:

   $$f_{q_i} > f_u \cdot \frac{c_u(v_i)}{c_{q_i}(v_t) - c_{q_i}(v_i)}$$

   where $v_t \in \mathcal{M}$ represents the least expensive materialization that can be used to answer q_i.

2. There exist at least two candidate views, v_j and v_k, such that v_i is the l.u.b. of v_j and v_k.

   It is enough to show that there exists at least one case in which materializing v_i provides some benefit. Assume that there exist two queries q_j and q_k associated to views v_j and v_k, respectively. The contribution of queries q_j and q_k and views v_j and v_k to the cost $C(\mathcal{Q}, \mathcal{M}, \mathcal{F})$, when v_j and v_k are materialized and v_i is not, is:

   $$C_1 = f_u \cdot c_u(v_j) + f_{q_j} \cdot c_{q_j}(v_j) + f_u \cdot c_u(v_k) + f_{q_k} \cdot c_{q_k}(v_k)$$

   The contribution to the cost $C(\mathcal{Q}, \mathcal{M}, \mathcal{F})$ if v_i is materialized and v_j and v_k are not, with v_i being the least expensive materialization for both q_j and q_k, is:

   $$C_2 = f_u \cdot c_u(v_i) + f_{q_j} \cdot c_{q_j}(v_i) + f_{q_k} \cdot c_{q_k}(v_i)$$

   Choosing v_i for materialization will decrease the total cost if $C_1 > C_2$, i.e., when:

$$f_u > \frac{f_{q_j} \cdot (c_{q_j}(v_i) - c_{q_j}(v_j)) + f_{q_k} \cdot (c_{q_k}(v_i) - c_{q_k}(v_k))}{c_u(v_j) + c_u(v_k) - c_u(v_i)}$$

with the hypothesis that $c_u(v_j)+c_u(v_k)-c_u(v_i) > 0$. The intuition is that if the cost of updating views v_j and v_k is greater than the cost of updating v_i, for a high enough update frequency f_u it may be convenient to materialize view v_i instead of v_j and v_k.

Moreover, if we want to ensure that candidate views are the only relevant views for the process of deciding which views to materialize, we must prove also that materializing a non-candidate view may never decrease the total cost. This is done in Theorem 4.3; before the theorem we need a new definition.

Definition 4.2 *Given a non-candidate view v_i with at least one candidate view depending on it, we call* **most directly dependent candidate view** *the unique candidate view v_j such that all the remaining candidate views that depend on v_i also depend on v_j.*

We can determine the most directly dependent candidate view of v_j by taking the set V' of all the candidate views that depend on v_j and computing the l.u.b. of all the views in V'. This view is unique (because a l.u.b. is always determined), is a candidate view (because is a l.u.b. of candidate views) and belongs to V' (because $v_a \preceq v_i \wedge v_b \preceq v_i \rightarrow (v_a \oplus v_b) \preceq v_i$).

Theorem 4.3 *Let v_i be a non-candidate view of an MD-lattice. Then, the choice of v_i for materialization is always dominated by the choice of a candidate view.*

Proof: We will assume that there exists a non-candidate view v_i that belongs to the set of materializations $\mathcal{M}$ of the optimal solution, and we will get a contradiction. We distinguish two cases.

1. There is no candidate view depending on v_i.

   The contribution of view v_i to the cost $C_1 = \mathcal{C}(\mathcal{Q}, \mathcal{M}, \mathcal{F})$ of the solution $\mathcal{M}$ for $\Theta(\mathcal{DB}, \mathcal{Q}, \mathcal{F})$ ($v_i \in \mathcal{M}$) is represented only by the contribution to the update cost $f_u \cdot c_u(v_i)$, since no query can use v_i (otherwise there would be candidate views depending on v_i). The cost of the solution $\mathcal{M} - v_i$ is equal to $C_2 = \mathcal{C}(\mathcal{Q}, \mathcal{M} - v_i, \mathcal{F})$, where $C_1 = C_2 + f_u \cdot c_u(v_i)$. Since materializing v_i must provide some benefit, it must happen that $C_1 < C_2$, i.e., $f_u \cdot c_u(v_i) < 0$. Then, we get a contradiction because both f_u and $c_u(v_i)$ must be positive.

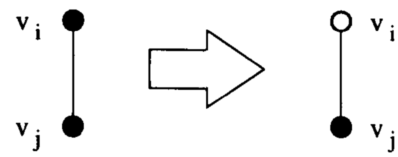

Figure 4: A configuration of materializations

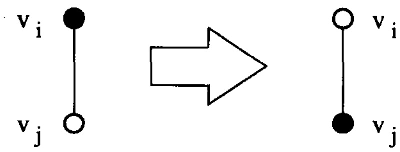

Figure 5: A configuration of materializations

2. There exists at least one candidate view depending on v_i.

   We first identify view v_j, the most directly dependent candidate view of v_i. We then consider separately two sub-cases, represented in Figures 4 and 5.

Case 1 Both v_j and v_i are materialized (represented in Figure 4). Let $\mathcal{M}' = \mathcal{M} - v_i - v_j$. The cost of this solution is:

$$C_3 = \sum_{q_i \in \mathcal{Q}} f_{q_i} \cdot c_{q_i}(\mathcal{M}' \cup v_j \cup v_i) + f_u \cdot \sum_{v_k \in \mathcal{M}' \cup v_j \cup v_i} c_u(v_k)$$

We observe that view v_i is not used by any query $q_i \in \mathcal{Q}$, because all the queries that depend on v_i also depend on v_j, and since $v_i \preceq v_j$ it follows that $c_{q_i}(v_i) > c_{q_i}(v_j)$. We can then remove it from the first term of the formula for C_3 obtaining:

$$C_3 = \sum_{q_i \in \mathcal{Q}} f_{q_i} \cdot c_{q_i}(\mathcal{M}' \cup v_j) + f_u \cdot \sum_{v_k \in \mathcal{M}' \cup v_j \cup v_i} c_u(v_k)$$

The cost of the solution with materialization $\mathcal{M}' \cup v_j$ is:

$$C_4 = \sum_{q_i \in \mathcal{Q}} f_{q_i} \cdot c_{q_i}(\mathcal{M}' \cup v_j) + f_u \cdot \sum_{v_k \in \mathcal{M}' \cup v_j} c_u(v_k)$$

The difference between C_3 and C_4 is represented by the single term $f_u \cdot c_u(v_i)$. In order for C_3 to be the optimum (i.e., $C_3 < C_4$), it must happen that $f_u \cdot c_u(v_i) < 0$. We get a contradiction since both f_u and $c_u(v_i)$ must be positive.

Case 2 v_i is materialized and v_j is not (represented in Figure 5). Let $\mathcal{M}' = \mathcal{M} - v_i$. The cost of this solution is:

$$C_5 = \sum_{q_i \in \mathcal{Q}} f_{q_i} \cdot c_{q_i}(\mathcal{M}' \cup v_i) + f_u \cdot \sum_{v_k \in \mathcal{M}' \cup v_i} c_u(v_k)$$

In particular, this solution must be better than the solution where v_j is materialized but v_i is not, which has the cost C_4 defined in the above formula. But each term $c_{q_i}(\mathcal{M}' \cup v_j)$ in C_4 must be less than $c_{q_i}(\mathcal{M}' \cup v_i)$ in C_5, because all the queries that depend on v_i must also depend on v_j, v_j is smaller than v_i and the query cost function is monotonic. Term $c_u(v_j)$ must also be smaller than $c_u(v_i)$, for the monotonicity of update cost with view size. It is then impossible for C_5 to be smaller than C_4. □

Once we have proved that candidate views are the only views that are relevant to decide which materializations minimize the total cost, we can define the sub-lattice obtained by considering only candidate views.

Definition 4.4 *Given an MD-lattice and a set of queries Q, the set of its candidate views identifies also a lattice, where the* join *and* meet *operations are identical to the ones defined for the MD-lattice. We call this sub-lattice the* **MDred-lattice**.

Given a set Q of queries and the MDDB attribute hierarchy, we can identify all the elements of the MDred-lattice. This is performed by means of the following algorithm.

Algorithm 4.5 *The MDred-lattice Construction Algorithm*

Function MDred-lattice(Q): <set of elements>;
/ input: a finite set Q of queries */*
/ output: the MDred-lattice obtained by Q */*

```
L := Q; lastViews := L; newViews := ∅;
while lastViews ≠ ∅
    for each v_i ∈ lastViews do
        for each v_j ∈ L, v_j ≠ v_i do
            if v_i ⊕ v_j ∉ L
                newViews := newViews ∪ (v_i ⊕ v_j);
    L := L ∪ newViews;
    lastViews := newViews; newViews := ∅;
return L;
```

Algorithm 4.5 iteratively extends the set L. L is initially equal to the set of views in Q. The l.u.b. of all the pairs of elements in L are added to L, and the process iterates by considering the l.u.b. of all the pairs of views obtained by combining the new elements of L with all the elements of L, until a fixpoint is reached. In Section 4.2 we present the experimental results obtained with the above technique.

4.1 A Heuristic Reduction

When building the MDred-lattice, a simple heuristic technique can be used to further reduce the size of the lattice, removing views which are not expected to contribute to the optimal solution. This heuristics is based on the estimate of the size of the materialization. These estimates can be done following traditional query estimation techniques for aggregates. If the skewness of data is of concern, it is also possible to use a recently described technique [SDNR96], which obtains an estimate which is quite precise for a wide range of skewness in the distribution.

According to the estimates on the size of views, it is possible to determine when the level of aggregation used is too detailed and the materialization offers a very limited help in answering the query with respect to a materialization of a higher level view.

Example 4.6 *Consider a dimension A with 1,000 tuples. Let a view contain an aggregation for the pair of attributes $\{A_1, A_2\}$, where each attribute has 100 distinct values. There are 10,000 possible pairs of values of the attributes, but since there are only 1,000 tuples, at most 1,000 tuples can be present in the view. If the data is uniformly distributed, the estimate of the size of the view is quite close to 1,000. Instead of materializing this view, it could be convenient to use the view which has the key of dimension A as aggregating attribute. The advantage is that it will be easier to reuse this view in the computation of other aggregates and the number of elements of the lattice will be reduced.*

We can easily modify the MDred-lattice construction algorithm to estimate at each step the dimension of a view, and substituting to it a higher level view if the reduction criteria are not met.

Definition 4.7
A **size estimating function** size(v^A) *is a function that, applied to a view characterized by a set of attributes A of an MDDB, returns an estimate of the number of tuples of the view.*

Algorithm 4.8 *Heuristic Reduction Algorithm*

function heuristicRed(A,$\overline{p}$): <set of attributes>;

repeat
  Stop := True;
  for each $fd_i \in FD_{DB}$
    if $(A \cap A_i^r \neq \emptyset) \wedge (\overline{p} \cdot size(v^{A_i^l}) < size(v^{A \cap A_i^r}))$
      $A := A - A_i^r$; $A := A \cup A_i^l$;
      Stop := False;
until Stop;
return A;

Algorithm 4.8 considers all the functional dependencies to identify when the attributes in A that appear on the right side (or a subset of them) of a functional dependency produce a size estimate which does

N. of queries	Views in the MDred-lattice	Views after the heuristic reduction
20	795	85
25	1,417	93
30	2,735	114
35	3,648	129
40	12,378	145
45	21,559	157

Table 1: The application of the techniques of Section 4

not differ more than a ratio $\overline{p}$ from the estimate of the size of the attributes on the left side. When this happens, the attributes on the right side are replaced by the attributes on the left side, effectively moving from a view v_i to a view v_j, where $v_i \preceq v_j$.

The parameter $\overline{p}$ represents the threshold on the amount of increase in size below which the left side of a dependency should be used in place of the right side. The user should provide this value, evaluating a trade-off between accuracy of the solution and reduction in the size of the solution space.

4.2 Experiments

We have applied the techniques proposed in Sections 4 and 4.1 to the MDDB of Section 1.1. The results are synthetically presented in Table 1. According to [Kim96], we have considered the following sizes: *Fact* table, 657 million tuples; *Product*, 30,000 tuples; *Store*, 300 tuples; *Time*, 730 tuples; *Promotion*, 2,000 tuples.

In the second column of Table 1, we show the number of views of the MDred-lattice. The table describes several runs of the algorithms with different tests, identified by the number of queries considered in each case. In the third column, we show the results obtained when applying our heuristic reduction to the same tests as before (with $\overline{p} = 0.95$). From these experiments, it can be seen that the number of views is drastically reduced. We recall that in Section 2.3 the original MD-lattice was shown to contain $1.3313 \cdot 10^{13}$ views.

5 Conclusions

We have dealt with the problem of selecting which views to materialize in a multidimensional database. We have proposed two techniques which can significantly reduce the number of views to consider, starting from a lattice representation of the solution problem.

A further contribution of this paper is a formal framework for the definition of attribute hierarchies, which allowed us to explicitly consider their effect during the lattice construction. This framework further formalizes and generalizes attribute hierarchy handling techniques previously proposed.

We have developed a small prototype that allowed us to obtain the experimental results described in Section 4.2. An extended version of this paper can be retrieved at *http://www.elet.polimi.it/idea/viewsel.ps*. There we present an incremental technique which complements the techniques presented in this paper.

Acknowledgements

The authors thank Stefano Ceri for his inspiration, direction and support.

References

[AAD+96] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 22nd VLDB*, pages 506–521, Mumbai, Sept. 1996.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proc. 12th ICDE*, pages 152–159, New Orleans, March 1996.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proc. 13th ICDE*, pages 208–219, Manchester, UK, April 1997.

[Gup97] H. Gupta. Selection of views to materialize in a data warehouse. In *Proc. Sixth ICDT*, pages 98–112, Delphi, Jan. 1997.

[HRU96] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. ACM SIGMOD '96*, pages 205–216, Montreal, June 1996.

[Kim96] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, 1996.

[RSS96] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proc. ACM SIGMOD '96*, pages 447–458, Montreal, June 1996.

[SDNR96] A. Shukla, P. M. Deshpande, J. F. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *Proc. 22nd VLDB*, pages 522–531, Mumbai, Sept. 1996.

[Wid95] J. Widom, editor. *Special Issue on Materialized Views and Data Warehousing, IEEE Data Engineering Bullettin*, volume 18, June 1995.

[ZGHW95] Y. Zhuge, H. Garcia Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proc. ACM SIGMOD '95*, pages 316–327, San Jose, May 1995.