

Multidimensional Data Modeling for Complex Data

Torben Bach Pedersen and Christian S. Jensen

November 13, 1998

TR-37

A TIMECENTER Technical Report

Abstract

Systems for On-Line Analytical Processing (OLAP) considerably ease the process of analyzing business data and have become widely used in industry. OLAP systems primarily employ multidimensional data models to structure their data. However, current multidimensional data models fall short in their ability 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, which addresses all nine requirements. Along with the model, we present an associated algebra, and outline how to implement the model using relational databases.

1 Introduction

With the continued advances in the underlying hardware technologies for on-line mass storage and the recent focus on data warehousing, the notion of On-Line Analytical Processing (OLAP) [5] is attracting increasing interest, as business managers attempt to extract useful information from large on-line databases in order to make informed management decisions.

Reports indicate that traditional data models, such as the ER model [2] 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* [9]. This appears to be a very valuable direction, as it is necessary to put more semantics into the database schema to support the typical OLAP style of working directly with the data instead of using pre-formatted reports.

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 modeling explicit hierarchies in the dimensions, to aid the user in navigating the data. Multiple hierarchies in each dimension is supported, to allow for different aggregation paths, and the non-strict hierarchies found in real-world dimensions, i.e., where a dimension item may have several parents, are also supported. The model treats dimensions and measures symmetrically, to allow measures to be used as dimensions and vice versa. Many-to-many relationships between facts and dimensions can be captured directly in the model, which is important as these relationships often occur in real-world data, e.g., a patient may have several diagnoses. The data model supports getting correct results when aggregating data, e.g., data will not be double-counted and non-additive data cannot be added. Data change over time, so support for handling change and time is part of the model. Aspects of the uncertainty often associated with data are also handled by the model. Finally, the model supports handling data with different levels of granularity, which is a need in some applications.

The model is equipped with an algebra that is closed and at least as strong as relational algebra with aggregation functions. Finally, we outline how the model can be implemented using relational databases.

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 more than four of 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 sets the stage by first presenting a real-world case study from the clinical world together with nine requirements to multidimensional data models; it then describes and evaluates previously proposed models against the requirements. Section 3 proceeds to first define the basic extended multidimensional data model, using examples from the case study for illustration, then adds support for handling time and uncertainty to the model. With the data structures of the model available, Section 4 defines an algebra for the model and discusses its properties. Section 5 evaluates the model against the requirements, and Section 6 summarizes and points to future directions. An appendix outlines how to implement the model using relational databases.

2 Motivation

This section illustrates the shortcomings of the previously proposed multidimensional models. First, we present a case study that shows some of these limitations. The case is taken from the domain of healthcare, where we look at patients, their diagnoses, and their place of residence. Second, we list the requirements for features that a data model should satisfy in order to meet the needs of the case study. Third, we relate these requirements to the existing multidimensional data models.

2.1 A Case Study

The case study concerns the patients in a hospital, their associated diagnoses, and their place 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 underlying data is seen in Figure 1.

The most important entities are the *patients*. For a patient, we record Name, Social Security Number (SSN), and Date of Birth. From the Date of Birth and the current date, we can derive the Age attribute, which is parenthesized to show that it is derived.

Each patient can have one or more *diagnoses*. The attribution of diagnoses to patients can vary over time, and we also record the time interval where a diagnosis is considered to be valid for a patient. We also record the *type of diagnostization*, to show whether a diagnosis is considered to be *primary* or *secondary*. A primary diagnosis is considered to be the most important reason for a treatment, while secondary diagnoses complete the view of the patient's condition. 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*. Examples of these are "Insulin dependent diabetes during pregnancy," "Insulin dependent diabetes," and "Diabetes," respectively. The higher-level diagnoses are both (imprecise) diagnoses in their own right, but also function as groups of lower-level diagnoses, as

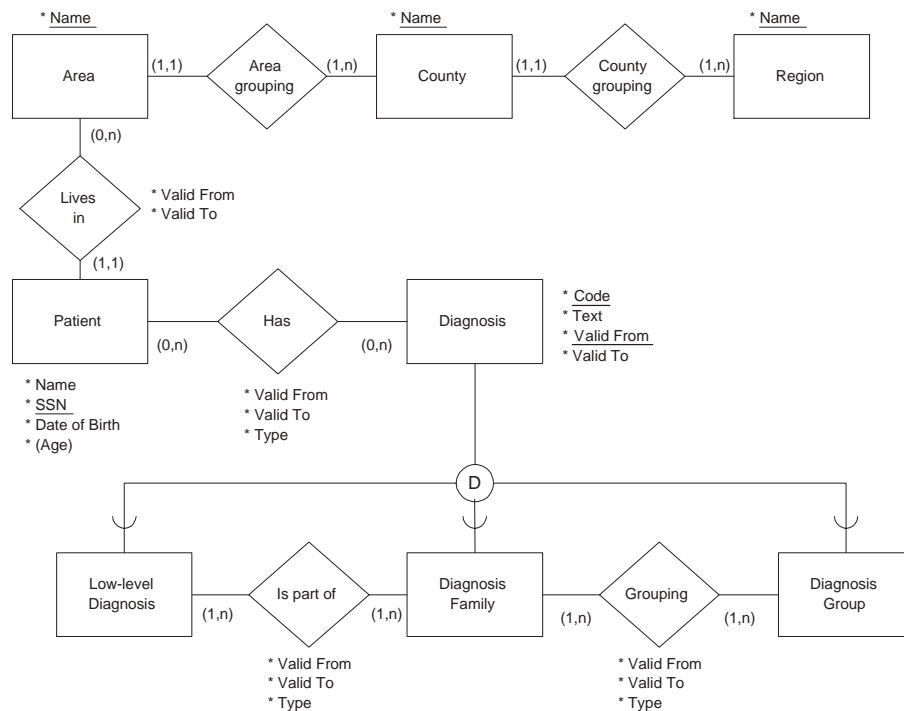


Figure 1: Patient Diagnosis Case Study

will be discussed later.

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) [13], but we also allow user-defined diagnoses.

Over time, medical knowledge evolves, and a disease classification reflects this by changing its contents. What often happens is that a diagnosis is superseded by one or more new diagnoses that better reflect the current understanding of this particular medical condition. To model this fact, we associate with each diagnosis a *period of validity*, represented by the attributes *Valid From* and *Valid To*. This period of validity is the time interval in the real world where the diagnosis can be used for diagnostization, and has the associated code and text. The classifications evolve only slowly, so the granularity of time used can be quite high, e.g., days.

As there are several thousand diagnoses in the classification, the diagnoses are grouped into *diagnosis families* and these in turn into *diagnosis groups*, thus creating a hierarchy in the classification. We have two types of hierarchies: the standard hierarchy determined by the classification owner, e.g., the WHO, and the user-defined hierarchy, which is used by physicians to group diagnoses on an ad-hoc basis in other ways than the standard classification allows.

First, the hierarchy groups low-level diagnoses into *diagnosis families*, each of which consists of 5–50 related diagnoses. For example, the diagnosis “Insulin dependent diabetes during pregnancy¹” is part of the family “Diabetes during pregnancy.” In the standard classification, a low-level diagnosis is part of exactly one diagnosis family. However, physicians often have a need to group diagnoses in other ways than the standard allows, so we also allow a *user-defined* hierarchy. The *Type* attribute on the relationship

¹The reason for having a separate pregnancy related diagnosis is that diabetes must be monitored and controlled particularly intensely during a pregnancy to assure good health of both mother and child.

determines whether the relation between two entities is part of the standard or the user-defined hierarchy.

Thus, a 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. The participation of individual diagnoses in a family may change over time, so we record the time interval during which a diagnosis is part of a family.

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 during pregnancy	01/01/70	31/12/79
4	O24	Diabetes during pregnancy	01/01/80	NOW
5	O24.0	Insulin dependent diabetes during pregnancy	01/01/80	NOW
6	O24.1	Non insulin dependent diabetes during pregnancy	01/01/80	NOW
7	P1	Other pregnancy related diseases	01/01/70	31/12/79
8	D1	Diabetes	01/10/70	31/12/79
9	E10	Insulin dependent diabetes	01/01/80	NOW
10	E11	Non insulin dependent diabetes	01/01/80	NOW
11	E1	Diabetes	01/01/80	NOW
12	O2	Other pregnancy related 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

Second, the diagnosis families are grouped into *diagnosis groups*, consisting of 5–25 families, and one family may be part of several groups. For example, the family “Diabetes during pregnancy” may be the part of the “Diabetes” and the “Other pregnancy related diseases” groups. In the standard hierarchy, however, a family belongs to exactly one group. Here we also use the *Type* attribute to distinguish between the standard and the user-defined hierarchy. The grouping of families into groups can also change over time, so we record the time interval during which a family is part of a group.

In the standard hierarchy, a lower-level item belongs to exactly one higher-level item, thus the standard hierarchy is a *strict, partitioning* hierarchy. In the user-defined hierarchy, a lower-level item can be a member

of zero or more higher-level items, making it a *non-strict, non-partitioning* hierarchy. Properties of the hierarchies will be discussed in more detail in Section 3.4.

We also record the place of residence for the patients. A patient may only live in one place at any one point in time. When people move, their previous address is still interesting, so we also record the associated period of residence. We record the place of residence at the granularity of an *area*, which designates a small, bounded area of a few square kilometers. An area is part of exactly one *county*, which in turn is part of exactly one *region*. Thus, we have a *strict, partitioning* hierarchy. For areas, counties, and regions we just record the name.

In order to list some example data, we assume a standard mapping of the ER diagram to relational tables, i.e., one table per entity type, one-to-many relationships handled using foreign keys, and many-to-many relationships handled using separate tables. Relationships that change over time are also handled using separate tables. We also assume the use of 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 value “NOW” value that denotes the current time² [25]. 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 diagnostizations, 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.

1. *Explicit hierarchies in dimensions.* The hierarchies in the dimensions should be captured explicitly by the schema, so the user has available the relation between the different levels in the hierarchy. In our example, the hierarchies *diagnosis < family < group* and *area < county < region* should be captured.
2. *Symmetric treatment of dimensions and measures.* The data model should allow measures to be treated as dimensions and vice versa. In our example, the attribute Age for patients would typically be treated as a measure, to allow for computations such as average age, etc., but we should also be able to define an Age dimension which allows us to group the patients into age groups.
3. *Multiple hierarchies in each dimension.* In one dimension, there can be more than one path along which to aggregate data. As an example, let us assume that we have a Time dimension on the Date of Birth attribute. Days roll up to weeks and to months, but weeks do not roll up to months. To model this, multiple hierarchies in each dimension are needed.
4. *Support for correct aggregation.* The data model should support getting results that are “correct,” i.e., meaningful to the user, when aggregating data. One aspect of this is to avoid double-counting of data. In our case study, when asking for the numbers of patients in different diagnosis groups, we should only count the same patient once per group, even though the patient has several diagnoses in a group. The user should also be able to specify what aggregations are considered meaningful for the different kinds of data available, and the model should provide a foundation for enforcing these specifications. For example, it may not be meaningful to add inventory levels together, but performing average

²Note that this value is *dynamic*, i.e., it continues to grow.

calculations on them does make sense. In the field of statistical databases, a closely related concept is *summarizability* [7, 8], which means that an aggregate result, e.g., total sales, can be computed by directly combining results from lower-level aggregations, e.g., the sales for each store.

5. *Non-strict hierarchies*. The hierarchies in a dimension are not always strict, i.e., we can have many-to-many relationships between the different levels in a dimension. In our example, the diagnosis hierarchy is not strict. The data model should be able to handle these just as well as “ordinary” strict dimensions.
6. *Many-to-many relationships between facts and dimensions*. The relationship between fact and dimension is not always the classical many-to-one mapping. In our case study, the same patient may have several diagnoses, even at the same point in time.
7. *Handling change and time*. Data change over time, but we should be able to get meaningful analysis results across changes. In the example, one diagnosis can be superseded by two new ones, but patients are still diagnosed with the old one. It should be possible to easily combine data across changes. The problem typically referred to as handling *slowly changing dimensions* [4, 20] is part of this problem.
8. *Handling uncertainty*. In our case study, one diagnosis is superseded by two new ones. We know that in 90% of the cases where we used the old diagnosis, we will now use the first of the new diagnoses. Thus, when requesting data grouped by diagnosis for a period that spans the change, we want the old diagnosis to be counted together with the first new diagnosis. The data model should allow expressing this and also support giving some kind of indications in the query results, indicating how many of these “converted” diagnoses are counted in the result.
9. *Handling different levels of granularity*. Fact data might be registered at different granularities. In our example, the diagnosis of a diabetes patient may be registered differently by different physicians. Some will use a very specific diagnosis such as “Insulin dependent diabetes,” while others will use the more imprecise “Diabetes,” which covers several lower-level diagnoses. It should still be possible to get correct analysis results when data is registered at different granularities.

2.3 Related Work

In this section 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 [7], Agrawal et al. [6], Gray et al. [3], Kimball [4], Li & Wang [11], Gyssens & Lakshmanan [10], Datta & Thomas [14], and Lehner [12]. The models can be divided into three groups: *simple cube models*, *structured cube models*, and *statistical objects*.

The simple cube models [3, 4, 10, 14] treat data as n-dimensional cubes. Generally, the data is divided into *facts*, or *measures*, e.g., Age, on which calculations should be performed, and *dimensions*, e.g., Diagnosis, which characterize the facts. Each dimension has a number of attributes, which can be used for selection and grouping. In our example, a “Residence” dimension having the attributes “Area,” “County,” and “Region” would be used to characterize the patients. The hierarchy between the attributes is not captured explicitly by the schema of the simple cubes, so the user will not be able to learn from the schema that Area rolls up to County and not the other way around. Star schema designs [4] are also considered as a simple cubes, as they are semantically equivalent to these.

The structured cube models [6, 11, 12] capture the hierarchies in the dimensions explicitly, providing better guidance for the user navigating the cubes. This information may also be useful for query optimization

[15]. The hierarchies are captured using either *grouping relations* [11], *dimension merging functions* [6], or an explicit tree-structured hierarchy as part of the cube [12].

The last group of models is the *statistical objects* [7]. For this group, a structured classification hierarchy is coupled with an explicit aggregation function on a single measure to produce a “pre-cooked” object that will answer a very specific set of questions. This approach is not as flexible as the others, but unlike most of these, it provides some protection (summarizability) against getting incorrect results from queries.

The results of evaluating the eight data models against our nine requirements are seen in Table 2. If a model supports all aspects of a requirement, we say that the model provides *full* support, denoted by “√”. If a model supports some, but not all, aspects of a requirement, we say that it provides *partial* support, denoted by “p”. When it is not possible for the authors to determine how support for a requirement should be accomplished in the model, we say that the model provides *no* support, denoted by “-”.

	1	2	3	4	5	6	7	8	9
Rafanelli & Shoshani [7]	√	-	-	√	p	-	-	-	-
Agrawal et al. [6]	p	√	√	-	p	-	-	-	-
Gray et al. [3]	-	√	√	p	-	-	-	-	-
Kimball [4]	-	-	√	p	-	-	p	-	-
Li & Wang [11]	p	-	√	p	-	-	-	-	-
Gyssens & Lakshmanan [10]	-	√	√	p	-	-	-	-	-
Datta & Thomas [14]	-	√	√	-	p	-	-	-	-
Lehner [12]	√	-	-	√	-	-	-	-	-

Table 2: Evaluation of the Data Models

1. *Explicit hierarchies in dimensions*: The simple cube models [3, 4, 10, 14] do not capture the hierarchies in the dimensions explicitly. Some models provide partial support by the *grouping relation* [11] and *dimension merging function* [6] constructs, but do not capture the complete hierarchy together with the cube. This is done by the last two models [7, 12], thus capturing the full cube navigation semantics in the schema.
2. *Symmetric treatment of dimensions and measures*: Half of the models [4, 7, 11, 12] distinguish sharply between measures and dimensions. An attribute designated as a measure cannot be used as a dimensional attribute and vice versa. This restricts the flexibility of the cube designs, e.g., if the Age attribute of the example is a measure, it cannot be used to group patients into age groups. The other half of the models [3, 6, 10, 14] do not impose this restriction. They either do not distinguish between measures and dimensions [3, 10], or they allow for the conversion of measures to dimensions and vice versa [6, 14].
3. *Multiple hierarchies in each dimension*: Some models [7, 12] require that the dimension hierarchies are tree-structured. To support multiple hierarchies, a more general lattice structure is required. All the other models [3, 4, 6, 10, 11, 14] allow multiple hierarchies.
4. *Support for correct aggregation*: Half of the models [3, 4, 10, 11] support correct aggregation partially, by implicitly requiring the dimension hierarchies to be *strict* and *partitioning*, i.e., a lower-level item maps to exactly one item on the next level. This is one of the conditions of summarizability [8]. Two of the models allow for non-strict hierarchies, while not addressing the issue of double-counting, thus providing no support [6, 14]. The remaining two models [7, 12] place explicit conditions on both

the hierarchy (strict and partitioning) and the aggregation functions used (only additive data may be added, etc.), thus providing full support for correct aggregation.

5. *Non-strict hierarchies*: Most of the models [3, 4, 10, 11, 12] implicitly or explicitly require that hierarchies be strict. Two models [6, 14] mention briefly that non-strict hierarchies are allowed, but does not go deeper into the issues raised by allowing this, e.g., the possibility of double-counting and the use of pre-computed aggregates. The remaining model [7] investigates the possible problems with allowing non-strict hierarchies and advises against using this feature.
6. *Many-to-many relationships between facts and dimensions*: None of the models allow many-to-many relationships between facts and their associated dimensions, such as the relationship between patients and diagnoses in the example.
7. *Handling change and time*: Only one model [4] discusses this issue, but none the proposed solutions fully support analysis across changes in the dimensions. None of the other models support analysis across changes, although one mention that this is a very important issue [12].
8. *Handling uncertainty*: None of the models provide built-in support for uncertainty in the data.
9. *Handling different levels of granularity*: None of the models handle different levels of granularity in the data.

To conclude, 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 [4]. Requirements 6, 8, and 9 are not supported by any of the models. The objective of the model proposed in this paper is to support all nine requirements.

3 An Extended Multidimensional 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 define extensions for handling time and uncertainty later.

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 $\mathcal{D} = \{\mathcal{T}_i, i = 1, \dots, n\}$ 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 considered to be *dimensional*, even attributes that would be considered as *measures* in other models.

A dimension type \mathcal{T} is a four-tuple $(\mathcal{C}, \leq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$, where $\mathcal{C} = \{\mathcal{C}_j, j = 1, \dots, k\}$ 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 $\perp_{\mathcal{T}} \in \mathcal{C}$ 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 it’s extension, logically containing all other elements.

We say that C_j is a category type of \mathcal{T} , written $C_j \in \mathcal{T}$, if $C_j \in \mathcal{C}$. We assume a function $Pred : \mathcal{C} \mapsto 2^{\mathcal{C}}$ that gives the set of immediate predecessors of a category type 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: $\perp_{Diagnosis} = Low\text{-}level\ Diagnosis < Diagnosis\ Family < Diagnosis\ Group < \top_{Diagnosis}$. We have that $Pred(Low\text{-}level\ Diagnosis) = \{Diagnosis\ Family\}$. 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 [12, 19], 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 $\Sigma = \{SUM, COUNT, AVG, MIN, MAX\}$, $\phi = \{COUNT, AVG, MIN, MAX\}$, 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, $Aggtype(Low\text{-}level\ Diagnosis) = c$, $Aggtype(Age) = \Sigma$, $Aggtype(Ten\text{-}year\ Age\ Group) = c$, and $Aggtype(DOB) = \phi$.

A dimension D of type $\mathcal{T} = (\{C_j\}, \leq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$ is a two-tuple $D = (C, \leq)$, where $C = \{C_j\}$ is a set of categories C_j such that $Type(C_j) = C_j$ 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 $Type(e) = C_j$.

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 $C_j \in \mathcal{C}$. 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. [3].

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* field 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 is greater than, i.e., 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 $e_1 \leq' e_2 \Leftrightarrow \exists C_1, C_2 \in C' (e_1 \in C_1, e_2 \in C_2 \wedge e_1 \leq e_2)$, that is, D' has a subset of the categories of D and \leq' is the restriction of \leq to these categories. We note that D is a subdimension of itself.

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 $Code(3) = \text{“O24”}$ and $Text(3) = \text{“Diabetes during pregnancy.”}$

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

Example 8 For the case study, we get a six-dimensional MO $M = (S, F, D, R)$, where $S = (Patient, \{Diagnosis, DOB, Residence, Name, SSN, Age\})$ and $F = \{1, 2\}$. The definition of the diagnosis dimension and its corresponding fact-dimension relation was 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.

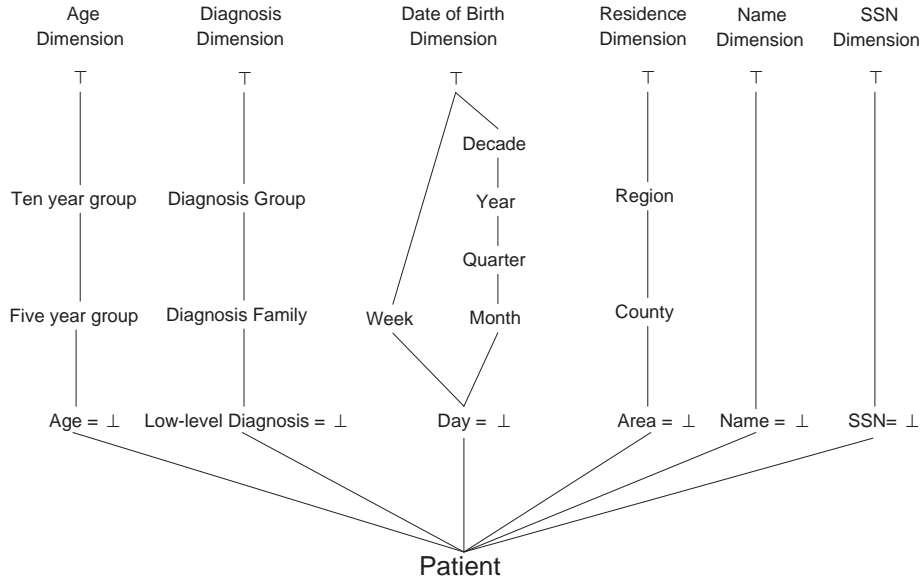


Figure 2: Schema of the Case Study.

A collection of multidimensional objects, possibly with shared subdimensions, is called a *multidimensional object family*.

Example 9 To illustrate the usefulness of shared subdimensions in multidimensional object families, imagine performing the following steps. Create a subdimension of the Diagnosis dimension that includes only *Diagnosis Group* and $\top_{Diagnosis}$, and a subdimension of the Age dimension that includes only *Ten-Year Group* and \top_{Age} . Make an MO with these two dimensions and the fact type Patient for all patients in the country. This results in an MO capturing all patients in the country together with their diagnosis groups and their ten-year age groups. Putting this MO together with the “Patient” MO from the example above, we obtain a multidimensional object family with two shared subdimensions. The shared subdimensions could be used to investigate how diagnoses versus age groups for the patient group from the case study compare to the national average.

To summarize the essence of 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 is a *set*, so we do not have duplicate *facts* in an MO.

3.2 Handling Time

We now investigate how to build temporal support into the model. The vast majority of research in temporal data models assumes a discrete time domain (for example, most data models in the most recent collection of temporal database papers [16] explicitly assume a discrete model of time). Also the temporal data types offered by the SQL standard [17] are discrete and bounded. Thus, 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 [22]. 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 in the real world, 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: $e_1 \leq_{T_{1v}} e_2 \wedge e_2 \leq_{T_{2v}} e_3 \Rightarrow e_1 \leq_{T_{1v} \cap T_{2v}} e_3$. 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 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 statement is the *maximal* set of chronons when the statement is valid, so the data is always “coalesced” [1]. Thus, we do not have the problem of “value-equivalent” statements [1, 21, 23], where the same statement 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, statements are valid for any subset of their 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 10 For our examples, we use interval notation for T_v , with the chronon size equal to Day. For the partial order for the Diagnosis dimension, we have $7 \leq_{[01/01/70-31/12/79]} 3$. For the representation, we have $Code(8)_{[01/01/70-31/12/79]} = D1$. For the fact-dimension relation, we have $(2, 3) \in_{[23/03/75-24/12/75]} R$. For the category membership, we have $10 \in_{[01/01/80-NOW]} Diagnosis\ Family$.

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 the dimension.

Example 11 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 diagnostizations 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 the diagnosis 8 to be logically contained in the diagnosis 11.

Valid time is not the only temporal aspects that may be interesting to our model. 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

Sometimes the data available contains uncertainties that cannot adequately be captured using standard techniques such as default values, etc. Instead, we handle this uncertainty explicitly in our data model. To do so, we introduce measures of probability. In general, it makes sense to attach a probability to a statement if the statement can be given a valid time. In our model, this applies to dimension partial orders, fact-dimension relations, mappings between values and representations, and category memberships for values. However, we attach probabilities to the former two only. It is of little use to have a probability for the mapping between values and representations, which would violate the requirement that a representation is an alternate key. Also, giving probabilities to the membership of categories is omitted, as values belong fully to one category at any given time.

First, we add probability to the partial order on dimension values. Given two dimension values e_1, e_2 and a number p such that $0 \leq p \leq 1$, we write that $e_1 \leq_p e_2$ if $e_1 \leq e_2$ with probability p . Second, we add probability to a fact-dimension relation R . Given a fact f , a dimension value e and a number p such that $0 \leq p \leq 1$, we write that $(f, e) \in_p R$ if $(f, e) \in R$ with probability p . We write $f \rightsquigarrow_p e$ if $(f, e') \in_{p_1} R \wedge e' \leq_{p_2} e \wedge p = p_1 \cdot p_2$. Note that a probability is assigned to *all* (ancestor,descendent) links in the partial order, not just the direct (parent,child) links.

Example 12 Before 1980, diagnostizations in the case study do not differentiate between insulin dependent and non insulin dependent diabetes. From 1980 and on this is the case. Suppose that we know that 90% of the diagnostizations made with the old “Diabetes” diagnosis were for insulin dependent diabetes cases. When looking for the number of insulin dependent diabetes patients from 1970 up till now, we want to count the old “Diabetes” diagnostizations too. We do this by extending the Diagnosis partial order with the information that $8 \leq_{0.9} 9$.

Suppose that physicians are allowed to express their belief in the correctness of a diagnostization by attaching a probability p to it. In the case study, the physician is 95% certain that John Doe ($ID = 1$) has insulin dependent diabetes ($ID = 9$). Thus, for the fact-dimension relation R , $(1, 9) \in_{0.95} R$.

To summarize, the addition of uncertainty to the model is orthogonal to the features for handling time, thus any combination of extensions is valid. If both time and probabilities are added to an MO, we assign a probability p_t for *each* chronon t in the chronon set T . This is done to avoid the problems of value-equivalent tuples. However, interval notation such as $8 \leq_{0.9[1980-NOW]} 9$ is used in examples. If probability is assigned to an MO, we call it a *probabilistic* MO. If no probability is assigned, we call it a *deterministic* MO.

3.4 Properties of the Model

In this section important properties of the model that relate to the use of pre-computed aggregates is defined and discussed. The first important concept is *summarizability*, which intuitively means that individual aggregate results can be combined directly to produce new aggregate results.

Definition 1 Given a type T , a set $S = \{S_j, j = 1, \dots, k\}$, where $S_j \in 2^T$, and a function $g : 2^T \mapsto T$, we say that g is *summarizable* for S if $g(\{g(S_1), \dots, g(S_k)\}) = g(S_1 \cup \dots \cup S_k)$. 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. Space and time constraints can be prohibitive for pre-computing all results, while computing aggregates from scratch results in long response times. In this case, an attractive alternative is the use of *sampling* techniques to answer the queries [24]. Using sampling, only a small sample of the available data is read and used to *estimate* the result of the query. This can produce very fast response times, while maintaining a relatively high degree of accuracy for the result.

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 [8]. 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 $\forall f \in F : f \rightsquigarrow e_1 \wedge f \rightsquigarrow e_2 \wedge e_1 \in C_j \wedge e_2 \in C_j \Rightarrow e_1 = e_2$ ³. The hierarchy in dimension D is *snapshot strict*, if at any given time t , the hierarchy is strict.

Definition 3 If $\forall C_1 \in C (C_1 \neq \top_D \wedge e_1 \in C_1 \Rightarrow \exists C_2 \in \text{Pred}(C_1) (\exists e_2 \in C_2 (e_1 < e_2)))$, 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 13 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 extensions for handling time and uncertainty. For some of the more complex operators, we provide examples of their use.

4.1 The Basic Algebra

We first define the fundamental operators. These are close to the standard relational algebra operators. For unary resp. binary operators, we assume a multidimensional object $M = (S, F, D = \{D_i\}, R = \{R_i\})$, $i = 1, \dots, n$ and multidimensional objects $M_k = (S_k, F_k, D_k = \{D_{k i_k}\}, R_k = \{R_{k i_k}\})$, $k = 1, 2$. We note that 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 values 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.

³Note that the paths from the set of facts F to the $\top_{\mathcal{T}}$ categories are always strict.

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

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 $C_k = \{C_{kj}\}$, $k = 1, 2$, $j = 1, \dots, m$, we define the union operator on dimensions, \cup_D , as: $D_1 \cup_D D_2 = (C', \leq')$, where $C' = \{C'_j\}$, $j = 1, \dots, m$, $C'_j = C_{1j} \cup C_{2j}$, where \cup denotes regular set union, and $e_1 \leq' e_2 \Leftrightarrow e_1 \leq_1 e_2 \vee e_1 \leq_2 e_2$.

selection: Given a predicate p on the dimension types $\mathcal{D} = \{\mathcal{T}_i\}$, we define the selection σ as: $\sigma[p](M) = (S', F', D', R')$, where $S' = S$, $F' = \{f \in F \mid \exists e_1 \in D_1, \dots, e_n \in D_n (p(e_1, \dots, e_n) \wedge f \rightsquigarrow_1 e_1 \wedge \dots \wedge f \rightsquigarrow_n e_n)\}$, $D' = D$, $R' = \{R'_i\}$, 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.

Example 14 If selection is applied to the “Patient” MO with the predicate $Name = \text{“John Doe”}$, the resulting MO has the same schema, the facts $F' = \{1\}$, the fact-dimension relations $R'_i = \{(1, e) \mid (1, e) \in R_i\}$, e.g., $R_2 = \{(1, 9)\}$, and the dimension $D' = D$.

projection: Without loss of generality, we assume that the projection is over the k dimensions D_1, \dots, D_k . We then define the projection π as: $\pi[D_1, \dots, D_k](M) = (S', F', D', R')$, where $S' = (F', D')$, $F' = F$, $D' = \{D_1, \dots, D_k\}$, and $R' = \{R_1, \dots, R_k\}$. Thus, we retain only the k 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.

Example 15 If projection over the Name and Diagnosis dimensions is applied to the “Patient” MO, the resulting MO has the same fact type, only the Name and Diagnosis dimension types, the same set of facts, the Name and Diagnosis dimensions, and the fact-dimension relations for these two dimensions. A graphical illustration of the resulting MO is seen to the left in Figure 3.

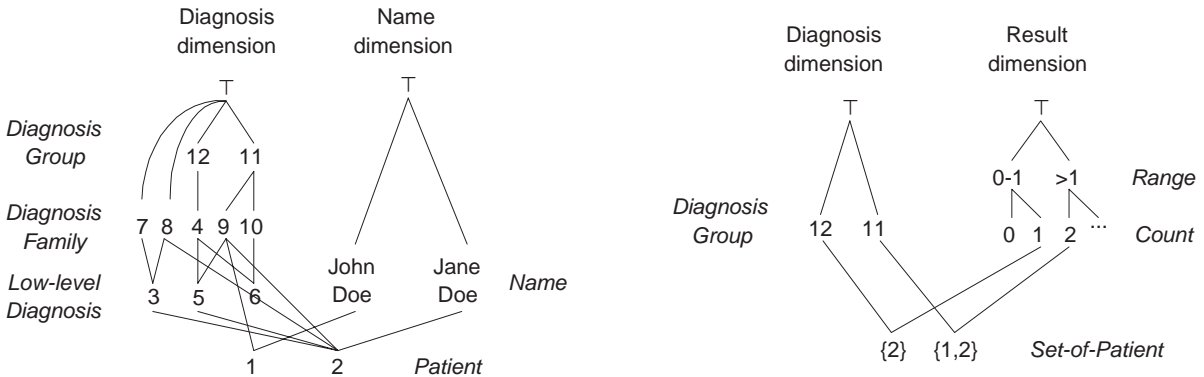


Figure 3: Resulting MO's for Projection and Aggregate Formation

rename: Given a multidimensional object, $M = (\mathcal{S}, F, D, R)$, 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 \cup as: $M_1 \cup M_2 = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = \mathcal{S}_1$, $F' = F_1 \cup F_2$, $D' = \{D_{1_i} \cup_D D_{2_i}, i = 1, \dots, n\}$, and $R' = \{R_{1_i} \cup R_{2_i}, i = 1, \dots, n\}$. In words, given two MO's with common schemas, we take the set union of the facts and the fact-dimension relations. The \cup_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 $\mathcal{S}' = \mathcal{S}_1$, $F' = F_1 \setminus F_2$, $D' = D_1$, $R' = \{R'_i, i = 1, \dots, n\}$, 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.

Example 16 Performing the difference operator on the MO resulting from the projection example and the MO resulting from applying the selection $Name = "Jane\ Doe"$ to the projection MO gives as a result an MO with the same schema, with the fact set $F = \{1\}$, the dimensions from the first argument, and the fact-dimension relations $R_1 = \{(1, 9)\}$ and $R_2 = \{(1, John\ Doe)\}$.

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 $(\mathcal{S}' = (\mathcal{F}', \mathcal{D}'), \mathcal{F}' = \mathcal{F}_1 \times \mathcal{F}_2, \mathcal{D}' = \mathcal{D}_1 \cup \mathcal{D}_2, F' = \{(f_1, f_2) \mid f_1 \in F_1 \wedge f_2 \in F_2 \wedge p(f_1, f_2)\}, D' = D_1 \cup D_2, R' = \{R'_i, i = 1, \dots, n_1 + n_2\},$ and $R'_i = \{(f', e) \mid f' = (f_1, f_2) \wedge f' \in F' \wedge ((i \leq n_1 \wedge (f_1, e) \in R_{1_i}) \vee (i > n_1 \wedge (f_2, e) \in R_{2_{i-n_1}}))\}$. In words, 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 relates a pair to an value if one member of the pair was related to that value before.

Example 17 We want to know if any patients are registered with more than one name. We take two copies of the “Patient” MO and perform projection over the Name dimension for both. For the second copy, the Name dimension type is renamed to “Name2”. We then perform an identity-based join of the two with the predicate $f_1 = f_2$. This gives us an MO with two dimension types, *Name* and *Name2*. The fact type is the type of pairs of patients; the set of facts is still $F = \{1, 2\}$, and the contents of the two dimensions are identical. The fact-dimension relations are also identical: $R_1 = \{(1, John\ Doe), (2, Jane\ Doe)\}$ and $R_2 = \{(1, John\ Doe), (2, Jane\ Doe)\}$. We can now perform a selection on this MO with the predicate $Name \neq Name2$ to find patients with more than one name.

aggregate formation: The aggregate formation operator is used to compute aggregate functions on the MO's. For notational convenience and following Klug [18], we assume the existence of a *family* of aggregation functions g that take some k -dimensional subset $\{D_{i_1}, \dots, D_{i_k}\}$ 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} of type \mathcal{T}_{n+1} , a function, $g : 2^F \mapsto D_{n+1}$ ⁴ such that $g \in \text{MIN}\{\text{Aggtype}(\perp_{D_j}), j \in \text{Args}(g)\}$, and a set of categories $C_i \in D_i, i = 1, \dots, n$, we define aggregate formation, α , as: $\alpha[D_{n+1}, g, C_1, \dots, C_n](M) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = 2^{\mathcal{F}}$, $\mathcal{D}' = \{\mathcal{T}'_i, i = 1, \dots, n\} \cup \{\mathcal{T}_{n+1}\}$, $\mathcal{T}'_i = (C'_i, \leq'_{\mathcal{T}'_i}, \perp'_{\mathcal{T}'_i}, \top'_{\mathcal{T}'_i})$, $C'_i = \{C_{ij} \in \mathcal{T}_i \mid \text{Type}(C_i) \leq_{\mathcal{T}_i} C_{ij}\}$, $\leq'_{\mathcal{T}'_i} = \leq_{\mathcal{T}_i|_{C'_i}}$, $\perp'_{\mathcal{T}'_i} = \text{Type}(C_i)$, $\top'_{\mathcal{T}'_i} = \top_{\mathcal{T}_i}$, $F' = \{\text{Group}(e_1, \dots, e_n) \mid (e_1, \dots, e_n) \in C_1 \times \dots \times C_n \wedge \text{Group}(e_1, \dots, e_n) \neq \emptyset\}$, $D' = \{D'_i, i = 1, \dots, n\} \cup \{D_{n+1}\}$, $D'_i = (C'_i, \leq'_i)$, $C'_i = \{C'_{ij} \in D_i \mid \text{Type}(C'_{ij}) \in C'_i\}$, $\leq'_i = \leq_{i|_{D'_i}}$, $R' = \{R'_i, i = 1, \dots, n\} \cup \{R'_{n+1}\}$, $R'_i = \{(f', e'_i) \mid \exists (e_1, \dots, e_n) \in C_1 \times \dots \times C_n (f' = \text{Group}(e_1, \dots, e_n) \wedge f' \in F' \wedge e_i = e'_i)\}$, and $R'_{n+1} = \cup_{(e_1, \dots, e_n) \in C_1 \times \dots \times C_n} \{(\text{Group}(e_1, \dots, e_n), g(\text{Group}(e_1, \dots, e_n))) \mid \text{Group}(e_1, \dots, e_n) \neq \emptyset\}$. 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, \dots, C_n are strict, and the hierarchies up to C_1, \dots, C_n are partitioning, then $\text{Aggtype}(\perp_{D_{n+1}}) = \text{MIN}\{\text{Aggtype}(\perp_{D_j}), j \in \text{Args}(g)\}$. Otherwise, $\text{Aggtype}(\perp_{D_{n+1}}) = c$. For the higher categories in the result dimension, $\text{Aggtype}(C'_m) = \text{MIN}\{\text{Aggtype}(C_m), \text{Aggtype}(\perp_{D_{n+1}})\}$.

Thus, for every combination (e_1, \dots, 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, \dots, 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.

Example 18 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 to their corresponding Diagnosis Group. 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 relate each group of patient 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 on the right side of Figure 3. Note that each patient is only counted once for each diagnosis group, even though patient 2 has *several* diagnoses in each group.

⁴The function g “looks up” the required data for the facts in the relevant fact-dimension relations, e.g., SUM_i finds its data in fact-dimension relation R_i .

Now, we will show how other common OLAP and relational operators can be defined in terms of the fundamental operators.

value-based join: A join of two MO's on common dimension values can be made in the usual way by combining Cartesian product (a special case of the identity-based join), selection, and projection. *Natural join* is a special case of the value-based join, where the selection predicate requires that values from the “matching” dimensions should be equal, followed by projecting “out” the duplicate dimensions. Performing *drill-across* from one MO to another is just the value-based join of the two MO's on their common dimensions.

duplicate removal: We can remove “duplicate values,” i.e., several facts characterized by the same combination of dimension values, by performing a *set-count* aggregate formation on the \perp categories, followed by projecting out the result dimension.

SQL-like aggregation: Computation of an SQL aggregate function on an MO, grouped by a set of dimension categories, is done by first applying the aggregate formation operator to the MO with the given categories⁵, and the given function. The dimensions not in the “GROUP BY” clause are then projected out.

star-join: A star-join as described in [4] is just a selection on the dimensions, usually combined with an aggregate formation with a given aggregate function on a set of category types.

drill-down: A drill-down on an MO means giving “more detail” by descending the dimension hierarchies. An implicit aggregation function, e.g., COUNT or SUM, is assumed. Thus, a drill-down corresponds to performing aggregate formation on “lower” category types with the given aggregate function. To get to the lower category types, a reference to the *original MO* is needed. In order to obtain the required detail, the aggregate formation is applied to the original object.

roll-up: A roll-up on an MO means giving “less detail” by ascending the dimension hierarchies, aggregating with an implicit aggregation function. This corresponds to performing aggregate formation on “higher” category types with the given aggregate function. Sometimes, we *also* need a reference to the original MO in this case. This is caused by the possible *non-summarizability* in the MO, which means that we cannot necessarily combine the aggregate results from intermediate levels into higher-level results, but need to compute the result directly from the lowest-level data (base data).

Theorem 1 *The algebra is closed.*

Proof: By examining the output of all operators, we see that the results are always MO's.

Theorem 2 *The algebra is at least as powerful as the relational algebra with aggregation functions [18].*

Proof: A relation r with schema $S_r = (a_1, \dots, a_n)$ is mapped to an n -dimensional MO $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (r, \{\mathcal{T}_i, i = 1, \dots, n\})$, $\mathcal{T}_i = (\{a_i, \top_{\mathcal{T}_i}\}, \leq_i, \top_{\mathcal{T}_i}, a_i)$, $a_i \leq_i a_i, a_i \leq_i \top_{\mathcal{T}_i}, \top_{\mathcal{T}_i} \leq_i \top_{\mathcal{T}_i}$, $F = \{(v_1, \dots, v_n) \in r\}$, $D = \{D_i\}, i = 1, \dots, n$, $D_i = (\{A_i, \top_i\}, \leq)$, $A_i = \text{Dom}(a_i), \forall v \in A_i : v \leq \top$, $R = \{R_i\}, i = 1, \dots, n$, and $R_i = \{(v_1, \dots, v_i, \dots, v_n), v_i \mid (v_1, \dots, v_i, \dots, v_n) \in r\}$. Thus, an n -ary relation is mapped to an MO with n “flat” dimensions, each containing the domain of the corresponding attribute.

⁵The categories not in the “GROUP BY” clause are the \top categories of their dimensions.

The facts, corresponding to tuples in the relation, are mapped to the corresponding values in the respective dimensions by the fact-dimension relations.

For every relational algebraic operator, we apply the corresponding operator in our algebra to the corresponding MO, followed by removing duplicates using the method described above. In this way we can emulate all the relational algebraic operations.

4.2 Handling Time in the Algebra

We will now turn our attention to how time can be handled in the algebra. Our requirements are 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 support the need 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 an MO, $M = (S, F, D, R)$, and a chronon t , we define the valid-timeslice operator, τ_v , as: $\tau_v(M, t) = (S', F', D', R')$, where $S' = S$, $F' = F$, $D' = \{D'_i\}, i = 1, \dots, n$, $D'_i = (C'_i, \leq'_i)$, $C'_i = \{e \mid e \in_T C_i \wedge t \in T\}$, $e_1 \leq'_i e_2 \Leftrightarrow (e_1 \leq_{i_T} e_2 \wedge t \in T)$, $R' = \{R'_i\}, i = 1, \dots, n$, and $R'_i = \{(f, e) \mid (f, e) \in_T R_i \wedge t \in T\}$. For a representation Rep of a category type 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 [21].

The last step is to define how the basic algebra operations deals with the time attached to MO’s. Neither the selection operator, the projection operator, or the rename operator change the time attached to the resulting MO’s. For the union operator, time attachments for the resulting MO is computed according to the following rules⁶. $(f, e) \in_{T_1} R_{1_i} \wedge (f, e) \in_{T_2} R_{2_i} \Rightarrow (f, e) \in_{T_1 \cup T_2} R'_i$, $e_1 \leq_{1_{T_1}} e_2 \wedge e_1 \leq_{2_{T_2}} e_2 \Rightarrow e_1 \leq'_{T_1 \cup T_2} e_2$, $Rep_1(e) =_{T_1} v \wedge Rep_2(e) =_{T_2} v \Rightarrow Rep'(e) =_{T_1 \cup T_2} v$, $e \in_{1_{T_1}} C_j \wedge e \in_{2_{T_2}} C_j \Rightarrow e \in'_{T_1 \cup T_2} C_j$. Thus, we simply take the union of the chronon sets for data that occur in both MO’s; otherwise, we just transfer the original time. For the difference operator, the following rules are used. $(f, e) \in_{T_1} R_{1_i} \wedge (f, e) \in_{T_2} R_{2_i} \wedge T_1 \setminus T_2 \neq \emptyset \Rightarrow (f, e) \in_{T_1 \setminus T_2} R'_i$, $F' = \bigcap_{i=1, \dots, n} \{f \mid \exists (f, e_i) \in R'_i ((f, e_i) \in_{T'} R'_i \wedge T' \neq \emptyset)\}$. 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

⁶We use subscript T_1 to denote time for the first argument MO, and T_2 for the second.

(e_1, \dots, e_n) from the grouping categories, $(Group(e_1, \dots, e_n), e_i) \in_{T'_i} R'_i$, where $T'_i = \bigcap_{f \in Group(e_1, \dots, e_n)} \{t_f \mid f \rightsquigarrow_{t_f} e_i\}$. 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 members of the set and that value. The fact-dimension relation for the result dimension is given by the following rule. Given a tuple of dimension values (e_1, \dots, e_n) from the grouping categories, $(Group(e_1, \dots, e_n), g(Group(e_1, \dots, e_n))) \in_{T'_{n+1}} R'_{n+1}$, where $T'_{n+1} = \bigcap_{f \in Group(e_1, \dots, e_n), i \in Args(g)} \{t_{f_i} \mid f \rightsquigarrow_{t_{f_i}} e_i\}$. 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 members of the set and the dimension values for the dimensions that g uses as arguments.

For transaction time support, we can define the *transaction-timeslice operator*, π , in the same way as the valid-timeslice operator. Given a transaction-time or bitemporal MO, it returns a snapshot or valid-time MO, respectively. The operators in the algebra support transaction time in the same way as valid time.

4.3 Handling Uncertainty in the Algebra

We now consider the integration of uncertain data in the algebra. Our requirements are that we are able to view only data which has at least a given probability associated and to do analysis related to the probabilities. To support the former, we introduce the *cut-off operator*, which is similar to the time-slice operators from the previous section.

cut-off operator: Given an MO, $M = (\mathcal{S}, F, D, R)$, and a probability p , we define the cut-off operator, κ as: $\kappa(M, p) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = \mathcal{S}$, $F' = F$, $D' = \{D'_i\}, i = 1, \dots, n$, $D'_i = (C'_i, \leq'_i)$, $C'_i = C_i\}$, $e_1 \leq'_i e_2 \Leftrightarrow e_1 \leq_{i_{p'}} e_2 \wedge p' \geq p$, $R' = \{R'_i\}, i = 1, \dots, n$, and $R'_i = \{(f, e) \mid (f, e) \in_{p'} R_i \wedge p' \geq p\}$. Thus, the cut-off operator returns the parts of the MO, with a probability of at least p , with *no probabilities attached*, i.e., the cut-off operator changes the probabilistic type of the MO from probabilistic to deterministic.

To support analysis related to probabilities, we allow predicates p and functions g that refer to the probabilities. Thus, we can compute results weighted by probability, etc. We will not go deeper into the structure of probabilistic predicates and functions [26].

Next, we define how the basic algebra operations handles the probabilities attached to the MO. Neither the selection operator, the projection operator, or the rename operator change the probabilities attached to the resulting MO's. For the union operator, we do the following. The union operator on dimensions attaches the maximum of the probabilities p_1, p_2 as: $e_1 \leq_{1_{p_1}} e_2 \wedge e_1 \leq_{2_{p_2}} e_2 \Rightarrow e_1 \leq'_{MAX(p_1, p_2)} e_2$. For the fact-dimension relations, the maximum of p_1 and p_2 is assigned as: $(f, e) \in_{p_1} R_{1_i} \wedge (f, e) \in_{p_2} R_{2_i} \Rightarrow (f, e) \in_{MAX(p_1, p_2)} R'_i$. The difference operator retains the probabilities from the first argument MO for the facts that do not appear in the second argument MO.

The identity-based join operator does not change the probabilities in the dimensions. For the fact-dimension relations, the probability of the participation of a pair is inherited from the relevant category of the pair as given by the rule: $(f_k, e) \in_p R_{k_i} \Rightarrow ((f_1, f_2), e) \in_p R'_{i+(k-1)n_1}, k = 1, 2$.

The aggregate-formation operator does not change the probabilities in the remaining parts of the argument dimensions or in the result dimension. For the fact-dimension relations between the facts and the argument dimensions the following rule is used. Given a tuple of dimension values (e_1, \dots, e_n) from the grouping categories, $(Group(e_1, \dots, e_n), e_i) \in_{p'_i} R'_i$, where $p'_i = AVG(\{p_f \mid f \in Group(e_1, \dots, e_n) \wedge f \rightsquigarrow_{p_f} e_i\})$. Thus, the probability we assign to the membership for a set of facts and a dimension value is the *aver-*

age of the probabilities of membership for the individual facts and that dimension value⁷. For the fact-dimension relation between the facts and the result of g , the rule is: Given a tuple of dimension values (e_1, \dots, e_n) from the grouping categories, $(Group(e_1, \dots, e_n), g(Group(e_1, \dots, e_n))) \in_{p'_{n+1}} R'_{n+1}$, where $p'_{n+1} = AVG(\{p_{f_i} \mid f \in Group(e_1, \dots, e_n) \wedge i \in Args(g) \wedge f \rightsquigarrow_{p_{f_i}} e_i\})$. Thus, the probabilities we assign to the result dimension are the *average* of the probabilities of the arguments used for computing the function g , giving a rough “measure of quality” for the results of g .

5 Addressing the Requirements

In this section, we discuss how our model addresses the nine requirements presented in Section 2.2.

The model captures the *explicit hierarchies in dimensions* using the lattice structure of the dimension types. The structure of the case study, seen in Figure 2, is an example. The model *treats dimensions and measures symmetrically* by treating all data as being dimensional. Computations can be performed on dimension values and the results are placed in a dimension. For example, the Age attribute from the case study is used both as a measure and as a dimensional entry. *Multiple hierarchies* are allowed in a dimension. The model requires that the dimension types form a lattice, i.e., with a unique top and bottom type, thus allowing several aggregation paths. The Time dimension in Figure 2 has multiple hierarchies in it. The *Aggtype* mechanism ensures that only aggregation functions that the user finds meaningful are applied to the data, and the specification of the aggregate-formation operator ensures that every fact is only counted once in each result. Thus, the model provides a foundation for *correct aggregation*. For example, in Example 18 every patient is only counted once per *Diagnosis Group*, even though the same patient has several diagnoses in a diagnosis group.

An value in a dimension may have several direct parents in the model, e.g., the diagnosis “Insulin dependent diabetes during pregnancy” has both “Insulin dependent diabetes” and “Diabetes during pregnancy” as direct parents in the Diagnosis dimension. Thus, *non-strict hierarchies* in dimensions are supported. The fact-dimension relations of the model support *many-to-many relationships between facts and dimensions*, e.g., the relationship between Patient and Diagnosis from the case study. By building valid- and transaction-time support into the model, we can view data as it appeared at any given point in time. By extending the partial order of a dimension, it is possible to link values that represent the “same” thing across change, e.g., the old and the new “Diabetes” diagnosis. In this way, we may obtain meaningful analysis results across changes in the data. In this respect, the model supports *handling change and time*. The model also *captures uncertainty* in the data, by assigning probabilities to the partial order and the fact-dimension relations. As an example, we can express that “John Doe” has “Diabetes” with probability 0.95. The dimension values that are part of the fact-dimension relations can belong to any category in the dimension, supporting in this manner *different levels of granularity* in the data. For example, we can express that some patients are diagnostized with *low-level diagnoses* and some with *diagnosis families*.

6 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 systems.

We present a real-world case study from the clinical world, where we track patients, their diagnoses, names, social security numbers, dates of birth, ages, and places of residence. We use the case study to

⁷This is not necessarily the true, mathematical probability for the membership. Thus, our approach is “fuzzy” rather than probabilistic. The average is chosen to suit our purpose of giving a rough “measure of inclusion.”

justify nine requirements that a multidimensional data model must satisfy in order to support the complex data found in real-world applications. Requirements not handled by current models include many-to-many relationships between facts and dimensions, handling change and time, handling uncertainty, and handling different levels of granularity. Eight previously proposed data models are evaluated according to the requirements, and it is shown that none of them satisfies more than four requirements fully or partially.

We propose a new, extended multidimensional data model, which addresses all nine requirements. The data model improves over previously proposed models by supporting non-strict hierarchies, many-to-many relationships between facts and dimensions, handling change and time, handling uncertainty, and handling different levels of granularity. Especially, 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. We propose an algebra on the multidimensional objects from the model, and we 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. Finally, we show how to represent the multidimensional objects as relational tables, thus providing a basis for implementing the model using relational technology.

In the future 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 for 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 can cope with the hundreds of dimensions found in some applications.

References

- [1] C. S. Jensen and C. E. Dyreson, (editors). A Consensus Glossary of Temporal Database Concepts - February 1998 Version. In [16], pp. 367–405.
- [2] P. P.-S. Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transaction on Database Systems*, 1(1):9–36, 1976.
- [3] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.
- [4] R. Kimball. *The Data Warehouse Toolkit*. Wiley Computer Publishing, 1996.
- [5] E. F. Codd. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. *Technical report*, E.F. Codd and Associates, 1993.
- [6] R. Agrawal, A. Gupta, S. Sarawagi. Modeling Multidimensional Databases. *IBM Technical Report 1995*. Also appeared in *Proceedings of the Thirteenth International Conference on Data Engineering*, pp. 232–243, 1997.
- [7] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *Proceedings of the 5th Conference on Statistical and Scientific Database Management*, pp. 14–29, 1990.
- [8] H. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proceedings of the 9th Conference on Statistical and Scientific Database Management*, 1997.
- [9] A. Shoshani. OLAP and Statistical Databases: Similarities and Differences. In *Proceedings of Sixteenth ACM Symposium on Principles of Database Systems*, pp. 185–196, 1997.

- [10] M. Gyssens and L. V. S. Lakshmanan. A Foundation for Multi-Dimensional Databases. In *Proceedings of the 23rd Conference on Very Large Databases*, pp. 106–115, 1997.
- [11] C. Li and X. S. Wang. A Data Model for Supporting On-Line Analytical Processing. In *Proceedings of the Fifth International Conference on Information and Knowledge Management*, pp. 81–88, 1996.
- [12] W. Lehner. Modeling Large Scale OLAP Scenarios. In *Proceedings of the Sixth International Conference on Extending Database Technology*, pp. 153–167, 1998.
- [13] World Health Organization. *International Classification of Diseases (ICD-10)*. Tenth Revision, 1992.
- [14] A. Datta and H. Thomas. A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases. In *Proceedings of the Seventh Annual Workshop on Information Technologies and Systems*, 1997.
- [15] W. Lehner and T. Ruf. A Redundancy-based Optimization Approach for Aggregation in Multidimensional Scientific and Statistical Databases. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications*, pp. 253–262, 1997.
- [16] O. Etzion, S. Jajodia, and S. Sripada (editors). *Temporal Databases: Research and Practice*. LNCS 1399, Springer-Verlag 1998.
- [17] J. Melton and A. R. Simon. *Understanding the new SQL - A Complete Guide*. Morgan Kaufmann, 1993.
- [18] A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, 1982.
- [19] M. Rafanelli and F. Ricci. Proposal of a Logical Model for Statistical Databases. In *Proceedings of the 2nd International Workshop on Statistical and Scientific Database Management*, 1983.
- [20] R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Systematic Change Management in Dimensional Data Warehousing. In *Proceedings of the Third International Baltic Workshop on DB and IS*, pp. 27–41, 1998.
- [21] R. T. Snodgrass et. al. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.
- [22] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, X. S. Wang. A Glossary of Time Granularity Concepts. In [16], pp. 406–413.
- [23] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying Temporal Data Models via a Conceptual Model. *Information Systems*, 19(7):513–547, 1994.
- [24] F. Olken, D. Rotem. Random Sampling from Databases - A Survey. *Statistics & Computing*, 5(1):25–42, March 1995.
- [25] J. Clifford, , C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of “Now” in Databases. *ACM Transactions on Database Systems*, 22(2):171–214, June 1997.
- [26] D. Barbará, H. García-Molina, and D. Porter. The Management of Probabilistic Data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, October 1992.

A Representation of the Model

This appendix outlines how to implement the model using relational databases. Our primary concern is a relational representation that allows efficient evaluation of queries in the model. The metadata specified in the model, e.g., the aggregation types, must be stored separate from the data and handled by the tool accessing the data, e.g., a query tool. We do not go into how to represent this metadata in a relational database.

The traditional way to map a multidimensional data model to a relational database is to use a *star schema* [4], where the *fact table* contains measures and foreign keys to the *dimension tables*. However, a star schema design requires the relationships between the fact and dimension tables to be many-to-one, and that the hierarchies in the dimensions be strict. To represent many-to-many relationships between facts and dimensions, several rows in the fact table are necessary for each fact. To represent non-strict hierarchies, several rows in the dimension tables are necessary for each dimension key. These violations of the pure star schema design can lead users to get incorrect results when aggregating data, as it is easy to accidentally double-count data. Alternatively, if the users understand the potential problems, they need to employ the expensive SELECT DISTINCT clause in SQL statements to get correct results.

To avoid these problems, we use a non-standard mapping to relational tables. The basic idea for representing the dimensions is to encode the partial order on a dimension composed with a category representation, directly in one table. Thus, for each representation Rep of a category C_j in the given MO, we get a table T_{C_j-Rep} that encodes the composition of Rep with the partial order on the dimension. If Rep is encoded in the table $T_{Rep} = (RepValue, DimensionValue)$ and the direct parent-child relationships in the partial order is encoded in the table $PO = (ParentValue, ChildValue)$ then $T_{C_j-Rep} = T_{Rep} \bowtie_{DimensionValue=ParentValue} PO^*$, where PO^* denotes the reflexive, transitive closure of PO . Note that T_{C_j-Rep} does not contain duplicates. This means that we will not get double-counting of data when computing aggregates in term of the base table if the hierarchy is non-strict, as would have been the case with the star schema representation described above. T_{C_j-Rep} can be updated incrementally during insertions to PO using the rule: $PO' = \{(e_1, e_2)\} \cup PO \Rightarrow PO'^* = PO^* \cup PO^* \bowtie \{(e_1, e_2)\} \bowtie PO^*$.

Example 19 The Grouping table in Table 1 encodes the direct parent-child relationships in the Diagnosis dimension. We perform the reflexive, transitive closure of the Grouping table, thus getting all ancestor-descendent pairs in the Diagnosis partial order. This is joined with the Diagnosis table, which encodes the *Code* representation for the Diagnosis Group category. This gives us the the table $T_{DiagGroup-Code}$ which can be seen in Table 3. The temporal aspects of the table will be discussed later.

Code	ParentValue	ChildValue	ValidFrom	ValidTo
E1	11	5	01/01/80	NOW
E1	11	6	01/01/80	NOW
E1	11	9	01/01/80	NOW
E1	11	10	01/01/80	NOW
E1	11	11	01/01/80	NOW
O2	12	4	01/01/80	NOW
O2	12	5	01/01/80	NOW
O2	12	6	01/01/80	NOW
O2	12	9	01/01/80	NOW
O2	12	12	01/01/80	NOW

Table 3: The $T_{DiagGroup-Code}$ Table for the Example

Several alternatives exist for the representation of the fact-dimension relations. If the fact-dimension relationships are many-to-one, a standard fact table approach with “foreign keys” to the dimension-encoding

tables will suffice. If relationships are many-to-many, there are three alternatives: a) maintain the dimension encoding tables joined with the fact-dimension relation, with no duplicates, making the resulting table “point to” the facts, b) make a new “lowest” level in the dimension-encoding tables for each *combination* of dimension values pointed to by one fact, and make the fact table point to the combination, and c) encode the fact-dimension relation directly as a separate table.

Example 20 For the example above, alternative a) would maintain the join of $T_{\text{DiagGroup-Code}}$ with the Has table. Alternative b) would give two combinations, $\{9\}$ and $\{3, 5, 8, 9\}$, which would be the bottom values in the extension of the $T_{\text{DiagGroup-Code}}$ table. The fact table would then point to these combinations, instead of the diagnoses directly. Alternative c) would just keep the $T_{\text{DiagGroup-Code}}$ and Has tables.

Each alternative has its own advantages. Alternative a) provides direct access to the facts, with no problems of double-counting, but the tables can become very big, as we have several rows for each fact-dimension pair, thus rendering the solution impractical. Alternative b) is attractive if the number of combinations is small, as we avoid the problems of double-counting, but if the number of combinations is large, we have the same size problems as in a). Alternative c) keeps the size of the tables to the minimum, but accidental double-counting is possible, thus `SELECT DISTINCT` clauses must be used in SQL statements.

When extending the representations to capture valid/transaction time, the basic dimension-encoding mechanism still works. The encoding table is extended with columns capturing the time when the tuple is true. We take the intersection of the time periods when tables are joined, thus capturing the time period when the combined tuples are valid. The $T_{\text{DiagGroup-Code}}$ table extended with time is seen in Table 3. Alternative a) and c) can be extended with time columns without any problems. For alternative b) we need to enumerate all combinations of dimension values *and* the associated time periods. This will probably lead to a number of combinations that is close to the number of facts, thus rendering the solution impractical.

When extending the representations to capture uncertainty, we just extend the tables with columns for the probabilities. When joining tables, the product of the probabilities is used for the resulting tuple. Alternatives a) and c) for representing the fact-dimension relations can be extended with probability columns without any problems. For alternative b) we need to enumerate all combinations of both the dimension values *and* the associated probabilities, perhaps yielding a considerably larger number of combinations.