A Foundation for Multi-Dimensional Databases

Marc Gyssens
Department WNI
University of Limburg (LUC)
B-3590 Diepenbeek, Belgium.
gyssens@charlie.luc.ac.be

Laks V.S. Lakshmanan
Department of Computer Science
Concordia University
Montreal, Quebec H3G 1M8, Canada
laks@cs.concordia.ca

Abstract

We present a multi-dimensional database model, which we believe can serve as a conceptual model for On-Line Analytical Processing (OLAP)-based applications. Apart from providing the functionalities necessary for OLAP-based applications, the main feature of the model we propose is a clear separation between structural aspects and the contents. This separation of concerns allows us to define data manipulation languages in a reasonably simple, transparent way. In particular, we show that the data cube operator can be expressed easily. Concretely, we define an algebra and a calculus and show them to be equivalent. We conclude by comparing our approach to related work.

The conceptual multi-dimensional database model developed here is orthogonal to its implementation, which is not a subject of the present paper.

1 Introduction

Currently, there is significant interest in multi-dimensional database systems for developing business analysis and decision support applications. Codd proposed the concept of On-Line Analytical Processing (OLAP) for rendering enterprise data in multi-dimensional perspectives, performing on-line analysis of data using mathematical formulas or more sophisticated statistical analyses, and consolidating and summarizing data [Cod93, CCS93]. It is believed that relational database technology is better suited for robust transaction management and ad-hoc querying. On the other hand, OLAP calls for sophisticated on-line analysis, something for which the traditional relational model offers little support. To fill this need, several vendors have already developed OLAP products including, e.g., Arbor Software's Essbase and Oracle Express to name just two. Many of these products suffer from the following limitations: (*i*) they are ad-hoc and they do not support a comprehensive "query" language similar to SQL; (*ii*) the user interaction is often limited to one operation at a time; (*iii*) viewing data in multi-dimensional perspectives involves treating certain attributes as *dimensional parameters* and the remaining ones as *measures*, and then analyzing them as a "function" of the parameters; many products treat dimensions and measures asymmetrically; and, finally, (*iv*) unlike for the relational model, there is no precise, commonly agreed, conceptual model for OLAP or the so-called multi-dimensional databases (MDD). Much of the success of relational databases has to do with the clear logical foundations for the data model first laid down by Codd and developed by numerous researchers subsequently.

In this paper, we make the following contributions.

1. We develop a simple conceptual model for OLAP or MDD.

2. We propose a four-layered architecture for OLAP query languages. We show that by separating the concerns of structure versus contents, we are able to develop a simple yet powerful algebra, and an equivalent calculus, both possessing features corresponding to all four levels of the architecture.

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

SALES			*TIME*						
			Year	1996			1997		
			Month	Jan	Feb	...	Jan	Feb	...
CATEGORY	*Part*	*City*	*(Cost, Sale)*						
	PC	Montreal		(5,6)	(5,7)	...	(4,6)	(4,8)	...
		Toronto		(5,7)	(5,8)	...	(4,8)	(4,9)	...
		⋮		⋮	⋮		⋮	⋮	
	Inkjet	Montreal		(7,8)	(7,9)	...	(6,9)	(6,8)	...
		New York		(6,9)	(6,9)	...	(5,8)	(5,9)	...
		⋮		⋮	⋮		⋮	⋮	
	⋮	⋮		⋮	⋮		⋮	⋮	

Figure 1: A sample two-dimensional table *Sales* with dimensions *Category* and *Time*. The associated parameter sets are $\{Part, City\}$ and $\{Year, Month\}$, respectively. The measure attributes are *Cost* and *Sale*.

To illustrate the above, we express in our algebra the popular data cube operator recently proposed by Gray et al. [GBLP95], as well as its useful and practically more attractive variants.

The paper is organized as follows. In Section 2, we present an informal introduction to our data model for MDD. In Sections 3 and 4, we present an algebra and a calculus for MDD and illustrate their expressive power via examples. We show both languages to be equivalent in expressive power. In Section 5, we conclude with a comparison with other recent related work, and we point out directions for future research. For brevity, we suppress all proofs and additional examples and details, all of which can be found in the full paper [GL97].

2 A Multi-Dimensional Data Model

From a conceptual standpoint, we contend that OLAP calls for the following four kinds of functionalities:

1. *Querying*: Ability to pose powerful ad-hoc queries through a simple and declarative interface.

2. *Restructuring*: Ability to restructure information in a multi-dimensional database exploiting the dimensionality of data and bringing out different perspectives of the data.

3. *Classification*: Ability to classify or group data sets in a manner appropriate for subsequent summarization.

4. *Summarization/Consolidation*: This is a generalization of the aggregate operators in standard SQL. In general, summarization maps multisets of values of a numeric type to a single, "consolidated" value.

We seek a conceptual model and query language that can support all the above functionalities, and allow them to interact with each other in a seamless manner.

The fundamental data structure of a multi-dimensional database is what we call an *n-dimensional table*. We first give the intuition behind it. We wish to be able to see values of certain attributes as a "function" of others, in whichever way suits us, exploiting possibilities of multi-dimensional rendering. Drawing on the terminology of statistical databases, we can classify the attribute set associated with the schema of a table into two kinds: *parameters* and *measures*. There is no a priori distinction between parameters and measures in that any attribute can play either role. An example of a two-dimensional table is given in Figure 1.

At the conceptual level, we want our model to remain as close to the standard relational model as possible, whence the rich body of theory and techniques well developed for the relational model will then accrue for the OLAP model. A natural way to realize this objective is to recognize that the multi-dimensionality of tables is an inherently *structural* feature, which is most significant when the table is rendered to the user. The actual *contents* of a table are essentially orthogonal to the associated structure, i.e., the distribution of attributes over dimensions and measure. Separating both features leads to a *relational* view of a table. For instance, the entry in the first (i.e., top left-most) "cell" of the table in Figure 1 containing the entry (5,6) corresponds to the tuple (`PC`, `Montreal`, `1996`, `Jan`, `5`, `6`) over the scheme $\{Part, City, Year, Month, Cost, Sale\}$ in a relational view of that table.

We now formalize the informal description given above, in the definition of an n-dimensional table.

SALES

rCATEGORY

Tid	*Part*	*City*
c1	PC	Montreal
c2	PC	Toronto
	⋮	
c6	Inkjet	Montreal
c7	Inkjet	New York
	⋮	

rTIME

Tid	*Year*	*Month*
t1	1996	Jan
t2	1996	Feb
	⋮	
t13	1997	Jan
t14	1997	Feb
	⋮	

rm

C.Tid	*T.Tid*	*Cost*	*Sale*
c1	t1	5	6
c1	t2	5	7
c6	t13	6	9
c6	t14	6	8

Figure 2: A conceptual view of the table of Figure 1. Actual storage structures and implementation need not be relational.

As is customary, we assume two infinite, disjoint sets of symbols: $\mathcal{N}$, a set of names, and $\mathcal{V}$, a set of values.

Definition 2.1 (Table Schemas and Instances) An n-dimensional *table schema* is a triple $\langle D, R, par\rangle$ where $D = \{d_1, \ldots, d_n\}$ is a set of dimension names, $R = \{A_1, \ldots, A_m\}$ is a set of attributes, and $par : D \to 2^{\{A_1,\ldots,A_m\}}$, such that

(*i*) for all $i, j = 1, \ldots, n$, $i \neq j$, $par(d_i) \cap par(d_j) = \emptyset$, and

(*ii*) $\bigcup_{d \in D} par(d) \subseteq R$.

We usually denote $par(d_i)$ by X_i.

Let $M = R - \bigcup_{1 \leq i \leq n} X_i$. An *instance* of an n-dimensional table schema $\langle D, R, par\rangle$ is a set of $n+1$ finite relations of the form $rd_1(Tid, X_1), \ldots, rd_n(Tid, X_n), rm(rd_1.Tid, \ldots, rd_n.Tid, M)$, such that

(*i*) the join $\pi_{Tid}(rd_1) \times \cdots \times \pi_{Tid}(rd_n)$ equals $\pi_{rd_1.Tid,\ldots,rd_n.Tid}(rm)$, i.e., for every combination of *Tid*-values in the relations $rd_1, \ldots, rd_n$, there is at least one corresponding tuple in *rm*, and every tuple in *rm* corresponds to some combination of *Tid*-values in the relations $rd_1, \ldots, rd_n$;

(*ii*) for all $i = 1, \ldots, n$, *Tid* is a key of the relation rd_i; and

(*iii*) for all $i, j = 1, \ldots, n$, $i \neq j$, $\pi_{Tid}(rd_i) \cap \pi_{Tid}(rd_j) = \emptyset$, i.e., the *Tid*-values in different relations rd_i and rd_j are disjoint.

A *multi-dimensional tabular database* (MDD) is a set of tables. □

In Definition 2.1, a table has a set of m attributes R and $n \geq 0$ dimensions $d_1, \ldots, d_n$ associated with it. Each dimension is characterized by a distinct subset of attributes from R, called the *parameters* of that dimension. Those attributes in R which are not parameters of any dimension are called the *measure* attributes of the table.

Intuitively, we can regard tuples in the relations $rd_1, \ldots, rd_n$ as "coordinates" in the dimensions $d_1, \ldots, d_n$, respectively. From this viewpoint, a table can intuitively be regarded as associating a set of tuples over the measure attributes with each point $(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ in the n-dimensional space, where, for $i = 1, \ldots, n$, $\mathtt{t}_i$ is the unique *Tid*-value associated with a tuple in relation rd_i. Conversely, for a tuple $(t_i, \vec{a}_i) \in rd_i$, we say that $\vec{a}_i$ is the X_i-tuple represented by t_i.

Clearly, classical relations correspond to 0-dimensional tables all of whose attributes are essentially measure attributes.

We should point out that the advocated conceptual view of multi-dimensional tables is completely independent of the storage structures or implementation strategies to be used for these tables.

Example 2.1 Consider the two-dimensional table schema *Sales* = $\langle$ {*Category*, *Time*}, {*Part*, *City*, *Year*, *Month*, *Cost*, *Sale*}, *par*$\rangle$, where *par*(*Category*) = {*Part*, *City*} and *par*(*Time*) = {*Year*, *Month*}. An instance of this schema, consisting of three relations *rCategory*(*Tid*, *Part*, *City*), *rTime*(*Tid*, *Year*, *Month*), and *rm*((*r*)*C*(*ategory*).*Tid*, (*r*)*T*(*ime*).*Tid*, *Cost*, *Sale*), is shown in Figure 2. It can be seen that this instance indeed satisfies the constraints in Definition 2.1. This table instance actually corresponds to the one shown in graphical form in Figure 1. □

We next show that every MDD table can be faithfully represented by a classical relation, and vice-versa, in a sense that we shall make precise below. We need this result in Section 3 to develop simple semantics for each of the classical algebraic operators.

Before we can show the above result, we must introduce the notion of *completion* of a relation with respect to a table schema.

Definition 2.2 (Completion) Let $r(A_1, \ldots, A_m)$ be a relation, and let $\mathcal{S} = \langle D, \{A_1, \ldots, A_m\}, par\rangle$ be a table schema. As usual, we denote $X_i = par(d_i)$. Then the *completion* of r with respect to $\mathcal{S}$, denoted $\overline{r}_{\mathcal{S}}$, is defined as the smallest relation satisfying (i) $r \subseteq \overline{r}_{\mathcal{S}}$ and (ii), for all combinations of tuple $\vec{a}_1, \ldots, \vec{a}_n$, for which, for $i = 1, \ldots, n$, $\vec{a}_i$ is in $\pi_{X_i}(r)$, but $(\vec{a}_1, \ldots, \vec{a}_n)$ is not in $\pi_{X_1,\ldots,X_n}(r)$, we have that $(\vec{a}_1, \ldots, \vec{a}_n, \vec{\bot})$ is in $\overline{r}_{\mathcal{S}}$. where $\vec{\bot}$ is a tuple of $\bot$ symbols of length $|M|$, matching the measure attributes. □

Intuitively, the relation r above is equivalent to its completion $\overline{r}_{\mathcal{S}}$: the additional $\bot$ symbols are merely there as "not-applicable nulls" to allow for a multi-dimension rendering of the data.

Let R be a relation scheme and, let $\mathcal{S} = \langle D, R, par\rangle$ be a table schema. Let $\mathcal{R}(R)$ be the class of all finite relations over R, and $\mathcal{T}(\mathcal{S})$ the class of all instances of the table schema $\mathcal{S}$. Finally, let $\overline{\mathcal{R}}(R)_{\mathcal{S}} = \{\overline{r}_{\mathcal{S}} \mid r \in \mathcal{R}(R)\}$. We say that the table schema $\mathcal{S}$ *faithfully represents* the relation scheme R (and vice-versa) provided there is a one-to-one correspondence from the class of tables $\mathcal{T}(\mathcal{S})$ to the class of relations $\overline{\mathcal{R}}(R)_{\mathcal{S}}$.

We now have the following theorem (proof omitted):

Theorem 2.1 *Let R be a relation scheme, and let $\mathcal{S} = \langle D, R, par\rangle$ be a table schema. Then R faithfully represents $\mathcal{S}$ (and vice versa).*

From the proof of Theorem 2.1, we retain for later use the existence of a one-to-one function f from $\mathcal{T}(\mathcal{S})$ to $\overline{\mathcal{R}}(R)_{\mathcal{S}}$, which can effectively be computed. Also its inverse, g can effectively be computed.

It follows from the above theorem that all the classical operators from relational algebra can directly be "imported" into the framework of multi-dimensional tables, by considering the tabular representation of the result of the operators applied to the relations represented by the input tabular database. This idea will be developed in the following section.

3 Algebra

In this section, we develop an algebra for multi-dimensional databases.

Before proceeding with the development of the actual algebra, we first illustrate with an example that a *direct* approach to defining the algebraic operators will lead to unnecessary complications.

Example 3.1 Consider the table shown in Figure 1. Suppose we wish to select the portion of the table corresponding to the performance of `PC` in the year `1996`. Operationally, the effect of such an operation amounts to what in the OLAP literature is referred to as "slicing and dicing." The resulting table is shown in Figure 3. A direct definition of this selection would involve selecting *Category* coordinates of the form (`PC`, `_C`) for various cities `_C`, and selecting *Time* coordinates of the form (`1996`, `_M`) for various months `_M`, and semijoining both with the measure relation *rm*. A simple relational selection becomes complicated if *directly* translated to MDD tables. Moreover, if the selection involves measure attributes, its definition would be somewhat different and asymmetric with the above. A direct definition of selection involving parameters and measures gets even more complicated. The reader can easily imagine the complexity in the case of a join between two MDD tables. □

SALES			*TIME*			
			Year	1996		
			Month	Jan	Feb	...
CATEGORY	*Part*	*City*	*(Cost, Sale)*	▨	▨	▨
	PC	Montreal	▨	(5,6)	(5,7)	...
		Toronto	▨	(5,7)	(5,8)	...
		⋮	▨	⋮	⋮	

Figure 3: Output of a "slicing and dicing" operation on the table of Figure 1. Slicing and dicing is just a special case of relational selection extended to MDD tables.

Thus, a direct approach to defining classical algebraic operators will result in complicated definitions. Our goal is to obtain a simple and elegant language close the conceptual view of MDD tables, rather than their formal definition. We shall obtain simple definitions of the operators of our algebra by exploiting the one-to-one correspondence between tables and relations established in Theorem 2.1, i.e, the functions f and g, thereby effectively separating contents from structural concerns. Let $\mathcal{S} = \langle D, R, par\rangle$ be a table schema. If τ is a table instance with schema $\mathcal{S}$, we denote its *relational representation* $f(\tau)$, a relation with scheme R, by $rep(\tau)$. Conversely, if r is a relation with scheme R, we denote its *tabular representation* $g(r)$, a table instance with schema $\mathcal{S}$, by $tab_{\mathcal{S}}(r)$.

Definition 3.1 (Classical Algebraic Operators)

1. *Unary Operators*: Let τ be a table instance with schema $\mathcal{S} = \langle D, R, par\rangle$, and let **op** be either the *selection* σ_C, the *projection* π_X, or the *renaming* $\rho_{B \leftarrow A}$, where C is a valid selection condition, defined as usual, X is a subset of the attribute set of the table, and A and B are attribute names. We define $\mathbf{op}(\tau) = tab_{\mathcal{S}}(\mathbf{op}(rep(\tau)))$.

2. *Union, Intersection, and Difference*: Let τ_1 and τ_2 be table instances, both with schema $\mathcal{S}$, and

SALES		*COMPONENT*			
		Part	PC	Inkjet	...
LOCATION	*City*	*(Year, Month, Cost, Sale)*	▨	▨	▨
	Montreal	▨	(1996, Jan, 5 ,6)	(1996, Jan, 5 ,6)	...
	Montreal	▨	(1996, Feb, 5, 7)	(1996, Feb, 5, 7)	...
	⋮	▨	⋮	⋮	
	Montreal	▨	(1997, Jan, 4 ,6)	(1997, Jan, 4 ,6)	...
	Montreal	▨	(1997, Feb, 4, 8)	(1997, Feb, 4, 8)	...
	⋮	▨	⋮	⋮	
	Toronto	▨	(1996, Jan, 5 ,7)	$\perp$	...
	Toronto	▨	(1996, Feb, 5, 8)	$\perp$	...
	⋮	▨	⋮	⋮	

Figure 4: Output of the expression $\mathbf{fold}^{Time}(\mathbf{unfold}^{Location}_{City}(\mathbf{unfold}^{Component}_{Part}(\mathbf{fold}^{Category}(Sales))))$ applied to the input table *Sales* of Figure 1. The resulting table shows *Year*, *Month*, *Cost*, and *Sale* as a measure of *Part* and *City*. It has two dimensions *Location* and *Component*, with parameters *City* and *Part*, respectively.

let **op** be either the *union* $\cup$, the *intersection* $\cap$, or the *difference* $\setminus$. We define τ_1 **op** $\tau_2 = tabs_{\mathcal{S}}(rep(\tau_1)$ **op** $rep(\tau_2))$, where $\mathcal{S} = \mathcal{S}_1 = \mathcal{S}_2$.

3. *Cartesian Product*: Let τ_1 and τ_2 be two tables with schemas $\mathcal{S}_1 = \langle D_1, R_1, par_1\rangle$ and $\mathcal{S}_2 = \langle D_2, R_2, par_2\rangle$, and assume $D_1 \cap D_2 = \emptyset$ and $R_1 \cap R_2 = \emptyset$[1]. We define the schema of the *Cartesian product* $\tau_1 \times \tau_2$ as $\mathcal{S} = \langle D_1 \cup D_2, R_1 \cup R_2, par_1 \cup par_2\rangle$ and the instance as $tabs_{\mathcal{S}}(rep(\tau_1) \times rep(\tau_2))$. □

The reader should notice (i) the simplicity of the operator definitions compared with the direct approach and (ii) the symmetric treatment of parameters and measures.

In addition to the above operators which mainly alter the *contents* of tables, we introduce two restructuring operators which only affect the *structure* of tables. Using these, it is possible to drop or add a dimension, to rename a dimension, to drop or add parameters from or to a dimension, or to change the status of an attribute from parameter to measure or vice-versa, while preserving the information content.

Definition 3.2 (Restructuring Operators)

1. *Unfold*: Let τ be a table with schema $\mathcal{S} = \langle D, R, par\rangle$, let d be a new name of $\mathcal{N}$ appearing nowhere else in τ, and let $X \subseteq M$ be a set of measure attributes. We define $\mathbf{unfold}^d_X(\tau)$ as a table with schema $\mathcal{S}' = \langle D \cup \{d\}, R, par'\rangle$, where, for all d_i in D, $par'(d_i) = par(d_i)$, and $par'(d) = X$, and with instance $tabs_{\mathcal{S}'}(rep(\tau))$.

2. *Fold*: Let τ be a table with schema $\mathcal{S} = \langle D, R, par\rangle$, and let d be one of the dimensions of D. We define $\mathbf{fold}^d(\tau)$ as a table with schema $\mathcal{S}' = \langle D \setminus \{d\}, R, par'\rangle$, where, for all d_i in $D \setminus \{d\}$, $par'(d_i) = par(d_i)$, and with instance $tabs_{\mathcal{S}'}(rep(\tau))$. □

Again, note the simplicity of operator definitions and the uniform treatment of parameters and measures. The following example illustrates the operations defined above.

Example 3.2 Figure 4 illustrates an application of the **fold** and **unfold** operators to our running example. □

Our next operation concerns classification. Classification is a generalization of the familiar SQL **group by** operator. The following example presents a typical practically arising query involving classification. Since the concept of classification is orthogonal to the structure of a table, we give a relational example.

Example 3.3 Consider a relation *Stocks* with scheme $\{Ticker, Day, Price, Vol\}$, containing the closing prices and volume of trading of various stocks in the New York Stock Exchange. A typical practically arising query is *Find the 3-day moving average closing price for each stock*. Even though this query involve aggregation, notice that it also involves *classifying* the data into various groups according to certain criteria, before aggregation is applied. Concretely, the above query involves classification using a window of three days. □

[1]Because of the presence of a renaming operator, there is no loss of generality in assuming that both attribute sets do not overlap. The restructuring operators we introduce next also allow us to rename dimensions.

Other instances of classification involving windows of variable width (e.g., cumulative averages) or windows of data-dependent width (e.g., average stock prices corresponding to bullish periods) are also covered by our framework.

We next formalize the notion of classification, addressing it first in the context of relations.

Definition 3.3 (Classification on Relations) Let $R = \{A_1, \ldots, A_m\}$ be a relation scheme, and let $X = \{A_1, \ldots, A_k\}$ be an arbitrary subset of R. A *classification function* over X, for relations over R, is a function

$$f : \mathcal{R}(R) \times dom(A_1) \times \cdots \times dom(A_k) \rightarrow 2^{dom(A_1) \times \cdots \times dom(A_k)}.$$

Let r be a relation with scheme R, and let f be a classification function over X, for relations over R. We define the result of the *classification operator*, $K(r, f)$, as the relation with scheme

$$(f.A_1, \ldots, f.A_k, A_1, \ldots, A_k, A_{k+1}, \ldots, A_m)$$

and instance

$$\{(a_1, \ldots, a_k, a'_1, \ldots, a'_k, a'_{k+1}, \ldots, a'_m) \mid (a'_1, \ldots, a'_k) \in f(r, a_1, \ldots, a_k) \wedge (a'_1, \ldots, a'_k, a'_{k+1}, \ldots, a'_m) \in r\}.$$

□

Classification essentially maps tuples of a relation to one or more (thus, not necessarily disjoint) groups. Intuitively, we can think of the attributes $f.A_1, \ldots, f.A_k$ as corresponding to the "group id." Thus, a tuple $(a_1, \ldots, a_k, a'_1, \ldots, a'_k, a'_{k+1}, \ldots, a'_m)$ in the classified relation says that the tuple $(a'_1, \ldots, a'_k, a'_{k+1}, \ldots, a'_m)$ of r belongs to the group whose "id" is $(a_1, \ldots, a_k)$.

Example 3.4 The classification part of the query of Example 3.3 can be expressed as follows. Define a classification function $f_1 : \mathcal{R}(\{Ticker, Day, Price, Vol\}) \times dom(Day) \rightarrow 2^{dom(Day)}$ by

$$f_1(r, \_Day) = \{(\_Day') \mid (\_Day' = \_Day) \vee (\_Day' = succ(\_Day)) \vee (\_Day' = succ(succ(\_Day)))\}.$$

Then apply classification using f_1 to *Stocks*. In the resulting relation, rename $f_1.Day$ to *Start*. □

To allow for efficient implementation, we shall only consider *first-order definable* classification functions, i.e., functions definable in first-order logic with equality over the vocabulary $r, <$, with r the predicate of the relation to which classification is applied.

We now extend classification from relations to tables:

Definition 3.4 (Classification on Tables) Let τ be a table instance with schema $\mathcal{S} = \langle D, R, par \rangle$, let $X = \{A_1, \ldots, A_k\}$ be an arbitrary subset of $R = \{A_1, \ldots, A_m\}$, and let f be a classification function over X, for relations over R. Define $\mathcal{S}' = \langle D, R \cup \{f.A_1, \ldots, f.A_k\}, par' \rangle$, such that, for $i = 1, \ldots, m$, $f.A_i \in par'(d)$, for some d in D, if and only if $A_i \in par(d)$. We define the result of the *classification operator*, $K(\tau, f)$, as $tab_{\mathcal{S}'}(K(rep(\tau), f))$. □

Finally, we consider summarization/consolidation, which includes not only applications of functions such as *max*, *min*, *avg*, *sum*, *count* to multisets of values defined by groups of tuples, but also statistical functions such as *variance* and *mode*, and business calculations such as *proportions* and *quartiles*.

To avoid the explicit handling of multisets, following Klug [Klu82], we model summarization functions as mapping sets of tuples of values to individual values. As numerical type, we consider the rational numbers, thus making it possible to apply the usual arithmetic operations $+$, $-$, $\times$, and $/$.

As for classification, we first define summarization on relations:

Definition 3.5 (Summarization on Relations) Let $R = \{A_1, \ldots, A_m\}$ be a relation scheme, and let $X = \{A_1, \ldots, A_k\}$ be an arbitrary subset of R. A *summarization function* over X, for relations over R, is a function

$$g_{B \leftarrow A_j} : 2^{dom(A_{k+1}) \times \cdots \times dom(A_m)} \rightarrow dom(B),$$

where A_j, for some j, $k+1 \leq j \leq m$, is one of the numeric attributes of the application, over which the actual summarization takes place, and which is of the same type as B, the attribute corresponding to the result. We assume $B \notin \{A_1, \ldots, A_k\}$.

Let r be a relation with scheme R, and let g be a summarization function as defined above. We define the result of the *summarization operator*, $A(r, g)$, as the relation with scheme $\{A_1, \ldots, A_k, B\}$ and instance

$$\{(a_1, \ldots, a_k, b) \mid b = g(\{(a_{k+1}, \ldots, a_m) \mid (a_1, \ldots, a_k, a_{k+1}, \ldots, a_m) \in r\})\}.$$

□

Note that the exact definition of g is left open to the application. In order to allow efficient implementation, we will again require that the summarization functions be first-order definable, in the extended sense of Grädel and Gurevich [GG95], i.e., in a vocabulary including the standard arithmetic operations $+$, $-$, $\times$, and $/$, and repeated additions $\sum$ and multiplications $\prod$ over a set of items (which includes counting). Our next examples illustrate first-order definable summarization functions.

Example 3.5 Consider a relation r with scheme $\{Part, City, Sale, Cost\}$. Assume that r in general contains several tuples for each part (corresponding to different cities). Consider the function

$$g_{Totprofit \leftarrow Sale} : 2^{dom(City) \times dom(Sale) \times dom(Cost)} \rightarrow dom(Totprofit),$$

defined by $g(S) = \sum_{(\_T,\_S,\_C) \in S} (\_S - \_C)$. Now, $A(r, g)$ is the relation with scheme $\{Part, Totprofit\}$ and instance $\{(\_P, \_Tp) \mid \_Tp = g(\{(\_T, \_S, \_C) \mid (\_P, \_T, \_S, \_C) \in r\})\}$. Thus, the aggregation performed computes part-wise total profit. □

Example 3.6 Consider again the query of Example 3.3. We illustrate how, from the classified relation computed in Example 3.4, we can obtain the final answer to the query. Let s be the classified relation. We recall its scheme, $\{Start, Day, Ticker, Price, Vol\}$. Let $avgpr_{Avgpr}$ be the summarization function with domain $2^{dom(Day) \times dom(Ticker) \times dom(Price) \times dom(Vol)}$ and range $dom(Avgpr)$, and defined by $avgpr(S) = (1/|S|) \sum_{(\_D,\_T,\_P,\_V) \in S} \_P$ ($|S|$ denotes the cardinality of the set S). Then the query under consideration can be expressed as $A(s, avgpr)$. □

We now extend summarization to tables:

Definition 3.6 (Summarization on Tables) Let τ be a table instance with schema $\mathcal{S} = \langle D, R, par \rangle$, let $R = \{A_1, \ldots, A_m\}$, $X = \{A_1, \ldots, A_k\} \subseteq R$ be the set of all parameters of τ, i.e., $X = \bigcup_{d \in D} par(d)$, and let $g_{B \leftarrow A_j}$, for some j, $k+1 \leq j \leq m$, and $B \notin \{A_1, \ldots, A_k\}$, be a summarization function over X, for relations over R. Let $\mathcal{S}' = \langle D, X \cup \{B\}, par \rangle$.

We define the result of the *summarization operator*, $A(\tau, g)$, as $tab_{\mathcal{S}'}(A(rep(\tau), g))$. □

Note the structure of the output table is the same as the input table, as far as the dimensions and parameters are concerned. The only change is that sets of measure tuples are summarized according to the aggregate function and transformed to single values, as indicated above.

We conclude this section with a pair of examples illustrating the power of the algebra. Thereto, we show that popular OLAP operators such as *data cube* and *monotone roll-up* [GBLP95] can be neatly expressed in our algebra. To our knowledge, this is the first time that such operators are being formally shown to be expressible in a rigorous algebra.

Example 3.7 (Data Cube) Let τ be a table with schema $\mathcal{S} = \langle \{D_1, \ldots, D_{m-1}\}, \{A_1, \ldots, A_m\}, par \rangle$ with, for $i = 1, \ldots, m-1$, $par(d_i) = \{A_i\}$. Thus, A_m is the only measure attribute. (The generalization to arbitrary table schemas is obvious.) The following algebraic program precisely captures the data cube operator applied to τ.

1. Consider the constant table instance *All* over $\mathcal{S}$, whose relational representation consists of the single m-tuple $(\mathtt{All}, \ldots, \mathtt{All}, \perp)$. The constant `All` does not appear anywhere else in the database.
2. Form the table $All \cup \tau$.
3. Let $X = \{A_1, \ldots, A_{m-1}\}$, and let r be a relation with scheme R. Define the following classification function over X, for relations over R:

   $$f(r, \mathtt{a}_1, \ldots, \mathtt{a}_{m-1}) = \{(\mathtt{b}_1, \ldots, \mathtt{b}_{m-1}) \mid \exists \mathtt{b}_m : (\mathtt{b}_1, \ldots, \mathtt{b}_m) \in r \land \forall i = 1, \ldots, m-1 : (\mathtt{b}_i = \mathtt{a}_i) \lor (\mathtt{a}_i = \mathtt{All})\}.$$

   Using this function, compute the classified table as $K(f, \tau)$. At this point, for $i = 1, \ldots, m-1$, dimension d_i has two parameters, A_i and $f.A_i$, $i = 1, \ldots, m-1$. Using **unfold** and **fold**, "push" the parameters $A_1, \ldots, A_{m-1}$ into the measure.
4. Define the summarization function

   $$g_{A_m \leftarrow A_m} : 2^{dom(A_1) \times \cdots \times dom(A_m)} \rightarrow dom(A_m)$$

   by $g(S) = \sum_{(\mathtt{b}_1, \ldots, \mathtt{b}_m) \in S,\ \mathtt{b}_m \neq \perp} \mathtt{b}_m$. Using this, compute the aggregate table as $A(K(f, \tau), g)$, the scheme of which is $\mathcal{S}$ and the relational representation of which is

   $$\{(\mathtt{a}_1, \ldots, \mathtt{a}_{m-1}, \mathtt{a}_m) \mid \mathtt{a}_m = g(\{(\mathtt{b}_1, \ldots, \mathtt{b}_m) \mid (\mathtt{a}_1, \ldots, \mathtt{a}_{m-1}, \mathtt{b}_1, \ldots, \mathtt{b}_m) \in K(f, \tau)\})\}.$$
5. Rename the parameter attributes $f.A_i$ to A_i.

The computation above is illustrated in pictorial form in Figure 5, for the case $m = 3$. □

As another illustration of the expressive power of our algebra, we show next that "monotone restrictions" of data cube are also expressible in the algebra.

Example 3.8 (Monotone Roll-Up) In many applications, only certain fragments of the data cube are of interest. Generally, if X is the set of parameters of a table and Y is a subset of X, then only aggregates with respect to Y and all its subsets may be of interest. Clearly, this is a subset of the data cube, which is sometimes referred to as the *monotone roll-up*. This can be expressed by modifying the construction of Example 3.7, by changing the single tuple of the constant table *All* to $\perp$ in attributes not in Y, and adapting the subsequent steps accordingly. □

Finally, we point out the above techniques are also valid for applications of data cube or monotone roll-up with other summarization operators than *sum*.

TABLE		DIMENSION2					
		A2	a21	a22	...	a2q	All
DIMENSION1	A1	M	▨	▨	▨	▨	▨
	a11	▨	v11	v12	...	v1q	⊥
	a12	▨	v21	v22	...	v2q	⊥
	⋮	▨					
	a1p	▨	vp1	vp2	...	vpq	⊥
	All	▨	⊥	⊥	...	⊥	⊥

(*a*)

TABLE		DIMENSION2					
		A2	a21	a22	...	a2q	All
DIMENSION1	A1	M	▨	▨	▨	▨	▨
	a11	▨	v11	v12	...	v1q	Σ_jv1j
	a12	▨	v21	v22	...	v2q	Σ_jv2j
	⋮	▨					
	a1p	▨	vp1	vp2	...	vpq	$\Sigma_j$vpj
	All	▨	$\Sigma_i$vi1	$\Sigma_i$vi2	...	$\Sigma_i$viq	$\Sigma_{i,j}$vij

(*b*)

Figure 5: Illustration of the data cube computation in Example 3.7 for $m = 3$: (a) the table resulting from Step 2; (b) the final table resulting from Step 5.

$$(\mathit{Profit}[\mathit{Component}(\mathit{Part} : \mathtt{\_P}), \mathit{Location}(\mathit{City} : \mathtt{\_C}) \rightarrow (\mathit{Year} : \mathtt{\_Y}, \mathit{Month} : \mathtt{\_M}, \mathit{Cost} : \mathtt{\_Ct}, \mathit{Sale} : \mathtt{\_S})] \mid$$
$$\mathit{Sales}[\mathit{Category}(\mathit{Part} : \mathtt{\_P}, \mathit{City} : \mathtt{\_C}), \mathit{Time}(\mathit{Year} : \mathtt{\_Y}, \mathit{Month} : \mathtt{\_M}) \rightarrow (\mathit{Cost} : \mathtt{\_Ct}, \mathit{Sale} : \mathtt{\_S})] \wedge \mathtt{\_S} > \mathtt{\_Ct}).$$

$$\mathit{avg}_{\mathit{Avgsale} \leftarrow \mathit{Sale}}(\mathit{Cumulative}[\mathit{Component}(\mathit{Part} : \mathtt{\_P}), \mathit{Location}(\mathit{City} : \mathtt{\_C}), \mathit{Interval}(\mathit{Upto} : \mathtt{\_M}) \rightarrow$$
$$(\mathit{Month} : \mathtt{\_M}', \mathit{Sale} : \mathtt{\_S})] \mid$$
$$\mathit{Sales}[\mathit{Category}(\mathit{Part} : \mathtt{\_P}, \mathit{City} : \mathtt{\_C}), \mathit{Time}(\mathit{Year} : 1996, \mathit{Month} : \mathtt{\_M}') \rightarrow (\mathit{Sale} : \mathtt{\_S})] \wedge$$
$$\mathit{Sales}[\mathit{Time}(\mathit{Year} : 1996, \mathit{Month} : \mathtt{\_M})] \wedge \mathtt{\_M}' \leq \mathtt{\_M}).$$

$$\mathit{prop}_{\mathit{Share} \leftarrow \mathit{Sale}}(\mathit{MarketShare}[\mathit{Category}(\mathit{Part} : \mathtt{\_P}), \mathit{Time}(\mathit{Year} : \mathtt{\_Y}) \rightarrow (\mathit{Type} : \mathtt{\_T}, \mathit{Month} : \mathtt{\_M}, \mathit{City} : \mathtt{\_C}, \mathit{City}' : \mathtt{\_C}',$$
$$\mathit{Sale} : \mathtt{\_S}, \mathit{Part} : \mathtt{\_P}', \mathit{Month}' : \mathtt{\_M}', \mathit{Sale}' : \mathtt{\_S}')] \mid$$
$$\mathit{Sales}[\mathit{Category}(\mathit{Part} : \mathtt{\_P}', \mathit{City} : \mathtt{\_C}'), \mathit{Time}(\mathit{Year} : \mathtt{\_Y}, \mathit{Month} : \mathtt{\_M}') \rightarrow (\mathit{Sale} : \mathtt{\_S}')] \wedge$$
$$\mathit{Types}[\rightarrow (\mathit{Part} : \mathtt{\_P}, \mathit{Type} : \mathtt{\_T})] \wedge$$
$$\mathit{Sales}[\mathit{Category}(\mathit{Part} : \mathtt{\_P}, \mathit{City} : \mathtt{\_C}), \mathit{Time}(\mathit{Year} : \mathtt{\_Y}, \mathit{Month} : \mathtt{\_M}) \rightarrow (\mathit{Sale} : \mathtt{\_S})].$$

Figure 6: The calculus expressions of Examples 4.1, 4.2, and 4.3.

4 Calculus

In this section, we propose a calculus for multi-dimensional databases equivalent to the algebra presented in Section 3. Whereas in the algebra, we separated the concerns of contents and structure, we provide a unified framework for the calculus, doing justice to its logic-based nature.

A query in our calculus is of the form $(\mathcal{A} \mid F)$, with $\mathcal{A}$ the *output expression* and F the *input expression*. Intuitively, F asserts the conditions that must be satisfied by the MDD database, and thus induces a set of "answer substitutions." The output expression $\mathcal{A}$ dictates how to structure these answer substitutions as the table, which is the result of the query.

Input expressions are constructed from built-in predicates and *database atoms*, of the form

$$\tau[d_1(\vec{A}_1 : \vec{\mathtt{T}}_1), \ldots, d_n(\vec{A}_n : \vec{\mathtt{T}}_n) \rightarrow (\vec{B} : \mathtt{T})],$$

asserting that in table τ, in one of the cells defined by the coordinates $(\vec{A}_i : \vec{\mathtt{T}}_i)$ in dimension d_i, $1 \leq i \leq n$, the values of the measure attributes $\vec{B}$ are $\vec{\mathtt{T}}$. In general, the specified dimensions, together with their specified parameter values, determine a *set* of cells. The expression states that in *one* of those cells the values of the specified measure attributes are as indicated. In output expressions, database atoms are also used, but then they *completely* describe the schema of the table in which the set of answer substitutions to the input expressions must be cast. Finally, we also allow summarization functions to be applied to the result of queries, according to the syntax $g_{B \leftarrow A}(\mathcal{A} \mid F)$, the intuitive meaning of which is as in the algebra, when the argument is interpreted as the answer table resulting from the query $(\mathcal{A} \mid F)$.

For a formal definition of syntax and semantics, we have to refer to the full version of this paper [GL97]. Here, we limit ourselves to bringing out its flavor by means of some examples.

Example 4.1 Consider the table of Figure 1. The first calculus query in Figure 6 computes a table containing those cells of the input table where the *Sales* is strictly larger than the *Cost* value, with the resulting table having two dimensions *Component* and *Location*, corresponding to *Part* and *City*, respectively. □

Example 4.2 Consider again the table of Figure 1. The second calculus query in Figure 6 computes monthly cumulative average sales for each part and city, for 1996. In this expression,

$$avg_{Avgsale \leftarrow Sale} : 2^{dom(Month) \times dom(Sale)} \rightarrow dom(Avgsale)$$

is the first-order summarization function defined by $avg(S) = (1/|S|)\sum_{(\mathtt{m}',\mathtt{s}) \in S} \mathtt{s}$. □

Example 4.3 Consider again the table of Figure 1. Consider the query which computes, for each part, its market share within its part type, in a given year, over all cities. Assume that a relation *Types* with scheme {*Part*, *Type*} (which is a 0-dimensional table!) is available. The required calculus expression is the last one shown in Figure 6. In this expression, the summarization function $prop_{Share \leftarrow Sale}$ computes the proportion of the total sales of each part within its part type. This function is definable in the first-order language of the calculus, as follows. Consider the coordinates *Part* : p in dimension *Category*, *Year* : y in dimension *Time* of the intermediate table *MarketShare*. Assume part p is of type t. Then the set of measure tuples associated with these coordinates is the set S consisting of all tuples $(\mathtt{t},\mathtt{m},\mathtt{s},\mathtt{p}',\mathtt{m}',\mathtt{s}')$ such that in month m, the sales of p is s, and p′ is any part of type t whose sales in some month m′ is s′. Thus

$$prop_{Share \leftarrow Sale}(S) = \frac{\sum_{\exists \mathbf{x},\mathbf{y},\mathbf{z}:(t,\mathbf{m},\mathbf{s},\mathbf{x},\mathbf{y},z) \in S}(\mathbf{s})}{\sum_{(\mathtt{t},\mathtt{m},\mathtt{s},\mathtt{p}',\mathtt{m}',\mathtt{s}') \in S}(\mathtt{s}')}$$

is a sound definition of the required summarization function. Note the use of existential quantifiers (a first-order construct) for the elimination of duplicates. The numerator computes the total sales of part p, while the denominator corresponds to the total sales of all parts of the same type as p. □

The above examples show that quite sophisticated queries can be expressed easily and elegantly in the calculus.

We prove the following result in the full paper [GL97].

Theorem 4.1 *The algebra and calculus are equivalent in expressive power.*

Consequently, operators such as data cube or roll-up can also be expressed in the calculus.

5 Discussion

Racing ahead of academic research, several industries have already put out their own OLAP/MDD engines, and two kinds of approaches have come forth. The first, so-called MOLAP approach (for multi-dimensional OLAP), is based on building separate dedicated engines based on multi-dimensional storage strategies. Arbor Software's Essbase is an example of this. The second, so-called ROLAP (for relational OLAP), approach is based on adapting relational database systems. Red Brick and Oracle are some examples. There is a wealth of industry white papers on the subject of OLAP (e.g., see [Arb93, Ban95, Col95, Eri95, Fin95, Red95]).

In terms of research, one of the significant developments is the proposal by Gray et al. [GBLP95] of the data cube operator. Since then, much work has gone into finding efficient data cube algorithms [A+96, HRU96]. Relatively little work has gone into modeling. The only two proposals we are aware of are Agrawal et al. [AGS95] and Li and Wang [LW96]. Since our main contribution is a conceptual model and query languages for MDD, we compare our work with the latter two papers.

Both research teams work with multi-dimensional tables, called cubes, having parameters and measures. However, restrictions are imposed on either the number of parameters per dimension or the number of measure attributes. Also, they see a table instance as a *function* from the Cartesian product of the domains of these parameters to the Cartesian product of the measure domains, as a consequence of which parameters and measures are treated asymmetrically. This in turn leads to not separating concerns of structure and contents. In [AGS95], this leads to complicated operator definitions, whereas, in [LW96], this leads to a framework in which cubes and plain relation are treated asymmetrically. An elegant idea in Li and Wang's model are the so-called grouping relations, which are used as classification in our model, to prepare cubes for various aggregations.

Another recent work relevant to ours is Gyssens et al. [GLS96] which proposes a two-dimensional data model and a complete algebraic query language for all conceivable restructuring transformations. Although it has some theoretical relevance for OLAP, it does not address important features classification and summarization necessary for practical applications.

Finally, we observe there are apparent similarities between our and the so-called *star schema* [Star95], as well as with *statistical databases* [Sho82, Sho96]. In our terminology, these models also suffer from an asymmetric treatment of parameters and measures.

In summary, our contribution is providing a comprehensive, generic conceptual model for MDD, which is neutral with respect to important design decisions, such as whether a ROLAP or a MOLAP approach should be adopted for realizing OLAP functionalities. Ours is the first model for MDD/OLAP where issues related to structure are separated from those related to contents. This has resulted in a simple yet powerful algebra, much simpler and better understood than the ones proposed above, as well as, for the first time, an equivalent calculus. Also for the first time, important operators like data cube and variants have been shown to be formally expressible in a rigorous algebra.

Several questions still remain open. (*i*) Properties of the algebraic operators need to be studied with a view to finding efficient query rewrite strategies. (*ii*) The expressive power of the equivalent languages proposed here needs to be characterized in a language-independent manner. (*iii*) We need an efficient implementation of the proposed languages, possibly exploiting techniques like multi-dimensional indexing from spatial and statistical databases. We are currently working on these issues.

Acknowledgment

This research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada and from the University of Limburg (LUC), and was done while the second author was visiting LUC as part of his sabbatical.

References

[AGS95] Agrawal, R., Gupta, A., and Sarawagi, S. "Modeling multi-dimensional databases," *IBM Research Report*, IBM Almaden Research Center, September 1995.

[A+96] Agrawal R., et al., "On the computation of multidimensional aggregates," in *Proceedings 22nd International Conference on Very Large Databases*, (Mumbai, India, September 1996).

[Arb93] Arbor Software Corporation, Sunnyvale, CA, *Multidimensional Analysis: Converting Corporate Data into Strategic Information.*, white paper, 1993.

[Ban95] Bansal, S.K., "Real world requirements for decision support—implications for RDBMS," *SIGMOD Record*, 24:2, 1995, p. 448.

[Cod93] Codd, E.F., "Providing OLAP (on-line 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.

[Col95] Colliat, G., "OLAP, relational and multidimensional database systems," *Technical Report*, Arbor Software Corporation, Sunnyvale, CA, 1995.

[Eri95] Erickson, C.G., "Multidimensionalism and the data warehouse," in *The Data Warehousing Conference* (Orlando, FL, February 1995).

[Fin95] Finkelstein, R., "MDD: database reaches the next dimension," in *Database Programming and Design*, pp. 27-28, April 1995.

[GG95] Grädel, E., and Gurevich, Y., "Metafinite model theory," in Proceedings *Logic and Computational Complexity* (Indianapolis, 1994), in *Lecture Notes in Computer Science*, vol. 960, 1995.

[GBLP95] Gray, J., Bosworth, A., Layman, A., and Pirahesh, H. "Data cube: a relational aggregation operator generalizing group-by, crosstabs, and subtotals," *Proceedings of ICDE '96*, New Orleans, LA, Feb. 1996.

[GLS96] Gyssens, M., Lakshmanan, L.V.S., and Subramanian, I.N., "Tables as a paradigm for querying and restructuring," *Technical Report*, Concordia University, Montreal, Canada, 1996, submitted for publication. (Preliminary extended abstract appears in *Proceedings 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Montreal, June 1996), pp. 93–103.)

[GL97] Gyssens, M. and Lakshmanan, L.V.S. "A foundation for multi-dimensional databases," Technical Report, Concordia University and University of Limburg, February 1997.

[HRU96] Harinarayanan, V., Rajaraman, A., and Ullman, J.D., "Implementing data cubes efficiently," *SIGMOD Record*, 25:2, 1996, pp. 205–227.

[Klu82] Klug, A., "Equivalence of relational algebra and relational calculus query languages having aggregate functions," *Journal of the ACM* 29:3, July 1982, pp. 699–717.

[LW96] Li, C. and Wang, X.S., "A data model for supporting on-line analytical processing," in *Proceedings Conference on Information and Knowledge Management* (Baltimore, MD, November 1996), pp. 81–88.

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

[Star95] Red Brick Systems White Paper. *Star Schemes and Star Join Technology*, Red Brick Systems, Los Gatos, CA, September 1995.

[Sho82] Shoshani, A., "Statistical databases: characteristics, problems, and some solutions", in *Proceedings 8th International Conference on Very Large Databases* (Mexico City, September 1982), pp. 208–213.

[Sho96] Shoshani, A., "Statistical databases and OLAP: similarities and differences," invited talk, *International Conference on Information and Knowledge Management* (Baltimore, MD, November 1996).