



D W Q

Foundations of **Data Warehouse Quality**

National Technical University of Athens (NTUA)
Informatik V & Lehr- und Forschungsgebiet Theoretische Informatik (RWTH)
Institute National de Recherche en Informatique et en Automatique (INRIA)
Deutsche Forschungszentrum für künstliche Intelligenz (DFKI)
University of Rome «La Sapienza» (Uniroma)
Istituto per la Ricerca Scientifica e Tecnologica (IRST)

M. S. Hacid, U. Sattler

**An Object-Centered Multi-dimensional Data Model
with Hierarchically Structured Dimensions**

Proc. of the IEEE Knowledge and Data Engineering Workshop
Newport, CA, USA, 1997

DWQ : ESPRIT Long Term Research Project, No 22469
Contact Person : Prof. Yannis Vassiliou, National Technical University of Athens,
15773 Zographou, GREECE Tel +30-1-772-2526 FAX: +30-1-772-2527, e-mail: yv@cs.ntua.gr

An Object-Centered Multi-dimensional Data Model with Hierarchically Structured Dimensions*

Mohand-Saïd Hacid

Ulrike Sattler

LuFG Theoretical Computer Science, RWTH Aachen
52074 Aachen, Germany

Abstract

In this paper, we propose a formal framework based on description logics as a basis for both modeling multidimensional databases and understanding related reasoning problems and services. We extend a description logic with new constructors for defining operators on cubes together with a representational framework for hierarchically structured dimensions. The constructors we propose correspond in part to the operators on data cubes given in [1]. Finally, our representation of cubes naturally allows a symmetric treatment of dimensions and measures.

1 Introduction

Recently, much attention has been focused on multidimensional databases. Many commercial systems based on their own data models have been developed. The main strength of multidimensional databases is their ability to view, analyze, and consolidate huge amounts of data. To do so, instead of presenting tables to the user, data is presented in the shape of so-called *data-cubes* which can be manipulated by using operators to cut out pieces from large cubes, change the granularity of dimensions, turn cubes, etc. Because of these functionalities, multidimensional databases play an important role in Decision Support Systems, for On Line Analytical Processing, and in Data Warehousing. In general, multidimensional databases provide two categories of tools: (1)Tools for efficient storage and retrieval of large volumes of data. (2)Tools for viewing and analyzing data from different perspectives. These tools allow interactive querying of data and their analysis, often referred to by “navigation” since this way of querying seems much more intuitive than classical ad-hoc queries.

For describing the interdependence of data in a multidimensional databases, *data cubes* are regarded as most appropriate data model. In a data cube, each axis is associated with a dimension (e.g., time, space, or products) and its according values. Then, points in the cube (called cells) can be associated with values (called measures) of additional dimensions (like sales volumes). This cube representation is well-suited for the representation of multidimensional data because it allows for tracking changes of values depending on the

*This work is partially supported by ESPRIT Basic Research Action no. 22469 - Foundations of Data Warehouse Quality.

changes of each of the dimensions. If the same information were represented in a table, one would have to skip lines for this kind of tracking. Displaying information to the user as well as navigation in this information are tasks that are heavily supported by this data model.

In OLAP, Decision Support Systems and Data Warehouse applications, aggregation is an important operation. Data is often summarized at various level of granularities and on various combinations of attributes. Therefore, queries are more complex and may take long time to complete. To face up to this problem, a great deal of effort has been invested in developing techniques for optimizing queries involving views and aggregate functions [9, 7, 10, 11, 12].

Unfortunately, a unifying framework for multidimensional databases is still missing. Hence, different multidimensional models, operators, techniques, etc. cannot be compared to each other or evaluated. Furthermore, there is a certain lack of common vocabulary and common understanding of multidimensional data models. In this paper, we propose an object-centered, logical framework for multidimensional data models to overcome this lack. It is a first step towards the development of a framework which (1) is powerful enough to facilitate comparison or evaluation of different multidimensional data models. Hence it has to have enough expressive power to represent the relevant properties of multidimensional data models. Second, it is equipped with well-defined semantics, and third, it allows the precise definition of relevant reasoning services and problems. These definitions are a prerequisite for the investigation of these problems with respect to their computational complexity and the comparison and evaluation of reasoning techniques.

The framework presented in this paper is based on Description Logics, a family of logics which have already proved their usefulness as unifying framework for object-centered representation formalism [5]. These logics are equipped with well-defined semantics and sound and complete reasoning algorithms. In fact, a main characteristics of these logics is that problems like satisfiability, containment or consistency are effectively decidable. Finally, they can be equipped with a kind of interface to specific “concrete domains” (e.g., integers, strings, reals).

To serve as a framework for multidimensional data model, we will describe data cubes within a description logic. We build on works by Agrawal et al. [1] and Baader and Hanschke [3] to develop a description logic that takes into account the basic operators on cubes. Like [1] operators are defined on cubes and produce as output cubes. Thus, we can define classes of cubes and build new cubes from already existing ones using operators like join, restrict, aggregate, etc. Furthermore, we propose a representational framework for hierarchically structured dimensions.

Paper outline: Section 2 gives an example illustrating the features of a data cube and hierarchically structured dimensions. Section 3 introduces the syntax and declarative semantics of our language based on $\mathcal{ALC}(\mathcal{D})$ [3]. We conclude in Section 4 by highlighting some perspectives.

2 Example

Consider a service for statistics which records information regarding the results of students in exams over time. We have exam results for each student for each year and for each subject (Suppose we have six subjects, *Mathematics*, *Physics*, *CS*, *Geography*,

History, and *Philosophy*). Figure 1 shows a representation of this data in the relational model. These data are called *micro data* or *base data*, since it is supposed to be the most fine-grained data available. In this example, there is more than one-for-one correspondence between the different fields.

Student	Year	Subject	Mark
S1	1995	Mathematics	14
S1	1995	Physics	12
...
S1	1996	History	13
S1	1996	Philosophy	10
...
S2	1995	Mathematics	12
...
S2	1996	Physics	14
...
S3	1995	Mathematics	10
...
S3	1996	Philosophy	13
...

Figure 1: A relational representation of exams results

A much concise and clearer way to represent this data would be as a three-dimensional matrix (Figure 2). This representation is commonly known as a data cube (in the multidimensional jargon). It is a cube with three intersecting 2D cross tabs. In this representation, each axis is associated with a dimension, that is a column of a relational table. Elements within a dimension are called positions. Points in a data cube are called cells, and each cell in Figure 2 is associated with the corresponding element of the column *Mark*. Then the cube is said to have dimensions *Student*, (examination-) *Year*, and *Subject*, and to have the measure *Mark*. The data in this representation is more organized, hence it is more easily accessible than the organization offered by the relational table. Note that the relational representation (Figure 1) is only possible because the *Mark* is uniquely determined by the *Student*, the (examination-) *Year* and the *Subject*.

As multidimensional databases are designed for ease and performance in manipulating and analyzing complex data structures, values of dimensions or measures can be aggregated, decomposed, or combined to new values to form what is commonly known as *macro data* or *summary data*. This corresponds to what is called data *consolidation* [6, 8].

Dimensions can be hierarchically structured¹: For example, each subject is part of a subject category (i.e., *Scientific-subject*, *Non-scientific-subject*), and each category is part of *All-subjects*. In general, dimensions can have more complex structures: In Figure 3, the dimension *Student* is given. The group of students is decomposed with

¹Hierarchies like this one are called *containment hierarchies* in the context of statistical databases

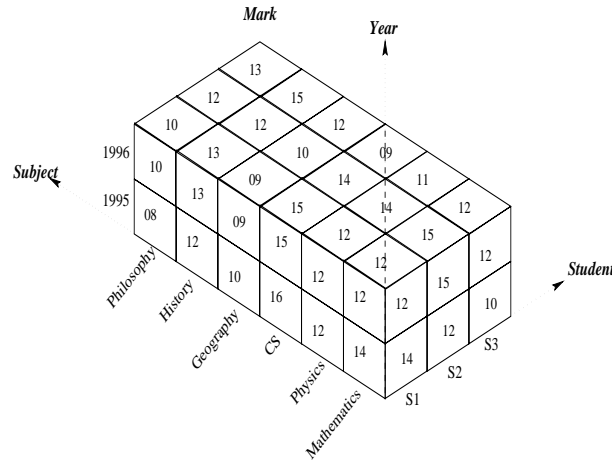


Figure 2: A cube representation of exams results

respect to two different *categories*: With respect to their residence, and with respect to their sex. While the gender hierarchy has only three levels (i.e., **students** = {S1, S2, ...}, **sex** = {Male, Female}, All_students), the residence hierarchy has four levels. Giving names to these levels enables us to drill down or roll-up information to the granularity we have in mind—by (possibly) skipping intermediate levels. In the following, we will call the most fine-grained level *base level*.

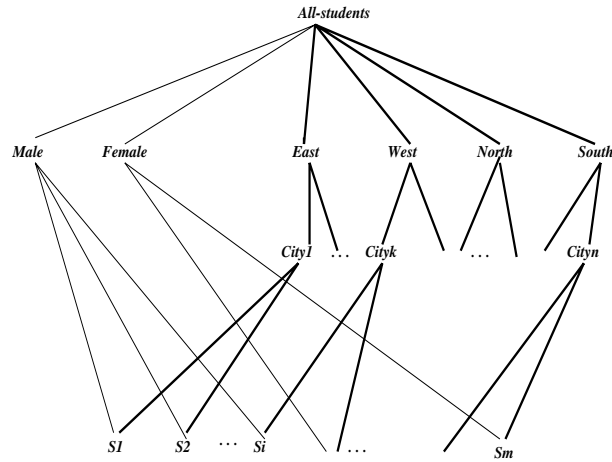


Figure 3: The multiple hierarchically structured Student dimension

Furthermore, an important difference between our approach and the one presented in [1] is that we want to distinguish between the internal representation of a cube and its visualization. Given an instance c of cube as described below, c can contain much more information than what can be shown in one single cube. To visualize the information contained in c , one chooses at most 3 dimensions d_1, d_2, d_3 plus n measures f_i . This part of the information contained in c can then be visualized as a cube. The push and pull operators take a cube and transform dimensions into measures and vice versa. In our

approach, this can be reduced to asking for a different visualization of the cube.

3 The language

In this section, the syntax and semantics of a description logic for the representation of multidimensional data is introduced. This logic is based on the one presented in [3] and extends it by a set of operators for the handling of data cubes.

3.1 Basic Definitions

In order to define aggregation appropriately, we first introduce the notion of *multisets*: In contrast to simple sets, in a multiset an individual can occur more than once, this is to say that, for example, the multiset $\{1\}$ is different from the multiset $\{1, 1\}$. Multisets are needed to ensure that, for example, the average age of one's children is calculated correctly in the case he has twins or triplets.

Definition 1 (Multisets) *A multiset (or bag) on a set S is a subset M of $S \times \mathbb{N}$ such that for every $a \in S$ there is an $n \in \mathbb{N}$ (the number of occurrences of a in M) such that $(a, m) \in M$ if and only if $m < n$. The set of all multisets on S is denoted by $\mathcal{M}(S)$.*

Next, we define *concrete domains*, which are used to incorporate application-specific domains (i.e., strings, reals, non-negative integers, etc.) into the abstract domain of objects.

Definition 2 (Concrete Domains) *A concrete domain $\mathcal{D} = (\text{dom}(\mathcal{D}), \text{pred}(\mathcal{D}), \text{funct}(\mathcal{D}), \text{agg}(\mathcal{D}))$ consists of:*

- the domain $\text{dom}(\mathcal{D})$,
- a set of predicate symbols $\text{pred}(\mathcal{D})$, where each predicate symbol $P \in \text{pred}(\mathcal{D})$ is associated with an arity n and an n -ary relation $P^{\mathcal{D}} \subseteq \text{dom}(\mathcal{D})^n$,
- a set of 2-ary function symbols $\text{funct}(\mathcal{D})$, where each function symbol $f \in \text{funct}(\mathcal{D})$ is associated with a function $f^{\mathcal{D}} : \text{dom}(\mathcal{D})^2 \rightarrow \text{dom}(\mathcal{D})$, and
- a set of aggregation symbols $\text{agg}(\mathcal{D})$, where each aggregation symbol $\Sigma \in \text{agg}(\mathcal{D})$ is associated with an aggregation function $\Sigma^{\mathcal{D}} : \mathcal{M}(\text{dom}(\mathcal{D})) \rightarrow \text{dom}(\mathcal{D})$.

In [3], concrete domains are restricted to so-called *admissible* concrete domains in order to keep the reasoning problems of this logic decidable. We recall that, roughly spoken, a concrete domain \mathcal{D} is called *admissible* iff (1) $\text{pred}(\mathcal{D})$ is closed under negation and contains a unary predicate name $\top_{\mathcal{D}}$ for $\text{dom}(\mathcal{D})$, and (2) satisfiability of finite conjunctions over $\text{pred}(\mathcal{D})$ is decidable. For example, the subsets of some \mathbb{R}^n defined by a finite number of polynomial equations or inequalities as defined in [4] are admissible concrete domains.

In the following, we define our extension of the description logic $\mathcal{ALC}(\mathcal{D})$ as presented in [3]. $\mathcal{ALC}(\mathcal{D})$ itself is an extension of \mathcal{ALC} , introduced in [13], a well-known description logic with high expressive power. In \mathcal{ALC} , concepts can be built using boolean operators (i.e., and, or, not), and value restrictions on those individuals associated to an individual via a certain role (binary relation). These include *existential* restrictions like in $(\exists \text{ has_child.Girl})$ as well as *universal* restrictions like $(\forall \text{ has_child.Human})$. Additionally, in $\mathcal{ALC}(\mathcal{D})$, (abstract) individuals which are described using \mathcal{ALC} can now be

related to values in a *concrete domain*, like, for example, the integers or strings. This allows us to describe, for example, persons whose savings balance are higher than their yearly income, by $\text{Human} \sqcap (\text{savings} > \text{income})$. In [3], it is shown that all interesting inference problems like consistency, satisfiability, and containment (called subsumption in Description Logics) are decidable for $\mathcal{ALC}(\mathcal{D})$.

3.2 The Concept Language

In this subsection, we first introduce *complex concepts* in $\mathcal{ALC}(\mathcal{D})$. These concepts can then be used to specify the terminology of a specific application in a so-called TBox. This TBox can be viewed as an encyclopedia in which the meaning of certain concepts is defined using other concepts (which are possibly defined themselves in this TBox). Next, a specific situation can be described in a so-called ABox, possibly using the concepts defined in the TBox. In the ABox, we introduce some individuals, describe their properties and their interrelationships. Our extension of $\mathcal{ALC}(\mathcal{D})$ allows for operators on cubes. Thus, we can define a cube as the result of the application of such a cube-operator to another cube.

Definition 3 (Syntax of $\mathcal{ALC}(\mathcal{D})$ - concepts) *Let N_C , N_R , and N_F be disjoint sets of concept, role, and feature names, and let \mathcal{D} be an admissible concrete domain. Then each concept name is a concept, and complex concepts are defined inductively by the following rules (where C , D are concepts, R denotes a role name or feature name, $P \in \text{pred}(\mathcal{D})$ is a n -ary predicate name, u_1, \dots, u_n are feature chains²)*

$$C, D \longrightarrow C \sqcap D \mid C \sqcup D \mid \neg C \mid \\ \forall R.C \mid \exists R.C \mid P(u_1, \dots, u_n)$$

A terminology (or TBox) \mathcal{T} is a (finite) set of concept definitions, each of the form $A \doteq C$, where A is a concept name and C a complex concept. We restrict our attention to those terminologies where each concept name A occurs at most once on the left hand side of a concept definition, and which does not contain definitional cycles.

An ABox \mathcal{A} is a (finite) set of assertions. Given a set of individual names N_I , assertions are of the following forms:

$$a : C, (a, b) : R, (a, b) : f, (a, x) : f, (x_1, \dots, x_n) : P, \\ a = \text{rename}(b, f), a = \text{destroy}(b, f), a = \text{restrict}(b, C), \\ a = \text{join}(b, c), a = \text{aggr}(b, f, \Sigma, g), \\ a = \text{Join}(b, c, \langle n_1, o_1, f_1, g_1 \rangle, \dots, \langle n_m, o_m, f_m, g_m \rangle), \\ a = \text{roll-up}(b, f, \Sigma, g : \text{level})$$

for (possibly complex) concept C , feature names f, f_i, g, g_i, n_i , role name R , function names $o_i \in \text{funct}(\mathcal{D})$, an aggregation function $\Sigma \in \text{agg}(\mathcal{D})$, (abstract) individuals $a, b, c \in N_I$, an n -ary predicate name P , and elements of a concrete domain $x, x_1, \dots, x_n \in \text{dom}(\mathcal{D})$.

The semantics of these constructs is now defined in a set-theoretic way.

²A feature chain $u = f_1 \circ \dots \circ f_n$ is a composition of features.

Definition 4 (Semantics) Let \mathcal{D} be an admissible concrete domain. The semantics is then given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consists of an (abstract) interpretation domain $\Delta^{\mathcal{I}}$, and an interpretation function $\cdot^{\mathcal{I}}$. The domain $\Delta^{\mathcal{I}}$ and the given concrete domain $\text{dom}(\mathcal{D})$ have to be disjoint. The evaluation function $\cdot^{\mathcal{I}}$ associates each concept C with a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each role R with a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each feature name f with a partial function $f^{\mathcal{I}}$ from Δ into $\Delta^{\mathcal{I}} \cup \text{dom}(\mathcal{D})$. Additionally, \mathcal{I} has to satisfy

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$

$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$

the following equations:

$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y, (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$$

$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y, (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$

$$P(u_1, \dots, u_n)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid (u_1^{\mathcal{I}}(x), \dots, u_n^{\mathcal{I}}(x)) \in P^{\mathcal{D}}\}$$

A concept C is *satisfiable* iff there exists an interpretation \mathcal{I} such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model* of C . A concept C is *contained in* a concept D (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each interpretation \mathcal{I} . An interpretation \mathcal{I} *satisfies* a TBox \mathcal{T} iff \mathcal{I} satisfies each concept definition in \mathcal{T} , and it satisfies a concept definition $A \doteq C$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$. If $(a, b) \in R^{\mathcal{I}}$ or $f^{\mathcal{I}}(a) = b$, we say that b is an *R-filler* (resp. *f-filler*) of a . Interpretations of ABoxes, additionally, associate each individual name a to an element of $\Delta^{\mathcal{I}}$, in such a way that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ holds for two different individual names a, b . Again, \mathcal{I} satisfies an ABox \mathcal{A} iff \mathcal{I} satisfies each assertion in \mathcal{A} , that is to say

$$\begin{array}{ll} a^{\mathcal{I}} \in C^{\mathcal{I}} & \text{for each } a : C \in \mathcal{A} \\ (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} & \text{for each } (a, b) : R \in \mathcal{A} \\ f^{\mathcal{I}}(a^{\mathcal{I}}) = b^{\mathcal{I}} & \text{for each } (a, b) : f \in \mathcal{A} \\ f^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}} & \text{for each } (a, x) : f \in \mathcal{A} \\ (x_1^{\mathcal{I}}, \dots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}} & \text{for each } (x_1, \dots, x_n) : P \in \mathcal{A} \end{array}$$

The semantics of assertions containing operators like **destroy** or **join** is more difficult. Before giving it, we need some additional definitions. Furthermore, for the operator **roll-up**, we first develop a mean for representing multi-level hierarchically structured dimensions. An ABox \mathcal{A} is said to be *consistent* with a TBox \mathcal{T} iff there exists a model for both, that is an interpretation \mathcal{I} satisfying \mathcal{A} and \mathcal{T} .

Note that the description of individuals in an ABox needs not to be complete: A model of an ABox might have more elements than those explicitly mentioned, and the individuals mentioned in the ABox might have more properties than those explicitly stated in the ABox. This is due to the so-called *Open World Assumption*.

Using $\mathcal{ALC}(\mathcal{D})$, cubes containing information on exams results can be described by the following concept definition:

$$\text{examsresults} \doteq \forall \text{has_cell}. (\forall \text{student}. \text{STUDENT} \sqcap \forall \text{year}. \text{INTEGER} \sqcap \forall \text{subject}. \text{SUBJECT} \sqcap \forall \text{mark}. \text{INTEGER})$$

where **has_cell** is a role name and **student**, **year**, **subject**, and **mark** are feature names. **STUDENT**³ is the top predicate of the concrete domain representing the student hierarchy

³Which corresponds to All-students in the hierarchy.

as given in Figure 3, thus holding for every object within this hierarchy. Other properties like the age or the courses a student is enrolled in can also be associated to this cube. Our object-centered view of the information about exams results is depicted in Figure 4. The symbol \ni on the figure stands for the feature name `has_cell`. This figure shows (parts of) a model \mathcal{I} of `examsresults`, where $\Delta^{\mathcal{I}}$ contains, besides others, the individual `exam_results` (a cube which is an instance of the concept `examsresults`) and the cells $c1, \dots, c36$. Another concrete domain mentioned in this example, besides `STUDENT`, is `INTEGER`.

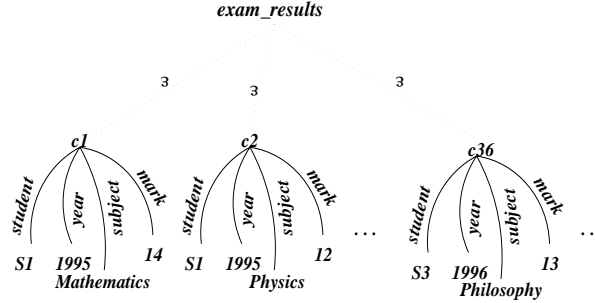


Figure 4: An object-centered representation of the exams results

3.3 Hierarchies within Dimensions

In this subsection, we propose a framework for representing multi-level hierarchically structured dimensions. As stated in Section 2, a hierarchically structured dimension is a set of objects interrelated by part-whole relations. For that the operators like `agg` and `destroy` can be defined correctly, we will not allow arbitrary decompositions along a dimension but will define certain restrictions. However, these restrictions are the only ones made—we want the user to be as free as possible in the definition of its domains.

A general assumption concerning part-whole relation is that there is a great variety of different part-whole relations [2, 14], all of them sub-relation of the general part-whole relation, denoted as \succ in the following. Though we will not distinguish these different part-whole relations explicitly, their presence is reflected in the fact that an object, say `All_students`, is decomposed in different ways, this is with respect to different part-whole relations. In the following, we will say that objects within a dimension are decomposed or aggregated within different *categories*.

Definition 5 (Hierarchies) *A hierarchy is a finite, partially ordered set (H, \succ) , where \succ is a strict ordering and a maximal $m \in H$ exists. Then, a level is a subset $\ell \subseteq H$ such that no two elements $x, y \in \ell$ are comparable by \succ . Furthermore, each element $h \in H$ has to belong to exactly one level. Finally, if for two levels ℓ, ℓ' we have $h \in \ell, i \in \ell'$ with $h \succ i$, then there exists no $h' \in \ell, i' \in \ell'$ with $h' \prec i'$.*

Hence in $\mathcal{ACC}(\mathcal{D} + \Sigma)$ ⁴, hierarchies are represented using a concrete domain for each dimension: A hierarchy-concrete domain $\mathcal{H} = (\text{dom}(\mathcal{H}), \text{pred}(\mathcal{H}))$ is a domain where

⁴For $\mathcal{ACC}(\mathcal{D})$ extended with operators on cubes and roll-up for navigating within the hierarchies of dimensions.

$(\text{dom}(\mathcal{H}), \succ)$ is a hierarchy as defined above, $\succ \in \text{pred}(\mathcal{H})$, and each level ℓ is associated with a unary predicate $\ell^{\mathcal{H}}$, where $\ell^{\mathcal{H}}(x)$ holds for all objects x belonging to the level ℓ .

As all hierarchy-concrete domains are finite, they are trivially admissible.

3.4 Semantics of Cube Operators

The operators for which we are going to define the semantics make sense only for “cubes”, which have been introduced only informally so far. In general, a cube is an object which is associated to cells which are all of similar form. This fact of being of the same form is described by the following concept **cube**. It is defined in such a way that, if a cell of an instance of **cube** has an f -filler for some feature name f , then all other cells have also f -fillers:

$$\text{cube} = \prod_{f \in \mathbb{N}_F} ((\exists \text{has_cell} . \exists f . \text{ALL}) \Rightarrow (\forall \text{has_cell} . \exists f . \text{ALL}))$$

where $C \Rightarrow D$ is used as shorthand for $\neg C \sqcup D$ and ALL is a shorthand for the “universal” concept $\top_{\mathcal{D}} \sqcup A \sqcup \neg A$ (A is a concept name). Given that—for each specific application— N_F is finite, **cube** is a concept and describes cubes according to our intuition. For the formal definition of the operators, further abbreviations are useful:

Definition 6 Let x, y be cubes in \mathcal{I} . Then $\text{cells}(x) := \{y \in \Delta^{\mathcal{I}} \mid (x, y) \in \text{has_cell}^{\mathcal{I}}\}$. We say x, y are disjoint if $x \neq y$ and they do not share cells, more formally $\text{disj}(x, y)$ iff $x \neq y$ and $\text{cells}(x) \cap \text{cells}(y) = \emptyset$. In the sequel, we will use R for a role name or a feature name, and $R^{\mathcal{I}}(a)$ as shorthand for all R -fillers of a , i.e. $R^{\mathcal{I}}(a) = \{b \in \Delta^{\mathcal{I}} \cup \text{dom}(\mathcal{D}) \mid (a, b) \in R^{\mathcal{I}}\}$.

Now we are ready to define the semantics of the operators. In general, so that \mathcal{I} satisfies an assertion of the form $x = \text{op}(y, \dots)$, there has to exist a mapping π from the cells of x into the cells of y . Further properties of the mapping π depend on the kind of operator and its parameters.

The first operator restricts the cells in a cube to those belonging to a given concept C . Thus it can be used in a straightforward and flexible way for slicing and dicing.

Semantics of the restrict operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{restrict}(y, C)$ iff $\text{disj}(x, y)$ and

$$\begin{aligned} \exists \pi : \text{cells}(y) \cap C^{\mathcal{I}} &\longrightarrow \text{cells}(x) \text{ that is bijective and} \\ \forall c \in \text{cells}(y) \cap C^{\mathcal{I}} \forall R &: R^{\mathcal{I}}(c) = R^{\mathcal{I}}(\pi(c)). \end{aligned}$$

The operator **destroy** removes the values of a feature f from all cells in a cube. An arbitrary feature f can be destroyed from a cube x even if not all cells in x have the same f -fillers. In this case, by applying **destroy**, (1) we really loose information and (2) might have different cells c_i whose remaining f_j -fillers coincide. (1) is fine since we might not be interested in these pieces of information. (2) does not matter either since “multiple occurrence” is not prohibited and aggregation still works in the expected way.

Semantics of the destroy operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{destroy}(y, f)$ iff $\text{disj}(x, y)$ and

$$\begin{aligned} \exists \pi : \text{cells}(y) &\longrightarrow \text{cells}(x) \text{ that is bijective and} \\ \forall c \in \text{cells}(y) : \pi(c) &\notin \text{dom}(f^{\mathcal{I}}) \text{ and} \\ \forall R \text{ if } R \neq f &\text{ then } R^{\mathcal{I}}(c) = R^{\mathcal{I}}(\pi(c)). \end{aligned}$$

where $\text{dom}(f^{\mathcal{I}})$ is the domain of $f^{\mathcal{I}}$, i.e. $\text{dom}(f^{\mathcal{I}}) = \{x \in \Delta^{\mathcal{I}} \mid f^{\mathcal{I}}(x) \text{ is defined}\}$.

The next operator, **join**, builds a new cube out of two cubes, quite similar to the join in the relational model.

Semantics of the join operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{join}(y, y')$ iff $\text{disj}(y, y')$, $\text{disj}(x, y)$, $\text{disj}(x, y')$, and

$$\begin{aligned} \exists \pi : \text{common}(y, y') &\longrightarrow \text{cells}(x) \text{ that is bijective and} \\ \forall (c, d) \in \text{common}(y, y') \forall R &: R^{\mathcal{I}}(c) \cup R^{\mathcal{I}}(d) = R^{\mathcal{I}}(\pi(c, d)). \end{aligned}$$

where **common** contains all pairs of cells of y and y' agreeing on all common features in y and y' , i.e., $\text{common}(y, y') := \{(c, d) \in \text{cells}(y) \times \text{cells}(y') \mid \forall f \in \mathbf{N}_F : c \notin \text{dom}(f^{\mathcal{I}}) \text{ or } d \notin \text{dom}(f^{\mathcal{I}}) \text{ or } f^{\mathcal{I}}(c) = f^{\mathcal{I}}(d)\}$.

Hence, for this join operator the features on which the cubes are joined are those which occur in both input cubes. If one wants to use **join** in a different way, one may use the operator **rename** on the input cubes to change the feature names:

Semantics of the rename operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{rename}(y, f, g)$ iff $\text{disj}(x, y)$ and

$$\exists \pi : \text{cells}(y) \longrightarrow \text{cells}(x) \text{ that is bijective and } \forall (c) \in \text{cells}(y) \forall R : R^{\mathcal{I}}(c) = R^+ [f \mapsto g]^{\mathcal{I}}(\pi(c)).$$

where $R^+[f \mapsto g] = R$ if $R \neq f$ and $R^+[f \mapsto g] = g$ if $R = f$.

The operator **Join** is more powerful than **join** in that it takes additional functions as parameters which are applied to the respective values of the cells before joining them.

Semantics of the Join operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{Join}(y, y', \langle n_1, o_1, f_1, g_1 \rangle, \dots, \langle n_m, o_m, f_m, g_m \rangle)$ iff $\text{disj}(y, y')$, $\text{disj}(x, y)$, $\text{disj}(x, y')$, and

$$\begin{aligned} \exists \pi : \text{common}(y, y') &\longrightarrow \text{cells}(x) \text{ that is bijective and } \forall (c, d) \in \text{common}(y, y') \\ &\forall 1 \leq i \leq m : n_i^{\mathcal{I}}(\pi(c, d)) = o_i(f_i^{\mathcal{I}}(c), g_i^{\mathcal{I}}(d)) \text{ and} \\ &\forall R : R \notin \{f_1, \dots, f_m, g_1, \dots, g_m\} \Rightarrow \\ &\quad R^{\mathcal{I}}(c) \cup R^{\mathcal{I}}(d) = R^{\mathcal{I}}(\pi(c, d)) \\ &\forall R : R \in \{f_1, \dots, f_m, g_1, \dots, g_m\} \Rightarrow \\ &\quad \pi(c, d) \notin \text{dom}(R^{\mathcal{I}}). \end{aligned}$$

The next operator is used to aggregate the values of the feature f over all those cells coinciding on all their role-fillers besides their g -fillers. Since this aggregation works over all g -fillers, the output cube has no values with respect to g .

Semantics of the aggr operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{aggr}(y, f, \Sigma, g)$ iff $\text{disj}(x, y)$ and

$$\begin{aligned} \exists \pi : \text{maxss}(y, f, g) &\longrightarrow \text{cells}(x) \text{ that is bijective and} \\ \forall c \in \text{cells}(x) : &c \notin \text{dom}(g^{\mathcal{I}}) \text{ and} \\ &f^{\mathcal{I}}(c) = \sum_{c' \in \pi^{-1}(c)} f^{\mathcal{I}}(c') \text{ and} \\ \forall R : &\text{if } R \notin \{f, g\} \text{ then} \\ &R^{\mathcal{I}}(c) = R^{\mathcal{I}}(c') \text{ for some } c' \in \pi^{-1}(c). \end{aligned}$$

where maxss contains all maximal subsets of cells of y which agree on all features but f and g , i.e.,

$$\text{maxss}(y, f, g) := \{S \subseteq \text{cells}(y) \mid \forall c, c' \in S \forall R \text{ if } R \notin \{f, g\} \text{ then } R^{\mathcal{I}}(c) = R^{\mathcal{I}}(c') \text{ and } \forall d \notin S \exists c \in S \exists R \notin \{f, g\} \text{ with } R^{\mathcal{I}}(d) \neq R^{\mathcal{I}}(c)\}.$$

The operator roll-up is more powerful than the last one: It aggregates over the g -fillers only up to a given level ℓ . This operation can be used to “jump” over several levels—hence we use \succ^+ as the transitive closure of the relation \succ .

Semantics of the roll-up operator:

An interpretation \mathcal{I} satisfies an assertion $x = \text{roll-up}(y, f, \Sigma, g : \ell)$ iff $\text{disj}(x, y)$ and

$$\begin{aligned} \exists \pi : \text{maxroll}(y, f, g : \ell) \longrightarrow \text{cells}(x) \quad & \text{that is bijective and} \\ \forall c \in \text{cells}(x) : g^{\mathcal{I}}(c) = d \text{ such that } \ell^{\mathcal{H}}(d) \text{ and} & \\ d \succ^+ g^{\mathcal{I}}(\pi^{-1}(c)) \text{ and} & \\ f^{\mathcal{I}}(c) = \Sigma f^{\mathcal{I}}(\pi^{-1}(c)) \text{ and} & \\ \forall R : \text{ if } R \notin \{f, g\} \text{ then} & \\ R^{\mathcal{I}}(c) = R^{\mathcal{I}}(c') \text{ for some } c' \in \pi^{-1}(c) & \end{aligned}$$

where maxroll contains all maximal subsets of cells of y which agree on all features but f and g , i.e.,

$$\begin{aligned} \text{maxroll}(y, f, g : \ell) := \{S \subseteq \text{cells}(y) \mid \forall c, c' \in S \forall R \text{ if } R \notin \{f, g\} \text{ then } R^{\mathcal{I}}(c) = R^{\mathcal{I}}(c') \text{ and} & \\ \exists d : \ell^{\mathcal{H}}(d) \text{ and } d \succ^+ f^{\mathcal{I}}(c) \text{ and } d \succ^+ f^{\mathcal{I}}(c') \text{ and} & \\ \forall e \notin S \exists c \in S \exists R \notin \{f, g\} \text{ with } R^{\mathcal{I}}(e) \neq R^{\mathcal{I}}(c) \text{ or} & \\ \exists d_1 : \ell^{\mathcal{H}}(d_1) \text{ and } d_1 \succ^+ f^{\mathcal{I}}(e) \text{ and } d_1 \neq d\}. & \end{aligned}$$

Note the following relationship between roll-up and aggr : $\text{destroy}(\text{roll-up}(y, f, \Sigma, g : \text{All}), g) = \text{aggr}(y, f, \Sigma, g)$.

Given this formal definition of hierarchies and the roll-up operator, this operator can easily be extended to take more than one level of a hierarchy as argument. Thus, for example, one can roll-up the *exams_result* cube on the student dimension in such a way that it displays the results for the male students in *City1*, for the female students in *City1*, for the male students in *City2*, etc.

3.5 Problems concerning drill-down

In multidimensional databases, besides slicing and dicing, roll-up and drill-down are the most famous operations used for the navigation in data cubes. In contrast to the first three operations, drill-down cannot be formulated within the framework presented here. This is so because the command “drill-down the dimension x of the cube C ” cannot be formulated in a precise way, in no formalism. In the following, we argue why drill-down can only be implemented as an “undo”-button or as a step from one to another precomputed cube—but not as a well-defined operation.

In general, drill-down operations are thought to take a given cube C and refine the granularity of dimension x to a level ℓ . The only case where no problems occur is when all dimensions but x are represented by their base-levels, and ℓ , too, is the base level of dimension x . In this case, the result of this operation is simply the information associated with the base data. If this is not the case, we are confronted with different problems:

If C is the result of roll-up operations, how to compute the new measure if a point p in dimension x is substituted by its parts? The measure could be evenly distributed over the parts of p but this would surely mislead the user. Thus, the more fine-grained cube has to be recomputed from the base data. In order to do so, one has to know the whole history of how the cube C has been computed. Then, one could think to recompute this history and just not do any aggregations on dimension x to a level above ℓ in order to obtain the desired new cube. Unfortunately, this will lead to unintuitive results, too: As the small example in Figure 5 shows, aggregation is not commutative. When starting from cube x and first aggregating the dimension $Space$ using \max , and the $Time$ using \min , what should be the result of a drill-down into the space dimension? If it were the cube x' , which is x aggregated on $Time$ using \min , then the application of \max to the dimension $Space$ yields a cube different from y . This simple example shows that, even if the whole history of a cube were used to compute a drill-down, and if there were no join operations applied, there is no unambiguous result to this operation—besides an immediate "undo".

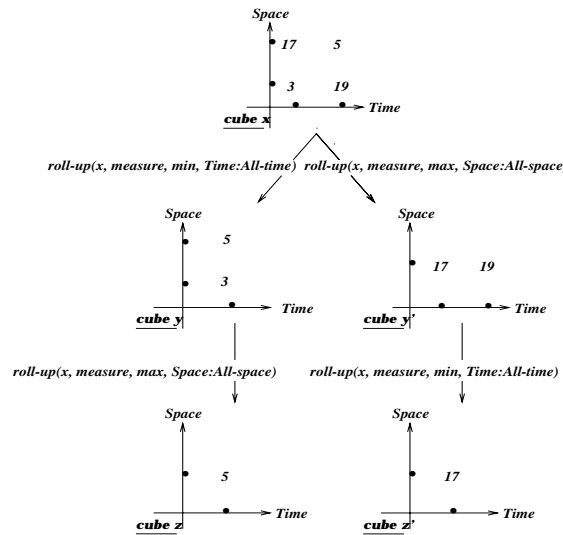


Figure 5: Aggregation is not commutative

4 Conclusion and Future Work

We presented an approach for modeling multidimensional databases. We provided in particular the syntax and the declarative semantics of an object-centered formalism. The description logic $\mathcal{ALC}(\mathcal{D})$ is extended by a set of operators that work on cubes, for example for aggregating information. Unlike [1] we incorporated in a clean way a mechanism for the declarative specification of hierarchically structured dimensions. Aggregation functions are not restricted to a fixed set, but we allow for arbitrary ones. A second point to be mentioned is that our approach explicitly supports hierarchically structured domains to be used in the roll-up operator. Furthermore, we investigated the problem of the drill-down operator, the counterpart to roll-up. Unfortunately, it seems as if this operator has to be limited to the trivial cases since aggregation is not commutative.

As the framework presented here is a first approach to an object-centered represen-

tation of multidimensional data models and the according operators, there remain some open questions and further work to be done: Until now, we have only defined the interesting reasoning problems like satisfiability, containment, consistency. This is a first step towards the investigation of the complexity of these problems and the design of suitable algorithms for them. Second, we have not yet incorporated a way to state functional dependencies between features. Perhaps this is not crucial for intensional reasoning, but it is crucial for extensional reasoning like consistency checking. Both points mentioned above will be part of future work.

References

- [1] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Proc. of ICDE'97*, Birmingham, UK, 1997.
- [2] A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-whole relations in object-centered systems: An overview. *Data & Knowledge Engineering*, 20:347–383, 1996.
- [3] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI'91. Sydney, Australia*, pages 452–457, 1991.
- [4] R. Benedetti and J.-J. Risler. *Real algebraic and semi-algebraic sets*. Hermann, editeurs des sciences et des arts, Paris, 1990.
- [5] R. J. Brachman and J. G. Schmolze. An overview of the KL-one knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
- [6] E. F. Codd, S. B. Codd, and C. T. Salley. Providing olap (on-line analytical processing) to user-analysts: An it mandate. White paper http://www.arborsoft.com/essbase/wht_ppr/coddTOC.html, 1993.
- [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 *Proc. of ICDE'96*, New Orleans, USA, pages 152–159, 1996.
- [8] M. Gyssens, L. V. S. Lakshmanan, and I. N. Subramanian. Tables as a paradigm for querying and restructuring. In *Proc. of PODS'96*, Montreal, PQ, Canada, June 1996.
- [9] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. of SIGMOD'96*, page 25, June 1996.
- [10] C.-T. Ho, R. Agrawal, and R. Srikant. Range-sum queries in data cubes. In *Proc. of SIGMOD'97*, USA, 1997.
- [11] A. Y. Levy and I. S. Mumick. Reasoning with aggregation constraints. In *Proc. EDBT'96*, Avignon, France, page 21, 1996.
- [12] K. A. Ross, D. Srivastava, P. J. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. In *Proc. PPCP'94*, LNCS 874, pages 193–204. Springer-Verlag, 1994.
- [13] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

- [14] A. C. Varzi. Parts, wholes, and part-whole relations: The prospects of mereotopology. *Data & Knowledge Engineering*, 20:259–286, 1996.