Supporting Imprecision in Multidimensional Databases Using Granularities

Torben Bach Pedersen† Christian S. Jensen‡ Curtis E. Dyreson‡

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

‡ Department of Computer Science, Aalborg University,
Fredrik Bajers Vej 7E, DK–9220 Aalborg Ø, Denmark,
email: {`csj,curtis`}`@cs.auc.dk`

May 4, 1999

Abstract

On-Line Analytical Processing (OLAP) technologies are being used widely for business-data analysis, and these technologies are also being used increasingly in medical applications, e.g., for patient-data analysis. The lack of effective means of handling data imprecision, which occurs when exact values are not known precisely or are entirely missing, represents a major obstacle in applying OLAP technology to the medical domain, as well as many other domains. OLAP systems are mainly based on a multidimensional model of data and include constructs such as dimension hierarchies and granularities. This paper develops techniques for the handling of imprecision that aim to maximally reusing these already existing constructs. With imprecise data now available in the database, queries are tested to determine whether or not they may be answered precisely given the available data; if not, alternative queries that are unaffected by the imprecision are suggested. When a user elects to proceed with a query that is affected by imprecision, techniques are proposed that take into account the imprecision in the grouping of the data, in the subsequent aggregate computation, and in the presentation of the imprecise result to the user. The approach is capable of exploiting existing multidimensional query processing techniques such as pre-aggregation, yielding an effective approach with low computational overhead and that may be implemented using current technology. The paper illustrates how to implement the approach using SQL databases.

Contents

1 Introduction

On-Line Analytical Processing (OLAP) [7] has attracted much interest in recent years, as business managers attempt to extract useful information from large databases in order to make better informed management decisions. OLAP tools focus on providing fast answers to ad-hoc queries that *aggregate* large amounts of detail data. Recently, the use of OLAP tools have spread to the medical world, where physicians use the tools to understand the data associated with patients. For example, the largest provider of healthcare IT in Denmark, Kommunedata, spends significant resources on applying OLAP technology to medical applications. The use of OLAP tools in the medical domain places additional emphasis on challenges that OLAP technology traditionally has not handled well, such as the handling of imprecise data.

Traditional data models, including the ER model [5] and the relational model, do not provide good support for OLAP applications. As a result, new data models that support a *multidimensional* view of data have emerged. These multidimensional data models typically categorize data as being *measurable business facts* (measures) or *dimensions*, which are mostly textual and characterize the facts. For example, in a retail business, *products* are sold to *customers* at certain *times*, in certain *amounts*, at certain *prices*. A typical fact would be a *purchase*, with the amount and price as the measures, and the customer purchasing the product, the product being purchased, and the time of purchase being dimensions.

If multidimensional databases are to be used for medical OLAP applications, it is necessary that to handle the "imperfections" that almost inevitable occur in the data. Some data values may be *missing*, while others are *imprecise* to varying degrees, i.e., in multidimensional database terms, they 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, but the problem also often occur when combining data from different organizations. Current OLAP tools and techniques assume that the data has a uniform granularity and that any granularity variances are handled in the *data cleansing* process, prior to admitting the data to the OLAP database. This is not a realistic assumption as mapping all data to a common granularity will introduce mapping errors and hide the true quality of the data from the user, possibly leading to erroneous conclusions based on the OLAP queries. Thus, it is very attractive to be able to handle all the occurring forms of imperfect data in order to give the physicians as meaningful and informative answers as possible to their OLAP queries.

The area of "imperfect information" has attracted much attention in the scientific literature [18]. We have previously compiled a bibliography on uncertainty management [9] 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 the focus of this paper. Aggregation of imprecise data has been examined in the context of both possibilistic (fuzzy) databases [24] and (to a lesser extent in) probabilistic databases [11], 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 [26], and in this paper, we similarly use tools from statistics to handle imprecise aggregate data.

The approach presented in this paper aims to maximally re-use existing concepts from multidimensional databases to also support imprecise data. The approach allows the re-use of existing query processing techniques such as *pre-aggregation* for handling the imprecision, resulting in an effective solution that can be implemented using current technology, which is important for the practical application of this research. 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 physician[1] accepts getting an imprecise result, 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 [6], which allow unknown data to be captured explicitly. Multiple imputation [22, 3] 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 [4]. 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 [28] 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 [23, 24] 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* [2] has been used extensively in temporal databases for a variety of purposes, including the handling of imprecision in the data [10]. However, aggregation of imprecise temporal data remains to be studied. In the area of multidimensional databases, only the work on *incomplete data cubes* [8] 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 a novel feature.

The paper is structured as follows. Section 2 motivates our approach by presenting a real-

[1]We use the term "physician" for the user of the system throughout the paper, although the approach presented is general and not limited to the medical domain.

world case study from the clinical world and using it to discuss problems that might arise due to imprecision in the data. Section 3 defines the multidimensional data model and associated query language used as the concrete context for the paper's contribution. Section 4 introduces our approach and shows how to suggest alternative queries if the data is not precise enough. Section 5 shows how to handle imprecision in the grouping of data and in the computation of the aggregate results, and how to present the imprecise result to the physician. Section 6 discusses the use of pre-aggregated data for query evaluation involving imprecision. Section 7 summarizes the article and points to future research topics. Appendix A describes how to implement the approach using SQL databases.

2 Motivation

In this section, we first present a real-world case study from the domain of diabetes treatment. Second, we discuss the queries physicians would like to ask and the problems they encounter due to data imprecision.

The case study concerns data on diabetes patients from a number of hospitals, their associated diagnoses, and their blood sugar levels. The goal is to investigate how the blood sugar levels vary among diagnoses. An ER diagram illustrating the underlying data is seen in Figure 1.

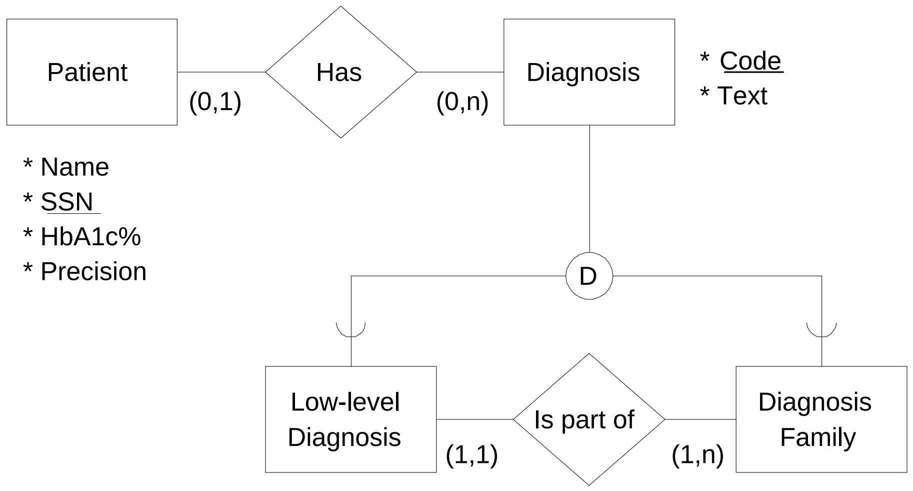

Figure 1: ER Schema of Case Study

The most important entities are the *patients*. For a patient, we record Name and Social Security Number (SSN). The HbA1c% and Precision attributes are discussed later. Each patient may have one *diagnosis*. The diagnosis may be *missing* for some patients, due to incomplete registrations in the computer by the hospital staff. When registering diagnoses for patients, physicians often use different levels of granularity. For example, for some patients, some physicians will use the very precise diagnosis "Insulin dependent diabetes," while the more imprecise diagnosis "Diabetes," which covers a wider range of patient conditions, corresponding to a number of more precise diagnoses, will be used for other patients. In terms of the ER diagram in Figure 1, we model this by having a relationship between patients and the supertype “Diagnosis.” The Diagnosis type has two subtypes, corresponding to different levels of granularity, the *low-level diagnosis* and the *diagnosis family*. Examples of these are the above-mentioned “Insulin dependent diabetes” and “Diabetes,” respectively. The higher-level diagnoses are both (imprecise) diagnoses in their own right, but also function as groups of lower-level diagnoses. Thus, the diagnosis hierarchy groups low-level diagnoses into diagnosis families, each of which consists of 2–20 related diagnoses. Each low-level diagnosis belongs to exactly one diagnosis family. For example, the diagnosis “Insulin dependent diabetes” is part of the family “Diabetes.”

For diagnoses, we record an alphanumeric code and a descriptive text. The code and text are usually determined by a standard classification of diseases, e.g., the World Health Organization’s International Classification of Diseases (ICD-10) [27], but we also allow user-defined diagnoses and groups of diagnoses.

One of the most important measurements for diabetes patients is *HbA1c%* [14], which indicates the long-time blood sugar level, providing a good overall indicator of the patient’s status during the recent months. However, sometimes this value is *missing* in the data that we have available for analysis. This may be because the physician simply did not measure the HbA1c%, or the value may not have been registred in the computer. Furthermore, the HbA1c% is measured using two different methods at the hospitals. Over time, the hospitals change the measurement method from an old, imprecise method to a new and precise method. This leads to a difference in the precision of the data. Thus, we also record the *precision* of the data, as *precise* or *imprecise*. When the value is missing, we record the precision as *inapplicable*.

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 and one-to-many relationships handled using foreign keys. We also assume the use of surrogate keys, named *ID*, with globally unique values. As the two subtypes of the Diagnosis type do not have any attributes of their own, both are mapped to a common Diagnosis table. The “Is part of” relationship is mapped to the “Grouping” table.

The data consists of three patients and their associated diagnoses and HbA1c% values. The resulting tables are shown in Table 1 and will be used throughout the paper.

The primary users are physicians that use the data to acquire important information about the overall state of the patient population. To do so, they issue queries that aggregate the available data in order to obtain high-level information. We use the case study to illustrate the kind of challenges faced by the physicians and addressed by this paper.

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 in which group of low-level diagnoses Jim Doe should be counted. Second, the HbA1c% values themselves are imprecise. Only John Doe has a value obtained with the new, precise measurement method, while Jane Doe has only

ID	Name	SSN	HbA1C%	Precision
0	Jim Doe	11111111	Unknown	Inapplicable
1	John Doe	12345678	5.5	Precise
2	Jane Doe	87654321	7	Imprecise

Patient Table

PatientID	DiagnosisID
0	5
1	3
2	4

Has Table

ID	Code	Text
3	E10	Insulin dependent diabetes
4	E11	Non insulin dependent diabetes
5	E1	Diabetes

Diagnosis Table

ParentID	ChildID
5	3
5	4

Grouping Table

Table 1: Data for the Case Study

an imprecise value and Jim Doe's HbA1c% is unknown.

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

3 Data Model and Query Language Context

This section defines the concepts needed to illustrate our approach to handling imprecision. First, we define an extended multidimensional data model that serves as the context for formulating the approach. Only the parts of the model that are necessary for the subsequent definitions will be given. The full model is described elsewhere [20]. Second, we describe the algebra associated with the model. Third, we define the additional concepts which are necessary for our approach, in terms of the model.

We have chosen to use the presented data model to illustrate our approach, rather than using "standard" models such as star or snowflake schemas, in order to present our approach more clearly. There is several reasons for this choice. First, the presented model allows for a precise, formal definition of multidimensional concepts such as hierarchies and granularities, as opposed to star and snowflake schemas, which only defines these concepts informally. Second, the presented model allows us to map facts directly to dimension values "higher up" in the dimension hierarchy, a feature which our approach uses to capture imprecision. This is not *directly* possible in star or snowflake schemas, but it can be emulated in both of these models, as well as in other multidimensional models. Thus, it it still possible to use our approach with existing multidimensional tools and techniques.

3.1 The Data Model

For every part of the data model, we define the *intension* and the *extension*, and give an illustrating example.

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 we will have *Patient* as the fact type, and *Diagnosis* 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 multidimensional models.

A dimension type $\mathcal{T}$ is a four-tuple $(\mathcal{C}, \leq_\mathcal{T}, \top_\mathcal{T}, \bot_\mathcal{T})$, where $\mathcal{C} = \{\mathcal{C}_j, j = 1, .., k\}$ are the *category types* of $\mathcal{T}$, $\leq_\mathcal{T}$ is a partial order on the $\mathcal{C}_j$'s, with $\top_\mathcal{T} \in \mathcal{C}$ and $\bot_\mathcal{T} \in \mathcal{C}$ being the top and bottom element of the ordering, respectively. Thus, the category types form a lattice. The intuition is that one category type is "greater than" another category type if members of the former's extension logically contain members of the latter's extension, i.e., they have a larger element size. The top element of the ordering corresponds to the largest possible element size, that is, there is only one element in its extension, logically containing all other elements.

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

Example 2 Low-level diagnoses are contained in diagnosis families. Thus, the *Diagnosis* dimension type has the following order on its category types: $\bot_{Diagnosis}$ = *Low-level Diagnosis* $<$ *Diagnosis Family* $< \top_{Diagnosis}$. We have that $Pred($*Low-level Diagnosis*$) = \{$*Diagnosis Family*$\}$. Precise values of HbA1c% are contained in imprecise values[2], e.g., the precise value "5.3" is contained in the imprecise value "5", which covers the range of (precise) values [4.5–5.4]. Thus, other examples of category types are *Precise* and *Imprecise* from the HbA1c% dimension type. Figure 2, to be discussed in detail later, illustrates the dimension types of the case study.

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

[2]The precise measurement method gives us results with one decimal point, while the imprecise method gives us only whole numbers.

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

We assume a partial order $\leq_C$ on the categories in a dimension, as given by the partial order $\leq_{\mathcal{T}}$ on the corresponding category types.

The category $\perp_D$ in 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, $e \leq \top$. Value $\top$ is similar to the *ALL* construct of Gray et al. [12]. 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.

Example 3 In our *Diagnosis* dimension we have the following categories, named by their type. *Low-level Diagnosis* = $\{3, 4\}$, *Diagnosis Family* = $\{5\}$, and $\top_{Diagnosis} = \{\top\}$. The values in the sets refer to the *ID* field in the Diagnosis table of Table 1. The partial order $\leq$ is given by the Grouping table in Table 1. Additionally, the top value $\top$ is greater than, i.e., logically contains, all the other diagnosis values.

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

Example 4 The fact-dimension relation R links patient facts to diagnosis dimension values as given by the Has table from the case study. We get that $R = \{$(0,5), (1,3), (2,4)$\}$. Note that we can relate facts to values in higher-level categories, e.g., fact 0 is related to diagnosis 5, which belongs to the *Diagnosis Family* category. Thus, we do not require that e belongs to $\perp_{Diagnosis}$, as do other multidimensional data models. This feature will be used later to explicitly capture the different granularity in the data. 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 5 For the case study, we get a four-dimensional MO $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = ($*Patient*, $\{$*Diagnosis, HbA1c%*$\})$ and $F = \{0, 1, 2\}$. The definition of the diagnosis dimension and its corresponding fact-dimension relation was given in the previous examples. The HbA1c% dimension has the categories *Precise*, *Imprecise*, and $\top_{HbA1c\%}$. 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 values precise values into the imprecise in the natural way, e.g., $4.5 \leq 5$ and $5.4 \leq 5$ (note that $\leq$ denotes logical inclusion, not less-than-or-equal on numbers). The fact-dimension relation for the HbA1c% dimension is: $R_2 = \{(0, \top), (1, 5.5), (2, 7)\}$. The Name and SSN dimensions are simple, i.e., they just have a $\perp$ category type, Name respectively SSN, and a $\top$ category type. We will refer to this MO as the Patient MO. A graphical illustration of its schema is seen in Figure 2.

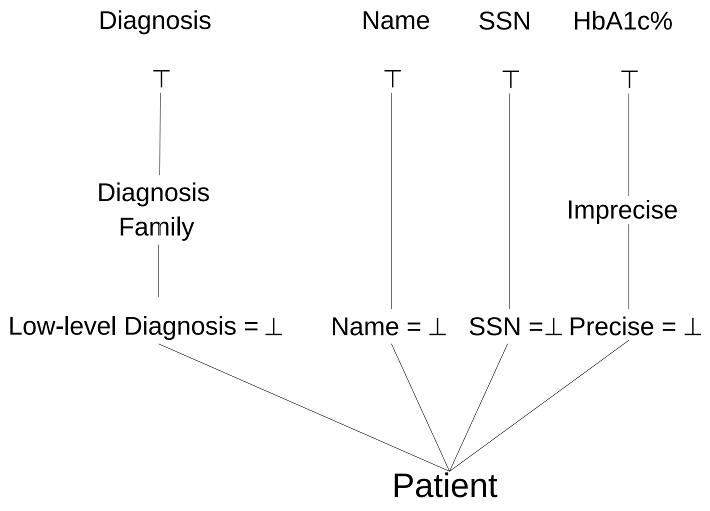

Figure 2: Schema of the Case Study

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.

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 is C_j. Dimension values in the $\perp$ category is said to have the *finest* granularity, while values in the $\top$ category has the *coarsest* granularity.

For dimension $D = (C, \leq)$, 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 $D_i = (C_i, \leq_i)$, 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.

3.2 The Algebra

When handling imprecision, it is not enough to record the imprecision of the data itself. We also need to handle imprecision in the queries performed on the data. Thus, we need a precise specification of the queries that can be performed on the data. To this end, we define an algebraic query language on the multidimensional objects just defined. The focus of this paper is on aggregation, so we will only give the definition of the operator used for aggregation. The other operators of the algebra are close to the standard relational algebra operators, and include selection, projection, rename, union, difference, and identity-based join [20]. The algebra is at least as powerful as Klug's [16] relational algebra with aggregation functions [20].

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

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

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), and a set of categories $C_i \in D_i, i = 1, .., n$, we define aggregate formation, α, as:

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

where

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

$$\mathcal{C}_i' = \{\mathcal{C}_{ij} \in \mathcal{T}_i \mid Type(C_i) \leq_{\mathcal{T}_i} \mathcal{C}_{ij}\}, \leq_{\mathcal{T}_i}' = \leq_{\mathcal{T}_i|_{\mathcal{C}_i'}}, \perp_{\mathcal{T}_i}' = Type(C_i), \top_{\mathcal{T}_i}' = \top_{\mathcal{T}_i},$$

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

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

$$\leq_i' = \leq_{i|_{D_i'}}, R' = \{R_i', i = 1, .., n\} \cup \{R_{n+1}'\},$$

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

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

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

Example 6 We want to know the number of patients in each diagnosis family. 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 five dimensions, but only the Diagnosis and Result dimensions are non-trivial, i.e., the remaining three dimensions contain only the $\top$ categories. The set of facts is $F = \{\{0, 1, 2\}\}$. The Diagnosis dimension is cut, so that only the part from *Diagnosis Family* 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 Family. The content is: $R_1 = \{(\{0, 1, 2\}, 5)\}$, meaning that the set of patients $\{0, 1, 2\}$ is characterized by diagnosis family 5. The fact-dimension relation for the result dimension relate the group of patients to the count for the group. The content is: $R_5 = \{(\{0, 1, 2\}, 3)\}$, meaning that the result of g on the set $\{0, 1, 2\}$ is 3. A graphical illustration of the MO, leaving out the trivial dimensions for simplicity, is seen in Figure 3.

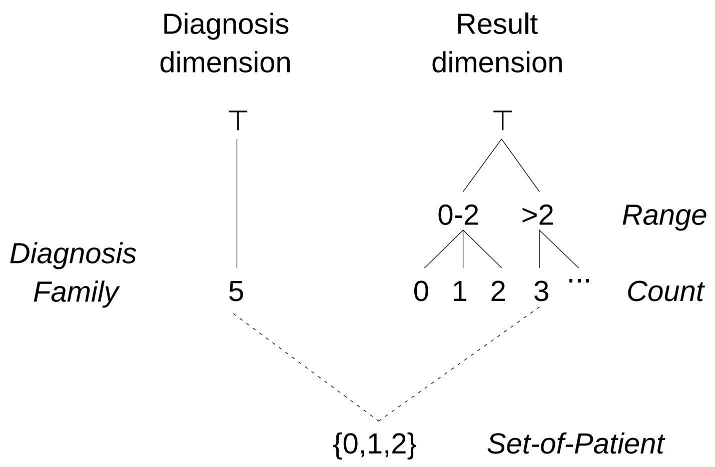

Figure 3: Resulting MO for Aggregate Formation

4 Handling Imprecision

We now describe our approach to handling imprecision in multidimensional data models. 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 granularites coarser than the “grouping” categories.

4.1 Overview of Approach

Along with the model definition, we presented how the case study would be handled in the model. This also showed 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* measurements are contained in one *imprecise* measurement. In turn, several imprecise measurements are contained in the $\top$ (unknown) value. Thus, the approach uses the different levels of the granularity already present 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 4.

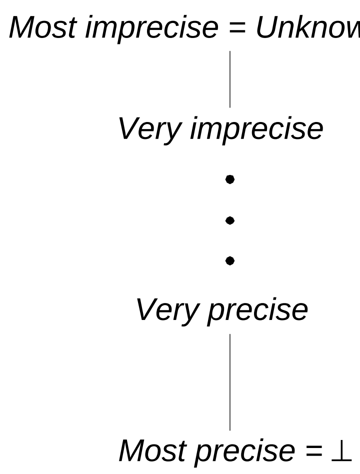

Figure 4: The Spectrum of Imprecision

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 straight away, even though the underlying facts may have *varying* granularities. For example, the query from Example 6 gives us the number of patients diagnosed with diagnoses in the *Diabetes* family, even though two of the patients have low-level diagnoses, while one is diagnosed directly with a Diabetes family. In this case, the data would *not* be precise enough to group the patients by Low-level Diagnosis.

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* exactly so much that the data is precise enough to answer the (alternative) query. Thus, the alternative query will give the most detailed *precise* answer possible, considering the imprecision in the data. For example, if the physician was asking for the patient count grouped by low-level diagnosis, the alternative query would be the patient count grouped by diagnosis family.

If the physician still wants to go ahead with the original query, we need to handle the imprecision explicitly. Examining our algebra, we see that imprecision in the data will only affect the result 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, we need only handle imprecision directly 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 straightforwardly, as 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 [18].

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 evaluation of the query proceeds (logically) 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. The text after the "%" sign are comments.

```
**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 allows for a both simple and efficient handling of imprecision.

4.2 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 Low-level Diagnosis, all fact-dimension relations from patients to the Diagnosis dimension must map to the Low-level Diagnosis category, not to Diagnosis Family 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, $Granularity_i$ 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 can 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 $\mathcal{S} = (\mathcal{F}, \mathcal{D})$, $\mathcal{D} = \{\mathcal{T}_i, i = 1,..n\}$, $\mathcal{T}_i = (\mathcal{C}_i, \leq_{\mathcal{T}_i})$, $\mathcal{C}_i = \{\mathcal{C}_{ij}\}$, $D = \{D_i, i = 1,..,n\}$, 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

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

$$F_p = F, D_p = \{D_{p_i}, i = 1,..,n\}, D_{p_i} = (C_{p_i}, \leq_{p_i}), C_{p_i} = \{Granularity_i, \top_{p_i}\},$$

$$Granularity_i = \{G_{D_i}(e) \mid e \in D_i\}, \top_{p_i} = \{\top_i\},$$

$$e_1 \leq_{p_i} e_2 \Leftrightarrow (e_1 = e_2) \vee (e_1 \in Granularity_i \wedge e_2 = \top_i, \text{ and}$$

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

Example 7 The MO from Example 5 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 $Gran_{Diagnosis}$ and $Gran_{HbA1c\%}$. The dimension type $Gran_{Diagnosis}$ has the category types $Granularity_{Diagnosis}$ and $\top_{GranDiagnosis}$. The dimension type $Gran_{HbA1c\%}$ has the category types $Granularity_{HbA1c\%}$ and $\top_{GranHbA1c\%}$. The set of facts is the same, namely $F_p = \{0, 1, 2\}$. Following the dimension types, there are two dimensions, $Gran_{Diagnosis}$ and $Gran_{HbA1c\%}$. The $Gran_{Diagnosis}$ dimension has the categories $Granularity_{Diagnosis}$ and $\top_{GranDiagnosis}$. The values of the $Granularity_{Diagnosis}$ category is the set of category types $\{$ *Low-level Diagnosis*, *Diagnosis Family*, $\top_{Diagnosis}\}$. The $Gran_{HbA1c\%}$ dimension has the categories $Granularity_{HbA1c\%}$ and $\top_{GranHbA1c\%}$. The values of the $Granularity_{HbA1c\%}$ category is the set $\{$ *Precise*, *Imprecise*, $\top_{HbA1c\%}\}$. 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 $R_1 = \{(0,$ *Diagnosis Family*$), (1,$ *Low-level Diagnosis*$), (2,$ *Low-level Diagnosis*$)\}$ and $R_2 = \{(0, \top_{HbA1c\%}), (1, Precise), (2, Imprecise)\}$. A graphical illustration of the precision MO is seen in Figure 5.

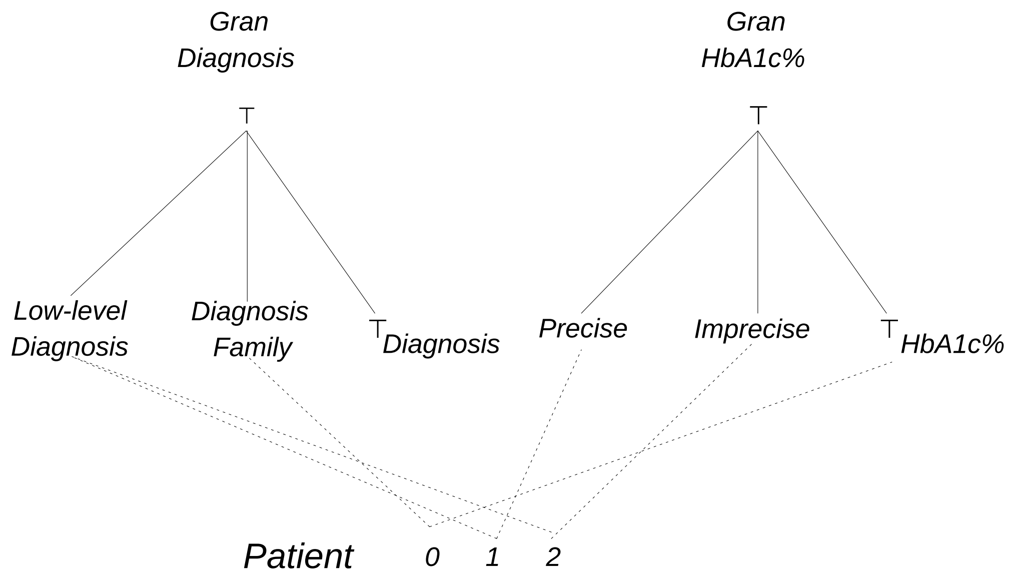

Figure 5: Precision MO

The test to see if the data is precise enough to answer a query Q can be 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 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 $\mathit{Type}(C_i) \leq_{\mathcal{T}_i} \mathit{Type}(C'_i) \wedge \forall C_{ij}(f \in F \wedge (f,e) \in R_i \wedge e \in C_{ij} \Rightarrow \mathit{Type}(C_{ij}) <_{\mathcal{T}_i} \mathit{Type}(C'_i))$, 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. The physician can then use this additional information to decide whether to run the alternative query or proceed with the original one.

Example 8 The physician wants to know the average HbA1c% grouped by Low-level Diagnosis. The query asked is then $Q = \alpha[$*Low-level Diagnosis* $, \top_{HbA1c\%}, D_3, AVG_2](M)$, thus effectively grouping only on Low-level Diagnosis, as the $\top_{HbA1c\%}$ component has only one value. The testing query then becomes $Q_p = \alpha[\mathit{Granularity}_{Diagnosis}, \top_{GranHbA1c\%}, D_3, \mathit{SetCount}](M_p)$, which counts the number of facts with the different Diagnosis granularity levels. The result of Q_p, described by the fact-dimension relations, is $R_1 = \{(\{1,2\},$ *Low-level Diagnosis*$), (\{0\},$ *Diagnosis Family*$)\}$, $R_2 = \{(\{1,2\}, \top_{GranHbA1c\%}), (\{0\}, \top_{GranHbA1c\%})\}$, and $R_3 = \{\{(\{1,2\},2), (\{0\},1)\}$. This tells us that 2 patients have a low-level diagnosis, while 1 has a diagnosis family diagnosis. Thus, the alternative query will be $Q = \alpha[$*Diagnosis Family*, $\top_{HbA1c\%}, D_3, AVG_2](M)$, which groups on Diagnosis Family rather than Low-level Diagnosis.

5 Handling 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 to handle imprecision in the grouping of data and in the computation of aggregate functions, followed by presenting the imprecise result to the physician.

5.1 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 that is *known* to belong to that group, and discards the data that is not precise enough 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 how likely the membership is. 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. We proceed to investigate how to compute the answers.

The conservative grouping is quite easy to compute. We just apply the *standard* aggregate formation operator from the algebra, which by default 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.

For the liberal grouping, we need to additionally capture the data that are 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 6, 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:

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

Thus, we allow the dimension values to range over the categories that have coarser or the same granularity as the grouping categories. We group according to the *most precise* values, of a granularity at least as coarse as the grouping categories, that characterize a fact.

Example 9 If we want to know the number of patients, grouped by Low-level Diagnosis, 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 low-level diagnoses and one for the Diabetes diagnosis family. The fact-dimension relations are $R_1 = \{(\{0\}, 5), (\{1\}, 3), (\{2\}, 4)\}$ 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 Low-level Diagnosis, 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 $Group^m(e_1, .., e_n) = \cup_{e'_1 \geq_1 e_1, .., e'_n \geq_n e_n} Group(e'_1, .., e'_n)$, 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 10 If we want the number of patients, grouped *liberally* by Low-level Diagnosis, we will get the set of facts $F' = \{\{0, 1\}, \{0, 2\}\}$, meaning that patient 0 goes into both of the two low-level diagnosis groups. The fact-dimension relations are $R_1 = \{(\{0, 1\}, 3), (\{0, 2\}, 4)\}$ and $R_2 = \{(\{0, 1\}, 2), (\{0, 2\}, 2)\}$. We see that each patient is mapped to all the low-level diagnoses that might be true for the patient. The count for each group is 2, meaning that for each of the two low-level diagnoses, there might be two patients with that diagnosis. Of course, this cannot be true for both diagnoses simultaneously.

The liberal approach overrepresents 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 only reflected once. It is desirable to get a result where the imprecise facts are reflected at most once in the overall result.

To do so we introduce a *weight* w for each fact f in a group, making the group a *fuzzy set* [28]. 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 $\leq$ on dimension values. For each pair of values e_1, e_2 such that $e_1 \leq e_2$, we assign a weight p, using the notation $e_1 \leq (p)e_2$, 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 $\Sigma_{e_1 \in C \wedge e_1 \leq (p) e}\ p = 1$, i.e., the weights for one dimension value w.r.t. any given category adds up 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 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 a 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 11 We know that 80% of Diabetes patients have insulin-dependent diabetes, while 20% have non-insulin-dependent diabetes. Thus, we have that $3 \leq (.8)5$ and $4 \leq (.2)5$, 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 weight on all other links is 1. Again, we want to know the number of patients, grouped by Low-level Diagnosis. The *Group* function divides the facts into two sets with weighted facts, giving the set of facts $F' = \{\{0_{.8}, 1_1\}, \{0_{.2}, 2_1\}$. Using subscripts to indicate membership weighting, the result of the computation is given in the fact-dimension relations $R'_1 = \{(\{0_{.8}, 1_1\},$ *Insulin-dependent Diabetes*$), (\{0_{.2}, 2_1\},$ *Non-insulin-dependent Diabetes* $)\}$ and $R'_2 = \{(\{0_{.8}, 1_1\}, 1.8), (\{0_{.2}, 2_1\}, 1.2)\}$, meaning that the weighted count for the group containing the insulin-dependent diabetes patients 0 and 1 is 1.8 and the count for the non-insulin-dependent diabetes patients 0 and 2 is 1.2.

5.2 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 here to compute the resulting aggregate value by "imputing" precise values for imprecise values, and 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 dimensions 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* [22]. 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.

However, using only generalized imputation, we do not know how precise the result is. To determine the precision of the result, 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 : CM $\times$ CM $\mapsto$ 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 : CM $\mapsto$ FM, that maps a computation measure value to a final measure value. The reason to distinguish between computation measure and final measures is only that this allows us to require that the MCF is 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 12 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 $CM = \mathcal{N} \times \mathcal{N}$, the pairs of natural numbers, and the GCM value for a dimension value e is $(Level(e), 1)$. 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 has level 0, imprecise values such as 5 has level 1, and the $\top$ value has level 2.

Example 13 The *standard deviation* $\sigma(X)$ of a set of values X from the average value $e(X)$ is a widely used estimate 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 choice of MCF means that the MCF is distributive and symmetric [25]. 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 get a precise estimate of the natural variation in the data as the imprecision measure, just as we would get using *multiple imputation* [22, 3].

For both the *conservative* and the *liberal* answer, 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 weight, both in the computation of the aggregate result and in the computation of the precision. We note that for aggregation functions g whose result depend only on one value in the group it is applied to, such as MIN and MAX, we get the minimum/maximum of the *expected values*.

Example 14 We want to know the average HbA1c% for patients, grouped by Low-level Diagnosis, and the associated precision of the results. As granularity measures and functions, we use the *level* approach described in Example 12. We discuss only the weighted result. As seen in Example 11, the resulting set of facts is $F' = \{\{0_{.8}, 1_1\}, \{0_{.2}, 2_1\}$, 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.

5.3 Presenting the Imprecise Results

The final step in the imprecision handling is to *present* the imprecision in the result to the physician. We have several 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 very precise estimate of the precision of a result value.

Example 15 For the example above, this would present the (*Low-level Diagnosis*, 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$, 6.5, 1.5)$. 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 (coarse) the value into a value of a granularity corresponding to the imprecision. A *value coarsing function (VCF)* for a dimension D and a FGM M is a function $c : \perp_D \times M \mapsto D$, where $c(e) = e_1$ such that $e \leq e_1$. 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 16 We choose the HbA1c% dimension, with the same granularities, as the result dimension. As the VCF we choose $r(x) = v$ such that $x \leq v \wedge Level(v) = Ceiling(x)$, 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 6. 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, yielding the resulting average value very imprecise.

6 Using Pre-aggregated Data

The approach we have presented above 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

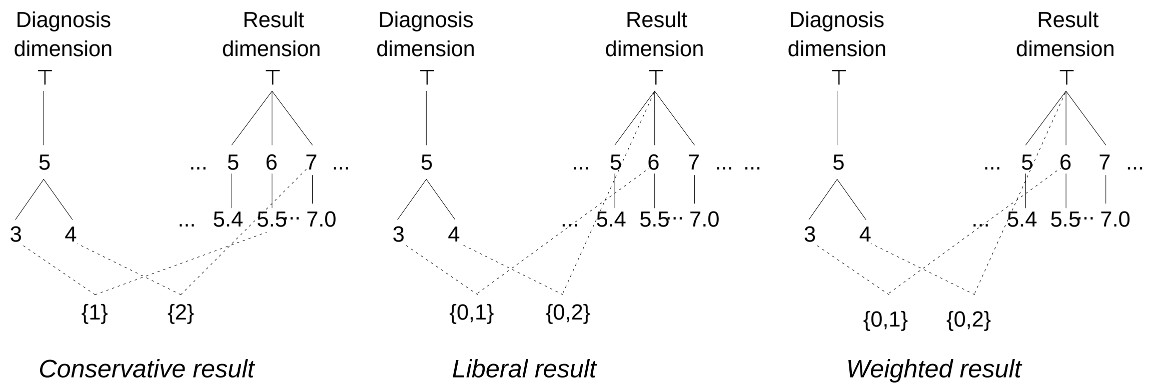

Figure 6: Resulting MO's for the Conservative, Liberal, and Weighted Answers

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 [23, 24, 4], it is attractive to lower the running time of queries even further. A very 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 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 updated 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 [19]. 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 [17], which intuitively means that lower-level results can be directly combined into higher-level results. It has been proven [17] that summarizability is equivalent to the hierarchies in dimensions being *strict*, *partitioning*, and *complete*, i.e., one lower-level dimension value map to exactly one higher-level value, and for every higher-level value there exist at least one lower-level value that map 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.

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 4.2. With 10 dimensions and 4 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 also quite dense. We need to store the result of the *SetCount* operation for each combination of dimension values. This does not take up very much space, so 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 [17] of the finest granularity, that "takes the place" of a coarser value. In our case, we 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. When doing this, we also get the side benefit that the *liberal* result is automatically 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 Low-level Diagnosis (and $\top_{HbA1c\%}$), we would issue queries that grouped by Low-level Diagnosis and $\top_{HbA1c\%}$, by Diagnosis Family and $\top_{HbA1c\%}$, and by $\top_{Diagnosis}$ 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 $\top_{Diagnosis}$ is 3, the count for Diabetes is also 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 $\top_{Diagnosis}$.

We also need to consider pre-aggregation in relation to the computation of the precision. The values that should be pre-aggregated is 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 applies. 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.

7 Conclusion and Future Work

Motivated by the increasing use of OLAP technology for medical applications, we investigate how to solve one of the most common problems with medical (as well as other) data, namely data imprecision, using concepts from the multidimensional data models most commonly used in OLAP systems.

The approach described in this paper generally uses the concept of *data granularity* to handle imprecision in the data. To have a concrete context for presenting our technique, we present a multidimensional data model and an associated algebraic query language that facilitate formal definition of the concepts used in the technique. Data imprecision is handled by first *testing* if the data is precise enough to answer a query precisely. If this is not the case, an *alternative query* that might be answered precisely is suggested. If the physician asking the query elects to proceed with the original query, the imprecision in the data is reflected in the *grouping* of data, as well as in the *aggregate computation*. The physician is presented with the three results. The *conservative* result includes only what is *known* to be true, the *liberal* answer includes everything that *might* be true, while the *weighted* answer includes everything that might be true, but gives precise data higher weights than imprecise data. Along with the aggregate computation, a separate computation of the *precision* of the result is carried out. As the last part of imprecise query handling, the imprecise result is presented to the physician. We discuss how to use pre-aggregated data for more efficient query processing and how to implement the approach using SQL.

Compared to previous approaches to handling imprecision, this work improves by showing how existing concepts and techniques from multidimensional databases, such as granularities and pre-aggregation, can be maximally re-used to also support imprecision. This yields an effective approach that can be implemented using current technology. Additionally, imprecision is handled for both the grouping of data and in the aggregate computation.

In future work, it would be interesting to pursue a more theoretical investigation of how to implement the technique using special-purpose data structures and algorithms, to achieve optimal concrete complexity. A further investigation of the issues related to “single-value” aggregation functions such as MIN and MAX in relation to data granularity is also interesting. Unlike other aggregation functions, these are not readily sensitive to weighting. We have showed how to present data imprecision in the result using the existing granularities, but it would also be very interesting to explore other means of graphically presenting imprecision in the result to facilitate the user interpretation of an imprecise result. Another issue for future research, related to presentation of the imprecise result, is to present the user with the data that *prevented* a given query from being precisely answerable, allowing the user to reformulate the query to avoid this data or to seek and obtain more precise data from outside sources.

The presented technique is applicable for the common case where the data has a degree of imprecision that cannot be ignored, but data precision in any given dimension is reasonably high compared to the precision requested by the queries. If the data is very imprecise, this technique will not be so helpful, as bad data can only produce bad results. An interesting topic for future research would be to give precise measures for the usefulness of technique, given the available data. In the cases where the presented technique is less useful, it would be interesting to investigate whether a combination with other known techniques for handling imprecision could widen the scope of applicability.

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] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, X. S. Wang. A Glossary of Time Granularity Concepts. In *Temporal Databases: Research and Practice*, pp. 406–413. LNCS 1399, Springer-Verlag, 1998.

[3] S. van Buuren, E. V. van Mulligen, 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.

[4] 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.

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

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

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

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

[9] C. E. Dyreson. A Bibliography on Uncertainty Management in Information Systems. In [17], pp. 413–458, 1997.

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

[11] 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.

[12] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.

[13] 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.

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

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

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

[17] 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.

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

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

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

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

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

[23] 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.

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

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

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

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

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

A SQL Implementation

This section discusses how to implement the presented approach using commercial relational databases. The goal is to provide a mapping to relational tables and a set of query templates that allows 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* [15]. 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*, even 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 [15], 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 [21], a technique optimized for "multidimensional" relational queries. If the performance obtained with this design is not sufficient, we can denormalize the fact table by putting the expected values (EV) and the granularity computation measures (GCM) into it. This gives a schema of the form $(FactId, DK_1, EV_1, GCM_1, .., DK_n, EV_n, GCM_n)$. 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, partitioning, and complete, 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 [15]. 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 $(DK, EV_\perp, GCM_\perp, DV_\perp, W_\perp, .., DV_\top, W_\top)$.

However, we would like to capture explicitly in the relational schema the situation that facts are mapped directly to dimension values of a coarser granularity. 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, 13], 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 $(A, L_A, D, L_D, W, EV_D, GCM_D)$.

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 $g(Comb(D_1.W, .., D_m.W) * D_k.EV)$, $GCF(Comb(D_1.W, .., D_m.W) * D_k.GCM)$
FROM $F, D_1, .., D_m$
WHERE
        $F.DK_1 = D_1.DK$ AND ... AND $F.DK_m = D_m.DK$ AND
        $F.DK_k = D_k.DK$ AND $D_k.L_A = D_k.L_D$ AND
        $(D_1.L_A = GL_1$ OR $(D_1.L_A > GL_1$ AND $D_1.L_A = D_1.L_D))$ AND
        ....
        $(D_m.L_A = GL_m$ OR $(D_m.L_A > GL_m$ AND $D_m.L_A = D_m.L_D))$
GROUP BY $D_1.A, .., D_m.A$
```

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, $D_1, .., D_m$ 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 join 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, and ensures that we only get one value for each fact. The following lines of the WHERE clause handle the grouping of facts. The part before the “OR” handles the *conservative* grouping, while the remainder handles the additional data in the *liberal* grouping.

Example 17 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 2. When using SQL to compute the weighted average of the HbA1c%, grouped by Low-level diagnoses seen in Example 14, we get 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 14. 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
```

Fact	DiagKey	HbA1Key
0	5	6
1	3	7
2	4	8

Fact Table

AnsID	DesID	Ancestor	AnsLevel	DesLevel	W
3	3	Ins. dep. diab.	0	0	1
4	4	Non-ins. dep. diab.	0	0	1
5	5	Diabetes	1	1	1
5	3	Diabetes	1	0	.8
5	4	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 2: Relational Implementation of the Case Study

```
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 to store 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 [15] to access the aggregate tables.