Multidimensional Data Modeling for Complex Data

Torben Bach Pedersen
Center for Health Information Services
Kommunedata, P. O. Pedersens Vej 2
DK-8200 Århus N, Denmark
email: tbp@kmd.dk

Christian S. Jensen
Department of Computer Science
Aalborg University, Fredrik Bajers Vej 7E,
DK–9220 Aalborg Øst, Denmark
email: csj@cs.auc.dk

Abstract

On-Line Analytical Processing (OLAP) systems considerably ease the process of analyzing business data and have become widely used in industry. Such systems primarily employ multidimensional data models to structure their data. However, current multidimensional data models fall short in their abilities to model the complex data found in some real-world application domains. The paper presents nine requirements to multidimensional data models, each of which is exemplified by a real-world, clinical case study. A survey of the existing models reveals that the requirements not currently met include support for many-to-many relationships between facts and dimensions, built-in support for handling change and time, and support for uncertainty as well as different levels of granularity in the data. The paper defines an extended multidimensional data model, and an associated algebra, which address all nine requirements.

1 Introduction

On-Line Analytical Processing (OLAP) [4] has attracted much interest in recent years, as business managers attempt to extract useful information from large databases in order to make informed management decisions. Reports indicate that traditional data models, such as the ER model and the relational model, do not provide good support for OLAP applications. As a result, new data models based on a *multidimensional* view of data have emerged. These multidimensional data models typically categorize data as being *measurable business facts* (measures) or *dimensions*, which are mostly textual and characterize the facts. For example, in a retail business, *products* are sold to *customers* at certain *times* in certain *amounts* at certain *prices*. A typical fact would be a *purchase*, with the amount and price as the measures, and the customer purchasing the product, the product being purchased, and the time of purchase as the dimensions. In OLAP research, most work has concentrated on performance issues; and higher-level issues such as conceptual modeling have received less attention. Several researchers have pointed to this lack in OLAP research, and it has been suggested to try to combine the traditional OLAP virtues of performance with the more advanced data model concepts from the field of *scientific and statistical databases* [8].

A data model for OLAP applications should have certain characteristics in order to support the complex data found in many real-world systems. We present nine advanced requirements that a multidimensional data model should satisfy and illustrate the requirements using a real-world case study from the clinical world. We present an extended multidimensional data model that addresses all nine requirements. The data model supports explicit hierarchies, multiple hierarchies, and non-strict hierarchies in dimensions. Dimensions and measures are treated symmetrically, and there is support for correct aggregation of data. The many-many relationships between facts and dimensions are captured directly, and data with different levels of granularity may be recorded. Finally, the model supports handling change over time and some aspects of uncertainty in the data. The model is equipped with an algebra that is closed and at least as strong as relational algebra with aggregation.

Eight previously proposed data models, which are representative for the spectrum of multidimensional data models, are evaluated against the nine requirements, and it is shown that no other model satisfies these requirements. Importantly, no other model supports many-to-many relationships between facts and dimensions, handling of uncertainty, and different levels of granularity at all, and no other model completely supports handling change and time or non-strict hierarchies.

The presentation is structured as follows. Section 2 presents a real-world case study, describes the nine requirements to multidimensional data models, and evaluates previously proposed models against the requirements. Sections 3 and 4 define the extended multidimensional data model and the associated algebra. Section 5 evaluates the model, summarizes, and points to future directions.

2 Motivation

This section presents a healthcare case study; the requirements that a data model should satisfy, examplified by the case study; and finally evaluates existing multidimensional data models according to the requirements.

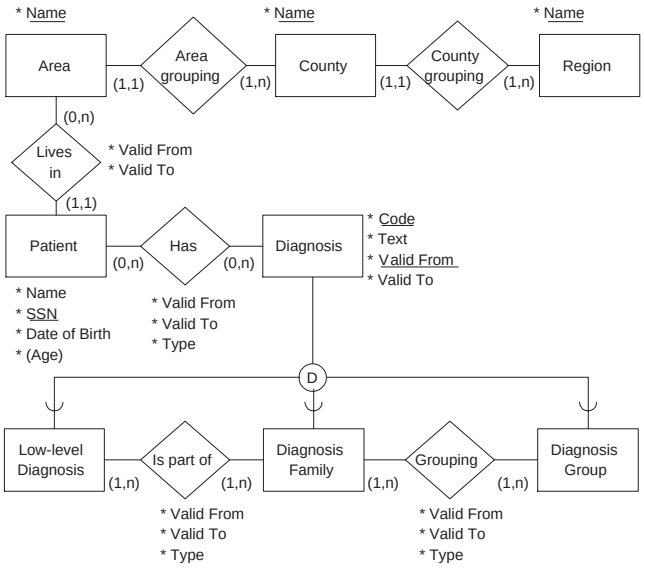

Figure 1. Patient Diagnosis Case Study

2.1 A Case Study

The case study concerns the patients in a hospital, their associated diagnoses, and their places of residence. The goal is to investigate whether some diagnoses occur more often in some areas than in others, in which case environmental or lifestyle factors might be contributing to the disease pattern. An ER diagram illustrating the case is seen in Figure 1.

The most important entities are the *patients*, for which we record Name, Social Security Number (SSN), Date of Birth, and Age (derived). Each patient can have one or more *diagnoses*. We record the time interval where a diagnosis is considered to be valid for a patient. We also record the *type of diagnosis*, to show whether a diagnosis is considered to be *primary* or *secondary*. A patient may have only one primary diagnosis at any one point in time. When registering a diagnosis of a patient, physicians often use different levels of granularity. Some will use the very precise diagnosis "Insulin dependent diabetes," while others will use the more imprecise diagnosis "Diabetes," which covers a wider range of patient conditions, corresponding to a number of more precise diagnoses. To model this, the relationship from patient to diagnoses is to the supertype "Diagnosis." The Diagnosis type has three subtypes, corresponding to different levels of granularity, the *low-level diagnosis*, the *diagnosis family*, and the *diagnosis group*. The higher-level diagnoses are both (imprecise) diagnoses in their own right, but also function as groups of lower-level diagnoses. A diagnosis family consists of 5–20 related low-level diagnoses. A diagnosis group consists of 5–20 diagnosis families. In the standard classification hierarchy, a lower-level item is part of exactly one item on the next level, making it a *strict, partitioning* hierarchy. However, to allow for more flexible grouping, a *user-defined* hierarchy is introduced, where a lower-level item can be a member of zero or more higher-level items, making it a *non-strict, non-partitioning* hierarchy.

ID	Name	SSN	Date of Birth
1	John Doe	12345678	25/05/69
2	Jane Doe	87654321	20/03/50

Patient Table

PatientID	DiagnosisID	ValidFrom	ValidTo	Type
1	9	01/01/89	NOW	Primary
2	3	23/03/75	24/12/75	Secondary
2	8	01/01/70	31/12/81	Primary
2	5	01/01/82	30/09/82	Secondary
2	9	01/01/82	NOW	Primary

Has Table

ID	Code	Text	ValidFrom	ValidTo
3	P11	Diabetes, pregnancy	01/01/70	31/12/79
4	O24	Diabetes, pregnancy	01/01/80	NOW
5	O24.0	Ins. dep. diab., pregn.	01/01/80	NOW
6	O24.1	Non ins. dep. diab., pregn.	01/01/80	NOW
7	P1	Other pregnancy diseases	01/01/70	31/12/79
8	D1	Diabetes	01/10/70	31/12/79
9	E10	Insulin dep. diabetes	01/01/80	NOW
10	E11	Non insulin dep. diabetes	01/01/80	NOW
11	E1	Diabetes	01/01/80	NOW
12	O2	Other pregnancy diseases	01/10/80	NOW

Diagnosis Table

ParentID	ChildID	ValidFrom	ValidTo	Type
4	5	01/01/80	NOW	WHO
4	6	01/01/80	NOW	WHO
7	3	01/01/70	31/12/79	WHO
8	3	01/01/70	31/12/79	User-defined
9	5	01/01/80	NOW	User-defined
10	6	01/01/80	NOW	User-defined
11	9	01/01/80	NOW	WHO
11	10	01/01/80	NOW	WHO
12	4	01/01/80	NOW	WHO

Grouping Table

Table 1. Data for the Case Study

For example, a low-level diagnosis can be part of several diagnosis families, e.g., the "Insulin dependent diabetes during pregnancy" diagnosis is part of both the "Diabetes during pregnancy" and the "Insulin dependent diabetes" family. Properties of the hierarchies will be discussed in more detail in Section 3.4. For diagnoses, we record an alphanumeric code and a descriptive text. The code and text are usually determined by a standard classification of diseases, e.g., the World Health Organization's International Classification of Diseases (ICD-10) [12], but we also allow user-defined diagnoses. As the diagnosis classification changes over time, we also record the time intervals where the diagnoses are "valid,"

i.e., can be used when diagnosing patients.

We also record the place of residence for the patients along with the period of residence to capture movement over time. We record the place of residence at the granularity of an *area*, which is part of exactly one *county*, which in turn is part of exactly one *region*, yielding a *strict, partitioning* hierarchy.

In order to list some example data, we assume a standard mapping of the ER diagram to relational tables, and we use surrogate keys, named *ID*, with globally unique values. Dates are written in the format dd/mm/yy. For the "Valid To" attribute, we use the special, continously-growing value "NOW" that denotes the current time [20]. As the three subtypes of the Diagnosis type do not have any attributes of their own, all three are mapped to a common Diagnosis table. The "is part of" and "grouping" relationships are also mapped to a common "Grouping" table. The data consists of two patients, four diagnoses made for the patients, and 10 diagnoses in a hierarchy. On January 1, 1980, a new, more detailed classification with a new coding scheme is introduced. The resulting tables are shown in Table 1 and will be used in examples throughout the paper.

2.2 Requirements for Data Analysis

This section describes the features that a data model should possess in order to fully support our sample case and other advanced uses. Current multidimensional models are evaluated against these features in the next section. The requirements are the following: 1) there should be *explicit hierarchies in dimensions* to aid the user in navigation, e.g., the hierarchy $area < county < region$ should be captured; 2) the *treatment of dimensions and measures should be symmetric*, e.g., the Age attribute could be used for average computations as well as defining age groups; 3) the model should support *multiple hierarchies in a dimension* to allow for different aggregation paths; e.g., with a time dimension on the Date of Birth attribute, days could roll up into weeks *or* months; 4) the model should support *correct aggregation* of data, closely related to *summarizability* [6, 7], so data is not doubly counted, and non-additive data is not added; e.g., when counting patients in different diagnosis groups, we should only count the same patient once per group, even though that patient may have several diagnoses; 5) *non-strict hierarchies* as found in many real-world situations, e.g., the user-defined diagnosis hierarchy, should be supported; 6) the oft-occuring *many-to-many relationships between facts and dimensions*, e.g., between patients and diagnoses, should be handled by the model; 7) the *change in data over time*, e.g., the changes in the diagnosis hierarchy, should be supported directly by the model; 8) the *uncertainty* often associated with data, e.g., a physician may be only 90% certain when diagnosing a patient, should also be handled directly by the model; 9) the model should allow data with *different levels of granularity*, e.g., the more or less precise diagnoses of patients, to be recorded.

2.3 Related Work

Next, we evaluate data models that have previously been proposed for data warehousing according to the requirements in the previous section. We consider the models of Rafanelli & Shoshani [6], Agrawal et al. [5], Gray et al. [2], Kimball [3], Li & Wang [10], Gyssens & Lakshmanan [9], Datta & Thomas [13], and Lehner [11]. The results of evaluating the eight data models against our nine requirements are seen in Table 2, where "√" denotes *full*, "p" denotes *partial*, and "-" denotes *no* support for a requirement. It can be seen that the models generally provide full or partial support for most of requirements 1–4. Requirement 5 (non-strict hierarchies) is partially supported by three of the models, while requirement 7 (handling change and time) is only partially supported by Kimball [3]. Requirements 6, 8, and 9 are not supported by any of the models. Further discussion of these issues may be found in the full paper [21]. The model proposed in this paper aims to support all nine requirements.

	1	2	3	4	5	6	7	8	9
Rafanelli [6]	√	-	-	√	p	-	-	-	-
Agrawal [5]	p	√	√	-	p	-	-	-	-
Gray [2]	-	√	√	p	-	-	-	-	-
Kimball [3]	-	-	√	p	-	-	p	-	-
Li [10]	p	-	√	p	-	-	-	-	-
Gyssens [9]	-	√	√	p	-	-	-	-	-
Datta [13]	-	√	√	-	p	-	-	-	-
Lehner [11]	√	-	-	√	-	-	-	-	-

Table 2. Evaluation of the Data Models

3 The Data Model

In this section we define our model. For every part of the model, we define the *intension*, the *extension*, and give an illustrating example. To avoid unnecessary complexity, we first define the basic model and then in turn define extensions for handling time and uncertainty.

3.1 The Basic Model

An *n-dimensional fact schema* is a two-tuple $\mathcal{S} = (\mathcal{F}, \mathcal{D})$, where $\mathcal{F}$ is a *fact type* and $\mathcal{D} = \{\mathcal{T}_i, i = 1, .., n\}$ is its corresponding *dimension types*.

Example 1 In the case study from Section 2.1 we will have *Patient* as the fact type, and *Diagnosis, Residence, Age, Date of Birth (DOB), Name*, and *Social Security Number (SSN)* as the dimension types. The intuition is that *everything* that characterizes the fact type is *dimensional*, even attributes that would be considered as *measures* in other models.

A dimension type $\mathcal{T}$ is a four-tuple $(\mathcal{C}, \leq_\mathcal{T}, \top_\mathcal{T}, \bot_\mathcal{T})$, where $\mathcal{C} = \{\mathcal{C}_j, j = 1, .., k\}$ are the *category types* of $\mathcal{T}$, $\leq_\mathcal{T}$ is a partial order on the $\mathcal{C}_j$'s, with $\top_\mathcal{T} \in \mathcal{C}$ and $\bot_\mathcal{T} \in \mathcal{C}$

being the top and bottom element of the ordering, respectively. Thus, the category types form a lattice. The intuition is that one category type is "greater than" another category type if members of the former's extension logically contain members of the latter's extension, i.e., they have a larger element size. The top element of the ordering corresponds to the largest possible element size, that is, there is only one element in its extension, logically containing all other elements.

We say that $\mathcal{C}_j$ *is a category type of* $\mathcal{T}$, written $\mathcal{C}_j \in \mathcal{T}$, if $\mathcal{C}_j \in \mathcal{C}$. We assume a function $Pred : \mathcal{C} \mapsto 2^{\mathcal{C}}$ that gives the set of immediate predecessors of a category type $\mathcal{C}_j$.

Example 2 Low-level diagnoses are contained in diagnosis families, which are contained in diagnosis groups. Thus, the *Diagnosis* dimension type has the following order on its category types: $\perp_{Diagnosis}$ = *Low-level Diagnosis* < *Diagnosis Familiy* < *Diagnosis Group* < $\top_{Diagnosis}$. We have that $Pred$(*Low-level Diagnosis*) = {*Diagnosis Family*}. Other examples of category types are *Age* and *Ten-year Age Group* from the Age dimension type, and *DOB* and *Year* from the DOB dimension type. Figure 2, to be discussed in detail later, illustrates the dimension types of the case study.

Many types of data, e.g., ages or sales amounts, can be added together to produce meaningful results. This data has an ordering on it, so computing the average, minimum, and maximum values make sense. For other types of data, e.g., dates of birth or inventory levels, the user may not find it meaningful in the given context to add them together. However, the data has an ordering on it, so taking the average, or computing the maximum or minimum values do make sense. Some types of data, e.g., diagnoses, do not have an ordering on them, and so it does not make sense to compute the average, etc. Instead, the only meaningful aggregation is to count the number of occurrences.

We can support correct aggregation of data by keeping track of what types of aggregate functions can be applied to what data. This information can then be used to either prevent users from doing "illegal" calculations on the data completely, or to warn the users that the result might be "wrong," e.g., the same patient is counted twice, etc. In line with this reasoning and previous work [11, 17], we distinguish between three types of aggregate functions: Σ, applicable to data that can be added together, ϕ, applicable to data that can be used for average calculations, and c, applicable to data that is constant, i.e., it can only be counted. Considering only the standard SQL aggregation functions, we have that Σ = {SUM, COUNT, AVG, MIN, MAX}, ϕ = {COUNT, AVG, MIN, MAX}, and c = {COUNT}. The aggregation types are ordered, $c \subset \phi \subset \Sigma$, so data with a higher aggregation type, e.g., Σ, also possess the characteristics of the lower aggregation types. For each dimension type $\mathcal{T} = (\mathcal{C}, \leq_{\mathcal{T}})$, we assume a function $Aggtype_{\mathcal{T}} : \mathcal{C} \mapsto \{\Sigma, \phi, c\}$ that gives the aggregation type for each category type.

Example 3 In the case study, we have *Aggtype(Low-level Diagnosis)* = *c*, *Aggtype(Age)* = Σ, and *Aggtype(DOB)* = ϕ.

A *dimension* D of type $\mathcal{T} = (\{\mathcal{C}_j\}, \leq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$ is a two-tuple $D = (C, \leq)$, where $C = \{C_j\}$ is a set of *categories* C_j such that $Type(C_j) = \mathcal{C}_j$ and $\leq$ is a partial order on $\cup_j C_j$, the union of all dimension values in the individual categories. A category C_j of type $\mathcal{C}_j$ is a set of *dimension values* e such that $Type(e) = \mathcal{C}_j$. The definition of the partial order is: given two values e_1, e_2 then $e_1 \leq e_2$ if e_1 is logically contained in e_2. We say that C_j is a category of D, written $C_j \in D$, if $C_j \in C$. For a dimension value e, we say that e is a dimensional value of D, written $e \in D$, if $e \in \cup_j C_j$. The category type $\perp_{\mathcal{T}}$ in dimension type $\mathcal{T}$ contains the values with the smallest value size. The category type with the largest value size, $\top_{\mathcal{T}}$, contains exactly one value, denoted $\top$. For all values e of the category types of D, $e \leq \top$. Value $\top$ is similar to the *ALL* construct of Gray et al. [2].

Example 4 In our *Diagnosis* dimension we have the following categories, named by their type. *Low-level Diagnosis* = $\{3, 5, 6\}$, *Diagnosis Family* = $\{4, 7, 8, 9, 10\}$, *Diagnosis Group* = $\{11, 12\}$, and $\top_{Diagnosis} = \{\top\}$. The values in the sets refer to the *ID* column in the Diagnosis table of Table 1. The partial order $\leq$ is given by the first two columns in the Grouping table in Table 1. Additionally, the top value $\top$ logically contains all the other diagnosis values.

We say that the dimension $D' = (C', \leq')$ is a *subdimension* of the dimension $D = (C, \leq)$ if $C' \subseteq C$ and $e_1 \leq' e_2 \Leftrightarrow \exists C_1, C_2 \in C'(e_1 \in C_1 \wedge e_2 \in C_2 \wedge e_1 \leq e_2)$, that is, D' has a subset of the categories of D and $\leq'$ is the restriction of $\leq$ to these categories.

Example 5 We obtain a subdimension of the Diagnosis dimension from the previous example by removing the *Low-level Diagnosis* and *Diagnosis Family* categories, retaining only *Diagnosis Group* and $\top_{Diagnosis}$.

It is desirable to distinguish between the dimension values in themselves and the real-world "names" that we use for them. The names might change or the same value might have more than one name, making the name a bad choice for identifying an value. In common database terms, this is the argument for *object ids* or *surrogates*. To support this feature, we require that a category C has one or more *representations*. A representation Rep is a bijective function $Rep : Dom(C) \leftrightarrow Dom_{Rep}$, i.e., a value of a representation uniquely identifies a single value of a category and vice versa, thus making the representation an "alternate key." We use the notation $Rep(e) = v$ to denote the mapping from dimension values to representation values.

Example 6 A diagnosis value has two representations, *Code* and *Text*. Using the ID's from the Diagnosis table to identify the values, we have *Code*(3) = "O24" and *Text*(3) = "Diabetes during pregnancy."

Let F be a set of facts and $D = (\{C_j\}, \leq)$ a dimension. A *fact-dimension relation* between F and D is a set $R = \{(f, e)\}$, where $f \in F$ and $e \in \cup_j C_j$. Thus R links facts to dimension values. We say that fact f is *characterized by* dimension value e, written $f \rightsquigarrow e$, if $\exists e_1 \in D\ ((f, e_1) \in R \wedge e_1 \leq e)$. We require that $\forall f \in F\ (\exists e \in \cup_j C_j\ ((f, e) \in R))$; thus we do not allow missing values. The reasons for disallowing missing values are that they complicate the model and often have an unclear meaning. If it is unknown which dimension value a fact f is characterized by, we add the pair $(f, \top)$ to R, thus indicating that we cannot characterize f within the particular dimension.

Example 7 The fact-dimension relation R links patient facts to diagnosis dimension values as given by the Has table in Table 1. Leaving out the temporal aspects for now, we get that $R = \{(1,9), (2,3), (2,5), (2,8), (2,9)\}$. Note that we can relate facts to values in higher-level categories, e.g., fact 1 is related to diagnosis 9, which belongs to the *Diagnosis Family* category. Thus, we do not require that e belongs to $\perp_{Diagnosis}$, as do the existing data models. If no diagnosis is known for patient 1, we would have added the pair $(1, \top)$ to R.

A *multidimensional object* (MO) is a four-tuple $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (\mathcal{F}, \mathcal{D} = \{\mathcal{T}_i\})$ is the schema, $F = \{f\}$ is a set of *facts* f where $Type(f) = \mathcal{F}$, $D = \{D_i, i = 1, .., n\}$ is a set of *dimensions* where $Type(D_i) = \mathcal{T}_i$, and $R = \{R_i, i = 1, .., n\}$ is a set of fact-dimension relations, such that $\forall i((f, e) \in R_i \Rightarrow f \in F \wedge \exists C_j \in D_i(e \in C_j))$.

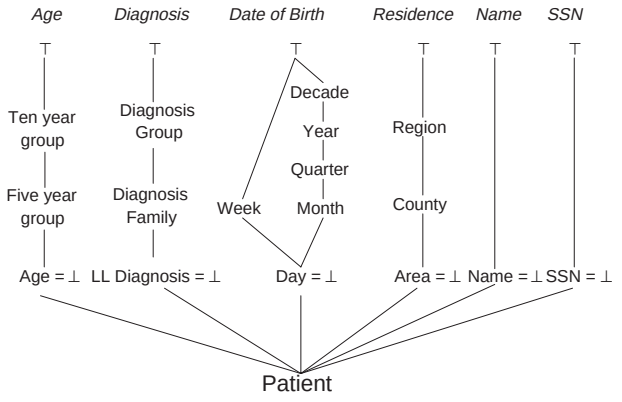

Figure 2. Schema of the Case Study

Example 8 For the case study, we get a six-dimensional MO $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (\textit{Patient}, \{\textit{Diagnosis, DOB, Residence, Name, SSN, Age}\})$ and $F = \{1, 2\}$. The definition of the diagnosis dimension and its corresponding fact-dimension relation were given in the previous examples. Due to space constraints, we do not list the contents of the other dimensions and fact-dimension relations, but just outline their structure. The Name and SSN dimensions are simple, i.e., they just have a $\perp$ category type, Name respectively SSN, and a $\top$ category type. The Age dimension groups ages (in years) into five-year and ten-year groups, e.g., 10–14 and 10–19. The Date-of-Birth dimension has two hierarchies in it: days are grouped into weeks, or days are grouped into months, with the further levels of quarters, years, and decades. We will refer to this MO as the "Patient" MO. A graphical illustration of the schema of the "Patient" MO is seen in Figure 2.

A collection of multidimensional objects, possibly with shared subdimensions, is called a *multidimensional object family*. The shared subdimensions can be used to "join" data from separate MO's.

To summarize our model, the facts are objects with a *separate identity*. Thus, we can test facts for equality, but we do not assume an ordering on the facts. The combination of dimensions values that characterize the facts of a fact set is *not* a "key" for the fact set. Thus, we may have "duplicate values," in the sense that several facts may be characterized by the same combination of dimension values. But, the facts of an MO are a *set*, so we do not have duplicate *facts* in an MO.

3.2 Handling Time

We proceed to build temporal support into the model. Consistently with the vast majority of temporal data models [14] and the SQL standard [15], we assume a time domain that is discrete and bounded, i.e., isomorphic with a bounded subset of the natural numbers. The values of the time domain are called *chronons*. They correspond to the finest granularity in the time domain [19]. We let T, possibly subscripted, denote a set of chronons.

The *valid time* of a statement is the time when the statement is true in the modeled reality [1]. Valid time is very important to capture because the real world is where the users reside, and we *allow* the attachment of valid time to the data, but do not require it. If valid time is not attached to the data, we assume the data to be *always* valid. If valid time is attached to an MO, we call it a *valid-time* MO. In general, valid time may be assigned to anything that has a truth value. In our model, this is the partial order between dimension values, the mapping between values and representations, the fact-dimension relations, and the membership of values in categories. It is important to be able to capture all these aspects.

We add valid time to the dimension partial order $\leq$ by adding T_v, the set of chronons during which the relation holds, to each relation between two values. We write that $e_1 \leq_{T_v} e_2$ if $e_1 \leq e_2$ during each chronon in T_v. The partial order $\leq_{T_v}$ has the following property: $e_1 \leq_{T_{1_v}} e_2 \wedge e_2 \leq_{T_{2_v}} e_3 \Rightarrow e_1 \leq_{T_{1_v} \cap T_{2_v}} e_3$. Similarly, we write $Rep(e) =_{T_v} v$ to denote that the representation Rep of the value e has value v during each chronon in T_v. For each fact-dimension relation between a fact f and a dimension value e, we capture the set of chronons T_v when the two are related. We write $(f, e) \in_{T_v} R$ when $(f, e) \in R$ during each chronon in T_v. We use the notation $f \rightsquigarrow_{T_v} e$ when $(f, e') \in_{T_v} R \wedge e' \leq_{T_v} e$.

Finally, we add valid time to the membership of dimension values in categories, writing $e \in_{T_v} C$ when $e \in C$ during each chronon in T_v.

The set of chronons that is attached to a piece of data is the *maximal* set of chronons when the data is valid, so the data is always "coalesced" [1]. Thus, we do not have the problem of "value-equivalent" data [1, 18], where the same data appears several times with different times attached to it, e.g., $e_1 \leq_{T_1} e_2$ and $e_1 \leq_{T_2} e_2$, where $T_1 \neq T_2$. However, by implication, data is valid for any subset of its attached time, e.g., $T_1 \subseteq T_2 \wedge e_1 \leq_{T_2} e_2 \Rightarrow e_1 \leq_{T_1} e_2$.

Example 9 In examples, we use interval notation for T_v, with a chronon size of Day. For the fact-dimension relation, we have $(2, 3) \in_{[23/03/75-24/12/75]} R$. For the category membership, we have $10 \in_{[01/01/80-NOW]}$ *Diagnosis Family*. For the partial order on the Diagnosis dimension, we have $7 \leq_{[01/01/70-31/12/79]} 3$. For the representation, we have $Code(8) =_{[01/01/70-31/12/79]}$ D1.

To sum up, by extending the dimension partial order with links between dimension values that represent the "same" thing across change, we have a foundation for handling analysis across changes. This allows us to obtain meaningful results when we analyze data across changes in a dimension.

Example 10 When looking at the data from the current point in time, we want to count the patients diagnosed with the old "Diabetes" diagnosis $(ID = 8)$ together with those diagnoses with the new "Diabetes" diagnosis $(ID = 11)$ when we look at diagnoses made from 1970 to the present. This is done by defining that $8 \leq_{[01/01/80-NOW]} 11$, i.e., from 1980 up till now, we consider diagnosis 8 to be logically contained in diagnosis 11.

In addition to valid time, it is also interesting to capture when statements are present in the database, as the time a statement is present in the database almost never corresponds to the time it is true in the real world. We need to know when data are present in the database for accountability and traceability purposes.

The *transaction time* of a statement is the time when the statement is current in the database and may be retrieved [1]. Generally, transaction time can be attached to anything that valid time can be attached to. The addition of transaction time is orthogonal to the addition of valid time. Additionally, transaction time can be added to data that does not have a truth value. In our model, we could record when facts, e.g., patients, are present in the database. We do not think that this is very interesting in itself, as facts are only interesting when they participate in fact-dimension relations. Thus, we do not record this. If transaction time is attached to an MO, we call it a *transaction-time* MO. If both valid and transaction time is attached to an MO, we call it a *bitemporal* MO. If no time is attached to an MO, we call it a *snapshot* MO. In our notation, we use T_t to denote the set of chronons when data is current in the database. We use $T_t \times T_v$ to denote sets of bitemporal chronons.

3.3 Handling Uncertainty

Uncertainty in the data can also be handled in the model. The basic idea is to add probabilities p to the parts of the model where it makes sense. This is for the partial order on dimension values and for the fact-dimension relations, with the notations $e_1 \leq_p e_2$ and $(f, e) \in_p R_i$, respectively. The probabilities are also handled by the algebra. Due to space constraints, we will not give a detailed description of the approach here, but refer to the full paper [21].

3.4 Properties of the Model

The model has several important properties that relate to the use of pre-computed aggregates. The first important concept is *summarizability*, which intuitively means that one set of aggregate results can be combined directly to produce other, higher-level aggregate results.

Definition 1 Given a type T, a set $S = \{S_j, j = 1, .., k\}$, where $S_j \in 2^T$, and an aggregate function $g : 2^T \mapsto T$, we say that g is *summarizable* for S if $g(\{g(S_1), .., g(S_k)\}) = g(S_1 \cup .. \cup S_k)$. The set of arguments on the left side of the equation is a multi-set, or bag, i.e., the same result value can occur multiple times.

Summarizability is an important concept as it is a condition for the flexible use of pre-computed aggregates. Without summarizability, lower-level results generally cannot be directly combined into higher-level results. This means that we cannot choose to pre-compute only a relevant selection of the possible aggregates and then use these to compute higher-level aggregates on-the-fly. Instead, we have to pre-compute the total results for all the aggregations that we need fast answers to, while other aggregates must be computed from the base data. It has been shown that summarizability is equivalent to the aggregation function being *distributive*, all paths being *strict*, and the hierarchies being *partitioning* in the relevant dimensions [7]. If data with time attached to it is aggregated such that data for one fact is only counted for one point in time, this result extends to hierarchies that are *snapshot strict* and *snapshot partitioning*. These concepts are formally defined below. In the definitions, we assume a dimension $D = (C, \leq)$.

Definition 2 If $\forall C_1, C_2 \in C(e_1, e_3 \in C_1 \wedge e_2 \in C_2 \wedge e_2 \leq e_1 \wedge e_2 \leq e_3 \Rightarrow e_1 = e_3)$ then the mapping between C_1 and C_2 is *strict*. Otherwise, it is *non-strict*. The hierarchy in dimension D is *strict* if all mappings in it are strict; otherwise, it is *non-strict*. Given a category $C_j \in D_i$, we say that there is a *strict path* from the set of facts F to C_j iff $\forall f \in F(f \rightsquigarrow$

$e_1 \wedge f \rightsquigarrow e_2 \wedge e_1 \in C_j \wedge e_2 \in C_j \Rightarrow e_1 = e_2)$[1]. The hierarchy in dimension D is *snapshot strict*, if at any given time *t*, the hierarchy is strict.

Definition 3 If $\forall C_1 \in C(C_1 \neq \top_D \wedge e_1 \in C_1 \Rightarrow \exists C_2 \in Pred(C_1)(\exists e_2 \in C_2(e_1 < e_2)))$, i.e., if every non-top value has a direct parent, we say that the hierarchy in dimension D is *partitioning*; otherwise, it is *non-partitioning*. The hierarchy in dimension D is *snapshot partitioning* if at any given time *t*, the hierarchy is partitioning.

Example 11 The hierarchy in the Residence dimension is strict and partitioning. The hierarchy in the Diagnosis dimension is non-strict and partitioning, but could have been non-partitioning. The sub-hierarchy of the Diagnosis dimension obtained by restriction to the standard classification is snapshot strict and snapshot partitioning.

4 The Algebra

This section defines an algebra on the multidimensional objects just defined. In line with the model definition, we first define the basic algebra and then define the extension for handling time. The extension to uncertainty is described in the full paper [21].

4.1 Fundamental Operators

The fundamental operators are close to the standard relational algebra operators. For unary resp. binary operators, we assume a multidimensional object $M = (\mathcal{S}, F, D = \{D_i\}, R = \{R_i\}), i = 1,..,n$ with schema $\mathcal{S} = (\mathcal{F}, \mathcal{D})$ and multidimensional objects $M_k = (\mathcal{S}_k, F_k, D_k = \{D_{k i_k}\}, R_k = \{R_{k i_k}\}), k = 1,2$. The representations of the categories in the resulting MO's are the same as in the argument MO's; thus we do not specify the representations for the resulting MO's. The aggregation types are only changed by the aggregate formation operator, so they are not specified for the other operators.

For the operator definitions, we need some auxilary functions. First, we define *Group* that groups the facts characterized by the same dimension values together. Given an n-dimensional MO, $M = (\mathcal{S}, F, D = \{D_i\}, R = \{R_i\}), i = 1,..,n$, a set of categories $C = \{C_i \mid C_i \in D_i\}, i = 1,..,n$, one from each of the dimensions of M, and an n-tuple $(e_1,..,e_n)$, where $e_i \in C_i, i = 1,..,n$, we define *Group* as: $Group(e_1,..,e_n) = \{f \mid f \in F \wedge f \rightsquigarrow_1 e_1 \wedge .. \wedge f \rightsquigarrow_n e_n\}$.

Next, we define a *union* operator on dimensions, which performs union on the categories and the partial orders. Given two dimensions $D_1 = (C_1, \leq_1)$ and $D_2 = (C_2, \leq_2)$ of type $\mathcal{T}$, where $C_k = \{C_{kj}\}, k = 1,2,\ j = 1,..,m$, we define the union operator on dimensions, $\bigcup_D$, as: $D_1 \bigcup_D D_2 = (C', \leq')$, where $C' = \{C'_j\}, j = 1,..,m, C'_j = C_{1j} \cup C_{2j}$, where $\cup$ denotes set union, and $e_1 \leq' e_2 \Leftrightarrow e_1 \leq_1 e_2 \vee e_1 \leq_2 e_2$.

[1]Note that the paths from the set of facts F to the $\top_{\mathcal{T}}$ categories are always strict.

selection: Given a predicate p on the dimension types $\mathcal{D} = \{\mathcal{T}_i\}$, we define the selection σ as: $\sigma[p](M) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = \mathcal{S}$, $F' = \{f \in F \mid \exists e_1 \in D_1,..,e_n \in D_n\ (p(e_1,..,e_n) \wedge f \rightsquigarrow_1 e_1 \wedge .. \wedge f \rightsquigarrow_n e_n)\}$, $D' = D$, $R' = \{R'_i\}$, and $R'_i = \{(f', e) \in R_i \mid f' \in F'\}$. Thus, we restrict the set of facts to those that are characterized by values where p evaluates to true. The fact-dimension relations are restricted accordingly, while the dimensions and the schema stay the same.

projection: Without loss of generality, we assume that the projection is over the k dimensions $D_1,..,D_k$. We then define projection π as: $\pi[D_1,..,D_k](M) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = \mathcal{F}$, $\mathcal{D}' = \{\mathcal{T}_1,..,\mathcal{T}_k\}$, $F' = F$, $D' = \{D_1,..,D_k\}$, and $R' = \{R_1,..,R_k\}$. Thus, we retain only the k specified dimensions, but the set of facts stays the same. Note that we do not remove "duplicate values." Thus the same combination of dimension values may be associated with several facts.

rename: Given an MO, $M = (\mathcal{S}, F, D, R)$, and fact schema $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, such that $\mathcal{D}$ is isomorphic with $\mathcal{D}'$, we define the rename ρ as: $\rho[\mathcal{S}'](M) = M'$, where $M' = (\mathcal{S}', F, D, R)$. We see that rename just return the contents of M with the new schema $\mathcal{S}'$, which has the same structure as the old schema $\mathcal{S}$. Rename is used to alter the names of dimensions so that dimensions with the same name, e.g., resulting from a "self-join," can be distinguished.

union: Given two n-dimensional MO's, $M_k = (\mathcal{S}_k, F_k, D_k, R_k), k = 1,2$, such that $\mathcal{S}_1 = \mathcal{S}_2$, we define the union $\bigcup$ as: $M_1 \bigcup M_2 = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = S_1$, $F' = F_1 \cup F_2$, $D' = \{D_{1_i} \bigcup_D D_{2_i}, i = 1,..,n\}$, and $R' = \{R_{1_i} \cup R_{2_i}, i = 1,..,n\}$. In words, given two MO's with common schemas, we take the set union of the facts and the fact-dimension relations. The $\bigcup_D$ operator is used to combine the dimensions.

difference: Given two n-dimensional MO's, $M_k = (\mathcal{S}_k, F_k, D_k, R_k), k = 1,2$, such that $\mathcal{S}_1 = \mathcal{S}_2$, we define the difference $\setminus$ as: $M_1 \setminus M_2 = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = S_1$, $F' = F_1 \setminus F_2$, $D' = D_1$, $R' = \{R'_i, i = 1,..,n\}$, with $R'_i = \{(f', e) \mid f' \in F' \wedge (f', e) \in R_{1_i}$. Thus, given two MO's with common schemas, we take the set difference of the facts, the dimensions of the first argument MO are retained, and the fact-dimension relations are restricted to the new fact set. Note that we do not take the set difference of the dimensions, as this does not make sense.

identity-based join: Given two MO's, M_1 and M_2, and a predicate $p(f_1, f_2) \in \{f_1 = f_2, f_1 \neq f_2, true\}$, we define the identity-based join $\bowtie$ as: $M_1 \bowtie_{[p]} M_2 = (\mathcal{S}', F', D', R')$, where $(\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = \mathcal{F}_1 \times \mathcal{F}_2$, $\mathcal{D}' = \mathcal{D}_1 \cup \mathcal{D}_2$, $F' = \{(f_1, f_2) \mid f_1 \in F_1 \wedge f_2 \in F_2 \wedge p(f_1, f_2)\}$, $D' = D_1 \cup D_2$, $R' = \{R'_i, i = 1,.., n_1 + n_2\}$, and $R'_i = \{(f', e) | f' = (f_1, f_2) \wedge f' \in F' \wedge ((i \leq n_1 \wedge (f_1, e) \in R_{1_i}) \vee (i > n_1 \wedge (f_2, e) \in R_{2_{i-n_1}}))\}$. The $\bowtie$ operator is used to combine

information from several MO's. It works as follows. The new fact type is the type of *pairs* of the old fact types, and the new set of dimension types is the union of the old sets. The set of facts is the subset of the cross product of the old sets of facts where the join predicate p holds. For p equal to $f_1 = f_2$, $f_1 \neq f_2$, and $true$, the operation is an *equi-join*, *non-equi-join*, and *Cartesian product*, respectively. For the instance, the set of dimensions is the set union of the old sets of dimensions, and the fact-dimension relations relate a pair to a value if one member of the pair was related to that value before.

aggregate formation: The aggregate formation operator is used to compute aggregate functions on the MO's. For notational convenience and following Klug [16], we assume the existence of a *family* of aggregation functions g that take some k-dimensional subset $\{D_{i_1}, .., D_{i_k}\}$ of the n dimensions as arguments, e.g., SUM_i sums the i'th dimension and SUM_{ij} sums the i'th and j'th dimensions. We assume a function $Args(g) = \{j \mid g \text{ uses dimension } j \text{ as argument}\}$ that returns the argument dimensions of g.

Given an n-dimensional MO, M, a dimension D_{n+1} with type $\mathcal{T}_{n+1}$, a function[2], $g : 2^F \mapsto D_{n+1}$ such that $g \in \min_{j \in Args(g)}(Aggtype(\perp_{D_{i_j}}))$, and a set of categories $C_i \in D_i, i = 1, .., n$, we define aggregate formation operator α as: $\alpha[D_{n+1}, g, C_1, .., C_n](M) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = 2^{\mathcal{F}}$, $\mathcal{D}' = \{\mathcal{T}_i', i = 1, .., n\} \cup \{\mathcal{T}_{n+1}\}$, $\mathcal{T}_i' = (\mathcal{C}_i', \leq_{\mathcal{T}_i}', \perp_{\mathcal{T}_i}', \top_{\mathcal{T}_i}')$, $\mathcal{C}_i' = \{\mathcal{C}_{ij} \in \mathcal{T}_i \mid Type(C_i) \leq_{\mathcal{T}_i} \mathcal{C}_{ij}\}$, $\leq_{\mathcal{T}_i}' = \leq_{\mathcal{T}_i|_{\mathcal{C}_i'}}$, $\perp_{\mathcal{T}_i}' = Type(C_i)$, $\top_{\mathcal{T}_i}' = \top_{\mathcal{T}_i}$, $F' = \{Group(e_1, .., e_n) \mid (e_1, .., e_n) \in C_1 \times .. \times C_n \wedge Group(e_1, .., e_n) \neq \emptyset\}$, $D' = \{D_i', i = 1, .., n\} \cup \{D_{n+1}\}$, $D_i' = (C_i', \leq_i')$, $C_i' = \{C_{ij}' \in D_i \mid Type(C_{ij}') \in \mathcal{C}_i'\}$, $\leq_i' = \leq_{i|_{D_i'}}$, $R' = \{R_i', i = 1, .., n\} \cup \{R_{n+1}'\}$, $R_i' = \{(f', e_i') \mid \exists (e_1, .., e_n) \in C_1 \times .. \times C_n\ (f' = Group(e_1, .., e_n) \wedge f' \in F' \wedge e_i = e_i')\}$, and $R_{n+1}' = \cup_{(e_1,..,e_n) \in C_1 \times .. \times C_n} \{(Group(e_1, .., e_n), g(Group(e_1, .., e_n))) \mid Group(e_1, .., e_n) \neq \emptyset\}$. The aggregation types for the remaining parts of the argument dimensions are not changed. The aggregation types for the result dimension is given by the following rule. If g is distributive, the paths to $C_1, .., C_n$ are strict, and the hierarchies up to $C_1, .., C_n$ are partitioning, then $Aggtype(\perp_{D_{n+1}}) = \min_{j \in Args(g)}(Aggtype(\perp_{D_j}))$. Otherwise, $Aggtype(\perp_{D_{n+1}}) = c$. For the higher categories in the result dimension, $Aggtype(C_m') = \min(\{Aggtype(C_m), Aggtype(\perp_{D_{n+1}})\})$.

Thus, for every combination $(e_1, .., e_n)$ of dimension values in the given "grouping" categories, we apply g to the set of facts $\{f\}$, where the f's are characterized by $(e_1, .., e_n)$, and place the result in the new dimension D_{n+1}. The facts are of type *sets* of the argument fact type, and the argument dimension types are restricted to the category types that are greater than or equal to the types of the given "grouping" categories. The dimension type for the result is added to the set of dimension types. The new set of facts consists of sets of facts, where the facts in a set share a combination of characterizing dimension values. The argument dimensions are restricted to the remaining category types, and the result dimension is added.

The fact-dimension relations for the argument dimensions now link sets of facts directly to their corresponding combination of dimension values, and the fact-dimension relation for the result dimension links sets of facts to the function results for these sets. If the function g is distributive, the paths up to the grouping categories are strict, and the hierarchy up to the grouping categories is partitioning, i.e., g is "summarizable," then the aggregation type for the bottom category in the result dimension is the minimum of the aggregation types for the bottom categories in the dimensions that g uses as arguments; otherwise, the aggregation type is c. For the higher categories, the minimum of the aggregation types given in the result dimension and the bottom category's aggregation type is used. Thus, aggregate results that are "unsafe" in the sense that they contain overlapping data cannot be used for further aggregation. This prevents the user from getting incorrect results by accidentally "double-counting" data.

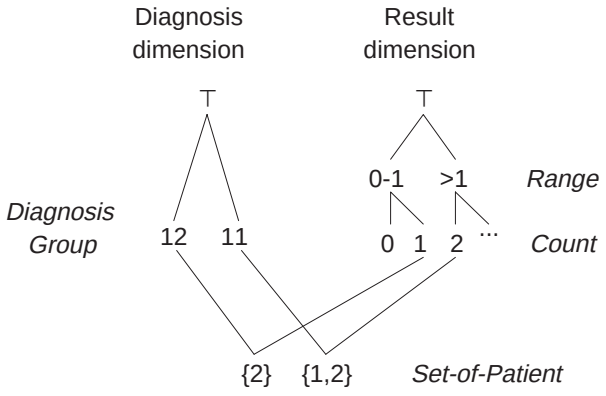

Figure 3. Result MO for Aggregate Formation

Example 12 We want to know the number of patients in each diagnosis group. To do so, we apply the aggregate-formation operator to the "Patient" MO with the *Diagnosis Group* category and the $\top$ categories from the other dimensions. The aggregate function g to be used is *set-count*, which counts the number of members in a set. The resulting MO has seven dimensions, but only the Diagnosis and Result dimensions are non-trivial, i.e., the remaining five dimensions contain only the $\top$ categories. The set of facts is still $F = \{1, 2\}$. The Diagnosis dimension is cut so that only the part from *Diagnosis Group* and up is kept. The result dimension groups the counts into two ranges: "0–1" and "$>$1". The fact-dimension relation for the Diagnosis dimension links the sets of patients

[2]The function g "looks up" the required data for the facts in the relevant fact-dimension relations, e.g., SUM_i finds its data in the relation R_i.

to their corresponding Diagnosis Groups. The content is: $R_1 = \{(\{1,2\},11),(\{2\},12)\}$, meaning that the sets of patients $\{1,2\}$ and $\{2\}$ are characterized by diagnosis groups 11 and 12, respectively. The fact-dimension relation for the result dimension relates each group of patients to the count for the group. The content is: $R_7 = \{(\{1,2\},2),(\{2\},1)\}$, meaning that the results of g on the sets $\{1,2\}$ and $\{2\}$ are 2 and 1, respectively. A graphical illustration of the MO, leaving out the trivial dimensions for simplicity, is seen in Figure 3. Note that each patient is only counted once for each diagnosis group, even though patient 2 has *several* diagnoses in each group.

Other common OLAP and relational operators, such as value-based join, duplicate removal, SQL-like aggregation, star-join, drill-down, and roll-up can easily be defined in terms of the fundamental operators [21]. The algebra satisfies the following two properties [21].

Theorem 1 *The algebra is closed.*

Theorem 2 *The algebra is at least as powerful as Klug's* [16] *relational algebra with aggregation functions* .

4.2 Handling Time in the Algebra

It is a requirement to be able to view data as it appeared at a given point in time, in the database or in the real world, and to do analysis related to time, including analysis across times of change in the data. We note that the operators do not introduce any "value-equivalent tuples"; thus the data stays coalesced. First, we consider valid-time MO's. To be able to view data as they appeared at any given point in time in the real world, we introduce the *valid-timeslice operator* [1].

valid-timeslice operator: Given a chronon t and an MO, $M = (\mathcal{S}, F, D, R)$, we define the valid-timeslice operator τ_v as: $\tau_v(M,t) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = \mathcal{S}$, $F' = F$, $D' = \{D_i'\}, i = 1,..,n$, $D_i' = (C_i', \leq_i')$, $C_i' = \{e \mid e \in_T C_i \wedge t \in T\}$, $e_1 \leq_i' e_2 \Leftrightarrow (e_1 \leq_{i_T} e_2 \wedge t \in T)$, $R' = \{R_i'\}, i = 1,..,n$, and $R_i' = \{(f,e) \mid (f,e) \in_T R_i \wedge t \in T\}$. For a representation Rep of a category type $\mathcal{C}_j$, we have that $Rep(e) = v \Leftrightarrow (Rep(e) =_T V \wedge t \in T)$. Thus, the valid-timeslice operator returns the parts of the MO that are valid at time *t*, with *no valid time attached*, i.e., the valid-timeslice operator changes the temporal type of the MO from valid-time or bitemporal to snapshot or transaction-time, respectively.

To support analysis related to time, we allow predicates p and functions g to be used in selections and aggregate formations that refer to time. We will not go deeper into the structure of temporal predicates and functions; for a full treatment, see, e.g., the TSQL2 language [18].

The last step is to define how the basic algebra operations deal with the time attached to MO's. Neither the selection operator, the projection operator, nor the rename operator change the time attached to the resulting MO's. For the union operator, time attachments for the resulting MO are computed using to the following rules[3]. $(f,e) \in_{T_1} R_{1_i} \wedge (f,e) \in_{T_2} R_{2_i} \Rightarrow (f,e) \in_{T_1 \cup T_2} R_i'$, $e_1 \leq_{1_{T_1}} e_2 \wedge e_1 \leq_{2_{T_2}} e_2 \Rightarrow e_1 \leq'_{T_1 \cup T_2} e_2$, $Rep_1(e) =_{T_1} v \wedge Rep_2(e) =_{T_2} v \Rightarrow Rep'(e) =_{T_1 \cup T_2} v$, $e \in_{1_{T_1}} C_j \wedge e \in_{2_{T_2}} C_j \Rightarrow e \in'_{T_1 \cup T_2} C_j$. Thus, we simply take the union of the chronon sets for data that occur in both MO's; otherwise, we just use the original time. For the difference operator, the following rules apply. $(f,e) \in_{T_1} R_{i_1} \wedge (f,e) \in_{T_2} R_{i_2} \wedge T_1 \backslash T_2 \neq \emptyset \Rightarrow (f,e) \in_{T_1 \backslash T_2} R_i'$, $F' = \bigcap_{i=1,..,n}\{f \mid \exists (f,e_i) \in R_i' \, ((f,e_i) \in_{T'} R_i' \wedge T' \neq \emptyset)\}$. Thus, the time for a pair in a fact-dimension relation for the first MO is cut by the time that the corresponding pair has in the fact-dimension relation for the second MO. Only pairs with non-empty chronon sets are retained. The facts in the resulting MO are those that participate in all the resulting fact-dimension relations during a non-empty set of chronons. As in the non-temporal case, we do not alter the dimensions of the first MO.

The identity-based join operator does not change the time attached to the dimensions of the resulting MO. For the fact-dimension relations, the following rule is used. $(f_k, e_k) \in_{T_k} R_{k_i}, k = 1,2 \wedge p(f_1,f_2) \Rightarrow ((f_1,f_2),e_k) \in_{T_k} R'_{i+(k-1)n_1}$. Thus the pair (f_1,f_2) inherits its time attachment from the fact-dimension relation of the relevant argument MO, i.e, $((f_1,f_2),e) \in_T R_i'$ gets T from $(f_1,e) \in_T R_{1_i}$ if $i \leq n_1$ and from $(f_2,e) \in_T R_{2_i}$ if $i > n_1$.

The aggregate formation operator does not change the time attached to the remaining parts of the argument dimensions or to the result dimension. The time attached to the fact-dimension relations between the facts and the argument dimensions is given by the following rule. Given a tuple of dimension values $(e_1,..,e_n)$ from the grouping categories, $(Group(e_1,..,e_n),e_i) \in_{T_i'} R_i'$, where $T_i' = \cap_{f \in Group(e_1,..,e_n)}\{t_f \mid f \rightsquigarrow_{t_f} e_i\}$. Thus, the time attached to the fact-dimension relation between a set of facts and a dimension value is the intersection of the time attached to the relations between the individual facts and the dimension value. The fact-dimension relation for the result dimension is given by the following rule. Given a tuple of dimension values $(e_1,..,e_n)$ from the grouping categories, $(Group(e_1,..,e_n), g(Group(e_1,..,e_n))) \in_{T'_{n+1}} R'_{n+1}$, where $T'_{n+1} = \bigcap_{f \in Group(e_1,..,e_n), i \in Args(g)}\{t_{f_i} \mid f \rightsquigarrow_{t_{f_i}} e_i\}$. Thus, the time attached to the fact-dimension relation between a set of facts and the result of g on that set is the intersection of the time attached to the relations between the individual facts and the dimension values for the dimensions that g uses as arguments.

For transaction time support, we can define the *transaction-timeslice operator*, τ_t, in the same way as the valid-timeslice operator. Given a transaction-time or bitemporal

[3] Subscript T_1 denotes time for the first argument MO, and T_2 for the second.

MO, this operator returns a snapshot or valid-time MO, respectively. The operators in the algebra support transaction time in the same way as valid time.

5 Conclusion and Future Work

Motivated by the popularity of On-Line Analytical Processing (OLAP) systems for analyzing business data, multidimensional data models have become a major database research area. However, current models do not handle well the complex data found in some real-world applications.

We present a real-world case study from the clinical world and use it to justify nine requirements that a multidimensional data model must satisfy in order to support the complex data found in real-world applications. Eight previously proposed data models are evaluated according to the requirements. Requirements not handled by these models include many-to-many relationships between facts and dimensions, handling change and time, handling uncertainty, and handling different levels of granularity.

We propose a new, extended multidimensional data model, which addresses all nine requirements. Non-strict hierarchies are supported by the partial order on dimension values, while many-to-many relationships between facts and dimensions and handling different levels of granularity is supported by the fact-dimension relations. Additionally, time is handled by adding valid time and transaction time to the basic model, while uncertainty is handled by adding probabilities to the basic model. Advanced features of current models are also supportted. Explicit and multiple hierarchies in dimensions are supported by the lattice structure of the dimension types, and the approach of treating all data as dimensional while still allowing computation allows for symmetrical treatment of dimensions and measures. The aggregation type mechanism helps the user to ensure that data is correctly aggregated. We propose an algebra on the objects from the model and show that it is closed and at least as strong as relational algebra with aggregation functions. The algebra is extended to handle time and probabilities.

In future work, it should be investigated how the model can be efficiently implemented using special-purpose algorithms and data structures. It is also interesting to investigate if the lattice structures of the schema can be used directly in the user interface of an OLAP tool based on the model. Next, a notion of completeness for multidimensional algebras, similar to Codd's relational completeness would be an exciting research topic. Finally, we believe that it is important to investigate how multidimensional models may cope with the hundreds of dimensions found in some applications.

Acknowledgements

This research was supported in part by the Danish Technical Research Council through grant 9700780, by the Danish Academy of Technical Sciences, contract no. EF661, and by a grant from the Nykredit corporation.

References

[1] C. S. Jensen and C. E. Dyreson, (editors). A Consensus Glossary of Temporal Database Concepts—February 1998 Version. In [14], pp. 367–405.

[2] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. In *Proc. of ICDE*, pp. 152–159, 1996.

[3] R. Kimball. *The Data Warehouse Toolkit*. Wiley, 1996.

[4] E. F. Codd. Providing OLAP to user-analysts: An IT mandate. *E.F. Codd and Associates*, 1993.

[5] R. Agrawal et al. Modeling Multidimensional Databases. In *Proc. of ICDE*, pp. 232–243, 1997.

[6] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *Proc. of SSDBM*, pp. 14–29, 1990.

[7] H. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Databases. In *Proc. of SSDBM*, pp. 39–48, 1997.

[8] A. Shoshani. OLAP and Statistical Databases: Similarities and Differences. In *Proc. of PODS*, pp. 185–196, 1997.

[9] M. Gyssens and L. V. S. Lakshmanan. A Foundation for Multi-Dimensional Databases. In *Proc. of VLDB*, pp. 106–115, 1997.

[10] C. Li and X. S. Wang. A Data Model for Supporting On-Line Analytical Processing. In *Proc. of CIKM*, pp. 81–88, 1996.

[11] W. Lehner. Modeling Large Scale OLAP Scenarios. In *Proc. of EDBT*, pp. 153–167, 1998.

[12] World Health Organization. *International Classification of Diseases (ICD-10)*. Tenth Revision, 1992.

[13] A. Datta and H. Thomas. A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases. In *Proc. of WITS*, pp. 91–100, 1997.

[14] O. Etzion, S. Jajodia, and S. Sripada (editors). *Temporal Databases: Research and Practice*. LNCS 1399, Springer-Verlag, 1998.

[15] J. Melton and A. R. Simon. *Understanding the new SQL - A Complete Guide.* Morgan Kaufmann, 1993.

[16] A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, 1982.

[17] M. Rafanelli and F. Ricci. Proposal of a Logical Model for Statistical Databases. In *Proc. of SSDBM*, pp. 264–272, 1983.

[18] R. T. Snodgrass et. al. *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.

[19] C. Bettini et al. A Glossary of Time Granularity Concepts. In [14], pp. 406–413.

[20] J. Clifford et al. On the Semantics of "Now" in Databases. *ACM TODS*, 22(2):171–214, 1997.

[21] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. TimeCenter TR-37, <www.cs.auc.dk/TimeCenter>, 1998.