A Functional Model for Data Analysis

Nicolas Spyratos*

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

Abstract. We present a functional model for the analysis of large volumes of detailed transactional data, accumulated over time. In our model, the data schema is an acyclic graph with a single root, and data analysis queries are formulated using paths starting at the root. The root models the objects of an application and the remaining nodes model attributes of the objects. Our objective is to use this model as a simple interface for the analyst to formulate queries, and then map the queries to a commercially available system for the actual evaluation.

1 Introduction

In decision-support systems, in order to extract useful information from the data of an application, it is necessary to analyse large amounts of detailed 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 warehouse functions just like a usual database, with the following important differences: (a) the data of a data warehouse is *not* production data but the result of integration of production data coming from various sources, (b) the data of a data warehouse is *historic* data, that is data accumulated over time, (c) access to the data warehouse by analysts is almost exclusively for reading and not for writing and (d) changes of data happen only at the sources, and such changes are propagated periodically to the data warehouse.

The end users of a data warehouse are mainly analysts and decision makers, who almost invariably ask for data aggregations such as "total sales by store", or "average sales by city and product category", and so on. In this context, the basic requirements by data analysts are (a) a data schema that is easy to understand and (b) a flexible and powerful query language in which to express complex data analysis tasks. The so called "dimensional schemas" and their associated "OLAP query languages" were introduced precisely to satisfy these requirements.

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

H.L. Larsen et al. (Eds.): FQAS 2006, LNCS 4027, pp. 51–64, 2006.
© Springer-Verlag Berlin Heidelberg 2006

This paper is focused on dimensional schemas and their OLAP 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.

On-Line Analytic Processing, or OLAP for short, is the main activity carried out by analysts and decision makers [3, 4]. However, although several SQL extensions are available today for OLAP, there seems to be no agreement as to a simple conceptual model able to guide data analysis. The objective of this paper is to propose such a model.

The products offered today by data warehouse vendors 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 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 modelling, 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 works (e.g. in [8]) offer a clearer separation between structural aspects and content (see [17] for a survey).

However, a common characteristic of most of 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 table attribute with a value from that attribute's domain; by contrast, in our model, it is each attribute that is seen as a function. Our approach is similar in spirit to the one of [6] although that work does not address OLAP issues.

Roughly speaking, in our model, the data schema is an acyclic graph with a single root, and a database is an assignment of finite functions, one to each arrow of the graph. The root of the graph is meant to model the objects of an application, while the remaining nodes model attributes of the objects. Data analysis queries (that we call OLAP queries) are formulated using paths starting at the root, and each query specifies three tasks to be performed on the objects: a classification of the objects into groups, following some criterion; a measurement of some property of objects in each group; a summarization of the measured properties in each group, with respect to some operation.

2 The Functional Algebra

In this section we introduce four elementary operations on (total) functions that we shall use 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, pairing is the tuple-forming operation.

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 easily 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:

$$f = \pi_Y \circ (f \wedge g) \text{ and } g = \pi_Z \circ (f \wedge g)$$

In other words, the original functions f and g can be reconstructed by composing their pairing with the appropriate projection.

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

The four operations on functions just introduced form our *functional algebra*. It is important to note that this algebra has the closure property, that is the arguments and the result of each operation are functions. 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 later.

3 The Data Schema and the Data Base

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

Definition 1. Data Schema
A *data schema*, or simply *schema*, is a finite, labelled dag, whose nodes and arrows satisfy the following conditions:

- *Condition 1* There is only one root
- *Condition 2* There is at least one path from the root to every other node
- *Condition 3* All arrow labels are distinct
- *Condition 4* Each node A is associated with a nonempty set of values, or *domain*, denoted as $dom(A)$

We recall that a directed acyclic graph always has one or more roots, a root being a node with no entering arrows. Condition 1 above requires that the graph have precisely one root. We shall label this root by O and we shall refer to it as the *origin*; it is meant to model the objects of an application.

Condition 2 requires that there be at least one path from the root to every other node. This condition makes sure that there are no isolated components in the schema (i.e. the graph is connected). We note that trees do satisfy conditions 1 and 2, therefore trees constitute the simplest form of schema in our model.

In a directed acyclic graph, it is possible to have "parallel" arrows (i.e. arrows with the same start node and the same end node). Such arrows can be distinguished only through their labels. This is the reason for having condition 3 above. In this respect, we shall use the notation $f : X \to Y$ to denote that f is the label of an arrow from node X to node Y; moreover, we shall call X the *source* of f and Y the *target* of f, that is $source(f) = X$ and $target(f) = Y$.

As we shall see shortly, each arrow $f : X \to Y$ will be interpreted as a total function from a set of X-values to a set of Y-values. Condition 4 makes sure that such values exist at every node.

Figure 1 shows an example of a schema that we shall use as our running example throughout the paper. This schema describes the data of a company that delivers products of various types to stores across the country. There is at most one delivery per store, per day. The data collected from delivery invoices is stored in a data warehouse and accumulated over long periods of time. Subsequently, they are analysed in order to discover tendencies in the movement of products. The knowledge extracted from the accumulated data is then used to improve the company operations.

The data that appears on an invoice consists of an invoice identifier, a date, the reference number of the store, and a sequence of products delivered during one visit; each product appearing on the invoice is characterized by a number (local to the voucher), followed by the product reference, and the number of units delivered from that product (the number of units is what we call Quantity in the schema). A pair composed of an invoice identifier and a product number on that invoice constitutes *one* object; and the origin of the schema shown in Figure 1 models the set of all such objects.

Each object is characterized by a Date, a Store, a Product, and a Quantity. These are the "primary" characteristics of the object. However, each of these characteristics determines one or more "secondary" characteristics of the

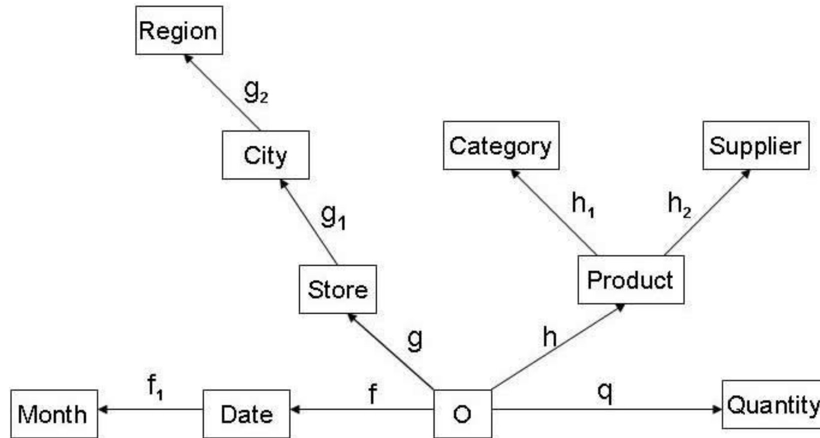

Fig. 1. Example of a data schema S

object. For example, Date determines Month; Store determines City, and City determines Region; finally, Product determines both, Category and Supplier. Although these secondary characteristics might not appear on the invoice, they can usually be inferred from the primary characteristics, and are useful for data analysis purposes (e.g. for aggregating the objects by region, by month and product, and so on). We shall refer to all the characteristics of the object (primary and secondary) as the *attributes* of the object.

Note that the schema of our running example is a tree, a choice made in order to simplify the presentation. However, it should be clear that what we will say in the remaining of this paper is valid for *all* forms of a schema, not just for tree schemas. In fact, non-tree schemas are important as they allow expressing multiple hierarchies among the attributes.

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

Definition 2. Database
Let S be a schema. A *database* over S is a function δ that associates:

- each node A of S with a finite nonempty subset $\delta(A)$ of its domain
- each arrow $f : X \rightarrow Y$ of S with a total function $\delta(f) : \delta(X) \rightarrow \delta(Y)$.

Figure 2(a), shows a database δ over the schema S of our running example. In this figure, each arrow is associated with a binary table containing the function assigned to it by δ; for example, the arrow $f : Store \rightarrow City$ is associated with the binary table whose headings are Store and City.

Several remarks are in order here concerning the above definition of a 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

f

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

g

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

h

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

m

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

g_1

Store	City
St1	Paris
St2	Paris
St3	Lyon

g_2

City	Region
Paris	R1
Lyon	R2

h_1

Prod	Cat
P1	C1
P2	C1
P3	C2

h_2

Prod	Suppl
P1	Sup1
P2	Sup2
P3	Sup1

(a) A database δ over S

$ans_{Q,\delta}$

Store×Supplier	TotQty
(St1, Sup1)	1300
(St1, Sup2)	800
(St3, Sup2)	200
(St2, Sup1)	900

(b) Answer to the query $Q = (\langle g \wedge (h_2 \circ h), q\rangle, sum)$

Fig. 2. Example of database and OLAP query over S

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; similarly, 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, that is as a set of pairs $\langle x, f(x)\rangle$, or intentionally, that is by giving a formula or some other means for determining $f(x)$ from x. For example, the function $q : O \rightarrow Quantity$ can only be given extensionally, as there is no formula for determining the quantity of products to be delivered to a store; whereas the function $f_1 : Date \rightarrow Month$ will be given intentionally, as given a date one can compute the month: dd/mm/yy $\mapsto$ mm/yy. In fact, this is why $\delta(f_1)$ is not given in Figure 2(a).

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 $f : X \rightarrow Y$ we have (a) $f(\perp) = \perp$, that is "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 functions entails the following property: 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 a particular way of looking at the functions of a database, namely as means for grouping objects together. Indeed, each function $f : X \to Y$ can be seen as grouping together the elements of X and "naming" each group using an element 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 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 shortly, inverse functions play a crucial role in the evaluation of OLAP queries.

4 Path Expressions and OLAP Queries

Roughly speaking, a path expression over a schema S is a well formed expression whose operands are arrows from S and whose operators are those of the functional algebra. A path expression represents a generalized notion of arrow, and therefore a path expression has a source and a target. For example, referring to Figure 1, the expression $g \wedge (h_2 \circ h)$ is a path expression, whose source is O and whose target is $Store \times Supplier$. Below, we give a more formal definition of path expression, in which we use the following simplifying notation:

- for attributes A and B we write $A \subseteq B$ to denote that $dom(A) \subseteq dom(B)$
- for attributes $A_1, .., A_r$ we write $A_1 \times ... \times A_r$ to denote an attribute such that $dom(A_1 \times ... \times A_r) = dom(A_1) \times ... \times dom(A_r)$

Definition 3. Path Expression
Let S be a schema. A *path expression* e over S is defined by the following grammar, where "::=" stands for "can be", and p and q are path expressions:

e::= **f**, where f is an arrow of S; $source(e) = source(f)$ and $target(e) = target(f)$
$q \circ p$, 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 $target(e) = target(p)$
$\pi_X(A_1 \times ... \times A_j)$, where $X = \{A_1, .., A_r\} \subseteq \{A_1), .., A_j)\}$; $source(e) = A_1 \times ... \times A_j$, $target(e) = A_1 \times ... \times A_r$

Here are some examples of path expressions over the schema S of Figure 1:

- $e_1 = f_1 \circ f$, with $source(e_1) = O$ and $target(e_1) = Month$
- $e_2 = f \wedge g$, with $source(e_2) = O$ and $target(e_2) = Date \times Store$
- $e_3 = ((g_2 \circ g_1 \circ g) \wedge (h_1 \circ h))$, with $source(e_3) = O$ and $target(e_3) = Region \times Category$

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

Intuitively, given a path expression e over schema S, and a database δ over S, the evaluation of e proceeds as follows:

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

It is important to note that the evaluation of a path expression e always returns a function from the source of e to the target of e. More formally, we have the following definition.

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 the function defined below, where p and q denote path expressions over S:

– if $e = f$, where f is an arrow of S, then $eval(e, \delta) = \delta(f)$;
– if $e = q \circ p$ then $eval(e, \delta) = eval(q, \delta) \circ eval(p, \delta)$;
– if $e = p \wedge q$ then $eval(e, \delta) = eval(p, \delta) \wedge eval(q, \delta)$;
– if $e = p/_E$ then $eval(e, \delta) = (eval(p, \delta))/dom(E)$
– if $e = \pi_X(A_1 \times ... \times A_j)$ then $eval(e, \delta) = \pi_X(\delta(A_1) \times ... \times \delta(A_j))$

A path expression of particular interest is obtained when we compose a path expression with a projection over the empty set. Indeed, if we apply the projection function $\pi_\emptyset$ on any nonempty Cartesian product $A_1 \times ... \times A_j$ the result is always the same, namely the *empty tuple*, denoted by λ. In other words, $\pi_\emptyset(A_1 \times ... \times A_j) = \{\lambda\}$, for any $A_1 \times ... \times A_j \neq \emptyset$. This particular path expression, is called the *constant path expression*, denoted by $\perp$. Clearly, the constant path expression evaluates to a constant function over any database and, as we shall see, it is useful in expressing OLAP queries of a special kind.

Path expressions are the basis for defining OLAP queries in our model. Roughly speaking, the purpose of an OLAP query is to perform a sequence of three tasks:

– *Grouping* (or *Classification*): group together the objects into mutually disjoint sets
– *Measuring*: in each group, for each object, measure some specified property of the object
– *Summarizing*: in each group, summarize the measured properties of the objects

Before giving formal definitions, let us illustrate these three tasks intuitively, using our running example. Suppose we want to evaluate the following query:

for each store-supplier pair, find the total quantity of products delivered

To do this, let us perform the three tasks described above, that is grouping, measuring, and summarizing.

Grouping. Two objects are put in the same group if they correspond to the same store-supplier pair. To check this condition, we need a function that takes as input an object and returns a store-supplier pair. Such a function can be obtained by evaluating a path expression with source O and with target $Store \times Supplier$. Referring to Figure 1, we can easily check that the only path expression having this property is the expression $u = (g \wedge (h_2 \circ h))$. Clearly, the inverse function u^{-1} associates each store-supplier pair to the set of all objects having that pair as image, and thus it groups the objects into the desired groups. (Note that, in presence of more than one such expression, a choice will have to be made by the user.) Concerning the actual calculations, we note that only the store-supplier pairs that belong to the range of u have nonempty inverses. Referring to Figure 1, we can easily check that the range of u contains four pairs: $\{(St1, Sup1), (St1, Sup2), (St3, Sup2) and (St2, Sup1)\}$; all other pairs of Store $\times$ Supplier have empty inverse images under u. The nonempty groups of objects obtained as inverse images of the pairs in the range of u are as follows:

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

These four inverse images form a partition of O, and this partition is the result of the grouping.

Measurement. Within each group of objects, as computed in the previous step, and for each object in the group, we apply the function q in order to find the quantity of delivered products for that object:

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

Summarizing. For each group in the previous step, we sum up the quantities found, in order to obtain the total quantity of the group:

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

As we can see through the above three steps, each store-supplier pair (St, Sup) is associated to a group of objects $u^{-1}((St, Sup))$; and this group of objects, in turn, is associated to a total quantity of products delivered. This process is depicted below, where we summarize the results of the computations that took place:

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

The important thing to retain is that the above process defines a function from *Store* $\times$ *Supplier* to *Sales*. It is precisely this function that answers our original question, that is "for each store-supplier pair, find the total quantity of products delivered". This query and its answer are shown in Figure 2(b).

The above considerations lead to the following definition of OLAP query and its answer

Definition 7. OLAP query and its answer

- *OLAP Query*
  Let S be a schema. An OLAP Query over S is a (ordered) triple $Q = (u, v, op)$, satisfying the following conditions:
  - u and v are path expressions such that $source(u) = source(v) = O$
  - op is an operation over the target of v

  The expression u will be referred to as the *classifier* of Q and the expression v as the *measure* of Q.
- *Answer*
  Let δ be a database over S. The answer to Q with respect to δ is a function $ans_{Q,\delta}$: $target(u) \rightarrow target(v)$ defined by $ans_{Q,\delta}(y) = op(v(u^{-1}(y)))$, for all $y \in range(u)$.

Here are two more examples of queries, over the schema of our running example:

- $Q_1 = (f \wedge (h_1 \circ h), q, avg)$, asking for the average quantity by date and category
- $Q_2 = (f \wedge g, q, min)$, asking for the minimal quantity by date and store

It is important to note that the notions of "classifier" and "measure" in the above definition are *local* to a query. That is, the same path expression can be classifier in one query and measure in another. As an extreme example, consider the following two queries over the schema of our running example:

- $Q = (g, h, count)$, asking for the number of product references by store
- $Q' = (h, g, count)$, asking for the number of stores by product reference

An interesting class of OLAP queries is obtained when the classifier u is the constant expression (i.e. $u = \perp$) and v is any measure. Such queries have the form $Q = (\perp, v, op)$. As $\perp$ evaluates to a constant function over any database with

nonempty set of objects, its inverse returns just one group, namely the set O of all objects. Hence the answer of Q associates the unique value λ in the range of u with $op(v(O))$. In our running example, the answer of the query $Q = (\perp, q, sum)$ will associate λ with 3200. Here, 3200 represents the total quantity delivered (i.e. for all dates, stores and products).

5 Optimization Issues

As we have seen in the previous section, the partition of O resulting from the grouping step, plays a crucial role in determining the answer. Given a query $Q = (u, v, op)$, the partition induced by the function u on the set of objects O is called the *support* of Q and it is denoted as s_Q. Query optimization consists in using the answer of an already evaluated query in order to evaluate the answer of a new query *without* passing over the data again; and the lattice of partitions of the set O is the formal tool to achieve such optimization.

Definition 8. The Lattice of Partitions
Let p, p' be two partitions of O. We say that p is *finer* than p', denoted $p \leq p'$, if for each group G in p there is a group G' in p' such that $G \subseteq G'$.

One can show that $\leq$ is a partial order over the set of all partitions of O (i.e. a reflexive, transitive and anti-symmetric binary relation over partitions). Under this ordering, the set of all partitions of O becomes a lattice in which the partition $\{O\}$ is the *top* (the coarsest partition) and the partition $\{\{o\}/o \in O\}$ is the *bottom* (the finest partition).

To see how this lattice can be used to achieve optimization, consider a query $Q = (u, v, op)$ which has already been evaluated. As we have explained earlier, if $y_1, .., y_k$ are the values in the range of u, then the support of Q is the following partition of O: $s_Q = \{u^{-1}(y_i)/i = 1, .., k\}$.

Based on the support, the answer to Q is expressed as follows: $ans_Q(y_i) = op(v(u^{-1}(y_i))), i = 1, .., k$.

Now, suppose that a new query $Q' = (u', v', op')$ comes in and we want to evaluate its answer. We claim that if $s_Q \leq s_{Q'}$ then the answer to Q' can be expressed in terms of the support of Q. This is based on a simple fact, which follows immediately from the definition of the partition ordering:

Fact : if $s_Q \leq s_{Q'}$ then each group G' in $s_{Q'}$ is the union of groups from s_Q.

As a result, if $G' = G_1 \cup \ldots \cup G_j$ then $op'(v'(G')) = op'(v'(G_1 \cup \ldots \cup G_j)) = op'$ $(v'\ (G_1)\ ,\ \ldots,\ (v'\ (G_j))$.

As the support of Q has already been computed (and is available), we can apply v' and then op' "off-line" (i.e. *without* passing over the data again). Moreover, if $v = v'$ then we can reuse the measurements of Q as well. That is, if $v = v'$ then we have:

$$op'(v'(G_1), \ldots, (v'(G_j)) = op'(v(G_1), \ldots, (v(G_j))$$

Finally, if in addition *op*= *op*$'$ then we can reuse even the summarizations of Q, provided that the following property holds:

$$op(v(G_1), \ldots, v(G_j)) = op(op(v(G_1)), \ldots, op(v(G_j))$$

One can show that this property holds for most of the usual operations, namely "sum", "count", "max", and "min", but not for "avg". For example, $sum(2,4,6,8) = sum(sum(2,4),(sum(6,8))$, while $avg(2,4,6,8) \neq avg(avg(2,4), avg(6,8))$.

However, all the above results hold under the condition that $s_Q \leq s'_Q$ (see Fact above), so the question is: given two queries, Q and Q', can we decide whether $s_Q \leq s'_Q$?

To answer this question, we observe first that the classifier u of an OLAP query is essentially the pairing of a number of compositions. Therefore it is sufficient to answer the above question for two separate cases: when the classifier is a composition and when the classifer is a pairing. The following proposition provides the answers.

Proposition 2. Comparing Classifiers

- Ĝrouping by Composition
  Let $Q = (u, v, op)$ and $Q' = (u', v', op')$ be two OLAP queries such $u = p$ and $u' = q' \circ p$, where p and q' are path expressions. Then $s_Q \leq s_{Q'}$.
- Ĝrouping by Pairing
  Let $Q = (u, v, op)$ and $Q' = (u', v', op')$ be two OLAP queries such $u = p \wedge q$ and $u' = p$, where p and q are path expressions. Then $s_Q \leq s_{Q'}$.

In our running example, if $Q = (g, q, sum)$ and $Q'= (g_1 \circ g, q, sum)$, then $s_Q \leq s_{Q'}$, therefore the answer of Q' can be computed from that of Q. Similarly, if $Q = (g \wedge h, q, sum)$ and $Q'= (g, q, sum)$, then again $s_Q \leq s_{Q'}$, and the answer of Q' can be computed from that of Q.

The proof of the above proposition follows from properties of function inverses, as stated in the following proposition.

Proposition 3. Properties of Inverses

- *Composition*
  Let $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ be two functions. 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-group under $g \circ f$ is the union of all y-groups under f, where y ranges over the z-group under g
- *Pairing*
  Let $f : X \rightarrow Y$ and $g : X \rightarrow Z$ be two functions. Then for all $(y, z) \in range(f \wedge g)$ we have: $(f \wedge g)^{-1}((y, z)) = f^{-1}(y) \cap g^{-1}(z)$

Lack of space does not allow further details on optimization. The interested reader is referred to the full paper.

6 Concluding Remarks

We have presented a functional model for data analysis, offering a clear separation between schema and data, as well as a functional algebra for data manipulation. We have also discussed some optimization issues, concerning the evaluation of OLAP queries.

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 first how the operations of the relational algebra can be embedded in our functional algebra. As for computational complexity, the most appropriate context for its study seems to be the lattice of partitions of the set O. 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].

Another generalization of the model concerns the existence of multiple business applications in the same enterprise. In our running example we have considered one such application, concerning delivery of products. A different business application (in the same enterprise) may concern investments; it will be modelled by a different schema with a different origin O', whose objects represent investment records. Although the two 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 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 in the fact table has two or more different keys.

Finally, one practical aspect concerning our model is its embedding into commercially available systems, and ongoing work considers its embedding into a relational system. In fact, a prototype is under development that uses our model as an interface for the definition of OLAP queries which are then passed on to a relational engine for the actual evaluation.

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 (2006)
17. P. Vassiliadis and T. Sellis: A survey of logical models for OLAP Databases. SIGMOD Record **28(4)** (1999) 64–69.