Cube Algebra: A Generic User-Centric Model and Query Language for OLAP Cubes

Ricardo Ciferri, Cristina Ciferri
Universidade de São Paulo em São Carlos, Brazil
Leticia Gómez
Instituto Tecnológico de Buenos Aires, Argentina
Markus Schneider
University of Florida, USA
Alejandro Vaisman, Esteban Zimányi
Université Libre de Bruxelles, Belgium

ABSTRACT

The lack of an appropriate conceptual model for data warehouses and OLAP systems has led to the tendency to deploy logical models (for example, star, snowflake, and constellation schemas) for them as conceptual models. ER model extensions, UML extensions, special graphical user interfaces, and dashboards have been proposed as conceptual approaches. However, they introduce their own problems, are somehow complex and difficult to understand, and are not always user-friendly. They also require a high learning curve, and most of them address only structural design, not treating operations over the cube metaphor. Therefore, they are not really an improvement and, in the end, only represent a reflection of the logical model. The essential drawback of offering this system-centric view as a user concept is that knowledge workers are confronted with the full and overwhelming complexity of these systems as well as complicated and user-unfriendly query languages such as SQL OLAP and MDX at the logical level. In this article, we propose a user-centric conceptual model for data warehouses and OLAP systems, called the Cube Algebra. It takes the cube metaphor literally and provides the knowledge worker with high-level cube objects and related concepts. A novel query language leverages well known high-level operations such as roll-up, drill-down, slice, and drill-across. As a result, the logical and physical levels are hidden from the unskilled end user.

Keywords: data warehouses, OLAP, cube, conceptual model, user-centric model, query language

INTRODUCTION

For more than a decade, data warehousing has been at the forefront of information technology applications as a way for organizations to effectively use and analyze information for business planning and decision making. Data warehouses contain large repositories of analytical and subject-oriented data integrated from several heterogeneous sources over a large period of time in an organization. The technique of performing complex analysis over the information stored in the data warehouse is commonly called Online Analytical Processing (OLAP). A review of the evolution of data warehouse technology reveals that research and development has mainly focused on system aspects such as the construction of data warehouses, materialization, indexing, and the implementation of OLAP functionality. This *system-centric* view has led to well-established and commercialized technologies such as relational OLAP

(ROLAP), multidimensional OLAP (MOLAP), and hybrid OLAP (HOLAP) at the logical and the physical levels. However, the unskilled user such as the manager in a consulting company or the analyst in a financial institution is confronted with the problem that the handling of data warehouses and OLAP systems requires expert knowledge due to complicated data warehouse structures and the complexity of OLAP systems and query languages. Two main reasons are responsible for this problem. First, due to the lack of a generic, user-friendly, and comprehensible conceptual data warehousing model, data warehouse design is usually directly performed at the logical level and leads to the exposure of the logical design schemas that are difficult to understand by the unskilled user. In a ROLAP environment, for example, the user is faced with the logical design of relational tables in terms of star, snowflake, or fact constellation schemas. The proposal to alleviate the problem by providing extensions to the Entity-Relationship Model and the Unified Modeling Language, or by offering specific graphical user interfaces or dashboards for data warehouse design is not really convincing since ultimately they represent a reflection and visualization of relational technology concepts and, in addition, reveal their own problems. Second, available OLAP query and analysis languages such as MDX and SQL OLAP operate at the logical level and require the user's deep understanding of the data warehouse structure in order to be able to formulate queries. These languages are quite complex, overwhelm the unskilled user, and are therefore inappropriate as end-user languages.

We conclude that a *generic*, *conceptual*, and *user-centric* data warehouse model that focuses on user requirements is missing and needed. Such a model should fulfill several design criteria. First, it should be located above the logical level. Second, it should abstract from and be independent of the models and technologies (ROLAP, MOLAP, HOLAP) at the logical level. Third, it should be able to cooperate with any of these logical models and technologies. Fourth, it should enable the user to *generically* and *abstractly* represent and query hierarchical multidimensional data. Fifth, it should provide the user with a query and analysis language that is exclusively based on the conceptual level, thus abstracts from any logical model, and provides high-level query operations for the user. The goal of this article is to propose and formally describe a conceptual and user-centric data warehouse model and query language that satisfies these design criteria. Surprisingly, the conceptual view this model adopts is not new; on the contrary, it is well-known. However, the way and resoluteness in which we offer this concept is novel. Our proposed conceptual model leverages the *cube* view of data warehouses but takes the cube metaphor *literally*. That means that the user's conceptual world is solely the cube that the user can create, manipulate, update, and query. The cube is used as the user concept that completely abstracts from any logical and physical implementation details. Technically, this implies that cubes can be regarded as an *abstract data type* that provides cubes as the only kind of values (objects), offers high-level operations on cubes or between cubes such as *slice*, *dice*, *drill-down*, *roll-up*, and *drill-across* as the only available access methods, and hides any data representation and algorithmic details from the user, who can concentrate on her main interest, namely to analyze large volumes of data. Another characterization is to say that we define a *universal algebra* with cubes as the only sort and a collection of unary and binary operations on cubes. We therefore name our approach *Cube Algebra*. We will show that this algebra develops its full power and

expressiveness if it is used as a high-level query language.

The paper is organized as follows. Next section discusses related work and compares available data warehousing models with our Cube Algebra. Then, we describe an application scenario that we use throughout the paper to illustrate important aspects of the Cube Algebra. In the same section, we provide a three-level architecture of a data warehouse and OLAP system that includes our Cube Algebra. We further specify the formal data model supporting the Cube Algebra. The section concludes with a sketch of a data definition language to specify the structure of a cube. Then, we define high-level OLAP cube operations such as *slice* and *drill-across*, and illustrate their use in a number of queries that refer to our application scenario. Finally, the last section draws some conclusions and sketches future work.

RELATED WORK

There are several data warehouse (DW) models in the literature, which focus on the logical and conceptual levels (see for example the survey in (Marcel, 1999)). Most of these models address the logical level, e.g., (Li & Wang, 1996; Cabibbo & Torlone, 1997; Cabibbo & Torlone, 1998, Lehner, 1998). Therefore, they are dependent of specific technologies, for example ROLAP, MOLAP and HOLAP, which lead to complex and non-user friendly query languages. In this section, we limit ourselves to address conceptual models, and discuss them with respect to our proposal, i.e., we focus on the user support provided by these models. We claim that most of the models aimed at addressing the conceptual level, actually rely on structures that are indeed close to the logical level, thus not addressing end-user needs. We call these models system-centric, opposite to the user-centric approach we present in this paper. Taking this into account, we next comment on this work, and present an analysis against our proposal.

We first classify existing models into four classes: (a) conceptual models based *on extensions to the Entity-Relationship* (ER) *Model* (Chen, 1976); (b) conceptual models based *on extensions to the Unified Modeling Language* (UML); (c) models based on a view of data as a cube.

We start with a discussion on models in class (a) (*ER-based* models). Rizzi (2007) proposed the Dimensional Fact Model (DFM), which uses the typical DW concepts of facts, dimensions, measures, hierarchies, descriptive and cross-dimension attributes; the model also supports shared, incomplete, recursive, and dynamic hierarchies, and notions such as additivity. To represent these concepts, DFM relies on a graphical notation that facilitates the understanding of the conceptual schema, and that is an abstraction of the star schema, in which there is a central fact entity and a graph per dimension to represent the attribute hierarchies. DFM can be applied to two approaches of the DW design: data-driven, which designs the DW starting from the analysis of the data sources, and requirements-driven, which starts from determining requirements of end users. Golfarelli and Rizzi (1998) extend DFM by presenting a methodological framework for DW conceptual modeling, which starts gathering user requirements and carries out the data warehouse design semi-automatically from the operational database schema. In addition to providing an abstraction of the star schema in terms of a central fact entity and several graphs, they also formalize each concept of the DFM and define a language to denote queries according to the DFM to validate the generated schema. Tryfona, Busborg & Christiansen (1999) propose the starER model, which combines the star structure with the constructs of the ER Model, in

addition to proposing special types of relationships to support attribute hierarchies on dimensions. The starER model encompasses the following main constructors: facts, entities, relationships among entities, and attributes, the latter represent properties of entities or relationships, or facts. Further, the starER model provides a graphical notation very close to the ER Model. Along similar lines (i.e., starting from an ER-based data model), Malinowski and Zimányi (2008) introduce a metamodel of hierarchy classification that encompasses from symmetric simple hierarchies until more complex ones, such as non-strict simple hierarchies, asymmetric and generalized simple hierarchies, multiple hierarchies, and parallel hierarchies. They also present a graphical notation for representing these hierarchies, close to a relational representation.

We now move on to discuss models in class (b) *UML-based* models. Abelló, Samos & Saltor (2001) investigate relationships between cubes in an object-oriented framework with navigation operations. Here, the data cube is defined in terms of a set of concepts, such as measures and cells, dimensions and aggregation levels, and facts. An algebra is defined as a set of multidimensional operations, such as base changes, dice, slice, drill-across, roll-up and drill-down. In sequels of this paper (Abelló, Samos & Saltor 2002; Abelló, Samos & Saltor 2006), the authors propose YAM^2, a multidimensional conceptual object-oriented model for data warehousing and OLAP tools extending the UML, which is defined in terms of its structures, integrity constraints and query operations. However, in spite of being defined at the conceptual level, YAM^2 relies on star schema-like design, and therefore is not completely independent of logical modeling concepts. Pardillo et al. (Pardillo, Mazón & Trujillo 2008) introduce platform-independent conceptual OLAP queries that can be automatically traced to their logical implementation, together with an OLAP algebra at the conceptual level by using the Object Constraint Language (OCL), aimed at allowing end-users to query data warehouses without being aware of logical details. The authors introduce cube manipulation operators (e.g., dimension addition and removal), and operators such as slice, dice, drill-across, multidimensional projection, roll-up and drill-down. Cabot et al. (Cabot, Mazón, Pardillo & Trujillo 2009) extend the former work through a conceptual specification of statistical functions using OCL. Finally, (Pardillo, Mazón & Trujillo 2010) further extends OCL for OLAP querying, introducing a code-generation architecture aligned with the model-driven architecture (MDA), to map an extension to the OCL as a set of predefined OLAP operators. The main drawback of this work is that they are based on the OCL, which is not a user-friendly way for manipulating the data cube.

Let us now comment on models based on the data cube (i.e., the ones in class (c)). Tsois et al. (Tsois, Karayannidis & Sellis 2001) propose a multidimensional aggregation cube (MAC) using the concepts of dimension, dimension level, dimension member, drilling relationship and dimension path. Although they address the DW modeling problem from the end-user point of view, and describe a set of requirements for the conceptual modeling of real-world OLAP scenarios, the authors do not present a language supporting the model. Along different lines, some authors formally define the notion of a cube and introduce operations for this. Agrawal et al. (Agrawal, Gupta & Sarawagi 1997) propose a data model whose core features are the symmetric treatment of dimensions and measures, the support of multiple hierarchies along each dimension and the possibility of performing ad-hoc aggregates. They also define a minimal set of algebraic operators that is composed of the following

operators: push and pull (to allow symmetric treatment of dimensions and measures), destroy dimension, restriction (slice and dice), join and associate. This minimal set of operators stays as close to relational algebra as possible and can be translated to SQL through an algebraic application programming interface. The conceptual multidimensional database model proposed by Gyssens and Lakshmanan (1997) focuses on the separation between structural aspects and the contents, allowing the definition of a data manipulation language that can express the cube operator. In this sense, they define an algebra (and an equivalent calculus), which is composed of the following main operators: unary operators (selection, projection and renaming), set operators (union, intersection and difference), Cartesian product, fold and unfold, classification on relations and on tables, and summarization on relations and on tables. Vassiliadis (1998) formally defines the concepts of dimensions, dimension hierarchies, and cube to propose a model for multidimensional databases. He also introduces a set of cube operators based on the notion of the base cube, which is used for providing the calculation of the results of cube operations. These operations are: level climbing, packing, function application, projection, navigation, slicing, and dicing. Also, he provides a mapping of the proposed multidimensional model to the relational model and to multidimensional arrays. The proposals in this class, although based on the concept of a cube, do not approach the problem from a conceptual modeling point of view, neither they consider user's need (i.e., the user-centric view). The proposed operators are not commonly used by high-level users such as managers is not completely covered.

Related Work	Focus	Cube Metaphor	Cube as ADT	Model Level	Model Extension	Algebra or Calculus
Abelló et al. (2001)	system-centric	✓	×	conceptual	object-oriented	✓
Abelló et al. (2002) Abelló et al. (2006)	system-centric	×	×	higher than logical, lower than conceptual	UML-based	✓
Agrawal et al. (1997)	system-centric	✓	×	higher than logical, lower than conceptual	not an extension	✓
Cabibbo and Torlone (1997)	system-centric	×	×	logical	not an extension	✓
Cabibbo and Torlone (1998)	system-centric	×	×	logical	not an extension	✓
Cabot et al. (2009)	system-centric	×	×	logical	UML-based	×
Ruiz (2009)	system-centric	×	×	logical	not an extension	×
Golfarelli and Rizzi (1998)	system-centric	×	×	logical	not an extension	×
Gyssens and Laksshmanan (1997)	system-centric	✓	×	logical	not an extension	✓
Lehner (1998)	system-centric	×	×	logical	not an extension	✓
Li and Wang (1996)	system-centric	✓	×	logical	not an extension	✓
Malinowski and Zimányi (2008)	system-centric	×	×	logical	not an extension	×
Pardillo et al. (2008)	system-centric	✓	×	higher than logical, lower than conceptual	UML-based	✓
Pardillo et al. (2010)	system-centric	✓	×	higher than logical, lower than conceptual	UML-based	✓
Rizzi (2007)	system-centric	✓	×	logical	not an extension	×

Tryfona et al. (1999)	system-centric	×	×	conceptual	ER-based	×
Tsois et al. (2001)	user-centric	✓	×	conceptual	not an extension	×
Vassiliadis (1998)	system-centric	✓	×	higher than logical, lower than conceptual	not an extension	✓
Cube Algebra	user-centric	✓	✓	conceptual	not an extension	✓

Table 1: Core properties of data warehouse models.

Table 1 refers to the following core characteristics of each studied DW model: (i) its focus (system-centric vs. user-centric); (ii) if the model supports the data cube metaphor; (iii) if the model provides a cube as an abstract data type; (iv) the level at which the model is defined (conceptual level, logical level or somewhere in-between them); (v) the model that it is based on; and (vi) if it includes an OLAP algebra or calculus. We can see that almost all proposals defined at the conceptual level are actually system-centric rather than user-centric. On the contrary, the Cube Algebra proposal applies to the conceptual level of a data warehouse architecture, independently from implementation issues. The only user-centric model is the proposal of Tsois et al. (Tsois, Karayannidis & Sellis, 2001). However, the model does not provide a cube as an abstract data type and, more important, it does not propose an OLAP algebra or OLAP calculus to support it.

Related Work	Operations defined *only* over the cube metaphor	Complexity of the query language	Kind of query language	Graphic tools or dashboards
Abelló et al. (2001)	×	complex	conceptual	×
Abelló et al. (2002)	×	complex	relational algebra	×
Abelló et al. (2006)				
Agrawal et al. (1997)	✓	complex	relational algebra	×
Cabibbo and Torlone (1997)	×	complex	calculus-based	×
Cabibbo and Torlone (1998)	×	complex	relational algebra	✓
Cabot et al. (2009)	×	complex	OCL-based	×
Ruiz (2009)	×	complex	MDX-based	×
Golfarelli and Rizzi (1998)	×	not available	not available	✓
Gyssens and Laksshmanan (1997)	×	complex	relational algebra/ calculus	×
Lehner (1998)	×	not available	relational algebra	×
Li and Wang (1996)	×	complex	relational algebra	×
Malinowski and Zimányi (2008)	×	not available	not available	✓
Pardillo et al. (2008)	×	complex	OCL-based	×
Pardillo et al. (2010)	✓	complex	relational algebra plus OCL	✓
Rizzi (2007)	×	not available	not available	✓
Tryfona et al. (1999)	×	not available	not available	✓
Tsois et al. (2001)	×	not available	not available	✓

Vassiliadis (1998)	×	complex	relational calculus	×
Cube Algebra	✓	user-friendly	relational algebra	×

Table 2: Query language and operations of data warehouse models.

Table 2 compares existing proposals against Cube Algebra, with respect to query language functionalities, namely: (i) if the operations are defined over the cube or over other data objects, at lower levels of abstraction (for example, tables, star schemas); (ii) the complexity of the query language for unskilled end-users; (iii) the kind of query language, such as cube-based, relational algebra-based, relational calculus-based, OCL-based and MDX-based, or no language is provided at all; and (iv) if the work provides some graphical notation or dashboard to aid the data warehouse modeling. As we can see in Table 2, no work (except the Cube Algebra) offers a user-friendly query language. Abelló et al. (Abelló, Samos & Saltor 2001) is the only exception, introducing a query language at the conceptual level. Nevertheless, the query language is complex for unskilled end users. Second, the operations defined over the cube, such as base changes, generalization, specialization, and derivation, are far from the knowledge of managers and analysts in an OLAP scenario.

Finally, Table 3 details and compares the set of operations that each proposal provides, that is, general functionalities to create, manipulate, and update the cube metaphor, and the set of high-level operations for querying the data cube (roll-up, drill-down, slice, dice, drill-across, pivot). As we can see, only the Cube Algebra encompasses all the operations. We can also see that most of the proposals do not offer operations to create, manipulate, and update the cube.

Related Work	General Functionalities			Query Cube				
	create cube	manipulate cube	update cube	roll-up (drill-down)	slice	dice	drill-across	pivot
Abelló et al. (2001)	×	✓	×	✓	✓	✓	✓	×
Abelló et al. (2002)	×	×	×	roll-up	×	✓	✓	×
Abelló et al. (2006)								
Agrawal et al. (1997)	×	×	×	✓	✓	✓	×	×
Cabibbo and Torlone (1997)	×	×	×	×	×	×	×	×
Cabibbo and Torlone (1998)	×	×	×	roll-up	×	×	×	×
Cabot et al. (2009)	×	×	×	×	×	×	×	×
Ruiz (2009)	×	×	×	×	×	×	×	×
Golfarelli and Rizzi (1998)	×	×	×	×	×	×	×	×
Lehner (1998)	×	×	×	✓	✓	×	×	×
Li and Wang (1996)	✓	✓	×	×	×	×	×	×
Malinowski and Zimányi (2005)	×	×	×	×	×	×	×	×
Pardillo et al. (2008)	×	✓	×	✓	✓	✓	✓	×
Pardillo et al. (2010)	×	✓	×	✓	✓	✓	✓	×
Rizzi (2007)	×	×	×	×	×	×	×	×
Tryfona et al (1999)	×	×	×	×	×	×	×	×
Tsois et al. (2001)	×	×	×	×	×	×	×	×

Vassiliadis (1998)	×	×	×	drill-down	✓	✓	×	×
Cube Algebra	✓	✓	×	✓	✓	✓	✓	✓

Table 3: Types of operations of data warehouse manipulation languages.

CUBE DATA MODEL AND THREE-LEVEL DATA ARCHITECTURE

In this section, we present our user-centric cube data model and how it fits into the landscape of data warehouses. By selecting the application scenario of pollution control, next section informally introduces the main cube concepts that a user should be able to understand. Then, we present a three-level architecture of a data warehouse and OLAP system that integrates our Cube Algebra, and formally define the underlying user-centric data model supporting such algebra. Finally, we sketch a high-level data definition language for data cubes.

Application Scenario: Cubes for Pollution Control

We illustrate the needed user concepts by leveraging an application scenario of pollution control in Belgium. Monitoring stations located at different locations enable the measurement of certain pollutants (such as carbon monoxide, CO). Thus, three aspects or perspectives play a role here for the knowledge worker: (i) the monitoring station where a measurement is captured, (ii) the time when a measurement is taken, and (iii) the kind of pollution that is measured. These aspects form the three different dimensions of a cube that are shown on the left hand side of Figure 1. In this example, we call the dimensions `Station`, `Time`, and `Pollution`. Dimensions are an essential and distinguishing first-class concept in cubes. They are visually or geometrically represented by the lateral faces of the (hyper) cube. A dimension is organized as a containment-like hierarchy. Each hierarchy level represents a different (aggregation) level of detail, as it is later required by the desired analyses. Figure 1 shows the three hierarchical structures of the `Station`, `Time`, and `Pollution` dimensions. Each hierarchical structure is called a *dimension schema*. For example, the dimension schema for `Pollution` models two hierarchies with `Pollutant` as their common lowest level, namely the hierarchy consisting of the levels `Pollutant`, `Type`, and `Group`, as well as the hierarchy consisting of the levels `Pollutant` and `Category`. The levels above `Pollutant` allow grouping and are therefore interesting for the knowledge worker. For example, similar pollutants can be grouped under the level `Category`. Note that there is a unique top level in the dimension schema, denoted `All`, to which all levels aggregate. Each dimension level includes a finite set of values called *members*. A *dimension instance* comprises all members at all levels of a *dimension hierarchy*. Figure 3 gives an example of a dimension instance with respect to the dimension schema for `Pollution` shown in Figure 1. The level `Type`, for example, contains the three members: `T1`, `T2`, and `T3`.

A combination of members taken from each dimension uniquely defines a cell of a cube and implicitly specifies a *fact* if the cell is not empty. For example, in Figure 1, at the station `S2` at the quarter `Q3`, the pollutant `P4` was measured since the cell is not empty. A value in a cell (such as 12 in our case) is called a *measure value*. A *measure* represents a numerical property of a fact. In our example, the measure is named concentration. A cube can have several different measures. Each measure

comes with an *aggregation function* that can combine several measure values into one.

A *base cube schema* is a cube with the highest level granularity for all dimensions. That is, it consists of the bottom levels of all dimension schemas and a finite set of measures. A *base cube instance* is a data cube that is constructed according to a base cube schema and that maps each combination of member values to one or more measure values. A *cube view schema*/*cube view instance* is any cube schema/cube instance unequal to a base cube schema/base cube instance but can be derived from it through a collection of high-level operations we describe later.

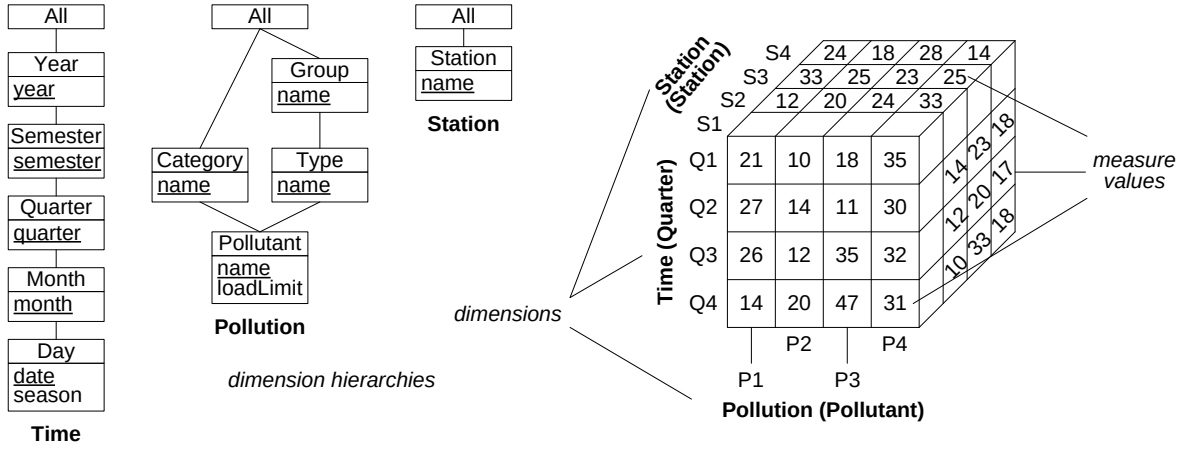


Figure 1: A three-dimensional cube for `AirQuality` data having dimensions `Time`, `Pollution`, and `Station`, and a measure `concentration`.

Data Warehouse Architecture

Figure 2 shows the three-level data warehouse architecture we devise. At the conceptual level (i.e., the highest abstraction level) of this architecture, there is the cube model described above (independent of how this cube is actually implemented), and the associated cube algebra we introduce later. At the logical level we have the implementation-dependent representation of the data cube. At this level we place the well-known star, snowflake, and constellation schemas (i.e., a ROLAP representation), as well as multidimensional (MOLAP), and hybrid representations (HOLAP). Query languages for these representations are relational query languages such as SQL dialects, and MDX. Finally, at the physical level, we find the different ways of efficiently implementing the data warehouse. For ROLAP implementations, for example, multidimensional indices such as variations of R-Trees can be used, as well as bitmap indices. For MOLAP implementations, efficient and proprietary algorithms for implementing sparse matrices are often used.

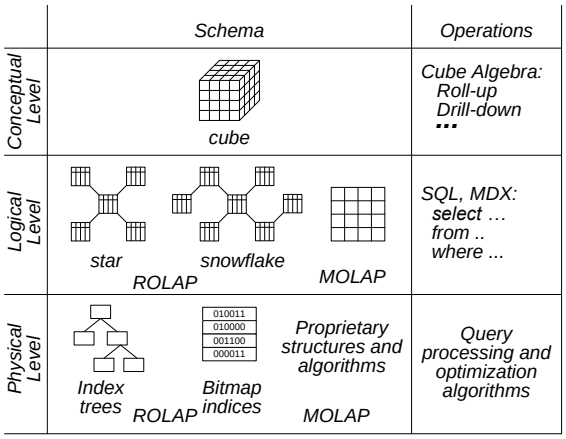


Figure 2: The Three-Level Architecture for Multidimensional Databases.

Formal Cube Data Model

Our cube-based, user-centric conceptual model is supported by a formal data model which we introduce next. In what follows, for simplicity and without lost of generality,

we assume that dimension level names are unique[i].

Definition 1 (Dimension Schema)
A *dimension schema* is a tuple $\langle nameDS, \mathcal{L}, \rightarrow \rangle$ where: (a) *nameDS* is the name of the dimension; (b) $\mathcal{L}$ is a non-empty finite set of pairs of the form $\langle l, LD \rangle$ such that l is a level (there is a distinguished level denoted *All*, such that $\langle All, \varnothing \rangle \in \mathcal{L}$), and *LD* is a set of level descriptors (attributes describing a level). Each level has a domain Dom_l, and each level descriptor has a domain Dom_{ld}; (c) $\rightarrow$ is a partial order on the levels $l \in \mathcal{L}$. This partial order defines a graph, whose nodes are the levels $l \in \mathcal{L}$, and are annotated by descriptors in *LD*; (e) The reflexive and transitive closure of $\rightarrow$, denoted $\rightarrow^*$, has a unique bottom level l_b, and a unique top level. The top level is the distinguished level *All*. We denote by $l_i \rightarrow^* l_j$ the fact that there is an edge between l_i and l_j in $\rightarrow^*$. Moreover, all levels l are such that $l_b \rightarrow^* l$, and $l \rightarrow^* All$. □

Intuitively, a dimension schema is a directed acyclic graph (DAG). Each node in this graph represents an aggregation level and is annotated with a list of descriptors. There is a distinguished level denoted *All* without descriptors, and a unique bottom level. All levels are (directly or transitively) reachable from the bottom, and all levels (directly or transitively) reach the level *All*.

Definition 1 could be simplified if we ignore the level descriptors, and consider each node in the graph as a single data element. However, we decided to state the formal model in this way to account for the way in which the user operates with the cube in real-world practice, where the user defines an attribute as an aggregation (dimension) level, and descriptors that are displayed after aggregation has been performed.

Definition 2 (Dimension Instance)
An instance *I* of a dimension schema $\langle nameDS, \mathcal{L}, \rightarrow \rangle$ consists of (a) a finite set of members $\mathcal{M}_l$, for each level l in the first component of each tuple in $\mathcal{L}$ in Definition 1, such that each member belongs to Dom_l and can be uniquely identified (the level *All* has a unique member *all*); (b) a set of partial functions, denoted as *Roll-up functions* (following (Cabibbo & Torlone, 1997)) of the form $Roll\text{-}up_{\text{li}}^{\text{lj}}$ from the members of level l_i to the members of level l_j, for each pair of levels l_i and l_j in the first component of the pairs in $\mathcal{L}$, such that $l_i \rightarrow l_j$ in $\rightarrow$; (c) a collection of functions $f_l^1,\ldots f_l^k$ mapping members of l to values in the domain of each descriptor $d_1,\ldots,d_k \in LD$, for each level l in the first component of each tuple in $\mathcal{L}$. □

Intuitively, a dimension instance is also a direct acyclic graph (DAG). Associated with an edge (l_i,l_j) in the schema graph such that $l_i \rightarrow l_j$, there is a function from l_i to l_j. This function describes how members in the lower level aggregate to members in the upper level. Therefore, a dimension instance is just a collection of such functions. The following example illustrates this.

Example 1 (Dimension Schema and Instance)
On the left hand side of Figure 1 we can see three dimension schemas. Let us consider the one for dimension `Pollution`. The schema of this dimension is formally defined as follows.

nameDS = `Pollution`,
$\mathcal{L}$= {⟨ `Pollutant`,(`name`, `loadLimit`)⟩,
⟨`Type`,(`name`)⟩,…,⟨`Group`,(`name`)⟩},
$\rightarrow$ = {`Pollutant` $\rightarrow$ `Type`, `Type` $\rightarrow$ `Group`, `Group` $\rightarrow$ `All`, `Pollutant` $\rightarrow$ `Category`, `Category` $\rightarrow$ `All` }

The instances for the dimension `Pollution` are depicted in Figure 3, and are of the form:
$\mathcal{M}_{\text{Pollutant}} = \{p_1,\ldots,p_5\}$, $\mathcal{M}_{\text{Type}} = \{t_1,t_2,t_3\}$, …, $\mathcal{M}_{\text{Category}} = \{c_1,c_2\}$
$Roll\text{-}up_{Pollutant}^{Type} = \{(p_1,t_1),(p_2,t_2),\ldots,(p_5,t_3)\}$
…

$Roll\text{-}up_{Pollutant}^{Category} = \{(p_1,c_1),(p_2,c_1),\ldots,(p_5,c_2)\}$
$Roll\text{-}up_{Category}^{All} = \{(c_1,all),(c_2,all)\}$
$f_{Pollutant}^{name}(p_1)$ = CO,…,$f_{Pollutant}^{name}(p_5)$ = PM.
$f_{Pollutant}^{loadLimit}(p_1)$ = 34,…,$f_{Pollutant}^{loadLimit}(p_5)$ = 44.
…
$f_{Category}^{name}(c_1)$= gas,…,$f_{Category}^{name}(c_2)$ = solid.

Intuitively, the aggregation hierarchy is used as follows: let us suppose that the concentrations of pollutants p_2 and p_3 measured at station s_1 in a given day d_1 are 30 and 40 mg/m^3, respectively. If we want to know the average concentration aggregated by type on that day, according to the instance depicted in Figure 3, p_2 and p_3 will contribute to the aggregation over type t_2. Thus, for d_1, s_1, t_2, we will have a value of 35. □

Definition 3 (Base Cube Schema)
A *base cube schema* is a tuple ⟨*nameCS*, $\mathcal{D}$, $\mathcal{M}$⟩ where *nameCS* is the name of the cube, $\mathcal{D}$ is a finite set of dimension levels, with $|\mathcal{D}|=d$, corresponding to d bottom levels of d dimension schemas, different from each other, and $\mathcal{M}$ is a finite set of m attributes called measures. Each measure also has an associated data type. □

Definition 4 (Base Cube Instance)
Consider a base cube schema ⟨*nameCS*, $\mathcal{D}$, $\mathcal{M}$⟩; each $l_{bi} \in \mathcal{D}$, i=1,…,d, has a set of members. Let us call *Points*={(c_1,…,c_d) | c_i is a member of l_{bi}, i=1,…,d }. A *base cube instance* C is a partial function C: *Points* → dom(M_1) ×…× dom(M_m) where $M_i \in \mathcal{M}$, i=1,…,m. □

Example 2 (Cubes)
Let us now define a base cube denoted `AirQuality`, composed by dimensions `Pollution`, `Time`, and `Station`, and a measure `concentration`, as introduced in our application scenario (Figure 1). The base cube has schema ⟨`AirQuality`, {`Pollutant`,`Time`,`Station`}, {`concentration`}⟩ with instances of the form $C(p_1,t_1,s_1)$=35,…, $C(p_5,t_3,s_4)$=44,…. □

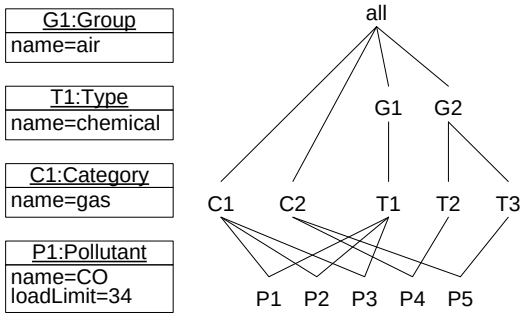


Figure 3: Some instances (left) and roll-up functions (right) of dimension `Pollution` of Figure 1.

From the base cube we can define so-called cube views, that means, cubes whose coordinates are not at the bottom level of the component dimensions, and/or their dimensionality is lower than the base cube. These cube views can be obtained through a collection of operators (see after) and are defined as follows.

Definition 5 (Cube View)
Consider a base cube schema S=⟨*nameCS*, $\mathcal{D}$, $\mathcal{M}$⟩. A cube view schema of S is a cube schema S_v=⟨*nameCV*, $\mathcal{D}_v$, $\mathcal{M}_v$⟩ such that $\mathcal{D}_v \subseteq \mathcal{D}$, with $|\mathcal{D}_v|=d_v$, and $\mathcal{M}_v \subseteq \mathcal{M}$ is a finite set of measures, with $|\mathcal{M}|=m_v$. There is a partial function mapping levels $l_{bi} \in \mathcal{D}$ to levels $l_i \in \mathcal{D}_v$ such that $l_{bi} \rightarrow^* l_i$. Let us assume that in S_v each $l_i \in \mathcal{D}_v$ has an associated set of members. Let us call $Points_v$= {(c_1,…,c_d) | $c_i \in \mathcal{M}_{li}$, i = 1,…,d_v}. A cube view instance C_v is a partial function C_v: $Points_v$ → dom(M_1) × …× dom(M_{mv}) where $M_i \in \mathcal{M}_v$, i =1, …, m_v. □

Example 3 (Cube View)
Given the base cube in Example 1, a cube view schema can be defined as:

⟨ `AirQuality_View`, {`Group`, `Month`, `Station`}, {`concentration`}.

with instances of the form:

$C_v(g_1,m_1,s_1)$=102,…, $C_v(g_2,m_3,s_4)$=85,… □

Cube Algebra Definition Language

We now sketch a language, which we denote CADL (standing for Cube Algebra Definition Language). The language aims at providing a description of the conceptual data cube model, and could be the basis of a Cube Definition Language at lower abstraction levels.

In CADL, a cube schema is defined using the keyword `CUBE` followed by the name given to the cube. Each cube is composed of a collection of dimensions, identified with the keyword `DIMENSION` followed by the dimension's name. After defining a dimension, we must list all dimension levels, along with their descriptors. A dimension level is defined with the keyword `LEVEL`, followed by its name, and the type of the members of the level (by default, an integer, but any type could be defined). Following Definition 2, dimension levels have set semantics (i.e., they have no duplicates). In addition, each level contains a collection of associated descriptors (Definition 1), which in CADL are denoted attributes, and defined with the keyword `ATTRIBUTES`, followed by a list of attribute names and their types. In addition, the optional keyword `UNIQUE` indicates that the value of the attribute is unique among all the values of such attribute for *all* the level members. In this way, the user can model the cube in different ways. For example, she may want to use identifiers to refer to level members (to decouple these names from the actual member names, as usual in OLAP), or represent level members with their actual values. For example, in the Time dimension, it is usual to refer to members using consecutive integers, and within the attributes indicate the actual date.

For our running example, we define the `AirQuality` data cube, along with its dimensions, dimension levels, and dimension attributes in CADL as follows.

```
CUBE AirQuality {
DIMENSION Time
LEVEL Day ATTRIBUTES {date
(date) UNIQUE, season (string)}
LEVEL Month ATTRIBUTES {month
(string) UNIQUE}
LEVEL Quarter ATTRIBUTES
{quarter (string) UNIQUE}
LEVEL Semester ATTRIBUTES
{semester (string) UNIQUE}
LEVEL Year ATTRIBUTES {year
(string) UNIQUE}
Day ROLL-UP to Month
Month ROLL-UP to Quarter
Quarter ROLL-UP to Semester
Semester ROLL-UP to Year
```

In this dimension, we have chosen, as usual, to give integer values to members in the level of the `Time` dimension, and represent the actual values as attributes (i.e., level descriptors).

```
DIMENSION Station
LEVEL Station ATTRIBUTES {name
(string) UNIQUE}
DIMENSION Road
LEVEL Road ATTRIBUTES {name
(string) UNIQUE, length (real)}
```

Here, we have chosen to represent roads with their actual names. We show how this is used in the query language in next section.

```
DIMENSION Pollution
LEVEL Pollutant ATTRIBUTES {
name (string) UNIQUE,
loadLimit (real)}
LEVEL Category ATTRIBUTES {
name (string) UNIQUE}
LEVEL Type ATTRIBUTES {
name (string) UNIQUE}
LEVEL Group ATTRIBUTES {
name (string) UNIQUE}
Pollutant ROLL-UP to Category
Pollutant ROLL-UP to Type
Type ROLL-UP to Group
DIMENSION Geography
LEVEL District ATTRIBUTES {name
(string) UNIQUE, area (real)}
LEVEL Province ATTRIBUTES {name
(string) UNIQUE}
```

```
District ROLL-UP to Province
MEASURE concentration (real)}
```

CUBE ALGEBRA QUERY LANGUAGE

In this section we sketch our proposal for a query language that implements the ideas discussed in the preceding sections. We define an algebra, which we denote *Cube Algebra*, such that the user could define her queries just by means of the typical OLAP operators. We first describe how the user would be able to operate in our application scenario, intuitively manipulating the cube using the traditional OLAP operations, e.g., slicing, dicing, rolling-up, drilling down, and pivoting. This basic set of operators can be extended with many other functionalities. We illustrate this extensibility introducing the *Map* operation which, in the style of functional programming, allows changing the values of the measures in a cube, for example, to convert currency units, or perform what-if analysis. Then, we formally define each operator in our algebra. We conclude with some example queries that illustrate the query language.

Application Scenario: OLAP Operations for Pollution Control

We will now discuss how the user-centric conceptual model we propose in this paper could be use to analyze data. Let us recall the cube in Figure 1, containing quarterly values of pollutant concentration at each measuring station. The end user can operate intuitively over this cube in order to analyze data in different ways. Figure 4 shows a sequence of such operations, which start from the initial cube of Figure 1. We describe her operations on a step by step basis.

The user first wants to compute the sum of concentrations per semester, station, and pollutant, to look for significant differences between these periods, if they exist. For this, the Cube Algebra offers her a *Roll-up* operation, which she applies along the `Time` dimension. The result is shown in Figure 4a: the new cube contains two values over the `Time` dimension, each corresponding to one semester (the original cube contained four values, one for each quarter). The remaining dimensions are not affected. All values in cells corresponding to the same pollutant and station (for example, `P1` and `S1`, respectively), and to quarters `Q1` or `Q2`, contribute to the aggregation to the values in the first semester (`S1`). We can see in Figure 1 that the concentration of `P1` measured at station `S1` for the first and second quarters are, respectively, 21 and 27. In Figure 4a we also see that these values are aggregated to 48 in the first semester. Computation of the cells corresponding to the second semester proceeds analogously.

Our user then notices that in the second semester the concentration of pollutant `P3` at station `S1` was unusually high. The Cube Algebra allows her to *drill down* along the Time dimension, to the month level, to find out if this high value is due to a particular month. In this way, she discovers that December presented a much higher concentration of this pollutant than the other months (Figure 4b). Note that since she now starts from data aggregated by semester, station, and pollutant, the user needs first to take the cube back to the quarter aggregation level, and then continues drilling-down to the month level.

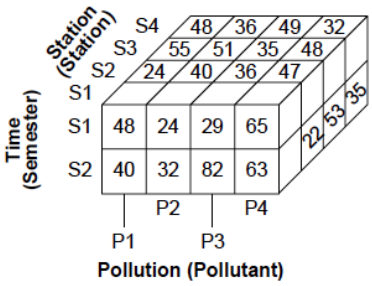
Continuing her browsing of the cube, our user now wants to see the cube with the Time dimension on the *x* axis. Therefore, she rotates the axes of the cube without changing granularities. This restructuring operation is called *pivoting* (Figure 4c). (Note that she previously *rolled-up* the cube back to the one of Figure 1).

She then wants to visualize time series of average pollutant concentration by quarter,

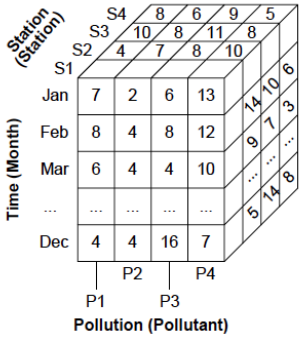
only for the station `S1`. For this, she first applies a *dice* operator that selects the sub-cube containing only values for the station `S1`, and then eliminates the `Station` dimension, applying a *slice* operation. This is depicted in Figure 4d. Here, she obtained a matrix, where each column represents the evolution of the concentration of a pollutant by quarter, i.e., a collection of time series.

Next, she wants pollution information corresponding only to stations `S1` and `S2` in the first two quarters. For this, starting over from the original cube, she produces a sub-cube, again using the *dice* operator (Figure 4e).

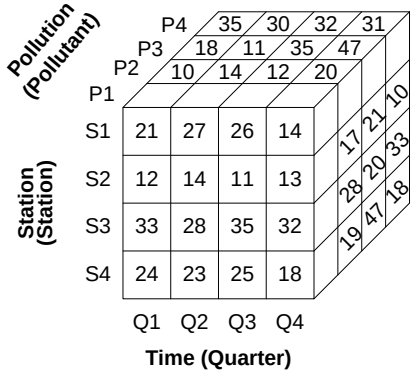
Finally, instead of a cube containing pollution values, she wants to produce a cube containing indicators of pollution classified in four categories: `Low` (`L`) for values greater or equal to 20; `Moderate` (`M`) for values greater than 20 and less or equal to 30; `High` (`H`) for values greater than 30 and less or equal to 45; and `Very High` (`VH`), for values greater than 45. For this, she uses the *Map* operation shown in Figure 4f.



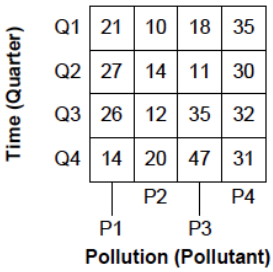
(a) Roll-up to the `Semester` level



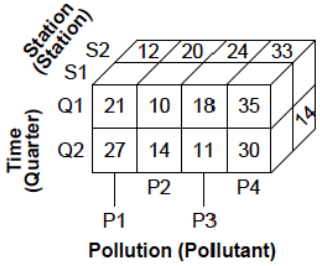
(b) Drill-down to the `Month` level



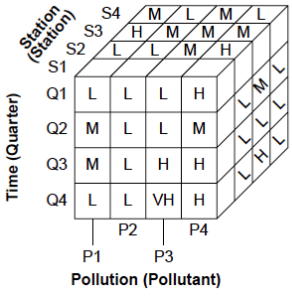
(c) Pivot



(d) Slice on `Station` for `StationId` = '`S1`'



(e) Dice on `Station` = '`S1`' or '`S2`' and `Time.Quarter` = '`Q1`' or '`Q2`'



(f) Map function for defining contamination levels

Figure 4: OLAP operations

In what follows, to make the examples more interesting, we add a `Geography` dimension, composed of levels `District` and `Province`, with a roll-up relationship defined between them, and a `Road` dimension, indicating the roads where the stations are located.

Cube Algebra Query Operators

We now formally define the operators of our Cube Algebra in terms of our data model. Even though there are many works describing multidimensional operations (Agrawal, Gupta & Sarawagi, 1997; Gyssens & Lakshmanan, 1997; Vassiliadis, 1998), curiously none of them describe a common whole set of them in terms of the well-known *slice*, *dice*, *roll-up* (and its inverse, *drill-down*) and *drill-across*, which are the ones that intuitively reflect how an OLAP user manipulates a cube, or combines two cubes. Existing work such as the one cited above usually defines a subset of these operators, combined with other ones, which adapt to the model proposed by the authors. It can be shown that these operators are orthogonal to each other, which means, no one of them can be expressed as a combination of the others.

This basic set of operators can be extended with many other ones, in order add functionalities to the language. As an example of this, we introduce the *map* operator, which applies the same function to *all* the cells in a data cube, allowing, for example, currency conversion, or more complex operations such as the one illustrated in the example of previous section.

Remark Although the *pivot* operator was introduced in Figure 4c, since it does not modify either the cube schema or the cube instances, and it is just used for (mainly) interactive visualization, we do not include it in the following discussion. □

Dice This operator receives a cube and a Boolean condition φ, and returns another cube containing only the cells that satisfy φ. The *syntax* for this operation is

```
DICE(cube_name, φ)
```

where φ is a Boolean condition over dimension levels and measures.

The *semantics* is the following. *Dice* receives a cube view with schema $Sv=\langle nameCV, \mathcal{D}_v, \mathcal{M}_v\rangle$ and instances with the form $C_v(c_{11},\ldots,c_{1d})= m_1,\ldots, C_v(c_{m1},\ldots,c_{kj}) = m_r$ (for simplicity let us assume only one measure), and returns a cube with the same schema, and the points $(c_{i1},\ldots,c_{id}, m_i)$ that make the condition φ true. We can consider *dice* analogous to a relational *selection*.

Roll-up and Drill-down The *roll-up* operator aggregates measures according to a dimension hierarchy (using an aggregate function), to obtain measures at a coarser granularity for a given dimension, based on the use of the dimension hierarchy.

The *syntax* for the *roll-up* operation is:

```
ROLL-UP(cube_name,
Dimension->level,
(measure,aggregate_function)*)
```

The term `Dimension->level` indicates to which level in a dimension we want the roll-up to be performed. Note that since there can be more than one measure, we must specify an aggregate function for each one of them. In what follows, we assume that if there is only one measure in the cube, for conciseness we only specify the aggregate function.

As for the *semantics*, *roll-up* receives a cube view with schema $S_v=\langle nameCV, \mathcal{D}_v, \mathcal{M}_v\rangle$, a level l in a dimension D, such that $l_s \in \mathcal{D}_v$, $l_s \rightarrow^* l$ in D, and an aggregate function F_{agg}. *Roll-up* returns a cube view whose cells are aggregated along D up to the level l. Thus, all values $v_i,\ldots,v_k$ in the

original cube view, such that $C_v(c_1,\ldots,c_{li},\ldots,c_d) = v_j$, j=1,…,k (Definition 5) contribute to the aggregation over *Roll-up*$(c_{li})_{ls}^{l}$.

Example 4 (Roll-up)
Suppose in Example 1 that we have the following coordinates for the cube `AirQuality`: (p_1,t_1,35), (p_5,t_3,44), (p_4,t_3,22). According to Figure 3:

Roll-up$(p_1)_{Pollutant}^{Category} = c_1$;
Roll-up$(p_4)_{Pollutant}^{Category} = c_2$
Roll-up$(p_5)_{Pollutant}^{Category} = c_2$.

Then rolling-up from the cube `AirQuality` to a new cube with schema ⟨`AirQCateg`, {`Category`, `Time`}, {`concentration`}⟩ yields the instance (c_1,t_1,35), (c_2,t_3,66). □

Drill-down de-aggregates previously summarized measures and can be considered the inverse of *roll-up*. Following Agrawal, Gupta & Sarawagi (1997), we consider *drill-down* a high-level operation that can be implemented by tracking the (stored) paths followed during user rolling-up. Therefore, we omit its definition.

Slice This operator reduces the dimensionality of a cube by removing one of its dimensions, i.e., a cube of n-1 dimensions is obtained from a cube of n dimensions. The selected dimension must contain a unique value in its domain. If the dimension has more than one value two approaches can be used: apply either a *roll-up* operator for summarizing into a singleton (i.e., *all*) (Agrawal, Gupta & Sarawagi, 1997), or (prior to slicing) a *dice* operator, to obtain a cube with only one value in the selected dimension.

The *syntax* of this operator is:

```
SLICE(cube_name, Dimension,
[ROLL-UP{Aggregate_function}])
```

According to what we explained above, `ROLL-UP{Aggregate_function}` stands for `ROLL-UP(cube_name, Dimension {All,Aggregate_function)}`. This yields a more concise expression. If the roll-up is not included, *slice* will have two arguments, meaning that the dimension instance has already been reduced to a single value. If this is not the case, the operator fails. That is, the operator is analogous to a relational *projection*.

The *semantics* is the following. *Slice* receives a cube view with schema S_v=⟨*nameCV*, $\mathcal{D}_v$, $\mathcal{M}_v$⟩, and instances of the form {(c_{11},…,c_s,…,c_{1d}, m_1),…, (c_{m1},…,c_s, …,c_{kd}, m_r)} (where c_s is the unique value in l_s), and a dimension name d_s (assume that the level $l_s \in \mathcal{D}_v$ belongs to dimension d_s). The operator returns a cube with schema Sv_s=⟨*nameCV*, ($\mathcal{D}_v \setminus l_s$), $\mathcal{M}_v$⟩, where the instances are of the form (c_{11},…, c_{s-1}, c_{s+1},…,c_{1d}, m_1),…,(c_{m1},…, c_{s-1}, c_{s+1},…,c_{kd}, m_r), that is, the same as in the original one, except the coordinate corresponding to l_s.

Drill-across This operator relates information contained in two data cubes having the same dimensions. Thus, measures from different cubes can be compared. According to Kimball and Ross (2002), *drill-across* can only be applied when both cubes have both the same schema dimensions and the same instances. However, other authors relax this restriction. This is the approach of Abelló, Samos & Saltor (2002), who define two concepts: (a) *Dimension-Dimension Derivation*: Used when two dimensions come from a common concept although their structures differ. For example, they do not have the same levels because their granularities are not the same. This would be the case of a spatial dimension with 'point' granularity, and another one with 'polygon' granularity. (b) *Dimension-Dimension Association*: Corresponds to the case in which two cubes have different dimensions, but one of them could be defined as the association of several ones. For example in one cube we define latitude and longitude as separated dimensions; in another one we store only one dimension containing the 'point' geometry. A mapping function can solve this problem. Other

authors also address this problem, all of them along the same lines (Cabibbo & Torlone, 2004; Riazati, Thom & Zhang, 2008). We assume that the operator receives two compatible cubes (i.e., sharing dimensions and instances). The syntax of the operator is:

```
DRILL-ACROSS(cube_name_1,
cube_name_2).
```

Let us now explain the *semantics*. *Drill-across* receives two cube views with schemas $Sv_1=\langle nameCV_1, \mathcal{D}_v, \mathcal{M}_{v1}\rangle$ and $Sv_2=\langle nameCV_2, \mathcal{D}_v, \mathcal{M}_{v2}\rangle$, that means, the dimension levels are the same. The corresponding instances (except for the measures) are the same. The result is a cube with schema $Sv_{out}=\langle nameCV_{out}, \mathcal{D}_v, \mathcal{M}_{v1} \cup \mathcal{M}_{v2}\rangle$, with the same instances of the input cubes. In other words, the operator is analogous to a relational *natural join*).

We now formally define the *Map* operator, which extends the basic set of operations defined above. We remark that this is only one of the many operators the Cube Algebra could be extended with.

Map This operator receives a cube and a collection of pairs (m_i,f_i), where m_i is a measure and f_i is a function mapping values in $Dom(m)$ to values in the same of another domain (see the example above, where concentration values in the domain of the real numbers are mapped to the domain of strings, by a partitioned function). The operator returns another cube, with the same dimension schema and instances, and with the values in each cell that correspond to the mappings produced by each function f_i. The *syntax* for the *Map* operation is:

```
MAP(cube_name, (measure,
function)*)
```

The *semantics* is the following. *Map* receives a cube view with schema $Sv=\langle nameCV, \mathcal{D}_v, \mathcal{M}_v\rangle$ and instances with the form $C_v(c_{11},\ldots,c_{1d}) = m_1,\ldots,C_v(c_{m1},\ldots,c_{kj}) = m_r$, (for simplicity let us assume only one measure), and returns a cube with the same schema, and instances of the form $C_v(c_{11},\ldots,c_{1d}) = f(m_1),\ldots,C_v(c_{m1},\ldots,c_{kj}) = f(m_r)$.

Cube Algebra QL by Example

In what follows we show queries that give the flavor of the language, and show that this algebra allows an OLAP user to express queries using just the operators she is acquainted to, instead, for example, of complex MDX expressions.

Query 1 *For each province and pollutant category, give the average concentration by quarter.*

```
c1 ← SLICE (AirQuality, Station,
ROLL-UP{Avg})
c2 ← SLICE (c1, Road, ROLL-
UP{Avg})
c3 ← ROLL-UP (c2,{Time->Quarter,
Pollution->Category, Geography->
Province}, Avg)
```

Remark The *roll-up* operator can only be applied over one dimension at a time. For simplicity, in the query above, the expression

```
c3 ← ROLL-UP(c2, {Time->Quarter,
Pollution->Category, Geography->
Province},Avg)}
```

is a shorthand for:

```
c3 ← ROLL-UP(c2, Time->Quarter,
Avg)};
c4 ← ROLL-UP(c3,
Geography->Province, Avg)};
c5 ← ROLL-UP(c4,
Pollution->Category, Avg)};
```

This is the syntax we use in the sequel. □

Remark In MDX, the *de facto* standard for OLAP, Query 1 would read:

```
SELECT
[Geography].[Province].Members
ON COLUMNS,
{CROSSJOIN([Pollutant].[Category
].Members,
[Time].[Quarter].Members)} ON
ROWS
```

```
FROM [AirQuality]
```

Compared to the simplicity of the Cube Algebra expression above, the MDX query looks cryptic and unintuitive. In part, this is due to the fact that, as shown in Figure 2, MDX is placed at the logical level, while the Cube Algebra is at the conceptual level. Besides, MDX has not a clearly defined semantics. On the contrary, we have defined the semantics of each one of the Cube Algebra operators. □

Query *2* *Number of districts where, for at least one pollutant, the average load of air pollution in 2011 was larger than the concentration limit.*

```
c1 ← ROLL-UP (AirQuality,
Time->Year,{Avg} )
c2 ← DICE (c1, Time.Year.year =
2011)
c3 ← SLICE (c2, Time,
ROLL-UP{Avg})
c4 ← SLICE (c3, Station,
ROLL-UP{Avg})
c5 ← SLICE (c4, Road,
ROLL-UP{Avg})
```

We have obtained a cube with dimensions `Geography` and `Pollution`. Now:

```
c6 ← DICE (c5, concentration >=
Pollution.Pollutant.loadLimit)
c7 ← SLICE (c6, Pollution,
ROLL-UP{Avg})
c8 ← ROLL-UP (c7,
Geography->All, {Count}})
```

The semantics of the expression

```
DICE (c4, concentration >=
Pollution.Pollutant.loadLimit)
```

is the following: each cell in `c4` is analyzed, and the values of the variables are instantiated with the values of the cell coordinates and measures (i.e., `Pollution.Pollutant` is instantiated with the identifier of the member of the pollutant corresponding to the cell). Analogously, `concentration` is obviously instantiated with the value of the measure in the cell. The only limitation is that we can move along a path starting from a bottom level of the cube used as argument in the operator.

Query 3 *Build a cube with the dimension of the original cube, containing only concentration corresponding to stations located in districts in the province of Limburg, and to polluting agents of 'organic' type such that the station had at least once registered a concentration for the pollutant, higher than the limit.*

This query is simply expressed as:

```
c1 ← DICE (AirQuality,
concentration >=
Pollution.Pollutant.loadLimit
AND Geography.Province.name
='Limburg' AND
Pollution.Category.name =
'organic').
```

Query 4 *For stations, and pollutants belonging to the 'organic' category, give the maximum concentration by month.*

```
c1 ← DICE (AirQuality,Pollution.
Category.name='organic')
c2 ← SLICE (c1, Road,
ROLL-UP{Max})
c3 ← SLICE (c2, Geography,
ROLL-UP{Max})
c4 ← ROLL-UP (c3, Time->Month,
{Max})
```

Query 5 *Stations located over the part of the E34 road within the Berchem district, with an average content of nitrates in the last quarter of 2011 above the load limit for that pollutant.*

```
c1 ← DICE (AirQuality,
Geography.District ='Berchem'
AND Time.Quarter.quarter ='Q4-
2011' AND Time.Year.year= 2011
AND Pollution.Category.name=
'Nitrates' AND Road.Road.name =
'E34')
c2 ← SLICE (c1, Road)
c3 ← SLICE (c2, Geography)
```

Note that in the expression `Road='E34'` we are referring to a level name and not to a descriptor name. Also, in the `SLICE` operation that generates cubes `c2` and `c3`, we do not use the third argument, since the previous dicing selected unique values for roads and districts. Cube `c3` has `Station`, `Time`, and `Pollution` dimensions.

```
c4 ← ROLL-UP (c5, Time->Quarter, Avg)
c6 ← DICE (c5, concentration >= Pollution.Pollutant.loadLimit)
```

The following query illustrates the use of the *Map* operator. In our running example, assuming that pollutant concentrations are expressed in mg/m^3 (milligrams per cubic meter), we want to express the results in $\mu g/m^3$ (micrograms per cubic meter).

Query 6 *Average concentration by Station, Time, District, Road, and pollutant Category, expressed in* $\mu g/m^3$.

```
c1 ← MAP (AirQuality, concentration, Mult(1000))
c2 ← ROLL-UP (c1, Pollution-> Category, Avg })
```

In the expression that generates `c1`, `Mult (1000)` is a function that given a value, multiplies it by a constant (in this case, 1000).

The next query shows the use of the *drill-across* operator. Since we need two cubes, assume that we have a cube denoted `Demography` with the dimensions `Time` and `Geography` described above, and measure `population`. The instances of the dimensions satisfy the operator's preconditions.

Query 7 *Total population and average pollutant concentration by province and year.*

```
c1 ← SLICE (AirQuality, Road, ROLL-UP{Sum})
c2 ← SLICE (c1, Pollution, ROLL-UP{Max})
c3 ← SLICE (c2, Station, ROLL-UP{Max})
```

Now, the cube `c3` contains just the dimensions `Time` and `Geography`.

```
c4 ← DRILL-ACROSS (c3, Demography)
c5 ← ROLL-UP (c4, Time->Year, concentration, Avg, population, Sum))
c6 ← ROLL-UP (c5, Geography-> Province, concentration, Avg, population, Sum)
```

CONCLUSIONS AND FURTHER WORK

In this article, we have identified the need for an appropriate conceptual model for data warehouses and OLAP systems. This need stems from the fact that logical models (for example, star, snowflake, and constellation schemas) have been deployed for these systems as conceptual models. But logical models represent a system-centric view of data warehouses and OLAP systems and are ultimately implementation concepts. In this article, we propose a user-centric conceptual model for data warehouses and OLAP systems, called the Cube Algebra. It takes the cube metaphor literally and provides the knowledge worker with high-level cube objects and related high-level concepts. A novel query language leverages high-level operations such as *roll-up*, *slice*, and *drill-across*. An important design criterion is that all aspects of the logical level and the physical level are hidden from the user.

We plan our future research in at least three directions. First, further data definition commands have to be added for updating cube schemas, dimension schemas, and measures. In addition, further data manipulation commands are needed for the insertion, deletion, and update of data into cubes. Second, transformation rules are needed that map the concepts of the Cube

Algebra at the conceptual level to corresponding ROLAP, MOLAP, and HOLAP concepts at the logical level. Third, we are interested in adding other data categories such as spatial, spatiotemporal, image, and multimedia data into our Cube Algebra. Questions are here, for example, how the different data categories are integrated and stored, what kind of aggregation operations exist on the different data categories, how the different aggregation operations are defined, and how these operations are implemented.

REFERENCES

Abelló, A., Samos, J., & Saltor, F. (2001) Understanding facts in a multidimensional object-oriented model. In *Proceedings of the 4th ACM International Workshop on Data Warehousing and OLAP*. ACM Press.

Abelló, A., Samos, J., & Saltor, F. (2002). On relationships offering new drill-across possibilities. In D. Theodoratos (Ed.), *Proceedings of the 5th International Workshop on Data Warehousing and OLAP* (pp. 7-13). ACM Press.

Abelló, A., Samos, J., & Saltor, F. (2006). YAM^2: a Multidimensional Conceptual Model Extending UML. *Information Systems, 31* (6), 541-567.

Agrawal, R., Gupta, A., & Sarawagi, S. (1997). Modeling Multidimensional Databases. In W.A. Gray and P. Larson (Eds.), *Proceedings of the Thirteenth International Conference on Data Engineering* (pp. 232-243). IEEE Computer Society Press.

Cabibbo, L., & Torlone, R. (1997). Querying multidimensional databases. In S. Cluet and R. Hull (Eds.), *Proceedings of Database Programming Languages,* LCNS 1369 (pp. 319–335). Springer-Verlag.

Cabibbo, L., & Torlone, R. (1998). A Logical Approach to Multidimensional Databases. In H. Schek, F. Saltor, I. Ramos and G. Alonso (Eds.), *Proceedings of the 6th International Conference on Extending Database Technology*, LCNS 1377 (pp. 253-269) . Springer-Verlag.

Cabot, J., Mazón, J-N., Trujillo, J., & Pardillo, J. (2009). Towards the Conceptual Specification of Statistical Functions with OCL. In *Proceedings of CAiSE Forum* (pp. 7-12).

Golfarelly, M., & Rizzi, S. (1998). A methodological framework for data warehouse design. In *Proceedings of the 1st International Workshop on Data Warehousing and OLAP* (pp. 3-9). ACM Press.

Gyssens, M., & Lakshmanan, L. (1997). A Foundation for Multi-dimensional Databases. In M. Jarke, M. Carey, K. Dittrich F. Lochovsky, P. Loucopoulos and M. Jeusfeld (Eds.), *Proceedings of 23rd International Conference on Very Large Data Bases* (pp. 106-115). Morgan Kaufmann.

Lehner, W. (1998). Modelling Large Scale OLAP Scenarios. . In H. Schek, F. Saltor, I. Ramos and G. Alonso (Eds.), *Proceedings of the 6th International Conference on Extending Database Technology*, LCNS 1377 (pp. 153-167). Springer-Verlag.

Li, C., & Wang, S. (1996). A Data Model for Supporting On-Line Analytical Processing. In *Proceedings of the Fifth International Conference on Information and Knowledge Management* (pp. 81-88). ACM Press.

Marcel, P. (1999). Modeling and Querying Multidimensional Databases: An Overview. *Networking and Information Systems,* 2 (5), 515-548.

Malinowski, E., & Zimányi, E. (2008). *Advanced data warehouse design: From conventional to spatial and temporal applications*. Springer-Verlag.

Pardillo, J., Mazón, J-N., & Trujillo, J. (2008). Bridging the Semantic Gap in OLAP Models: Platform-independent Queries. In (Eds.), *Proceedings of the ACM 11th International Workshop on Data warehousing and OLAP* (pp. 89-96).

Pardillo, J., Mazón, J-N., & Trujillo, J. (2010). Extending OCL for OLAP Querying on Conceptual Multidimensional Models of data warehouses. *Information Sciences 180* (5), 584-601.

Rizzi, S. (2007) Conceptual modeling solutions for the data warehouse. In Wrembel, R., Koncilia, R. (Eds.), *Data Warehouses and OLAP: Concepts, Architectures and Solutions* (pp. 1-26). Idea Group (IGI). Hershey, PA.

Tryfona, N., Busborg, F., & Borch Christiansen, J. (1999). starER: a conceptual model for data warehouse design. In *Proceedings of the 2nd ACM International Workshop on Data warehousing and OLAP* (pp. 3-8). ACM Press.

Tsois, A., Karayannidis, N. & Sellis, T. (2001). MAC: Conceptual data modeling for OLAP. In D. Theodoratos, J. Hammer, M. Jeusfeld and M. Staudt (Eds.), *Proceedings of Data Mining and Data Warehousing* (pp. 5). CEUR-WS.org.

Vassiliadis, P. (1998). Modeling Multidimensional Databases, Cubes and Cube Operations. In M. Rafanelli and M. Jarke (Eds.), *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, (pp. 53-62) . IEEE Computer Society.

[i] We use this simplification in the formal model to avoid the need of referring to a dimension level as `dimension.level`, which would make the formal definition too verbose. However, in the algebra defined in next section, we drop this restriction, and qualify level names with dimension names.