Mathematical foundations of the multidimensional database models

Alexander Russakovskii

Hyperion Solutions Corporation

Abstract

Mathematical foundations of the OLAP models are formulated and discussed. Rich combinatorial aspects of the multidimensional space are being heavily exploited to achieve completeness and expressability. The main new results are: 1) a rigorously defined closed operator algebra with a true multidimensional approach; 2) consistent treatment of structured metadata, 3) complete consideration of navigational, computational, partial and scoped attributes; 4) resolution of computational dependencies; 5) independence of relational models; 6) non-grid models. Comparison to other published models is performed.

1 Introduction

This note deals with the mathematical foundations of multidimensional database models (MDM). Various flavors of MDM have been initially used in the *OLAP* (on-line analytic processing) applications without extensive and rigorous mathematical foundation.

After the applications appeared, a number of models have been suggested for the OLAP domain as an extension of the relational algebra. The common feature of the majority of these models is that they essentially map multidimensional data structures onto relational and apply operators of the relational algebra. Although this approach is certainly justifiable, it is not clear why multidimensional models should be used at all when there is relational model out there. It should be noted that the primary advantage of the MDM approach is the reduction of the number of descriptive parameters from a product of values to their sum. Close interaction and connection with the relational database model (RDM) is often necessary because of the number of applications but is not required for MDM to exist. Besides that, the expense of conversion back and forth between the models is fairly high for the implementations.

A good introduction into the concepts and problem scope of OLAP applications is [MK99]. Codd ([C93], [CCS93]) has formulated a set of rules for OLAP and introduced the term itself. Several other publications need to be mentioned here.

The paper [PJ99] contains an overview of a number of models by other researchers ([AGS95], [GBLP97], [LW96], [GL96], [K96], [DT97], [RS90], [L98]) from the perspective of requirements of a particular application domain. The authors come up with a set of particular requirements, conclude that none of the above mentioned models satisfies them and propose their own model.

It should be noted that there is no compelling reason to apply MDM for a particular problem unless it helps to solve it efficiently. It is applicable when the data has specific multidimensional nature. Sometimes the RDM is more efficient, and often the problem requires heterogeneous approach combining both MDM and RDM. It can also be argued that given any set of requirements it is always possible to come up with additional requirements (in fact this naturally happens as the application domain deepens over time) so that the existing models would need to be extended. In this note we start with some basic extensible model and gradually add layers to it to cover

wider functionality. The endpoint is the OLAP model. The accent is being made on the multidimensional nature of the data and the best ways to analyze it.

Several OLAP vendors have developed quite sophisticated products with interesting underlying models but their overview is beyond the scope of this note.

The instances of the MDM are traditionally called *cubes* and are thought of as $n-$dimensional objects. One of the features of our approach is that we systematically exploit the rich combinatorial aspects of the multidimensional space, something often overlooked by other published research, where the authors consider in a cube either slices corresponding to members of one dimension, or individual cells, i.e. sets of either codimension 1 or codimension n, dropping all the intermediate cases from consideration.

Another problem that model designers are trying to solve is that selection operator in a multidimensional cube may produce a set of cells that does not have a cube shape. In order to keep the operator algebra closed, either selection is dropped from consideration, or completion of the set to a cube is suggested resulting in extra unwanted cells, or everything is considered as a set of cells which is essentially the same as the RDM. Here, a different approach is suggested which allows to keep the maximal possible dimensionality in the model with a rich set of operators and to have a closed algebra. We also briefly examine computational dependencies which arise in the process of analysis of data.

Those of the existing models that deal with structured metadata combine together navigational and computational properties of the attributes. In our approach, we distinguish attributes by these properties which provides additional expressive power.

The questions of design of multidimensional models for particular problems and various normal forms of multidimensional databases are out of the scope of this note and are subject of a forthcoming investigation.

The paper is organized as follows. In section 2, we introduce the basic model (multidimensional data set) and discuss its relational representations. In section 3, we develop some notation and explore multidimensional grids. Section 4 defines a set of operators on the cubes. In section 5 we gradually extend the basic model to cover structured metadata and computational dependencies. Appendix briefly discusses non-grid models.

2 Multidimensional data sets

A *model* is understood as a way to describe data and issue queries against it. The key issue is the definition of what a query actually is, in other words, the definition of the model's data structure set (space). The model then defines a set of operators (algebra) on this space to answer any query against the data. It is preferable that the set of operators be *closed,* i.e. their results belong to the same space. The model itself does not describe *how* to store the data.

In this note, we use layered approach to modeling, i.e. we start with a minimal set of features needed to define a particular set of operators. Additional features are introduced when they are necessary to define new operators, and we verify that the previously introduced operators are well defined in the extended model. This allows to keep the exposition simple.

As a starting point, we describe a *multidimensional data set* (MDS). We represent the data by means of a multidimensional *grid* (N-grid) Q which is a Cartesian product of a finite number N of certain finite discrete sets D_i called *dimensions*:

$$Q = \bigotimes_{i=1}^{N} D_i.$$

The elements of a dimension are called *members*. The sets D_i and their members are usually referred to as *metadata*. Fixing a member m_i in each of the dimensions D_i uniquely defines a *cell* c in the grid. We call the tuple $(m_1, ...m_N)$ coordinates of c and write $c_i = m_i$ to signify that the i-th coordinate of the cell c is m_i. The order of dimensions and the order of members within each dimension are usually fixed. To each cell one can associate a *data value* of some type. In general, type may be defined on a per cell basis, but that would somewhat contradict the multidimensional paradigm. Therefore in most implementations, types are defined by means of members of one of the dimensions, called *Measures*. The data type for each cell is then determined by its Measures coordinate. The Measures dimension corresponds more closely to the relational metadata because

relational metadata determines data types. However, the Measures dimension does not have to be present in the grid explicitly. In particular, all the data types may be the same, and there is a measure implicitly associated with the grid. Another way to express this is to always add a Measures dimension to the grid, even if it consists of one single member. Except for type definitions, the Measures dimension has the same properties as all the other dimensions. In certain cases, more than one measures dimension is required, i.e. the cell's data type is a function of several of its coordinates.

A cell may contain a value of the associated type or be empty (in which case we say it has NULL value). The cells may also be ordered by means of any space filling curve. So, there is dimension ordering within a grid, cell ordering within a grid and member ordering within a dimension.

A multidimensional data set (MDS) M is a pair (Q, V) where Q is an N-grid (with associated data types) and V is a mapping which assigns values to cells, i.e.

$$V(c) = V(c_1, ..., c_N)$$

is the value associated with the cell c. For a MDS M we will sometimes write $Q = Q(M)$ to distinguish between the "empty" grid (metadata) and the MDS itself in case confusion is possible. An instance of a MDS is traditionally called a "*cube*" although it only marginally resembles the well known geometric body. The cube's grid is also called its *scheme*. A multidimensional database consists of a collection of cubes. The *database scheme* includes the cubes' schemes and relationships between their dimensions and members. The way metadata elements are identified is irrelevant for our purposes, so for simplicity, in this note we assume that members, dimensions and cubes within the database scheme are uniquely identified by their names.

In the terminology of [PJ99], MDS is a kind of a "simple cube model".

2.1 Cubes and relational tables

Note the differences and similarities between the two-dimensional MDS and a relational table. Both are visualized as 2-dimensional objects. In the two-dimensional MDS, both dimensions have equal properties (except that one may define types) whereas in a relational table rows and columns are not interchangeable. In a table, given the metadata, one can keep as many records as desired, therefore it possesses one degree of freedom. In a MDS, the metadata fully determines the number of cells and hence limits the amount of data in the database. This difference reflects the usage of the two models (or, rather, the usage reflects the difference). Relational databases are usually used to store raw data whereas the multidimensional databases deal with clean organized data and are more suitable for certain data analysis. It is usually more appropriate for a data warehouse to organize its data multidimensionally (if the data has multidimensional nature).

Once all data in a relational table is complete and fixed, part of it can be transformed into metadata to form a cube. The major advantage is a significant reduction of the number of parameters (keys): to describe each individual data value, instead of the product of the dimension sizes used in the RDM, only a sum of the sizes is needed in MDM.

Certainly, any cube can be also represented by a relational table. As mentioned above, the type of data associated with every particular cell in a cube can be determined individually for each cell. But in that case we are losing the advantages of the parameter reduction. So assume that we use the Measures dimension to define types. In the latter case the data in an N-dimensional cube with a Measures dimension of size M can be represented by a relational table with $N - 1 + M$ columns where $N - 1$ columns ("metadata" columns) correspond to the $N - 1$ non-measure dimensions, and the remaining M columns ("data" columns, or "measure" columns) correspond to the measures. The table is called a *fact table* (FT).

Since the metadata columns form a key, they may be replaced by a single *Key* column containing integer keys.

Such a fact table realizes a 2-dimensional representation of a cube and is called its *2-dimensional fact table representation* (FTR-2).

One could further coalesce the columns corresponding to same types. If all the types are the same (and hence either any dimension may be considered measure dimension or there is an implicit measure associated with the whole cube) we

Dimensional data			Measures dimension				
Dim1	**Dim2**	**Dim3**	**Mem41**	**Mem42**	**Mem43**	**Mem44**	**Mem45**
Mem11	Mem21	Mem31	345.28	true	389.17	203.67	2/28/99
Mem11	Mem21	Mem32	351.16	false	$<>$	$<>$	1/1/00

Figure 1: A fact table (FT-2)

obtain a table with $N+1$ columns, N of which correspond to metadata and one corresponds to data (measure). Again, metadata columns may be replaced by a single Key column. This kind of a fact table realizes a 1-dimensional representation of a cube (FTR-1). Such representation is possible with varying data types as well, if we wrap all types into a data object wrapper.

If data types are determined on a per cell basis, only a 1-dimensional representation is possible (either "long" or "wide").

It is easy to see that a fact table reflecting an MDM is not an arbitrary table since it needs to have a group of metadata attributes and a group of measure attributes. The immediate restriction of the relational algebra to the fact tables is not closed (union, difference and selection preserve the fact table structure whereas certain projections and Cartesian products break it) and may be not very useful. That is where multidimensional models enter the game.

3 Exploring a multidimensional data set

When defining operators on multidimensional data sets, we need to start with the definition of a query. In order to be able to apply further queries to the result of a query, we need a query to return another multidimensional data set. So our query is a mapping from a MDS to another MDS.

Prior to defining operators on MDS cubes, we explore grids more closely in this section.

3.1 Multiindices, subgrids, projections and homogeneity

We first develop some notation. In what follows it will be convenient to use the notion of a multiindex. Let $\Im$ be the set $\{1...N\}$. Any subset of $\Im$ is called a *multiindex*.

Let $I = \{i_1...i_k\}$ be a multiindex. For given sets $A_i, i = i_1...i_k$ denote by A^I the Cartesian product

$$A^I = \bigotimes_{i \in I} A_i,$$

so that in particular $Q = D^{\Im}$.

Given a multiindex $I = \{i_1...i_k\}$, for a cell c with coordinates $(m_1...m_N)$, we write c_I for the set $(m_{i_1}...m_{i_k})$.

A subset of the grid Q which is a grid itself is called a *subgrid* of Q. We characterize such subsets below.

The operator of *dimensional projection* π_i acts on a subset X of the grid Q by returning the members of the dimension D_i present in the coordinates of the cells of X:

$$\pi_i(X) = \{m \in D_i \quad | \quad \exists c \in X, \quad c_i = m\}.$$

In particular, $\pi_i(Q) = D_i$.

For a multiindex I, the *general projection* operator π_I is defined as

$$\pi_I(X) = \{m \in D^I \quad | \quad \exists c \in X, \quad c_I = m\}$$

and the *composite projection* operator π^I is defined as

$$\pi^I(X) = \bigotimes_{i \in I} \pi_i(X).$$

Note, that the composite projection of any set contains its general projection, but not necessarily coincides with it. In particular, for any subset X of Q, $\pi_{\Im}(X) = X$, but $\pi^{\Im}(X) = X$ if and only if X is a subgrid of Q. The operator $\pi^{\Im}$ is also called *grid completion*. For a given subset $X \subset Q$, it returns the minimal (circumscribed) grid G containing X.

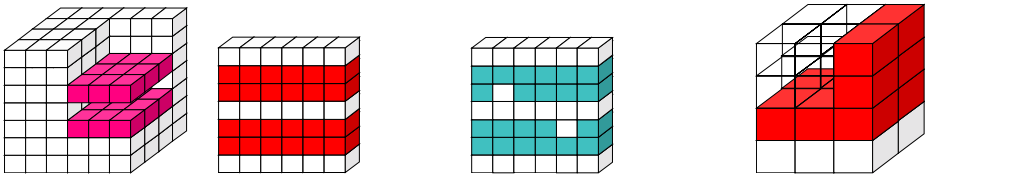
Similar notational conventions are used throughout the paper: I as a superscript is used to signify a product of some sets whereas I as a subscript refers to some subset of a product.

We say that a subset $X \subset Q$ is *homogeneous* with respect to index i (or dimension D_i) if it has product structure

$$X = \pi_i(X) \bigotimes \pi_{\Im \setminus i}(X).$$

Dim1	Dim2	Dim3	Dim4	Data
Mem11	Mem21	Mem31	Mem41	345.28
Mem11	Mem21	Mem31	Mem42	true
Mem11	Mem21	Mem31	Mem43	389.17
Mem11	Mem21	Mem31	Mem44	203.67
Mem11	Mem21	Mem31	Mem45	12/31/99

Figure 2: A fact table (FT-1)



a) A subgrid b) Sets with and without product structure c) A set with hdim=1

Figure 3: Homogeneity illustrated

The total number of dimensions with respect to which X has product structure is called its *homogeneous dimensionality* and is denoted hdim(X).

From the definition of the homogeneous dimensionality one easily derives the following

Proposition 3.1 *A subset $X \subset Q$ is a subgrid of Q if and only if* hdim(X) = N. *Otherwise* $0 \leq$ hdim(X) $\leq$ N $- 2$.

We say that a subset $X \subset Q$ is *homogeneous* with respect to a multiindex I if it has product structure

$$X = \pi_I(X) \bigotimes \pi_{\Im \setminus I}(X).$$

The set on the figure 3.c) is homogeneous with respect to multiindex $I_3 = \{3\}$ and with respect to multiindex $I_{12} = \{1, 2\}$ where indices 1, 2 and 3 correspond to cube's width, height and depth respectively.

We say that a subset $X \subset Q$ is *composite homogeneous* with respect to a multiindex I if it has product structure

$$X = \pi^I(X) \bigotimes \pi_{\Im \setminus I}(X).$$

The set on the figure 3.c) is composite homogeneous only with respect to multiindex I_3.

If X is composite homogeneous with respect to a multiindex I, then it is also homogeneous with respect to I, but the converse is not necessarily true.

From the definition of hdim(X) it follows that there exists a (*structural*) multiindex $I = I(X)$ of length $k =$ hdim(X) such that X is composite homogeneous with respect to $I(X)$. For the set on the picture I_3 is the structural multiindex.

From the above definitions one easily derives the following

Proposition 3.2 *The set of all multiindices for which a subset $X \subset Q$ is composite homogeneous is partially ordered by inclusion (or by length) and the structural multiindex $I(X)$ is the unique maximal element in it, i.e. it contains all the other multiindices with this property.*

If a subset $X \subset Q$ is homogeneous with respect to a multiindex I, its completion may be represented as the product of the grid completion of $\pi_I(X)$ and the grid completion of $\pi_{\Im \setminus I}(X)$:

$$\pi^{\Im}(X) = \pi^I(X) \bigotimes \pi^{\Im \setminus I}(X).$$

If X is composite homogeneous, it is enough to take the grid completion only of the second factor.

Note that determining hdim and the structural multiindex of a set without any additional information about this set is an expensive operation.

When dealing with non-homogeneous subsets of the grid, we are still interested in maintaining a multidimensional view of them. This interest is motivated by the goal of maintaining the number of descriptive parameters as low as possible

(otherwise FTR-1 or FTR-2 would be an obvious choice). In particular, we may be interested in considering the non-homogeneous subsets as homogeneous ones of lower dimensionality.

3.2 Restructuring and composite dimensions

Let X be a subset of a grid Q with hdim(X) < N. Then X is not a subgrid of the initial grid Q. Nevertheless it is possible to consider X as a subgrid of another grid Q', obtained by *restructuring* of Q. To define the required restructuring, consider the following construction.

Let I be some multiindex, such that $I \subset I(X)$, let N' be its length (so that $N' \leq$ hdim (X)), and let $I' = \Im \setminus I$. Make the $(N - N')$-dimensional set $D^{I'}$ into one *composite (virtual) dimension* D' whose members are all possible $(N - N')$-tuples of elements of the dimensions that make up $D^{I'}$. The projection of X onto $D^{I'}$, $\pi_{I'}(X)$, becomes this way a subset X' of D'. The set $Q' = D' \otimes D^I$ becomes a $(N' + 1)$-grid. It consists of the same number of cells as the original grid Q and is called the restructuring of Q. The set $X'' = X' \otimes D^I$ consists of the same cells as X, and from the properties of the structural multiindex formulated in the previous section it follows that X'' now becomes a subgrid of Q'. One says that the set X is *embeddable* as a subgrid into Q'. The multiindex I' is called the *embedding map*, or the *composite dimension map*. In order to define the restructuring completely, one also needs to specify a way to order the members in the composite dimension. We discuss this in more detail later on.

From the description of the restructuring by means of composite dimensions, it is easy to derive the following

Proposition 3.3 *Let X be a subset of an N-grid Q which is not a subgrid itself. Then the maximal dimensionality of a grid into which X is embeddable as a subgrid (using a composite dimension) is equal to* hdim(X) + 1. *The composite dimension is formed from the combinations of members of those dimensions with respect to which X is not homogeneous, and the map of the embedding is the complement of the structural multiindex $I(X)$.*

The representation of X as a grid of maximal dimensionality is called its *best homogeneous representation* and is denoted $H(X)$. Obviously X is a grid if and only if $X = H(X)$.

We now continue studying certain transformations of the grid that result in its restructuring.

3.3 Dicing and reshuffling

The *dicing* operator θ acts on a grid by changing the order of dimensions. Obviously, the number of cells and the dimensionality of the grid is not changed. A dicing may be encoded by specifying a multiindex, $I \subset \Im$ and its permutation I' and is denoted $\theta_{I,I'}$. If $I = \Im$, we omit the first multiindex and write simply $\theta_{\Im'}$.

Another family of operators that preserves the number of cells and the dimensionality of the grid may be defined as follows. A *dimensional reshuffling* τ_i (ω) is an operator that changes the order of members in the dimension D_i by means of a member *reordering transformation* ω. Given a multiindex I, a *general reshuffling* operator $\tau_I(\omega)$ changes the order of cells within D^I. A superposition of several dimensional reshufflings induces a *composite cell reshuffling* τ^I, but not every general reshuffling may be obtained in this manner.

3.4 Slicing, composite dimension maps and reordering transformations

The situation in which we had to create a composite dimension is a typical example of *slicing*. By that we mean combining two or more of the cube dimensions into one or more composite dimensions without changing the number of cells. We have already mentioned that the fact table FT-1 is essentially a set of pairs key-value and hence can be regarded as a one-dimensional grid. In order to consider it indeed as a one-dimensional grid we need to create a composite dimension in which we place all the combinations of the metadata column values in a certain order. It is this order that matters in the definition of slicing. The simplest possibility is the *dimensional slicing (with induced ordering)* λ_{ij} which maps the N-grid onto an $(N-1)$-grid by creating a composite dimension D' out of the member pairs from $D_i \bigotimes D_j$ ordered first by their i-th coordinate and then by their j-th coordinate. Denote by I_{ij} the multiindex $\{ij\}$. We write

$$\lambda_{ij} = \lambda(\nu_{I_{ij}}, \omega_{ij})$$

where the *composite dimension map* $\nu_{I_{ij}}$ is the mapping that defines which dimensions are trans-

a) A non-grid subset of a cube b) Its best homogeneous representation

Figure 4: "Deflation" of a cube subset

formed into a composite dimension ($D^{I_{ij}} \longmapsto D'$); this composite dimension is combined with the rest of the dimensions (which is essentially a dicing), and the reordering transformation ω_{ij} is the induced ordering in D'. Note that $\lambda_{ji} \neq \lambda_{ij}$ because of different ordering. Any other ordering ω (for instance, Z-ordering) in the grid $D_i \bigotimes D_j$ produces a different slice operator $\lambda(\nu_{I_{ij}}, \omega)$. More generally, given a multiindex $I = \{i_1 ... i_k\}$, a multiindex $J = \{j_1 ... j_m\}$, $m < k$, and an ordering ω in the composite m-grid D' obtained from D^I, one constructs a slice operator $\lambda(\nu_{I,J}, \omega)$ which maps the N-grid onto an M-grid ($M = N-k+m$) obtained by making D^I into D' with ordering ω and producing the product

$$Q' = \theta_{\Im_N \setminus I, \Im_M \setminus J}(D^{\Im_N \setminus I}) \otimes D',$$

so that $D' = D^J(Q')$. Note that all the grids of the same dimensionality obtained by slicing are equivalent up to a metadata remapping.

Dimensional slicing is similar to the GROUPBY clause in SQL.

By applying dimensional slicing subsequently (e.g., $\lambda_{12}\lambda_{23}...\lambda_{N-1,N}$) we eventually arrive at a one- or two-dimensional grid which is essentially equivalent to the fact table if we ignore the ordering. Normally, presentation of the data to the user on a two-dimensional computer screen or paper requires some sort of slicing.

Slicing reduces the dimensionality of the set ("deflates" the set) to which it is applied but the advantage is that non-grid subsets can still be considered as grids of maximal possible dimensionality (homogeneous representations), keeping the number of metadata parameters small.

3.5 Inflations and other shapings

As mentioned above, any slicing "deflates" the set, i.e. decreases the number of its dimensions while leaving the number of cells unchanged. Since the reordering transformation is a one-to-one mapping, given a map of composite dimensions, it is always possible to "inflate" the set and increase the number of dimensions. We can use similar notation $\mu(\nu_{I,J}, \omega)$ for an inflation, except that in this case the length m of J is larger than the length k of I. Obviously,

$$\lambda(\nu_{I,J}, \omega) = [\mu(\nu_{J,I}, \omega)]^{-1}.$$

Inflations, slicing, dicing and reshuffling belong to the set of *shaping* operators that do not change the number of cells in a grid but change its shape. It is easy to see that any shaping operator may be expressed as a combination of these (if we allow a composite dimension to coincide with an actual dimension, then reshuffling becomes a particular case of slicing).

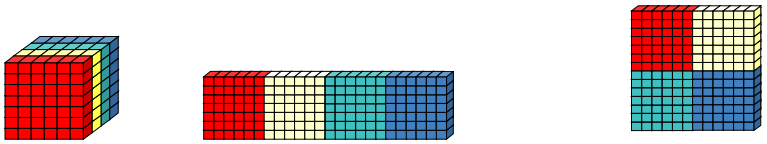
To summarize, there are several reasons why shapings are helpful:

- shapings can be used to make a subset of a grid into a subgrid of lower dimensionality (keep product structure);
- shapings allow presentation of data on a 2-dimensional computer screen or paper;
- shapings produce FTR-1 and FTR-2.

3.6 Products, quotients and common context

Let I be a multiindex of length k and let $m \in D^I$. Consider the subgrid $Q' = m \otimes D^{\Im \setminus I}$ of the initial grid Q as a $(N-k)$-grid. The latter grid is called the *section* of Q *via* m and is denoted $Q[m]$. Note that number of cells in $Q[m]$ and $Q' = m \otimes Q[m]$ is the same.

Let Q be an N-grid and let Q' be a $k-$grid. The *product* of these grids is the $(N+k)$-grid $Q \otimes Q'$.



a) Initial grid b) Example of dimensional slicing c) Example of general slicing

Figure 5: Slicing illustrated

The definition of the product assumes that the grids Q and Q' have no common dimensions. If some of their dimensions are composite, it is assumed that the composite dimension maps have no common components either.

Given two grids, Q_1 and Q_2, define the *intersection multiindex* $I_{Q_1}(Q_2)$ as the maximal multiindex $\{i_1...i_k\}$ such that dimensions D_{i_j} of Q_1 appear also in Q_2 (i.e. either D_{i_j} is a dimension of Q_2 or is present in a composite dimension map of Q_2).

Define the *quotient* of two grids Q and Q' by projecting out those dimensions in Q which occur in Q'. More precisely, the quotient Q/Q' is the best homogeneous representation of the set obtained by projection of Q onto $D^{\Im\setminus I_Q(Q')}$:

$$Q/Q' = H(\pi_{\Im\setminus I_Q(Q')}(Q)).$$

In particular, $Q/\{m\} = Q[m]$, $(Q \otimes Q')/Q' = Q$ and $Q/Q = \emptyset$.

Given any two grids Q_1 and Q_2, we sometimes need to embed them into some ("circumscribed") $N'-$grid as $N'-$subgrids. If two cubes have the same set of dimensions, we say that they are *within common context*.

Here is a procedure to bring two cubes within common context. If there are no composite dimensions, the dimensionality N' of their common context is $N' = \dim(Q_1)+\dim(Q_2)-k$, where k is the length of the intersection multiindex. There are numerous ways to bring each of the two grids to dimensionality N' if their dimensionality is different. For instance, in order to embed Q_1, choose an element $m \in Q_2/Q_1$ and form the product $Q_1' = m \otimes Q_1$. Any other subset $X \subset (Q_2/Q_1)$ of the quotient may be used as well. The mapping that embeds Q_1 into its common circumscribed grid with Q_2 by taking $Q_1' = X \otimes Q_1$, is denoted κ_{X,Q_2}.

If composite dimensions are present and their maps do not coincide, the procedure requires additional steps after Q_1' and Q_2' are formed. Namely, deflation must be applied to obtain identical composite dimension maps. The dimensionality of the common context is then less than N'.

4 Operators on a cube

In the previous section we defined operators on metadata. In this section we introduce operators on a cube.

4.1 Restrictions and selections

Consider a grid Q with dimensions D_i.

For a given index i and a subset $A_i \subseteq D_i$, the *dimensional restriction* operator ρ_{A_i} forms a subset of Q by replacing D_i with A_i in the Cartesian product:

$$\rho_{A_i}(Q) = A_i \otimes D^{\Im\setminus i}.$$

As it follows from the definition, the set $\rho_{A_i}(Q)$ is a grid.

More generally, for a given multiindex I and a subset $A_I \subseteq D^I$, one might want to define the *general restriction* operator ρ_{A_I} as forming a subset of Q in a similar way:

$$\rho_{A_I}(Q) = A_I \otimes D^{\Im\setminus I}{}^1.$$

However the set $A_I \otimes D^{\Im\setminus I}$ is not necessarily a grid. More precisely, it is a grid if and only if $A_I = A^I$ in which case ρ_{A_I} is a *composite restriction*, i.e. a superposition of $\rho_{A_i}, i \in I$ (it is clear that the dimensional restrictions commute with each other). In order for general restriction to result in a grid, we define it as

$$\rho_{A_I}(Q) = H(A_I \otimes D^{\Im\setminus I}).$$

[1] The domain of the restriction operator can be naturally extended to include arbitrary (not necessarily grid) subsets X of Q by setting

$$\rho_{A_I}(X) = \{c \in X \quad | \quad c_I \in A_I\}.$$

Here H is the best homogeneous representation introduced in the previous section. If the set A_I consists of a single element m, then the restriction is obviously a grid:

$$\rho_{\{m\}}(Q) = m \otimes Q[m].$$

Restriction operators also apply to the cubes as restrictions on their grids. We do not distinguish between operators on the cubes and operators on the grid in this section unless there is a potential for confusion.

Similar to projection in the relational algebra, restriction is defined in terms of metadata. Note that we deviate slightly from the relational algebra terminology by calling the just defined operation "restriction". The term "projection" is reserved for metadata projections in the context of the MDM.

Note that each restriction is defined in terms of some set A; equivalently it is possible to say that a restriction is based on a predicate on metadata.

Another operator, *selection,* denoted by σ_P, chooses a subset X of the grid based on a predicate P on data values and then takes its best homogeneous representation. It is clear that selection actually results in a (data-driven) general restriction. The property of X being a subgrid may depend on the data but may also be determined by properties of the predicate P.

For the purpose of providing examples to illustrate certain notions, we define a sample cube consisting of four dimensions:

D_1 *("Time")* = {*"Year", "Q1", "Q2", "Q3, "Q4", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"*},

D_2 *("Locations")* = {*"Regions", "East", "North", "West", "South", "NY", "NJ", "MN", "CA", "OR", "TX", "New York", "San Francisco", "Los Angeles", "My favorite locations"*},

D_3 *("Products")* = {*"All Products", "Soda", "Beer", "Diet", "Coke", "Sprite", "Pepsi", "Diet Pepsi", "Diet Coke", "Budweiser", "Heineken", "Guinness"*},

D_4 *("Measures")* = {*"Sales", "Units", "Expenses", "Profit", "Profit%", "Tax"*}.

Consider a simple example regarding this sample cube. Suppose the user wants to restrict his view of the data by the months and locations in which the value of *Sales* of *Coke* is greater than 600. By the "value of *Sales*" we mean the data value (meant to be a number in this case) associated with cells c, whose *Measures* coordinate is *Sales*. From the formulation of the query it is clear that the cells of interest must be picked from the section $Q[Coke, Sales]$. Neither the *Location* nor the *Time* is specified. So the resulting subset (let us call it *"Outstanding sales"*) is defined as "$Outstanding\ sales$" = $X \otimes Products \otimes Measures$, where $X = \{(t, l) \in Time \otimes Locations \mid V(t, l, Coke, Sales) > 600\}$.

Examining the values, we find that pairs {*Aug, New York*}, {*Jun, Los Angeles*}, {*Jul, Los Angeles*}, {*Aug, Los Angeles*}, {*Sep, Los Angeles*} qualify for X. One easily sees that hdim(X) = 0, so it does not have the product structure, hence hdim("Outstanding sales″) = 2 and therefore it is not a subgrid.

In order to make *"Outstanding sales"* into a cube we create a composite dimension D' (call it something like *"Time & Location"*) and associate X with a subset of D'. The multiindex $I = \{Time, Location\}$ is the map of D'.

On the other hand, if the user asks for the subset of data for which the values of *Sales* of *Coke* is $\leq$ 300, we obtain "$Poor\ sales$" = $\{Jan, Dec\} \otimes \{New\ York, Los\ Angeles\} \otimes Products \otimes Measures$ which is clearly a subgrid.

Obviously, the property of the result being a subgrid in our example depends on the data distribution. However for the type of predicate just examined, we can conclude that hdim of the result is never less than 2 regardless of data. One can associate with every predicate the minimal value of hdim over its possible results and call it the predicate's *dimensionality.*

As an example of a predicate P that has dimensionality $N = dim(Q)$, consider a query like "Restrict to those *Products* for which the value of *Sales* is greater than 600 in at least some *Location* at some *Time*", i.e. "$Popular\ products$" = $Time \otimes \pi_{Products}(X) \otimes Locations \otimes Measures$. where $X = \{(t, l, p) \in Time \otimes Locations \otimes Products \mid V(t, l, p, Sales) > 600\}$.

It is easy to see that for the mentioned predicate we always obtain a subgrid as a result regardless of the data values. Hence it has dimensionality 4.

Note that the metadata for this selection is ob-

Sales, Coke	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
New York	300	350	350	400	450	500	600	650	600	450	350	300
San Francisco	370	550	400	400	400	500	450	550	450	500	400	350
Los Angeles	300	450	500	500	600	650	700	750	700	600	500	300

Figure 6: Sample data for sales of coke

tained by first selecting the set X, then projecting X onto the *Products* dimension, and finally producing the product. However we cannot represent the whole selection operator as a composition of three operators because otherwise after the first selection we would have lost the values of those cells which have not been selected.

4.2 Joins and other binary operators

Restrictions and selections are unary operators, operating on a single cube. We now define a binary operator join. In order to formulate the definition, we first introduce an auxiliary operator, called product, which marginally resembles the relational Cartesian product. This operator simply adds one or more dimensions to the cube's grid by taking a product with some other grid. Recall that the number of cells in $Q[m]$ and $Q[m]\otimes m$ is the same, which poses no problem regarding the values associated to those cells. If products are formed of grids which have more than one cell, the way to assign values should be specified.

Let $C = (Q, V)$ be an $N-$cube, let Q' be a $k-$grid and let $f = \{f_c, c \in Q'\}$, be a family of functions on values parametrized by cells of $Q\prime$ (allocation functions). The *product* $C \otimes (Q', f)$ is defined as the $(N + k)$-cube $C' = (Q \otimes Q', V')$ where

$$V' = f_{\pi_{Q'}(C')}(V(\pi_Q(C')).$$

In other words, we take the product of the two grids and assign to the cells values obtained by applying the appropriate allocation function from the family f to the values from the original cube. If each f_c is the identity mapping $f_c(v) = v$, then the values in C' are simply replicated from C for every member combination in Q'. In the latter case we omit f in the notation and write simply $C \otimes Q'$.

As before, the definition assumes that the grids Q and Q' have no common dimensions, counting the dimensions in the composite dimension maps as well.

Recall that bringing two grids to the same dimensionality (embedding into their common circumscribed grid) by means of the mappings $\kappa_{X,.}$ requires a product operation. If κ is defined on cubes, a family of functions f is also involved. So for cubes we denote the operator as $\kappa_{X,f,.}$.

We now define a family of *join* operators that operate on a pair of cubes. They all assume the same basic algorithm.

First, two cubes C_1 and C_2 have to be brought within common context, i.e. both need to be embedded into a single cube as subsets. This is done by means of the mappings $\kappa_{X,f,.}$. Assume for now that the grids Q_1 and Q_2 are already within common context.

Next, given a predicate θ relating a pair of values v_1 and v_2 and a function φ defined on v_1 and v_2 and returning another value, define the *$\theta\varphi$-join* of C_1 and C_2 as follows. Let Q' $= Q_1 \cup Q_2, Q'' = Q_1 \cap Q_2$. Select a subset Q''' of those cells in Q'' for which the value of θ on the values v_1 and v_2 corresponding to C_1 and C_2 is *true* and associate with every such cell the value $\varphi(v_1, v_2)$. The join is defined as the set

$$Q = H((Q' \setminus Q'') \cup Q''').$$

Thus, the result of a $\theta\varphi-$join of two cubes is another cube.

The function

$$\varphi(v_1, v_2) = \begin{cases} v_1 & \text{if } \mathrm{v_1} \neq \mathrm{NULL}, \\ v_2 & \text{if } \mathrm{v_1} = \mathrm{NULL}, \mathrm{v_2} \neq \mathrm{NULL}, \\ NULL & \text{otherwise.} \end{cases}$$

is called the *standard joining function.*

If φ is the standard joining function, we write "$\theta-$join" instead of "$\theta\varphi-$join". A particular case of a $\theta-$join is the *natural join* for which the predicate is just $\theta = (v_1 = v_2)$, or $\theta = (v_1 = v_2 \wedge v_1 \neq NULL)$.

It follows that θ defines the metadata for the $\theta\varphi-$join by performing a selection involving two cubes and φ defines the data.

If $Q_1 \cap Q_2 = \emptyset$, then the result of a $\theta-$join is the union of the two cubes. If a cube has been split across one of its dimensions into two subcubes, then their join restores the original cube. Note that in the relational case, the $\theta-$join basically operates in a similar way, the difference being that the way to embed two relational tables into a single table is the Cartesian product of the tables which is different from the MDM product. In the case of MDM, due to rich combinatorial possibilities, there are many ways to bring two grids within common context (which we parameterize by sets X and function families f). Certainly, if the two participating cubes have been previously deflated from subsets of the same grid, there is an option of inflating them back, using the maps of composite dimensions and the reordering transformations. Otherwise we need to create composite dimensions.

As already mentioned, in order to determine the circumscribed grid, a possible strategy is to inflate both grids to (possibly non-grid) subsets of a grid with no composite dimensions, perform the join operations there and deflate the result. Another possibility is to perform deflation of both grids to grids which would have identical maps of composite dimensions, perform join operations there and inflate the result if possible. It is easy to see that both strategies lead to the same result since the underlying selection is cell based, and the number and contents of cells never change. The second approach seems to be preferable from the implementation point of view, because finding out whether a composite dimension can be inflated is normally less expensive an operation than finding out how to deflate a general set a cells. Of course, to be efficient, deflation should be performed using homogeneous representations (with maximal possible dimensionality). The relational approach is the other extreme consisting in deflating everything to the minimal possible number of dimensions (fact tables), and further inflation becomes as expensive as in the first approach.

The operators of *binary restriction* and *binary selection* make selection or restriction based on the relative position of two cubes. They let one form sets like $Q_1 \cap Q_2$, or $Q_1 \setminus Q_2$, etc[2].

The main difference between the relational join and the join of two cubes within the same context in MDM is that in the latter case metadata completely determines the cells, and hence there is no need to search for matching values to concatenate the records.

4.3 A summary of the multidimensional data set model

The multidimensional data set ("simple cube") model defined above is the basis of the MDM. It allows to define such operators as restriction, selection, product, join, various shapings. The result of any such operation is another cube of possibly different dimensionality. Hence the algebra of these operators is closed.

By adding "layers" to the multidimensional data set, we will gradually extend the model's properties.

5 Attributes, hierarchies and other metadata features

In this section, we define additional metadata-based features which allow to extend the multidimensional data set (MDS) model to a more sophisticated model with a richer operator algebra. This next model is called the *extended multidimensional data set* (EMDS) and reflects structured metadata. EMDS is gradually constructed by adding several layers to the MDS. For simplicity of naming, we call the extended models EMDS-1, EMDS-2, etc.

The term *attribute* will be used in this section in the following sense: it is a named entity γ corresponding to a subset α of elements of a certain set S. The mentioned subset is called the *attribute holder set*, and we say that γ is an attribute on S with base α. If α consists of a single element, the corresponding attribute is called *trivial*. We write $m \in \alpha(\gamma)$ to specify that m is an attribute holder for γ. We require that $\gamma \notin \alpha(\gamma)$ for any nontrivial attribute.

As an example, form the *My favorite locations"* subset of the *Locations* dimension of our sample cube consisting of, say, *New York* and *San Francisco*. The members *New York* and *San Francisco* are *attribute holders* for *"My favorite locations"* attribute. In this example, S is a set of

[2]An alternative option is to include these operators as part of the join definition algorithm. This way it is possible for a join to return any subset of $Q_1 \cup Q_2$. However the definition then becomes somewhat overloaded.

members of the *Locations* dimension, and the attribute *"My favorite locations"* is a *dimensional attribute*. In our model, a dimensional attribute may be identified with some member of some dimension, not necessarily of the one on which it is defined. We do not distinguish between an attribute and the corresponding member unless there is a potential for confusion.

The attributes are used for grouping the elements of a set for various purposes. There may exist several different attributes whose attribute holder sets coincide. For example, *"My favorite locations"* holder set may be defined to consist of *San Francisco* and *Los Angeles*, and *"CA"* holder set would consist of exactly the same. At the same time, dimensional attributes of two different grids may be identified, for instance, *"My favorite locations"* may also be defined on a different cube whose scheme possesses a *Locations* dimension as well, and include, say, *Boston* and *Dallas*. If the two cubes are joined, equivalences between the attributes help construct the attributes of the join.

Dimensional attributes may also be defined based on data, e.g. as dimensional metadata projection of a value-based selection in a cube. Examples of such attributes are dimensional components of *"Outstanding sales"* and *"Poor sales"* from the previous section.

An *EMDS-1* model is an MDS model with dimensional attributes, i.e. with the additional property that members of each dimension D_i are dimensional attributes. Since any member can be considered as a trivial attribute, MDS is a particular case of EMDS-1.

A typical dimension D in EMDS-1 is constructed as follows. Start with some set of members B called *dimension base* (or*level* 0*of* D). Consider a dimensional attribute γ associated with an element of 2^B, i.e. with a set $\alpha \subset B$ consisting of one or more elements of B. The elements of the set α are attribute holders for γ. We say that γ is a *level 1 attribute* if it is non-trivial. Let Γ be the set of those level 1 attributes γ that we chose to be members of the dimension D. We also say that Γ forms *level* 1 *of* D. Next consider the set $B' = B \cup \Gamma$. Repeat the previous step, i.e. consider elements γ' of $2^{B'}$, the set of subsets of B'. We say that γ' is a *level 2 attribute* if it is non-trivial and if its attribute holder set α' contains at least one level 1 attribute. The set Γ' consisting of those level 2 attributes on B' that we chose to be members of D, forms *level* 2 *of* D. Let $B'' = B' \cup \Gamma'$, and so on. The procedure just described is referred to as the *canonical bottom-up construction*.

As the result of the introduction of attributes, the structure of each dimension D in EMDS-1 may be represented by means of a finite (directed) graph G whose nodes correspond to members of D and whose edges connect attributes to their attribute holders (or vice versa, whichever is convenient). One says that the dimension has a *hierarchy*, and G is referred to as the dimension's *hierarchy graph*. The base of the dimension may also be defined as the set of members corresponding to those nodes in G that do not have outgoing edges. The*top* of the dimension is defined as the set of members corresponding to those nodes that do not have any incoming edges. In other words, the top consists of those attributes which are not attribute holders for any attribute. In the hierarchy terminology, if there is a path from one node to another, the latter is called the descendant of the former and the former is called the ascendant (or ancestor) of the latter.

For practical purposes, it is undesirable to have cycles in the hierarchy graph. Not only is there a potential for the relevant algorithms to go into an infinite loop, but also performance of these algorithms usually significantly degrades. Therefore, if G contains cycles (trivial attributes are not considered cycles), some procedure must be applied to eliminate them. Such procedure is called *resolution of references*. It amounts to a finite number of eliminations of some attribute holder from the attribute holder set corresponding to an attribute in a cycle, that is, removing an edge from the graph G. It remains to determine which one to remove. Here is one way to do it. Let the set $A = \{\gamma_1\gamma_2...\gamma_k\gamma_1\}$ be a cyclical path in G, i.e. $\gamma_1 \in \alpha(\gamma_2)$, $\gamma_2 \in \alpha(\gamma_3)$, ..., $\gamma_k \in \alpha(\gamma_1)$. Recall that the members in a dimension are totally ordered. Walk through the path A and remove the edge between the pair(s) of nodes which does not follow the mentioned total order (i.e. $\gamma_i > \gamma_j$, where $j = i + 1$ if $i < k$ and $j = 1$ if $i = k$). The just described resolution procedure is also known as *elimination of forward references*.

Once the graph G contains no cycles, it induces a partial order $\succeq$ on the members of the dimension D ($a \succeq b$, if a is an ascendant of b). With

respect to this order, an attribute always follows its holders (*bottom-up order*). This partial order may be different from the above mentioned total order. However there exists a total order on D which is compatible with this partial order, i.e. if two elements m_1 and m_2 are comparable in the partial order and $m_1 \succeq m_2$ with respect to this order, then also $m_1 \geq m_2$ with respect to the total order. In other words, this compatible total order is also a bottom up order. Such order can be constructed, for instance, if, after resolution of references, the members are enumerated within levels starting from level 0 followed by level 1 and so forth.

Observe that if a dimension is built by the canonical bottom-up construction, then it automatically has bottom up order, and we have the following

Proposition 5.1 *The bottom-up canonical procedure produces a graph with no cycles.*

Note that the graph G is not the "canonical" representation of the partial order it induces, because it may be redundant with respect to transitivity property; for instance, we can have $m \in \alpha(\gamma)$ and both $m, \gamma \in \alpha(\gamma')$, and transitivity is "ignored". Minimal elements with respect to this order correspond to the dimension base members. All other members are nontrivial attributes. There may be one or more maximal elements which make up the dimension's top.

Unless specifically mentioned, in what follows we assume that cycles in the dimensional hierarchies have been resolved.

A dimension in EMDS-1 is stratified by levels of its attributes. Also, the dimension is stratified by *generations*. Generation 0 (top) consists of maximal elements, generation 1 consists of their attribute holders, generation 2 consists of those attribute holders of the generation 1 attributes which are not generation 1 attributes themselves, and so on. In the graph terminology, here the level of a node is defined as the maximal number of edges needed to reach from it a node corresponding to a minimal element, and generation of a node is defined as the minimal number of edges needed to reach it from a node corresponding to a maximal element. The level-based stratification adequately reflects the bottom-up canonical procedure of dimension building. Similarly, one can start from the maximal elements (canonical top-down construction). Other stratifications as well as other definitions of levels and generations are also possible.

If stratification by generations produces the same sets as stratification by levels, the hierarchy is called *balanced*. Otherwise it is called *ragged* or *unbalanced*. If there exists more than one path from one node of the graph to another, the hierarchy is called *redundant* (sometimes they say that the dimension *has multiple hierarchies*). If there exists a node with more than one incoming edge, the hierarchy is called *mixed*. Otherwise it is called *pure*. Obviously, a redundant hierarchy is mixed, but the converse is not necessarily true. A hierarchy is called *connected* or *disconnected* if such is its graph.

A hierarchy is called *regular* if it is pure, balanced and connected (i.e. the graph is a tree). Otherwise it is *irregular*. In a regular hierarchy the top consists of a single member.

To provide an example, we turn our sample cube into an EMDS-1 cube. We structure the dimensions as follows:

D_1 ("Time") =
"Q1"={"Jan", "Feb", "Mar"},
"Q2"={"Apr", "May", "Jun"},
"Q3"={"Jul", "Aug", "Sep"},
"Q4"={"Oct", "Nov", "Dec"},
"Year"={"Q1", "Q2", "Q3", "Q4"}},

D_2 ("Locations") =
"NY"={"New"York"},
"CA"={"San Francisco", "Los Angeles"},
"East"={"NY", "NJ"},
"North"={"MN"},
"South"={"TX"},
"West"={"CA", "OR"},
"My favorite locations"={"New York", "San Francisco", "CA", "South"},
"Regions"={"East", "North", "West", "South"}},

D_3 ("Products") =
"Soda"={"Coke", "Sprite", "Pepsi", "Diet Pepsi", "Diet Coke"},
"Beer"={"Budweiser", "Heineken", "Guinness"},
"Diet"={"DietPepsi", "DietCoke"},
"All Products"={"Soda", "Beer"}},

D_4 ("Measures") =
"Profit"={"Sales", "Expenses"},
"Profit%"={"Profit", "Sales"},
"Units",
"Tax"}.

The *Time* dimension's hierarchy is regular. The other dimensions in this example are irregular.

For instance, the *Locations* dimension has the following stratifications:
level 0: {*"New York", "NJ", "MN", "TX", "San Francisco", "Los Angeles", "OR"*},
level 1: {*"NY", "CA", "North", "South"*},
level 2: {*"East", "West", "My favorite locations"*},
level 3: {*"Regions"*},
generation 0: {*"Regions", "My favorite locations"*},
generation 1: {*"East", "North", "West", "South", "New York", "San Francisco", "CA"*},
generation 2: {*"NY", "NJ", "MN", "TX", "Los Angeles", "OR"*}.

Its hierarchy is ragged and redundant but is not disconnected. The *Products* dimension's hierarchy is ragged and mixed but not redundant. However if we include *Diet* into *"All Products"* then the hierarchy would be redundant but not ragged. The *Measures* dimension has a hierarchy that is ragged, redundant and disconnected.

Denote by $\mathcal{A}_D$ the set of all non-trivial attributes on the dimension D. For two cubes C_1 and C_2, equivalent attributes on their common dimension D are simply identified (recall that we assume uniqueness of identification by name). The set of such attributes is $\mathcal{E}_D(C_1, C_2) = \mathcal{A}_D(C_1) \cap \mathcal{A}_D(C_2)$. For the attributes which are not in $\mathcal{E}_D(C_1, C_2)$, we assume for convenience of notation that they still have an equivalent attribute in the other cube with an empty attribute holder set.

With these conventions, the following rules apply to attributes:

a1) if a restriction or selection operator is applied, each of the attribute holder sets in a dimension is intersected with the projection of the restriction or selection result onto that dimension: $\alpha(\rho(C)) = \alpha(C) \cap \pi_i(\rho(C))$;

a2) if a cube is deflated, to form the attribute holder sets within a composite dimension $D\prime$, take each of the dimensions $D_i, i \in I$, that are being mapped to D', and produce Cartesian products of attribute holder sets in that dimension with $D^{I\setminus i}$:

$$\mathcal{A}_{D'}(\lambda(\nu_{I,J}, \omega)(C)) = \cup_{i\in I}\{\alpha \otimes D^{I\setminus i} \mid \alpha \in \mathcal{A}_i(C)\};$$

a3) if a dimension D_i is restored from a composite dimension D' in the process of inflation, take the projections onto that dimension of the attribute holder sets in the composite dimension from which that dimension is restored, then remove those that projected to the entire dimension:

$$\mathcal{A}_i(\mu(\nu_{I,J}, \omega)(C)) = \{\pi_i(\alpha) \mid \alpha \in \mathcal{A}_{D'}\};$$

a4) if a cube C_1 is brought within common context with another cube C_2 by means of a product with a subset $X \subset Q_2/Q_1$, the attributes of the product are inherited from those of C_1 and the attributes of X as a restriction of C_2;

a5) if two cubes C_1 and C_2 have a common dimension D, for their union, $\mathcal{A}_D(C_1 \cup C_2) = A_D(C_1) \cup A_D(C_2)$, and for each attribute γ, the corresponding set $\alpha(\gamma) = a(\gamma, C_1) \cup \alpha(\gamma, C_2)$. The union may produce a cycle, so resolution of references may be required;

a6) if the join operator is applied, use the above rules to transform the attributes on each step of the join definition algorithm, i.e. for the product, the union of cubes, and then for the selection.

Note that the deflation operation necessarily produces a redundant hierarchy in the composite dimension D' with the base and top being the product of respectively bases and tops of the dimensions that are mapped to D'(which does not hold for other levels and generations). Inflation however removes the redundancies acquired in the process of previous deflation.

Unions, intersections and other set operations on the attribute sets may result in cycles within the hierarchy graphs and hence may require resolution of references.

Application of the above set of rules yields the following

Proposition 5.2 *With the above rules, the operator algebra on MDS extends as a closed algebra to EMDS-1.*

The attributes and their holder sets are designated for two main purposes: classification of

metadata and derivation of data values. The former allows to navigate the cube and manipulate its views, the latter is used to produce values for cells corresponding to attributes from the values of their attribute holders. Attribute holder sets are also used as predefined subsets for restrictions.

Recall that we have been identifying the attribute and the dimension member to which it is associated. We now allow two attributes with possibly different holder sets to be associated with a dimension member if they serve different purposes, one - to navigate, the other - to derive values. If both holder sets coincide, the corresponding member is called *regular attribute member*, otherwise it is *irregular*.

The main difference between classification and derived data attributes is that the first type does not need to have any associated values, and even if there is a value, it is not necessarily a function of the attribute holders' values. Some relevant examples will be provided later.

On EMDS-1, two navigation operators can be defined in the following way. Associate with each dimensional attribute a boolean property called *Expanded*. The property controls the presentation of the cube data to the user by hiding all those attribute holders (members or cells) whose attribute has $Expanded = false$. It allows to view the data in the cube with more or less detail.

The *zoom-out (roll-up) operator* Z^- is the one that, given a set of attributes, assigns the value $false$ to the *Expanded* property of these attributes and their descendants (hides holders of these attributes and their descendants).

The *zoom-in (drill-down) operator* Z^+ is the one that, given a set of attributes, assigns the value *true* to their *Expanded* property (shows holders of these attributes).

Obviously, the two operators do not change the grid structure of the entire cube. They are navigational operators of purely presentational nature. In what follows, we assume that cycles in the hierarchies have been resolved, possibly by means of ordering. Otherwise implementation of the zooming operators may go into an infinite loop. Condensed or detailed views help achieve classification of data so that an analyst is provided with navigational means. However what values (summaries) he can expect to find in the cells is determined by computational dependencies.

Separation of navigational and computational properties of the attributes adds flexibility to the traditional approach and simplifies it.

Returning to our model example, we can declare *"My favorite locations"* a classification attribute and at the same time declare it a derived value attribute, for instance, by providing a formula

$$\text{“}My\ favorite\ locations\text{”} = \text{“}New\ York\text{”} + \text{“}San\ Francisco\text{”} + South.$$

In the above formula we do not specify the members of all the other dimensions that would fully qualify the cells, which means that the formula holds for all combinations of these members applied simultaneously to each term, i.e.

$$V(t, \text{“}My\ favorite\ locations\text{”}, p, m) = V(t, \text{“}New\ York\text{”}, p, m) + V(t, \text{“}San\ Francisco\text{”}, p, m) + V(t, South, p, m),$$
$$\forall t \in Time, p \in Product, m \in Measures.$$

Of course, the way to derive values does not necessarily have to be a sum. We leave until a later time the question how exactly the derived values for attributes are obtained from attribute holder values. In any case, a computational attribute must be provided together with a way to derive its value, for instance, some formula.

To illustrate the fact that attribute relations in an EMDS-1 model may be quite sophisticated, consider a more tricky example when a classification attribute's value is used to derive values of its holders, so that each of its holders is a derived value attribute for which the initial attribute is a holder. An example would be allocation of total budget value down to particular locations. Note that none of the cases creates a cycle in the hierarchy graph because, although the member is the same, the two corresponding attributes are different since they have different nature.

To simplify the structure of the hierarchy graph, we may consider two different graphs for each dimension, the classification (navigation) graph $G^{(n)}$ and the computational dependency graph $G^{(d)}$. However, if there are regular members, the information in these graphs is necessarily redundant, so it is left up to the database designer whether to combine these attributes. In many

known implementations, the graph $G^{(d)}$ is constructed by the OLAP provider implicitly based on the given set of formulas. For instance, in the *Measures* dimension of our sample cube, we could have provided formulas like $Profit = Sales - Expenses$, $Profit\% = 100 \times Profit \ / \ Sales$ instead of explicitly specifying attribute holder groups.

5.1 Further extensions of the model

Here we define EMDS-2, EMDS-3 and our final model, OLAP model.

Let γ be a dimensional attribute for a dimension D_i in a cube and let $\eta \subset D^{\Im\setminus i}$. We say that (γ, η) is a *partial* attribute if the attribute γ is only valid within $D_i \otimes \eta$. The set η is called the *definition domain* of the partial attribute. Every dimensional attribute may be considered as partial with $\eta = D^{\Im\setminus i}$. We require that if (γ, η_1) and (γ, η_2) are partial attributes and $\eta_1 \cap \eta_2 \neq \emptyset$, then $\eta_1 = \eta_2$. A dimensional attribute γ is said to have *global scope* for its cube if there exists a partition $D^{\Im\setminus i} = \cup_j \eta_j$ by pairwise disjoint sets such that for every j, the pair (γ, η_j) is a partial attribute on D_i. Otherwise it is said to have *local scope* and the cube itself is said to be *ragged*. Obviously usual dimensional attributes have global scope.

As an illustration when partial attributes may be needed, add to our sample cube's *Measures* dimension a level 0 member *"Customer preference"* reflecting customer votes taken for each level of members of *Products.* Votes for level 1 attributes, say, for *Soda* vs *Beer,* can not be derived from votes on their attribute holder level, because some customers may prefer sodas to beers in general but accidentally like most *Heineken* beer. So for $\eta = Time \otimes Locations \otimes$ "$Customer\ preference$", the attributes *Soda*, and *Beer* serve as classification attributes of level 1 but are trivial input level computational attributes. At the same time, with respect to the complement of η, summing up values of their attribute holders to derive their values makes sense[3]. Hence *Soda* and *Beer* would be partial attributes with global scope.

Partial attributes are used not only for computational purposes. We may declare *"My favorite locations"* as a local scope attribute, valid for the *West* slice and consisting there of *"San Francisco"* and *"Los Angeles"* and valid for the *East* slice and consisting there of *"New York"*.

Yet another example is hierarchy order changes for different regions. For instance, suppose for the *West* region, *Profit%* is derived from *Profit* and *Sales* values, and hence *Profit* is a descendant of *Profit%* in the hierarchy, but for the *East* region *Profit*'s value is derived from *Profit%* and *Sales* values, so there *Profit%* is a descendant of *Profit*. That justifies the use of partial attributes.

Note that since the definition domains of partial attributes are pairwise disjoint, no cycles appear in the hierarchy graph.

The rules $a1) - a6)$ require certain modifications for partial attributes:

b1) if a restriction or selection operator is applied, each of the partial attribute's holder sets in a dimension is intersected with the projection of the restriction or selection result onto that dimension: $\alpha(\rho(C)) = \alpha(C) \cap \pi_i(\rho(C))$. The definition domain is intersected with projection onto the rest of the dimensions: $\eta \cap \pi_{\Im\setminus i}(\rho(C))$. If this intersection is empty, the attribute is eliminated;

b2) if a cube is deflated, to form the partial attribute's holder sets within a composite dimension $D\prime$, take each of the dimensions $D_i, i \in I$, that are being mapped to D', and produce Cartesian products of attribute holder sets in that dimension with the projection of the corresponding deflation of η onto $D^{I\setminus i}$:

$$\mathcal{A}_{D'}(\lambda(\nu_{I,J}, \omega)(C)) = \cup_{i\in I}\{\alpha \otimes \pi_{I\setminus i}(\lambda(\eta)) \mid \alpha \in \mathcal{A}_i(C)\}.$$

The definition domain is the projection of the deflation of η onto the rest of the dimensions;

b3) if a dimension D_i is restored from a composite dimension D' in the process of inflation, take the projections onto that dimension of the attribute holder sets in the composite dimension from which that dimension is restored, then remove those that projected to the entire dimension:

$$\mathcal{A}_i(\mu(\nu_{I,J}, \omega)(C)) = \{\pi_i(\alpha) \mid \alpha \in A_{D'}\}.$$

The definition domain is the corresponding inflation of η;

[3]The way to derive the value for *Profit%* for level 1 members is diferent from summation. This case is discussed in more detail later on.

b4) if a cube C_1 is brought within common context with another cube C_2 by means of a product with a subset $X \subset Q_2/Q_1$, the partial attributes of the product are inherited from those of C_1 and the partial attributes of X as a restriction of C_2. The definition domains are formed as products $\eta \otimes X$ and $\eta \otimes Q_1$ respectively;

b5) if two cubes C_1 and C_2 have a common dimension D, for their union, $\mathcal{A}_D(C_1 \cup C_2) = A_D(C_1) \cup A_D(C_2)$, and for each partial attribute γ, the corresponding set $\alpha(\gamma) = a(\gamma, C_1) \cup \alpha(\gamma, C_2)$. If definition domains of γ in C_1 and C_2 overlap but do not coincide, further partition them to produce pairwise disjoint definition domains. The union may produce a cycle, so resolution of references may be required;

b6) if the join operator is applied, use the above rules to transform the partial attributes on each step of the join definition algorithm, i.e. for the product, the union of cubes, and then for the selection.

An *EMDS-2* model is an EMDS-1 model with partial attributes, i.e. with the additional property that each attribute multiindex has a definition domain. Since any dimensional attribute is a partial attribute, EMDS-1 becomes a particular case of EMDS-2.

The rules b1)-b6) imply the following

Proposition 5.3 *With the above rules, the operator algebra on EMDS-1 extends as a closed algebra to EMDS-2.*

We now add the next layer to EMDS-2.

Start with some notions. Let I be a multiindex. A *general (spatial) attribute* δ is a named entity corresponding to a subset β of D^I called the *attribute holder set or domain*. When $I = \{i\}$, a general attribute becomes a dimensional (possibly partial) attribute. General attributes can be added as members to cubes with composite dimensions whose map includes D_I. As an example, consider the domain "*Diet products at my favorite locations*" = *Diet* $\otimes$ "*My favorite locations*", where *Diet* is a dimensional attribute associated with the set {*Diet Pepsi,Diet Coke*} and *"My favorite locations"* is defined as above.

An *EMDS-3* model is an EMDS-2 model with general attributes, i.e. with the additional property that for each multiindex I there exists one or more general attributes. Since dimensional attributes are a particular case of general attributes, EMDS-2 becomes a particular case of EMDS-3.

The six above rules of dealing with the attribute holder sets may be modified to deal with attribute domains by replacing the index i with a multiindex I. Details are left to the reader. Thus one concludes that the operator algebra on EMDS-2 extends as a closed algebra to EMDS-3.

To see why spatial attributes are important, consider the problem of calculation order. Suppose we have computational dimensional attributes on some dimensions. The simplest possible scenario of computations of derived values is aggregation of data from attribute holders to their attribute. The elementary aggregation functions, such as SUM, MIN, MAX, COUNT, AVERAGE, etc., are commutative, so that the order of computations does not make a difference, otherwise one needs to determine the order in which the cell data values are computed. This problem can be resolved by using the dimension order (so results may change with dicing!) or an additional order indicator.

In terms of the dependency graphs the calculation order problem can be described as follows. In order to compute all the derived values in a cube C (or in its subcube), we need to form the product $G^{(d)}(C)$ of all the dimensional dependency graphs $G^{(d)}(D_i)$ which is necessarily redundant. The redundancy property would make the model *non-summarizable* since then such aggregation functions as SUM, COUNT, AVERAGE would produce incorrect results due to double counting (note that functions like MIN and MAX are applicable even to redundant graphs). So we need to remove some of the edges in $G^{(d)}(C)$ acquired in the process of taking products of $G^{(d)}(D_i)$. Determining which ones to remove means exactly the establishment of the calculation order. If the aggregation functions are not commutative or the data is non-numeric, this removal must be performed very carefully to produce the desired results.

Another problem arises if dimensions are built not by the canonical procedure but through pro-

viding formulas for computation of derived values. It may happen then, that the graph $G^{(d)}(C)$ would have cycles. In order for our computations not to enter an infinite loop, cycles need to be eliminated. There are many ways to resolve references, and this is a problem that every OLAP provider needs to face. Some of the providers avoid this problem by allowing formulas only on one dimension but such approach definitely reduces the model application domain. The times when elementary aggregation functions have been sufficient for most applications seem to be gone. Contemporary analytic problems require complex statistical analysis, curve fitting, stochastic process simulations, optimization, data mining, etc., which makes the dependency graphs highly sophisticated. Hence OLAP vendors should be ready to handle the compexity of the analytic applications domain.

To complete our definition of the *OLAP model*, which is based on EMDS-3, it remains to define one more operator - *aggregation* Δ- which acts on a cube or its subset by computing the values of those cells which correspond to derived data attributes. Obviously, the aggregation operator does not change the cube's metadata. The way to derive data values is not necessarily dimension based, i.e. formulas may involve slices of different dimensionality. In order to perform aggregation, cycles and redundancies in $G^{(d)}(C)$ have to be dealt with. It is up to the OLAP provider whether to perform aggregation for the whole cube before issuing any queries, perform aggregation on the result of the corresponding restriction, combine both ways or do nothing unless specifically instructed.

5.2 OLAP model summary

The multidimensional database (OLAP) model is an extension of the multidimensional data set by means of structuring metadata and forming computational dependency graphs. The operator algebra on MDS is extended as a closed algebra to the OLAP model and, additionally, zooming and aggregation operators are defined. Within this model, a query interested in any subset of the cube can be satisfied, and maximum possible dimensionality of the query result is maintained. The first property means that our model is at least as complete as the relational model (with fact tables). The second one ensures that it is suitable for analysis of multidimensional data.

6 Comparisons to other models

In this section we compare the suggested model to the ones in [AGS95], [LW96], [PJ99] and [GL96]. Some comparison of these models has been done in [PJ99], however that comparison was from a different perspective.

[AGS95] suggests a model in which a cube consists of k dimensions, and each cell contains an n-tuple of values. This closely corresponds to the FTR-2 relational approach. In our interpretation this can be modeled as a $(k+1)$-grid with a measures dimension of size n. The suggested operators in our terminology are dimensional restriction and various joins. Data-based restriction (selection) is introduced by discretization of the value domain, then making it into another dimension and applying dimensional restriction. Non-grid sets are augmented with empty cells to produce grids. Any other operation is described as a join (either a self-join or a join of two cubes), but is essentially dimensional, not general. Metadata is not structured. The model was designed to closely follow the relational model.

[LW96] suggests another extension of the relational algebra based on a grid with "groupings". Groupings are either ordered subsets of a dimension or ordered sets of cells in dimensional slices (i.e. products of a subset of one dimension against the rest). They provide means for dimensional restrictions. An aggregation operation on a set of cells with values is then defined as producing another group of cells with aggregated values along one dimension. An operation is defined to transfer a member from one dimension to another which means (in our terms) that a kind of composite dimension is created, but it is complemented to a grid with $NULL$ values for the additional cells. Union is defined for cubes which have the same dimensional projections except for one dimension in which projections do not intersect. Other operations are defined that basically convert a cube into FT-1 and vice versa. The model essentially operates on cells so it follows the relational algebra very closely and takes little advantage of dimensionality.

[PJ99] considers a particular application domain which deals with data of both multidimen-

sional and one-dimensional nature which is put together into one single multidimensional object (MO). A MO consists of n dimensions and data (facts). The model space is a n-grid but "duplicate values" are allowed. Each dimension has a hierarchy of levels and data is bound to levels via a set of relations. Hence the MO essentially reflects a well-known relational snowflake schema consisting of a fact table and dimension level tables. An additional feature added to the model is that some of the records may have a validity period stamp. Another layer is a numerical property of time chronons interpreted as probability. These layers are orthogonal to the multidimensional structure of the data. The operators defined on MO correspond to level based restrictions, sections of the model that produce duplicate values. Set operations on the facts without changing the grid are allowed as well as a close analog of the relational join that also produces duplicate values. An aggregation operator is also defined based on grouping by levels.

[GL96] defines a $n-$dimensional table (MDD) as a set of dimensions, measures and partitions (subsets of a dimension). The model space is the product of the dimensions, and each cell contains a tuple of measures. The tuple may be augmented to required size by adding NULLs. This is used to prove that the model is equivalent to FTR-2. Every operation on the model is performed by converting the MDD table to the corresponding FTR-2, performing a relational algebra operator there and converting the obtained fact table back to the MDD. Additionally, restructuring operators are defined which are similar to deflation and inflation of a grid. Finally, classification and summarization operators are defined which help express certain kinds of computations of derived values.

6.1 Summary

None of the mentioned models takes full advantage of the multidimensional structure of the data and as a result either considers a cube as a set of cells or slices it by members of one dimension. The former means that the corresponding models are equivalent to our model in the sense of completeness but lose the advantages of the multidimensional approach (reduction of descriptive parameters) whereas the latter means restricted expressive power of the queries and hence incompleteness with respect to our model.

The OLAP model suggested in this note possesses the folowing features:

- it is at least as complete as the relational model restricted to fact tables;
- it preserves the multidimensional structure of the data;
- it carries an expressive and closed operator algebra;
- it features structured metadata with distinction of computational and navigational attributes;
- it provides the basis for resolving computational dependencies in the process of data analysis.

7 Acknowledgements

The author is indebted to Igor Urisman for introducing him to the subject in 1998 and to Krishnan Subramaniam for numerous helpful discussions.

References

[AGS95] Agarwal, R., Gupta, A., and Sarawagi, S. Modeling multidimensional databases. IBM Research Report. IBM Almaden Research Center, September 1995.

[C93] Codd, E.F. Providing OLAP (online analytical processing) to user-analysts: an IT mandate. Technical report. E.F. Codd and Associates, 1993

[CCS93] Codd, E.F., Codd, S.B., and Salley, C.T. Beyond decision support. Computerworld, 27:30, July 1993.

[DT97] Datta, A., and Thomas, H. A conceptual model and algebra for on-line analytical processing in decision support databases. Proc. of the 7th Annual Workshop on ITS, 1997.

[GBLP97] Gray, J., Bosworth, A., Layman, A., Pirahesh, H. Data Cube: a relational aggregation operator generalizing group-by, cross-tab, and subtotals. Data Mining and Knowledge Discovery, 1, 1997, pp. 29-54.

[GL96] Gyssens, M., and Lakshmanan, L.V.S. A foundation for multidimensional databases. Proc. of the 22nd VLDB Conference, 1996.

[L98] Lehner, W. Modeling large scale OLAP scenarios. Proc. of the 6th International Conference on Extending Database Technology, 1998, pp. 153-167.

[LW96] Li, C., and Wang, X.S. A data model for supporting on-line analytical processing. Proc. of Conference on Information and Knowledge Management, 1996, pp. 699-717.

[K96] Kimball, R. The data warehouse toolkit. Wiley Computer Publishing, 1996.

[MK99] McGuff, F., and Kador, J. Developing analytical database applications. Prentice Hall, 1999.

[PJ99] Pedersen, T.B., and Jensen, C.S. Multidimensional data modeling for complex data. Proc. of ICDE, 1999.

[RS90] Rafanelli, M., and Shoshani, A. STORM: a statistical object representation model. Proc. of 5th Conference on Statistical and Scientific Database Management, 1990, pp. 14-29.

8 Appendix: Non-grid models

A grid is a convenient placeholder for the multidimensional data because of its homogeneity and isotropy properties. However in many cases the grid is very sparsely populated because either not all of the member combinations make sense or not all data is available. In the case of the former, it sometimes makes sense to use other (non-cube) shapes in modeling. For instance, consider a currency exchange database with the dimension $Measures = \{Bid, Ask\}$ and two dimensions *Currency1* and *Currency2* consisting of similar members like *USD, GBP, HFl*, etc. Within $Currency1 \otimes Currency2$ we may only need one half of the space to record the exchange rates. Therefore a "prism" would perhaps be a more appropriate model than a cube.

The way to visualize metadata and data in space is especially important when modeling the *Time* dimension in a multidimensional setting.

8.1 Time dimension modeling

We briefly describe a possible approach to handling the *Time* dimension or any other dimension with *periodic attributes*. Periodicity assumes some metric properties associated with the members of the dimension which allows to introduce some sort of distance with respect to which attributes become evenly distributed. For instance, minutes tend to occur every 60 seconds, hours - every 60 minutes, days - every 24 hours, quarters - every 3 months, years - every 4 quarters, and so on.

When we measure time, we usually use some round clock with periodically moving clock hands. This motivates the possibility of different visualization of time. Once we acquired the ability to look at things multidimensionally, nothing prevents us from taking the advantage of the flexibility such a view provides. We usually visualize multidimensional objects as cubes because they are simple. A cube is a shape we get as a result of the Cartesian product of line segments. But why restrict oneself to line segments? One might as well take a product of a circle and a segment to obtain a cylinder.

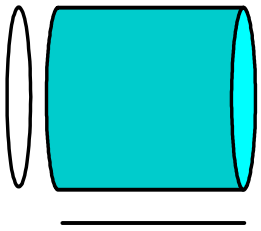


Figure 7: Cylinder as a product

For our *Time* dimension, we let the years go along the segment and let the quarters and months go along the circle.

The way to traverse the time cylinder is a spiral curve which provides linearization and hence time continuity. Periodicity breaks on the month level since months have a different number of days. However, if we want to further decompose days into periodic entities like hours, one has to imag-

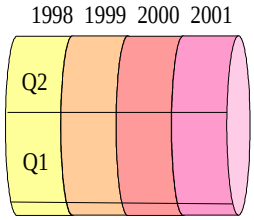


Figure 8: The Time cylinder

ine that what goes around the cylinder it is not a simple line but a spiral. Details and implementations of this approach are left to the reader.

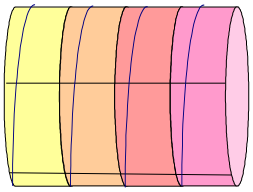


Figure 9: Time dimension traversal