A Systematic Approach to Multidimensional Databases *

Luca Cabibbo and Riccardo Torlone

Dipartimento di Informatica e Automazione
Università degli Studi di Roma Tre
Via della Vasca Navale 84 — 00146 Roma, Italy
cabibbo@inf.uniroma3.it, torlone@iasi.rm.cnr.it

Abstract. Multidimensional databases are large collections of data, often historical, used for sophisticated analysis oriented to decision making. This activity is supported by an emerging category of software technology, called On-Line Analytical Processing (OLAP). In spite of a lot of commercial tools already available, a fundamental study for OLAP systems is still lacking. In this paper we introduce a model and a query language to establish a theoretical basis for multi-dimensional data. The model is based on the notions of dimension and f-table. Dimensions are linguistic categories corresponding to different ways of looking at the information. F-tables are the constructs used to represent factual data, and are the logical counterpart of multi-dimensional arrays, the way in which current analytical tools store data. The query language is a calculus for f-tables, and as such it offers a high-level support to multi-dimensional data analysis. Scalar and aggregate functions can be embedded in calculus expressions in a natural way. We compare our model and language with other approaches, and discuss on several issues related to multidimensional databases. Finally, we identify important research topics that need to be investigated in this context.

1 Introduction

1.1 Data warehouses and multidimensional databases

Database technology provides a solid basis for the management of data and, in the last two decades, several enterprises have migrated their corporate data to relational database systems. These systems are especially effective in managing large and shared collections of operational data. In more recent years, decision makers have realized that an enterprise can obtain a great competitive advantage from the analysis of historical data. For instance, the identification of unusual trends in data can suggest opportunities for new business, whereas the analysis of past consumer demand can be useful for forecasting production needs. To support this activity, there is a need for tools that allow to build and manage "data

* This work was partially supported by *Consiglio Nazionale delle Ricerche* and by *MURST*.

warehouses," and to perform complex processing on their content. A *data warehouse* is an integrated collection of enterprise-wide data, which is maintained primarily to support data analysis and decision making [14]. The process of extracting data from the various departmental sources of information (databases, raw data, legacy systems) and integrating them into a single repository is called *data warehousing*. Even if all the operational data is stored in departmental databases, the data warehouse is still separated from them, for several reasons. First, very often operational systems do not need to maintain historical data. Second, the various departmental databases may be heterogeneous, while the warehouse offers an integrated view of the whole enterprise data. Finally, traditional database management systems are optimized for on-line transaction processing (OLTP), corresponding to large numbers of concurrent transactions, often involving very few records, while the data warehouse should be designed for the so-called *on-line analytical processing* [8] (OLAP), that is, few queries over very large numbers of records.

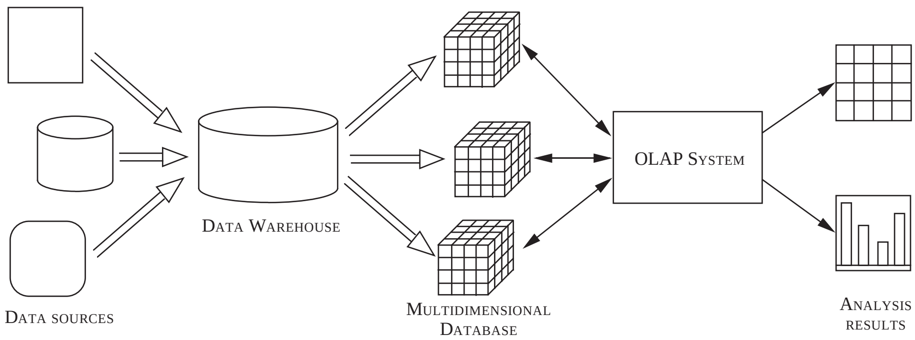


Fig. 1. Data Warehousing architecture

Actually, as shown in Figure 1, data analysis is not always performed directly on the data warehouse, but rather on special data stores, often called *hypercubes* or *multi-dimensional* "fact" tables. These terms originate from the fact that the effectiveness of the analysis is related to the ability of describing and manipulating factual data (or measures) according to different and often independent perspectives or "dimensions", and that this picture can be naturally represented by means of n-dimensional arrays (or cubes). As an example, in a commercial enterprise, single sales of items (the factual data) provide much more information to business analysis when organized according to dimensions like category of product, geographical location, and time. Figure 2 reports the corresponding hypercube for a chain of toy stores. The collection of fact tables of interest for an enterprise forms the *multidimensional database*.

The category of software technology that provides a multi-dimensional view of factual data and suitable tools for its analysis is called OLAP systems. Specif-

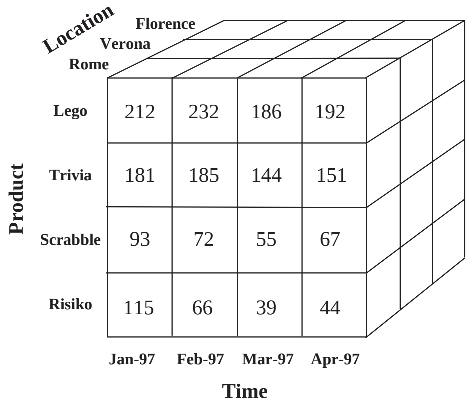


Fig. 2. A multi-dimensional fact table

ically, these systems enable the users to: (i) define analytical equations across multiple data dimensions, possibly involving complex calculations, to represent numerous, speculative enterprise model scenarios; (ii) summarize data sets, aggregating and disaggregating over the various dimensions; and (iii) evaluate and view the outcomes of the analysis. To understand the effect of changes in environmental factors, this process is often iterated by changing equations and parameters. An example of the result of an analysis done by an OLAP system is reported in Figure 3.

Current technology provides both OLAP data servers and client analysis tools. OLAP data servers can be either relational systems (ROLAP) or proprietary multi-dimensional database systems (MOLAP) [9]. A ROLAP is an extended relational system that maps operations on multi-dimensional data to standard relational operations (SQL). On the other hand, a MOLAP is a special server that directly implements multi-dimensional data and operations. The clients offer querying and reporting tools, usually based on interactive graphical user interfaces, similar to spreadsheet ones. While this graphical approach allows the user to easily summarize and view data, spreadsheet-like environments suffer from several limitations in constructing and maintaining analytical models over the enterprise data. The main point is that these models rely an a logic that is often left implicit, leading to several problems, including redundancy and inconsistency [15]. Moreover, the integration with database technology and the optimization mechanisms are based on ad-hoc techniques, rather than any systematic approach. As others [12], we believe that the problem is the lack of a formal theoretical foundation.

1.2 Contribution of the paper

In this paper, we propose the MultiDimensional data model and query language, which provide a first step towards a logical foundation of OLAP systems.

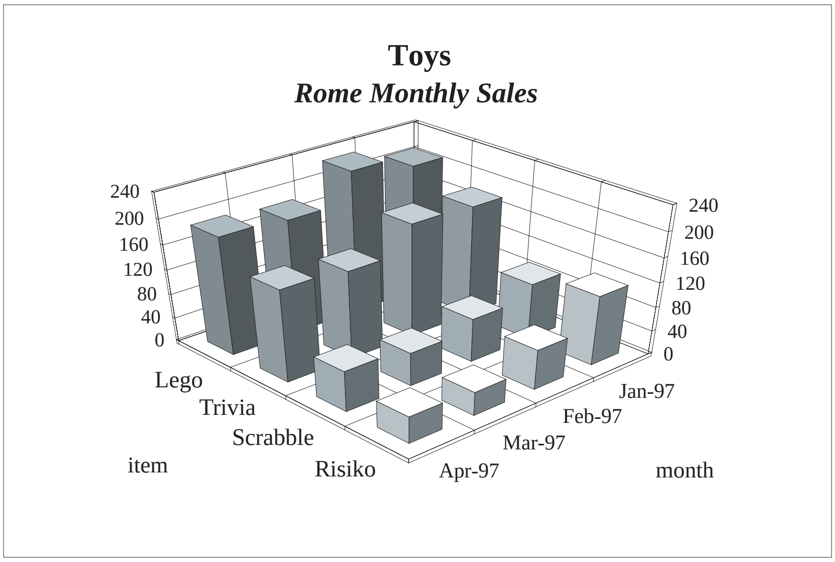


Fig. 3. A graphical outcome of a data analysis

The model allows to describe the logical structure of the enterprise data according to multiple perspectives, by providing an explicit notion of *dimension*. Dimensions are the linguistic categories used to characterize the structure of data, according to a conceptual business perspective. They are organized into hierarchies of levels, corresponding to possible granularities of data. Factual data are then represented by *f-tables*, the logical counterpart of "multi-dimensional arrays" (the way in which OLAP systems store values). Values in f-tables are accessed through symbolic coordinates. The query language enables the user to express cross-dimensional analytical equations, based on logical expressions over f-tables, in a simple and declarative way. Queries make use of interpreted functions, but the language is parametric with respect to the ones chosen.

We also compare our model and language with other approaches, and identify the main research topics that need to be investigated in the context of multi-dimensional databases and OLAP systems.

1.3 Organization of the paper

The paper is organized as follows. The MultiDimensional data model is presented in Section 2. The associated query language is introduced informally, by means of examples, in Section 3, and described formally in Section 4. A comparison with related work is given in Section 5. Finally, Section 6 discusses further research topics.

2 The MultiDimensional data model

This section introduces the MultiDimensional data model (MD for short). The model is based on the notion of *dimension* that allows to specify multiple "ways" to look at information, according to natural business perspectives under which its analysis can be performed. Each dimension is organized in a hierarchy of *levels*, corresponding to data domains at different granularities. A MultiDimensional *scheme* consists of a set of *f-tables* that are defined with respect to particular combinations of levels. A MultiDimensional *instance* associates *measures*, which correspond to data being tracked, with *symbolic coordinates* over f-tables.[1] Finally, within a dimension, values of a finer granularity can *roll up* to (that is, can be grouped into) values of a coarser one.

Example 1. A marketing analyst of a chain of toy stores may organize its business data along dimensions like time, product, and location. The time dimension may be organized in levels day, quarter, week, and year, and Feb 19, 97 is an element of the day level. The elements of this level roll up to elements of levels week and quarter. Similarly, both weeks and quarters roll up to years. Note however that weeks do not roll up to months, since months do not divide evenly into weeks. In this framework, a measure can be the number of items sold by the chain. This measure could be represented by means of an f-table SALES, having symbolic coordinates on the levels day, item, and store: an instance of SALES might state the fact that on Feb 19, 97 the store Colosseum has sold 11 pieces of Lego.

2.1 MultiDimensional schemes

Let us fix two disjoint countable sets of *names* and *values*. We denote by $\mathcal{L}$ a set of names called *levels* such that: (i) each level $l \in \mathcal{L}$ is associated with a countable set of values $\text{DOM}(l)$, called the *domain of* l; and (ii) the various domains associated with different levels are pairwise disjoint.

Definition 2 (Dimension). A *dimension* d is a triple $(L, \preceq, \text{R-UP})$, where:

- $L \subseteq \mathcal{L}$ is a finite set of levels;
- $\preceq$ is a partial order defined among the levels of d, inducing a lattice on L. Whenever $l_1 \preceq l_2$ we say that l_1 *rolls up to* l_2 or that l_2 *drills down to* l_1;
- R-UP is a family of functions, called *roll-up functions*, satisfying the following conditions:
  - for each pair of levels l_1, l_2 such that $l_1 \preceq l_2$, the roll-up function $\text{R-UP}_{l_1}^{l_2}$ maps each element of $\text{DOM}(l_1)$ to an element of $\text{DOM}(l_2)$. Whenever $\text{R-UP}_{l_1}^{l_2}(o_1) = o_2$ we say that o_1 *rolls up to* o_2, or that o_2 *drills down to* o_1;

[1] Actually, the 'f' in the term 'f-table' has a double meaning. On one hand, it stands for 'function', because each f-table is indeed a function, from coordinates to measures. On the other hand, it stands also for 'fact', since f-tables represent a form of information that practitioners implement by means of the so-called 'fact tables'.

- given levels l_1, l', and l_2 such that $l_1 \preceq l' \preceq l_2$, (and thus, $l_1 \preceq l_2$) the function $\text{R-UP}_{l_1}^{l_2}$ equals the composition $\text{R-UP}_{l'}^{l_2} \circ \text{R-UP}_{l_1}^{l'}$. This implies that: (i) for each level l, the function R-UP_{l}^{l} is the identity on $\text{DOM}(l)$; and (ii) whenever a level l_1 rolls up to l_2 in different ways (e.g., rolling up through either l' or l'') then the elements of l_1 roll up to elements of l_2 in a consistent way.

Example 3. Consider Example 1. The relevant information is organized along dimensions time, product, and location, and involves numeric data describing sales and prices. The dimension hierarchies are depicted on top of Figure 4; note that each dimension takes the name from one of its levels (often the least upper bound of its lattice). The figure shows that, e.g., level item rolls up to both category and brand; because of reflexivity, item rolls up also to itself and, because of transitivity, it rolls up to product. The domain associated with the level day contains, among others, values Jan 5, 97, Feb 19, 97, and Mar 10, 97, all of which roll up to the element 1Q-97 of the level quarter. Level store contains values Colosseum and Navona, both of them rolling up to Rome (in level city) and Italy (in level area). The level numeric is a built-in level type, having as domain the real numbers.

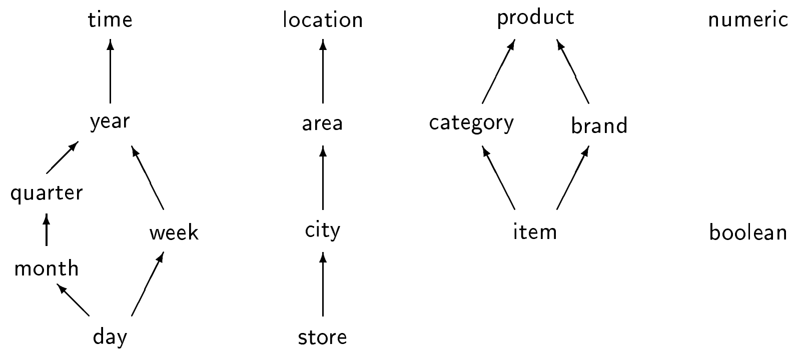
Definition 4 (Scheme). A *MultiDimensional scheme* $\mathcal{S}$ is a pair (D, F), where:

- D is a finite set of *dimensions*, with the condition that different dimensions have pairwise disjoint sets of levels;
- F is a finite set of *f-table schemes* of the form $f[A_1 : l_1, \ldots, A_n : l_n] : l_0$, where f is a name (with the condition that different f-table schemes have distinct names), each A_i, for $1 \leq i \leq n$, is a distinct name (called an *attribute of* f), and each l_i, for $0 \leq i \leq n$, is a level of some dimension in D.

Example 5. Figure 4 shows the MD scheme `Toys`, having two f-tables, named SALES and PRICE-LIST. Intuitively, the f-table SALES represents summary data for the sales of the chain in terms of pieces sold (dimension numeric), organized along dimensions time (at day level), location (at store level), and product (at item level). F-table PRICE-LIST is instead used to price the various items, assuming that prices may vary from month-to-month, and that different stores sell each item at the same price.

2.2 MultiDimensional instances

Definition 6 (Coordinate). Let $\mathcal{S} = (D, F)$ be a MultiDimensional scheme and $f[A_1 : l_1, \ldots, A_n : l_n] : l_0$ be an f-table scheme in F. A *(symbolic) coordinate* γ *over* f is a function mapping each attribute name A_i (with $1 \leq i \leq n$) to an element $o_i \in \text{DOM}(l_i)$. If γ is a coordinate over f such that $\gamma(A_i) = o_i$, for $1 \leq i \leq n$, we denote γ by $[A_1 : o_1, \ldots, A_n : o_n]$.



SALES [day : day, $item$: item, $store$: store] : numeric
PRICE-LIST [$item$: item, $month$: month] : numeric

Fig. 4. The sample `Toys` scheme.

Definition 7 (Instance). Let $\mathcal{S} = (D, F)$ be a MultiDimensional scheme and $f[A_1 : l_1, \ldots, A_n : l_n] : l_0$ be an f-table scheme in F. An *instance over* f is a partial function having finite graph, which maps coordinates over f to elements of $\text{DOM}(l_0)$. An *instance over* $\mathcal{S}$ is a function mapping each f-table f in F to an instance over f.

An *entry* of an f-table instance f is a coordinate over which f is defined. The actual value that f associates with an entry is called a *measure*.

It is apparent that our notion of "symbolic coordinate" is related with that of "tuple" in the relational model. This is motivated by the intuition that an f-table is a "logical" counterpart of the "physical" notion of a multi-dimensional array. It can be noted also that the notation we use for symbolic coordinates resembles subscripting into a multi-dimensional array (although in a non-positional way). There is however an important difference between f-tables and multi-dimensional arrays. Specifically, in arrays, "physical" coordinates vary over intervals within linearly-ordered domains (in particular, over initial segments of natural numbers), whereas we do not pose any restrictive hypothesis on the domains over which coordinates range. In this sense, our notion of coordinate is "symbolic."

Example 8. A possible instance for the sample scheme `Toys` defined in Example 5 is shown in Figure 5. Note that two different (graphical) representations for f-tables are used in the figure. A symbolic coordinate over the f-table SALES is [day : Jan 5, 97, item : Scrabble, store : Navona]. The actual instance associates the measure 32 with this entry.

Figure 5 suggests that several different representations of a same f-table are possible. A tabular representation for an f-table f (like the one used for SALES)

day	*item*	*store*	SALES
Jan 5, 97	Scrabble	Navona	32
Jan 5, 97	Risiko	Navona	27
Jan 5, 97	Lego	Sun City	42
Jan 5, 97	Risiko	Sun City	22
Feb 19, 97	Scrabble	Navona	32
Feb 19, 97	Lego	Navona	25
Feb 19, 97	Lego	Colosseum	11
Mar 10, 97	Risiko	Navona	5
Mar 10, 97	Lego	Sun City	6

PRICE-LIST	Jan-97	Feb-97	Mar-97
Lego	$12.^{99}$	$9.^{99}$	$9.^{99}$
Risiko	$14.^{99}$	$12.^{99}$	$12.^{99}$
Scrabble	$12.^{99}$	$12.^{99}$	$12.^{49}$
Trivia		$18.^{99}$	$17.^{99}$

Fig. 5. A sample instance over the `Toys` scheme.

consists of a relation over the attributes of f, plus a further column for the measures provided by the instance. This representation suggests a way to implement f-tables with the relational model. If an f-table has n attributes, it can be also represented as a n-dimensional array (like the one used for PRICE-LIST) in which an entry corresponds to a measure of the instance. This representation recalls the way in which multidimensional systems usually store data.

3 The MultiDimensional calculus by examples

In this section, we present MD-CAL, a query language for the MD model. This language is a calculus for f-tables, and allows the analyst to express analytical queries in a declarative way.

Interpreted scalar and aggregate functions can be used in queries, but the semantics of the language is parametric with respect to them. This gives us the freedom of choosing the most suitable collection of functions, according to the specific application domain. Then, given a collection $\mathcal{G}$ of interpreted functions, we denote by MD-CAL$^{\mathcal{G}}$ the MD query calculus that allows to use the functions in the collection $\mathcal{G}$.

The presentation is mainly based on examples that refer to the `Toys` sample scheme introduced in the previous section.

3.1 Basic queries

Intuitively, a *MultiDimensional query* is a mapping from instances over an input MD scheme to instances over an output MD scheme. The input and output schemes are defined over the same dimensions but distinct f-tables. For the sake of simplicity, we shall assume that the output scheme of a query contains just a single f-table, called the *output f-table* of the query.

If the output f-table of a query has scheme $f[A_1 : l_1, \ldots, A_n : l_n] : l$, then an MD-CAL *query* is specified by means of an expression of the following form.

$$\{x_1, \ldots, x_n : x \mid \psi(x, x_1, \ldots, x_n)\}$$

In the first part of the query, called the *target list*, $x, x_1, \ldots, x_n$ are distinct *variables*; the distinguished variable x is called the *result variable*. Furthermore, $\psi(x, x_1, \ldots, x_n)$ is a first-order formula in which $x, x_1, \ldots, x_n$ are the only free variables. The formula ψ is composed by equality atoms involving f-tables, roll-up functions, and interpreted scalar and aggregate functions.

Intuitively, the result of the query is an instance over the output f-table, associating a measure m to the entry $[A_1 : c_1, \ldots, A_n : c_n]$ for those values $m, c_1, \ldots, c_n$ that, respectively substituted to $x, x_1, \ldots, x_n$, satisfy the formula.

An important aspect in MD-CAL is what we call "definiteness" of queries. Intuitively, this property guarantees that queries define indeed f-tables, which, by definition, must be finite and satisfy a sort of functional dependency from coordinates to measures. We shall discuss on this issue in Section 4.3; for the time being, we present only examples that obviously satisfy this property.

As a first example, the following query is used to define an f-table

$$\textsc{Rome-Sales}[\mathit{day} : \mathsf{day}, \mathit{item} : \mathsf{item}, \mathit{store} : \mathsf{store}] : \mathsf{numeric}$$

to represent the same information as SALES, but limited to the stores in Rome.

$$\big\{x_1, x_2, x_3 : x \mid$$
$$x = \textsc{Sales}[\mathit{day} : x_1, \mathit{item} : x_2, \mathit{store} : x_3] \wedge \text{Rome} = \text{R-UP}_{\mathsf{store}}^{\mathsf{city}}(x_3)\big\}$$

3.2 Scalar functions

As we have said, an atom in the formula of a query can use a predefined set $\mathcal{G}$ of interpreted functions. This set can include system-defined or user-defined *scalar* functions, that is, functions that use only atomic values as inputs and outputs (e.g., all the standard mathematical operators, such as $+$ and $*$). Special care must be devoted in defining the semantics of a scalar function when one or more of its arguments is undefined. In what follows, unless otherwise stated, we will assume that the result of a function is undefined whenever one of its argument is undefined.

The following query defines the f-table with scheme

$$\textsc{Daily-Revenues}[\mathit{day} : \mathsf{day}, \mathit{item} : \mathsf{item}, \mathit{store} : \mathsf{store}] : \mathsf{numeric}$$

that represents the daily revenues, for each store and item. A measure for a certain item is obtained by multiplying the number of pieces sold in a day, by the price of the item in that month.

$$\big\{x_1, x_2, x_3 : x \mid$$
$$\exists x_4, x_5, x_6 \big(x_4 = \text{R-UP}_{\mathsf{day}}^{\mathsf{month}}(x_1) \wedge x_5 = \textsc{Price-List}[\mathit{item} : x_2, \mathit{month} : x_4] \wedge$$
$$x_6 = \textsc{Sales}[\mathit{day} : x_1, \mathit{item} : x_2, \mathit{store} : x_3] \wedge x = x_5 * x_6\big)\big\}$$

3.3 Aggregate functions

The set $\mathcal{G}$ can include also *aggregate* functions, that is, functions that applied to a collection of values yield an atomic value; these are of special interest in OLAP systems. Typical aggregate functions are those of SQL, that is, `min`, `max`, `count`, `sum`, and `avg`, which apply to expressions over columns.

For instance, the following query defines the f-table having scheme

$$\textsc{Summary-Sales}[\mathit{week} : \mathsf{week}, \mathit{item} : \mathsf{item}, \mathit{area} : \mathsf{area}] : \mathsf{numeric},$$

which represents summary data of sales, detailed by week, item, and area.

$$\begin{gathered}\big\{x_1, x_2, x_3 : x \mid \\ x = \mathtt{sum}\big(y_1, y_2 : y \mid y = \textsc{Sales}[\mathit{day} : y_1, \mathit{item} : x_2, \mathit{store} : y_2] \wedge \\ x_1 = \textsc{r-up}^{\mathsf{week}}_{\mathsf{day}}(y_1) \wedge x_3 = \textsc{r-up}^{\mathsf{area}}_{\mathsf{store}}(y_2)\big)\big\}\end{gathered}$$

The argument of the operator `sum` in the above query is a query itself, consisting of a target list (τ) and a formula (ψ). The target list τ specifies "local" variables, and the result variable is used for the aggregation. Intuitively, the result of the whole query is as follows. Let T_{c_1,c_2,c_3} be the set of tuples over y, y_1, y_2 that satisfy ψ when c_1, c_2, c_3 are substituted to x_1, x_2, x_3, respectively. Then, (m, c_1, c_2, c_3) is in the result if m equals the sum of the first component of the tuples in T_{c_1,c_2,c_3}.

Note that, similarly to SQL, only entries of f-tables (which are non-null by definition) take part of the computation of the sum.

Assume now that we want to compute the f-table having scheme

$$\textsc{Weekly-Revenues}[\mathit{week} : \mathsf{week}, \mathit{item} : \mathsf{item}, \mathit{store} : \mathsf{store}] : \mathsf{numeric},$$

which represents the weekly revenues, detailed by item and store. To do so, we can make use of the previously defined f-table Daily-Revenues, summarizing by weeks, as follows.

$$\begin{gathered}\big\{x_1, x_2, x_3 : x \mid \\ x = \mathtt{sum}\big(y_1 : y \mid y = \textsc{Daily-Revenues}[\mathit{day} : y_1, \mathit{item} : x_2, \mathit{store} : x_3] \wedge \\ x_1 = \textsc{r-up}^{\mathsf{week}}_{\mathsf{day}}(y_1)\big)\big\}\end{gathered}$$

However, we can also write the following query, which does not require the definition of Daily-Revenues.

$$\begin{gathered}\big\{x_1, x_2, x_3 : x \mid x = \mathtt{sum}\big(y_1 : y \mid x_1 = \textsc{r-up}^{\mathsf{week}}_{\mathsf{day}}(y_1) \wedge \\ \exists y_2, y_3, y_4 \big(y_2 = \textsc{r-up}^{\mathsf{month}}_{\mathsf{day}}(y_1) \wedge y_3 = \textsc{Price-List}[\mathit{item} : x_2, \mathit{month} : y_2] \wedge \\ y_4 = \textsc{Sales}[\mathit{day} : y_1, \mathit{item} : x_2, \mathit{store} : x_3] \wedge y = y_3 * y_4\big)\big)\big\}\end{gathered}$$

3.4 Abstraction queries

In the context of multi-dimensional data, it is often useful to transform measures into components of coordinates of f-tables, and vice versa. We call *abstractions* such transformations. The following example shows how to perform an abstraction in MD-CAL. The query generates the boolean f-table

$$\text{TOTAL-SALES}[\mathit{item} : \mathsf{item}, \mathit{store} : \mathsf{store}, \mathit{year} : \mathsf{year}, \mathit{items\text{-}sold} : \mathsf{numeric}] : \mathsf{boolean}$$

by summarizing on the number of sales and using the result as an element of a symbolic coordinate.

$$\{x_1, x_2, x_3, x_4 : x \mid x = \text{true} \wedge$$
$$x_4 = \mathtt{sum}\big(y_1 : y \mid y = \text{SALES}[\mathit{day} : y_1, \mathit{item} : x_1, \mathit{store} : x_2] \wedge x_3 = \text{R-UP}^{\mathsf{year}}_{\mathsf{day}}(y_1)\big)\}$$

Intuitively, the effect of this query is the following. For each triple c_1, c_2, c_3 of values over the variables x_1, x_2, x_3, the two formulas in the body are evaluated, using x_4 and x to hold the respective results, say, c_4 and m. Then m (that is, true) is assigned to the entry having coordinate $[c_1, c_2, c_3, c_4]$. Note that the f-table we obtain represents, in every respect, a relation of the relational model.

4 The MultiDimensional calculus

In this section we formally introduce the MultiDimensional query calculus MD-CAL. In what follows we fix a MultiDimensional scheme $\mathcal{S}$ and an instance $\mathcal{I}$ over $\mathcal{S}$. We also fix a collection $\mathcal{G}$ of scalar and aggregate interpreted functions. Each function in $\mathcal{G}$ is characterized by a *signature* and an *interpretation*. For a *scalar function* $g \in \mathcal{G}$, the signature has the form $g : l_1 \times \ldots \times l_n \to l$, where $l, l_1, \ldots, l_n$ are levels; an interpretation for g is a function from $\text{DOM}(l_1) \times \ldots \times \text{DOM}(l_n)$ to $\text{DOM}(l)$. For an *aggregate function* $h \in \mathcal{G}$, the signature has the form $h : 2^{l'} \to l$, where l and l' are levels; an interpretation for h is a function from finite *multi*-sets of elements in $\text{DOM}(l')$ to elements of $\text{DOM}(l)$.

4.1 Syntax

For each level l, assume the existence of a countable set of *variables of type* l.

The *terms* (over $\mathcal{S}$ and $\mathcal{G}$) and their respective types are recursively defined as follows.

- A variable of type l is a term of type l;
- a value in $\text{DOM}(l)$ is a term of type l;
- if t is a term of type l and l rolls up to a level l', then $\text{R-UP}^{l'}_{l}(t)$ is a term of type l';
- if $f[A_1 : l_1, \ldots, A_n : l_n] : l$ is an f-table scheme and $t_1, \ldots, t_n$ are terms of type $l_1, \ldots, l_n$, respectively, then $f[A_1 : t_1, \ldots, A_n : t_n]$ is a term of type l;
- if $g : l_1 \times \ldots \times l_n \to l$ is a scalar function and $t_1, \ldots, t_n$ are terms of type $l_1, \ldots, l_n$, respectively, then $g(t_1, \ldots, t_n)$ is a term of type l;
- if $h : 2^{l'} \to l$ is an aggregate function, and $\tau \mid \psi$ is a query (defined below) whose result variable is of type l', then $h(\tau \mid \psi)$ is a term of type l.

An *atom* (over $\mathcal{S}$ and $\mathcal{G}$) is an expression of the form $t = t'$, where t and t' are terms (over $\mathcal{S}$ and $\mathcal{G}$) of the same type. The *formulas* (over $\mathcal{S}$ and $\mathcal{G}$) are defined as follows.

- An atom is a formula;
- if ψ_1 and ψ_2 are formulas, then $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, and $\neg\psi_2$ are formulas;
- if ψ is a formula and x is a variable, then $\exists x(\psi)$ and $\forall x(\psi)$ are formulas.

The notions of *free* and *bound* occurrences of variables are as usual, with the following additional consideration: the variables in the target list of an aggregation term are bound outside the term.

An MD-CAL *query* is an expression of the form

$$\big\{x_1, \dots, x_n : x \mid \psi(x, x_1, \dots, x_n)\big\},$$

where $\psi(x, x_1, \dots, x_n)$ is a formula having $x, x_1, \dots, x_n$ as distinct free variables. The expression $x_1, \dots, x_n : x$ is called the *target list*, and x the *result variable*.

4.2 Semantics

Let q be an MD query of the form $\big\{x_1, \dots, x_n : x \mid \psi(x, x_1, \dots, x_n)\big\}$. The *pre-result* of q on $\mathcal{I}$, denoted by $\textsc{pre}(q(\mathcal{I}))$, is the set of tuples of values $c, c_1, \dots, c_n$ that, respectively substituted to $x, x_1, \dots, x_n$, satisfy the formula ψ with respect to $\mathcal{I}$. In such tuples, the first component c is called the *result value*.

The notion of *satisfaction* of a formula with respect to a substitution θ and an instance $\mathcal{I}$ is defined in the usual way, with the following considerations.

- The substitutions are typed, so that variables vary over values of the corresponding types. For the time being, we assume that values are chosen from the domain $\textsc{dom}(\mathcal{S})$, that is, the union of the domains of the levels occurring in $\mathcal{S}$.
- Consider an atom of the form $t = h(\tau \mid \psi)$, where h is an aggregate function, and a substitution θ over the free variables of the atom. Let T be the pre-result of the query $\big\{\tau \mid \theta(\psi)\big\}$ over $\mathcal{I}$ and let M be the multi-set containing the result values of T, with the respective multiplicities. Then, the atom is satisfied if $\theta(t) = h(M)$.

Thus, the pre-result of an MD-CAL query is a set of tuples, to be used as coordinates and measures of the result f-table. This is however not always possible, since there are pre-results that do not correspond to f-table instances. We say that the pre-result of a query over an instance is *functional* if it does not contain a pair of different tuples that coincide on all values, but the result value.

Let q be a query having $f[A_1 : l_1, \dots, A_n : l_n] : l$ as output scheme. If the pre-result $F = \textsc{pre}(q(\mathcal{I}))$ of q is functional, then we can build in the natural way an f-table instance $\textsc{ft}(F)$ from it, as follows. For each tuple $c, c_1, \dots, c_n$ in F, $\textsc{ft}(F)$ associates the result value c to the symbolic coordinate $[A_1 : c_1, \dots, A_n : c_n]$. Then, the *result* of q over $\mathcal{I}$, denoted by $q(\mathcal{I})$ is defined as $\textsc{ft}(\textsc{pre}(q(\mathcal{I})))$.

4.3 Definiteness

Apart from functionality, the result of a query should satisfy another important property: the finiteness of the result. Actually, in the context of the relational calculus, a more general notion, the *domain independence*, has been defined to capture the finiteness of queries. In this section, we introduce and discuss the issue of *definiteness* as a desirable property for MD-CAL queries: intuitively, this notion combines the properties of domain independence (in the context of the MD model) and functionality.

Indeed, the notion of domain independence has been further generalized for queries involving interpreted functions, in particular, to *bounded depth domain independence* [1]. Now, it is straightforward to define the result of an MD-CAL query relativized to a domain D rather than to the domain $\mathrm{DOM}(\mathcal{S})$. Then, we can say that, intuitively, an MD-CAL query q (using functions from a collection $\mathcal{G}$) is bounded depth domain independent if, for any instance $\mathcal{I}$, its result depends only on a domain including the active domain $\mathrm{ADOM}(q, \mathcal{I})$ of $\mathcal{I}$ and of q, plus a further small set of values obtained by applying a bounded number of times the roll-up functions and the functions in $\mathcal{G}$ to $\mathrm{ADOM}(q, \mathcal{I})$.

We say that an MD-CAL query q is *definite* if, for any input instance $\mathcal{I}$, it is bounded depth domain independent and functional.

Syntactic characterizations that ensure bounded depth domain independence have been proposed, for instance, *embedded allowedness* [10]. On the other hand, the property of functionality can be reduced to a problem of implication of *functional dependencies* for the MD-CAL language.

Example 9. Let us consider the query in Section 3.2 defining the f-table DAILY-REVENUES. Intuitively, this query is bounded depth domain independent since: (i) the variables x_1, x_2, x_3 are bounded to values occurring in the input instance; (ii) the variable x_4 is bounded to values that can be obtained by applying a roll-up function to a bounded variable; and (iii) the variable x is bounded to values that can be obtained by a single application of the scalar function $*$. Moreover, the query is functional since the functional dependency $x_1, x_2, x_3 \to x$ is implicated by the the facts that: (i) x_4 functionally depends on x_1, because of the roll-up function; and (ii) x functionally depends on x_1, x_2, x_3, x_4, because of the application of a scalar function to two measures that functionally depends on x_1, x_2, x_3 and x_2, x_4, respectively. Hence, the query is definite.

If we restrict MD-CAL to queries involving no functions (neither roll-up nor interpreted ones), definiteness of MD-CAL queries corresponds to domain independence and functionality in the context of the relational model. It is well-know that both properties are undecidable, but become decidable if the language is restricted to *positive existential* calculus queries [2]. It is also clear that definiteness is undecidable in MD-CAL, but decidable in positive existential MD-CAL without functions. We can show that definiteness is decidable for positive existential MD-CAL queries involving roll-up functions.

5 Related work

The term OLAP has been recently introduced by Codd et al. [8] to characterize the category of analytical processing over large, historical databases (data warehouses) oriented to decision making. Further discussion on OLAP, multi-dimensional analysis, and data warehousing can be found in [6,14,21,23].

An important OLAP operation is summarization of data over one or more dimensions (roll up). Klug [16] provided a first theoretical basis in this respect, by extending the relational algebra and calculus with aggregate functions, that is, interpreted functions taking a set of tuples as argument and producing a single value as result. Our approach is more general than Klug's one, since we consider, in addition to aggregate functions, also more general functions, which produce structured results, and scalar functions.

Many authors [5,11,19,22] have noticed that SQL is unsuited to data-analysis applications, since some aggregate and grouping queries are difficult to express and optimize. Thus, they have considered the problem of extending SQL with aggregation and analysis-oriented operators that are more powerful, but also specific to particular application domains. Gray et al. [11] proposed `cube` as an operator generalizing `group by`. Chatziantoniou and Ross [5] have extended both SQL and the relational algebra with an operator, called just Φ, dealing with "aggregation variables", to succinctly express common queries, providing also a basis for improved query optimization. Rao et al. [19] considered the issue of supporting quantified queries, a class of queries that is difficult to deal with in SQL; they introduced a number of "generalized quantifiers", such as `some`, `all`, and `at-least`, which are essentially boolean aggregate functions over sets. Many of the features considered in these proposals can be easily expressed in our language using a limited collections of scalar and aggregate interpreted functions.

Agrawal at al. [4] have proposed a simple hypercube-based data model, and a few algebraic operators for this model. Actually, this approach is oriented to a direct SQL implementation into a relational database, and therefore it is less general than ours.

Libkin et al. [17] have defined a language for querying data organized in multi-dimensional arrays, to support the scientific computing community with database technology. The MultiDimensional model is at a different, and perhaps higher, abstraction level; our notion of f-table is indeed a "logical" counterpart of a "physical" multi-dimensional array. It should be noted also that our approach is motivated by a business context.

Gyssens et al. [12] have proposed the tabular database model, together with a complete algebraic language for querying and restructuring, as a first theoretical foundation for OLAP systems. A main difference with respect to their approach is that we introduce an explicit logical notion of dimension, allowing for multi-dimensional structures, whereas their tables are bidimensional. Their query language covers only the aspect of restructuring, whereas we allow complex computations based on formulas and functions.

6 Research issues and current work

In this section we discuss the main research topics that are currently being investigated in the context of multi-dimensional databases and OLAP systems.

Dimensional modeling. This area focuses on how information can be effectively organized according to natural business concepts (that is, the way decision-makers look at their business data) to enable decision support. It should be said that, on one hand, this is a data analysis issue, with the need for a suitable conceptual model for multi-dimensional databases and methodologies for their development. On the other hand, the problem is strictly related with the other main topic of optimization, since data needs to be organized with the goal of minimizing the effort required to retrieve significant information and generate meaningful, integrated reports from it.

Optimization. This issue concerns the ways in which factual data can be efficiently stored and manipulated in multidimensional databases. There are two main approaches to this problem in the context of decision-support applications: materialization of views and query optimization. The first approach is generally based on a preliminary phase in which relations are denormalized and some data is aggregated. This technique is widely used in the context of OLAP systems, since it can greatly improve performance, especially in a context of read-only data. However, storing summary information at every possible level is too time-consuming and resource-intensive. Thus, it is challenging to select the right set of queries to materialize, taking into account the expected load of the database, but also the effective usage pattern [13,20,21]. Optimization of complex queries is another important and very difficult task, especially in presence of aggregate functions [3] and user-defined functions [7]. It involves also the study of specialized indexing techniques [18], join and scan methods, and its extension in the context of parallel systems.

Query languages. OLAP systems require a number of query languages, at different abstraction levels. On one hand, the final user should be enabled to perform point-and-click operations by means of graphical metaphores. Typical ways of manipulating a multi-dimensional data collection are the following: roll up (summarize data), drill down (go to more detailed data), slice and dice (select and project on a bidimensional view), pivot (reorient a data cube, projecting on different dimensions). On the other hand, the sophisticated user that needs to express more complex queries should be allowed to use a declarative, high-level language, such as a calculus or an extension of SQL. Finally, query optimization can be effectively performed by referring to a procedural, algebraic language. Thus, a family of different languages should by adopted by an OLAP system, and mapping between them should be defined.

The formal nature of the MultiDimensional model, proposed in this paper, is well-suited for an investigation of several problems mentioned above. In particular, we are currently developing an algebra for the MD model, to study the efficient evaluation of multidimensional queries, possibly in presence of some materialized views. We are also investigating special methodologies for minimizing redundancy and inconsistency in the context of MD databases.

References

1. S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. Technical Report 846, INRIA, 1988.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
3. S. Agarwal et al. On the computation of multidimensional aggregates. In *Twenty-second Int. Conf. on Very Large Data Bases, Bombay*, pages 506–521, 1996.
4. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. Research report, IBM Almaden Research Center, 1996.
5. D. Chatziantoniou and K. Ross. Querying multiple features of groups in relational databases. In *Twenty-second Int. Conf. on Very Large Data Bases, Bombay*, pages 295–306, 1996.
6. S. Chaudhuri and U. Dayal. Decision support, Data Warehousing, and OLAP. In *Tutorials of the Twenty-second Int. Conf. on Very Large Data Bases*, 1996.
7. S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In *Twenty-second Int. Conf. on Very Large Data Bases, Bombay*, pages 87–98, 1996.
8. E.F. Codd, S.B. Codd, and C.T. Salley. Providing OLAP (On Line Analytical Processing) to user-analysts: An IT mandate. Arbor Software White Paper, *http://www.arborsoft.com.*
9. G. Colliat. OLAP, relational, and multidimensional database systems. *ACM SIGMOD Record*, 25(3):64–69, September 1996.
10. M. Escobar-Molano, R. Hull, and D. Jacobs. Safety and translation of calculus queries with scalar functions. In *Twelfth ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems*, pages 253–264, 1993.
11. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Twelfth IEEE International Conference on Data Engineering, Vienna*, pages 152–159, 1996.
12. M. Gyssens, L.V.S. Lakshmanan, and I.N. Subramanian. Tables as a paradigm for querying and restructuring. In *Fifteenth ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems*, pages 93–103, 1996.
13. V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD International Conf. on Management of Data*, pages 205–216, 1996.
14. W.H. Inmon. *Building the Data Warehouse.* John Wiley, second edition, 1996.
15. T. Isakowitz, S. Schocken, and H.C. Lucas. Toward a logical/physical theory of spreadsheet modeling. *ACM Trans. on Inf. Syst.*, 13(1):1–37, January 1995.
16. A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, 1982.
17. L. Libkin, R. Machlin, and L. Wong. A query language for multidimensional arrays: Design, implementation, and optimization techniques. In *ACM SIGMOD International Conf. on Management of Data*, pages 228–239, 1996.
18. P. O'Neill and G. Graefe. Multi-table joins through bitmapped join indexes. *ACM SIGMOD Record*, pages 8–11, September 1995.

19. S. Rao, A. Badia, and D. Van Gucht. Providing better support for a class of decision support queries. In *ACM SIGMOD International Conf. on Management of Data*, pages 217–227, 1996.
20. D. Srivastava, S. Dar, H.V. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Twenty-second Int. Conf. on Very Large Data Bases, Bombay*, pages 318–329, 1996.
21. Stanford Technology Group, Inc. Designing the data warehouse on relational databases, 1995. White paper.
22. Red Brick Systems. Decision-makers, business data, and RISQL, 1995. White paper, *http://www.redbrick.com*.
23. J.L. Weldon. Managing multidimensional data: Harnessing the power. *Database Programming & Design*, 8(8):24–33, August 1995.