A Data Model for Supporting On-Line Analytical Processing*

Chang Li
George Mason University
cli@isse.gmu.edu

X. Sean Wang
George Mason University
xywang@isse.gmu.edu

Abstract

A database application, called "on-line analytical processing" (or OLAP) and aimed at providing business intelligence through on-line multidimensional data analysis, has become increasingly important due to the existence of huge amounts of on-line data. This paper formalizes a multidimensional data (MDD) model for OLAP, and develops an algebraic query language called *grouping algebra*. The basic component of the MDD model is a multidimensional cube, consisting of a number of relations (called *dimensions*) and for each combination of tuples (called a *coordinate*), one from each dimension, there is an associated data value. Each dimension is viewed as a basic grouping, i.e., each tuple in the dimension corresponds to the group consisting of all the coordinates that contain this tuple. In order to express user queries, relational algebra expressions are then extended to those on basic groupings for obtaining complex groupings, including order-oriented groupings (for expressing, e.g., cumulative sum). The paper then considers the environment where the multidimensional cubes are materialized views derived from base data situated at remote sites. A multidimensional cube algebra is introduced in order to facilitate the data derivation. The purpose of the paper is to establish a formal foundation for further research regarding database support for OLAP applications.

1 Introduction

Recently, database system supports for business data analysis have become popular [Ban95]. This trend is evident from the popularity of many on-line analytical processing (OLAP) systems [CCS93, Dre93] such as Essbase by Arbor Software and Express by Oracle. Based on a multidimensional conceptual view of data, these systems are specially designed for data analysis, with characteristics significantly different from those of relational databases. The emphasis of OLAP systems is on flexible data grouping and efficient aggregation evaluation on obtained groups. The purpose of this paper is to develop a multidimensional data model for OLAP in order to establish a formal foundation for OLAP applications.

An OLAP system concerns mostly with intuitive user interface and quick on-line processing. The language used by an OLAP application must be able to flexibly express various grouping methods, including grouping by attributes, by positions in (time or other) sequences and by aggregation results on some other groups. Furthermore, most OLAP systems are based on the concept of a data warehouse, storing materialized views derived from base data that are (or used to be) located at remote sites. This brings in a new aspect on optimization, especially the view maintenance problem.

In this paper, we define a formal multidimensional data (MDD) model for OLAP systems. The central component of an MDD model is the notion of a multidimensional cube, where each dimension consists of a relation and there is a data value associated with each combination (called a coordinate) of the tuples, one from each dimension. A cube is similar to the concept of category in most of the SSDB data models in the literature (e.g., [CM89]).

In our model, a multidimensional database (MDDB) consists of a finite set of multidimensional cubes and a finite set of relations. OLAP queries are posed on the MDDB using a grouping algebra, an extension of the relational algebra. The grouping algebra is based on the observation that a tuple on a dimension of a multidimensional cube can serve as a pointer pointing to all the data values that are associated with all the coordinates containing that tuple. Therefore, a dimension provides a basic grouping method. We then use relational (and other) operations to manipulate these basic groupings to obtain complex ones, providing a powerful mechanism for analyzing data.

A novel feature of the grouping algebra is its inclusion of order-related operations. Order plays an important role in business data analysis. Examples include retrieving top n elements from a sequence and grouping over a time sequence (e.g., cumulative sum). The grouping algebra uses the notion of a tuple order to provide an ordered view of relations, based on which order-related grouping and selection are defined.

We also study OLAP systems in a data warehouse environment. We define a data warehouse as a set of

*Part of research supported by the NSF research initiation award IRI-9409769.

materialized cubes and relations derived from base data. In order to facilitate such a derivation, we introduce a cube algebra for manipulating cubes and defining cube views.

The rest of the paper is organized into 7 sections. A motivating example (also used as a running example) is given in Section 2. In Section 3, multidimensional cubes and grouping relations are defined. The grouping algebra is introduced in Section 4, and MDDB and MDDB queries are discussed in Section 5. In Section 6, the multidimensional cube algebra is presented. Related research is discussed in Section 7 and the paper is concluded with Section 8.

2 Motivating example

Consider a nation-wide department chain store that owns member stores located in different geographical regions. Assume that all the member stores in the chain sell the same products. Sales data are collected daily. That is, at the end of each day, each member store will report the total sales amount of each product to the national headquarters. Figure 1 shows a part of the data reported on May 15, 1996. The collected data may be used by the executives at the national headquarters to analyze the activities of the chain store.

	Store	Product		Date			
	Loc	Item	Manu	y	m	d	Sales
1	NewYork	Mac	Apple	1996	May	15	7000
2	Washington	Mac	Apple	1996	May	15	3800
3	LosAngeles	Mac	Apple	1996	May	15	15500
4	NewYork	PC	IBM	1996	May	15	2700
5	Washington	PC	IBM	1996	May	15	3900
6	LosAngeles	PC	IBM	1996	May	15	6530

Figure 1: Data of a chain store.

Example 1 One indicator of the activities of the chain store may be the growth trend (in sales) of each member store. These growth trends can be obtained via the following query:

(a) *For each member store, find out the year-to-date total sales amounts for each day of this year (i.e., the daily cumulative sales amounts over this year).*

Another indicator may be the current behavior (in sales) of the last year's top selling products:

(b) *Find the year-to-date total sales amounts, in each region (east, west, etc.), of each product whose last year's nation-wide total sales ranked among top five.* □

A critical step towards database support for data analysis involves formulating "analysis-oriented" queries as those given in Example 1. A central requirement for expressing these queries is the flexible representation of complex data grouping methods. Consider query (a). Two grouping methods are involved. One is by attributes (i.e., by store) and the other by position in a time sequence (i.e., by day). Specifically, for a given member store s and day d, the corresponding group consists of all the sales amounts that are reported by store s in this year up to day d. For query (b), two rounds of grouping are performed. The first is grouping by attributes (i.e., by product). That is, all the sales data (in last year) of a product form a group. A summation is performed on each group. The results are then used to select the top-five products. The second round uses these computation results. Specifically, for each region r and each of the five products, a group is formed by collecting all its sales amounts reported this year by all the stores in region r.

It is easily seen that the queries in Example 1 cannot straightforwardly be expressed by using the traditional relational query languages such as the relational algebra.

3 Cubes and grouping relations

In this section, we define n-dimensional cubes. Throughout of the paper, we assume two disjoint sets: a set of attribute names and a set of dimension names. We use A, B etc. to denote attribute names, R to denote finite sets of attribute names, and $\mathtt{D}$ to denote dimension names. Each symbol may be subscripted to denote a different name of the same kind. Each attribute A is associated with a domain $\mathsf{dom}(A)$. Furthermore, we use $\mathcal{V}$ to denote a set of scalar values (integers, reals, etc) which includes a special null value.

Definition Let n be a positive integer. An *n-dimensional cube scheme* is a set $\{(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\}$, where $\mathtt{D}_1, \ldots, \mathtt{D}_n$ are distinct dimension names and $R_1, \ldots, R_n$ are sets of attribute names. An *n-dimensional cube*, or a *cube*, on the scheme $\{(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\}$ is a pair (F, μ), where (i) $F = \{(\mathtt{D}_1, r_1), \ldots, (\mathtt{D}_n, r_n)\}$ with r_i being a relation on R_i for each $1 \leq i \leq n$, and (ii) μ is a mapping from $\{\{(\mathtt{D}_1, t_1), \cdots, (\mathtt{D}_n, t_n)\} \mid \forall 1 \leq i \leq n : t_i \in r_i\}$ to $\mathcal{V}$.

In the above definition, r_i is called the *dimension relation* for $\mathtt{D}_i$. Thus, a multidimensional cube consists of a set of dimension relations and a value mapping. The value mapping maps each combination of tuples, one from each dimension relation, to a scalar value. A special case in the above definition is that one (or more) dimension relation is empty. In this case, the value mapping is undefined.

Note that unlike an ordinary array, we use names to label different dimensions and there is no order among these dimensions. Operations like "transpose" are of no meaning to a cube. However, for convenience and when no confusion arises, we will assume that there is an order among the dimension names in a cube. That is, we will assume that a cube scheme is a list $\langle(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\rangle$, and we will write a cube on the scheme $\langle(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\rangle$ as $(r_1, \ldots, r_n, \mu)$, assuming that r_i is a relation on R_i for each $1 \leq i \leq n$, and that μ is a mapping from $\{(t_1, \ldots, t_n) \mid t_i \in r_i \text{ for each } 1 \leq i \leq n\}$ to the set $\mathcal{V}$ of scalar values. Each $(t_1, \ldots, t_n)$ here is called a *coordinate*.

Example 2 Consider the chain store example we gave in Example 1. A 3-dimensional cube can be used to formalize the sales of the chain store: $\mathtt{SALES} = (r_\mathtt{s}, r_\mathtt{p}, r_\mathtt{d}, amount)$ on the cube scheme $\langle(\mathtt{Store}, R_\mathtt{s}), (\mathtt{Prod}, R_\mathtt{p}), (\mathtt{Date}, R_\mathtt{d})\rangle$, where $R_\mathtt{s} = \{\mathtt{loc}\}$ (location), $R_\mathtt{p} = \{\mathtt{p}, \mathtt{m}\}$ (item and manufacturer) and $R_\mathtt{d} = \{\mathtt{y}, \mathtt{m}, \mathtt{d}\}$ (year, month, day). The value

$amount(t_{\mathtt{s}}, t_{\mathtt{p}}, t_{\mathtt{d}})$ is the sales amount of product $t_{\mathtt{p}}$ reported by store $t_{\mathtt{s}}$ on day $t_{\mathtt{d}}$. □

One may view that a dimension relation gives all the possible values (tuples) for the corresponding dimension. A more important point of view, however, is that the dimension relation actually provides a "basic" way of grouping. Indeed, for a given dimension D, each tuple t of the dimension can be taken as the "name" of a group and the data in the group are all the values mapped from all the possible combinations of t (from dimension D) with tuples from the other dimensions. That is, dimension D has the fixed value t while all the other dimensions take all the possible values. In the above example, a store (represented by its location) gives the group of all the sales data of all the products ever reported by the store. A day on the Date dimension gives the group of the sales data reported by all the stores of all the products on that day. The above intuition of using a relation to group data in a cube is extended to an arbitrary relation, leading to the notion of a "grouping relation". First, we have:

Definition A *dimension attribute* is a pair of a dimension name and an attribute name. A *grouping (relation) scheme* is a finite set of dimension attributes and attribute names.

Hence, a grouping scheme (denoted by G) is simply a traditional relation on ordinary attributes as well as dimension attributes. In the sequel, we shall use the notation $\mathtt{D}.A$ to denote a dimension attribute. The difference between a dimension attribute and an ordinary attribute is that the dimension attribute is prefixed with a dimension name. In order to simplify the presentation, we assume there exists a dummy "dimension name" and each ordinary attribute A is viewed as a *dummy dimension attribute*. Hence, when we refer to *dimension attributes*, it may refer to a dimension name and attribute name pair or an attribute name (prefixed by the dummy dimension name).

A *grouping relation* on a grouping scheme G is simply a relation on G by ignoring the dimension names. That is, a *(grouping) tuple* on G is an "ordinary" tuple such that $t[\mathtt{D}.A] \in \mathtt{dom}(A)$ for each $\mathtt{D}.A$ in G, and a grouping relation is a set of grouping tuples. The symbol g, possibly with a subscript, is used to denote a grouping relation.

Note that a grouping relation is basically a traditional relation with dimension attributes, although dimension attributes carry useful information. Hence, all the traditional relational operations are applicable, except for the rename and cross product operations (explained later). The relational operations simply take the dimension attribute $\mathtt{D}.A$ as an ordinary attribute with a complex name.

Let $C = (r_1, \ldots, r_n, \mu)$ be a cube on the scheme $\langle(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\rangle$. Then each r_i gives a grouping relation g_i that consists of all the tuples in r_i. The only difference is that the (dimension) attributes of g_i are the attributes of r_i prefixed with the dimension name $\mathtt{D}_i$. It will become clear when we define the semantics of a grouping relation below, that the information carried by the dimension name is important. First, we have:

Definition Let $S = \{(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\}$ be a cube scheme and G a grouping relation scheme. Then G is said to be *applicable to* S if for each non-dummy dimension attribute $\mathtt{D}.A \in G$, $A \in R_i$ and $\mathtt{D}_i = \mathtt{D}$ for some i.

Thus, if a grouping relation scheme G is applicable to a cube scheme, then for each non-dummy dimension attribute $\mathtt{D}.A$ appearing in G, D must be a dimension name in the cube scheme, and A must be in the relation scheme for dimension D. Intuitively, a tuple (with dummy dimension attributes taken out) on G can be extended to be a set of coordinates of a cube on S.

Let g be a grouping relation on G and C a cube on S, where G is applicable to S. Then each tuple t of g gives a group of coordinates from C that are "extensions" of t. For example, each tuple t in g_d (g_d is the basic grouping relation copied from r_d in the cube SALES) gives all the coordinates of SALES whose Date dimension is t. These coordinates correspond to the sales data reported by all the stores for all the products on day t. This observation is the basis of the following definition. However, we carry this observation one step further: We combine the SQL "group by" into our grouping mechanism.

Definition Let G be a grouping scheme, X a subset of G and $S = \langle(\mathtt{D}_1, R_1), \ldots, (\mathtt{D}_n, R_n)\rangle$ a cube scheme such that G is applicable to S. Let g be a grouping relation on G and $C = (r_1, \ldots, r_n, \mu)$ a cube on S. Then each tuple t in $\pi_X(g)$ gives the following set of coordinates, denoted by $\Gamma_X^{C,g}(t)$:

$$\{(t_1, \ldots, t_n) \mid t_i \in r_i \text{ for each } 1 \le i \le n \text{ and there exists } t' \text{ in } \sigma_{X=t}(r) \text{ such that } t'[R \cap R_i] = t_i[R \cap R_i] \text{ for each } 1 \le i \le n\}.$$

In the above definition, the set X first partitions the relation r into subrelations such that two tuples belong to one subrelation iff they have the same X values. This is the same as the "group by" clause in SQL. Each subrelation then gives a group of coordinates which is a union of the coordinates obtained by each tuple (as explained earlier) in the subrelation.

$g_r =$

Region	Store.loc
East	NewYork
East	Washington
West	LosAngeles

For $t = \langle$East$\rangle$, $\Gamma^{\mathtt{SALES},g_r}_{\mathtt{Region}}(t)$ gives the coordinates in Sales corresponding to the tuples 1, 2, 4 and 5 in Figure 1 since these coordinates have the Store.loc value NewYork or Washington.

Figure 2: Grouping using regions on SALES.

The "group by" extension here is rather useful. A grouping relation g_r on the scheme $G = \{\mathtt{Region}, \mathtt{Store.loc}\}$ may be intuitively used to group stores into regions. Clearly, $\Gamma^{\mathtt{SALES},g_r}_{\mathtt{Region}}(t)$, where t is a tuple of scheme $\langle\mathtt{Region}\rangle$ (i.e.,

representing a region), yields all the data reported by all the stores in region t. (Figure 2 shows an example of g_r and a corresponding grouping on the cube SALES.) This expressiveness is from the combination of the "group by" and the grouping relations. Note that the above grouping mechanism includes a "group by" component and is different from the "group by". The grouping mechanism here uses two levels of grouping: The first is from the relation itself (similar to the SQL "group by"), and the second from the cube.

4 Grouping algebra

Relational operations provide a powerful grouping mechanism to derive grouping relations from basic ones. However, these operations are still limited for many purposes in OLAP applications. In this section, we extend the relational algebra to include order-oriented operations, and aggregation operations.

Relational operations

As mentioned earlier, each traditional relational operation, except for the rename operation, can be applied to grouping relations. Due to the special semantics of grouping relations, we revise the rename operation as follows:

Operation (rename) Let $\mathtt{D}.A$ be a dimension attribute ($\mathtt{D}$ could be dummy) and B an attribute name. Let g be a grouping relation on scheme G such that $\mathtt{D}.A \in G$ and $B \notin G$. Then $\mathtt{ren}_{\mathtt{D}.A,B}(g)$ is the operation that changes the attribute $\mathtt{D}.A$ in g to B. □

By definition, a rename operation can change a regular attribute to a regular one or change a dimension attribute to a regular one. This restriction is due to our intention that grouping relations represent groups formed from the basic groups given by cubes. This is what we call the "property of non-empty reference". As an example, consider the relation $g = \{(1, 1995, \mathtt{Fairfax}), (2, 1996, \mathtt{Fairfax})\}$ on the scheme $\{\mathtt{Group}, \mathtt{Year}, \mathtt{Store.loc}\}$. Suppose we rename year (a regular attribute) to Date.y (denoting the renamed grouping relation as g'), and suppose that the Date dimension of cube SALES does not contain any tuple with year 1995. Then the sum of the data for the coordinates in $\Gamma^{\mathtt{SALES},g'}_{\mathtt{Group}}(1)$ would return zero, and the sum of $\Gamma^{\mathtt{SALES},g'}_{\mathtt{Group}}(2)$ would be the total sales amount of the Fairfax store in year 1996. This will cause confusion since the result zero could be interpreted as (i) "there is no sales data by the store in 1995" or (ii) "the sales in 1995 is zero".

A similar restriction applies to the cross product operation. In order to maintain the non-empty reference property, the two operands of the cross product must not contain the same non-dummy dimension name.

Order-oriented operations

In order to express groupings that are order related, we first introduce the concept of a "tuple order" on relations. By using the tuple order, we define the ordered roll operations on groupings. We assume there exists a total order on the domain of each attribute name. And we use the notation $\vec{X}$, where X is a set of (dimension) attributes, to denote a permutation of the attributes in X.

Definition Let X be a set of dimension attributes and $\vec{X} = \mathtt{D_1}.A_1, \ldots, \mathtt{D_k}.A_k$ a permutation of X. Then $O = (\vec{X}, \theta)$, where θ is a symbol in the set $\{\mathbf{desc}, \mathbf{asc}\}$, is called a *tuple order on* X and gives a total order $\preceq_O$ on the tuples on scheme X as follows: For tuples t_1 and t_2 on X, $t_1 \preceq_O t_2$ iff either $t_1 = t_2$, or there exists a $1 \leq j < k$ such that (1) $t_1[\mathtt{D_1}.A_1, \ldots, \mathtt{D_j}.A_j] = t_2[\mathtt{D_1}.A_1, \ldots, \mathtt{D_j}.A_j]$ and (2) if $\theta = \mathbf{desc}$, then $t_1[\mathtt{D_{j+1}}.A_{j+1}] > t_2[\mathtt{D_{j+1}}.A_{j+1}]$, and if $\theta = \mathbf{asc}$, then $t_1[\mathtt{D_{j+1}}.A_{j+1}] < t_2[\mathtt{D_{j+1}}.A_{j+1}]$.

That is, the tuples on X are ordered lexicographically, descendingly (**desc**) or ascendingly (**asc**), by looking at the attributes in the order given by the permutation $\vec{X}$. For example, if $\vec{X} = A, B$ and $\theta = \mathbf{asc}$, then the tuple $(1, 2)$ is before the tuple $(1, 3)$. It is clear that each tuple order is a total order. And hence, for each relation on X, it is easily seen that a tuple order on X induces a sequence of all the tuples in the relation as follows:

Definition Let $O = (\vec{X}, \theta)$ be a total order on X and g a grouping relation on scheme X. We use $O[g]$ to denote the sequence $t_1, \ldots, t_n$, where $t_i \preceq_O t_j$ for all $i < j$ and $t_1, \ldots, t_n$ are all the tuples in g.

The sequence $O[g]$ is simply a "sorted" relation by using the attribute order specified in O. Clearly, this is similar to the "Ordered by" clause in SQL.

We now define the operations that generate interval groupings. In order to do so, we need the following notation regarding attribute name conversions. For each attribute name A (not a dimension attribute), we always assume that $\mathtt{T}$-A, where $\mathtt{T}$ is an arbitrary symbol, is a new attribute name that have the same domain as A. Also, for a given set $X = \{\mathtt{D_1}.A_1, \ldots, \mathtt{D_k}.A_k\}$ of dimension attributes, let $\mathtt{T}$-X be the set of "regular" attributes $\{\mathtt{T}\text{-}A_1, \ldots, \mathtt{T}\text{-}A_k\}$. We are now ready to define our roll operation:

Operation(roll) Let b (*begin*), s (*step*) and l (*length*) be positive integers (s can be 0), g a grouping relation on scheme G, X a subset of G such that no (regular) attribute name appearing more than once in X and O a tuple order on X. Assume $O[\pi_X(g)] = t_1, \ldots, t_n$. Let $m \geq 0$ and let $i_m = b + s * m$ and $j_m = i_m + l$. If $i_m \leq j_m \leq n$, then for each $i_m \leq k \leq j_m$, let t_{mk} be the tuple on the scheme $\mathtt{start}\text{-}X \cup \mathtt{end}\text{-}X \cup X$ such that $t_{mk}[\mathtt{start}\text{-}X] = t_{i_m}$, $t_{mk}[\mathtt{end}\text{-}X] = t_{j_m}$ and $t_{mk}[X] = t_k$. Let g' be the relation consisting of all t_{mk} defined above. Finally let $\rho^{b,s,l}_O(g)$, called a *roll* of g, be the grouping relation $g \bowtie g'$. □

Thus, a roll operation first sorts the tuples in $\pi_X(g)$ by the given tuple order. Then it generates groups by taking intervals from the sequence (the result of the sorting). Specifically, tuples from the positions between b and $b + l$ are grouped together, tuples from the positions between

A_1	A_2	A_3
a	4	c
a	4	h
b	2	b
c	1	d

(i) relation g

A_1
a
b
c

(ii) $O[g]$ with tuple order $O = (A_1, \mathbf{asc})$

start-A_1	end-A_1	A_1	A_2	A_3
a	b	a	4	c
a	b	a	4	h
a	b	b	2	b
b	c	b	2	b
b	c	c	1	d

(iii) $\rho_O^{1,1,2}(g)$

Figure 3: Example of tuple order order and roll operation

$[b+s, b+s+l]$ are grouped together, and so on. (Here, b is the beginning position of the first interval, and s is the step length, i.e., the distance between the beginning of each two neighbor intervals, and l is the length of each interval.) Finally, each interval group generates a resulting group by joining to the original relation. Figure 3 shows an example of tuple order and roll operation.

As another example, assume g is a grouping relation on the scheme $\{\mathtt{y}, \mathtt{m}, \mathtt{d}\}$. Then $\rho^{1,3,3}_{(\mathtt{y},\mathbf{asc})}(g)$ creates a grouping relation that collects every three years (non-overlapping) into a group; $\rho^{1,3,3}_{(\mathtt{y},\mathtt{m},\mathbf{asc})}(g)$ creates a grouping that collects every three months (i.e., a quarter) into a group.

The roll operation is extended to the form $\rho_O^{b,s,*}(g)$ which is the union of $\rho_O^{b,s,l}(g)$ for all $l \leq |g|$. (The symbol $|g|$ denotes the number of tuples in g). The most often used roll operations in OLAP applications include the following:

- Overlapped moving of length k: $\rho_O^{1,1,k}$;
- Non-overlapped moving of length k: $\rho_O^{1,k,k}$;
- Forward cumulation: $\rho_O^{1,0,*}$; and
- All intervals: $\rho_O^{1,1,*}$.

Backward cumulation can be achieved simply by reversing the tuple order O in a forward cumulation, i.e., change $\mathbf{asc}$ in O to $\mathbf{desc}$ or vise versa.

The roll operation is very powerful and can be used to derive other interesting operations. The roll operation with a projection operation $\pi_R(\rho_O^{b,0,e}(g))$ selects the b-th to e-th items in g along the order given by O. Other complex order-related selection is also possible by using the roll operation. For instance, selecting every other tuple from a sequence obtained by a tuple order: $\pi_R \rho_O^{1,2,1}(g)$.

Aggregation operation

The final operation in our extension is the aggregation operation. We define an *aggregation function* as a mapping from the multi-subsets of $\mathcal{V}$ to $\mathcal{V}$. For example, sum, avg, min and count are the often used ones.

Operation(aggregation) Let g be a grouping relation on scheme G and C a cube on scheme S, such that G is applicable to S. Let f be an aggregation function, and A an attribute name not appearing in G. Then $f^X_{C,A}(g)$, called an *aggregation* operation, is the grouping relation g' on $X \cup \{A\}$ such that $\pi_X(g') = \pi_X(g)$ and for each $t \in g'$, $t[A]$ is the result of applying f to the multiset $\{\!\{\mu(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in \Gamma^{C,g}_X(t[X])\}\!\}$. □

Intuitively, the aggregation operation $f^X_{C,A}(g)$ uses X and g to group the coordinates in C, and then apply the aggregation function f on the data from each group. The result is a grouping relation on $X \cup \{A\}$, where A holds the aggregation values for each group. To illustrate, let g be the grouping relation on $\mathtt{Date.y}$. Then each tuple (y, tol) in $\mathtt{sum}^{\mathtt{Date.y}}_{\mathtt{SALES.total}}(g)$ describes the national total sales amount tol of year y.

5 MDDB and MDDB queries

In this section, we present the notion of a multidimensional database (MDDB) and its query language.

A *multidimensional database* is a finite set of multidimensional cubes and a finite set of grouping relations. The cubes register basic values (such as sales amounts) with basic groupings (provided by the dimension relations), and the grouping relations give certain "built-in groupings". In addition, the cubes in an MDDB "share" dimension relations. That is, if two cubes have the same dimension name, the dimension relation on the dimension must be the same. This assumption not only saves storage, but also models realistic situations. Indeed, there may be many different data collected (as cubes), where "products" is one dimension in different cubes. The values for the "products" dimension for all these cubes should be the same.

The other restriction is that if g is a built-in grouping relation, then the values in g under a (non-dummy) dimension attribute must appear in the corresponding dimension. This requirement is to ensure the property of non-empty reference (see Section 4). As an example, suppose g is a built-in grouping relation on attributes $\{\mathtt{Region}, \mathtt{Store.loc}\}$. Then each value under $\mathtt{Store.loc}$ in g must appear in the relation $r_{\mathtt{s}}$, i.e., the relation for the store dimension. The non-empty reference property is to ensure unambiguous semantics of grouping relations. Indeed, if this property is not satisfied and the summation on a group returns a zero value, then it could be either that the summation does have a value zero, or that the dimension attribute values in the grouping relation do not appear in the cube. In order to distinguish the two cases, run-time checking would be necessary. In this paper, we advocate static checking. That is, if the MDDB itself has the non-empty reference property, then through a syntactical

restriction on query expressions, we can ensure that the result of the query has the non-empty reference property.

Grouping algebra expressions and their corresponding semantics on MDDB can now be defined by using the operations introduced so far. Due to space limitation, we omit the details here but give some query examples below.

Example 3 We now use four query examples to illustrate the grouping algebra. Suppose we have an MDDB on the scheme $(\mathbf{D}, \mathbf{C}, \mathbf{G})$ with $\mathbf{D} = \{\texttt{Date}, \texttt{Prod}, \texttt{Store}\}$, $\mathbf{C} = \{\texttt{SALES}\}$ and $\mathbf{G} = \{\texttt{Region}\}$. The schemes of the dimensions $\texttt{Date}, \texttt{Prod}, \texttt{Store}$, as well as the scheme of the cube SALES, are as in Example 2. The grouping relation for the built-in grouping relation Region is on scheme $\{\texttt{reg}, \texttt{Store.loc}\}$

Q.1 *Find out the the names of the last year's (1994) top 5 selling product (including all manufacturers)*

$$T_{5+} = \pi_{\texttt{Prod.p}}(\rho^{1,0,5}_{\texttt{total,desc}}(\texttt{sum}^{\texttt{Prod.p}}_{\texttt{SALES,total}}(\sigma_{\texttt{Date.y}=1994}(\texttt{Prod} \bowtie \texttt{Date}))))$$

Note that the result may contain more than five products since there may be more than one product having the same total sales amount. If we used $\rho^{1,0,5}_{\texttt{total,Prod.p,desc}}$ in the above expression instead of $\rho^{1,0,5}_{\texttt{total,desc}}$, we will get at most five products.

Q.2 *For each member store, find out the year-to-date total sales amounts for each day this year (i.e., the daily cumulative sales amounts over 1995)*

$$\texttt{sum}^{\texttt{end-m,end-d,Store.loc}}_{\texttt{SALES,total}}(\rho^{1,0,*}_{\langle\texttt{Date.m,date.d}\rangle,\texttt{asc}}(\sigma_{\texttt{Date.y}=1995}(\texttt{Date} \bowtie \texttt{Store})))$$

In this expression, for each day (i.e., day end-d in month end-m) and each store (i.e., located at Stroe.loc) a summation is performed and the result is in total.

Q.3 *Find the year-to-date total sales amounts, in each region, of each product whose last year's nation-wide total sales amount was ranked among top five.*

$$T_{5+} \bowtie \texttt{sum}^{\texttt{reg,Prod.p}}_{\texttt{SALES,total}}(\texttt{Region} \bowtie \texttt{Prod}).$$

Q.4 *For all those products that are in the set of products manufactured by m_1 and m_2, find the total sales amounts of this year (1995).*

$$\texttt{sum}^{\texttt{Prod.p}}_{\texttt{SALES,total}}(\pi_{\texttt{Prod.p}}(\sigma_{\texttt{Prod.m}=m_1 \vee \texttt{Prod.m}=m_2}(\texttt{Prod})))$$

The result relation of this expression is a set of pairs $\{(p, tot)\}$, with each tuple p being a product that is on the list of products manufactured by m_1 and/or m_2 and tot is the total sales of p in the chain store, which may include those that are not manufactured by m_1 and m_2. (For example, suppose m_1 and m_2 both produce telephone sets. Then the above query will retrieve total sales of telephone sets, regardless the manufacturer.) □

6 Multidimensional cube algebra

As mentioned in the introduction, in a data warehouse environment, multidimensional cubes in an MDDB are materialized views expressed in a query language, which retrieve data from distributed local databases. In this section, we present a multidimensional cube algebra that can be used as such a query language.

The cube algebra consists of six operations that are mappings from cubes to cubes, as well as from relations to cubes. The purpose is to construct data from local databases into suitable multidimensional cubes. The first operation is the "add dimension" operation that adds an additional dimension to a cube. Due to space limitation, we only give an informal description of the cube algebra operations.

Operation(add dimension) The add dimension operation $\alpha_{\texttt{D}}(C)$ generates a new cube C' from the input cube C. Cube C' has a new dimension named D whose relation scheme is the empty set, and the relation for the dimension has only one tuple, namely the empty tuple []. The value for the coordinate $(t_1, \ldots, t_n, [])$, with the last dimension being D, in C' is the value for the coordinate $(t_1, \ldots, t_n)$ in C. □

One use of this operation is to make the input cube to have the same dimensions as another one to make the union (an operation on cubes defined later) of the two cubes possible. Note that the new dimension appearing as the last dimension is for notational convenience. Indeed, as mentioned earlier, the order of the dimensions does not matter in a multidimensional cube. This comment applies to the definitions of the other operations in this section.

The second operation is the transfer operation that rearranges the dimensions of a cube.

Operation(transfer) The transfer operation $\tau^{\texttt{D}_2,B}_{\texttt{D}_1,A}(C)$ generates a new cube by transferring attribute A of dimension $\texttt{D}_1$ to be a (new) attribute B on dimension $\texttt{D}_2$. More specifically, the transfer operation projects out the values under an attribute (A) from the first dimension, and put these values into the second dimension (via a Cartesian product). The new value mapping changes accordingly as follows: Given tuples t'_1 and t'_2 from the first and second dimensions of the new cube, the the value mapping tries to see if these two tuples are formed from t_1 and t_2 (tuples from the first and the second dimensions of the original cube) such that t'_1 and t'_2 can be obtained by "transferring" A value in t_1 into a B value in t_2 (while the other attribute values do not change). If this is the case, then the new value mapping uses the value from the original value mapping. Otherwise, the new value mapping has to use the null value. Figure 4 shows a particular transfer operation. □

To illustrate the usage of the transfer operation, consider a multidimensional cube C consisting of a Date dimension, with attributes $\texttt{y}, \texttt{m}, \texttt{d}$. Let $C' = \tau^{\texttt{Year},y}_{\texttt{Date},y}(\alpha_{\texttt{Year}}(C))$. Then cube C' "extracts" out the year information into a new dimension. This may be necessary in order for C to be combined with another cube into a cube view. As a further example, suppose that C has another dimension called Store. We may transfer the year attribute into store dimension. The result may be more intuitive to a user who

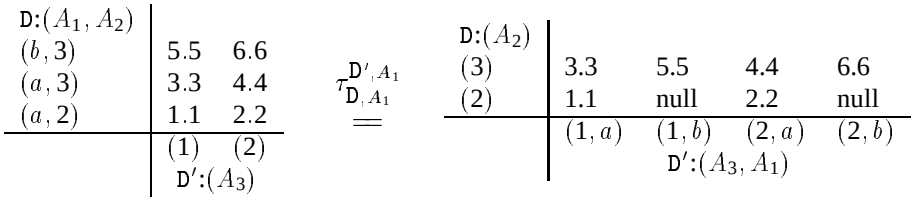

Figure 4: Example of transfer operations.

wants to view the behavior of store together with the year.

The third operation combines two cubes into one.

Operation(union) The union of cubes C_1 and C_2 along the dimension $\mathtt{D}_1$ is denoted by $C_1 \uplus_{\mathtt{D}_1} C_2$. The cubes C_1 and C_2 must have the same schemes and must not share any tuple on dimension $\mathtt{D}_1$. The union operation simply unions all the coordinates of the two given cubes together, and each coordinate get its original value. In the process, there may be new coordinates generated. The null value is used on these new coordinates. Note that union of two cubes require that at least one dimension are disjoint (i.e., $\mathtt{D}_1$ in the above definition). This guarantees that each coordinate in the new cube have at most one origin, either from the first cube or the second, avoiding any ambiguity. □

The fourth operation is the cube aggregation.

Operation(cube aggregation) For a given aggregate function f, the cube aggregation $f^{R'_{i_1},\ldots,R'_{i_m}}_{\mathtt{D}_{i_1},\ldots,\mathtt{D}_{i_m}}(C)$ gives a cube C' on the scheme $\langle(\mathtt{D}_{i_1}, R'_{i_1}), \ldots, (\mathtt{D}_{i_m}, R'_{i_m})\rangle$. Cube C' is obtained by "compressing" the cube into a smaller one. Each coordinate $(t'_{i_1}, \ldots, t'_{i_m})$ in C' corresponds to all the coordinates in the original cube that match the partial tuples t'_{i_1}, $\ldots$, t'_{i_m}. The data at the coordinate $(t'_{i_1}, \ldots, t'_{i_m})$ in the compressed cube is the aggregation value (via f) of the data in the original cube at the coordinates that are extensions of $(t'_{i_1}, \ldots, t'_{i_m})$. Figure 5 shows an example. □

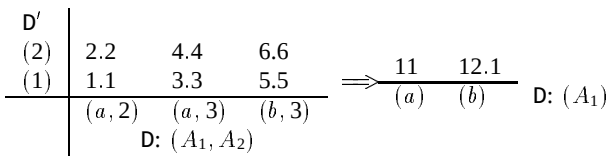

Figure 5: Cube aggregation $\mathtt{sum}^{A_1}_{\mathtt{D}}$.

Note that cube aggregation operations, together with other operations, can be used to "project out" attributes from a dimension relation in a cube. (Note that a direct projection on a dimension relation may cause semantic ambiguity, since each coordinate after the projection may correspond to more than one coordinate in the original cube.) Indeed, let C be a cube on $\langle(D_1, R_1 \cup \{A\}), (D_2, R_2)\rangle$, and $C' = \mathtt{sum}^{R_1,R_2}_{\mathtt{D}_1,\mathtt{D}_2}(\tau^{\mathtt{D}',B}_{\mathtt{D}_1,A}(\alpha_{\mathtt{D}'}(C)))$. (That is, C' is obtained first by adding a temporary dimension $\mathtt{D}'$ and the transferring A to this temporal dimension. Then $\mathtt{D}'$ is compressed out via sum.) It is clear that the difference between C and C' is the the attribute A is dropped from C, and all the data at the coordinates that have the same A value (in dimension $\mathtt{D}_1$) are summed together.

The next two operations deal with the interaction between (regular) relations and cubes. We believe these operations are important especially if in the data warehouse environment, some local databases are relational. The first such operation is a join operation that joins a (regular) relation into a dimension relation.

Operation(rc-join) An rc-join $r\hat{\bowtie}_{\mathtt{D}_1} C$ joins the relation r into dimension $\mathtt{D}_1$ of cube C. The result is a new cube with dimension $\mathtt{D}_1$ having the join result of r of the dimension $\mathtt{D}_1$ relation of C. The cell data on $(t_1, \ldots, t_n)$ for the new cube (assuming $\mathtt{D}_1$ is the first dimension) is the cell data on $(t'_1, \ldots, t_n)$ with t'_1 being the projection of t_1 on the attributes of $\mathtt{D}_1$ in C. □

Finally, we can construct a cube from a relation:

Operation(construct) The construct operation $\kappa^A_{\mathtt{D}}(r)$ generates a cube from relation r such that the new cube is one dimensional (with the only dimension $\mathtt{D}$) whose dimension relation is r and the cell value for the coordinate (t) is $t[A]$, where A is an attribute of r. □

The following example shows two cube views using the multidimensional cube algebra.

Example 4 Consider in our store chain example. Suppose each store reports a cube $C_i = (r_{\mathtt{d}}, r_{\mathtt{p}}, amount_i)$ to the headquarters. These cubes and some other localized relations like region relation $r_{\mathtt{g}}$ on the scheme $\{\mathtt{reg}, \mathtt{loc}\}$, etc., are used to construct cubes in the chain store headquarters data warehouse. Two of the cubes are given below as cube algebra expressions:

$$\mathtt{SALES} = (\alpha_{\mathtt{Store}}(C_1)\hat{\bowtie}_{\mathtt{Store}}\{\langle \mathtt{loc}_1\rangle\}) \uplus_{\mathtt{Store}} \cdots \uplus_{\mathtt{Store}} (\alpha_{\mathtt{Store}}(C_m)\hat{\bowtie}_{\mathtt{Store}}\{\langle \mathtt{loc}_m\rangle\}),$$

where m is the number of member stores, $\mathtt{loc}_i$ is the location of the ith store and $\langle \mathtt{loc}_i\rangle$ is a unary "constant" tuple; and

$$\mathtt{R\_SALES} = \hat{\mathtt{sum}}^{R_{\mathtt{p}},\{\mathtt{reg}\}}_{\mathtt{Prod},\mathtt{Region}}(\mathtt{SALES}\hat{\bowtie}_{\mathtt{Store}} r_{\mathtt{g}})$$

The cube SALES is the view of daily sales per product of each store, while R_SALES is the up-to-date sales of each product in each region. □

7 Related research

In database research, systems similar to OLAP systems have been studied in the domain of statistical and scientific databases (SSDB) [Mic91]. Sharing many features with OLAP systems, SSDB systems typically contain a large

amount of data, and are designed for helping apply statistical data analysis to the stored data. Tremendous progress has been made in this area. Special data models have been proposed and query languages introduced [Su83, OOM85, RR93]. Optimization techniques, in particular efficient physical storage management [SW94, NR94, SS93] and pre-aggregation [CM89], have been investigated. Many of these results, especially the optimization techniques, can be directly used in OLAP systems. In this paper, we focused on providing a data model and algebraic query languages directly related to OLAP applications.

There are other researches related to OLAP applications. The paper [GBLP96] introduced a new grouping operation `CUBE` for the SQL group-by clause. (Note that `CUBE` is the name of an operation while we use to term "cube" to refer to a data "unit" like a "relation".) The basic idea is that for a given relation containing the attributes $A_1, \ldots, A_k$, the group-by-cube will group data (in the standard SQL group-by sense) using all possible proper subsets of these k attributes (totally $2^k - 1$ in all including the empty set excluding the whole set). The `CUBE` clause is aimed at the OLAP queries which require simultaneous aggregation over different combinations. It is easily seen that the model presented in this paper can also be extended with such a group-by-cube clause. There are other extensions made to SQL to accommodate OLAP. For example, Red Brick's RI-SQL [Red95] includes certain order related aggregation functions such `MovingAvg(n)` and `Cume` (cumulative total). In contrast, this paper tried to introduce a general framework to express these functions.

Recently, [AGS96] introduced a data model for multidimensional databases. Basically, [AGS96] uses a similar notion of cubes and gives a number of operations on cubes. The operations are similar to the six cube operations introduced here but with somewhat different capabilities. However, [AGS96] does not consider the grouping queries. Another difference is that the data model of [AGS96] only has cubes and does not include "regular" relations which we consider important in this paper.

Optimizing OLAP queries is another important research area that is beyond the scope of this paper, where pre-computation is a promising idea. Certain cube aggregations can be pre-computed to expedite query processing. However, due to storage limitation, the system needs to select a subset of these aggregations. The paper [HRU96] provided an algorithm that efficiently selects such a subset under system restrictions.

8 Conclusion

This paper formalized an MDDB data model and introduced a grouping algebra as its query language. The query language is flexible in expressing many intuitive OLAP queries, including order-related queries. We then presented a cube algebra which we used to define cube views in a data warehouse environment. The purpose was to provide algebraic query languages to support OLAP applications. There are many issues worth further studying. For example, the model with its languages should be studied for its expressiveness, and properties of the introduced operations should be investigated. Furthermore, optimization techniques for the queries in the algebras need to be considered. Finally, the extension of the model may be considered. For example, one may extend the cubes to have cell values being a tuple or even a relation.

References

[AGS96] R. Agrawal, A. Gupta, and S. Sarawagi. "A framework for research in multidimensional databases." Personal communication. 1996.

[Ban95] S. K. Bansal. Real world requirements for decision support - implications for RDBMS. *SIGMOD*, 1995.

[CCS93] E. F. Codd, S. B. Codd, and C. T. Salley. Beyond decision support. *Computerworld*, 27, 26 July 1993.

[CM89] M. C. Chen and L. P. McNamee. On the data model and access method of summary data management. *IEEE TKDE*, 1(4), 1989.

[Dre93] H. Dresner. Multidimensionality: Ready or not, here it comes, 1993. *Office Infor. Syst. Research Notes*. The Gartner Group. File: Technology, T-MD-1137.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. *Proc. of the 12th International Conference on Data Engineering*, 1996.

[HRU96] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. *SIGMOD*, 1996.

[Mic91] Z. Michalewicz, editor. *Statistical And Scientific Databases*. Elli Horwood, 1991.

[NR94] W. Ng and C. Ravishankar. A physical storage model for efficient statistical query processing. *Statistical and Scientific Database Management*, 1994.

[OOM85] G. Ozsoyoglu, Z. M. Ozoyoglu, and F. Mata. A language and a physical organization technique for summary tables. *PODS*, 1985.

[Red95] Red Brick Systems White Paper. *Decision-Makers, Business Data and RI-SQL*. RedBrick Systems, Los Gatos, CA, 1995.

[RR93] M. Rafanelli and F. L. Ricci. Mefisto: A function model for statistical entities. *IEEE TKDE*, 5(4), 1993.

[SS93] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. Technical Report S2K-93-32, U.C. Berkeley, 1993.

[Su83] S. Y. W. Su. Sam*: A semantics association model for corporate and scientific/statistical databases. *Information Science*, 29, 1983.

[SW94] K. Seamons and M. Winslett. Physical scheme for large multidimensional arrays in scientific computing applications. *Statistical and Scientific Database Management*, pages 218–227, 1994.