Pergamon

Information Systems Vol. 19, No. 4, pp. 33–54, 1994

0306-4379(94)E0004-9
0306-4379/94 $7.00 + 0.00

A FOUNDATION FOR CAPTURING AND QUERYING COMPLEX MULTIDIMENSIONAL DATA

TORBEN BACH PEDERSEN[1], CHRISTIAN S. JENSEN[1] AND CURTIS E. DYRESON[2]

[1]Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø, Denmark, {tbp,csj}@cs.auc.dk

[2]School of Electrical Engineering and Computer Science, Washington State University, PO Box 642752, Pullman, Washington 99164-2752, USA, cdyreson@eecs.wsu.edu

1. INTRODUCTION

On-Line Analytical Processing (OLAP) [12] is an area of active commercial and research interest. Continued advances in hardware for on-line mass storage have made possible the warehousing of large amounts of data. OLAP tools focus on providing fast answers to ad-hoc queries that *aggregate* the warehouse data. This enables users to quickly analyze the data and make informed decisions.

Traditional data models, such as the ER model [9] 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. Multidimensional models typically categorize data as either *measurable business facts* (measures), which are numerical in nature, 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*. Typical measures would be the amount and price of the purchase. Typical dimensions would be the location of the purchase, the type of product being purchased, and the time of the purchase.

Most OLAP research to date has concentrated on performance issues. Higher-level issues, such as conceptual modeling, have received less attention. Several researchers have identified this deficiency and have suggested combining the good performance of OLAP systems with the advanced data modeling capabilities of *scientific and statistical databases* [50]. Such a combination would inject more semantics into the database schema and would support the typical OLAP style of working directly with the data instead of relying on pre-formatted reports.

The use of OLAP tools has recently spread to medical applications, where physicians study the data associated with patients. Denmark's largest provider of healthcare IT, KMD, spends significant resources on applying OLAP technology to medical applications.

The use of OLAP tools in medical and other real-world applications raises new challenges for OLAP technology. Based on previous studies of requirements in the medical domain [37, 38], this paper presents eleven modeling-related requirements that a multidimensional data model should satisfy. The requirements are illustrated using a medical case study. Fourteen previously proposed data models, which are representative of the spectrum of multidimensional data models, are evaluated against the requirements. No existing model satisfies more than six of these requirements. This paper presents an extended multidimensional data model that addresses all eleven requirements.

The medical case study presented in Section 2 highlights two main problem areas for current OLAP technology. The first problem area is "imperfect" data. Imperfections, which invariably occur in medical data, include data values that are *missing* and values that are *imprecise* to varying degrees. Generalizing a multidimensional database term, we say that values with varying degrees of precision have *varying granularities*. The problem of varying granularities surfaces in OLAP applications for several different reasons. Some data, such as the data in the case study presented in Section 2, has naturally varying granularities. Data at varying granularities is also common

when data from different organizations is combined. Such data cannot be handled by existing OLAP tools and techniques since they require that the data have a uniform granularity. In a process called *data cleansing*, granularity variances are removed prior to admitting the data into an OLAP database. Data is cleansed by mapping it to a common, coarse granularity. But this degrades the quality of the data, possibly leading to erroneous conclusions based on the results of subsequent OLAP queries. Rather, offering the physicians the ability to access the imperfect data enables them to obtain as meaningful and informative answers as possible to their OLAP queries.

The second problem area is "imperfect" hierarchies. Many OLAP data models support hierarchies in the dimensions. These enable a user to drill-down and roll-up in the aggregate data. Our model adds to the traditional hierarchies in several ways. Multiple hierarchies in each dimension are supported, to allow for different aggregation paths within a dimension. *Non-strict* hierarchies, where a dimension item may have several parents, are also supported. Our 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 since such relationships often occur in real-world data. The data model supports the use of the aggregation semantics of the data to obtain correct results when aggregating data, e.g., data will not be double-counted, and non-additive data cannot be added. Data changes over time, so support for handling change and time is part of the model.

This paper is structured as follows. Section 2 sets the stage by presenting a real-world, medical case study together with eleven requirements to multidimensional data models. It also describes and evaluates previously proposed models against the requirements. Section 3 defines the basic extended multidimensional data model, using examples from the case study for illustration, then extends the model with support for handling time. With the data structures of the model available, Section 4 defines the algebraic query language of the model and discusses its properties. The query language is closed and at least as strong as relational algebra with aggregation functions. Section 5 extends the model to handle imprecision. It also describes how to suggest alternative queries if the original query is affected by imprecision in the data. Section 6 covers imprecision in the grouping of data and in the computation of the aggregate results, as well as in the presentation of the imprecise result to the user. Section 7 evaluates the model against the requirements. Sections 8 and 9 describe how to use relational databases technology to implement the new model and to handle imprecise data during querying, respectively. Section 10 discusses the use of pre-aggregated data for more efficient query evaluation involving imprecision. Section 11 summarizes the paper and identifies pertinent research directions.

2. MOTIVATION AND RELATED WORK

Section 2.1 presents a medical case study. The study illustrates the paper's contributions and also motivates the requirements for multidimensional models, which are presented in Section 2.2. Section 2.3 relates these requirements to existing multidimensional data models. Finally Section 2.4 considers related work in imprecise-data management.

2.1. A Case Study

The case study concerns diabetes patients. Over a period of several years, the study monitors each patient's blood sugar level, diagnosis (i.e., what kind(s) of diabetes they suffer from), and their place of residence. The goal of the study is to determine how blood sugar levels vary among diagnoses. Also of interest is whether specific diagnoses are more frequent in some areas than in others. A high frequency may indicate environmental factors that contribute to a disease pattern. An ER diagram illustrating the underlying data is shown in Figure 1.

The most important entity type is *Patient*. The study records a patient's Name, Date of Birth, Age, HbA1c%, and Precision. Age is parenthesized to show that it is derived. HbA1c% is the long-term blood sugar level [22]. It is an important measurement for diabetes patients since it provides a good overall indicator of the patient's status. But sometimes this value is *missing*, usually because a physician does not test the HbA1c% during a patient's visit. The HbA1c% is

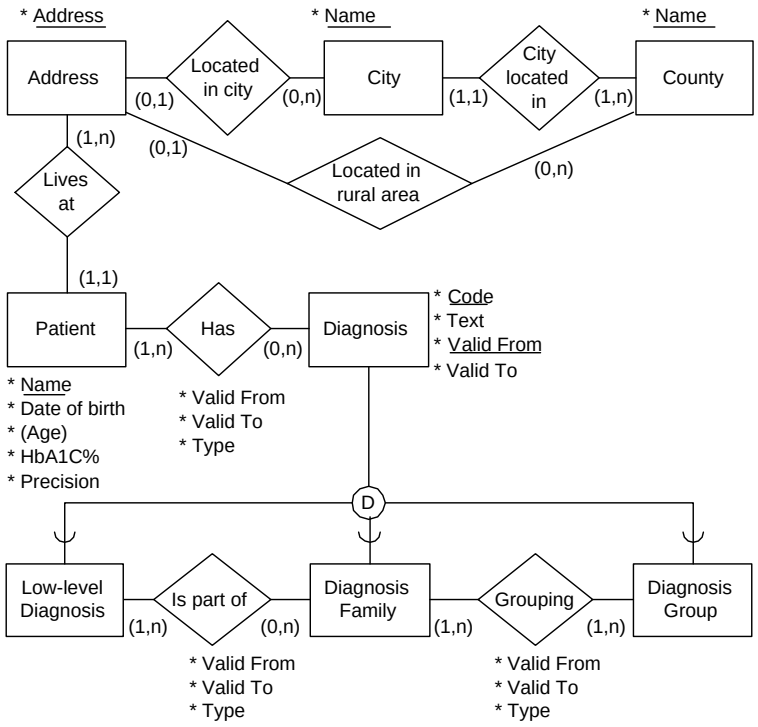

Fig. 1: Patient Diagnosis Case Study

measured using two different methods. Several hospitals use an older, imprecise method, but some have improved their practice and now use a more precise method. The precision attribute records this difference in HbA1c% measurement methods. The value of the attribute is either *precise*, *imprecise*, or *inapplicable* (when the HbA1c% is missing).

The entity type *Diagnosis* represents a condition that a physician identifies in a patient. The code and text description of a diagnosis are determined by a standard classification of diseases, e.g., the World Health Organization's International Classification of Diseases (ICD-10) [54]. A patient always has at least one diagnosis, but may be ill enough to have several diagnoses. The time interval when each diagnosis is considered valid for a patient is also stored since the diagnoses for a patient vary over time. The *type* of a diagnosis indicates whether it is considered *primary* or *secondary*. A patient can have only one primary diagnosis at any time. A primary diagnosis constitutes the most important reason for a treatment, while secondary diagnoses complete the description of a patient's condition.

Diagnosing a patient is inherently difficult and inexact. When registering a diagnosis for a patient, a physician may make a very precise diagnosis such as "Insulin dependent diabetes." But often a physician will be less sure of the diagnosis and will use a less precise diagnosis such as "Diabetes," which covers a wider range of patient conditions corresponding to a number of more precise diagnoses. To model this, the entity type Diagnosis is specialized into three subtypes: *Low-level Diagnosis*, *Diagnosis Family*, and *Diagnosis Group*. Each of the subtypes is a diagnosis. For example, "Diabetes" is a diagnosis group. The most precise diagnosis is a low-level diagnosis. A diagnosis family is a less precise diagnosis. Each diagnosis family consists of 5–25 related low-level diagnoses. A diagnosis group is an even less precise diagnosis. Each group consists of 5–50 diagnosis families.

The diagnosis subtypes capture a *hierarchy* of diagnoses, from low-level diagnoses to diagnosis groups. Hierarchies are central to OLAP tools since they enable users to query aggregate data at any level of precision within the hierarchy, drilling down to a more precise view or rolling up to

a more abstract view when desired. Three aspects of the hierarchy of diagnoses deserve special attention. First, the diagnosis hierarchy is *non-strict*. In a non-strict hierarchy a lower-level item can be a member of several items at a higher-level. Traditionally, OLAP systems only permit strict hierarchies where every lower-level item belongs to a single higher-level item. In the diagnosis hierarchy, a low-level diagnosis can be part of several diagnosis families or groups. In particular, the "Insulin dependent diabetes during pregnancy" diagnosis is part of both the "Diabetes during pregnancy" diagnosis family and the "Insulin dependent diabetes" diagnosis family. In this paper, we develop a model that correctly and efficiently supports non-strict hierarchies.

Second, the hierarchy evolves over time. Standards like the World Health Organization classification for diseases constantly evolve. New diseases are added to the standard, and old diseases are reclassified as new knowledge becomes available. While changes in the hierarchy are commonplace in the real-world, OLAP support for change is rare. In this paper, we present a model that fully supports changing hierarchies. To capture the diagnosis classification as it changes over time, we also record the time intervals where the diagnoses are "valid," i.e., when they can be used for diagnosing patients.

Third, the hierarchy is not *onto* (not balanced), e.g., the "Cancer" Diagnosis Group has a "Lung Cancer" Diagnosis Family under it in the hierarchy, but there is no further subdivision into more precise Low-level Diagnosis cancer types. The model presented here also addresses this aspect of hierarchies.

We also record the place of residence for the patients. A patient lives at one *address*. For simplicity, we do not capture the changes of addresses over time. Some addresses are in *cities*, whereas others are rural, in which case we just record the *county* they are in. Cities are part of exactly one *county*. This hierarchy also has a property that occurs in real-world hierarchies, but is not supported by OLAP systems. Not every address is in a city. Hence cities does not cover the same portion of the underlying domain as do addresses (and counties). We call this a *non-covering* hierarchy, and it is important that OLAP systems support such hierarchies, e.g., to permit drilling down from counties to cities and then to addresses. In this paper we describe a model that supports non-covering hierarchies.

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. Surrogate keys, named *ID*, with globally unique values are used. Dates are written in the format dd/mm/yy. For the *Valid To* attribute, the special value "NOW" denotes the continuously changing current time [10]. As the three subtypes of the Diagnosis type have the same attributes, 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 three patients and their associated diagnoses and addresses, 12 diagnoses in a hierarchy, four addresses, two cities, and three counties. On January 1, 1980, a new, more detailed classification with a new coding scheme was introduced. The resulting tables are shown in Table 1 and will be used in examples throughout the paper.

A particularly interesting problem in the case study is the presence of imprecise data. The primary users are physicians that issue queries that aggregate the available data in order to obtain high-level information. For example, it is important to keep the HbA1c% as close to normal as possible, as patients might collapse or get liver damage if the HbA1c% is too low or too high, respectively. Thus, a typical query is to ask for the *average HbA1c% grouped by low-level diagnosis*. This shows the differences in the blood sugar level for the different patient groups, as determined by the diagnoses, indicating which patients will benefit the most from close monitoring and control of the HbA1c%.

However, as the example data shows, there are some problems in answering this query. First, one of the patients, Jim Doe, is diagnosed with "Diabetes," which is a diagnosis family. Thus, the diagnosis is not precise enough to determine the low-level diagnosis for Jim Doe. Second, the HbA1c% values themselves are imprecise. John Doe has a value obtained with the new, precise measurement method, Jane Doe has only an imprecise value, and Jim Doe's HbA1c% is unknown.

The imprecision of the data must be communicated to the physicians so that the level of

ID	Name	D.O.B.	HbA1C%	Precision	AddressID
0	Jim Doe	03/05/57	Unknown	Inapplicable	52
1	John Doe	12/21/69	5.5	Precise	50
2	Jane Doe	10/10/50	7	Imprecise	51

Patient Table

ID	Address
50	21 Central Street
51	34 Main Street
52	123 Rural Road
53	1 Sandy Dunes

Address Table

ID	Name	CountyID
20	Sydney	30
21	Melbourne	31

City Table

AddressID	CityID
50	20
51	21

LocatedInCity Table

AddressID	Name
52	31
53	32

LocatedInRuralArea Table

ID	Name
30	Sydney
31	Melbourne
32	Outback

County Table

PatientID	DiagnosisID	ValidFrom	ValidTo	Type
0	11	12/21/82	NOW	Primary
1	10	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
13	A1	Cancer	01/01/80	NOW
14	A11	Lung cancer	01/01/80	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
13	14	01/01/80	NOW	WHO

Grouping Table

Table 1: Data for the Case Study

imprecision can be taken into account when interpreting the query results. This helps to ensure that the physicians will not make important clinical decisions on a "weak" basis. Several strategies are possible for handling the imprecision. First, the physicians may only be allowed to ask queries on data that is *precise enough*, e.g., the grouping of patients must be by diagnosis family, not

low-level diagnosis. Second, the query can return an imprecise result. Possible alternatives to this can be to include in the result only what is *known* to be true, everything that *might* be true, and a *combination* of these two extremes. Our paper presents an approach to handling imprecision that integrates both the first and the second strategy, i.e., only "precise enough" data or imprecise results, and provides all the three above-mentioned alternatives for returning imprecise results.

2.2. Requirements for Data Analysis

This section describes the requirements that a data model should satisfy in order to fully support our sample case and other uses. The next section evaluates current multidimensional models are evaluated against these features.

1. *Explicit hierarchies in dimensions.* The hierarchies in the dimensions should be captured explicitly by the schema. This permits the user to drill-down and roll-up. In our example, the hierarchies *diagnosis* < *family* < *group* and *address* < *city* < *county* should be captured.

2. *Multiple hierarchies in each dimension.* A single dimension can have several paths for aggregating data. As an example, 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.

3. *Support for aggregation semantics.* The data model should capture the aggregation semantics of the data and use this to provide a "safety net" that catches queries that might give results that have no meaning to the user. Aspects of this include built-in support for avoiding double-counting of data and avoiding addition of non-additive data.

   For example, when asking for the numbers of patients in different diagnosis groups, we should only count the same patient once per group, even if the patient has several diagnoses in a group. The user should also be able to specify which aggregations are considered meaningful for the different kinds of data available, and the model should provide a foundation for enforcing these specifications. As an illustration, it may not be meaningful to add inventory levels together across time, but performing average calculations on them does make sense. In the field of statistical databases, a closely related concept is *summarizability* [30, 43], 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.

4. *Non-strict hierarchies.* The hierarchies in a dimension may be non-strict, i.e., we can have many-to-many relationships between the different levels in a dimension. In our example, the diagnosis hierarchy is non-strict. The data model should be able to handle these just as well as "ordinary" strict dimensions. Non-strict dimensions occur in many real-world situations such as diagnosis hierarchies [44], Web concept hierarchies such as that of Yahoo! [55], and organization hierarchies [59]. Because such dimensions make sense to the users, they should be supported by the data model.

5. *Non-onto hierarchies.* Often, the hierarchies in a dimension are not balanced, i.e., the path from the root to the leaves has varying length. In our case, this occurs in the diagnosis hierarchy, where the "Lung Cancer" diagnosis has no low-level diagnosis children.

6. *Non-covering hierarchies.* Another common feature of real-world hierarchies is that links between two nodes in the hierarchy "skips" one or more levels. For example, the address "123 Rural Road" in the residence hierarchy is mapped directly to the county, bypassing the city level.

7. *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.

8. *Many-to-many relationships between facts and dimensions.* The relationship between fact and dimension is not always the classical many-to-one one. In our case study, the same patient may have several diagnoses, even simultaneously.

9. *Handling change and time.* Although data change over time, it should be possible to perform meaningful analyses across times when data change. 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* [5, 26] is part of this problem.

   In passing, we note that a somewhat related stream of research exists [56, 57]. This research considers issues related to temporal view maintenance in the context of the relational model. Its relationships to data warehousing are that the relational model may be used as the basis for a data warehouse and that data warehouses typically store data derived from other sources. Multidimensional concepts are not considered, and focus is on implementation techniques.

10. *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 less precise "Diabetes," which covers several lower-level diagnoses. It should still be possible to get correct analysis results when data is registered at different granularities.

11. *Handling imprecision.* Finally, it is very important to be able to capture directly the imprecision in the data and allow queries to take it into account. For example, the HbA1c% for patients has varying precision, and it is important that this is captured and communicated to the users, as described in Section 2.1.

Many other requirements may be posed to multidimensional data models (some such are identified in Section 11 where research directions are covered). We have chosen the eleven requirements above for several reasons. First, they are non-trivial and are not satisfied by all existing models. Second, they are "model" requirements that affect the core data model of a multidimensional data model. This contrasts less fundamental requirements that may be met by simply adding a new facility to the data model's query language. Third, they derive from previous studies of the types of data and desired analyses that may be found in clinical data warehousing applications [37, 38] and thus have some basis in practice. Fourth, we find that they have relevance beyond medical applications.

2.3. Existing Multidimensional Models

We proceed to evaluate fourteen data models for data warehousing on the requirements just presented. We consider the models of Rafanelli & Shoshani [43], Agrawal et al. [2], Gray et al. [19], Dyreson [14], Kimball [26], Li & Wang [31], Gyssens & Lakshmanan [21], Cabbibo and Torlone [7], Datta & Thomas [13], Lehner [29], Vassiliadis [52], Jagadish et al. [23], Mendelzon & Vaisman [33], and Microsoft's OLE DB for OLAP standard [34]. These models are representative of the current state of the art in both the research community and commercial systems. The models can be divided into *simple cube models*, *structured cube models*, and *statistical object models.*

The simple cube models [13, 19, 21, 26] 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 "Address," "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 Addresses rolls up to County and not the other way around. The simple cube models include Star schema designs [26].

Kimball's star schema model is based on plain SQL and does not embody multidimensional concepts per se. We include it here because it is the most widely used implementation model

for multidimensional databases. Additionally, most relational OLAP tools assume a star schema structure of the database and cannot handle more complex designs. Thus, the evaluation of the star schema model is based on what can be achieved using a plain star schema design and the corresponding (simple) SQL queries, not on what can be done using full-fledged SQL.

The structured cube models [2, 7, 14, 23, 29, 31, 33, 34, 52] 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 [28]. The hierarchies are captured using either *grouping relations* [31], *dimension merging functions* [2], measure graphs [14], roll-up functions [7, 33], level lattices [52], hierarchy schemas and instances [23], or an explicit tree-structured hierarchy as part of the cube [29, 34].

The last group of models is the *statistical object models* [43]. 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, using aggregation semantics, against getting query results that are incorrect or not meaningful to the user.

The results of evaluating the fourteen data models against our eleven 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 has not been possible 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	10	11
Rafanelli & Shoshani [43]	√	-	√	p	p	-	-	-	-	-	-
Agrawal et al. [2]	p	√	-	p	-	-	√	-	-	-	-
Dyreson [14]	√	√	p	-	-	-	-	-	-	p	p
Gray et al. [19]	-	√	p	-	-	-	√	-	-	-	-
Kimball [26]	-	√	p	-	-	-	-	-	p	-	-
Li & Wang [31]	p	√	p	-	-	-	-	-	-	-	-
Gyssens & Lakshmanan [21]	-	√	p	-	-	-	√	-	-	-	-
Cabbibo & Torlone [7]	√	√	p	-	-	-	-	-	-	-	-
Datta & Thomas [13]	-	√	-	p	-	-	√	-	-	-	-
Lehner [29]	√	-	√	-	-	-	-	-	-	-	-
Vassiliadis [52]	√	√	√	-	-	-	-	-	-	-	-
Jagadish et al. [23]	√	√	-	-	√	√	-	-	-	√	p
Mendelzon & Vaisman [33]	√	√	p	-	-	-	-	-	√	-	-
MS OLE DB for OLAP [34]	√	√	p	-	-	-	-	-	-	-	-

Table 2: Evaluation of the Data Models

1. *Explicit hierarchies in dimensions*: The simple cube models [13, 19, 21, 26] do not capture the hierarchies in the dimensions explicitly. Some models provide partial support by the *grouping relation* [31] and *dimension merging function* [2], but do not capture the complete hierarchy together with the cube. This is done by the remaining models [7, 14, 23, 29, 33, 34, 43, 52], thus capturing the full cube navigation semantics in the schema.

2. *Multiple hierarchies in each dimension*: Some models [29, 43] require that the schema of dimension hierarchies is tree-structured. To support multiple hierarchies, a more general lattice structure is required. All the other models [2, 7, 13, 14, 19, 21, 23, 26, 31, 33, 34, 52] allow multiple hierarchies.

3. *Support for aggregation semantics*: Most of the models [7, 14, 19, 21, 26, 31, 33, 34] support aggregation semantics partially, by implicitly requiring the dimension hierarchies to be *strict*,

onto, and *covering*, i.e., the hierarchies should be balanced trees. This is one of the conditions of summarizability [30] and means that data will not be double-counted. Two of the models allow for non-strict hierarchies, while not addressing the issue of double-counting, thus providing no support [2, 13]. One model [23] allows non-onto and non-covering hierarchies but does not address the issue of data not being counted, thus providing no support. Two models [29, 43] place explicit conditions on both the hierarchy (strict, onto, and covering) and the aggregation functions used (only additive data may be added, etc.), thus providing full support for aggregation semantics. One model [52] provides the support by always keeping a reference to the base data and computing from that when the aggregation semantics indicate the need to do so.

4. *Non-strict hierarchies*: Most of the models [7, 14, 19, 21, 23, 26, 29, 31, 33, 34, 52] implicitly or explicitly require that hierarchies be strict. Two models [2, 13] mention briefly that non-strict hierarchies are allowed, but do not explore the issues raised by allowing this, e.g., the possibility of double-counting and the use of pre-computed aggregates. The remaining model [43] investigates the possible problems with allowing non-strict hierarchies and advises against using this feature.

5. *Non-onto hierarchies*: One model [43] discusses the possibility of having non-onto hierarchies, but advises against using this feature. One model [23] allows non-onto hierarchies. All the other models do not allow non-onto hierarchies.

6. *Non-covering hierarchies*: Only one model [23] allows hierarchies to be non-covering. All the other models disallow non-covering hierarchies.

7. *Symmetric treatment of dimensions and measures*: Most of the models [7, 14, 23, 26, 29, 31, 33, 34, 43, 52] 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 models [2, 13, 19, 21] do not impose this restriction. They either do not distinguish between measures and dimensions [19, 21], or they allow for the conversion of measures to dimensions and vice versa [2, 13].

8. *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.

9. *Handling change and time*: Two models [26, 33] discuss this issue, but only the model of Mendelzon & Vaisman [33] fully supports analysis across temporal changes in the dimensions. None of the other models support analysis across changes, although one mentions this as a very important issue [29].

10. *Handling different levels of granularity*: Dyreson [14] specifies an *incomplete data cube* to be a union of *cubettes*. Each cubette may have a different data granularity, thus providing some support for different levels of granularity. However, the granularity is fixed at the schema level, rather than at the data level, so the support is only partial. One model [23] allows the granularity to vary at the data level. None of the other models handle different levels of granularity in the data.

11. *Handling imprecision*: For the reasons mentioned above, two models [14, 23] provide partial support for imprecision in the data, as varying granularities can provide a basis for handling imprecision. However, these models do not offer all the features necessary for handling the imprecision. None of the other models provide explicit means for handling imprecise data.

To conclude, the models generally provide full or partial support for most of requirements 1–3. Requirement 4 (non-strict hierarchies) is partially supported by three of the models, while requirement 5 (non-onto) is supported, partially or fully, by only two models. Requirement 7

(symmetric treatment of dimensions and measures) is supported by four models. Requirement 9 (handling change and time) is supported by Mendelzon & Vaisman [33] and partially supported by Kimball [26]. Requirements 10 and 11 (handling different levels of granularity and imprecision) are only partially supported by Dyreson [14] and Jagadish et al. [23]. Requirement 6 is only supported by Jagadish et al. [23]. Requirement 8 (many-to-many fact-dimension relationships) is not supported by any of the models. The objective of the model presented next is to support all eleven requirements.

2.4. Related Work on Imprecision

The area of "imperfect information" has attracted much attention in the scientific literature [35]. We have previously compiled a bibliography on uncertainty management [15] that describes the various approaches to the problem. Considering the amount of previous work in the area, surprisingly little work has addressed the problem of *aggregation of imprecise data*, which is an important aspect of this paper's contribution. Aggregation of imprecise data has been examined in the context of both possibilistic (fuzzy) databases [48] and (to a lesser extent in) probabilistic databases [18], but not to date in a data warehousing or multidimensional model. Statistical techniques have also been applied to the problem of managing uncertain information in databases [53], and in this paper, we similarly use tools from statistics to handle imprecise aggregate data.

The approach presented in this paper aims to maximally reuse existing concepts from multidimensional databases to support imprecise data. In particular, query processing techniques, such as *pre-aggregation* for handling the imprecision are reused. This yields an effective and practical solution that can be implemented using current technology. It is shown how to test if the underlying data is *precise enough* to give a precise result to a query; and if not, an *alternative query* is suggested, that can be answered precisely. If the user accepts an imprecise result, the imprecision is handled as well in the grouping of data as in the actual aggregate computation.

A number of approaches to imprecision exist that allow us to characterize this paper's contribution. It is common to distinguish between *imprecision*, which is a property of the *content* of an attribute value, and *uncertainty*, which concerns the *degree of truth* associated with an attribute value, e.g., it is 100% certain that the patient's age is in the (imprecise) range 20–30 vs. it is only 85% certain that the patient's age is (precisely) 25. Our work concerns only imprecision. The most basic form of imprecision is *missing* or *applicable null* values [11], which allow unknown data to be captured explicitly.

Multiple imputation [6, 46] is a technique from statistics, where multiple values are *imputed*, i.e., substituted, for missing values, allowing data with some missing values to be used for analysis, while retaining the natural variance in the data. In comparison with our approach, multiple imputation handles only *missing* values, not imprecise values, and the technique does not support efficient query processing using pre-aggregated data.

The concept of null values has been generalized to *partial values*, where one of a set of possible values is the true value. Work has been done on aggregation over partial values in relational databases [8]. Compared to our approach, the time complexity of the operations is quite high, i.e., at least $O(n^{5/2})$, where n is the number of tuples, compared to the $O(n \log n)$ complexity of our solution. Additionally, all values in a partial value have the same weight, and the use of pre-aggregated data is not studied.

Fuzzy sets [58] allows a *degree of membership* to be associated with a value in a set, and can be used to handle both uncertain and imprecise information. The work on aggregation over fuzzy sets in relational databases [47, 48] allows the handling of imprecision in aggregation operations. However, the time complexity is very high, i.e., exponential in the number of tuples, and the issue of pre-aggregation has not been studied.

The concept of *granularities* [4] has been used extensively in temporal databases for a variety of purposes, including the handling of imprecision in the data [16]. However, aggregation of imprecise temporal data remains to be studied. In the area of multidimensional databases, only the work on *incomplete data cubes* [14] has addressed the issue of handling imprecise information. Compared to this paper's approach, the incomplete data cubes have the granularity of the data fixed at schema

level, rather than the instance level. Additionally, imprecision is only handled for the grouping of data, not for the aggregate computation.

To our knowledge, imprecision in the actual aggregate result for multidimensional databases has not been supported previously; and in general, no one has studied the use of pre-aggregated data for speeding up query processing involving imprecision. Also, the consequent use of the multidimensional concept of granularities in all parts of the approach, we believe is novel.

3. AN EXTENDED MULTIDIMENSIONAL DATA MODEL

In this section, our multidimensional data model [39] is developed in detail. The basic elements of the model are presented first. The basic model is then extended to handle change over time. Finally, several important properties of the model are presented. For every part of the model, we define the *intension*, the *extension*, and give an illustrating example.

3.1. The Basic Model

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

Example 1 In the case study from Section 2.1 we will have *Patient* as the fact type, and *Diagnosis, Residence, Age, Date of Birth* (DOB), *Name*, and *HbA1c%* 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 $(\mathcal{C}, \sqsubseteq_\mathcal{T}, \top_\mathcal{T}, \bot_\mathcal{T})$, where $\mathcal{C} = \{\mathcal{C}_j, j = 1, .., k\}$ are the *category types* of $\mathcal{T}$, $\sqsubseteq_\mathcal{T}$ is a partial order on the $\mathcal{C}_j$'s, with $\top_\mathcal{T} \in \mathcal{C}$ and $\bot_\mathcal{T} \in \mathcal{C}$ being the top and bottom element of the ordering, respectively. Thus, the category types form a lattice. The intuition is that one category type is "greater than" another category type if members of the former's extension logically contain members of the latter's extension, i.e., they have a larger element size. The top element of the ordering corresponds to the largest possible element size, that is, there is only one element in it's extension, logically containing all other elements. We say that $\mathcal{C}_j$ *is a category type of* $\mathcal{T}$, written $\mathcal{C}_j \in \mathcal{T}$, if $\mathcal{C}_j \in \mathcal{C}$. We assume a function $Pred : \mathcal{C} \mapsto 2^\mathcal{C}$ that gives the set of immediate predecessors of a category type $\mathcal{C}_j$.

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

Many types of data, e.g., ages or sales amounts, can be added together to produce meaningful results. This data has an ordering on it, so computing the average, minimum, and maximum values make sense. For other types of data, e.g., dates of birth or inventory levels, the user may not find it meaningful in the given context to add them together. However, the data has an ordering on it, so taking the average, or computing the maximum or minimum values do make sense. Some types of data, e.g., diagnoses, have no meaningful ordering, 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 the aggregation semantics of the 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 [29, 42], 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 $\Sigma = \{\text{SUM, COUNT, AVG, MIN, MAX}\}$, $\phi = \{\text{COUNT, AVG, MIN, MAX}\}$, and $c = \{\text{COUNT}\}$. 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 $\mathcal{T} = (\mathcal{C}, \sqsubseteq_{\mathcal{T}})$, we assume a function $Aggtype_{\mathcal{T}} : \mathcal{C} \mapsto \{\Sigma, \phi, c\}$ that gives the aggregation type for each category type.

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

A *dimension* D of type $\mathcal{T} = (\{\mathcal{C}_j\}, \sqsubseteq_{\mathcal{T}}, \top_{\mathcal{T}}, \bot_{\mathcal{T}})$ is a two-tuple $D = (C, \sqsubseteq)$, where $C = \{C_j\}$ is a set of *categories* C_j such that $Type(C_j) = \mathcal{C}_j$ and $\sqsubseteq$ is a partial order on $\cup_j C_j$, the union of all dimension values in the individual categories. A category C_j of type $\mathcal{C}_j$ is a set of *dimension values* e such that $Type(e) = \mathcal{C}_j$.

The definition of the partial order is: given two values e_1, e_2 then $e_1 \sqsubseteq e_2$ if e_1 is logically contained in e_2. We say that C_j is a category of D, written $C_j \in D$, if $C_j \in C$. For a dimension value e, we say that e is a dimensional value of D, written $e \in D$, if $e \in \cup_j C_j$.

We assume that the partial order on category types and the function *Pred* work directly on categories, with the order given by the corresponding category types. The category $\bot_{\mathcal{T}}$ in a dimension D contains the values with the smallest value size. The category with the largest value size, $\top_D$, contains exactly one value, denoted $\top$. For all values e of the categories of D, $e \sqsubseteq \top$. Value $\top$ is similar to the *ALL* construct of Gray et al. [19].

Example 4 In our Diagnosis dimension we have the following categories, named by their type. Low-level Diagnosis = $\{3, 5, 6\}$, Diagnosis Family = $\{4, 7, 8, 9, 10, 14\}$, Diagnosis Group = $\{11, 12, 13\}$, and $\top_{\text{Diagnosis}} = \{\top\}$. The values in the sets refer to the *ID* field in the Diagnosis table of Table 1. The partial order $\sqsubseteq$ 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 distinguish between the dimension and category types versus the actual dimensions and categories to allow several dimensions or categories to have the same type. This provides a way to ensure that two dimensions or categories are *type compatible*. Type compatibility is useful for ensuring meaningful results for several types of operations, as described in Example 5 below. If the type compatibility feature is not needed for a concrete application of the model, the model may be simplified by removing category and dimension types altogether, specifying hierarchies directly on categories instead.

Example 5 Suppose we have an "Order Time" and a "Shipment Time" dimension, both of the dimension type "Gregorian Calendar." We want to take the union of those two dimensions (and the categories in them) to produce a combined "Time" dimension with both types of times in it. This should be allowed since the dimensions and categories have the same types. However, if we have a "Local Order Time" dimension of type "Islamic Calendar" (since that calendar is used in the country the data comes from) with the same structure (days, months, quarter, years), we should not be allowed to take the union of that and the "Time" dimension since their types are different, meaning that they are incomparable and thus should not be mixed. For example, the year 2000 AD in the Gregorian Calendar overlaps the years 1420 AH and 1421 AH in the Islamic Calendar. Another example would be to subtract Order Times from Shipment Times, e.g., when performing aggregations on the data to get the numbers of days to it takes to ship orders. This should be allowed since the two categories have the same type. In contrast, it does not make sense to subtract Local Shipment Time from Order Time. The union and aggregation operators are discussed in detail in Section 4.

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

Example 6 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 $\top_{\text{Diagnosis}}$.

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 a 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 7 A diagnosis value has two representations, *Code* and *Text*. Using the ID's from the Diagnosis table to identify the values, we have $Code(3)$ ="P11" and $Text(3)$ ="Diabetes during pregnancy"

Let F be a set of facts, and $D = (\{C_j\}, \sqsubseteq)$ 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 $\exists e_1 \in D\ ((f, e_1) \in R \wedge e_1 \sqsubseteq e)$.

Note that facts can be characterized by dimension values in several dimension categories, e.g., the fact "John Doe" is characterized by the values "Non insulin dependent diabetes" and "Diabetes." The intuition is that a fact is characterized by all dimension values that are either directly (through a fact-dimension relation) or indirectly related to it (the value is an ancestor of a directly related value).

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 8 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 = {(0,11),(1,10), (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 10, which belongs to the *Diagnosis Family* category. Thus, we do not require that e belongs to $\bot_{\text{Diagnosis}}$, as do most of the existing data models. If no diagnosis is known for patient 1, we would have added the pair $(1, \top)$ to R.

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

Example 9 For the case study, we get a six-dimensional MO, $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S}$ = (Patient, {Diagnosis, DOB, Residence, Name, Age, HbA1c%}) and $F = \{0, 1, 2\}$. 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 dimension is simple, i.e., it just has a $\bot$ category type, Name, 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. The Residence dimension groups addresses into cities or counties, and cities into counties. The HbA1c% dimension has the categories Precise, Imprecise, and $\top_{\text{HbA1c\%}}$. The Precise category has values with one decimal point, e.g., "5.5", as members, while the Imprecise category has integer values. The values of both categories fall in the range

2–12. The partial order on the HbA1c% dimension groups the precise values into the imprecise in the natural way, e.g., $4.5 \sqsubseteq 5$ and $5.4 \sqsubseteq 5$ (note that $\sqsubseteq$ denotes logical inclusion). 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.

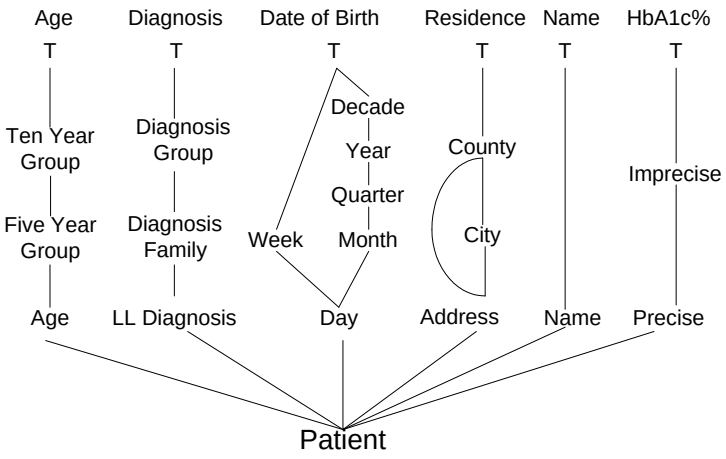

Fig. 2: Schema of the Case Study

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

Example 10 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 $\top_{\text{Diagnosis}}$, and a subdimension of the Age dimension that includes only Ten Year Group and $\top_{\text{Age}}$. 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 handle the imprecision, we need an additional definition.

For a dimension value e such that $e \in C_j$, we say that the *granularity* of e is C_j. For a fact f such that $(f, e) \in R_i$ and $e \in C_j$, we say that the *granularity* of f in dimension i is C_j. Dimension values in the $\perp$ category are said to have the *finest* granularity, while the value in the $\top$ category has the *coarsest* granularity.

For dimension $D = (C, \sqsubseteq)$, we assume a function $G_D : D \mapsto C$, that gives the granularity of dimension values. For an MO $M = (\mathcal{S}, F, D, R)$, where $D_i = (C_i, \sqsubseteq_i)$, we assume a family of functions $G_{F_i} : F \mapsto C_i,\ i = 1, .., n$, each giving the granularities of facts in dimension D_i.

To summarize the essence of our model, the facts are objects with *separate identity.* Thus, we can test facts for equality, but we do not assume an ordering on the facts. The combination of the dimension values that characterize the facts in an MO do *not* constitute a "key" for the MO. 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 [17] explicitly assume a discrete model of time). Also the temporal data types offered by the SQL standard [32] 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 [4]. 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 [25]. 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 $\sqsubseteq$ by adding T_v, the set of chronons during which the relationship holds in the real world, to each relationship between two values. We write that $e_1 \sqsubseteq_{T_v} e_2$ if $e_1 \sqsubseteq e_2$ during each chronon in T_v. The partial order $\sqsubseteq_{T_v}$ has the following property: $e_1 \sqsubseteq_{T_{1_v}} e_2 \wedge\ e_2 \sqsubseteq_{T_{2_v}} e_3 \Rightarrow e_1 \sqsubseteq_{T_{1_v} \cap T_{2_v}} e_3$. Similarly, we write $Rep(e) =_{T_v} v$ to denote that the representation Rep of the value e has value v during each chronon in T_v. For each fact-dimension relation between a fact f and a dimension value e, we capture the set of chronons T_v when the two are related. We write $(f, e) \in_{T_v} R$ when $(f, e) \in R$ during each chronon in T_v. We use the notation $f \rightsquigarrow_{T_v} e$ when $(f, e') \in_{T_v} R \wedge e' \sqsubseteq_{T_v} e$. 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" [25]. Thus, we do not have the problem of "value-equivalent" statements [24, 25, 51], where the same statement appears several times with different times attached to it, e.g., $e_1 \sqsubseteq_{T_1} e_2$ and $e_1 \sqsubseteq_{T_2} e_2$, where $T_1 \neq T_2$. However, by implication, statements are valid for any subset of their attached time, e.g., $T_1 \subseteq T_2 \wedge e_1 \sqsubseteq_{T_2} e_2 \Rightarrow e_1 \sqsubseteq_{T_1} e_2$.

Example 11 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 $7 \sqsubseteq_{[01/01/70-31/12/79]} 3$. For the representation, we have $Code(8)_{[01/01/70-31/12/79]} = \text{D1}$. 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 12 If we want to look at the data only from the current point in time, we want to include count data for patients with the "old" diagnoses, i.e., the diagnoses that were valid until 1980, together with data for patients with the new diagnoses. One way to do this is to count the patients diagnosed with the old "Diabetes" diagnosis ($ID = 8$) together with those diagnosed with the new "Diabetes" diagnosis ($ID = 11$) from 1970 to the present. This is done by defining that $8 \sqsubseteq_{[01/01/80-NOW]} 11$, i.e., from 1980 up till now, we consider the diagnosis 8 to be logically contained in the diagnosis 11. Another way of enabling analysis across time would be to introduce a new category for holding diagnoses that are common across time, i.e., we would introduce a new, common "Diabetes" value in this category and define the hierarchy so that the new, common value was a parent of the two values ($ID = 8$) and ($ID = 11$). Thus, both the old and the new "Diabetes" patients would be counted under the new, common "Diabetes" value.

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

Summarizability is an important concept as it is a condition for the flexible use of pre-computed aggregates. Without summarizability, lower-level results generally cannot be directly combined into higher-level results. This means that we cannot choose to pre-compute only a relevant selection of the possible aggregates and then use these to compute higher-level aggregates on-the-fly. Instead, we have to pre-compute the total results for all the aggregations that we need fast answers to, while other aggregates must be computed from the base data. Space and time constraints can be prohibitive for pre-computing all results, while computing aggregates from scratch results in long response times.

It has been shown that summarizability is equivalent to the aggregation function being *distributive*, all paths being *strict*, and the hierarchies being *covering* and *onto* in the relevant dimensions [30]. 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*, *snapshot covering*, and *snapshot onto*. These concepts are formally defined below. In the definitions, we assume a dimension $D = (C, \sqsubseteq)$.

Definition 2 Given two categories, C_1, C_2 such that $C_2 \in Pred(C_1)$, we say that the mapping from C_1 to C_2 is *onto* iff $\forall e_2 \in C_2(\exists e_1 \in C_1\ (e_1 \sqsubseteq e_2))$. Otherwise, it is *non-onto*. If all mappings in a dimension are onto, we say that the dimension hierarchy is *onto*. The hierarchy in D is *snapshot onto* if it is onto at any given time t.

Definition 3 Given three categories, C_1, C_2, C_3 such that $Type(C_1) \sqsubset Type(C_2) \sqsubset Type(C_3)$, we say that the mapping from C_2 to C_3 is *covering with respect to* C_1 iff $\forall e_1 \in C_1\ (\forall e_3 \in C_3\ (e_1 \sqsubseteq e_3 \Rightarrow \exists e_2 \in C_2\ (e_1 \sqsubseteq e_2 \wedge e_2 \sqsubseteq e_3)))$. Otherwise, it is *non-covering with respect to* C_1. If all mappings in a dimension are covering w.r.t. any category, we say that the dimension hierarchy is *covering*. The hierarchy in D is *snapshot covering* if it is covering at any given time t.

Definition 4 If $\forall C_1, C_2 \in C(e_1, e_3 \in C_1 \wedge e_2 \in C_2 \wedge e_2 \sqsubseteq e_1 \wedge e_2 \sqsubseteq e_3 \Rightarrow e_1 = e_3)$ 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 $\forall f \in F : f \rightsquigarrow e_1 \wedge f \rightsquigarrow e_2 \wedge e_1 \in C_j \wedge e_2 \in C_j \Rightarrow e_1 = e_2$[†]. The hierarchy in dimension D is *snapshot strict*, if at any given time t, the hierarchy is strict.

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

Example 13 The hierarchy in the Residence dimension is strict, onto and non-covering (due to the rural addresses). The hierarchy in the Diagnosis dimension is non-strict (due to multiple parent diagnoses), non-onto (the "Lung cancer" diagnosis), and covering. The sub-hierarchy of the Diagnosis dimension obtained by restriction to the standard classification is snapshot strict, snapshot covering, and snapshot non-onto.

4. THE ALGEBRA

This section defines an algebra for multidimensional objects. The presentation first defines the basic algebra. The algebra is then extended to handle time. Examples of the more complicated operators are provided. We have chosen to define an algebra with clean, orthogonal operators that are close to the operators in relational algebra to precisely capture the different types of operations that can be performed on multidimensional data. The algebra is not meant to be a basis for implementation. However, we show how typical OLAP operations can be logically specified in terms of the basic algebraic operators. A comparison of the expressiveness of multidimensional algebras is beyond the scope of this paper and will not be discussed here.

4.1. The Basic Algebra

In this section the fundamental (non-temporal) operators are defined. These operators are similar to the standard relational algebra operators. For unary operators, we assume a single multidimensional object $M = (\mathcal{S}, F, D = \{D_i\}, R = \{R_i\})$, where $i = 1, \ldots, n$. For binary operators we assume two multidimensional objects, $M_1 = (\mathcal{S}_1, F_1, D_1 = \{D_{1_i}\}, R_1 = \{R_{1_i}\})$, $i = 1, \ldots, n$ and M_2, which is similarly defined. We note that the representations of the categories in the resulting MO's are the same as in the argument MO's, so we do not specify the representations for the values in the resulting MO's. The aggregation types are only changed by the aggregate formation operator, so they are not specified for the other operators. Two auxiliary definitions are helpful in several of the operator definitions.

1. The *Group* operator groups the facts characterized by the same dimension values. Given an n-dimensional MO, $M = (\mathcal{S}, F, D = \{D_i\}, R = \{R_i\}), i = 1, .., n$, a set of categories $C = \{C_i \mid C_i \in D_i\}, i = 1, .., n$, one from each of the dimensions of M, and an n-tuple $(e_1, .., e_n)$, where $e_i \in C_i, i = 1, .., n$, we define *Group* as: $Group(e_1, .., e_n) = \{f \mid f \in F \wedge f \leadsto_1 e_1 \wedge .. \wedge f \leadsto_n e_n\}$.

2. The *union* operator on dimensions performs union on the categories and the partial orders. Given two dimensions $D_1 = (C_1, \sqsubseteq_1)$ and $D_2 = (C_2, \sqsubseteq_2)$ of type $\mathcal{T}$, where $C_k = \{C_{kj}\}, k = 1, 2,\ j = 1, .., m$, we define the union operator on dimensions, $\bigcup_D$, as: $D_1 \bigcup_D D_2 = (C', \sqsubseteq')$, where $C' = \{C'_j\}, j = 1, .., m$, $C'_j = C_{1j} \cup C_{2j}$, where $\cup$ denotes regular set union, and $e_1 \sqsubseteq' e_2 \Leftrightarrow e_1 \sqsubseteq_1 e_2 \vee e_1 \sqsubseteq_2 e_2$.

selection:

Given a predicate p on the dimension types $\mathcal{D} = \{\mathcal{T}_i\}$, we define the selection, σ, as: $\sigma[p](M) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = \mathcal{S}$, $F' = \{f \in F \mid \exists e_1 \in D_1, .., e_n \in D_n\ (\ p(e_1, .., e_n) \wedge f \leadsto_1 e_1 \wedge .. \wedge f \leadsto_n e_n)\}$, $D' = D$, $R' = \{R'_i\}$, 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 $Name =$"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., $R_2 = \{(1, 10)\}$, and the dimension $D' = D$.

projection:

Without loss of generality, we assume that the projection is over the k dimensions $D_1, .., D_k$. We then define the projection, π, as: $\pi[D_1, .., D_k](M) = (\mathcal{S}', F', D', R')$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = \mathcal{F}$, $\mathcal{D}' = \{\mathcal{T}_1, .., \mathcal{T}_k\}$, $F' = F$, $D' = \{D_1, .., D_k\}$, and $R' = \{R_1, .., R_k\}$. Thus, we retain only the k 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 in Figure 3.

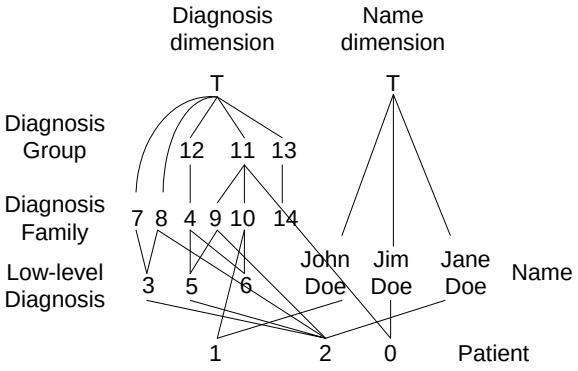

Fig. 3: Resulting MO for Projection

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

union:

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

difference:

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

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 $F = \{0, 1\}$, the dimensions from the first argument, and the fact-dimension relations $R_1 = \{(0, 11), (1, 10)\}$ and $R_2 = \{(0, \text{Jim Doe}), (1, \text{John Doe})\}$.

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 $(\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = \mathcal{F}_1 \times \mathcal{F}_2$, $\mathcal{D}' = \mathcal{D}_1 \cup \mathcal{D}_2$, $F' = \{(f_1, f_2) \mid f_1 \in F_1 \wedge\ f_2 \in F_2 \wedge p(f_1, f_2)\}$, $D' = D_1 \cup D_2$, $R' = \{R'_i, i = 1, .., n_1 + n_2\}$, and $R'_i = \{(f', e) | f' = (f_1, f_2) \wedge f' \in F' \wedge ((i \leq n_1 \wedge (f_1, e) \in R_{1_i}) \vee (i > n_1 \wedge (f_2, e) \in R_{2_{i-n_1}}))\}$. In other 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 example, if p is $f_1 = f_2$ an *equi-join* results, whereas if p is *true* a *Cartesian product* is implemented. 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 a value if one member of the pair was previously related to that value.

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 $F = \{0, 1, 2\}$, and the contents of the two dimensions are identical. The fact-dimension relations are also identical: $R_1 = \{(0, \text{Jim Doe}), (1, \text{John Doe}), (2, \text{Jane Doe})\}$ and $R_2 = \{(0, \text{Jim Doe}), (1, \text{John Doe}), (2, \text{Jane Doe})\}$. 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 [27], we assume the existence of a *family* of aggregation functions g that take some k-dimensional subset $\{D_{i_1}, .., D_{i_k}\}$ of the n dimensions as arguments, e.g., SUM_i sums the i-th dimension and SUM_{ij} sums the i-th and j-th dimensions. We assume a function $Args(g) = \{j \mid g$ uses dimension j 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, $g : 2^F \mapsto D_{n+1}$ (the function g "looks up" the required data for the facts in the relevant fact-dimension relations, e.g., SUM_i finds its data in fact-dimension relation R_i) such that $g \in MIN\{Aggtype(\perp_{D_{i_j}}), j \in Args(g)\}$, and a set of categories $C_i \in D_i, i = 1, .., n$, we define aggregate formation, α, as follows.

$$\alpha[D_{n+1}, g, C_1, .., C_n](M) = (\mathcal{S}', F', D', R'),$$

where

$$\mathcal{S}' = (\mathcal{F}', \mathcal{D}'), \mathcal{F}' = 2^{\mathcal{F}}, \mathcal{D}' = \{\mathcal{T}'_i, i = 1, .., n\} \cup \{\mathcal{T}_{n+1}\}, \mathcal{T}'_i = (\mathcal{C}'_i, \sqsubseteq'_{\mathcal{T}_i}, \perp'_{\mathcal{T}_i}, \top'_{\mathcal{T}_i}),$$

$$\mathcal{C}'_i = \{\mathcal{C}_{ij} \in \mathcal{T}_i \mid Type(C_i) \sqsubseteq_{\mathcal{T}_i} \mathcal{C}_{ij}\}, \sqsubseteq'_{\mathcal{T}_i} = \sqsubseteq_{\mathcal{T}_i|_{\mathcal{C}'_i}}, \perp'_{\mathcal{T}_i} = Type(C_i), \top'_{\mathcal{T}_i} = \top_{\mathcal{T}_i},$$

$$F' = \{Group(e_1, .., e_n) \mid (e_1, .., e_n) \in C_1 \times .. \times C_n \wedge Group(e_1, .., e_n) \neq \emptyset\},$$

$$D' = \{D'_i, i = 1, .., n\} \cup \{D_{n+1}\}, D'_i = (C'_i, \sqsubseteq'_i), C'_i = \{C'_{ij} \in D_i \mid Type(C'_{ij}) \in C'_i\},$$

$$\sqsubseteq'_i = \sqsubseteq_{i|_{D'_i}}, R' = \{R'_i, i = 1, .., n\} \cup \{R'_{n+1}\},$$

$$R'_i = \{(f', e'_i) \mid \exists (e_1, .., e_n) \in C_1 \times .. \times C_n\ (f' = Group(e_1, .., e_n) \wedge f' \in F' \wedge e_i = e'_i)\},\ \text{and}$$

$$R'_{n+1} = \cup_{(e_1,..,e_n) \in C_1 \times .. \times C_n} \{(Group(e_1,..,e_n), g(Group(e_1,..,e_n))) \mid Group(e_1,..,e_n) \neq \emptyset\}.$$

The aggregation types for the remaining parts of the argument dimensions are unchanged. The aggregation types for the result dimension is given by the following rule. If g is distributive, the paths to $C_1,..,C_n$ are strict, and the hierarchies up to $C_1,..,C_n$ are onto and covering, then $Aggtype(\perp_{D_{n+1}}) = MIN\{Aggtype(\perp_{D_j}), j \in Args(g)\}$. Otherwise, $Aggtype(\perp_{D_{n+1}}) = c$, i.e., no further aggregation is allowed. For the higher categories in the result dimension, $Aggtype(C'_m) = MIN\{Aggtype(C_m), Aggtype(\perp_{D_{n+1}})\}$.

Thus, for every combination $(e_1,..,e_n)$ of dimension values in the given "grouping" categories, we apply g to the set of facts $\{f\}$, where the f's are characterized by $(e_1,..,e_n)$, and place the result in the new dimension D_{n+1}. The new 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 the original facts*, where the original 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 onto and covering, 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," meaning 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 $F = \{0, 1, 2\}$. 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: "0–2" and ">2". The fact-dimension relation for the Diagnosis dimension links the sets of patients to their corresponding Diagnosis Group. The content is: $R_1 = \{(\{0, 1, 2\}, 11), (\{2\}, 12)\}$, meaning that the sets of patients $\{0, 1, 2\}$ 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: $R_7 = \{(\{0, 1, 2\}, 3), (\{2, \}, 1)\}$, meaning that the results of g on the sets $\{0, 1, 2\}$ and $\{2\}$ are 3 and 1, respectively. A graphical illustration of the MO, leaving out the trivial dimensions for simplicity, is seen in Figure 4. Note that each patient is only counted once for each diagnosis group, even though patient 2 has *several* diagnoses in each group.

We proceed to 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.

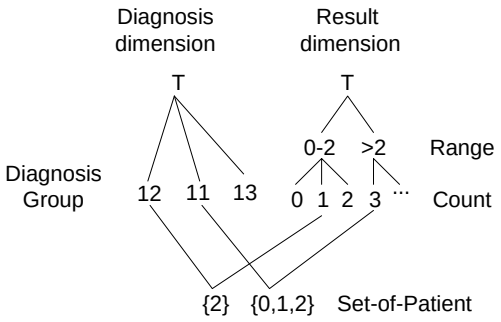

Fig. 4: Resulting MO for Aggregate Formation

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 categories[†] and the given function. The dimensions not in the "GROUP BY" clause are then projected out.

star-join

A star-join as described in [26] 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 gives "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 gives "less detail" by ascending dimension hierarchies, aggregating using 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* [27].

[†]The categories not in the "GROUP BY" clause are the $\top$ categories of their dimensions.

Proof: A relation r with schema $S_r = (a_1, .., a_n)$ is mapped to an n-dimensional MO $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (r, \{\mathcal{T}_i, i = 1, .., n\})$, $\mathcal{T}_i = (\{a_i, \top_{\mathcal{T}_i}\}, \sqsubseteq_i, \top_{\mathcal{T}_i}, a_i)$, $a_i \sqsubseteq_i a_i, a_i \sqsubseteq_i \top_{\mathcal{T}_i}$, $\top_{\mathcal{T}_i} \sqsubseteq_i \top_{\mathcal{T}_i}$, $F = \{(v_1, .., v_n) \in r\}$, $D = \{D_i\}, i = 1, .., n$, $D_i = (\{A_i, \top_i\}, \sqsubseteq)$, $A_i = Dom(a_i)$, $\forall v \in A_i : v \sqsubseteq \top$, $R = \{R_i\}, i = 1, .., n$, and $R_i = \{((v_1, .., v_i, .., v_n), v_i) \mid (v_1, .., v_i, .., v_n) \in r\}$. Thus, an n-ary relation is mapped to an MO with n "flat" dimensions, each containing the domain of the corresponding attribute. 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 appears 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 it appears at any given point in time in the real world, we introduce the *valid-timeslice operator* [25].

valid-timeslice operator

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

To support analysis related to time, we allow predicates p and functions g, to be used in selections and aggregate formations that refer to time. We will not go deeper into the structure of temporal predicates and functions since details are available elsewhere (cf., TSQL2 [51]).

The last step is to define how the basic algebraic operators deal 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 rules[†]. $(f, e) \in_{T_1} R_{1_i} \wedge (f, e) \in_{T_2} R_{2_i} \Rightarrow (f, e) \in_{T_1 \cup T_2} R_i'$, $e_1 \sqsubseteq_{1_{T_1}} e_2 \wedge e_1 \sqsubseteq_{2_{T_2}} e_2 \Rightarrow e_1 \sqsubseteq'_{T_1 \cup T_2} e_2$, $Rep_1(e) =_{T_1} v \wedge Rep_2(e) =_{T_2} v \Rightarrow Rep'(e) =_{T_1 \cup T_2} v$, $e \in_{1_{T_1}} C_j \wedge e \in_{2_{T_2}} C_j \Rightarrow e \in'_{T_1 \cup T_2} C_j$. 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. $(f, e) \in_{T_1} R_{i_1} \wedge (f, e) \in_{T_2} R_{i_2} \wedge T_1 \setminus T_2 \neq \emptyset \Rightarrow (f, e) \in_{T_1 \setminus T_2} R_i'$, $F' = \bigcap_{i=1,..,n} \{f \mid \exists (f, e_i) \in R_i' \ ((f, e_i) \in_{T'} R_i' \wedge T' \neq \emptyset)\}$. Thus, the time for a pair in a fact-dimension relation for the first MO is cut by the time that the corresponding pair has in the fact-dimension relation for the second MO. Only pairs with non-empty chronon sets are retained. The facts in the resulting MO are those that participate in all the resulting fact-dimension relations during a non-empty set of chronons. As in the non-temporal case, we do not alter the dimensions of the first MO.

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

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

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

5. HANDLING IMPRECISION

We now describe our approach to handling imprecision in multidimensional data models [40]. We start by giving an overview of the approach, and then describe how *alternative queries* may be used when the data is not precise enough to answer queries precisely, i.e., when the data used to group on is registered at granularities coarser than the "grouping" categories. The approach proposed here extends the data model defined in Section 3 to additionally capture imprecision. We assume that the chronons attached to the MOs are precise, e.g., the time when data is loaded into a data warehouse. With this assumption, the handling of imprecision works seamlessly together with the handling of time. We will not deal with the separate subject of imprecise time in this paper, as this has been covered extensively in previous research [15, 16].

5.1. Overview of Approach

When the model definition was presented, it was also described how the case study would be handled using the model. As part of this, it was shown how imprecision could be handled, namely by mapping facts to dimension values of *coarser granularities* when the information was imprecise, e.g., the mapping to $\top$ when the diagnosis is unknown. The HbA1c% dimension generalizes this approach, as several *precise* measurement values are contained in one *imprecise* measurement value. In turn, several imprecise values are contained in the $\top$ (unknown) value. Thus, the approach uses the different levels of the granularities already available in multidimensional data models to also capture imprecision in a general way. An illustration of the approach, showing how the possible spectrum of imprecision in the data is captured using categories in a dimension, is seen in Figure 5.

The approach has a nice property, provided directly by the dimensional "imprecision" hierarchy described above. When the data is *precise enough* to answer a query, the answer is obtained immediately, even though the underlying facts may have *varying* granularities. For example, the query from Example 18 gives us the number of patients diagnosed with diagnoses in the Diabetes diagnosis group, even though two of the patients are diagnosed with diagnosis families, while one is diagnosed directly with the Diabetes diagnosis group. In this case, the data would *not* be precise enough to group the patients by Diagnosis Family.

Our general approach to handling a query starts by *testing if the data is precise enough* to answer the query, in which case the query can be answered directly. Otherwise, an *alternative query* is suggested. In the alternative query, the categories used for grouping are *coarsened* to fit the imprecision of the data. Thus, the alternative query will give the most *precise* answer possible,

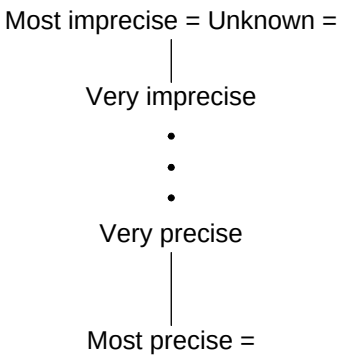

Fig. 5: The Spectrum of Imprecision

considering the imprecision in the data. For example, if a physician asks for the patient count grouped by diagnosis family, but some of the data is known only at the diagnosis group level, then an alternative query would be the patient count grouped by diagnosis group.

Other strategies for handling the imprecision are also available. If the physician still wants to go ahead with the original query, the imprecision should be handled explicitly. An analysis of the algebra shows that imprecision in the data will only affect the results of two operators, namely *selection* and *aggregate formation* (the join operator tests only for equality on *fact identities*, which are not subject to imprecision). Thus, imprecision support is needed only for these two operators; the other operators will just "pass on" the results containing imprecision untouched. However, if we can handle imprecision in the *grouping* of facts, ordinary OLAP style "slicing/dicing" selection is also handled since slicing/dicing is just selection of data for one of a set of groups. Because our focus is on OLAP functionality, we will not go into the more general problem of imprecision in selections, but refer to the existing literature [35].

Following this reasoning, the general query that we must consider is $\alpha[C_1, .., C_n, D_{n+1}, g](M)$, where M is an n-dimensional MO, $C_1, .., C_n$ are the "grouping" categories, D_{n+1} is the result dimension, and g is the aggregation function. The query can be evaluated as follows. First, facts are grouped according to the dimension values in the categories $C_1, .., C_n$ that characterize them. Second, the aggregate function g is applied to the facts in each group, yielding an "aggregate result" dimension value in the result dimension for each group. The evaluation approach is given by the pseudo-code below. Comments begin with "%".

```
Procedure EvalImprecise(Q,M)          % Q is a query, M is an MO.
if PreciseEnough(Q,M) then Eval(Q,M)  % if data is precise enough, use normal evaluation
else
    Q' = Alternative(Q,M)             % suggest alternative query
    if Q' is accepted then Eval(Q',M) % use normal evaluation for alternative query
    else
        Handle Imprecision in Grouping for Q
        Handle Imprecision in Aggregate Computation for Q
        Return Imprecise Result of Q
    end if
end if
```

Our overall approach to handling the imprecision in all phases will be to use the *granularity* of the data, or measures thereof, to represent the imprecision in the data. This enables simple and efficient handling of imprecision.

5.2. Alternative Queries

The first step in the evaluation of a query is to test whether the underlying data is *precise enough* to answer the query. This means that all facts in the MO must be linked to categories that are "less-than-or-equal" to the "grouping" categories in the query, e.g., if we want to group by Diagnosis Family, all fact-dimension relations from patients to the Diagnosis dimension must map to the Diagnosis Family category or lower, not to Diagnosis Group or $\top$.

In order to perform the test for data precision, we need to know the granularities of the data in the different dimensions. For this, for each MO, M, we maintain a separate *precision MO*, M_p. The precision MO has the same number of dimensions as the original MO. For each dimension in the original MO, the precision MO has a corresponding "granularity" dimension. The i'th granularity dimension has only two categories, $\text{Granularity}_\text{i}$ and $\top_{p_i}$. There is one *value* in a "Granularity" category for each category in the corresponding dimension in M. The set of facts F is the same as in M, and the fact-dimension relations for M_p map a fact f to the dimension value corresponding to the category that f was mapped to in M. The determination of whether a given query can be answered precisely is dependent on the actual data in the MO, and this may change when the data in the MO is changed. Thus, we need to update the precision MO along with the original MO when data changes.

Formally, given an MO, $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (\mathcal{F}, \mathcal{D})$, $\mathcal{D} = \{\mathcal{T}_i, i = 1, ..n\}$, $\mathcal{T}_i = (\mathcal{C}_i, \sqsubseteq_{\mathcal{T}_i})$, $\mathcal{C}_i = \{\mathcal{C}_{ij}\}$, $D = \{D_i, i = 1, .., n\}$, and $R_p = \{R_{p_i}, i = 1, .., n\}$, we define the *precision MO*, M_p, as:

$$M_p = (\mathcal{S}_p, F_p, D_p, R_p),$$

where

$$\mathcal{S}_p = (\mathcal{F}_p, \mathcal{D}_p), \mathcal{F}_p = \mathcal{F}, \mathcal{D}_p = \{\mathcal{T}_{p_i}, i = 1, .., n\}, \mathcal{T}_{p_i} = \{\text{Granularity}_\text{i}, \top_{\text{p}_\text{i}}\},$$

$$F_p = F, D_p = \{D_{p_i}, i = 1, .., n\}, D_{p_i} = (C_{p_i}, \sqsubseteq_{p_i}), C_{p_i} = \{\text{Granularity}_\text{i}, \top_{\text{p}_\text{i}}\},$$

$$\text{Granularity}_\text{i} = \{\text{G}_{\text{D}_\text{i}}(\text{e}) \mid \text{e} \in \text{D}_\text{i}\}, \top_{\text{p}_\text{i}} = \{\top_\text{i}\},$$

$$e_1 \sqsubseteq_{p_i} e_2 \Leftrightarrow (e_1 = e_2) \vee (e_1 \in \text{Granularity}_\text{i} \wedge \text{e}_2 = \top_\text{i}, \text{ and}$$

$$R_{p_i} = \{(f, G_{D_i}(e)) \mid (f, e) \in R_i\}$$

Example 19 The MO from Example 9 has the precision MO $M_p = (\mathcal{S}_p, F_p, D_p, R_p)$, where the schema $\mathcal{S}_p$ has the fact type Patient and the dimension types $\text{Gran}_\text{Diagnosis}$ and $\text{Gran}_\text{HbA1c\%}$.[†] The dimension type $\text{Gran}_\text{Diagnosis}$ has the category types $\text{Granularity}_\text{Diagnosis}$ and $\top_\text{GranDiagnosis}$. The dimension type $\text{Gran}_\text{HbA1c\%}$ has the category types $\text{Granularity}_\text{HbA1c\%}$ and $\top_\text{GranHbA1c\%}$. The set of facts is the same, namely $F_p = \{0, 1, 2\}$. Following the dimension types, there are two dimensions, $\text{Gran}_\text{Diagnosis}$ and $\text{Gran}_\text{HbA1c\%}$. The $\text{Gran}_\text{Diagnosis}$ dimension has the categories $\text{Granularity}_\text{Diagnosis}$ and $\top_\text{GranDiagnosis}$. The values of the $\text{Granularity}_\text{Diagnosis}$ category are those in the set of category types { Low-level Diagnosis, Diagnosis Family, Diagnosis Group, $\top_\text{Diagnosis}$}. The $\text{Gran}_\text{HbA1c\%}$ dimension has the categories $\text{Granularity}_\text{HbA1c\%}$ and $\top_\text{GranHbA1c\%}$. The values of the $\text{Granularity}_\text{HbA1c\%}$ category are those in the set { Precise, Imprecise, $\top_\text{HbA1c\%}$}. The partial orders on the two dimensions are the simple ones, where the values in the bottom category are unrelated and the $\top$ value is greater than all of them. The fact-dimensions relations R_1 and R_2 have the contents $R_1 = \{(0, \text{Diagnosis Group}), (1, \text{Diagnosis Family}), (2, \text{Diagnosis Family})\}$ and $R_2 = \{(0, \top_\text{HbA1c\%}), (1, \text{Precise}), (2, \text{Imprecise})\}$. A graphical illustration of the precision MO is seen in Figure 6.

The test to see if the data is precise enough to answer a query Q is performed by rewriting the query $Q = \alpha[C_1, .., C_n, D_{n+1}, g](M)$ to a "testing" query $Q_p = \alpha[G_1, .., G_n, G_{n+1}, \mathit{SetCount}](M_p)$, where G_i is the corresponding "granularity" component in D_{p_i} if $C_i \neq \top_i$. Otherwise, $G_i = \top_i$. Thus, we group *only* on the granularity components corresponding to the components that the

[†]To avoid unnecessary complexity in the examples only one diagnosis is considered for Jane Doe in this section, namely the diagnosis "9" (Insulin dependent diabetes, current version).

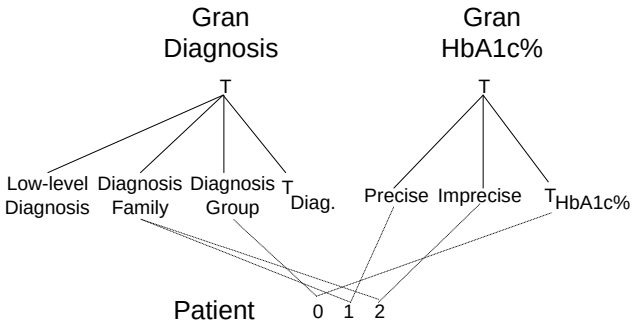

Fig. 6: Precision MO

physician has chosen to group on. The dimension G_{n+1} is used to hold the result of counting the members in each "granularity group." The result of the testing query shows how many facts map to each *combination* of granularities in the dimensions that the physician has chosen to group on. This result can be used to suggest alternative queries, as it is now easy for each dimension D_i to determine the minimal category C_i' that has the property that $Type(C_i) \sqsubseteq_{\mathcal{T}_i} Type(C_i') \wedge \forall C_{ij}(f \in F \wedge (f,e) \in R_i \wedge e \in C_{ij} \Rightarrow Type(C_{ij}) \sqsubset_{\mathcal{T}_i} Type(C_i'))$, i.e., in each dimension we choose the minimal category greater than or equal to the original "grouping" category where the data is "precise enough" to determine how to group the facts. We can also *directly present* the result of the testing query to the physician, to inform about the level of data imprecision for that particular query. A physician can then use this additional information to decide whether to run the alternative query or proceed with the original one.

Example 20 A physician wants to know the average HbA1c% grouped by Diagnosis Family. The query, Q, is $\alpha[\text{Diagnosis Family}, \top_{\text{HbA1c\%}}, D_3, AVG_2](M)$. Q groups only on Diagnosis Family since the $\top_{\text{HbA1c\%}}$ component has only one value. The testing query then becomes

$$Q_p = \alpha[\text{Granularity}_{\text{Diagnosis}}, \top_{\text{GranHbA1c\%}}, \text{D}_3, SetCount](M_p),$$

which counts the number of facts with the different Diagnosis granularity levels. The results of Q_p, described by the fact-dimension relations are the three sets listed below.

- $R_1 = \{(\{1,2\}, \text{Diagnosis Family}), (\{0\}, \text{Diagnosis Group})\}$
- $R_2 = \{(\{1,2\}, \top_{\text{GranHbA1c\%}}), (\{0\}, \top_{\text{GranHbA1c\%}})\}$
- $R_3 = \{(\{1,2\}, 2), (\{0\}, 1)\}$

This informs us that two patients are diagnosed with a diagnosis family, while one is diagnosed with a diagnosis group diagnosis. Thus, an alternative query is $Q = \alpha[\text{Diagnosis Group}, \top_{\text{HbA1c\%}}, D_3, AVG_2](M)$, which groups on Diagnosis Group rather than Diagnosis Family.

This approach for finding alternative queries has no problems as long as we only have *single* hierarchies in the dimensions, which is most often the case. When multiple hierarchies occur in a dimension, e.g., a time hierarchy with days that roll up to weeks *or* months, as described in Example 9, more care is required. For example, if the user asks for a grouping by Day and some facts are mapped directly to weeks and other facts directly to months, it is not straightforward to find the category on which the data can be grouped precisely, as weeks and months are not directly related. One approach is to take the *least upper bound* of the categories of the data, which in this case is the $\top$ category, meaning that we cannot distinguish the facts on the Time dimension. However, in other cases, this approach can be quite useful, e.g., if weeks rolled up to quarters. If we could provide an (imprecise) mapping from all but one of the unrelated categories to the remaining category, in this case from weeks to months, we could then choose the remaining category as the grouping category. This results in an imprecise query answer, which should be communicated to the user. The evaluation of imprecise queries is described next.

6. IMPRECISION IN QUERY EVALUATION

If the physician wants the original query answered, even though the data is not precise enough, we need to handle imprecision in the query evaluation. This section shows how imprecision is addressed in the grouping of data, in the computation of aggregate functions, and when presenting the imprecise result to the physician.

6.1. Imprecision in Grouping

We first need the ability to handle imprecision in the data used to group the facts. If a fact maps to a category that is finer than or equal to the grouping category in that dimension, there are no problems. However, if a fact maps to a coarser category, we do not know with which of the underlying values in the grouping category it should be grouped. To remedy the situation, we give the physician *several answers* to the query. First, a *conservative* answer is given that includes in a group only data *known* to belong to that group, but discards the data that is too imprecise to determine group membership. Second, a *liberal* answer is given that includes in a group all data that *might* belong to that group. Third, a *weighted* answer is given that also includes in a group all data that might belong to it, but where the inclusion of data in the group is *weighted* according to the likelihood of membership. Any subset of these three answers can also be presented if the physician so prefers. These three answers give a good overview of how the imprecision in the data affects the query result and thus provide a good foundation for making decisions taking the imprecision into account.

The conservative grouping requires no changes to the algebra. By default, the *standard* aggregate formation operator groups only the facts that are characterized by dimension values having a granularity finer than or equal to the granularity of the grouping components in the respective dimensions. The rest of the facts are discarded, leaving just the conservative result.

The liberal grouping additionally captures the data that is mapped directly to categories coarser than the grouping categories. To allow for a precise definition of the liberal grouping, we change the semantics of the aggregate formation operator. In Section 10, we discuss how to get the same result using only the standard aggregate formation operator, thus maintaining the ability to implement the approach without the need for new operators. We change the semantics of the aggregate formation operator so that the facts are grouped according to dimension values of the *finest granularity coarser than or equal to the grouping categories* available. Thus, *either* a fact is mapped to dimension values in categories at least as fine as the grouping categories, i.e., the data is "precise enough," *or* the fact is mapped *directly* to dimension values of a coarser granularity than the grouping categories. The formal semantics of the modified aggregate formation operator is given by replacing the original definitions with the ones given below:

$$F' = \{Group(e_1, .., e_n) \mid (e_1, .., e_n) \in D_1 \times .. \times D_n \wedge Type(C_1) \sqsubseteq_{\mathcal{T}_1} G_1(e_1) \wedge .. \wedge Type(C_n) \sqsubseteq_{\mathcal{T}_n} G_n(e_n) \wedge Group(e_1, .., e_n) \neq \emptyset \wedge (\forall i(\neg\exists e_i'(e_i' <_i e_i \wedge Group(e_1, .., e_i', .., e_n) \subseteq Group(e_1, .., e_i, .., e_n)))\}$$ and

$$R_i' = \{(f', e_i') \mid \exists (e_1, .., e_n) \in D_1 \times .. \times D_n \ (f' = Group(e_1, .., e_n) \wedge f' \in F' \wedge e_i = e_i')\}.$$

In these definitions, the dimension values are allowed to range over the categories that have coarser or the same granularity as the grouping categories. Groups are formed according to the *most precise* values, of a granularity at least as coarse as the grouping categories, that characterize a fact.

Example 21 If we want to know the number of patients, grouped by Diagnosis Family, and project out the other three dimensions, we will get the set of facts $F' = \{\{0\}, \{1\}, \{2\}\}$, meaning that each patient goes into a separate group, one for each of the two diagnosis families and one for the Diabetes diagnosis group. The fact-dimension relations are $R_1 = \{(\{0\}, 11), (\{1\}, 10), (\{2\}, 9)\}$ and $R_2 = \{(\{0\}, 1), (\{1\}, 1), (\{2\}, 1)\}$. We see that each group of patients (with one member) is mapped to the most precise member of the Diagnosis dimension with a granularity coarser than or equal to Diagnosis Family, that characterize the group. The count for each group is 1.

We can use the result of the modified aggregate formation operator to compute the liberal grouping. For each group characterized by values in the grouping categories, i.e., the "precise enough" data, we add the facts belonging to groups characterized by values that "contain" the precise values, i.e., we add the facts that *might* be characterized by the precise values. Formally, we say that $Group^l(e_1, .., e_n) = \cup_{e'_1 \geq_1 e_1, .., e'_n \geq_n e_n} Group(e'_1, .., e'_n)$, where the $Group(e'_1, .., e'_n)$'s are the groups in the result of the modified aggregate formation operator. Thus, the liberal (and conservative) grouping is easily computed from the result of the modified aggregate formation operator.

Example 22 If we want the number of patients, grouped *liberally* by Diagnosis Family, we will get the set of facts $F' = \{\{0,1\},\{0,2\}\}$, meaning that patient 0 goes into both of the two diagnosis family groups. The fact-dimension relations are $R_1 = \{(\{0,1\},10),(\{0,2\},9)\}$ and $R_2 = \{(\{0,1\},2),(\{0,2\},2)\}$. Here, each patient is mapped to all the diagnosis families that might be true for the patient. The count for each group is 2, meaning that for each of the two diagnosis families, there might be two patients diagnosed with that diagnosis family. Of course this cannot be true for both diagnosis families simultaneously.

The liberal approach over-represents the imprecise values in the result. If the same fact ends up in, say, 20 different groups, it is undesirable to give it the same weight in the result for a group as the facts that certainly belong to that group, because this would mean that the imprecise fact is reflected 20 times in the overall result, while the precise facts are reflected only once. A result is desirable where the imprecise facts are reflected at most once in the overall result.

We therefore introduce a *weight* w for each fact f in a group, making the group a *fuzzy set* [58]. We use the notation $f \in_w Group(e_1, .., e_n)$ to mean that f belongs to $Group(e_1, .., e_n)$ with weight w. The weight assigned to the membership of the group comes from the partial order $\sqsubseteq$ on dimension values. For each pair of values e_1, e_2 such that $e_1 \sqsubseteq e_2$, we assign a weight p, using the notation $e_1 \sqsubseteq (p)\ e_2$, meaning that e_2 should be counted with weight p when grouped with e_1. Normally, the weights would be assigned so that for a category C and a dimension value e, we have that $\Sigma_{e_1 \in C \wedge e_1 \sqsubseteq (p) e}\ p = 1$, i.e., the weights for one dimension value w.r.t. any given category sums to one. This would mean that imprecise facts are counted only once in the result set. However, we do not assume this, to allow for a more flexible attribution of weights.

Formally, we define a new *Group* function that also computes the weighting of facts. The definition of this function is $Group^w(e_1, .., e_n) = \cup_{e'_1 \geq_1 (p_1) e_1, .., e'_n \geq_n (p_n) e_n} Group(e'_1, .., e'_n)$, where the $Group(e'_1, .., e'_n)$'s are the groups from the result of the modified aggregate formation operator. The weight assigned to facts is given by the group membership as: $f \in Group(e'_1, .., e'_n) \Rightarrow f \in_{Comb(p_1,..,p_n)} Group^w(e_1, .., e_n)$, where the e_i's, the e'_i's, and the p_i's come from the $Group^w$ definition above. The function *Comb* combines the weights from the different dimensions to one, overall weight. The most common combination function will be $Comb(p_1, .., p_n) = p_1 \cdot ... \cdot p_n$, but for flexibility, we allow the use of more general combination functions, e.g., functions that favor certain dimensions over others. Note that all members of a group in the result of the modified aggregate formation operator get the same weight, as they are characterized by the same combination of dimension values.

The idea is to apply the weight of facts in the computation of the aggregate result, so that facts with low weights only contribute little to the overall result. This is treated in detail in the next section, but we give a small example here to illustrate the concept of weighted groups.

Example 23 We know that 80% of Diabetes patients have insulin dependent diabetes, while 20% have non insulin dependent diabetes. Thus, we have that $9 \sqsubseteq (.8)\ 11$ and $10 \sqsubseteq (.2)\ 11$, i.e., the weight on the link between Diabetes and Insulin dependent diabetes is .8 and the weight on the link between Diabetes and Non insulin dependent diabetes is .2. The weights on all other links are 1. Again, we want to know the number of patients, grouped by Diagnosis Family. The $Group^w$ function divides the facts into two sets with weighted facts, yielding the set of facts $F' = \{\{0_{.8}, 2_1\}, \{0_{.2}, 1_1\}\}$. Using subscripts to indicate membership weighting, the result of the computation is given in the fact-dimension relations $R'_1 = \{(\{0_{.8}, 2_1\},$ Insulin dependent diabetes$), (\{0_{.2}, 1_1\},$ Non insulin dependent diabetes $)\}$ and $R'_2 = \{(\{0_{.8}, 2_1\}, 1.8), (\{0_{.2}, 1_1\}, 1.2)\}$,

meaning that the weighted count for the group containing the insulin dependent diabetes patients 0 and 2 is 1.8 and the count for the non insulin dependent diabetes patients 0 and 1 is 1.2.

6.2. Imprecision in Computations

Having handled imprecision when grouping facts during aggregate formation, we proceed to handle imprecision in the computation of the aggregate result itself. The overall idea is to compute the resulting aggregate value by "imputing" precise values for imprecise values, and to carry along a computation of the imprecision of the result "on the side."

For most MO's, it only makes sense to the physician to perform computations on *some* of the dimensions, e.g., it makes sense to perform computations on the HbA1c% dimension, but not on the Diagnosis dimension. For a dimension D, where computation makes sense, we assume a function $E : D \mapsto \perp_D$ that gives the *expected value*, of the finest granularity in the dimension, for any dimension value. The expected value is found from the probability distribution of precise values around an imprecise value. We assume that this distribution is known. For example, the distribution of precise HbA1c% values around the $\top$ value follows a normal distribution with a certain mean and variance.

The aggregation function g then works by "looking up" the dimension values for a fact f in the argument dimensions, applying the expected value function, E, to the dimension values, and computing the aggregate result using the expected values, i.e., the results of applying E to the dimension values. Thus, the aggregation functions need only work on data of the finest granularity. The process of substituting precise values for imprecise values is generally known as *imputation* [46]. Normally, imputation is only used to substitute values for *unknown* data, but the concept is easily generalized to substitute a value of the finest granularity for any value of a coarser granularity. We term this process *generalized imputation*. In this way, we can use data of any granularity in our aggregation computations.

Generalized imputation does not indicate the precision of a result, so we need to carry along in the computation a measure of the precision of the result. A *granularity computation measure* (GCM) for a dimension D is a type CM that represents the granularity of dimension values in D *during* aggregate computation. A *measure combination function* (MCF) for a granularity computation measure CM is a function $h : \text{CM} \times \text{CM} \mapsto \text{CM}$, that combines two granularity computation measure values into one. We require that an MCF be *distributive* and *symmetric*. This allows us to directly combine intermediate values of granularity computation measures into the overall value. A *final granularity measure* (FGM) is a type FM, that represents the "real" granularity of a dimension value. A *final granularity function* (FGF) for a final granularity measure FM and a granularity computation measure CM is a function $k : \text{CM} \mapsto \text{FM}$, that maps a computation measure value to a final measure value. The reason to distinguish between computation measures and final measures is only that this allows us to require that the MCF be distributive and symmetric. The choice of granularity measures and functions is made depending on how much is known about the data, e.g., the probability distribution, and what final granularity measure the physician desires.

Example 24 The *level* of a dimension value, with 0 for the finest granularity, 1 for the next, and so on, up to n for the $\top$ value, provides one way of measuring the granularity of data. A simple, but meaningful, FGM is the *average level* of the dimension values that were counted for a particular aggregate result value. As the intermediate average values cannot be combined into the final average, we need to carry the sum of levels and the count of facts during the computation. Thus the GCM is $\text{CM} = \mathcal{N} \times \mathcal{N}$, the pairs of natural numbers, and the GCM value for a dimension value e is $(\mathit{Level}(e), 1)$. The MCF is $h((n_1, n_2), (n_3, n_4)) = (n_1 + n_3, n_2 + n_4)$. The FGM is $\mathcal{R}$, the real numbers, and the FGF is $k(n_1, n_2) = n_2/n_1$. In the case study, precise values such as 5.5 have level 0, imprecise values such as 5 have level 1, and the $\top$ value has level 2.

Example 25 The *standard deviation* $\sigma(X)$ of a set of values X from the average value $e(X)$ is a widely used estimate of how much data varies around e. Thus, it can also be used as an estimate of the *precision* of a value. Given the probability distribution of precise values p around an imprecise

value i, we can compute the standard deviation of the p's from $E(i)$ and use it as a measure of the granularity of i. However, we cannot use σ as a GCM directly because intermediate σ's cannot be combined into the overall σ. Instead we use as GCM the type CM $= \mathcal{N} \times \mathcal{R} \times \mathcal{R}$, computing using the *count* of values, the *sum* of values, and the *sum of squares* of values as the GCM values. For a value x, the GCM value is $(1, x, x^2)$. The MCF is $h((n_1, x_1, y_1), (n_2, x_2, y_2)) = (n_1 + n_2, x_1 + x_2, y_1 + y_2)$. This MCF is distributive and symmetric [49]. The FGM is FM $= \mathcal{R}$, which holds the standard deviation, and the FGF is $k(n, x, y) = \sqrt{(y - x^2)/(n-1)}$. For values of the finest granularity, only data for one X is stored. For values of coarser granularities, we store data for several X values, chosen according to the probability distribution of precise values over the imprecise value. In the case study, we would store data for 1 X value for precise values such as 5.5, for 10 X values for imprecise values such as 5, and for 100 X values for the $\top$ value. This ensures that we obtain a precise estimate of the natural variation in the data as the imprecision measure, just as we would had *multiple imputation* [6, 46] been used.

For the *conservative* and *liberal* answers, we use the above technique to compute the aggregate result and its precision. All facts in a group contribute equally to both the result and the precision of the result. For the *weighted* answer, the facts in a group are counted according to their weights, both in the computation of the aggregate result and in the computation of the precision. We note that for an aggregation function, whose result depends only on one value in the group it is applied to, such as MIN and MAX, we obtain the minimum/maximum of the *expected values*.

Example 26 We want to know the average HbA1c% for patients, grouped by Diagnosis Family, and the associated precision of the results. As granularity measures and functions, we use the *level* approach described in Example 24. We discuss only the weighted result. As seen in Example 23, the resulting set of facts is $F' = \{\{0_{.8}, 2_1\}, \{0_{.2}, 1_1\}\}$, and the *SetCount* is 1.8 for the first group and 1.2 for the second. When computing the *sum* of the HbA1c% values, we impute 7.0 and 6.0 for the imprecise values 7 and $\top$, respectively. For the first group, we multiply the values 6.0 and 5.5 by their group weights .8 and 1, respectively, before adding them together. For the second group, 5.5 and 6.0 are multiplied by 1 and .2, respectively. Thus, the result of the sum for the two groups is 10.3 and 6.7, giving an average result of 5.7 and 5.6, respectively.

The computation of the precision proceeds as follows. The level of the values $\top$, 5.5, and 7 is 2, 0, and 1, respectively. The *weighted sum* of the levels for each group is found by multiplying the level of a value by the group weight of the corresponding fact, yielding 1.6 for the first group and 1.4 for the second. The *weighted count* of the levels is the same as that for the facts themselves, namely 1.8 and 1.2. This gives a *weighted average level* of .9 for the Insulin dependent diabetes group and 1.2 for the Non insulin dependent diabetes group, meaning that the result for the first group is more precise. The relatively high imprecision for the first group is mostly due to the high weight (.8) that is assigned to the link between Diabetes and Insulin dependent diabetes. If the weights instead of .8 and .2 had been .5 and .5, the weighted average levels would have been .7 and 1.3.

6.3. Presenting the Imprecise Results

The final step in the imprecision handling is to *present* the imprecision in the result to the physician. We consider two alternatives for this step. The most straightforward approach is to present the result values along with their corresponding *final granularity measure* values. This gives a good estimate of the precision of a result value.

Example 27 For the result in Example 26, this approach would present the (Diagnosis Family, *AVG*(HbA1c%), *AVG*(Level)) tuples from the *conservative*, the *liberal*, and the *weighted* answers. For the conservative answer, the result is {(Insulin dependent diabetes, 5.5, 0), (Non insulin dependent diabetes, 7, 1)}. For the liberal answer, the result is {(Insulin dependent diabetes, 5.8, 1), (Non insulin dependent diabetes, 6.5, 1.5)}. Finally, for the weighted answer, the result is {(Insulin dependent diabetes, 5.7, .9), (Non insulin dependent diabetes, 5.6, 1.2)}.

The other alternative for presenting the imprecision is one which follows our overall approach of using the granularity itself as an estimate of the precision of data. We use the imprecision of a result value to convert the value into a value of a (coarser) granularity corresponding to the imprecision. A *value coarsing function* (VCF) for a dimension D and an FGM M is a function $c : \perp_D \times M \mapsto D$, where $c(e) = e_1$ such that $e \sqsubseteq e_1$. Thus, the VCF maps values of the finest granularity into "containing" values of a possibly coarser granularity, determined by the imprecision. The VCF and the granularities of the result dimension are chosen so that the granularity of the result gives a good overview of the true precision.

Example 28 We choose the HbA1c% dimension, with the same granularities, as the result dimension. As the VCF we choose $r(x) = v$ such that $x \sqsubseteq v \wedge Level(v) = Ceiling(x)$, i.e., for a number x, we choose the value that "contains" x and has the level of the least natural number greater than or equal to x, e.g., $r(.9) = 1$ and $r(1.2) = \top$. A graphical illustration of the resulting MO's for the conservative, liberal, and weighted results are seen in Figure 7. We note that the liberal and weighted answers are identical, suggesting that this is closer to the truth than the conservative answer in this case. The result value for AVG(HbA1c%) is $\top$ in both the liberal and the weighted answer for the Non insulin dependent group because half of the input data is unknown, rendering the resulting average value very imprecise.

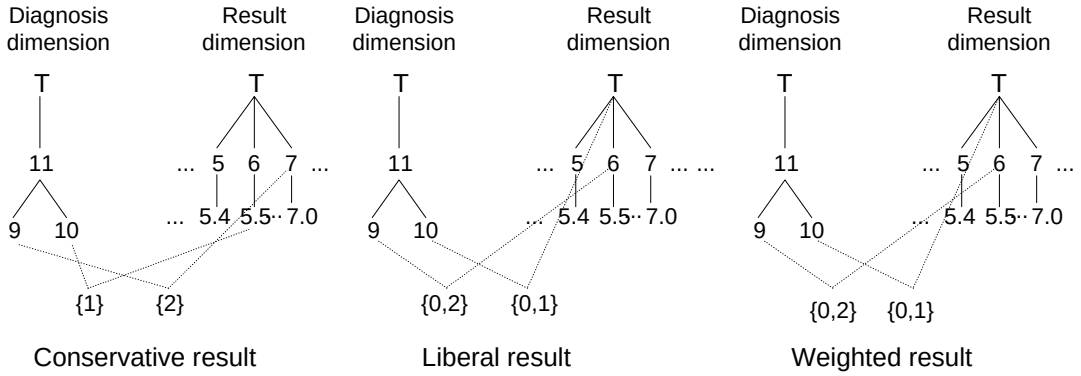

Fig. 7: Resulting MO's for the Conservative, Liberal, and Weighted Answers

7. ADDRESSING THE REQUIREMENTS

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

Our 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 using *aggregation semantics*. 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.

A 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. *Non-onto* hierarchies are directly supported, e.g., the "Lung Cancer" diagnosis has no low-level diagnosis child. Similarly, the model supports *non-covering hierarchies*, e.g., as illustrated by the rural addresses in the Residence dimension. 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, data can be viewed as it appears 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, thereby obtaining meaningful analyses across changes in the data. In this respect, the model *handles change and time*. 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 have been diagnosed with *low-level diagnoses*, some with Diagnosis Families, and some with Diagnosis Groups. We have shown how the model can use the concept of granularities to handle *imprecise data*, e.g., as exemplified by the HbA1C% values.

8. RELATIONAL REPRESENTATION OF THE MODEL

This section outlines how to implement the model using relational database technology. The objective is to obtain a relational representation that permits the 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 query tool accessing the data. The representation of this metadata in a relational database is not considered here.

The traditional mapping of a multidimensional data model to a relational database uses a *star schema* [26], 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, as well as requires the hierarchies in the dimensions to 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 $T_{\mathrm{C_j-Rep}}$ that encodes the composition of *Rep* with the partial order on the dimension. If *Rep* is encoded in the table $T_{\mathrm{Rep}} = (\mathit{RepValue}, \mathit{DimensionValue})$ and the direct parent-child relationships in the partial order are encoded in the table $PO = (\mathit{ParentValue}, \mathit{ChildValue})$ then $T_{\mathrm{C_j-Rep}} = T_{\mathrm{Rep}} \bowtie_{\mathit{DimensionValue}=\mathit{ParentValue}} PO^*$, where PO^* denotes the reflexive, transitive closure of PO. Note that $T_{\mathrm{C_j-Rep}}$ 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. $T_{\mathrm{C_j-Rep}}$ 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 29 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 obtaining 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 $T_{\mathrm{DiagGroup-Code}}$ 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
A1	13	13	01/01/80	NOW
A1	13	14	01/01/80	NOW

Table 3: The $T_{\text{DiagGroup}-\text{Code}}$ 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 30 For the previous example, alternative a) would maintain the join of $T_{\text{DiagGroup}-\text{Code}}$ with the Has table. Alternative b) would give three combinations, $\{10\}$, $\{11\}$, and $\{3,5,8,9\}$, which would be the bottom values in the extension of the $T_{\text{DiagGroup}-\text{Code}}$ table. The fact table would then point to these combinations, instead of the diagnoses directly. Alternative c) would just keep the $T_{\text{DiagGroup}-\text{Code}}$ 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 table sizes to a minimum, but accidental double-counting is possible, so 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 $T_{\text{DiagGroup}-\text{Code}}$ table extended with time is seen in Table 3. Alternatives 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.

9. SQL IMPLEMENTATION OF IMPRECISION

We proceed to describe how to implement the imprecision handling approach using commercial relational database technology. The goal is to provide a mapping to relational tables and a set of query templates that allow the physician to get the same results as in the presented approach, with reasonable efficiency.

In most relational representations of multidimensional data, the tables are divided into *fact tables* and *dimension tables* [26]. As the names suggest, a fact table contain data related to a particular fact, while the dimension tables contain information about the dimension values and the hierarchies between them. In the presented data model, *all* data is considered to be *dimensional*, including data that would normally be treated as "measures" in other multidimensional models, e.g., the HbA1c% measurements. We follow this approach in the relational design, leading to a "factless" fact table [26], i.e., a fact table where all the columns are *dimension keys* (DK), i.e., foreign keys to dimension tables. However, as the combination of dimension values for a fact f is *not* a "key" for f in our model, we also need to include a column to represent the *fact identity* in the fact table. Thus, the fact table has the schema $(FactId, DK_1, ..., DK_n)$. All data about the dimension values will be kept in dimension tables. We can still have reasonably fast access to the data using techniques such as *star join* query processing [45], a technique optimized for "multidimensional" relational queries. If the performance obtained with this design is not sufficient, we can denormalize the fact table by placing the expected values (EV) and the granularity computation measures (GCM) in it. This gives a schema of the form $(\text{FactId}, \text{DK}_1, \text{EV}_1, \text{GCM}_1, .., \text{DK}_\text{n}, \text{EV}_\text{n}, \text{GCM}_\text{n})$. We include the EV's and GCM's only for the dimensions on which computation is meaningful. Assuming that the size (in bytes) of EV's and GCM's is the same as the size of the dimension keys, and that computation makes sense for half of the dimensions, this will *double* the space required for the fact table.

The design of the dimension tables depends on the complexity of the data. If the hierarchies are strict, onto, and covering, and we only map facts to dimension values of the finest granularity, we can capture the dimensions using ordinary "flat" dimension tables, leading to "star schema" type design [26]. We record the dimension values (DV) for the different granularities as different columns. We need to store the weights (W) for each of relations between a dimension value of the finest granularity and the values of coarser granularities. Because of the restrictions, we need only to record the expected values and granularity computation measures for the finest granularity. The schema of the dimension tables will have the structure $(\text{DK}, \text{EV}_\bot, \text{GCM}_\bot, \text{DV}_\bot, \text{W}_\bot, .., \text{DV}_\top, \text{W}_\top)$.

However, we would like to capture explicitly in the relational schema the situation that facts are mapped directly to dimension values of coarser granularities. This can be captured by storing a table of pairs of all *ancestors* (A) and *descendents* (D) in the dimension partial order, i.e., the *transitive closure* of the direct parent-child relationships. The computation and maintenance of materialized transitive closures has been studied intensively in the scientific literature [1, 20], so we do not discuss it further. For each (A,D) pair of dimension values, we record the *levels* (L) of the ancestor and descendent, i.e., $0, 1, .., n$, as well as the weight (W) on the link between A and D. Additionally, we record the EV's and GCM's for the *descendents only*, where it makes sense. The schema of the dimension tables will have the structure $(\text{A}, \text{L}_\text{A}, \text{D}, \text{L}_\text{D}, \text{W}, \text{EV}_\text{D}, \text{GCM}_\text{D})$. We note that we can still take advantage of star join processing with this schema.

The aggregate formation queries must be translated into standard SQL queries. The most general type of query is the one that computes the *liberal* grouping, while taking the *weighting* into account. We will deal with this; the SQL queries needed for the other parts of query evaluation are just special cases. The general SQL query has the form seen below.

```
SELECT g(Comb(D1.W, .., Dm.W) * Dk.EV), GCF(Comb(D1.W, .., Dm.W) * Dk.GCM)
FROM F, D1, .., Dm
WHERE
      F.DK1 = D1.DK AND ... AND F.DKm = Dm.DK AND
      F.DKk = Dk.DK AND Dk.LA = Dk.LD AND
      (D1.LA = GL1 OR (D1.LA > GL1 AND D1.LA = D1.LD)) AND
      ....
      (Dm.LA = GLm OR (Dm.LA > GLm AND Dm.LA = Dm.LD))
GROUP BY D1.A, .., Dm.A
```

In the query, g is the aggregation function, *Comb* is the weighting combination function, D_k is the dimension on which we compute, F is the fact table, GCF is the granularity combination function, $\text{D}_1, .., \text{D}_\text{m}$ are the m dimensions where we group on something else than the $\top$ category,

and $GL_1, .., GL_m$ is the corresponding grouping levels. We can use this type of query only if the weights can be multiplied directly into the results, e.g., when g is *SUM*. For other types of aggregation functions, e.g., *AVG*, we need to use several queries and combine the results. The first line of the WHERE clause specifies join predicates on the fact table and the dimension tables used for grouping. The second line specifies join predicates on the fact table and the dimension table holding the data to be computed on; this ensures that we only get one value for each fact. The remaining lines of the WHERE clause handle the grouping of facts. The part before the "OR" in each line handles the *conservative* grouping, while the remainder handles the additional data in the *liberal* grouping.

Example 31 We implement the MO from the case study† with only the Diagnosis and HbA1c% dimensions, using the basic fact table design and (A,D) type dimension tables. We include the text of the ancestors for readability. The resulting tables are seen in Table 4. When using SQL to compute the weighted average of the HbA1c%, grouped by Diagnosis Family, as seen in Example 26, we obtain two SQL queries: one for computing the weighted sum and one for computing the weighted count. The results of these two queries can then be combined into the total weighted result, as described in Example 26. The SQL statements are seen below.

```
SELECT D.Ancestor, SUM(H.EV * D.W), SUM(H.GCM * D.W)
FROM Fact F, Diagnosis D, HbA1C H
WHERE
      F.DiagKey = D.DesID AND
      F.HbA1Key = H.DesID AND H.AnsLevel = H.DesLevel AND
      (D.AnsLevel = 0 OR (D.AnsLevel > 0 AND D.AnsLevel = D.DesLevel))
GROUP BY D.Ancestor

SELECT D.Ancestor, SUM(D.W)
FROM Fact F, Diagnosis D, HbA1C H
WHERE
      F.DiagKey = D.DesID AND
      F.HbA1Key = H.DesID AND H.AnsLevel = H.DesLevel AND
      (D.AnsLevel = 0 OR (D.AnsLevel > 0 AND D.AnsLevel = D.DesLevel))
GROUP BY D.Ancestor
```

If pre-aggregation is used, we also need tables for storing the pre-aggregated values. These should have the format of the denormalized fact table described above. If (A,D) type dimension tables are used in the design, we can re-use these to access the aggregate tables. If "flat" dimension tables are used, we need to construct new dimension tables with only the relevant (higher category) columns [26] to access the aggregate tables.

10. USING PRE-AGGREGATED DATA

The approach presented is this paper handles imprecision by storing a few extra attributes for the dimension values and computing the imprecision based on these attributes during normal query evaluation. No new algorithms, loops, etc., are introduced. Thus, the computational complexity of query evaluation is only changed by a constant factor and is unchanged in big-O terms. The computational complexity of query evaluation is dominated by the grouping of data. Using normal sorting, this can be accomplished in $O(n \log n)$ time, where n is the number of facts. Even though this is a low complexity compared to previously suggested approaches [8, 47, 48], it is attractive to lower the running time of queries even further. A decisive factor in the success of commercial OLAP products is the successful use of *pre-aggregated data* for speeding up query execution. Ideally, the handling of imprecision in OLAP systems should also take advantage of pre-aggregated data, so

†To avoid unnecessary complexity, we consider again only diagnosis "9" for Jane Doe, and we consider only the Diagnosis Family and Diagnosis Group categories in the Diagnosis dimension.

Fact	DiagKey	HbA1Key
0	11	6
1	10	7
2	9	8

Fact Table

AnsID	DesID	Ancestor	AnsLevel	DesLevel	W
9	9	Insulin dependent diabetes	0	0	1
10	10	Non insulin dependent diabetes	0	0	1
11	11	Diabetes	1	1	1
11	9	Diabetes	1	0	.8
11	10	Diabetes	1	0	.2

Diagnosis Dimension Table

AnsID	DesId	Ancestor	AnsLevel	DesLevel	W	EV	GCM
6	6	Unknown	2	2	1	6.0	2
7	7	5.5	0	0	1	5.5	0
8	8	7	1	1	1	7.0	1
6	7	Unknown	2	0	.01	5.5	0
6	8	Unknown	2	1	.1	7.0	1

HbA1c% Dimension Table

Table 4: Relational Implementation of the Case Study

that query evaluation remains fast when handling imprecision. This section investigates how our approach can exploit pre-aggregated data.

The most common strategies for pre-aggregation is *full*, *no*, and *partial* pre-aggregation. With full pre-aggregation, aggregates are stored for *all combinations* of granularities in the different dimensions. This provides fast response time, but requires very large amounts of storage space, and the cost of keeping the aggregates up to date is very high. In some real-world cases, full pre-aggregation requires up to 200 times as much space as the raw data, making it a very expensive option. However, if the multidimensional space for an MO is small and *dense*, i.e., facts exist for most combinations of dimension values, full pre-aggregation is attractive [36]. If full pre-aggregation is too expensive, partial pre-aggregation is an option.

With partial pre-aggregation, a number of combinations of dimension granularities is chosen, and the aggregate values are stored for these. The aggregate values are then *re-used* for coarser granularities, e.g., the aggregate results for Low-level Diagnosis could be re-used to compute the results for Diagnosis Family. The condition for re-use is that we have *summarizability* for the MO [30], which intuitively means that lower-level results can be directly combined into higher-level results.

It has been proven [30] that summarizability is equivalent to the hierarchies in dimensions being *strict*, *onto*, and *covering*, i.e., one lower-level dimension value maps to exactly one higher-level value, and for every higher-level value, there exists at least one lower-level value that maps to it. Additionally, facts must be mapped only to dimension values of the finest granularity, and the aggregation function must be distributive. This insight is important when investigating the use of pre-aggregated data. In previous work [41], it has been demonstrated how to normalize the kinds of non-strict, non-onto, and non-covering dimensions and fact-dimension relations that are possible with this paper's multidimensional model, so that summarizability is ensured. This normalization, which occurs transparently to the user, establishes partial pre-aggregation as a practical option for our more general multidimensional model.

The first step in the query evaluation is the *test* for sufficient data precision, and the possible

suggestion of an *alternative query*. This step was achieved by rewriting the original query to a "testing" query on the precision MO, as described in Section 5.2. With ten dimensions and four levels in each dimension, the size of the multidimensional space for the precision MO will be $4^{10} = 2^{20} \approx 1,000,000$, which is very small and probably quite dense. The result of the *SetCount* operation is stored for each combination of dimension values. Assuming four-byte integers for storing the counts, this takes only about 4MB. With such a small, dense multidimensional space, full pre-aggregation is feasible, yielding very fast response time for this part of the query evaluation.

The next steps in the query evaluation are the grouping of facts and the aggregate computation. With respect to pre-aggregation, it only makes sense to consider these two steps in conjunction. For the computation of the *aggregate result* itself, using the expected values, ordinary pre-aggregation techniques can be applied. If we want to use *partial* pre-aggregation, we need to make sure that we have summarizability. When checking the conditions for our case, we see that facts are mapped directly to values of coarser granularity, e.g., patient 0 is mapped directly to the Diabetes value. To ensure summarizability, we must introduce "placeholder" values [30] of the finest granularity, that "takes the place" of a coarser value. In our case, we could introduce a "Diabetes" placeholder value in the Low-level Diagnosis category and map patient 0 to it. The placeholder value is then mapped to the "real" Diabetes value, in this case through another "Diabetes" placeholder value in the Diagnosis Family category. When doing this, we get the side benefit that the *liberal* result is computed using the standard aggregate formation operator.

If we do not want to alter the MO in this way, we need to use *full* or *no* pre-aggregation, which may or may not be sensible in the given case. We note that full pre-aggregation can be applied even though we do *not* have summarizability. If the hierarchies are not altered to achieve summarizability, we can still compute the liberal result using the standard aggregate formation operator. This is done by issuing a series of queries, one for *each combination of granularities coarser than or equal to the grouping categories*. If grouping by Diagnosis Family (and $\top_{HbA1c\%}$), we would issue queries that grouped by Diagnosis Family and $\top_{HbA1c\%}$, by Diagnosis Group and $\top_{HbA1c\%}$, and by $\top_{\text{Diagnosis}}$ and $\top_{HbA1c\%}$. From the result of these queries, we can deduce the aggregate result for the part of the liberal answer *not* in the conservative answer, e.g., when knowing that the count of patients for $\top_{\text{Diagnosis}}$ is 3, the count for Diabetes is 3, the count for Insulin dependent diabetes is 1, and the count the Non insulin dependent diabetes is 1, we can deduce that the count for patients mapped directly to Diabetes is 1, and that no patients are mapped directly to $\top_{\text{Diagnosis}}$.

We also need to consider pre-aggregation in relation to the computation of the precision. The values that should be pre-aggregated are the aggregate values for the *granularity computation measures*. With respect to pre-aggregation, GCM values are just ordinary values, so the criteria and conditions discussed above for choosing full or partial pre-aggregation also apply. The measure combination function is required by definition to be distributive, so partial pre-aggregation can be applied if the rest of the summarizability conditions are met, meaning that intermediate precision values can be re-used to compute the total precision value. Thus, the computation of the precision of the result is *fully* supported by pre-aggregation.

For both the computation of the aggregate result and the computation of the imprecision, we note that the introduction of *weighting* does not disturb the pre-aggregation. We just store the weighted results and imprecisions instead of the un-weighted.

11. CONCLUSION AND FUTURE WORK

Motivated by the popularity of On-Line Analytical Processing (OLAP) systems for analyzing business data, multidimensional data models are being utilized in new applications areas. One such application area is medical case studies. The case study presented in this paper tracks patient information involving diagnoses, names, dates of birth, ages, places of residence, and HbA1C%, an indicator of the long-term blood sugar level. The case study illustrates that current models only have limited support for the complex and imprecise data found in many real-world systems. We identified eleven requirements for multidimensional data models to meet the needs of these new application areas. Current models are limited in their ability to satisfy some of the requirements,

such as many-to-many relationships between facts and dimensions, handling change and time, handling imprecise data, and handling different levels of granularity. Fourteen previously proposed data models from the research literature are evaluated on the eleven requirements. None of the fourteen models satisfies more than six requirements.

We propose a new, extended multidimensional data model, which addresses all eleven requirements. The data model improves over previously proposed models by supporting non-onto, non-covering, and non-strict hierarchies, many-to-many relationships between facts and dimensions, handling change and time, handling imprecise data, and handling different levels of granularity. In particular, time is supported by adding valid time and transaction time to the basic model. An important part of the proposed model is a fully developed algebra for multidimensional objects. Thus algebra is closed and at least as strong as relational algebra with aggregation functions.

The proposed data model captures imprecise data by using the concept of *data granularity*. Data imprecision is handled by first *testing* whether the data is precise enough to answer a query. If the data is sufficiently precise, the query is evaluated and a correct, precise answer results. New strategies are suggested for data that lacks sufficient precision. The first strategy is to suggest an *alternative query* that can be answered precisely. The second strategy handles imprecision explicitly by presenting three different answers to the user. The first type of answer discards the imprecise data and uses only the known, precise data, which leads to a *conservative* answer. The second type, the *liberal* answer, includes everything that could *possibly* be true, which allows some imprecise data to be included. The third type of answer is a *weighted* answer that includes everything that might be true, but assigns heavier weights to precise data than to imprecise data. Along with the aggregate computation, a separate computation of the *precision* of the result is carried out.

The proposed model can be implemented using current technology. We show that multidimensional objects can be represented as relational tables and that the imprecision handling approach can be implemented in SQL. This provides a foundation for implementing the model using relational database technology. Although the model supports time, imprecision, and advanced modeling capabilities, we also indicate how it is possible to exploit full and partial pre-aggregation for efficient query evaluation.

In future work, we plan to investigate how the model and query handling techniques can be efficiently implemented using special-purpose algorithms and data structures, to achieve optimal concrete complexity.

This paper has focussed on what we perceive as central modeling requirements of multidimensional models. In addition to supporting the features implied by these requirements, the integration of many other features is desirable and merits further study. It is important to consider scalability in dimensions: methods are needed that enable multidimensional models to cope with the hundreds of dimensions found in some applications. The inclusion of special support for time-series analysis is also relevant to a broad range of applications. More generally, multidimensional data models would benefit from a tight integration with the many concepts used widely in data mining, e.g., association and classification rules, classification trees, clustering, segmentation, and categorization.

There are several research directions in relation to imprecision. First, it should be investigated how "single-value" aggregation functions such as MIN and MAX, which are not readily sensitive to weighting, can be handled. Second, visualization of the imprecise results may greatly facilitate user interpretation. Suitable visualization techniques need to be developed, and would additionally aid in formulating queries. One idea is to display to the user the data that *prevented* a query from being precisely answered.

Finally, it would be desirable to obtain a notion of completeness for multidimensional algebras, similar to Codd's relational completeness.

12. ACKNOWLEDGMENTS

The authors would like to thank special issue co-editors Kenneth A. Ross and Patrick O'Neil and the expert anonymous reviewers. Their constructive and insightful comments improved the

paper substantially.

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

REFERENCES

[1] R. Agrawal and J. Kiernan. An Access Structure for Generalized Transitive Closure Queries. In *Proceedings of the Ninth International Conference on Data Engineering*, pp. 429–438, 1993.

[2] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. *IBM Technical Report 1995*. Appeared also in *Proceedings of the Thirteenth International Conference on Data Engineering*, pp. 232–243, 1997.

[3] 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.

[4] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, and X. S. Wang. A Glossary of Time Granularity Concepts. In [17], pp. 406–413.

[5] R. Bliujūtė, S. Šaltenis, 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.

[6] S. van Buuren, E. V. van Mulligen, and J. P. L. Brand. Routine Multiple Imputation in Statistical Databases. In *Proceedings of the Seventh International Conference on Scientific and Statistical Database Management*, pp. 74–78, 1994.

[7] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. In *Proceedings of the Sixth International Conference on Database Programming Languages*, pp. 319–335, 1997.

[8] A. L. P. Chen, J-S. Chiu, and F. S. C. Tseng. Evaluating Aggregate Operations over Imprecise Data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):273–284, 1996.

[9] P. P.-S. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transaction on Database Systems*, 1(1):9–36, 1976.

[10] 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.

[11] E. F. Codd. Extending the Data Base Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[12] E. F. Codd. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. Technical report, E.F. Codd and Associates, 1993.

[13] 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*, pp. 91–100, 1997.

[14] C. E. Dyreson. Information Retrieval from an Incomplete Data Cube. In *Proceedings of the Twentysecond Conference on Very Large Databases*, pp. 532–543, 1996.

[15] C. E. Dyreson. A Bibliography on Uncertainty Management in Information Systems. In [17], pp. 413–458.

[16] C. E. Dyreson and R. T. Snodgrass. Supporting Valid-time Indeterminacy. *ACM Transactions on Database Systems*, 23(1):1–57, 1998.

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

[18] E. Gelenbe and G. Hebrail. A Probability Model of Uncertainty in Databases. In *Proceedings of the Second International Conference on Data Engineering*, pp. 328–333, 1986.

[19] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.

[20] K-C. Guh and C. Yu. Efficient Management of Materialized Generalized Transitive Closure in Centralized and Parallel Environments. *IEEE Transaction on Knowledge and Data Engineering*, 4(4):371–380, 1992.

[21] M. Gyssens and L. V. S. Lakshmanan. A Foundation for Multi-Dimensional Databases. In *Proceedings of the Twentythird Conference on Very Large Databases*, pp. 106–115, 1997.

[22] K. J. Isselbacher, R. D. Adams, E. Braunwald, R. G. Petersdorf, and J. D.Wilson. *Principles of Internal Medicine*, Ninth Edition. McGraw-Hill, 1980.

[23] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What Can Hierarchies Do for Data Warehouses? In *Proceedings of the Twentyfifth International Conference on Very Large Data Bases*, pp. 530–541, 1999.

[24] 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.

[25] C. S. Jensen and C. E. Dyreson, (editors). A Consensus Glossary of Temporal Database Concepts—February 1998 Version. In [17], pp. 367–405.

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

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

[28] 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.

[29] W. Lehner. Modeling Large Scale OLAP Scenarios. In *Proceedings of the Sixth International Conference on Extending Database Technology*, pp. 153–167, 1998.

[30] H. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Databases. In *Proceedings of the Ninth International Conference on Scientific and Statistical Databases*, pp. 39–48, 1997.

[31] 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.

[32] J. Melton and A. R. Simon. *Understanding the New SQL—A Complete Guide.* Morgan Kaufmann, 1993.

[33] A. O. Mendelzon and A. A. Vaismann. Temporal Queries in OLAP. In *Proceedings of the Twentysixth International Conference on Very Large Data Bases*, pp. 242–253, 2000.

[34] Microsoft Corporation. OLE DB for OLAP Specification Version 1.0. *Microsoft Technical Document*, 1998.

[35] A. Motro and P. Smets (eds.). *Uncertainty Management in Information Systems—From Needs to Solutions.* Kluwer Academic Publishers, 1997.

[36] The OLAP Report. *Database Explosion.* The OLAP Report White Paper. URL: <www.olap-report.com/Databaseexplosion.html>. Current as of January 4th, 1999.

[37] T. B. Pedersen and C. S. Jensen. Clinical Data Warehousing—A Survey. In *Proceedings of the VIII Mediterranean Conference on Medical and Biological Engineering and Computing*, Section 20.3, 1998.

[38] T. B. Pedersen and C. S. Jensen. Research Issues in Clinical Data Warehousing. In *Proceedings of the Tenth International Conference on Scientific and Statistical Database Management*, pp. 43–52, 1998.

[39] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. In *Proceedings of the Fifteenth International Conference on Data Engineering*, pp. 336–345, 1999. Extended version available as TimeCenter Technical Report TR-37, URL: <www.cs.auc.dk/TimeCenter>, 1998.

[40] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Supporting Imprecision in Multidimensional Databases Using Granularities. In *Proceedings of the Eleventh International Conference on Scientific and Statistical Database Management*, pp. 90–101, 2000.

[41] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. In *Proceedings of the Twentyfifth International Conference on Very Large Databases*, pp. 663–674, 1999.

[42] M. Rafanelli and F. Ricci. Proposal of a Logical Model for Statistical Databases. In *Proceedings of the Second International Workshop on Statistical and Scientific Database Management*, 1983.

[43] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *Proceedings of the Fifth Conference on Statistical and Scientific Database Management*, pp. 14–29, 1990.

[44] National Health Service (NHS). *Read Codes version 3.* NHS, September 1999.

[45] Red Brick Corporation. Star Schema Processing for Complex Queries. *White Paper, Red Brick Inc.*, 1997.

[46] D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys.* Wiley, 1987.

[47] E. A. Rundensteiner and L. Bic. Aggregates in Possibilistic Databases. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, pp. 287–295, 1989.

[48] E. A. Rundensteiner and L. Bic. Evaluating Aggregates in Possibilistic Relational Databases. In *Data and Knowledge Engineering*, 7(3):239–267, 1992.

[49] S.-C. Shao. Multivariate and Multidimensional OLAP. In *Proceedings of the Sixth International Conference on Extending Database Technology*, pp. 120–134, 1998.

[50] A. Shoshani. OLAP and Statistical Databases: Similarities and Differences. In *Proceedings of Sixteenth ACM Symposium on Principles of Database Systems* , pp. 185–196, 1997.

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

[52] P. Vassiliadis. Modeling Multidimensional Databases, Cubes, and Cube Operations. In *Proceedings of the Tenth International Conference on Statistical and Scientific Database Management*, pp. 53–62, 1998.

[53] E. Wong. A Statistical Approach to Incomplete Information in Database Systems. *ACM Transactions on Database Systems*, 7(3):470–488, 1982.

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

[55] Yahoo! Corporation. Yahoo home page <www.yahoo.com>. Current as of September 29, 2000.

[56] J. Yang and J. Widom. Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing. In *Proceedings of the Sixth International Conference on Extending Database Technology*, pp. 389–403, 1998.

[57] J. Yang and J. Widom. Temporal View Self-Maintenance. In *Proceedings of the Seventh International Conference on Extending Database Technology*, pp. 395–412, 2000.

[58] L. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.

[59] T. Zurek and M. Sinnwell. Data Warehousing Has More Colours Than Just Black and White. In *Proceedings of the Twentyfifth International Conference on Very Large Data Bases*, pp. 726–729, 1999.

Recommended by Patrick O'Neil and Kenneth A. Ross, Co-Editors