From a Procedural to a Visual Query Language for OLAP *

Luca Cabibbo and *Riccardo Torlone*
Dipartimento di Informatica e Automazione
Università di Roma Tre

Abstract

We address the issue of designing effective query languages for OLAP databases. The basis of our investigation is $\mathcal{MD}$, a new data model for multidimensional databases that, unlike other multidimensional models, is independent of any specific implementation and as such provides a clear separation between practical and conceptual aspects. In this framework, we present and compare two query languages, based on different paradigms, for OLAP databases. The first language is algebraic and provides an effective way to manipulate multidimensional data in a procedural fashion. Although this language is clean and powerful, it is clearly not suited for final users. We therefore propose a high-level graphical language that allows the user to specify analytical queries in a natural and intuitive way. It turns out that the two languages have the same expressive power.

1. Introduction

Multidimensional databases are large collections of data, used for statistical analyses oriented to decision making, in which factual data is described according to different perspectives or "dimensions." For instance, in a commercial enterprise, single sales of items (the factual data) can be effectively analyzed when organized according to dimensions like category of product, geographical location, and time.

Multidimensional analysis is supported by an emerging category of software technology, called On-Line Analytical Processing (OLAP) systems. These environments allow the user to easily summarize and view data, but suffer from some limitation in constructing and maintaining complex analytical models over the enterprise data. Indeed, to increase their effectiveness, OLAP systems should support the users with several query languages, possibly at different abstraction levels. On one hand, the final user should be enabled to perform point-and-click operations by means of graphical metaphors. On the other hand, the sophisticated user that needs to express more complex queries should be allowed to use a declarative, high-level query language, such 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 be an OLAP system, and mapping between them should be defined.

In this paper, we present and investigate different query languages for OLAP databases. The framework for our investigation is $\mathcal{MD}$, a logical model for OLAP systems proposed in [3]. This model includes a number of concepts that generalize the notions of dimensional hierarchies, fact tables, and measures, commonly used in commercial systems. In this framework, we propose two query languages, based on different paradigms, for multidimensional databases. We start by considering a procedural query language, based on an algebra, similar in spirit to relational algebra. This language provides many insights on the way in which multidimensional data can be manipulated and, for its procedural nature, can be profitably used to specify query optimization. However, it is clearly unsuited for a direct use. We then investigate another query language, based on a graphical paradigm, that lies at a higher abstraction level with respect to the algebra. By using several examples, we show that this language yields a declarative and easy-to-use tool to query multidimensional databases, since it allows the user to manipulate multidimensional data in a natural and intuitive way. We then provide a mapping between graphical queries and algebraic expressions and prove the equivalence of the expressive power of the two languages. The mapping can be effectively used to perform optimizations over graphical queries. It turns out that, in both cases, $\mathcal{MD}$ is well-suited for specifying OLAP queries, since it allows the user to abstract from implementation details.

The paper is organized as follows. The $\mathcal{MD}$ data model is briefly presented in Section 2. An algebra and a graphical query language for $\mathcal{MD}$ are first introduced informally, by means of examples, in Section 3. Then, in Sections 4 and 5, we present, in a more systematic way, their syntax and semantics and state their equivalence. In Section 6, we

*This work was partially supported by *CNR* and by *MURST*.

briefly compare our work with other approaches and finally, in Section 7, we draw some conclusions.

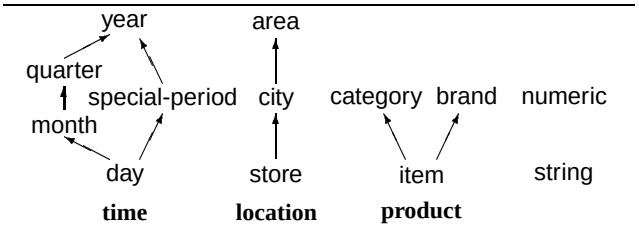
2. The $\mathcal{MD}$ data model

The MultiDimensional data model ($\mathcal{MD}$ for short) is based on two main constructs: dimension and f-table. *Dimensions* are syntactical categories that allow us to specify multiple "ways" to look at the 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 granularity. When a level l_1 precedes a level l_2 (in symbols, $l_1 \preceq l_2$) in the hierarchy we say that l_1 *rolls up to* l_2. A level can have *descriptions* associated with it. Within a dimension, values of different levels are related through a family of *roll-up functions*. If a roll-up function associates a value v_1 of a certain level to a value v_2 of an upper level in the hierarchy, we also say that v_1 *rolls up to* v_2. *F-tables* are partial functions from *symbolic coordinates* (defined with respect to particular combinations of levels) to *measures*: they are used to represent factual data. An *entry* of an f-table f is a coordinate over which f is defined.

For example, as shown on top of Figure 1, a marketing analyst of a chain of toy stores may organize its business data along dimensions **time**, **product**, **location**. The **time** dimension is organized in a hierarchy of levels involving day, month, quarter, year, and special-period. The domain associated with the level day contains, among others, values *Jan 5, 98*, *Feb 19, 98*, and *Mar 10, 98*, all of which roll up to the element *1Q-98* of the level quarter. Similarly, the **location** dimension is based on a hierarchy of levels involving store, city, and area. The level store contains, for instance, values *Colosseum* and *Navona*, both of them rolling up to *Rome* (in level city) and *Italy* (in level area). A description of the level store, in the **location** dimension, can be its *address*. Finally, the **product** dimension contains levels item, category, and brand. According to the corresponding hierarchy, any element of the level item rolls up to both a brand and a category. Note that there are two further *atomic* dimensions (that is, having just one level) that are used to represent **numeric** values and **string**s.

In this framework, we can define, for instance, the f-tables SALES and COSTOFITEM. The former describes summary data for the sales of the chain, organized along dimensions **time** (at day level), **product** (at item level), and **location** (at store level). The measures for this f-table are *NSales* (the number of items sold) and *Income* (the gross income), both having type numeric. The f-table COSTOFITEM is instead used to represents the costs of the various items, assuming that costs may vary from month-to-month.

A possible instance for these f-tables is shown in Figure 2. Note that two different (graphical) representations



Address (store) : string, *Name* (item) : string

SALES[*Period* : day, *Product* : item, *Location* : store] → [*NSales* : numeric, *Income* : numeric]

COSTOFITEM[*Product* : item, *Month* : month] → [*Cost* : numeric]

Figure 1. Dimensions, descriptions and f-tables of a sample MD scheme

are used in the figure: a table and an array. A symbolic coordinate over the f-table SALES is [day : *Jan 5, 98*, item : *Scrabble*, store : *Navona*]. The actual instance associates with this entry the value 32 for the measure *NSales* and the value $543.^{68}$ for the measure *Income*.

SALES

Period	*Product*	*Location*	*NSales*	*Income*
Jan 5, 98	*Scrabble*	*Navona*	32	$543.^{68}$
Jan 5, 98	*Risiko*	*Navona*	27	$512.^{73}$
Jan 5, 98	*Lego*	*Sun City*	42	$713.^{58}$
Jan 5, 98	*Risiko*	*Sun City*	22	$439.^{78}$
Feb 19, 98	*Scrabble*	*Navona*	32	$479.^{68}$
Feb 19, 98	*Lego*	*Navona*	25	$299.^{75}$
Feb 19, 98	*Lego*	*Colosseum*	11	$142.^{89}$
Mar 10, 98	*Risiko*	*Navona*	5	$69.^{95}$
Mar 10, 98	*Lego*	*Sun City*	6	$71.^{94}$

COSTOFITEM

Cost	*Jan-98*	*Feb-98*	*Mar-98*
Lego	$12.^{99}$	$9.^{99}$	$9.^{99}$
Risiko	$14.^{99}$	$12.^{99}$	$12.^{99}$
Scrabble	$12.^{99}$	$12.^{99}$	$12.^{49}$
Trivia		$18.^{99}$	$17.^{99}$

Figure 2. An MD instance

Roll-up functions are a distinctive feature of our model: they describe *intensionally* how values of different levels are related. Moreover, as we will demonstrate shortly, they provide a powerful tool for querying multidimensional data, since allow us to specify how data must be aggregated, and how f-tables involving data at different levels of granularity can be joined.

3. Querying $\mathcal{MD}$ Databases

We now informally present and compare two query languages for OLAP databases in the context of the $\mathcal{MD}$ model. The presentation is mainly based on examples that refer to the sample scheme introduced in the previous section.

The first language is an algebra for the $\mathcal{MD}$ model and provides an effective way to manipulate f-tables in a procedural fashion. This language is based on a set of operators over f-tables, similar in spirit to the relational algebra ones. Actually, most of them are just generalizations of relational operators; others are specific of the multidimensional framework.

The second query language has a graphical nature, and it is suitable for end-users since it provides a simple way for specifying queries over a multidimensional database. This language is based on the observation that there is a natural way to describe an $\mathcal{MD}$ f-table with a graph that we call *f-graph*. Consider for instance the SALES f-table defined in Figure 1; its scheme can be represented by the f-graph reported in Figure 3. In this representation, the ovals denote

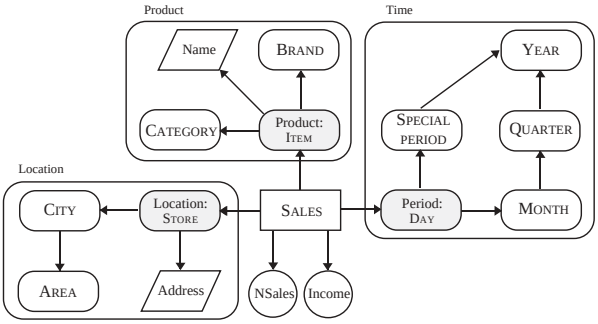


Figure 3. A graphical representation of an f-table scheme

levels of dimensions (which are represented by hypernodes of the given graph), the parallelograms denote level descriptions, and the circles denote measures of f-tables. The central node is called the *f-node* and represent the entries of the f-table. The various nodes of the graph are connected by directed arcs that represent functional relationships; for instance, an arc between two levels represents a roll-up function, whereas an arc between the fact node and a measure node represents the function associating a measure with the entries of the f-table. Note that the levels on which the f-table is defined have a further label (the corresponding attribute name) and are emphasized, but also the other levels of each dimension have been represented. As we will show shortly, this is useful for specifying roll-up operations. Finally, for measures and level descriptions we have just indicated the corresponding names.

3.1. Basic Queries

Intuitively, an $\mathcal{MD}$ *query* is a mapping from instances over an input $\mathcal{MD}$ scheme to instances over an output $\mathcal{MD}$ 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 *output f-table*.

As a first example, assume we need to define an f-table ROMESALES (the output f-table) having scheme

$$[\mathit{Period} : \mathsf{day}, \mathit{Product} : \mathsf{item}, \mathit{Location} : \mathsf{store}] \rightarrow [\mathit{NSales} : \mathsf{numeric}]$$

to represent the number of sales of each item in each day, only for the stores in Rome. It is easy to see that this can be obtained from the SALES f-table, having scheme

$$[\mathit{Period} : \mathsf{day}, \mathit{Product} : \mathsf{item}, \mathit{Location} : \mathsf{store}] \rightarrow [\mathit{NSales} : \mathsf{numeric}, \mathit{Income} : \mathsf{numeric}]$$

which is therefore the input f-table of the query.

In the algebra, we first need to extend the input f-table with a new attribute over the level city, holding the city in which each store is located. This operation can be specified with the special *roll-up* operator $\varrho^{A_2:l_2}_{A_1:l_1}(\mathrm{F})$, which extends an f-table F involving an attribute A_1 over a level l_1, with a new attribute A_2 over a level $l_2 \succeq l_1$, making use of the roll-up function from l_1 to l_2. We can then perform a selection over the new attribute, and finally project out the unneeded attributes and measures. All of this can be specified with the following algebraic expression.

$$\pi_{[\mathit{Period},\mathit{Product},\mathit{Location}] \rightarrow [\mathit{NSales}]}\,(\sigma_{\mathit{City}=\mathit{Rome}}(\varrho^{\mathit{City}:\mathsf{city}}_{\mathit{Location}:\mathsf{store}}(\mathrm{SALES})))$$

In the graphical language, we can specify the same query with a sequence of graphs that denotes a cascade of scheme restructurings: the first graph, called the *source*, describes the input scheme; the last graph, called the *target*, describes the output scheme; the intermediate ones (called *s-graphs*) form the *query specification* and describe the intended transformations.

In the example above, the source is the graph in Figure 3, the target is reported on the bottom of Figure 4, and the query specification is composed by one s-graph only, shown on the top of Figure 4. In s-graphs, nodes can be *marked* and *labeled*. Marks specify the levels and the measures of the graph that follows in the sequence (in our case of the target). Labels specify selections and renamings: a label preceded by a comparison predicate specifies a selection over the corresponding node, whereas a label involving

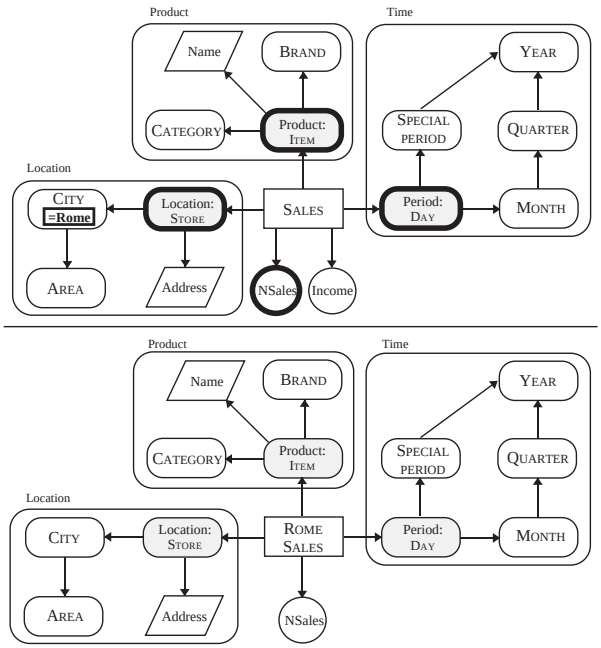


Figure 4. A simple graphical query

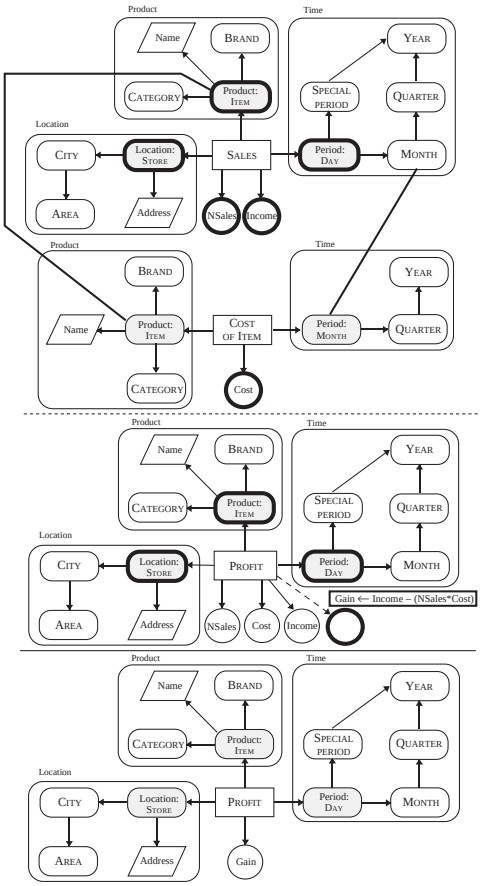


Figure 5. A query involving scalar functions

an arrow specifies a renaming. In our example, we have marked the levels (store, item, and day) and the measures (*NSales*), which are of interest in the output, and we have specified a selection over the level city. The name of the resulting f-table is ROMESALES. Note that it is possible to specify a selection over a node that is not marked, since we assume that the needed roll-up is performed automatically by the system.

From an operational point of view, we forecast the possibility to specify marks and labels directly over an s-graph of the sequence forming a query, and the automatic derivation of the "skeleton" of the s-graph that follows.

3.2. Scalar Functions

An important feature needed in querying numeric data is the ability to apply scalar functions. A scalar function makes use of atomic values as input and output (e.g., all the standard mathematical operators, such as $+$ and $*$). For instance, scalar functions can be used to obtain, from our sample database, an f-table PROFIT over the scheme

$$[\mathit{Period} : \mathsf{day}, \mathit{Product} : \mathsf{item}, \mathit{Location} : \mathsf{store}] \rightarrow [\mathit{Gain} : \mathsf{numeric}]$$

representing the daily profit, for each store and item. In fact, this value can be obtained, for each item, by multiplying the number of pieces sold by a store in a day, by the cost of the item in the corresponding month (recall that the costs are given on a monthly basis). To this end, we first need to roll up the attribute *Period* of the f-table SALES to the level month (defining a new attribute *Month*), and then join the resulting f-table with the f-table COSTOFITEM, having scheme

$$[\mathit{Product} : \mathsf{item}, \mathit{Month} : \mathsf{month}] \rightarrow [\mathit{Cost} : \mathsf{numeric}],$$

over the common attributes *Product* and *Month*. We can now compute the desired values by applying the formula $\mathit{Income} - (\mathit{NSales} * \mathit{Cost})$ over the result of the join. This can be done by means of a special operator $\varphi^{M=\phi}(\mathrm{T})$ that extends the scheme of the f-table T with a new measure M obtained by applying the scalar function ϕ to each entry of T. The result is finally obtained with a projection over the needed attributes and measures. In sum, we have the following expression:

$$\pi_{[\mathit{Period},\mathit{Product},\mathit{Location}]\rightarrow[\mathit{Gain}]}\left(\varphi^{\mathit{Gain}=\mathit{Income}-(\mathit{NSales}*\mathit{Cost})}\left(\mathrm{COST} \bowtie \varrho^{\mathit{Month}:\mathsf{month}}_{\mathit{Period}:\mathsf{day}}(\mathrm{SALES})\right)\right).$$

The same query can be specified in a graphical way as

described in Figure 5. The source (which is not shown) consists of a disconnected graph describing the input f-tables: SALES and COSTOFITEM. The query specification consists of the two s-graphs depicted on top of Figure 5. The first one specifies the join of the input f-tables by means of two arcs that connect the levels over which the join has to be performed. Also in this case, we assume that the system is able to execute automatically the needed roll-up operation over the f-table SALES. The scheme of the result of the join forms the skeleton of the second s-graph of the query specification. This s-graph introduces a new measure (denoted by a dashed arc) that is computed from the other measures by the corresponding formula; again, marks are used to specify levels and measures of interest. The result of this transformation is the target of the query, shown on the bottom of Figure 5.

3.3. Aggregate Functions

Aggregate functions are of special interest in OLAP systems: they take as input a collection of values and return an atomic value. Typical aggregate functions are those of SQL, that is, `min`, `max`, `count`, `sum`, and `avg`, which apply to columns of relational tables.

An aggregate function can be used, in our running database, to define the f-table SUMMARYSALES over the scheme

$$[\mathit{Period} : \mathsf{month}, \mathit{Location} : \mathsf{area}, \mathit{Product} : \mathsf{item}] \rightarrow [\mathit{TotSales} : \mathsf{numeric}]$$

representing summary data of sales, detailed by month, item, and area. This f-table can be obtained with the algebra by first using the roll-up operator to extend the f-table SALES with two new attributes holding area and month of each sale, and then applying the `sum` aggregate function to *NSales* while grouping by month, area, and product. The aggregation can be specified with a special operator $\psi^{M=g(M_i)}_{A_1,\ldots,A_k}(\mathrm{T})$, whose result contains just the attributes $A_1, \ldots, A_k$ of T and the new measure *M*, obtained by applying the aggregate function g to the measure M_i of T, grouping over the attributes $A_1, \ldots, A_k$. We then have the following expression, in which the outer operator ρ simply performs a renaming of attributes:

$$\rho_{\mathit{Area}\rightarrow\mathit{Location},\mathit{Month}\rightarrow\mathit{Period}}(\psi^{\mathit{TotSales}=\mathsf{sum}(\mathit{NSales})}_{\mathit{Month},\mathit{Area},\mathit{Product}}(\varrho^{\mathit{Area}:\mathsf{area}}_{\mathit{Location}:\mathsf{store}}(\varrho^{\mathit{Month}:\mathsf{month}}_{\mathit{Period}:\mathsf{day}}(\mathrm{SALES}))))$$

The same query can be specified in a graphical way as described in Figure 6. Again, the source is the f-graph shown in Figure 3. The query specification consists of a single s-graph, reported on top of Figure 6. In this s-graph, the nodes area, item, and month have been marked to specify the desired levels of the target scheme. Since the granularity of these levels is coarser than the levels of the source (the emphasized ones) this operation also requires the specification of an aggregation over the selected measures. This is done by labeling the node *NSales* with the aggregate function `sum` and with the name *TotSales* of the resulting measure. The target of the graphical query is shown on the bottom of Figure 6. Note that, in this f-graph, only the levels that can be used to specify further aggregations (or selections) are shown.

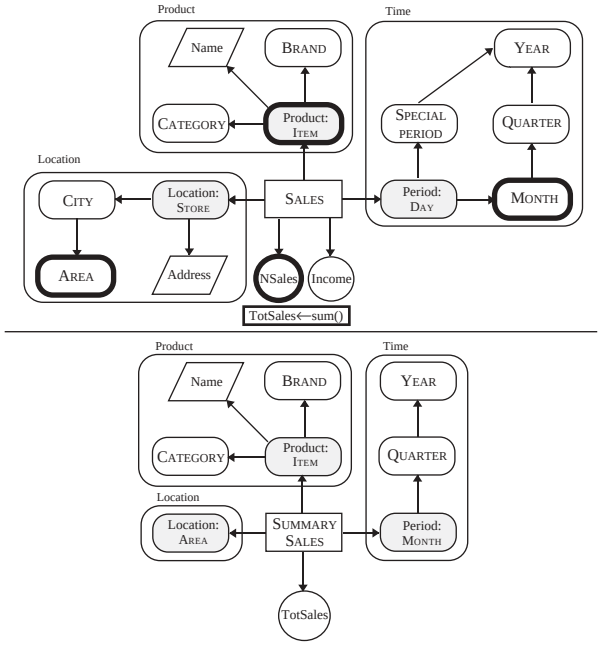


Figure 6. A query involving an aggregate function

Assume now that we want to compute the f-table MONTHLYPROFIT to represent the monthly profit, detailed by item, defined over the scheme

$$[\mathit{Period} : \mathsf{month}, \mathit{Product} : \mathsf{item}] \rightarrow [\mathit{TotGain} : \mathsf{numeric}].$$

To do so, we can use the above defined f-table PROFIT, summarizing by months over all stores, as follows.

$$\rho_{\mathit{Month}\rightarrow\mathit{Period}}(\psi^{\mathit{TotGain}=\mathsf{sum}(\mathit{Gain})}_{\mathit{Product},\mathit{Month}}(\varrho^{\mathit{Month}:\mathsf{month}}_{\mathit{Period}:\mathsf{day}}(\mathrm{PROFIT})))$$

The same query can be specified in a graphical way as described in Figure 7. The source of this query is the target of the query reported in Figure 5, and its specification needs just one s-graph describing the levels of aggregation, the aggregate function, and some renaming.

This example shows that both graphical and algebraic queries can be composed, since the target of a query can be used as source of another one.

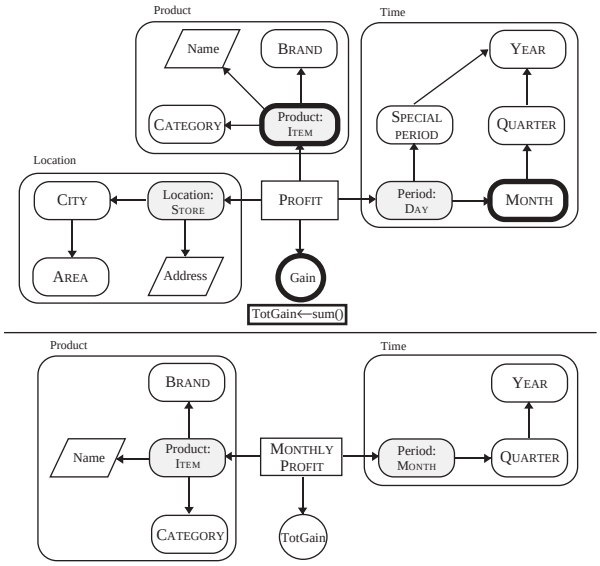


Figure 7. A query over a derived f-table

3.4. Abstraction Queries

In the context of multidimensional data, it is often useful to transform measures into attributes of f-tables, and vice versa. This allows the user to change dynamically the perspectives under which the analysis is performed. We call *abstractions* such transformations.

Assume for instance that we intend to define an f-table TOTALSALES with no measures having scheme

$$[Prod : \mathsf{item}, Loc : \mathsf{store}, Per : \mathsf{year}, TS : \mathsf{numeric}] \rightarrow [\,].$$

This f-table can be obtained from SALES by first summarizing on the number of sales with a roll-up operation, and then applying the abstraction operator $\alpha_M(\mathrm{T})$, which "promotes" the measure M of the f-table T to an attribute. The resulting expression is the following:

$$\rho_{Year \rightarrow Period}(\alpha_{TotSales}(\psi^{TotSales=\mathsf{sum}(NSales)}_{Product,Location,Year}(\varrho^{Year:\mathsf{year}}_{Period:\mathsf{day}}(\mathrm{SALES}))))$$

The same query can be specified in a graphical way as described in Figure 8. It is possible to see that the graphical specification requires two s-graphs. The first s-graph specifies the aggregation over the desired levels. The second one defines a new dimension starting from a measure of the intermediate result of the query. The abstraction is specified by the creation of a new node, defined with respect to an existing node, that belongs to a new dimension named. Similarly to the creation of new measures, dimension creation is denoted by a dashed arc.

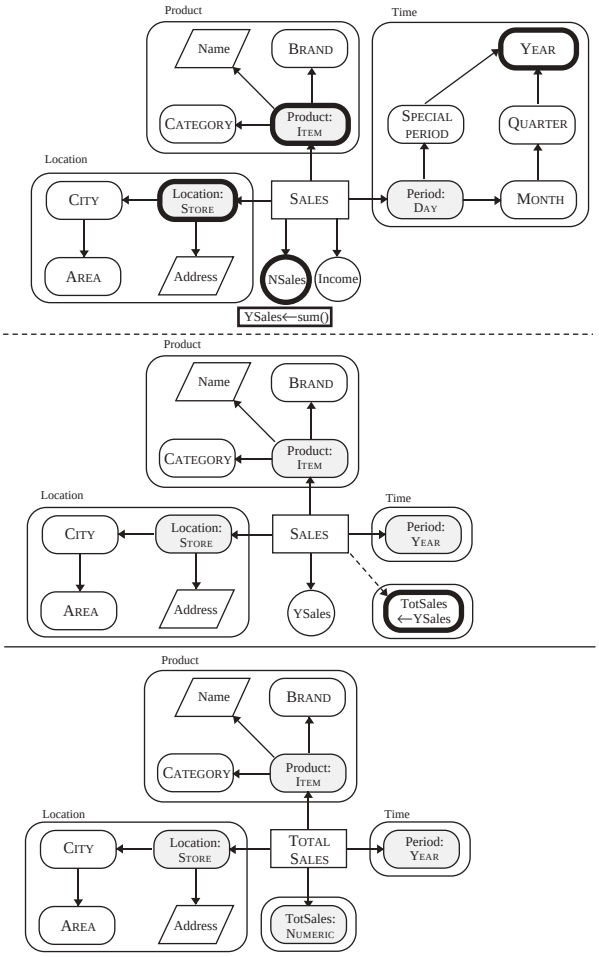


Figure 8. An abstraction query

4. The Algebraic Query Language

In this section we present more formally the $\mathcal{MD}$ algebra. Similarly to what happens with the relational algebra, the operators of the $\mathcal{MD}$ algebra are closed, that is, they apply to f-tables and produce an f-table as result. In this way, the various operators can be composed to form the *f-expressions* of the language.

It is worth noting that, to obtain closed operators, each f-expression must satisfy a *functional dependency* from its attributes to its measures. To guarantee this property, suitable syntactical conditions need to be introduced over some operator.

F-expressions are defined recursively as follows.

F-tables. If F is an f-table having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$, then F is an f-expression over the same scheme as F, that is, an f-expression over the attributes $A_1, \ldots, A_n$ and the measures $M_1, \ldots, M_m$.

The result of F is, trivially, F itself.

Cartesian product. If E, E' are f-expressions having schemes $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$ and $[A'_1, \ldots, A'_{n'}] \rightarrow [M'_1, \ldots, M'_{m'}]$, respectively, over disjoint attributes and measures, then $E \times E'$ is an f-expression with schema over the attributes $A_1, \ldots, A_n, A'_1, \ldots, A'_{n'}$ and measures $M_1, \ldots, M_m, M'_1, \ldots, M'_{m'}$.

As usual, the result contains an entry for each pair of entries in the two f-expressions, having as measure the juxtaposition of the two corresponding measures.

Natural join. If E and E' are f-expressions over $[A_1, \ldots, A_k, A_{k+1}, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$ and $[A_1, \ldots, A_k, A'_{k+1}, \ldots, A'_{n'}] \rightarrow [M'_1, \ldots, M'_{m'}]$, respectively, that is, having $A_1, \ldots, A_k$ as common attributes (defined over the same levels) and no common measures, then $E \bowtie E'$ is an f-expression with schema over the attributes $A_1, \ldots, A_k, A_{k+1}, \ldots, A_n, A'_{k+1}, \ldots, A'_{n'}$ and measures $M_1, \ldots, M_m, M'_1, \ldots, M'_{m'}$.

The result has an entry for each pair of entries in the two f-expressions with the same values on the common attributes; the corresponding measures are the juxtaposition of the measures of the original entries.

Roll up. Let E be an f-expression having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$, and let $A_i : l$ be an attribute of E. If A' is an attribute name and l' is a level such that l rolls up to l', then $\varrho_{A_i:l_i}^{A':l'}(E)$ is an f-expression having scheme $[A_1, \ldots, A_n, A'] \rightarrow [M_1, \ldots, M_m]$.

The semantics of this f-expression is defined as follows. If γ is an entry of E, and $\gamma(A_i)$ rolls up to o' in the level l', then the result contains an entry γ' obtained by extending γ with the attribute A' in such a way that $\gamma'(A') = o'$.

Note that the result of this operation satisfies the functional dependency $A_i \rightarrow A'$.

Level description. Let E be an f-expression having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$, and let $A_i : l$ be an attribute of E. If A is an attribute name and δ is a level description of the level l, then $\delta^{A=\delta(A_i)}(E)$ is an f-expression having scheme $[A_1, \ldots, A_n, A] \rightarrow [M_1, \ldots, M_m]$.

The semantics of this f-expression is defined as follows. If γ is an entry of E, then the result contains an entry γ' obtained by extending γ with the attribute A in such a way that $\gamma'(A) = \delta(\gamma(A_i))$.

Note that the result of this operation satisfies the functional dependency $A_i \rightarrow A$.

Scalar function application. If E is an f-expression over $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$, M is a measure name, and ϕ is a scalar formula (defined next), then $\varphi^{M=\phi}(E)$ is an f-expression over the same attributes of E and over the measures $M, M_1, \ldots, M_m$.

A *scalar formula* over an f-table scheme is an expression built from attributes, measures, and constants by using scalar functions taken from a vocabulary $\mathcal{F}$ of available functions.

The result contains the same entries as E. The measures associated with an entry γ are the same as E, plus the new measure M, having value $\phi(\gamma)$.

Renaming. If E is an f-expression having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$, A_i is an attribute of E, and A is a new attribute name, then $\rho_{A_i \rightarrow A}(E)$ is an f-expression over the scheme $[A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, A] \rightarrow [M_1, \ldots, M_m]$.

The result contains an entry γ' for each entry γ in E, where γ' is obtained from γ by renaming the attribute name A_i into A.

The result satisfies the functional dependencies of E, renamed accordingly.

Selection. If E is an f-expression having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$ and ϑ is a condition (defined next), then $\sigma_\vartheta(E)$ is an expression over the same scheme as E.

A *condition* over an f-table scheme is a boolean expression of the form $t \theta t'$, where: t and t' are attributes, measures, or constants; and θ is a built-in comparison predicate such as $=, \neq, >$, etc.

The result contains an entry of E and the corresponding measures, only they satisfy the condition ϑ.

In general, the result satisfies the same functional dependencies as E. However, if the condition ϑ has the form $A_i = A_j$, where A_i and A_j are attributes, then the result satisfies also the functional dependencies $A_i \rightarrow A_j$ and $A_j \rightarrow A_i$.

Simple projection. If E is an f-expression having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$, $h \leq n$ and $k \leq m$, and E satisfies the functional dependency $A_1, \ldots, A_h \rightarrow A_i$, for each $i > h$, then $\pi_{[A_1, \ldots, A_h] \rightarrow [M_1, \ldots, M_k]}(E)$ is an f-expression over the scheme $[A_1, \ldots, A_h] \rightarrow [M_1, \ldots, M_k]$.

The result contains an entry γ' for each entry γ of E, which is the restriction of γ to the attributes $A_1, \ldots, A_h$; the measures associated with γ' are the restriction of the measures associated with γ to $M_1, \ldots, M_k$.

It is worth noting that, because of the functional dependencies satisfied by E, the result contains the same number of entries as E.

Aggregation. Let E be an f-expression having scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$. Let $k \leq n$, $N_1, \ldots, N_l$ be measure names, and $g_1, \ldots, g_l$ be aggregate function from $\mathcal{F}$. Then $\psi_{A_1, \ldots, A_k}^{N_1 = g_1(M_{i_1}), \ldots, N_l = g_l(M_{i_l})}(E)$ is an f-expression over the scheme $[A_1, \ldots, A_k] \rightarrow [N_1, \ldots, N_l]$.

The semantics is as follows. Let γ be a tuple over the attributes $A_1, \ldots, A_k$, and let T_γ be the set of entries in E whose restriction to $A_1, \ldots, A_k$ coincides with γ. If T_γ is not empty, then γ is an entry of the result; the value for

the measure N_j associated with γ is the result of applying the aggregate function g_j to the multiset of measures M_{i_j} associated with the entries in T_γ, for $1 \leq j \leq l$.

Abstraction. If E is an f-expression with scheme $[A_1, \ldots, A_n] \rightarrow [M_1, \ldots, M_m]$ and $k \leq m$, then $\alpha_{M_1,\ldots,M_k}(E)$ is an f-expression over $[A_1, \ldots, A_n, M_1, \ldots, M_k] \rightarrow [M_{k+1}, \ldots, M_m]$.

The result contains an entry γ' for each entry γ of E. The entry γ' is obtained by extending γ with the attributes $M_1, \ldots, M_k$, which take their values from the corresponding measures. The measures associated with γ' are the restriction of the measures associated with γ to $M_{k+1}, \ldots, M_m$.

The result satisfies the functional dependency $A_1, \ldots, A_n \rightarrow M_1, \ldots, M_k$.

Definition 4.1 (Algebraic Query) *An $\mathcal{MD}$* algebraic query *Q over an $\mathcal{MD}$ scheme S is an f-expression over the f-tables of S.*

5. The Graphical Query Language

We now present the syntax and the semantics of the graphical query language for the $\mathcal{MD}$ model. We first show how it is possible to associate a graph with an f-table, which we call its f-graph.

Definition 5.1 (F-graph) *Let $F[A_1 : l_1, \ldots, A_n : l_n] \rightarrow [M_1 : l'_1, \ldots, M_m : l'_m]$ be an f-table scheme over a set D of dimensions. The* f-graph *of F is an directed acyclic graph g_F, such that:*

- *there is just one node in g_F with no incoming arc, called the* f-node *of g_F; all the other nodes are associated with some level of D: if a node n is associated with a level l we say that n* is over *l;*
- *for each attribute A_i of F, $1 \leq i \leq n$, there is a node n_i in g_F, named A_i and over l_i, and there is an arc from the f-node to n_i — these nodes are called the* a-nodes *of g_F;*
- *for each measure M_j of F, $1 \leq j \leq m$, there is a node n'_j in g_F, named M_j and over l'_j, and there is an arc from the f-node to n'_j — these nodes are called the* m-nodes *of g_F;*
- *for each a-node n over a level l in a dimension $d \in D$ there is also a node in g_F for each level $l' \neq l$ of d such that $l \preceq l'$ — these nodes are called* r-nodes *of g_F and form a* g-dimension over *d;*
- *for each node n over a level l having a level description $\delta(l) : l'$, there is a node n' in g_F over l' and an arc from n to n' — these nodes are called* d-nodes *of g_F;*
- *within a g-dimension over d there is an arc in g_F from a node n_1 over a level l_1 to a node n_2 over a level $l_2 \succeq l_1$ if and only if l_1 immediately precedes l_2 in the partial order defined over d.*

In an f-graph, we also call l-nodes *all the nodes different from the f-node.*

Example 5.2 The f-node of the f-graph in Figure 3 is the one named SALES (we have used the convention to name the f-node with the name of the corresponding f-table). This f-graph has three g-dimensions (over **Product**, **Time**, and **Location**), three a-nodes (named *Location*, *Product*, and *Period* over Store, Item, and Day, respectively), two m-nodes (named *NSales* and *Income*), and two d-nodes (named *Address* and *Name*). The others nodes are r-nodes. ■

We now introduce the notion of graphical query specification (s-graph for short) that is built over a set of f-graphs, as follows.

Definition 5.3 (S-graph) *Let S be a non-empty set of f-graphs. An* s-graph *G_S over S is a labeled graph, such that:*

- *G_S has all the nodes and arcs of the f-graphs in S, that is, the f-graphs in S are disjoint subgraphs of G_S;*
- *G_S can have new m-nodes: for each of them there is a new arc from an f-node n to it, labeled with an expression of the form $M \leftarrow \phi$, where M is a name and ϕ is a scalar formula over the names of the m-nodes of n;*
- *G_S can have new arcs between pairs of l-nodes over the same level, each of which is labeled with a built-in comparison predicate;*
- *an l-node in G_S over a level l can be labeled with an expression of the form θv, where θ is a comparison predicate and v is a constant in the domain of l;*
- *an l-node in G_S can be labeled with an expression of the form $A \leftarrow$, where A is a new name;*
- *an m-node in G_S can be labeled with an expression of the form $M \leftarrow g()$, where M is a new name and g is an aggregate function;*
- *G_S can have new a-nodes: for each of them there is a new arc from an f-node n to it, a new g-dimension for it, and it is labeled with an expression of the form $A \leftarrow B$, where A is a new name and B is a name of an m-node of n;*
- *an l-node in G_S can be* marked, *with the limitation that there is at most one marked l-node for each g-dimension.*

An s-graph specifies a restructuring operation over f-graphs. This operation is captured by two mappings: Φ and Θ. The former is a mapping from s-graphs to f-graphs and specifies the structure of the result of the restructuring. The latter is a mapping between f-table instances that specifies the underlying data transformation. The two mappings are defined below; because of space limitation, some technical details are omitted.

Let G_S be an s-graph over a set S of f-graphs: $\Phi(G_S)$ is an f-graph defined as follows.

- $\Phi(G_S)$ contains one f-node;
- there is one m-node in $\Phi(G_S)$ for each marked m-node of G_S;
- there is one a-node in $\Phi(G_S)$ for each marked a-nodes of G_S;
- the r-nodes, d-nodes, and g-dimensions in $\Phi(G_S)$ are defined from m-nodes and a-nodes as in Definition 5.1;
- the m-nodes and a-nodes in $\Phi(G_S)$ have the same names and levels of the corresponding nodes of G_S, except when they are labeled: in this case, a node labeled $A \leftarrow \gamma$ is named A in $\Phi(G_S)$ and is over a level that depends on γ.

Now, let G_S be an s-graph over a set S of f-graphs: the mapping $\Theta(G_S)$ takes as input a set of instances over the schemes described by S and returns an instance over the scheme described by $\Phi(G_S)$. This mapping is defined using the $\mathcal{MD}$ algebra: intuitively, each element of G_S corresponds to the application of an algebraic operator, as follows.

- The presence of several f-graphs in G_S describes a cartesian product over the corresponding f-tables;
- the presence of m-nodes in G_S not occurring in S describes a scalar function application over the corresponding measures, as specified by their labels;
- the presence of arcs in G_S not occurring in S and l-nodes labeled by comparison predicates describes a selection, possibly followed by a level description and a roll-up;
- the presence of l-nodes in G_S labeled by expressions of the form $A \leftarrow$ describes a renaming;
- the presence of l-nodes in m-nodes in G_S labeled by expressions of the form $M \leftarrow g()$ describes a roll-up followed by an aggregation (marked nodes specify the desired level of aggregation);
- the presence of a-nodes in G_S not occurring in S describes an abstraction;
- finally, the presence of marked l-nodes in G_S may describe a projection over the specified attributes and measures.

We are now ready to define the notion of query.

Definition 5.4 (Graphical Query) *An $\mathcal{MD}$ graphical query Q is a sequence of s-graphs $\mathcal{G}_Q = \langle G_s, G_1, \ldots, G_k, G_t \rangle$, where $k > 0$. The s-graph G_s is called the* source *of $\mathcal{G}_Q$ and is composed by a finite set of f-graphs. The s-graph G_t is called the* target *of $\mathcal{G}_Q$ and is composed by one f-graph only. The sequence $G_1, \ldots, G_k$ is called* query specification*. Each s-graph G_i in the query specification is over the f-graph defined by the previous s-graph G_{i-1}. The result of the query is defined as the composition of the mappings $\Theta(G_i)$.*

Example 5.5 Consider the graphical query in Figure 5. The source of this query is composed by the f-graphs corresponding to the f-tables SALES (shown in Figure 3) and COSTOFITEM. The query specification is composed by two s-graphs. The mapping defined by the first one corresponds to the algebraic expression E_1:

$$\pi_{[S.Period, S.Product, Location] \rightarrow [Cost]} (\sigma_{S.Product = C.Product, C.Period = Month} (\text{COST} \times \varrho^{Month:\text{month}}_{Period:\text{day}} (\text{SALES}))).$$

The second s-graph defines the mapping E_2:

$$\pi_{[Period, Product, Location] \rightarrow [Gain]} (\varphi^{Gain = Income - (NSales * Cost)} (E_1))$$

The composition of these two expressions define the result of the query, whose scheme is described by the target of the graphical query shown on the bottom of Figure 5. ■

The given definitions suggest that the each graphical query can be expressed by an $\mathcal{MD}$ algebraic expression. Actually, we can also show that the converse holds. We then have the following result.

Theorem 5.6 *The $\mathcal{MD}$ algebraic and graphical languages have the same expressive power.*

6. Related Work

The term OLAP has been recently introduced 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 [5]. A comparison between OLAP concepts and the area of statistical databases is given in [8].

The multidimensional data model illustrated in this paper has been proposed in [2, 3]. In the previous papers we have proposed a design methodology for multidimensional databases [3], and we have presented a declarative query language and studied its expressiveness [2]. The present paper is devoted to the investigation of two further paradigms for querying OLAP databases.

Many authors [1, 4, 7] claim that SQL is unsuited to data-analysis applications, since some aggregate and grouping queries are difficult to express and optimize. They thus consider the problem of extending SQL with specific aggregation and analysis-oriented operators. Gray et al. [7] proposed `cube` as an operator generalizing `group by`. Chatziantoniou and Ross [4] extend both SQL and the relational algebra with an operator, which deals with "aggregation variables", to succinctly express common queries. Agrawal et al. [1] have proposed a framework for studying multidimensional databases, consisting of a data model based on the notion of multidimensional cube, and a powerful algebraic query language. Many of the features considered in such proposals can be expressed in our languages using a suitable collection of scalar and aggregate functions.

Gebhardt et al. [6] propose a tape-based model of multidimensional data, together with an operational framework for visualization and querying that makes use of operations on tapes (e.g., scrolling and intersection) as graphical metaphors. This framework yields a way to build worksheets from operational sources and define their visualization. Thus, it can be viewed as a graphical language having visual and (simple) restructuring capabilities (essentially roll-up and group-by operations). Our approach is instead focused on data restructuring and therefore the languages we propose are more powerful in expressing multidimensional queries. Moreover, our graphical language is based on different metaphors (graphs and trees) that, we believe, are easier to understand for non-expert users.

The aspect of manipulating multidimensional databases is covered by commercial OLAP systems essentially in a pragmatic way, providing the users with powerful visualization tools, and with query tools having various levels of capabilities. Notable examples are MetaCube [9] and Oracle Express [10]. Both of them provide graphical interactive interfaces in which, again, visual and restructuring capabilities are mixed. The queries that can be expressed through these interfaces apply to single fact tables and are generally quite simple: roll-up, drill-down, slice and dice, pivoting. All of them are based on intuitive concepts and rely on informal definitions. Our approach is different: we have first defined an algebra that allows us to define multidimensional operations in a precise way, and we have then developed a graphical language that is based on the algebra and so have a clear semantics. The resulting language is in general more expressive since allows the user to express complex queries possibly involving several f-tables. Moreover, the metaphors we have used are more abstract and so more natural for naive users than the ones used in the above systems, which are mainly based on nested folders.

7. Conclusion

In this paper we have investigated a framework for the manipulation of multidimensional databases. Our proposal relies on $\mathcal{MD}$, a model for multidimensional data, which provides a clear separation between practical and conceptual aspects of OLAP systems. We have studied, in this context, two query languages, over different paradigms, that can be effectively used for querying multidimensional databases, demonstrating that $\mathcal{MD}$ is well-suited for this purpose, since it allows the user to disregard implementation aspects. The first language is based on an algebra and provides an effective way to manipulate multidimensional data in a procedural fashion. The second query language has a graphical nature, and it is suitable for end-users since it provides a simple tool for specifying queries over an OLAP database. We have also shown that the two languages have indeed the same expressive power.

References

[1] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *13th IEEE Int. Conf. on Data Engineering*, pag. 232–243, 1997.

[2] L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Sixth Int. Workshop on Database Programming Languages (DBPL'97), Springer-Verlag*, 1997.

[3] L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. In *6th Int. Conf. on Extending Database Technology*, pag. 183–197, 1998.

[4] D. Chatziantoniou and K. Ross. Querying multiple features of groups in relational databases. In *22th Int. Conf. on Very Large Data Bases*, pag. 295–306, 1996.

[5] S. Chaudhuri and U. Dayal. An overview of Data Warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[6] M. Gebhardt, M. Jarke, and S. Jacobs. A toolkit for negotiation support interfaces to multi-dimensional data. In *ACM SIGMOD Int. Conf. on Manag. of Data*, pag. 348–356, 1997.

[7] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *20th IEEE Int. Conf. on Data Engineering*, pag. 152–159, 1996.

[8] A. Shoshani. OLAP and statistical databases: Similarities and differences. In *16th ACM Symp. on Principles of Database Systems*, pag. 185–196, 1997.

[9] MetaCube Explorer User Guide. Informix Software Inc., *http://www.informix.com*.

[10] Oracle OLAP products: adding value to a data warehouse. Oracle White Paper, *http://www.oracle.com*.