Maintaining Data Cubes under Dimension Updates

Carlos A. Hurtado
University of Toronto
chl@db.toronto.edu

Alberto O. Mendelzon
University of Toronto
mendel@db.toronto.edu

Alejandro A. Vaisman
Universidad de Buenos Aires
av2n@dc.uba.ar

Abstract

OLAP systems support data analysis through a multidimensional data model, according to which data facts are viewed as points in a space of application-related "dimensions", organized into levels which conform a hierarchy. The usual assumption is that the data points reflect the dynamic aspect of the data warehouse, while dimensions are relatively static. However, in practice, dimension updates are often necessary to adapt the multidimensional database to changing requirements. Structural updates can also take place, like addition of categories or modification of the hierarchical structure. When these updates are performed, the materialized aggregate views that are typically stored in OLAP systems must be efficiently maintained. These updates are poorly supported (or not supported at all) in current commercial systems, and have received little attention in the research literature. We present a formal model of dimension updates in a multidimensional model, a collection of primitive operators to perform them, and a study of the effect of these updates on a class of materialized views, giving an algorithm to efficiently maintain them.

1 Introduction

A *data warehouse* is a large collection of integrated data, built to assist the decision-making process. The class of queries required on data warehouses for data analysis and visualization is known as OLAP, for on-line analytical processing. OLAP systems often support a multidimensional data model, according to which a data *fact* is viewed as a mapping from a point in a space of *dimensions* into one or more spaces of *measures*. For example, suppose we have data about policies sold by an insurance company. We may have different kinds of measures, such as number of policies sold, dollar amount, coverage amount, which can be analyzed in terms of dimensions such as kind of coverage, customer, salesperson, date of sale, geographic region, etc. Dimensions are often organized into hierarchical levels; for example, a geographic region could consist of a town or city name at the lowest level, with counties and then states or provinces and then areas at higher levels. These levels enable the definition of different levels of data aggregation, a central issue in data analysis.

In a relational implementation of OLAP, we can think of facts as being stored in *fact tables*, while each dimension is described in a *dimension table*. The usual assumption in OLAP systems is that fact tables reflect the dynamic aspect of the data warehouse, while data in dimension tables are relatively static. However, changes in the data will often require updates to the dimension tables: new salespersons may be hired or fired, new kinds of coverage introduced or discontinued, etc. In addition, structural changes to the dimension hierarchies also occur: for example, regions may be reorganized or merged or split. Kimball [11] talks about *slowly changing dimensions*, covering only the first kind of change discussed above, not structural changes.

An OLAP database usually stores *materialized views* which precompute aggregates in order to speed-up the complex queries often needed. This implies that, whenever an update is performed over the underlying database, the corresponding changes should also be applied to these views, which constitutes the well-known problems of *view maintenance* and *incremental view maintenance* [13, 6, 14].

In this paper we address the dimension update problem. Its main contributions are : (a) a formal model of dimension updates in a multidimensional model, covering updates to the domains of the dimensions and structural updates to the dimension hierarchies, together with a definition of a collection of primitive operators to perform these updates, and (b) a study of the effect of these updates over a class of materialized views over the dimension levels, and an algorithm to efficiently maintain them, which turns out to be more efficient than the well-known *summary-delta method*, when dealing with dimension updates.

The rest of the paper is organized as follows : In Section 2 we introduce our data model, over which we will describe dimension updates. In Section 3 we define dimension updates and the basic set of operators to perform them. In Section 4 we approach the problem of cube maintenance. We conclude in Section 5, presenting related work, conclusions, and comments on future directions.

2 Multidimensional Model

2.1 Dimensions and Fact Tables

Assume the following sets: a set of level names $\mathbf{L}$, where each level $l \in \mathbf{L}$ is associated with a set of values $dom(l)$;a set of dimension names $\mathbf{D}$; and a set of fact table names $\mathbf{F}$.

Definition 1 (Dimension Schema and Instance) *A* **dimension schema** *is a tuple* $(dname, L, \preceq)$, *where* $dname \in \mathbf{D}$ *is the name of the dimension,* $L \subseteq \mathbf{L}$ *is a finite set of levels, which contains distinguished level name All, such that* $dom(All) = \{all\}$. *Finally,* $\preceq$ *is a relation over levels, such that* $\preceq^*$, *its transitive and reflexive closure , is a partial order, with a unique bottom level, called* l_{inf}, *a unique top level, All, and, for every level* $l \in L$, $l_{inf} \preceq^* l$ *and* $l \preceq^*$ *All hold. Moreover, if* l_a *and* l_b *are levels in* L, *and* $l_a \preceq^* l_b$, *then* $l_a \not\preceq l_b$.

A **dimension instance** *is a tuple* (D, RUP) *where* D *is a dimension schema, and* RUP *is a set of partial functions such that (a) for each pair of levels* l_1, l_2 *such that* $l_1 \preceq l_2$, *there exists a roll-up function (partial function)* $RUP_{l_1}^{l_2} : dom(l_1) \rightarrow dom(l_2)$, *(b) for each pair of paths, in the graph with nodes in* L *and edges in* $\preceq$, $\tau_1 =< l_1, l_2, \ldots, l_{n-1}, l_n >$, *and* $\tau_2 =< l_1, l'_2, \ldots, l'_{m-1}, l_m >$,$l_n = l_m$, *we have* $RUP_{l_1}^{l_2} \circ \ldots \circ RUP_{l_{n-1}}^{l_n} = RUP_{l'_1}^{l'_2} \circ \ldots \circ RUP_{l'_{m-1}}^{l'_m}$, *and (c)for each triple of levels* $l_1, l_2, l_3 \in L$ *such that* $l_1 \preceq l_2$ *and* $l_2 \preceq l_3$,$ran(RUP_{l_1}^{l_2}) \subseteq dom(RUP_{l_2}^{l_3})$.

Example 1 *Figure 1 shows an example of a dimension schema and instance for a dimension* Product. *Here, for the dimension schema, dname = Product,* $L =$ *{ItemId, Brand, Company, Category, Corporation, Product, All}. Relation* $\preceq$ *contains the following pairs: ItemId* $\preceq$ *Brand,Brand* $\preceq$ *Company,Company* $\preceq$ *Corporation,Corporation* $\preceq$ *All, ItemId* $\preceq$ *Category,Category* $\preceq$ *Corporation.*

In what follows, dimension will stand for dimension instance, except when noted. We will also denote by RUP^* the set of roll up functions that indirectly relate

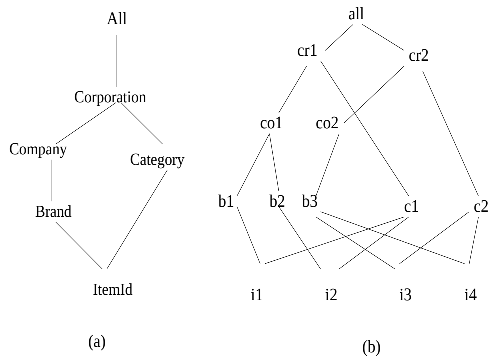

Figure 1: (a) A dimension schema for Product, (b) A dimension instance for Product.

the instances of the levels (RUP denotes the direct rollup functions); this set contains a roll up function for each pair of levels $l_m, l_n \in L$, $l_m \preceq^* l_n$, such that if $l_m = l_n$, $RUP_{l_m}^{l_n} = identity$; otherwise, if $< l_m \ldots l_n >$ is a path from l_m to l_n in the graph with nodes in L and edges in $\preceq$, $RUP_{l_m}^{l_n} = RUP_{l_m}^{l_{m+1}} \circ \ldots \circ RUP_{l_{n-1}}^{l_n}$.

It can be shown that roll up functions defined over the same level must have the same domain, the set of all values at that level. More formally, given a dimension instance, for each triple of levels l, l' and l'' of it, such that $l \preceq l'$ and $l \preceq l''$, $dom(RUP_l^{l'}) = dom(RUP_l^{l''})$. Given a dimension and a pair of levels l and l', such that $l \preceq l'$, the *instance set* of l, or $instset(l)$, is the set containing values for the level, defined by $instset(l) = dom(RUP_l^{l'})$. As an example, for the dimension instance *Product* given in example 1 and Figure 1, the instance set of level *ItemId*, is $\{i_1, i_2, i_3, i_4\}$.

Definition 2 (Fact Table) *A* **fact table schema** *is a tuple* $s = (fname, Lset, m)$, *where* $fname \in \mathbf{F}$ *is a fact table name, Lset is a set of levels, and m is a level, called the* measure *of the fact table. Moreover, given a fact table schema* $(fname, Lset, m)$, *a* point *is a mapping from each level in Lset, to a value in* $dom(l_i)$. *Given a fact table schema* $s = (fname, Lset, m)$, *a* **fact table instance** *over it is a partial function which maps points of s to elements in* $dom(m)$.

Definition 3 (Multidimensional Database) *A* **multidimensional database schema** *is a pair* $MS = (DS, FS)$, *where DS is a set of dimension schemas, and FS is a set of fact table schemas. A* **multidimensional database instance** *is a tuple* $MD = (D, F)$, *where D is a set of dimensions, and F a set of fact tables.*

Notation Given a dimension set D, a *level group* is a set of levels containing exactly one level for each dimension in $D\backslash\{Measure\}$, where $Measure$ is a distinguished dimension(see below). GB_D will denote the set of all the possible level groups, and $GBottom_D$ a level group containing the bottom levels of each dimension in D. We also define a *base fact table* as a fact table with schema $(fname, GBottom_D, m)$.

2.2 Data Cubes

Several classes of aggregate views have been used to fulfill different requirements in OLAP systems. Gray et al [5] introduced the *data cube operator* as a shorthand for a set of cube views that contains data from a base fact table, aggregated over all the possible groups of attributes in it. We will extend it in order to to include views computing aggregates over the levels of the dimensions , and define the operator *CubeView*, to express them. We also assume that every set of dimensions contains a distinguished one, called $Measure$, with level m, over which aggregation takes place, and define an *aggregate function* Ag over a level m as a function having signature $Ag : 2^{dom(m)} \rightarrow dom(m)$.

Definition 4 (Cube View) *Given a set of dimensions D, a base fact table f_{base}, with measure m, and a level group $GB = \{l'_1, \ldots l'_n\}$; $CubeView(f_{base}, D, GB)$ yields a fact table f, with* schema $s = (fname, GB, m)$, *and* instance *defined as follows: given a point c, $S(c)$ is the set that contains points c' in the domain of f_{base}, such that for each pair of levels $l_1 \in GBottom_D$ and $l_2 \in GB$, belonging to the same dimension, we have $c(l_2) = RUP_{l_1}^{l_2}(c'(l_1))$, where $RUP_{l_1}^{l_2}$ is in the set of roll up functions RUP^* of that dimension. Then, for the points c such that $S(c) \neq \phi$, we have $f(c) = Ag(S(c))$. Notice that m and Ag are not parameters of $CubeView$, because they can be inferred from the context.*

Example 2 *Consider a set of dimensions* D= {Product,Store,Time,Sales}. *The schema and roll up functions for* Product *are given in figure 1(a). Schemas for* Time *and* Store *can be inferred from the rollup functions given below . For* Store *we have $RUP_{StoreId}^{Region} = \{s_1 \mapsto r_1, s_2 \mapsto r_2, s_3 \mapsto r_3\}$; for* Time *we have: $RUP_{Day}^{Week} = \{d_1 \mapsto w_1, d_2 \mapsto w_1, d_3 \mapsto w_2\}$ (we have omitted the roll ups to* All*). Let us denote the following fact table as* DailySales*:*

ItemId	StoreId	Day	Sales
i_1	s_1	d_1	10
i_2	s_1	d_1	20
i_2	s_2	d_1	20
i_2	s_2	d_2	40
i_3	s_3	d_3	30

Then, assuming that Ag is the SUM *aggregate function, and the measure dimension is* Sales *(with the measure level also named Sales), the fact table $CubeView(DailySales, D, \{ItemId, StoreId,$Week$\})$, call it* Sales_ISW*, is the following:*

ItemId	StoreId	*Week*	Sales
i_1	s_1	w_1	10
i_2	s_1	w_1	20
i_2	s_2	w_1	60
i_3	s_3	w_2	30

Definition 5 (Data Cube Operator) *Given a dimension set D, a base fact table f_{base} with measure m, and a set of level groups $GBSET$, the data cube operator, $DataCube(D, f_{base}, GBSET)$, gives a multidimensional database $(D, F \cup \{f_{base}\})$, such that for every level group GB in $GBSET$ there is one and only one fact table f in F, where $f = CubeView(f_{base}, D, GB)$.*

Example 3 *For the dimension set D and the base fact table* Daily_Sales *of example 2. we want to materialize views over the following set of level groups: $GBSET = \{\{ItemId, StoreId, Day, Sales\}, \{\{ItemId, StoreId, Week, Sales\}, \{Brand, StoreId, Week, Sales\}, \{Brand, All, All, Sales\}\}$.*
$DataCube(D, f_{base}, GBSET)$ will contain a base fact table Daily_sales*, and four aggregate views, which represent the sum of the measure Sales, grouped by each level group in $GBSET$.*

2.3 ROLAP Storage of a Multidimensional Database

In this subsection we will define how the multidimensional database is stored using the relational model, and denote this representation as a ROLAP database. We will not get into the details due to space available. In ROLAP terminology, D will be a set of dimension tables, and F, a set of fact tables (i.e. , we will use the same names as in section 2.1, confusions are avoided by the context).The ROLAP database that stores (D, F) will have a dimension table $d \in D$ for each dimension $d = ((dname, L, \preceq), RUP) \in D$, with name $dname$.The set of levels in d, excluding level All, will be its attributes, each attribute denoted as A_l. For every pair of levels, l_1, l_2 in L, where $l_1 \preceq l_2$, the functional dependency $l_1 \rightarrow l_2$ holds. Its instances will be such that for each $e \in instset(l_{inf})$ there exists a tuple t, s.t. $t.A_{l_{inf}} = e$, and if $e' \in instset(l)$, $RUP_{l_{inf}}^{l*}(e) = e'$, then $t.A_l = e'$.i Moreover, for every fact table f in F, with schema $(fname, Lset, m)$, we will have a relation f, with schema name $fname$, and attributes $Lset \cup \{m\}$. The functional dependency $Lset \rightarrow m$ holds.The fact table instance contains the tuples that represent the graph of the fact table.

Example 4 *Dimension* Product *in figure 1, is mapped to the table shown below. The functional dependencies $ItemId \to Brand$, $Brand \to Company$, $Company \to Coorporation$, $ItemId \to category$, and $Category \to Corporation$ hold.*

ItemId	Brand	Company	Category	Corporation
i_1	b_1	co_1	c_1	cr_1
i_2	b_2	co_1	c_1	cr_1
i_3	b_2	co_1	c_1	cr_1
i_4	b_3	co_2	c_2	cr_2

3 Dimension Updates

We will define a basic set of operators that will allow us to modify either the schema or an instance of a given dimension, classifying them into two subsets : Structural Update Operators and Instance Update Operators. Operators in the first set modify the structure of a dimension, while operators in the second one do the same with an instance of a dimension.

3.1 Structural Update Operators

The *Generalize* operator creates a new level, l_n, to which a pre-existent one, l, rolls up. A function f must be defined from the instance set of l, to the domain of l_n.

Operator 1 (Generalize) *Given a dimension $d = ((dname, L, \preceq), RUP)$, two levels $l \in L$, $l_n \notin L$, and a function $f_l^{l_n} : instset(l) \to dom(l_n)$,* Generalize($d, l, l_n, f_l^{l_n}$) *is a new dimension $((dname, L \cup \{l_n\}, \preceq \cup\{(l, l_n), (l_n, All)\} \setminus (l, All)), RUP')$, where RUP' is the set containing the roll up functions $RUP'^{l_j}_{l_i}$, such that: $RUP'^{l_n}_{l} = f_l^{l_n}$;$RUP'^{All}_{l_n} = \{(e, all) \mid e \in ran(f_l^{l_n})\}$; $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$, for all other levels l_i, l_j.*

The *Specialize* operator adds a new level l_n to a dimension. Level l_n will roll up to the lowest level of L, l_{inf}, becoming the lowest level of the dimension. Again, a function must be defined for this roll up.

Operator 2 (Specialize) *Given a dimension $d = ((dname, L, \preceq), RUP)$, a level $l_n \notin L$, and a function $f^{l}_{l_n} : dom(l_n) \to instset(l_{inf})$,* Specialize($d, l_n, f^{l_{inf}}_{l_n}$) *is a new dimension of the form $((dname, L \cup \{l_n\}, \preceq \cup\{(l_n, l_{inf})\}), RUP')$, where RUP' is the set containing the roll up functions $RUP'^{l_j}_{l_i}$, such that $RUP'^{l_{inf}}_{l_n} = f^{l_{inf}}_{l_n}$, and $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$ for all other levels l_i, l_j.*

Example 5 *Suppose we want to define a new level in the* Store *dimension, say, the type of store, with two possible values, t_1 and t_2. The operation would be defined as:* Generalize(Store, StoreId, Type, $f^{Type}_{StoreId}$)*, where $f^{Type}_{StoreId} = \{(s_1, t_1), (s_2, t_1), (s_3, t_2)\}$. See figure 2(a). Then, we specialize the* Time *dimension. The only level at which we can do this, is Day. The operator will be defined as :* Specialize(Time, Hour, f^{Hour}_{day})*. Figure 2(b) shows a limited portion of the dimension instance after specialization.*

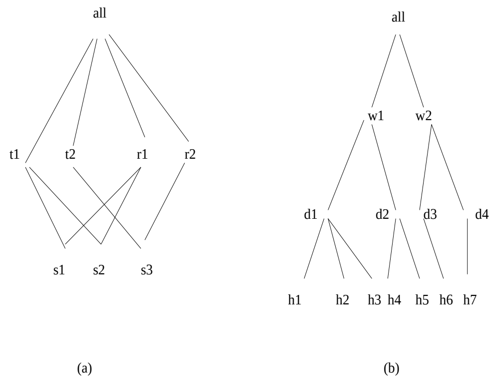

Figure 2: (a)Store dimension after Generalization (b) Time dimension after Specialization.

Before we introduce the next operator, let us define the following notation: we say that two levels are independent, denoted $l_a \parallel l_b$, when $l_a \npreceq'^* l_b$ and $l_b \npreceq'^* l_a$.

The *Relate* operator defines a roll up function between two independent levels belonging to a dimension. A necessary condition for this, is the existence of a function f between the instance sets of the levels being related, such that the dimension instance remains consistent. Otherwise, the *Relate* cannot be applied. Moreover, when conditions for relating two levels l_a and l_b are met, we must delete all the redundant roll up functions that may appear, which are not admitted in our model (we only include direct roll ups in it). For instance, we must delete the roll up functions between levels l and l_b such that $l \preceq^* l_a$ and $l \preceq l_b$.

Operator 3 (Relate Levels) *Given a dimension, say $d = ((dname, L, \preceq), RUP)$, a pair of levels $l_a \in L$, and $l_b \in L$, such that: $l_a \parallel l_b$, and the existence of a function $f^{l_b}_{l_a}$ between $instset(l_a)$ and $instset(l_b)$, defined as $f^{l_b}_{l_a} = \{i_a \mapsto i_b \mid \exists i \in l_{inf}, RUP^{*l_a}_{l_{inf}}(i) = i_a,\ RUP^{*l_b}_{l_{inf}}(i) = i_b\}$;* Relate($d, l_a, l_b$) *is the dimension $((dname, L, \preceq'), RUP')$, where $\preceq' = \preceq \cup\{(l_a, l_b)\} \setminus \{(l_i, l_b) | l_i \preceq^* l_a \wedge l_i \preceq l_b\} \setminus \{(l_a, l_k) | l_a \preceq l_k \wedge l_b \preceq^* l_k\}$, and $RUP'^{l_b}_{l_a} = f^{l_b}_{l_a}$, and $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$, for all other levels l_i, l_j.*

Note that we do not need to specify $f^{l_b}_{l_a}$ in the definition of the operator, because it is uniquely determined by

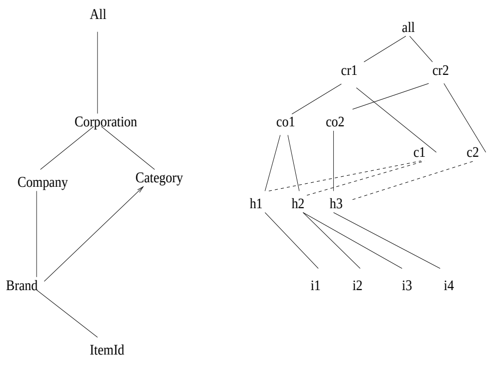

Figure 3: Dimension Product before and after an application of Relate Levels.

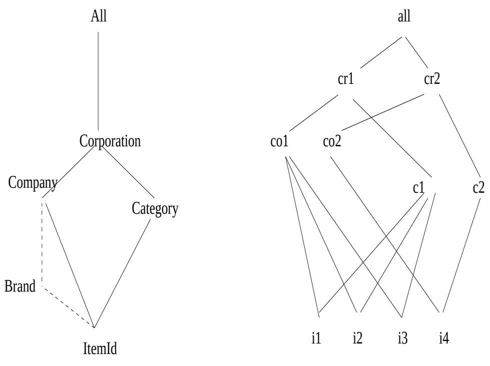

Figure 4: An example of Delete level.

the pre-existent roll up functions.

The *Unrelate* operator deletes a relation $\preceq$ between two levels l_a and l_b, such that $l_a \preceq l_b$. The operator must guarantee that levels which are below than l_a in the hierarchy, will still be able to reach the same levels they reached before the unrelate operation. For instance, if $l_a \preceq l_b$ and $l_b \preceq l_c$, we must preserve $l_a \preceq l_c$ by making it explicit, in case we delete $l_a \preceq l_b$, because, again, this relation was only implicit in the model.

Operator 4 (Unrelate Levels) *Given a dimension $d = ((dname, L, \preceq), RUP)$, and two levels, $l_a \in L$ and $l_b \in L$, such that $l_a \preceq l_b$,* Unrelate(d, l_a, l_b) *is a new dimension $((dname, L, \preceq'), RUP')$, where $\preceq' = \preceq \setminus \{(l_a, l_b)\} \cup \{(l_i, l_b) | l_i \preceq l_a \wedge l_i \not\preceq^*_{ab} l_b\} \cup \{(l_a, l_j) | l_b \preceq l_j \wedge l_a \not\preceq^*_{ab} l_j\}$, $\preceq_{ab} = \preceq \setminus \{(l_a, l_b)\}$* [1] *, and $RUP'^{l_j}_{l_a} = RUP^{l_b}_{l_a} \circ RUP^{l_j}_{l_b}$, if $l_b \preceq l_j$; $RUP'^{l_b}_{l_i} = RUP^{l_a}_{l_i} \circ RUP^{l_b}_{l_a}$, if $l_i \preceq l_a$; $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$, for all other levels l_i, l_j.*

Example 6 *Operation* Relate(Product, Brand, Category) *will relate level Brand with level Category (figure 3). Notice that relation ItemId $\preceq$ Category was deleted as a colateral action. After this, we unrelate levels Category and Corporation by means of* Unrelate(Product, Category, Corporation). *We must add to $\preceq$, the relation Category $\preceq$ All. Otherwise, level Category would not be able to reach level* All. *Note also that Brand $\preceq$ All is not required because All is reached from Brand in $\preceq_{CategoryCorporation}$.*

The *DelLevel* operator deletes a level and its roll up functions. The level to be deleted cannot be the lowest one in the dimension (l_{inf}), unless it rolls up to only one higher level(for instance, we could delete level "hour" in the schema corresponding to the instance of figure 2(b), but we could not delete level ItemId in figure 1). As it was the case with the *Relate* operator, taking into account that we only define the direct roll ups, when deleting a level we must add the functions between levels above and below it.

Operator 5 (Delete Level) *Given a dimension $d = ((dname, L, \preceq), RUP)$ and a level $l \in L, l \neq All$, such that, if $l = l_{inf}$, then there is only one level l_j such that $l \preceq l_j$,* DelLevel(d, l) *is a new dimension $((dname, L' = L \setminus \{l\}, \preceq'), RUP')$, where $\preceq' = \preceq \setminus \{(l_1, l_2) \mid (l_1 = l) \vee (l_2 = l)\} \bigcup \{(l_1, l_2) \mid (l_1 \preceq l) \wedge (l \preceq l_2) \wedge (l_1 \not\preceq^* l_2\}$*[2]*, and $RUP'^{l_j}_{l_i} = RUP^{l}_{l_i} \circ RUP^{l_j}_{l}$, if $l_i \preceq l$ and $l \preceq l_j$, or $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$ otherwise.*

Example 7 *Let us suppose we want to delete level Brand from the* Product *dimension. We would obtain the schema and instance depicted in figure 4.*

3.2 Instance Update Operators

The following operators add or delete instances to and from a level in a dimension. They also impose some constraints on the way these updates can be performed.

The *AddInstance* operator inserts a new element, say x, into a level l_a(i.e., an element not belonging to the instance set of l_a). We must provide the operator with the pairs (l_i, x_i), such that every l_i is a level to which l_a directly rolls up($l_a \preceq l_i$), and $RUP^{l_i}_{l_a}(x) = x_i$).

Operator 6 (Add Instance)
Given a dimension $d = ((dname, L, \preceq), RUP)$, a level l_a, an element $x_a \in dom(l_a)$, $x_a \notin instset(l_a)$, and a set of pairs $P = \{(l_1, x_1), \ldots (l_n, x_n)\}$, where l_k is a level, and x_k is an element, and:(a) for each $(l_k, x_k) \in P$, $x_k \in instset(l_k)$,(b) $dom(P) = \{l_k \mid l_a \preceq$

[1] The last conditions prevent the addition of the arcs in case alternative paths between (l_i, l_b) or (l_a, l_j) (that is, paths not including $l_a l_b$ as a subpath) existed in $\preceq$.

[2] this last condition, like in the former operator, implies that no alternative paths between (l_1, l_2) exist.

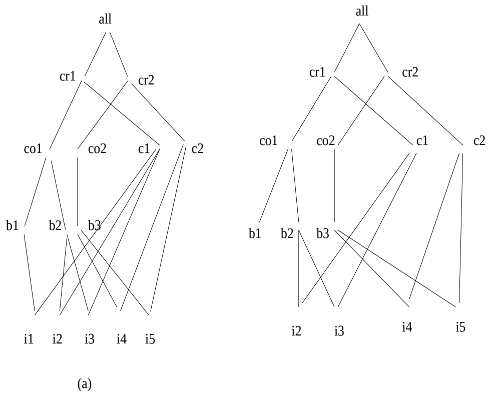

Figure 5: (a)Add instance (b) Delete instance.

$l_k\}$, and (c) for each pair $(l_k, x_k), (l_s, x_s) \in P$ such that exists a level $l \in L, l_k \preceq l$ and $l_s \preceq l$: $RUP^{l}_{l_k}(x_k) = RUP^{l}_{l_s}(x_s)$, $\mathrm{AddInstance}(d, l, x_a, P)$ is a new dimension $((dname, L, \preceq), RUP')$, where RUP' is the set containing the roll up functions $RUP'^{l_j}_{l_i}$, such that $RUP'^{l_j}_{l_a} = RUP^{l_j}_{l_a} \cup \{(x_a, x)\}$, if $(l_j, x) \in P$; $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$, for all other levels l_i, l_j.

The *DelInstance* operator deletes an element belonging to the instance set of a level l_a. It is only defined when no element of any level l_i, such that $l_i \preceq l_a$, rolls up to the element being deleted.

Operator 7 (Delete Instance) *Given a dimension $d = ((dname, L, \preceq), RUP)$, a level $l_a \in L$, and an element $x_a \in instset(l_a), x_a \notin \bigcup_{l \in L} ran(RUP^{l_a}_{l})$,* DelInstance$(d, l_a, x_a)$ *is the dimension $((dname, L, \preceq)$, $RUP',)$ where RUP' is the set containing the roll up functions $RUP'^{l_j}_{l_i}$, such that $RUP'^{l_j}_{l_a} = RUP^{l_j}_{l_a} \setminus \{(x_a, RUP^{l_j}_{l_a}(x_a))\}$; $RUP'^{l_j}_{l_i} = RUP^{l_j}_{l_i}$, for all other levels l_i, l_j.*

Example 8 *Suppose we want to add a new item, say i_5, to an instance of the* Product *dimension, by means of :* AddInstance (Product ,*ItemId*, $i_5, \{(Brand, b_3)$, $Category, c_2)\})$. *Figure 5(a) shows the result. After that, we delete i_1 from level* ItemId. *Note that we can only, in this case, delete an element belonging to* instset(ItemId), *due to the operator preconditions. The operation is invoked as* DelInstance(Product, ItemId, i_1). *See figure 5(b).*

4 Maintenance

In this section we treat the evolution of the data cube under dimension updates, addressing separately structural and instance updates. Due to space limitations, all proofs will be omitted.

In order to specify the maintenance algorithms, we will use the relational algebra with bag semantics, extended with the *generalized projection operator* to express aggregation([7]). The generalized projection operator will be denoted as Π_A, where the projected attributes can include aggregate functions. Then, computing $CubeView(D, f_{base}, GB)$ is equivalent to computing the relation: $\Pi_{GB \setminus \{All\}, Ag(m)} f_{base} \bowtie d_1 \ldots \bowtie d_n$, where $d_1, \ldots, d_n$ are the dimensions of the levels that are in $GB \setminus \{All\}$ but not in $GBottom_D$. For notation, when using the generalized projection, we will refer to $GB \setminus \{All\}$ just as GB. In this way, the cube view of example 2 can be expressed as: $\Pi_{ItemId, StoreId, Week, Sum(Sales)} Daily\_Sales \bowtie Time$

4.1 Structural Updates

When a dimension structural update occurs, we must guarantee that the new data cube represents the aggregation of the base fact table in the new dimension set. However, when a bottom level of some dimension is deleted, or the dimension is specialized, the base fact table of the new set is no longer the same as the base fact table of the original set.

To handle this problem, we propose the following data cube adaptation: if a *DelLevel* of a bottom is specified, recompute f_{base} as the cube view of the old base fact table grouped by the bottom level group of the new dimension set D'. If the update is *Specialize*, f_{base} will not longer be a base fact table of D', and the new base fact table over D' of which f_{base} is a cube view is not uniquely determined. We will not address in this paper the problem of finding an appropriate new base fact table, but it can be seen that one always exists. The determination of a new base fact table is related to the *reconstruction problem* [4], which consists on the estimation and computation of a fact table based on an aggregate view of it.

We can summarize our simple incremental maintenance algorithm for structural updates as follows. For the operators *Relate, UnRelate*, and *Generalize*, no maintenance is required. If the update is $DelLevel(l, i)$: (a) if $l \in GBottom_D$, compute a new base fact table as $\Pi_{GBottom_{D'}, Ag(m)} f_{base} \bowtie d$(i.e., perform data cube adaptation, if needed) (b) delete every table f in F s.t. l belongs to its schema. Finally, if the update is $Specialize(d, l_0, RUP^{l_0}_{l_{inf}})$, compute a fact table f'_{base} such that $f_{base} = \Pi_{GBottom_D, Ag(m)} f'_{base} d'$, and drop f_{base}.

4.2 Instance Updates

Assuming a ROLAP storage of the data cube, an instance update reduces to the insertion and/or deletion of some tuples in the dimension tables. We could therefore apply existing incremental maintenance algorithms for materialized relational views with aggregates [1, 15, 14]. However, we will see that we can do better by exploiting the special form of these updates. In particular, we will show how to improve on the "summary delta" method of Mumick et al. [14].

In general, incremental maintenance involves (1) computing the set of changes, sometimes called the delta table(*propagation phase*), and (2) applying the changes represented in the delta table to the materialized view(*refresh phase*). In the case of views with aggregations, this approach applies only if the aggregate functions are *self maintainable* [13]. We will assume in what follows that the aggregate operator Ag is *SUM*, and that we extend the tables and the cube views to store the *count* value required to make *SUM* self-maintainable with respect to insertions and deletions.

In what follows, we will use a reduced version of the data cube of Example 3.

Example 9 *To show the ideas depicted in this section, we will be considering the set of dimensions* D={Product,Store, Time,Sales}*, with schemas specified as follows:* Product *has hierarchy: $ItemId \preceq Brand \preceq All$;* Store *has hierarchy: $StoreId \preceq All$; and* Time *has hierarchy: $Day \preceq Week \preceq All$. The roll up functions are the following (the roll ups to* All *are omitted, where possible): $RUP_{ItemId}^{Brand} = \{i_1 \mapsto b_1, i_2 \mapsto b_2, i_3 \mapsto b_2, i_4 \mapsto b_4\}$; $RUP_{StoreId}^{All} = \{s_1 \mapsto all, s_2 \mapsto all\}$; $RUP_{Day}^{Week} = \{d_1 \mapsto w_1, d_2 \mapsto w_1, d_3 \mapsto w_2\}$. We will use the same base fact table of example 2, called* DailySales*, so we will not repeat it here. The data cube is* dc=DataCube(D,f_{base},GBSET)*, and each cube view will be denoted as "*Sales*" appended with the initials of the levels in the group level defining it (e.g.* Sales_ASD *represents the cube view where $GB = \{All, StoreId, Day\}$).*

4.2.1 Maintaining Cube Views Independently

Under instance updates in a dimension, we can proceed as in the summary delta method, i.e., separately compute for each cube view a maintenance expression, derived using the rules presented by Quass [15]. The maintenance expression produces the change to each cube view, which is then applied to the cube view using a *refresh algorithm*. Let $Refresh(f, \Delta f)$ be the result of applying the refresh algorithm for set of changes Δf to fact table f. Given a cube view $f = CubeView(D, f_{base}, GB)$, and an instance update u that updates cube view f to cube view $f' = CubeView(D', f_{base}, GB)$, we will denote by Δf the change to the cube view f wrt the update, that is, Δf is a fact table such that $f' = Refresh(f, \Delta f)$.

Example 10 *Consider the data cube of example 3, and an update* DelInstance(Product,ItemId,i_2) *in the* Product *dimension, represented by $\Delta^{-}Product$. The maintenance expression for the cube view $Sales\_BSW{=}CubeView(D, f_{base}, \{Brand, Store, Week\})$, would be $\Delta Sales\_BSW{=}\Pi_{Brand,StoreId,Week,-SUM(Sales)}$ $Daily\_Sales \bowtie \Delta^{-}Product \bowtie Time$. The new cube view, $CubeView(D', Daily\_Sales, \{Company, Store, Week\})$, is computed, in the refreshing stage, from Sales_BSW and the delta change of f, using the refreshing algorithm, call it $Refresh(f, \Delta f)$. Thus, $Sales\_BSW'{=}Refresh(Sales\_BSW, \Delta Sales\_BSW)$. This is done separately for every cube view in dc.*

Now, we can say that if the instance update is over a level that is not the bottom level, any cube view change is empty. These updates are called *irrelevant updates* [8] in the view maintenance literature. Formally, given a dimension d, a dimension update, $DelInstance(d, l, i)$ or $AddInstance(d, l, i, P)$ is irrelevant (i.e., the dimension changes do not have an effect in any cube view) if l is not the bottom level of d, or $\sigma_{l=i} f_{base} = \phi$ Thus, no maintenance is required in case the update is performed over a level that is not the bottom of the dimension. Furthermore, if the update is relevant, the delta changes that have an effect over the cube views are: $\Delta^{+}d = \sigma_{l=i} d'$, in case of an $AddInstance$, and $\Delta^{-}d = \sigma_{l=i} d$, when a $DelInstance$ takes place. Both of these contain exactly one tuple, which we will call $\delta^{+}d$ and $\delta^{-}d$, respectively. In Figure 6 we give a set of rules to derive maintenance expressions for instance updates. Note that if rule 1 holds, no aggregation is needed.

Thus, the incremental maintenance algorithm can be stated as follows: check if the updates are irrelevant. If not, using the rules in figure 6, derive the maintenance expression to compute the changes for every cube view, and then compute the updated cube view using the refresh operator.

Example 11 *Consider the retail data cube of example 9 and the update* DelInstance(Product,ItemId,i_2)*. Using rule 1, we can compute the delta change associated to a cube view* Delta Sales_BSD*, from the fact table* DailySales *(see example 2), with the following expression, avoiding any aggregate computation. The resulting table is also shown.*

Given a instance update u, and a cube view $f = CubeView(D, f_{base}, GB)$

1 If $u = DelInstance(d, l, i)$, and $GBottom_D \setminus GB = \{l\}$, then:
$\Delta f = \Pi_{GB \setminus \{l'\}, l'=\delta^- d(l') m=-m} \sigma_{l=i} f_{base}$, where $l' = GB \setminus GBottom_D$
else
$\Delta f = \Pi_{GB \setminus \{l'\}, l'=\delta^- d(l'), m=-Ag(m)} \sigma_{l=i} f_{base} \bowtie d_1 \bowtie \ldots \bowtie d_n$, where l' is the level in GB that belongs to d, and $d_1 \ldots d_n$ are the dimensions such that the levels in $GB \setminus \{d\} \setminus GBottom_D$, belong to.

2 If $u = AddInstance(d, l, i, P)$ then
$\Delta f = \Pi_{GB \setminus \{l'\}, l'=\delta^+ d(l), m=Ag(m)} \sigma_{l=i} f_{base} \bowtie d_1 \bowtie \ldots \bowtie d_n$, where l' is the level in GB that belongs to d, and $d_1 \ldots d_n$ are the dimensions such that the levels in $GB \setminus \{d\} \setminus GBottom_D$, belong to.

Figure 6: Maintenance Expressions for instance updates.

$\Delta Sales\_BSD = \Pi_{Brand=b_2, StoreId, Day, Sales=-Sales} \sigma_{ItemId=i_2}\ DailySales$.

Brand	StoreId	Day	Sales
b_2	s_1	d_1	-20
b_2	s_2	d_1	-20
b_2	s_2	d_2	-40

4.2.2 Maintenance Using the View Derived Lattice

Because we have a set of related views, and, as we will see, the set of cube view changes are also related, we can take advantage of previous cube view changes to compute other ones which depend on them. We will present an efficient algorithm for the cases of instance updates, by adapting the view lattice introduced by Harinarayan et al. [10], and the summary delta method, to our specific needs.

The following data cube contains the cube view changes caused by our dimension update: Let $\Delta f_{base} = \Pi_{GBottom_D, m=-m} \sigma_{l=i} f_{base}$, if the update is $DelInstance(d, l, i)$, and $\Delta f_{base} = \sigma_{l=i} f_{base}$ if the update is $AddInstance(d, l, i, P)$. Then the delta cube, $\Delta dc = DataCube(D \setminus \{d\} \cup \Delta d,\ \Delta f_{base}, GBSET)$, where $\Delta d = \delta^- d = \sigma_{l=i} d$ for $DelInstance$, and $\Delta d = \delta^+ d = \sigma_{l=i} d'$ for an $AddInstance$.

The data cube defined in this way contains the cube view changes to the data cube $(D, f_{base}, GBSET)$. We now present an algorithm to compute this cube efficiently. We assume that $GBSET = GB_D$, i.e., the data cube is totally materialized; the algorithm that we will propose can be extended to a partially materialized data cube.

We define the view lattice as follows: there is a node, denoted $N(GB)$, for each level group GB in GB_D. The edges are defined as follows. If there is a 1-1 function co between two level groups GB_1 and GB_2, such that for each level l in GB_1, $l \preceq co(l)$, then: there is an edge from $N(GB_1)$ to $N(GB_2)$, and this edge is labeled by the dimensions $d_1, \ldots, d_n$ of each level l in GB_1 such that $l \neq co(l)$. We denote this edge by $N(GB_1) \preceq_{d_1,\ldots,d_n} N(GB_2)$. Finally, we associate with each edge $N(GB_1) \preceq_{d_1 \ldots d_n} N(GB_2)$ a bag algebra expression to compute the cube view change Δf_2 associated to GB_2 from the cube view change Δf_1 associated to GB_1: $\Delta f_2 = \Pi_{GB_2, Ag(m)} \Delta f_1 \bowtie d_1 \bowtie \ldots \bowtie d_n$.

We now present a set of rules that, given an update to level l of dimension d, modify the algebra expressions associated with the edges of the view lattice. We only consider "direct" edges, that is, those edges between sets GB_1 and GB_2 that differ only on one level.

RULE A For the direct edges $N(GB_1) \preceq_d N(GB_2)$ such that $l \in GB_1$, $GB_1 \setminus GB_2 = \{l\}$, $\Delta f_2 = \Pi_{GB_2 \setminus \{l'\}, l_1=\delta d(l_1), \ldots l_n=\delta d(l_n), m} \Delta f_1$, where δd is $\delta^+ d$, if the update is $AddInstance(d, l, i, P)$ or $\delta^- d$ if it is $DelInstance(d, l, i)$; l_i are the levels in d, and $l' = GB_2 \setminus GB_1$.

RULE B For the direct edges $N(GB_1) \preceq_d N(GB_2)$ s.t. $l \notin GB_1$ and $GB_1 \setminus GB_2 = \{l'\}, l' \in d$, $\Delta f_2 = \Delta f_1$.

RULE C The remaining direct edges, of the form $N(GB_1) \preceq_{d_i} N(GB_2)$, are modified in the following way : For each level $l_i \in d$, consider the nodes $N(GB)$ such that $l_i \in GB$, and the edges between these nodes. These edges form a sub-lattice of the view lattice (see Figure 7). Let τ be a path from the bottom level of this sub-lattice s.t. $N(GB_1) \preceq_{d_i} N(GB_2) \in \tau$. If there is an edge of the form $\preceq_{d_i}$ in τ before τ reaches $N(GB_2)$, then $\Delta f_2 = \Pi_{GB_2 \cup A_{d_i}, Ag(m)} \Delta f_1$, where A_{d_i} represents the attributes of the dimension d_i. If not (i.e., this is the first occurrence of d_i in the path), then $\Delta f_2 = \Pi_{GB_2 \cup A_{d_i}, Ag(m)} \Delta f_1 \bowtie d_i$.

Example 12 *An optimized lattice for our running example is depicted in figure 7. The dashed lines represent the derived expressions changed using rule A; the dotted lines represent the derived expressions changed using rule B; the solid lines represent the derived expressions changed with rule C. Here, the edges are labeled with a dimension name, when the join with the dimension is included in the expression.*

The following algorithm generates a plan that leads to reduce the number of joins and aggregate computations while performing view maintenance due to instance updates.

Algorithm 1
Input*: A data cube dc, its derived lattice, and an instance*

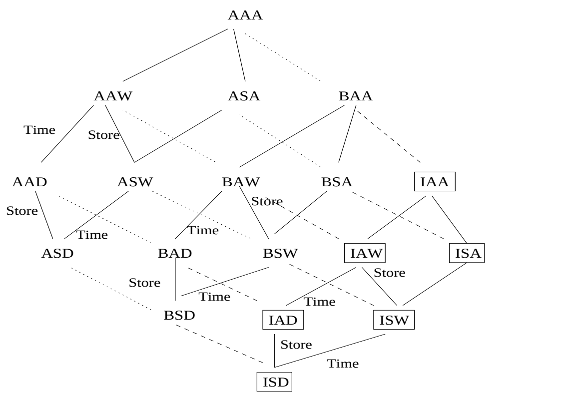

Figure 7: An optimized view lattice for the running example.

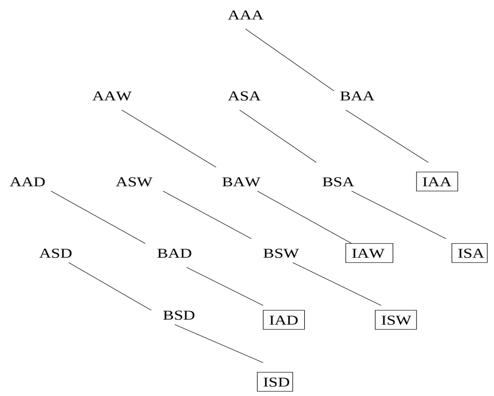

Figure 8: The plan generated for update $DelInstance(Product, Item, i_2)$.

update **u***.*
Output: *A plan (a subgraph of the optimized derived lattice), that chooses one direct predecessor for each node.*

- *For each node $N(GB)$ such that $l \in GB$:*
  case u = *$DelInstance(d, l, i)$, the view change will be: $\Delta f = \Pi_{GB, l_1=\delta d(l_1), \ldots l_n=\delta d(l_n), m=-m} \sigma_{l=i} f.$, where l_i are the levels of d, s.t. $l_i \neq l$.*
  case u = *$AddInstance(d, l, i, P)$, choose a node GB', immediately below GB, using one of the well-known methods for computing aggregates (v.g. [2]). As a default, the predecessor with the least estimated size could be used ([2]).*
- *For each node $N(GB)$ such that $l \notin GB$: choose as predecessor the node $N(GB')$, immediately below (i.e., connected by a direct edge), s.t. that $GB' \setminus GB = \{l'\}, l' \in d$.*

The strength of Algorithm 1 resides in that only in the case of an *AddInstance*, and for the nodes $N(GB)$ such that l is in GB, the cube view is computed using derived expressions containing aggregations and joins. For instance, a totally materialized data cube with n dimensions, each one having k_i levels, would require, using the summary delta method, computing $\Pi_{i=1}^{n} k_i$ aggregates and n joins with the dimension tables, in case of a *DelInstance* operation. Using our method, neither aggregates nor joins must be computed. This can be seen in the following example.

Example 13 *Figure 8 shows the graph for the plan derived for the data cube of example 9, for a $DelInstance(Product, ItemId, i_2)$ update. The nodes inside a box represent the cube views such that $l \in GB$. The sub-lattice indicates which predecessors must be chosen. We can proceed analogously for the case of an AddInstance operation.*

5 Discussion

As dimensions represent the framework within which factual data is summarized for analysis, it is our belief that changes in analysis requirements and/or in the structure of the data sources, almost always imply changes in the dimensions of the model. These changes are not limited to the addition or deletion of attributes, but they may also involve the hierarchical structure according to which dimensions are organized. All these kinds of dimension updates, together with the data cube maintenance under them, are poorly supported (or not supported at all) in current commercial systems. Our focus in this paper has been to introduce the problem and present a framework for it.

Several multidimensional models have already been presented [9, 12, 2]. We chose the work by Cabibbo and Torlone [3] as a starting point for our model. We developed a set of operators for structural and instance updates over dimensions. We also presented algorithms to perform maintenance when using ROLAP storage of the data cube, under both structural and instance updates.

Mumick et al [14], proposed the *Summary Delta Method* for incremental maintenance of a set of materialized aggregate views defined over the same base table. Their approach is applied to *generalized cube views*, which differ from our cube views in their aggregate functions and the joins they perform with dimension tables. This work focuses mostly on updates to the fact table, but has a brief discussion of instance updates to the dimension tables. In the framework presented there, maintenance of the cube views is performed separately for each derived table, without taking into account the view lattice. We devised instead an algorithm that takes advantage of the view lattice to prevent, whenever possible, joins and aggregate computations.

In future work we will give a more in-depth treatment of the update operators, which we are in the process of implementing. We are also defining higher level operators, such as re-classifying a level within a dimension hierarchy, and maintenance algorithms for them.

Acknowledgments

This work was partially supported by the Institute of Robotics and Intelligent Systems, and by the University of Buenos Aires.

References

[1] Gupta A., I.S. Mumick, and D Subrahmanian. Maintaining views incrementally. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Washington D.C.,USA, 1993.

[2] R. Agrawal, A. Gupta, S. Sarawagi, P Deshpande, S. Agarwal, J. Naughton, and R. Ramakrishnan. On the computation of multidimensional aggregates. In *Proceedings of the 22nd VLDB Conference*, Bombay, India, 1996.

[3] L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Proceedings of the 6th International Workshop on Database Programming Languages*, pages 253–269, East Park, Colorado, USA, 1997.

[4] C. Faloutsos, H. Jagadish, and N. Sidiropoulos. Recovering information from summary data. In *Proceedings of the 23rd VLDB Conference*, Athens, Greece, 1997.

[5] J. Gray, A. Bosworth, A. Layman, and H. H. Pirahesh. Data cube : A relational operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery 1, pgs. 29-53*, 1997.

[6] A. Gupta, V. Harinarayan, and D Quass. Aggregate query processing in data warehousing environments. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995.

[7] A. Gupta, V. Harinarayan, and D Quass. Generalized projections:a powerful approach to aggregation. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995.

[8] A. Gupta and I.S. Mumick. Maintenance of generalized views:problems, techniques and applications. In *Bulletin of the Technical Committee on Data Engineering, Vol 18, No 2*, 1995.

[9] M. Gyssens and L. Lakshmanan. A foundation for multi-dimensional databases. In *Proceedings of the 22nd VLDB Conference*, pages 106–115, Bombay, India, 1996.

[10] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM-SIGMOD Conference*, pages 205 – 216, Montreal, Canada, 1996.

[11] R. Kimball. *The Data Warehouse Toolkit.* J.Wiley and Sons, Inc, 1996.

[12] C. Li and S. Wang. A data model for supporting online analytical processing. In *Proceedings of the Conference on Information and Knowledge Management*, pages 81–88, 1996.

[13] D. Lomet and Editors Widom, J. *IEEE Data Engineering Bulletin. Special Issue on Materialized Views and Data Warehousing*, June 1995.

[14] I. Mumick, D. Quass, and B. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *Proceedings of the ACM - SIGMOD Conference*, Tucson, Arizona, 1997.

[15] D. Quass. Maintenance expressions for views with aggregations. In *ACM Workshop on Materialized Views: Techniques and Applications*, Montreal, Canada, 1996.