A Functional Model for Dimensional Data Analysis

Nicolas Spyratos *

Laboratoire de Recherche en Informatique,
Université de Paris-Sud,
91405 Orsay Cedex, France
spyratos@lri.fr

Abstract. In decision-support applications one often needs to analyse transactional data accumulated over time and usually stored in a data warehouse. The data is analysed along various dimensions, and at various levels in each dimension. Although several SQL extensions are available today for the analysis of dimensional data, there seems to be no agreement as to a conceptual model able to guide such analysis. The objective of this paper is, precisely, to propose such a model. In our model, a dimensional schema is a labelled, directed, acyclic graph with a single root, and a dimensional database is an assignment of finite functions, one to each arrow of the dimensional schema. Data analysis is performed based on a path expression language and its associated language for Online Analytic Processing (OLAP). The main contribution of the paper is the proposal of a formal model for dimensional data analysis, offering a clear separation between schema and data, as well as a simple yet powerful functional algebra for data manipulation. The expressive power of the model is demonstrated by showing how it can serve as a formal basis for multi-dimensional OLAP (MOLAP) and for relational OLAP (ROLAP).

1 Introduction

Motivation
In decision support applications one often needs to analyse large volumes of transactional data accumulated over time, typically over a period of several months. The data is usually stored in a so-called "data warehouse", and it is analysed along various dimensions and at various levels in each dimension [5, 10, 12].

A data warehousing system consists of three main levels: the source level, the data warehouse level and the data mart level (see Figure 1). At the source level we find the various sources from which data is extracted and fed into an integration module before loading at the warehouse; these sources can be operational databases, collections of files, collections of Web pages, and so on. At the data warehouse level, the integrated data is stored and maintained, usually in

*Work conducted in part while the author was a visitor at the Meme Media Laboratory, University of Hokkaido, Sapporo, Japan.

the form of a relational database. At the data mart level we find smaller, subject oriented data warehouses, called "data marts", which are actually views of the data warehouse (either virtual or materialized). End users interact with the warehouse either directly or through a data mart. Actually, a data warehouse functions just like a usual database, with some important differences, that we briefly discuss below.
Access Mode: In a data warehouse, access to data by the users is almost exclusively for reading and not for writing, i.e., data warehouses can be seen as read-only databases. Changes of data happen only at the sources, and such changes are propagated to the warehouse.
Nature of the Data: The data stored in a data warehouse differs from that stored in a traditional (transactional) database, in that (a) it is historic data, i.e. data accumulated over time, and (b) it is not production data but the result of integration of production data coming from various sources.
User Needs: The end users of a data warehouse are mainly analysts and decision makers, who almost invariably ask for data aggregations along various dimensions (e.g. "total sales by store", or "average sales by city and product category", and so on). Such aggregations require not only efficient processing of very complex queries but most importantly the use of special kinds of schemas, called "dimensional schemas" that facilitate the formulation of such queries.

This paper is focused on dimensional schemas and their query languages, as opposed to normalized relational schemas and their transaction processing languages. Schema normalization was introduced in relational databases with the goal of increasing transaction throughput. Normalized schemas, however, rarely reflect the "business model" of the enterprise, that is, the way the enterprise actually functions; their main concern is to make database updating as efficient as possible, usually at the cost of rendering the schema virtually incomprehensible by the non specialist. Therefore normalized schemas are not suitable for data warehouses, as the analysts and decision makers of the enterprise are unable to "read" the schema and to formulate the queries necessary for their data analyses. Then the question is what kind of data model is most appropriate for easy formulation and efficient evaluation of such queries. Unfortunately, no generally accepted model has emerged so far.

The products offered by data warehouse vendors today are not satisfactory because (a) none offers a clear separation between the physical and the conceptual level, and (b) schema design is based either on methods deriving from relational schema normalization or on *ad hoc* methods intended to capture the concept of dimension in data. Consequently, several proposals have been made recently to remedy these deficiencies. The model proposed in this paper is a contribution in that direction.

Related Work

On-Line Analytic Processing, or OLAP for short, is the main activity carried out by analysts and decision makers. The term OLAP appeared first in a white paper written for Arbor Software Corporation in 1993 [3, 4], though the concept

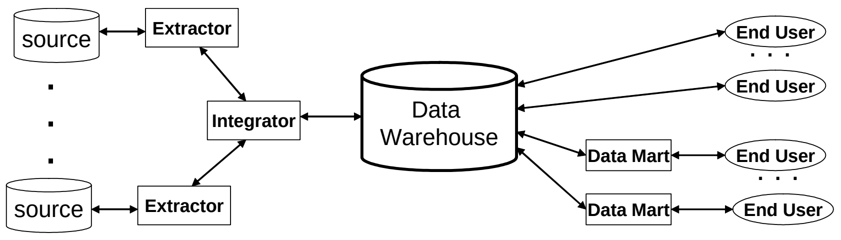


Fig. 1. Data Warehouse Architecture.

seems to be much older. Informally, OLAP can be defined as the interactive process of creating, managing and analyzing data, as well as reporting on the data; the data is usually perceived and manipulated as though it were stored in a multi-dimensional array. Two main approaches have been followed by industrial vendors. The first, known as MOLAP (for Multidimensional OLAP), is based on building separate dedicated engines using multidimensional storage strategies. The second, known as ROLAP (for Relational OLAP), is based on adapting relational database systems.

In terms of research, the proposal of the cube operator [7] is one of the early, significant contributions, followed by much work on finding efficient data cube algorithms [2, 9]. Relatively little work has gone into modeling, with early proposals based on multidimensional tables, called cubes, having parameters and measures [1, 11]. However, these works do not seem to provide a clear separation between schema and data. More recent work (e.g. in [8]) offer a clearer separation between structural aspects and content (see [17] for a survey).

However, a common characteristic of all these models is that they somehow keep with the spirit of the relational model, as to the way they view a tuple in a table. Indeed, in all these models, implicitly or explicitly, a tuple (or a row in a table) is seen as a function associating each attribute with a value from that attribute's domain. In our model, by contrast, it is each attribute that we see as a function; such an "attribute function" associates each object in a set of objects being modeled with a value from that attribute's domain (thus describing a property of the objects, much in the spirit of [14]). We then construct *sets of tuples* by "gluing" together these attribute functions, using function "pairing" (an operation to be introduced shortly). Our approach is similar in spirit to the one proposed in [6] although that work does not address OLAP issues. The main contribution of our paper is the proposal of a formal model for dimensional data analysis, offering a clear separation between schema and data, as well as a simple yet powerful functional algebra for data manipulation.
In the remainder of the paper, in section 2 we present our functional algebra, while in section 3 we define dimensional schemas and dimensional databases. In section 4, we first define a path expression language for dimensional schemas and then use it to define the OLAP language of our model; in doing so, we also explain how our model can serve as a formal basis for multi-dimensional OLAP (MOLAP). In section 5 we discuss in detail how our model can serve as a formal

basis for relational OLAP (ROLAP) as well. Finally, in section 6, we offer some concluding remarks and outline ongoing research and perspectives.

2 The Functional Algebra

In this section we introduce four elementary operations on functions that constitute what we call the functional algebra. We shall use this algebra in the evaluation of path expressions and OLAP queries later on.

Composition
Composition takes as input two functions, f and g, such that range(f) $\subseteq$ def(g), and returns a function g $\circ$ f: def(f) $\rightarrow$ range(g), defined by: (g $\circ$ f)(x)= g(f(x)) for all x in def(f).

Pairing
Pairing takes as input two functions f and g, such that def(f)=def(g), and returns a function f $\wedge$ g: def(f) $\rightarrow$ range(f) $\times$ range(g), defined by: (f $\wedge$ g)(x)= $\langle f(x), g(x) \rangle$, for all x in def(f). The pairing of more than two functions is defined in the obvious way. Intuitively, this is the tuple-forming operation.

Of particular interest are pairings $f_1 \wedge .. \wedge f_n$: X $\rightarrow$ range(f_1) $\times$ .. $\times$ range(f_n) that are one-to-one functions. Such a pairing provides an unambiguous representation of the elements of X, in the following sense: for all x, x' in X we have: if $x \neq x'$ then there is $i \in \{1, 2, .., n\}$ such that $f_i(x) \neq f_i(x')$. In other words, such a pairing sets up an n-dimensional coordinate space with origin X, in which the functions f_1 , .., f_n are the coordinate functions, and in which each point x of X is represented by its coordinates $\langle f_1(x), .., f_n(x) \rangle$.
Note: Throughout this paper, we consider that the product of n sets is always the same (up to isomorphism), no matter how the factors are ordered or parenthesized. For example, the notations $A \times (B \times C)$, $(A \times B) \times C$, $A \times (C \times B)$, and so on, will all stand for $A \times B \times C$.

Projection
This is the usual projection function over a Cartesian product. It is necessary in order to be able to reconstruct the arguments of a pairing, as expressed in the following proposition (whose proof follows immediately from the definitions).

Proposition 1
Let $f : X \rightarrow Y$ and $g : X \rightarrow Z$ be two functions with common domain of definition, and let π_Y and π_Z denote the projection functions over the product $Y \times Z$. Then the following hold:
$f = \pi_Y \circ (f \wedge g)$ and $g = \pi_Z \circ (f \wedge g)$

In other words, the original functions f and g can be reconstructed by composing their pairing with the appropriate projection. This double commutative property is depicted in Figure 2. The extension to pairings with more than two arguments is obvious.

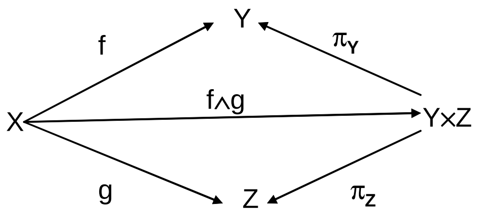

$$\begin{tikzcd} & Y & \\ X \arrow[ur, "f"] \arrow[rr, "f \wedge g"] \arrow[dr, "g"'] & & Y \times Z \arrow[ul, "\pi_Y"'] \arrow[dl, "\pi_Z"] \\ & Z & \end{tikzcd}$$

Fig. 2. Double commutative diagramme.

Restriction
It takes as argument a function $f : X \to Y$ and a set E, such that $E \subseteq X$, and returns a function $f/_E : E \to Y$, defined by: $f/_E(x) = f(x)$, for all x in E.

We note that the set E can be given either extensionally, or intentionally. We also note that one can use domain restriction, as defined above, to define the restriction of f to a desired subset of its range. Indeed, assuming $F \subseteq Y$, this can be done by defining the restriction $f/_E$, where $E = \{x \in X/f(x) \in F\}$.

The four operations just introduced form our functional algebra. Note that this algebra has the closure property, i.e. the result of each operation is a function. Well formed expressions of the functional algebra, their evaluation, and the evaluation of their inverses lie at the heart of the OLAP query language that we shall present in the remainder of this paper. A relevant issue in this respect is how to compute the inverse of a functional expression in terms of the inverses of its component functions. The following proposition gives some elementary properties of inverses, whose proofs follow immediately from the definitions.

Proposition 2 - Properties of Inverses
Composition: Let $f : X \to Y$ and $g : Y \to Z$. Then for all $z \in range(g \circ f)$ we have: $(g \circ f)^{-1}(z) = \cup\{f^{-1}(y)/y \in g^{-1}(z))$ that is, a z-block under $g \circ f$ is the union of all y-blocks under f, where y ranges over the z-block under g
Pairing: Let $f : X \to Y$ and $g : X \to Z$. Then for all $(y, z) \in range(f \wedge g)$ we have: $(f \wedge g)^{-1}((y, z)) = f^{-1}(y) \cap g^{-1}(z)$
Restriction: Let $f : X \to Y$ and $E \subseteq X$. Then for all $y \in range(f/_E)$ we have: $(f/_E)^{-1}(y) = E \cap f^{-1}(y)$

These and other properties of inverses can be used to reduce the computational effort when evaluating inverses of functional expressions. Indeed, by "caching" and re-using previously computed inverses, one can save computational time.

3 Dimensional Schema and Dimensional Database

Following our model, a data warehouse operates from a "dimensional schema" over which one formulates "OLAP queries". In this section, we define the concepts of a dimensional schema and a database and in the following section we use them to define OLAP queries and their answers.

3.1 Dimensional Schema

In our model, a dimensional schema is actually a directed acyclic graph (dag) satisfying certain properties, as stated in the following definition.

Definition 1 -Dimensional Schema
A *dimensional schema* is a connected, labeled dag, whose nodes and arrows satisfy the following properties (see also Figure 3):

Nodes

1. There is only one root; it is labeled O, and called the *origin*
2. There is a distinguished node other than the root, called the *unit node*; it is labeled $\perp$
3. Each node A is associated with a set of values, or *domain*, denoted as $dom(A)$; the domain of $\perp$ is required to be a singleton

Arrows

1. There is no arrow with the unit node $\perp$ as its source
2. All arrow labels are distinct; we use the notation $f : X \to Y$ to denote that f is the label of arrow $X \to Y$
3. The arrows with source O are of two distinguished kinds: dimensional arrows and measure arrows; we use the following notation:
   dimensional arrows $f_1 : O \to D_1$, $f_2 : O \to D_2, .., f_n : O \to D_n$
   measure arrows $m_1 : O \to M_1$, $m_2 : O \to M_2$, .., $m_k : O \to M_k$
4. There is an arrow $\perp$: $O \to \perp$, called the *unit arrow*; it is considered as a dimensional arrow
5. There is no path of length greater than one from the origin O to the target of a dimensional arrow.

Note that, in the above definition, we use the label $\perp$ to denote both the unit node and the unit arrow. Hereafter, in all our discussions, we shall refer to the values in the domain of the origin O as *objects* (and will denote them by integers in our examples). Moreover, we shall refer to all nodes other than the origin and the unit node as *attributes* (the intension being to view attributes as properties of the objects).

Figure 3 shows an example of a dimensional schema that we shall use as our running example throughout the paper. In this schema, we assume the arrows f, g and h to be the dimensional arrows, and the arrow m to be the only measure arrow (in reality, it is the designer who decides which arrows are the dimensional arrows and which are the measure arrows). It should be easy to check that the graph of figure 3 satisfies all the requirements of the above definition. Note that the schema of figure 3 is a tree. This choice was made only to simplify the presentation; a schema need not always be a tree.

Intuitively, in the schema of our running example, each object o represents a sales record containing a date, a store number, a product reference number, and the number of units sold of that product; moreover, each of the attributes Date, Store and Product has "levels" for aggregation purposes. More formally, we call *dimensional path* any path beginning with a dimensional arrow $f_i : O \rightarrow D_i$, and we call each node in the path other than O an *aggregation level*, or simply *level* (D_i being the *base level*). Similarly, we call *measure path* any path beginning with a measure arrow $m_j : O \rightarrow M_j$, and we call each node in the path other than O a *measure level* (M_j being the base level).

Referring to Figure 3, we see that there is one dimensional path beginning with f, and having as levels Date and Month; one beginning with g, and having as levels Store, City and Region; and two beginning with h, one having as levels Product and Category, and the other having as levels Product and Supplier. There is only one measure path beginning with m, having Sales as its only level.

3.2 Dimensional Database

Having defined what a dimensional schema is, we can now define the concept of a database over such a schema.

Definition 2 - Dimensional Database
Let S be a dimensional schema. A *dimensional database* over S is a function δ that associates: each node A of S with a finite subset $\delta(A)$ of its domain; the unit arrow with a constant function; and each other arrow $f : X \rightarrow Y$ of S with a total function $\delta(f) : \delta(X) \rightarrow \delta(Y)$, such that the following constraint is satisfied:
Dimensional constraint: the pairing $\delta(f_1) \wedge ... \wedge \delta(f_n)$ is a one-to-one function.

In Figure 4(a), we see a database, in the form of a set of binary tables giving the finite functions assigned to the arrows by δ. Hereafter, we call *dimensional functions* the functions assigned by δ to the dimensional arrows, and *measure functions* the functions assigned by δ to the measure arrows.

What the above definition says is that an assignment δ of functions is a database only if it sets up an n-dimensional space, with origin O, for which the dimensional functions are its coordinate functions (recall also our remarks following the definition of pairing in the previous section). As a consequence, a dimensional database can be visualised as an n-dimensional cube, with origin O, and each n-tuple of coordinates $\langle \delta(f_1)(o), \delta(f_2)(o), .., \delta(f_n)(o) \rangle$ can be visualised as a "cell" of that cube. Moreover, the k-tuple of measures $\langle \delta(m_1)(o), \delta(m_2)(o), .., \delta(m_k)(o) \rangle$

associated with o can be viewed as the "value" of the cell $\langle \delta(f_1)(o), \delta(f_2)(o), .., \delta(f_n)(o)\rangle$. In data warehouse jargon, this cube is referred to as "the data cube".

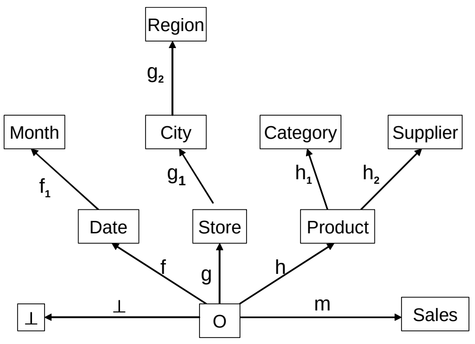
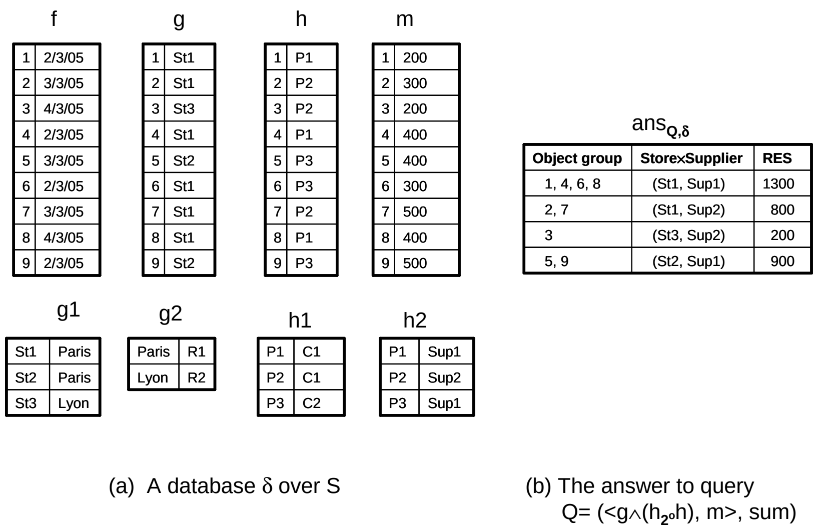


Fig. 3. A Dimensional Schema S.



f

1	2/3/05
2	3/3/05
3	4/3/05
4	2/3/05
5	3/3/05
6	2/3/05
7	3/3/05
8	4/3/05
9	2/3/05

g

1	St1
2	St1
3	St3
4	St1
5	St2
6	St1
7	St1
8	St1
9	St2

h

1	P1
2	P2
3	P2
4	P1
5	P3
6	P3
7	P2
8	P1
9	P3

m

1	200
2	300
3	200
4	400
5	400
6	300
7	500
8	400
9	500

g1

St1	Paris
St2	Paris
St3	Lyon

g2

Paris	R1
Lyon	R2

h1

P1	C1
P2	C1
P3	C2

h2

P1	Sup1
P2	Sup2
P3	Sup1

$\text{ans}_{Q,\delta}$

Object group	Store×Supplier	RES
1, 4, 6, 8	(St1, Sup1)	1300
2, 7	(St1, Sup2)	800
3	(St3, Sup2)	200
5, 9	(St2, Sup1)	900

(a) A database δ over S

(b) The answer to query $Q = (\langle g \wedge (h_2 \circ h), m\rangle, sum)$

Fig. 4.

Several remarks are in order here concerning the above definition of a dimensional database. Our first remark concerns notation. In the remainder of this paper, in order to simplify the presentation, we adopt the following abuse of notation: we use an arrow label such as f to denote both the arrow f and the function $\delta(f)$ assigned to f by δ; similarly, we use an attribute label such as X to denote both the attribute X and the finite set $\delta(X)$ assigned to X by δ.

This should create no confusion, as more often than not the context will resolve ambiguity. For example, when we write $def(f)$ it is clear that f stands for the function $\delta(f)$, as "def" denotes the domain of definition of a function; or when we say "function f", it is clear again that f stands for the function $\delta(f)$ and not for the arrow f. We hope that this slight overloading of the meaning of symbols will facilitate reading.

Our second remark concerns the manner in which functions are assigned to arrows by the database δ. Each function f in a database can be given either extensionally, i.e., as a set of pairs $\langle x, f(x)\rangle$, or intentionally, i.e., by giving a formula or some other means for determining $f(x)$ from x. For example, referring to Figure 3, the function $m : O \to Sales$ can only be given extensionally, as there is no formula for determining the sales transactions that took place on a particular date; whereas the function $f : Date \to Month$ will be given intentionally, as given a date one can compute the month: dd/mm/yy $\mapsto$ mm/yy.

Our third remark concerns the requirement that all functions assigned by the database δ to the arrows of S be total functions. This restriction could be relaxed, by endowing each attribute domain with a bottom element $\perp$ (meaning "undefined") and requiring that for any function $f : X \to Y$ we have (a) $f(\perp) =\perp$, i.e. "bottom can only map to bottom" and (b) if $x \notin def(f)$ then $f(x) =\perp$. Under these assumptions, the functions can again be considered as total functions. However, the resulting theory would be more involved and would certainly obscure some of the important points that we would like to bring forward concerning OLAP queries. Keep in mind, however, that the restriction that all functions assigned by δ be total entails satisfaction of the following constraint:

referential constraint: for every pair of functions of the form $f : X \to Y$ and $g : Y \to Z$ we have $range(f) \subseteq def(g)$.

Our fourth and final remark concerns the intuitive meaning of dimension levels and measure levels, in a database. Indeed, each function $f : X \to Y$ from a level X to a level Y, can be seen as grouping together, or aggregating the elements of X and "naming" the groups using elements of Y. This is expressed by the inverse function f^{-1} which maps each y in the range of f to a nonempty subset of X as follows: $f^{-1}(y) = \{x \in X/f(x) = y\}$, for all $y \in range(f)$. For example, consider the function $g_2 : City \to Region$ of our running example. The inverse g_2^{-1} maps each region r to the set of cities belonging to that region. As we shall see later, inverse functions play a crucial role in the evaluation of OLAP queries, and provide the link between the formal model presented here and the Group-by instruction of SQL.

4 Path Expressions and OLAP

Our definition of OLAP queries relies on the set of path expressions that one can define over the dimensional schema. Consequently, in this section, we first define a language of path expressions and then use it to define the OLAP query language of our model.

4.1 The Language of Path Expressions

Intuitively, a path expression over a dimensional schema S is a well formed expression whose operands are arrows from S and whose operators are those of the functional algebra. More formally, we have the following definition.

Definition 3 -The Path Expression Language
Let S be a dimensional schema. A *path expression* e over S is defined by the following grammar, where "::=" stands for "can be", p and q are path expressions, and $source(e)$, $target(e)$ are self-explanatory:

e::= **f**, where $f : X \rightarrow Y$ is an arrow of S; $source(e) = X$ and $target(e) = Y$
$q \circ p$, where $target(p) = source(q)$; $source(e) = source(p)$ and $target(e) = target(q)$
$p \wedge q$, where $source(p) = source(q)$; $source(e) = source(p)$ and $target(e) = target(p) \times target(q)$
$p/_E$, where $E \subseteq source(p)$; $source(e) = E$ and $target(e) = target(p)$
$\pi_X(e_1 \wedge ... \wedge e_j)$, $X = \{X_1, .., X_r\} \subseteq \{target(e_1), .., target(e_j)\}$;
$source(e) = range(e_1) \times ... \times range(e_j), target(e) = X_1 \times ... \times X_r$

The set of all path expressions is called the *path expression language* of S.

For example, in the dimensional schema of Figure 3, $g \wedge (h_2 \circ h)$ is a path expression. In our discussions, we shall call *dimensional expression* any path expression that involves only dimensional arrows and *measure expression* any path expression that involves only measure arrows.

Now, the functions stored in a dimensional database represent information about some application being modelled. By combining these functions we can derive new information about the application. Specifying what kind of new information we need is done using a path expression; and finding the actual information is done by evaluating the path expression.

Definition 4 -The Evaluation of a Path Expression
Let S be a dimensional schema and e a path expression over S. Given a database δ over S, the *evaluation* of e with respect to δ, denoted $eval(e, \delta)$, is defined as follows :

1. replace each arrow f in e by the function $\delta(f)$;
2. perform the operations of the functional algebra (as indicated in the expression);
3. return the result

Note that the result of the evaluation is obviously a function from the source of e to the target of e.

We note that the path expression language just introduced allows defining views as well, in much the way as this is done in the relational model. Intuitively, a view of a dimensional database is again a dimensional database whose data is derived from those of the original database. More formally, a view of a dimensional schema is defined as follows:

Definition 5 View of a Dimensional Schema
Let S be a dimensional schema. A *view* over S is a dimensional schema S such that every node of S is a node from S and every arrow v of S is defined by a path expression e over S, i.e. an expression of the form v $= e$.

As an example, refer to Figure 3, and suppose that we wish to define a view over S dealing only with monthly sales per City and Product. Such a view will have the following dimensional schema S:
$v_1 : O \to Month$, $v_2 : O \to City$, $v_3 : O \to Product$, $v_4 : O \to Sales$
Here, $v_1 = f1 \circ f$, $v_2 = g_1 \circ g$, $v_3 = h$, $v_4 = m$
In other words, S is a dimensional schema with "custom-made arrows" to fit some particular application. This intuition conforms to a widely used notion in the data warehouse community, that of a data mart. A *data mart* is a subject-oriented data warehouse whose data is derived from those of the "central" data warehouse.

As in the case of relational views, the data of a data mart are derived from those of the data warehouse, by evaluating the path expressions defining the arrows of the data mart schema. Moreover, this derived data may or may not be stored -depending on the application. In other words, as in the case of relational views, we may have virtual or materialized data marts.

We note that, as the cost for setting up a central data warehouse is usually very high, several medium size enterprises start by setting up subject oriented data warehouses, i.e. (materialized) data marts. The advantage of this approach is that one can get acquainted with data warehouse technology, at low cost, before embarking on full scale data warehousing. The disadvantage is having to integrate two or more data marts into a single, "central" warehouse, if the enterprise decides to invest in that direction at some later time.

4.2 The OLAP Language of a Dimensional Schema

The definition of an OLAP query is based on what we call an "OLAP pattern", which specifies what part of the dimensional schema is to be used for carrying out analysis tasks.

Definition 6 - OLAP Pattern
Let S be a dimensional schema. An *OLAP pattern* over S is a pair $\langle u, v \rangle$, where u is a dimensional expression, v a measure expression and $source(u) = source(v) = O$.

In our discussions below, we shall assume that:

– *target*$(u) = DL_1 \times DL_2 \times ... \times DL_{dl}$, for some integer $dl \geq 1$, where $DL_1, DL_2, ..., DL_{dl}$, are dimensional levels; and we shall refer to the product $DL = DL_1 \times DL_2 \times ... \times DL_{dl}$, as the *aggregation level* of the pattern
– *target*$(v) = ML_1 \times ML_2 \times ... \times ML_{ml}$, for some integer $ml \geq 1$, where $ML_1, ML_2, ..., ML_{ml}$, are measure levels; and we shall refer to the product $ML = ML_1 \times ML_2 \times ... \times ML_{ml}$, as the *measure level* of the pattern

For example, in Figure 3, the following is an OLAP pattern: $\langle g \wedge (h_2 \circ h), m\rangle$; here, $u = g \wedge (h_2 \circ h)$ and $v = m$. The aggregation level of this pattern is $DL = Store \times Supplier$, and its measure level is $ML = Sales$. In all our discussions, it is important to remember that both expressions of a pattern, u and v, have O as their common source.

Intuitively, each pattern $\langle u, v\rangle$ provides a setting where data analysis tasks can be performed. To see concretely what kind of analysis tasks one can perform, consider again the pattern $\langle g \wedge (h_2 \circ h), m\rangle$. Here, $u = g \wedge (h_2 \circ h)$ and $v = m$ (refer to figures 3 and 4):

– Compute the inverse $(g \wedge (h_2 \circ h))^{-1}$ to partition O into groups of objects; here, range$(g \wedge (h_2 \circ h)) = \{(St1, Sup1), (St1, Sup2), (St3, Sup2), (St2, Sup1)\}$, so the groups of objects obtained by the inversion are as follows:

  $(g \wedge (h_2 \circ h))^{-1}((St1, Sup1)) = \{1, 4, 6, 8\}$
  $(g \wedge (h_2 \circ h))^{-1}((St1, Sup2)) = \{2, 7\}$
  $(g \wedge (h_2 \circ h))^{-1}((St3, Sup2)) = \{3\}$
  $(g \wedge (h_2 \circ h))^{-1}((St2, Sup1)) = \{5, 9\}$

  Note that all objects of a group map to the same *Store* $\times$ *Supplier*-value.

– Within each group, use m to compute the tuple of images of all objects in the group:

  $\{1, 4, 6, 8\} \rightarrow \langle 200, 400, 300, 400\rangle$
  $\{2, 7\} \rightarrow \langle 300, 500\rangle$
  $\{3\} \rightarrow \langle 200\rangle$
  $\{5, 9\} \rightarrow \langle 400, 500\rangle$

  Note that each tuple of images gives the sales figures for the objects in the corresponding group.

– Assuming that we are interested in "total sales" by store and supplier, we apply the operation "sum" to each tuple of images, to obtain the following results:

  $\langle 200, 400, 300, 400\rangle \rightarrow 1300$
  $\langle 300, 500\rangle \rightarrow 800$
  $\langle 200\rangle \rightarrow 200$
  $\langle 400, 500\rangle \rightarrow 900$

  So now each *Store* $\times$ *Supplier*-value y_i, is associated with a total-sales figure, call it RES_{yi})

– Return each group $(g \wedge (h_2 \circ h))^{-1}(y_i)$ of objects together with its associated pair (y_i, RES_{yi}):

$$\begin{array}{ll} \{1,4,6,8\} & \rightarrow ((St1, Sup1), 1300) \\ \{2,7\} & \rightarrow ((St1, Sup2), 800) \\ \{3\} & \rightarrow ((St3, Sup2), 200) \\ \{5,9\} & \rightarrow ((St2, Sup1), 900) \end{array}$$

So now each group of objects is associated with a pair of items; the first item is the $Store \times Supplier$-value to which all objects in the group map, and the second item is the sum of sales for that $Store \times Supplier$-value. This final outcome of the computations is shown in Figure 4(b).

Note that one could have specified that the total sales are needed only for stores supplied by supplier number 1, in which case only the first and the last pair should be returned; such a specification is possible by restricting the domain of definition of h_2 to its subset $\{Sup1\}$. Also note that *any* operation other than "sum" (but applicable over Sales) could have been applied on each tuple of images. For example, one could have applied the operation "avg" on each tuple of images to obtain the average sales by Store and Supplier.

The above considerations lead naturally to the definition of OLAP query and its answer. In the following definition, we denote by π_u the partition of O induced by the dimensional expression u, i.e. $\pi_u = \{u^{-1}(y)/y \in range(u)\}$.

Definition 7 - OLAP Query and its answer
Query: Let S be a dimensional schema. An OLAP query over S is a pair $Q = (P, op)$, where $P = \langle u, v\rangle$ is a pattern over S and *op* is an operation applicable over the measure level of P.
Answer: Let δ be a database over S. The *answer* to Q with respect to δ is a function $ans_{Q,\delta}$: $\pi_u \rightarrow DL \times ML$ defined as follows:
For each $y \in range(u)$, let $B_y = u^{-1}(y) = \{o_1, o_2, ..., o_r\} \in \pi u$, let $t(B_y) = \langle v(o_1), v(o_2), .., v(o_r)\rangle$ and let $RES_y = op(t(B_y))$; then define $ans_{Q,\delta}(B_y) = (y, RES_y)$

In our previous computations, the OLAP query was $Q = (\langle g \wedge (h_2 \circ h), m\rangle$, sum), the database δ was the one shown in Figure 4(a), and the answer, $ans_{Q,\delta}$, is the one shown in Figure 4(b). Here are two more examples of queries, from our running example:

– $Q = (\langle f \wedge (h_{1o}h), m\rangle$, avg), asking for the average sales by date and category
– $Q = (\langle f \wedge g, m\rangle$, min), asking for the minimal sales by date and store

In several practical applications the following simplifying conditions are present, and lead to a more convenient notation for OLAP queries:

– the dimensional schema is a tree

– there is only one measure arrow in the schema, hence only one measure level
– restrictions are allowed only at the aggregation level

Under these conditions, the dimensional expression u can be specified by simply giving the dimensional levels $DL_1, DL_2, ..., DL_{dl}$; the measure expression v can be specified by simply giving the (unique) measure level, say M; and the restrictions can be specified by giving subsets of $DL_1, DL_2, ..., DL_{dl}$. In this case, an OLAP query can be specified as follows:

Select $DL_1, DL_2, ..., DL_{dl}$ *op*(M) *as* RES
Having $E_1, E_2, ..., E_{dl}$

Here, $E_1, E_2, ..., E_{dl}$ are subsets of $DL_1, DL_2, ..., DL_{dl}$, respectively. This construct is to be interpreted as follows:
Let P_i be the unique path with source O and target DL_i, and u_i be the expression obtained by composition of all arrows along $P_i, i = 1, ..., d_l$; then $u = u_1 \wedge u_2 \wedge ... \wedge u_{dl}$ is the dimensional expression of the query; the unique measure arrow with target M is the measure expression; *op* is the operation; and RES is a (user given) name for the result (technically, RES is the co-domain of op).

As an example, the query $Q = (P, sum)$, where $P = \langle g \wedge (h_2 \circ h, m \rangle$, and where the result is needed only for supplier number 1, can be specified as follows:

Select Store, Supplier *sum*(Sales) *as* RES
Having Supplier= {Sup1}

An interesting class of OLAP queries is obtained when $u = \perp$, that is, when u is the unit arrow (and v any measure expression). Such queries have the form $Q = (\langle \perp, v \rangle, op)$. As the unit arrow is associated with a constant function in any dimensional database, its inversion returns just one aggregate, namely the set O of all objects. Hence the answer associates the whole of O with the pair $(\perp, RES_\perp)$. In our running example, the answer of the query $Q = (\langle \perp, m \rangle, sum)$ will associate O with the pair $(\perp, 3200)$. Here, 3200 represents the total sales (i.e. for all dates, stores and products).

We end this section with a few remarks concerning the definition of an OLAP query and the evaluation of its answer.

1. The answer to an OLAP query requires the following computational steps:
   Step 1 Evaluate the expressions u and v with respect to the database δ
   Step 2 Compute the inverse $u^{-1}(y)$, for each $y \in range(u)$;
   let $B_y = u^{-1}(y) = \{o_1, o_2, .., o_r\}$
   Step 3 Compute the tuple of images $t(B_y) = \langle v(o_1), v(o_2), .., v(o_r) \rangle$
   Step 4 Compute the result $RES_y = op(t(B_y))$
   Step 5 Define $ans_{Q,\delta}(B_y) = (y, RES_y)$

   Of these five steps, the first three depend only on the OLAP pattern (and on the database), while the last two depend on the operation. Therefore, the evaluation of two or more queries with the same pattern requires steps 1, 2 and 3 to be executed only once (for all queries), while steps 4 and 5 will be executed once for each query. In other words, two or more queries over the same pattern can actually share computations. In fact, in this case one can use the notation $Q = (P, op_1, op_2, .., op_n)$, as a more convenient notation for a set of OLAP queries sharing the same pattern P.

2. Although the answer to an OLAP query is a function associating each group $u^{-1}(y)$ of objects with the pair (y, RES_y), only the pair (y, RES_y) is actually of interest in practice. Thus, in our previous example, it is the total sales by Store and Supplier that are of interest to the analyst, and not the groups $u^{-1}(y)$ of objects used in the calculations. Nevertheless, keeping the groups of objects is useful for optimization purposes, in two important cases:
   - When two or more OLAP queries share the same pattern $P = \langle u, v \rangle$, then the groups of objects created by u will be the same for all evaluations, therefore they can be re-used (see also our previous remark).
   - When the dimensional expression u of an OLAP query Q contains sub-expressions of the dimensional expression of another query Q' (that has already been evaluated), then the properties of our functional algebra (as expressed in Proposition 2) can be applied, to generate the groups of objects of Q by re-using the groups of objects of Q'.

   We note that query optimization issues lie outside the scope of the present paper, and are treated in a separate paper [16].
3. As we have seen, the dimensional expression of an OLAP query is actually the result of pairing a set of dimensional paths with targets $DL_1, DL_2, ..., DL_{dl}$. Each of these paths, once evaluated, returns a function, and these functions have $DL_1, DL_2, ..., DL_{dl}$ as targets. Therefore we can view the answer to an OLAP query as a "cube" of dl dimensions, each cell of which holds the value of the calculated measure. This way of viewing the answer to an OLAP query shows clearly that our model can serve as a formal basis for the cube model and multidimensional OLAP (MOLAP). In the following section, we shall see how our model can serve as a formal basis for relational OLAP (ROLAP) as well.
4. In data warehouse applications, when one and the same OLAP query is asked periodically to the data warehouse it is referred to as a "continuous query", or as a "temporal query". For example, it is conceivable that a query asking the total monthly sales per store and product is asked to the data warehouse at the end of each month. Such a query is a continuous or temporal query. In view of our previous remarks, the results of such queries can be visualized on a screen, in the form of a cube or some other visual presentation; they can also be used as input to a so called report generation module, where the data is rearranged into appropriate cross-tabulations called "reports". These are actually tables, indexed by the aggregation levels present in the query, and in which each cell contains the measure(s) corresponding to the coordinates. In our previous example of continuous query, the coordinates would be Month, Store and Product, and the values placed in the cells would be total sales.

5 Relational OLAP (ROLAP)

In this section we discuss how a dimensional database can be represented as a relational database, so that one can take advantage of relational technology for the evaluation of OLAP queries. The main tools that we use for this representation are the definition of dimensional schema and database, on the one hand, and the basic properties of our functional algebra (as expressed in Proposition 1) on the other. The driving idea here is to use our dimensional schema as an interface, through which the analyst can formulate OLAP queries, then to use the relational representation for evaluating such queries.

Throughout this section, in order to simplify the presentation, we assume that the dimensional schema S is a tree (as in our running example). As a consequence, all dimensional and measure paths will be denoted by their targets.

The first thing to note is that the objects of the origin O should be represented as tuple identifiers. However, tuple identifiers are not "visible" in the relational model, i.e. they are not treated as "first class citizens". As a consequence the objects of O are not "visible" in the relational representation either. Nevertheless, the dimensional constraint (see Definition 2) provides a good solution to this problem. Indeed, as $f_1 \wedge ... \wedge f_n$ is a one-to-one function, every measure function $m_j : O \rightarrow M_j, j = 1, 2, .., k$, induces a function m'_j as follows(see Proposition 1):

- m'_j: $range(f_1 \wedge ... \wedge f_n) \rightarrow M_j$, such that $m'_j = m_{jo}(f_1 \wedge ... \wedge f_n)^{-1}, j = 1, 2, .., k$

The pairing of all functions m'_j, in turn, induces the following function φ:

- φ: $range(f_1 \wedge ... \wedge f_n) \rightarrow range(m_1 \wedge ... \wedge m_k)$, such that $\varphi(\langle f_1(o), .., f_n(o)\rangle) = \langle m_1(o), .., m_k(o)\rangle$

Intuitively, it is the graph of φ that constitutes the basis for the representation of a dimensional database as a relational database. The following definition summarizes our discussion so far, by defining the graph of φ as a relational table, called the "fact table".

Definition 8 - Fact Table
Given a dimensional schema S and a dimensional database over S, define the *fact table* of S, denoted $FT(S)$, or FT for short, to be the relational table whose attributes, dependencies and tuples are defined as follows:
Attributes:
For each dimensional arrow $f_i : O \rightarrow D_i$, i=1,.., n, define D_i to be an attribute of FT with the same domain as in S; and for each measure arrow $m_j : O \rightarrow M_i$, j=1,.., k, define M_j to be an attribute of FT with the same domain as in S. As a result, the table FT has the following $n + k$ attributes: $D_1, .., D_n, M_1, .., M_k$.
Keys:
The only functional dependency of FT is $\{D_1, .., D_n\} \rightarrow \{M_1, .., M_k\}$; this dependency is actually the representation of the function φ (therefore the set $\{D_1, .., D_n\}$ is the only key and FT is in Boyce-Codd Normal Form).
Tuples:

For each o in O, the tuple $\langle f_1(o), .., f_n(o), m_1(o), .., m_n(o)\rangle$ is in FT.

For example, referring to Figure 3, the fact table of S will have the following schema: $FT(Date, Store, Product, Sales)$, with the first three attributes making up the key. Each tuple of this table describes the number of products sold in a given store, at a specific day, and this is seen as the basic fact of concern to the enterprise (hence the name "fact table"); dimensional levels and measure levels are seen as auxiliary parameters for analysing the data of the fact table.

The fact table FT actually embeds the image of each object o of O, under the dimensional functions and the measure functions. Once the fact table has been created, the dimensional functions $f_1, .., f_n$, and the measure functions $m_1, .., m_k$, can be recovered from this table using the basic property expressed by Proposition 1: $f_i(o) = \pi_{D_i}(FT(o))$ and $m_j(o) = \pi_{M_j}(FT(o))$, where $FT(o) = \langle f_1(o), .., f_n(o), m_1(o), .., m_n(o)\rangle$. In other words, we have the following:

- The dimensional functions and the measure functions of the dimensional database are represented by the projections π_{D_i} and π_{M_j} of the fact table, in the relational representation, $i = 1, 2, .., n$ and $j = 1, 2, ..k$.

It remains now to see how the non-dimensional and non-measure functions can be represented in the relational representation (i.e. those functions whose domain of definition is not O). In principle, we could define one binary table for each non-dimensional and each non-measure arrow of S. The resulting set of tables together with the fact table would then constitute a relational representation of S. However, as we have seen, the evaluation of OLAP queries requires several function compositions along paths, and several function pairings. These operations would require several joins in the relational representation, each time an OLAP query is submitted to the system.

Therefore it seems natural to represent all paths of a given dimension in a single table. This table will contain all joins concerning that dimension, which in this case can be calculated just once (or so to speak "pre-calculated"), at the time when the relational representation is created; and similarly for all paths of a given measure. These observations lead to the definition of one relational table per dimension, and one relational table per measure.

Definition 9 - Dimension Tables and Measure Tables
Given a dimensional schema S, and a dimensional database over S, we call D-*dimension table* of the relational representation of S, denoted $DT(S)$, or DT for short, the relational table whose attributes, dependencies and tuples are defined as follows:
Attributes: For each dimension D_i of S, define a table DT_i with attributes all levels of D_i (each of these attributes with the same domain as in S), $i = 1, .., n$; and for each measure M_j of S, define a table MT_j with attributes all levels of M_j (each of these attributes with the same domain as in S), $j = 1, .., k$.
Functional dependencies:
For each dimension D_i, each arrow $L \rightarrow L'$ between levels of D_i becomes a functional dependency of the table DT_i in the relational representation, $i = 1, .., n$;

and for each measure M_j, each arrow $L \to L'$ between levels of M_j becomes a functional dependency of the table MT_j in the relational representation, $j = 1,..,k$. As a result, each dimension D_i is the only key of the table DT_i, and each measure M_j is the only key of the table $MT_j, i = 1,..,n, j = 1,..,k$.
Tuples:
For each dimension D_i, perform a relational join between all function graphs assigned to arrows between levels of D_i in the dimensional database over S, and assign the result to the table DT_i of the relational representation, $i = 1,..,n$ (and similarly for each measure $M_j, j = 1,..,k$).

The fact table, together with all the dimensional and all the measure tables thus defined, constitutes a representation of the dimensional database as a relational database. In our running example, this relational representation consists of the following tables:

Fact table: $FT(Date, Store, Product, Sales)$
Table of dimension Date: $DateT(Date, Month)$
Table of dimension Store: $StoreT(Store, City, Region)$
Table of dimension Product: $ProductT(Product, Category, Supplier)$

Note that as there is only one measure, and it has no non-base levels, there is no measure table (Sales is an attribute in the fact table). Also note that the base level of every dimension is an attribute in both the fact table and the corresponding dimension table. In fact, each of the base levels, Date, Store and Product, is a key in its dimension table, and all three of them, collectively, constitute the key of the fact table. Finally, note that the following referential constraint holds, for each dimensional table DT_i: $\pi_{D_i}(FT) \subseteq \pi_{D_i}(DTi)$

The above representation of a dimensional schema as a relational schema consisting of a fact table, a set of dimensional tables and a set of measure tables, is known as the *star-join schema*, or *star schema* for short [13].

Summarizing our discussion so far, we have seen how a dimensional database can be represented as a relational database over a star schema. Let us see now how an OLAP query over a dimensional database can be evaluated in the star schema representation. The basic tool for doing this is an obvious extension of Proposition 1 that we shall explain now.

Consider a set of r functions with common source X, say $w_1 : X \to Y_1, ..., w_r : X \to Y_r$, and a subset of s functions, say $w_{i1}, ..., w_{is}$. Then $w_{i1} \wedge ... \wedge w_{is} = \pi_{Y_{i1},..,Y_{is}} \circ (w_1 \wedge ... \wedge w_r)$, i.e. any sub-pairing $w_{i1} \wedge ... \wedge w_{is}$ of $w_1 \wedge ... \wedge w_r$ can be reconstructed from $w_1 \wedge ... \wedge ...w_r$ by projection of $Y_1 \times ... \times Y_r$ over the set $\{Y_{i1}, ..., Y_{is}\}$.

Now, if we join the fact table and a dimensional table DT_i, then we obtain a table that contains the (representations of) the following functions:

- all functions of the dimension D_i, and all measure functions
- all compositions of functions along a path with origin D_i and all pairings of such compositions

Therefore any path expression whose target $Y_{i1} \times ... \times Y_{is}$ consists only of levels of dimension D_i is represented by the projection $\pi_{Y_{i1},..,Y_{is}}$ of the join between FT and DT_i. Arguing in a similar way, one can extend this fact as follows: any path expression whose target $Y_{i1} \times ... \times Y_{is}$ consists of levels belonging to two dimensions D_{j1} and D_{j2} is represented by the projection $\pi_{Y_{i1},..,Y_{is}}$ of the join between FT, DT_{j1} and DT_{j2} (the extension to more than two dimensions is obvious). In particular, the unit function of the dimensional database is represented by the projection π_ϕ of the fact table FT over the empty set. Note that projection over the empty set always returns the empty tuple, thus π_ϕ is a constant function, as required for the representation of the unit function (see Definition 2).

We are now ready to define the evaluation of an OLAP query over a dimensional schema S, using the star schema representation of S. Let $Q = (\langle u, v\rangle, op)$ be an OLAP query over a dimensional schema S, and suppose that the aggregation level of Q is $Y_1 \times ... \times Y_s$ and the measure level of Q is $Z_1 \times ... \times Z_p$ (i.e. $target(u) = Y_1 \times ... \times Y_s$ and $target(v) = Z_1 \times ... \times Z_p$). Then the evaluation of Q in the star schema representation of S is done as follows:

Step 1 Take the join of the fact table FT with each dimension table containing at least one non-base level among $Y_1, ..., Y_s$, and each measure tables containing at least one non-base level among $Z_1, ..., Z_p$. Call the result of this join J.
Step 2 Compute the projection $\pi_{Y_1..Y_s}(J)$; this projection represents u
Step 3 Compute the inverse projection $\pi^{-1}_{Y_1..Y_s} : Y_1 \times ... \times Y_s \rightarrow J$; the result is a set of pairs $\{\langle y, B_y\rangle / y \in Y_1 \times ... \times Y_s\}$, where $B_y = \pi^{-1}_{Y_1,..,Y_s}(y)$
Step 4 Let $Y = \{Y_1, ..., Y_s\}$ and $Z = \{Z_1, ..., Z_p\}$. For each $y \in Y_1 \times ... \times Y_s$, do the following:
   4.1 Let $B_y = \{t_1, .., t_{ny}\}$. Form the tuple $T_y = \langle \pi_Z(t_1), .., \langle \pi_Z(t_{ny})\rangle$; this tuple contains the projections over Z of all tuples in B_y (and note that each $\pi_Z(t_i)$ is the measure associated with $\pi_Y(t_i)$)
   4.2 Apply the operation op on the tuple T_y, let RES_y be the result, and define $ans_{Q,\delta}(B_y) = (y, RES_y)$.

To illustrate these evaluation steps, consider once more the OLAP query $Q = (\langle g \wedge (h_2 \circ h, m\rangle, sum)$ of our running example, which asks for the total sales by store and supplier. The aggregation level of this query is $Store \times Supplier$ (which is the target of $g \wedge (h_2 \circ h)$), the measure level is Sales (which is the target of m), and the operation is sum. Applying the above steps we proceed as follows:

Step 1 Compute the join $J = FT \bowtie ProductT$
Step 2 Compute the projection $\pi_{Store,Supplier}(J)$
Step 3 Compute the inverse $\pi^{-1}_{Store,Supplier}(y)$, for each $y \in range(\pi_{Store,Supplier})$
Step 4 For each $y \in range(\pi_{Store,Supplier})$, do the following:
   4.1 Let $B_y = \{t_1, .., t_{ny}\}$. Form the tuple $T_y = \langle \pi_{Sales}(t_1), .., \langle \pi_{Sales}(t_{ny})\rangle$; this tuple contains the projections over Sales of all tuples in B_y (and note that each $\pi_{Sales}(t_i)$ is the measure associated with $\pi_{Store,Supplier}(t_i)$)

4.2 Apply the operation sum on the tuple T_y, let RES_y be the result, and define $ans_Q, \delta(B_y) = (y, RES_y)$.

Note that, as Store is a base level (therefore an attribute of the fact table), the store table, StoreT, does not participate in the join of the first step; only the product table participates, as Supplier is a non-base level contained in this table (and not contained in the fact table). Also note that if the aggregation level and the measure level of an OLAP query consist only of base levels, then step 1 will be omitted altogether. Indeed, as all base levels appear in the fact table, no join is needed in this case. For example, an OLAP query asking for the total sales by store and product will not use step 1 in its evaluation.

Now, to carry out the above steps in the star schema representation, we need inversions of projections, and such inversions are not possible in the relational algebra. However, the Group-by instruction of SQL *does* support the inversion of projections, and therefore it can support the evaluation of OLAP queries in our model. To illustrate this, consider once again the OLAP query $Q = (\langle g \wedge (h_2 \circ h), m \rangle, sum)$ of our running example. The SQL instruction that represents this query is the following:

Select Store, Supplier, *sum*(Sales) *as* RES
FromJoin FT, ProductT
Group by (Store, Supplier)

In the Select clause, the attributes Store and Supplier form the aggregation level; *sum*(Sales) means that Sales is the measure level and *sum* is the operation to be applied on Sales-values; and "RES" is a (user given) name for the result (compare also with a similar construct proposed in the previous section). The From clause indicates that the tables FT and ProductT are to be joined. Finally the Group-by clause indicates that this join is to be projected over the attributes Store and Supplier (and the projection is to be inverted to partition the set of tuples in the join). The restriction operation of our functional algebra is also supported by the Having clause of SQL. In fact, SQL offers several other possibilities as a supporting (commercial) language for our model. However, a detailed discussion of this subject lies outside the scope of the present paper.

6 Concluding Remarks

We have presented a formal model for dimensional data analysis, offering a clear separation between schema and data, as well as a simple yet powerful functional algebra for data manipulation. We have also explained how our model can serve as a formal basis for multi-dimensional OLAP (MOLAP) and for relational OLAP (ROLAP).

Two important aspects of the model that are not treated in this paper are its expressive power and the computational complexity of OLAP queries. Regarding expressive power, we believe that one can gain useful insights by studying how the operations of the relational algebra can be embedded in our functional algebra. The case study presented in section 5 should prove useful in this respect. As for computational complexity, the most appropriate context for its study seems to be the lattice of partitions of the set O of all objects. Indeed, as we have seen, during the evaluation of each OLAP query, the inversion of the dimensional expression induces a partition of O, and this partition is the basis for all subsequent calculations. Work on computational complexity and optimization issues is ongoing, based on previous work by the author [15], and will be reported in a forthcoming paper [16].

A possible generalization of the model is based on the following observation. Although the distinction between dimension arrows and measure arrows seems to be necessary for OLAP applications, this distinction is not essential in principle. However, what *is* essential in every case is the presence of a "representation constraint" allowing different objects to be differentiated, or "separated" by their coordinates.

Another generalization of the model concerns the existence of multiple business applications in the same enterprise. In our example we have considered one such application, concerning the sales of products over time in different stores; it was modeled by a dimensional schema with origin O, whose objects represented sales records. A different business application (in the same enterprise) may concern investments; it will be modeled by a different dimensional schema with a different origin O', whose objects represent investment records. Although the two dimensional schemas may share some of their attributes, they will not be the same in general. Therefore the question arises how one does "joint" analysis in order to correlate results from both applications. Note that the need for two different dimensional schemas may arise even within the same business application, when one wants to consider the same data but from different perspectives (each perspective corresponding to a different set of dimensions). In relational terminology, this happens when the set of attributes has two or more different keys.

References

1. R. Agrawal, A. Gupta, and S. Sarawagi, S.: Modelling Multi-dimensional Databases. IBM Research Report, IBM Almaden Research Center (1995)
2. R. Agrawal et al.: On the computation of multidimensional aggregates. In Proceedings 22nd International Conference on Very Large Databases (1996)
3. Arbor Software Corporation, Sunnyvale, CA: Multi-dimensional Analysis: Converting Corporate Data into Strategic Information. White Paper (1993)
4. E.F. Codd: Providing OLAP (On-Line Analytical Processing) to User Analysts: an IT Mandate. Technical Report, E.F. Codd and Associates (1993)
5. C.J. Date: An introduction to database systems (8th edition). Addison-Wesley (2005)
6. R. Fagin et al.: Multi-structural databases PODS June 13-15, 2005, Baltimore, MD (2005)

7. J. Gray, A. Bosworth, A. Layman and H. Pirahesh: Data Cube: a relational aggregation operator generalizing group-by, crosstabs, and subtotals. Proceedings of ICDE'96(1996)
8. M. Gyssens, and L. Lakshmanan, L.: A foundation for Multidimensional databases. In Proceedings 22nd International Conference on Very Large Databases (1996)
9. V. Harinarayanan, A. Rajaraman, and J.D. Ullman: Implementing data cubes efficiently. SIGMOD Record, **25:2** (1996) 205–227
10. R. Kimball:
    The data warehouse toolkit. J. Wiley and Sons, Inc (1996)
11. C. Li and X.S. Wang: A data model for supporting on-line analytical processing. Proceedings Conference on Information and Knowledge Management (1996) 81–88
12. R. Ramakrishnan and J. Gehrke: Database Management Systems (third edition). McGraw-Hill (2002)
13. Red Brick Systems White Paper: Star schemes and star join technology. Red Brick Systems, Los Gatos, CA (1995)
14. N. Spyratos.: The Partition Model: A Functional Approach. INRIA Research Report **430** (1985)
15. N. Spyratos: The partition Model : A deductive database Model. ACM Transactions on Database Systems **12:1** (1987) 1–37
16. N. Spyratos: A Partition Model for Dimensional Data Analysis. LRI Research Report (2005)
17. P. Vassiliadis and T. Sellis: A survey of logical models for OLAP Databases. SIGMOD Record **28(4)** (1999) 64–69,