Aspects of Data Modeling and Query Processing for Complex Multidimensional Data

Torben Bach Pedersen

Ph.D. Dissertation

A dissertation submitted to the faculty of Engineering and Science at Aalborg University, Denmark, in partial fulfillment of the requirements for the Ph.D. degree in computer science.

Aspects of Data Modeling and Query Processing for Complex Multidimensional Data

Torben Bach Pedersen

Ph.D. Dissertation

A dissertation submitted to the faculty of Engineering and Science at Aalborg University, Denmark, in partial fulfillment of the requirements for the Ph.D. degree in computer science.

Danish Academy of Technical Sciences (ATV) and
Erhvervsfremmestyrelsen.
Industrial Research Education, Ph.D., EF–661.

Project Title	The Clinical Data Warehouse
Company	Kommunedata A/S, Århus
Institute and University	Department of Computer Science, Aalborg University

Abstract

This thesis is about data modeling and query processing for complex multidimensional data. Multidimensional data has become the subject of much attention in both academia and industry in recent years, fueled by the popularity of data warehousing and On-Line Analytical Processing (OLAP) applications.

One application area where complex multidimensional data is common is within medical informatics, an area that may benefit significantly from the functionality offered by data warehousing and OLAP. However, the special nature of clinical applications poses different and new requirements to data warehousing technologies, over those posed by conventional data warehouse applications. This thesis presents a number of exciting new research challenges posed by clinical applications, to be met by the database research community. These include the need for complex-data modeling features, advanced temporal support, advanced classification structures, continuously valued data, dimensionally reduced data, and the integration of complex data.

OLAP systems typically employ multidimensional data models to structure their data. This thesis identifies eleven modeling requirements for multidimensional data models. These requirements are derived from a realistic assessment of complex data found in real-world applications. A survey of twelve multidimensional data models reveals shortcomings in meeting some of the requirements. Existing models do not support many-to-many relationships between facts and dimensions, do not have built-in mechanisms for handling change and time, lack support for imprecision, and are unable to insert data with varying granularities. Additionally, most of the models do not support irregular dimension hierarchies and aggregation semantics. This thesis defines an extended multidimensional data model and algebraic query language that address all eleven requirements. The model reuses the common multidimensional concepts of dimension hierarchies and granularities to capture imprecise data. For queries that cannot be answered precisely due to the imprecise data, 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. In addition, alternative queries unaffected by imprecision are offered. The presented data model and query evaluation techniques can be implemented using relational database technology. The approach is also capable of exploiting multidimensional query processing techniques like pre-aggregation. This yields a practical solution with low computational overhead.

Pre-aggregation, the prior materialization of aggregate queries for later use, is an essential technique for ensuring adequate response time during data analysis. Full pre-aggregation, where all combinations of aggregates are materialized, is infeasible. Instead, modern OLAP systems adopt the *practical pre-aggregation* approach of materializing only select combinations of aggregates and then re-use these for efficiently computing other aggregates. However, this re-use of aggregates is contingent on the dimension hierarchies and the relationships between facts and dimensions satisfying stringent constraints. This severely limits the scope of the practical pre-aggregation approach. This thesis significantly extends the scope of practical pre-aggregation to

cover a much wider range of realistic situations. Specifically, algorithms are given that transform "irregular" dimension hierarchies and fact-dimension relationships, which often occur in real-world OLAP applications, into well-behaved structures that, when used by existing OLAP systems, enable practical pre-aggregation. The algorithms have low computational complexity and may be applied incrementally to reduce the cost of updating OLAP structures. The transformations can be made transparently to the user. A prototype implementation of the techniques is reported.

OLAP systems provide good performance and ease-of-use for queries that aggregate large amounts of data. However, the complex structures and relationships inherent in data in non-standard applications are not accommodated well by OLAP systems. In contrast, object database systems are built to handle such complexity, but do not support OLAP-type querying well. This thesis presents the concepts and techniques underlying a flexible, "multi-model" federated system that enables OLAP users to exploit simultaneously the features of OLAP and object systems. The system allows data to be handled using the most appropriate data model and technology: OLAP systems for dimensional data and object database systems for more complex, general data. As a vehicle for demonstrating the capabilities of the system, a prototypical OLAP language is defined and extended to naturally support queries that involve data in object databases. The language permits selection criteria that reference object data, queries that return combinations of OLAP and object data, and queries that group dimensional data according to object data. The system is designed to be aggregation-safe, in the sense that it exploits the aggregation semantics of the data to prevent incorrect or meaningless query results. These capabilities may also be integrated into existing languages. A prototype implementation of the system is reported.

Acknowledgements

I owe my thanks for contributions to this thesis to a great many persons. First of all, I would like to thank my Ph.D. advisor, Christian S. Jensen, for his great interest in, and support of, the work that led to this thesis. He has taught me, primarily by his own example, how to conduct high-quality research at an international level. I would also like to thank my industrial advisor, Preben Etzerodt, for providing a good work environment for me at Kommunedata. My co-author Curtis Dyreson showed me how to conduct research while still maintaining the spirit of a true surfer.

Together with my family, I spent seven wonderful months in Berkeley, California, being the guest of Dr. Arie Shoshani in the Scientific Data Management Group at Lawrence Berkeley National Laboratory. Arie and my colleagues at LBNL made my stay great both scientifically and personally. I would like to thank Arie for being a source of constant inspiration and good discussions, and Junmin Gu and Henrik Nordberg for the many nice lunches we had together. I hope to come back for another visit someday.

Thanks are also due to my great colleagues at Kommunedata and in the Database Group at Aalborg University. Especially, I would like to thank Henning Peter Jensen for always reminding me of the value of practical work.

Last, but certainly not least, I would like to thank my family for all their support. Especially, I would like to thank Pia for being a “good researcher’s wife” as well as a wonderful woman, and Andrea and Amalie for being so good at steering my mind away from research.

This work was supported by Kommunedata and by grant no. EF-661 from the Danish Academy of Technical Sciences.

Contents

Chapter 1

Introduction

The interest in analyzing data has grown tremendously in recent years, as businesses in all sectors have discovered the potential of using the data scattered in diverse business systems as one coherent whole for better understanding and management of the business. To analyze data, a multitude of technologies is needed, namely technologies from the areas of Data Warehousing (DW), On-Line Analytical Processing (OLAP), Data Mining (DM), Data Visualization (DV), and Customer Relationship Management (CRM). The market for these technologies is already large, and will grow rapidly in the coming years. The well-known analysis firm Meta Group estimates that the DW market alone will reach USD 15 billion in 2000 [76], while the analysis firm Palo Alto Management Group expects the larger *business intelligence* market, which consists of DW, OLAP, DM, DV, and CRM, to grow by an average of 50% over the next three years and reach a total of USD 113 billion in 2002 [91].

Data analysis systems are increasingly based on a *multidimensional* data model, in which measured values, termed facts, are characterized by descriptive values, drawn from a number of dimensions; and the values of a dimension are typically organized in a containment-type hierarchy. A prototypical query applies an aggregate function, such as average, to the facts characterized by specific values from the dimensions. Multidimensional data models are used as they are generally better suited for data analysis tasks than other data models such as the relational, entity/relationship, or object-oriented models. Compared to these models, multidimensional models provide additional ease-of-use and better performance for the specific queries used for data analysis.

However, some application areas cannot be handled satisfactorily using current multidimensional technology, as the data is too complex. One such area is *clinical data*, which has been the source of inspiration for the research that led to this thesis. This thesis is the result of an industrial Ph.D. project and is a collaboration between the Department of Computer Science at Aalborg University and Kommunedata, the largest supplier of healthcare informatics in Denmark. As a concrete benefit of the collaboration, all the examples used in the thesis are real-world case studies based on Kommunedata's systems. The study of how data analysis technology can be applied to the clinical area has led to the focus of this thesis, namely that of solving some of the problems related to using multidimensional technology for handling complex

multidimensional data, e.g., as found in clinical applications. The remainder of the thesis is structured as follows.

Chapter 2 investigates the exciting new challenges that data warehousing and OLAP technology face from the area of clinical data warehousing. Challenges especially important to the general database research community include the following: advanced data models including temporal support, advanced classification structures, continuously valued data support, dimensional reduction of data, and integration of complex data.

Chapter 3 presents eleven requirements that multidimensional data models data should support to accommodate analysis of complex multidimensional data. Twelve previously proposed data models are evaluated against the eleven requirements, and it is seen that existing models do not support many-to-many relationships between facts and dimensions, do not have built-in mechanisms for handling change and time, lack support for imprecision, and are unable to insert data with varying granularities. Additionally, most of the models do not support irregular dimension hierarchies and aggregation semantics. Chapter 3 presents an extended multidimensional data model and algebraic query language that addresses all eleven requirements. In particular, imprecise data is handled using the common multidimensional constructs of dimension hierarchies and granularities. The presented data model and query evaluation techniques can be implemented using standard OLAP technology such as RDBMSes and the query performance can be enhanced using pre-aggregation.

Chapter 4 investigates the practical use of pre-aggregated data over irregular OLAP hierarchies. The scope of practical pre-aggregation is significantly extended to cover a much wider range of realistic situations. Specifically, algorithms are given that transform irregular dimension hierarchies and fact-dimension relationships, which often occur in real-world OLAP applications, into well-behaved structures that, when used by existing OLAP systems, enable practical pre-aggregation. The algorithms have low computational complexity and can be applied incrementally to reduce the cost of updating OLAP structures. The transformations can be made transparently to the user.

Chapter 5 presents the concepts and techniques underlying a flexible, "multi-model" federated system for extending OLAP querying to external object databases. The system eases the integration of OLAP data with complex, external data considerably and allows data to be handled using the most appropriate data model and technology: OLAP systems for dimensional data and object database systems for more complex, general data. A prototypical OLAP language is defined and extended to naturally support queries that involve data in object databases. The language permits selection criteria that reference object data, queries that return combinations of OLAP and object data, and queries that group dimensional data according to object data. The system is designed to be aggregation-safe, in the sense that it exploits the aggregation semantics of the data to prevent incorrect or meaningless query results. A prototype implementation of the system is reported.

Appendix A surveys the field of clinical data warehousing, both from an industrial and an academic point of view, as of Mid 1997. Seven evaluation criteria are presented and nine products and projects are evaluated against them, giving a good overview of the (by then) current state of the art. The field is still in its infancy, but

the potential for clinical benefits of the technology is large. However, the products surveyed do not address several advanced requirements for clinical use, including richer data models, temporal support, and intelligent integration of complex data.

Appendix B presents the TreeScape system that, unlike any other system known to the authors, enables the reuse of pre-computed aggregate query results for irregular dimension hierarchies, which occur frequently in practice. The system establishes a foundation for obtaining high query processing performance while pre-computing only limited aggregates, even when the hierarchies are irregular. This is done using the dimension transformation techniques described in Chapter 4. It is described how the transformations can be made transparent to the user by applying a query re-write mechanism.

Appendix C presents the OLAP++ system for federating OLAP and object databases. The system allows users to easily pose OLAP queries that reference external object databases. This enables very flexible and fast integration of object data in OLAP systems without the need for prior physical integration. It is shown how the user interface allows the user to easily specify OLAP queries over the federation and how these queries are processed.

This thesis is organized as a collection of individual papers. This means that the individual chapters and appendices are self-contained and can be read in isolation, e.g., related work is treated in each individual paper. However, it also means that there are small overlaps, mostly in basic definitions, between the chapters. Specifically, Section 4.3.1 may be skipped when reading the thesis from the beginning. Sections 4.3.2, 5.3.1, and 5.3.2 mostly contain material already covered in Section 3.3, but should be read, as they also include additional material that is important for the topics of the chapters. Most importantly, the data model presented in Section 5.3.1 is somewhat different from the model used in Chapters 3 and 4 due to the introduction of measures with automatic aggregation. Appendices B and C cover some material already presented in Chapters 4 and 5, respectively, but most of their contents is new material on the implementation of the systems.

The papers have been modified slightly to support the integration into the thesis, e.g., the references have been combined into one bibliography and references to "this paper" changed to "this chapter," etc. The papers included in the thesis are listed below. Chapter 2 is based on Paper 1. Chapter 3 is based on Paper 4, which is an integration of the extended versions of Paper 2 [95] and Paper 3 [96]. Chapter 4 is based on the extended version of Paper 5 [97]. Chapter 5 is based on an extended version of Paper 6. Appendix A is based on Paper 7. The demonstration proposals in Appendices B and C are based on Papers 8 and 9, respectively.

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

2. 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.

3. 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 Statistical and Scientific Database Management*, pp. 90–101, 1999.

4. T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Manuscript, awaiting submission*, March 2000, 35 pages.

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

6. T. B. Pedersen, A. Shoshani, J. Gu, and C. S. Jensen. Extending OLAP Querying to Object Databases. *Submitted for publication*, February 2000, 23 pages.

7. T. B. Pedersen and C. S. Jensen. Clinical Data Warehousing - A Survey. In *Proceedings of the VIII Mediterrannean Conference on Medical and Biological Engineering and Computing*, Section 20.3 (CDROM proceedings), 1998, 6 pages.

8. T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. The TreeScape System: Reuse of Pre-Computed Aggregates over Irregular OLAP Hierarchies. *Demonstration proposal, submitted for publication*, February 2000, 4 pages.

9. J. Gu, T. B. Pedersen, and A. Shoshani. OLAP++: Powerful and Easy-to-Use Federations of OLAP and Object Databases. *Demonstration proposal, submitted for publication*, February 2000, 4 pages.

Chapter 2

Research Issues in Clinical Data Warehousing

2.1 Introduction

Modern businesses use a multitude of different computer systems to manage their daily business processes such as sales, production, planning, etc. These systems, commonly referred to as *operational systems*, have been acquired from several vendors over a long period of time and are often based on different technologies. The integration between the operational systems is thus typically poor. However, integration is needed when the business must combine data from several operational systems in order to answer important business questions, e.g., sales and production data must be combined to determine the profitability of a product. The *data warehousing* approach solves the problem by integrating data from the operational systems into one common data store, known as the data warehouse, which is optimized for data analysis purposes [130, 129].

Data warehousing technology has traditionally been used in a business context, in order to answer questions about sales and other important events in the business of concern. The data models employed conceptually provide a multidimensional view of data, whether implemented in relational or dedicated multidimensional DBMS's, and this has proven very successful in the traditional application areas. However, some application areas have a need for more complex data structures. One such area is clinical data warehousing, where clinical data about a large patient population is analyzed to perform clinical quality management and medical research. Clinical data warehousing is a substantial application area in itself, and we focus on describing the requirements of this area. The issues described also apply to other application areas, in science or business, but such areas are beyond the scope of this chapter. We will also concentrate on the use of clinical data for analysis purposes. Discussion of the operational use of clinical data, e.g., for cooperative purposes or remote diagnostization, is also not covered here.

The clinical domain requires more powerful data model constructs than conventional multidimensional approaches, and the data model should also provide advanced temporal support, e.g., for bitemporal data. More advanced classification

structures are also needed, including means of managing dynamic, non-strict hierarchies, and of handling change. Continuously valued data, e.g., measurements, is very common and has special demands for aggregation and computation compared to conventional business data. The number of dimensions in clinical data is often very large, sparking a need for intelligent ways of dimensionally reducing the data into high-level abstractions.

There should also be a way of integrating very complex data, e.g., X-rays, in the data warehouse for analysis purposes, by more advanced means than just allowing the raw data to be retrieved. Clinical treatment protocols should be tightly integrated with the clinical data warehouse, to allow for follow-up on the corresponding quality of treatment, e.g., outcomes, for the individual protocols. Finally, medical research should be supported directly by the clinical data warehouse, e.g., by integrating data mining capabilities tailored to the specific domain.

The chapter is outlined as follows. Section 2 describes the conventional use of data warehousing, as used primarily in business settings. To illustrate the various issues, a case study concerning a small clinical data warehouse is included. Section 3 describes the concept of an Electronic Patient Record (EPR), and lays out a roadmap for a new foundation for Clinical Data Warehouses (CDWs). The primary rationale is that a CDW should be very tightly integrated with the EPR, to support physicians and other clinical users throughout their daily work. We argue that it is attractive to base the CDW on the EPR and introduce EHCRA, the European Standard for EPR's. This standard has some nice features w.r.t. using EPR data for data warehousing. Section 3 also describes the research challenges that CDWs provide, and it compares CDWs with ordinary data warehouse applications. Section 4 summarizes the chapter and offers suggestions for next steps.

2.2 Background

This section provides a definition of a data warehouse, describes previous work, and presents a case study of a CDW.

2.2.1 A Brief Characterization of Data Warehousing

The term "Data Warehouse" (DW) was first used by Barry Devlin [28], but Bill Inmon has won the most acclaim for introducing the concept, defined as follows. "A Data Warehouse is a *subject oriented*, *integrated*, *non-volatile* and *time-variant* collection of data in support of *management's decisions*" [58]. Let us have a closer look at these interesting properties.

- *Subject Oriented:* In operational systems, data is organized to support specific business processes. Thus, the same data might be organized very differently in different operational systems. For example, it is likely that person data in a Human Resource application is organized differently from person data in a Point-of-Sale application. In a DW, data is organized by subject, or topic, e.g., Person, rather than by function.

- *Integrated:* A business typically employs many different operational systems, each optimized for a special business process, and each with its own data store. In the DW, data from all these systems is integrated, both by *definition*, i.e., the same data has the same type, and by *content*, i.e., the value sets of an attribute are the same, wherever they occur. Integration does not imply data warehousing—an appropriate organization is also required. If all operational data is in one operational system, e.g., the SAP system, it is still necessary to have a DW, where data is organized w.r.t. data analysis instead of data entry [1].

- *Non-volatile:* In the typical operational system, data is often kept only for a short period of time, e.g., 3 to 6 months, as it is only interesting for the daily business during this timespan. In a data analysis situation, however, the need to discover trends in the way business is doing and compare them with those of previous periods sparks a need to keep data for longer periods of time. Most DW's keep data for at least a couple of years, and many intend to keep it much longer.

- *Time-variant:* Operational data does not always have an explicit temporal dimension. It might not be interesting for an application, e.g., an inventory system, to know when a transaction actually took place. Also, operational systems often only store the current state of data. In the DW, time is a whole different matter. When analyzing data for trends, it is almost always important to know "the time of the data," so that all data in a DW can be related to a specific time point or interval. Also, not only the current value of data is stored, but often either snapshots of data at specific points in time, or a complete history of changes of the data.

- *Management's decisions:* Both words in this phrase are interesting in their own right. The word "decisions" indicates the very important fact that data in a DW is optimized for data analysis, not data entry. Thus normal database design principles do not necessarily apply, and managed redundancy of data is usually appropriate in a DW because it simplifies the database schema and improves analysis performance. The word "management's" indicates that DW data is traditionally used at the strategic level, by top management, for setting the course for the entire business. We would like to modify this to "management decisions," to capture the tendency that the DW is now also used at a "lower" level of the organization, by non-management employees, to get to know their part of the business better, thus providing better "micro" management in the daily work.

2.2.2 Previous Work

Like the database management area itself, the birth and rise of data warehousing has almost entirely taken place in the business world. Data warehousing was born out of

[1] In fact, the SAP company has done just this, and is now marketing a DW solution as an addition to their operational system.

the need of many businesses to view and analyze data from their many different operational systems together, to get a complete understanding of the business. Until recently, academia did not take interest in the area, and thus the field has been driven by the market, rather than by the research community. Research in distributed databases on issues such as global schemas and schema integration address some of the same challenges [7], but data warehousing still differs by employing data scrubbing, data cleaning, non-automatic data mappings, and bulkloading. Among other differences a DW stores more data than the sources and data is aggregated [130, 58, 64].

The focus of DW vendors as well as researchers has been on support for OLAP (OnLine Analytical Processing) functionality with good performance. In database research terms, the work has concentrated on the physical rather than the conceptual level. The data models employed have been of the multidimensional variety, where data is divided into *measurable business facts* and mostly textual *dimensions*, which characterize the facts and have hierarchies in them. In a retail business, *products* are sold to *customers* at certain *times* in certain *amounts* at certain *prices*. A typical fact would be a *purchase*, with the amount and price as the measures, and the customer purchasing the product, the product being purchased, and the time of purchase as the dimensions. A good visualization of the model, is to envisage data as living in an n-dimensional cube, with facts in the cells and the dimensions along the dimension axes [40].

OLAP systems have typically been implemented using two technologies: ROLAP (Relational OLAP), where data is stored in an RDBMS, and MOLAP (Multidimensional OLAP), where a dedicated multidimensional DMBS (MDDMBS) is used. Reports indicate that traditional database design techniques, i.e., ER modeling [19] and normalized tables, are not well suited for DW applications; as a result, new techniques, e.g., *star schemas* [64], have emerged that better support the DW purpose of data analysis. As mentioned above, most work has concentrated on performance issues; and higher-level issues, such as conceptual modeling, have largely been ignored so far, at least in academia.

Recently, several researchers have pointed to this lack in DW research, and it has been suggested to try to combine the traditional DW virtues of performance with the more advanced data model concepts from the field of *scientific and statistical databases* [116]. This appears to be a very valuable direction, as users of a DW tend to work directly with the data, creating a need to put more semantics directly into the database schema, as opposed to storing the data semantics in application programs, as is the case in operational systems.

2.2.3 A Case Study

The case study illustrates the special demands of clinical data warehousing. The simplified case is taken from the domain of diabetes treatment [67, 82]. An ER diagram of the case using standard notation [34] is seen in Figure 2.1.

The most important entity type is the *patient*, as indicated by the placement in the middle of the diagram. A patient is identified by a Social Security Number (SSN) and has additional attributes Name, Birth Date, and Height, all of which we will consider to be static. A patient has many relationships to other entities, whose main

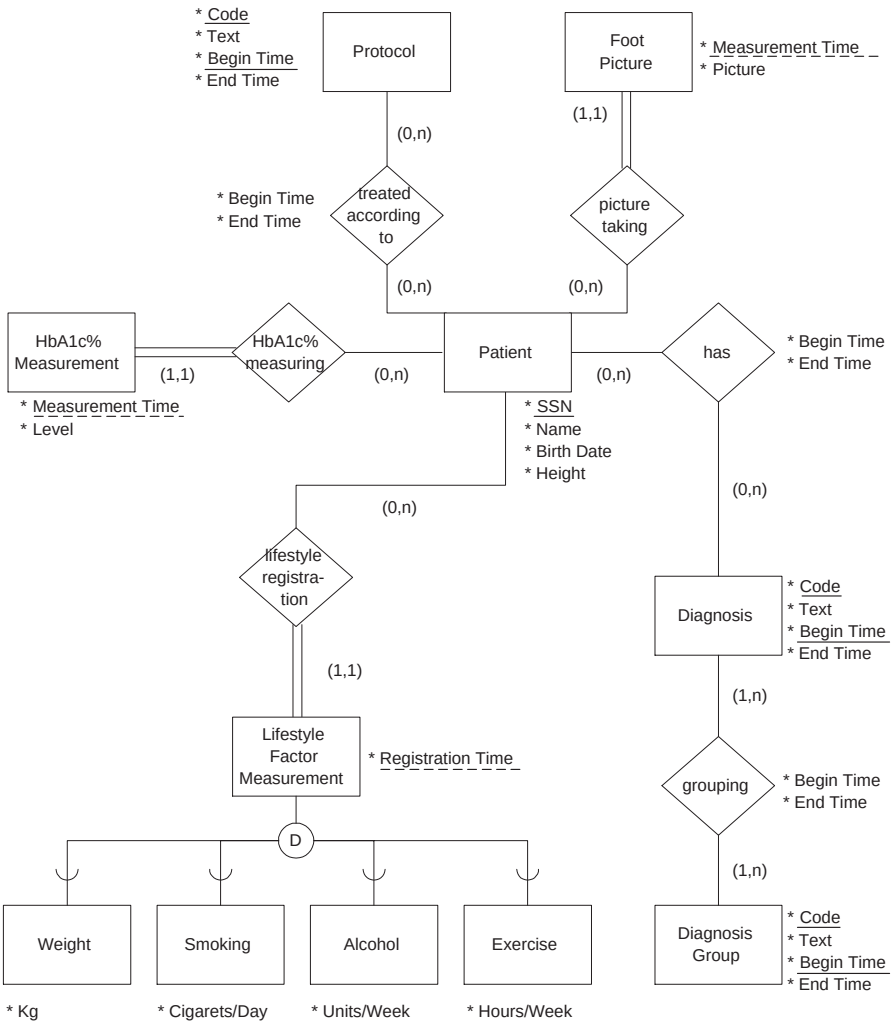

Figure 2.1: Case Study of a CDW for Diabetes

purposes are to *characterize* the patient. Thus, these other entities might be viewed as dimensions of the particular patient.

First, a patient can be given one or more *diagnoses*. These are only valid in specified time intervals, as the patient's condition changes over time. The set of possible diagnoses is given by a classification of diseases, e.g., the World Health Organisation's ICD-10 standard [133]. A diagnosis has an alphanumeric code, a descriptive text, and an associated period of validity. A specific diagnosis might be superseded by another as medical knowledge evolves, thus ending its validity, but for historical reasons it is important to keep it in the classification. Diagnoses are grouped into diagnosis groups, e.g., "Diabetes diseases" or "Pregnancy-related diseases," for overview purposes. One diagnosis can be a part of multiple groups, e.g., "Diabetes

during pregnancy[2]" can be a part of both of the just-mentioned groups. The participation in the diagnosis groups of diagnoses can change over time, as the demands for grouping vary. The groups also have an alphanumeric code and a descriptive text.

A patient is treated according to a *protocol*, which is a formal description of how a treatment should progress. Different protocols are used depending on the characteristics, e.g., the age, of the patient. We will not go into the very complex internal structure of a protocol, but will just record a code, a text, and a period of validity. The protocol used for treating a patient may vary over time.

An important indicator for the status of a diabetes patient is the condition of the *feet*, e.g., the blood circulation and presence of wounds. From time to time, the feet are photographed, and the pictures are stored along with the times they were taken.

One of the most important measurements for diabetes patients is *HbA1c%*, which indicates the long-time blood sugar level and provides a good overall indicator of the patient's status during the recent months. This measurement is taken approximately every three months.

For diabetes patients, a healthy lifestyle is even more important than normally, as it can literally make the difference between life and premature death. To monitor the lifestyle, several *lifestyle factors* are measured. These include weight and smoking, alcohol, and exercise habits. These factors are not measured on a regular basis, but rather considered to be valid from the time of registration until a new registration is being made. As the lifestyle factors have a lot in common, they are modeled using subtypes.

2.3 Clinical Data Warehousing Requirements

In this section we characterize the special requirements that surface in a CDW. We will start by introducing the concept of the Electronic Patient Record (EPR), explaining how it can serve as a solid foundation for a CDW.

2.3.1 The Electronic Patient Record

It is important to define exactly what is meant by an EPR. The Medical Records Institute (MRI), an independent, customer-owned non-profit organization, provides a six-page definition of electronic patient records [74], organized into five levels of computerization of patient information. We will use the definition from Level 3 "The Electronic Medical Record," as this is the lowest level where all patient information originating from one healthcare enterprise, e.g., all patient data kept by a single hospital, resides in the EPR in a structured format, i.e., as separate data items rather than simply as scanned documents.

To paraphrase the standard, an Electronic Medical Record (EMR) shall be able to uniquely identify the person that the information concerns, e.g., through the use of an enterprise-wide patient index. It is the *complete* collection of information; thus data

[2]The reason for having a separate pregnancy related diagnosis is that the diabetes must be monitored and controlled much more intensely during a pregnancy to assure good health of both mother and child.

from other clinical systems should be integrated in the EMR and harmonized accordingly. The EMR shall be used directly by all healthcare staff to record information. It shall have legal validity as any other document. This places severe demands on the security system for access control, electronic signatures, auditing, and data integrity, i.e., data can only be corrected by amendments.

The EPR is thus the central component in the IT-infrastructure of a modern healthcare enterprise. It is the common tool used by all healthcare professionals working in the enterprise. It is the point of entry for most patient information, and it provides access to the data born in other systems, e.g., laboratory or financial systems. In spite of these characteristics, the EPR cannot be considered a data warehouse in itself. Data in the EPR is used and organized according to operational purposes, where many kinds of data about one patient is presented to get an overview of the health status of the patient. Thus, data in the EPR is used and organized in a *by-patient* fashion.

In a DW, specific aspects of properties for a large population of patients are analyzed for trends, thus data is used and organized in a *by-property* fashion. The EPR is more akin to what Inmon defines as an *operational data store* (ODS) [57]: the integrated data store used as the basis for building the DW. The most important obstacle in using the EPR as a basis for a CDW is the multitude of different EPR systems on the market; the task of integrating data from several EPR systems is a hard one. This creates the need for a common standard for EPR data.

2.3.2 The EHCRA Standard for EPRs

The EHCRA standard [36] is the result of a European EPR standardization effort. It describes how to structure an EPR and lists demands that an EPR should meet. The ideas in the EHCRA standard originated from the Norwegian NORA project [118], which has led to the development of the DocuLive EPR system by Siemens Nixdorf Norway. DocuLive EPR is a tool for implementing EPR's based on the EHCRA standard.

EHCRA defines the EPR by means of a *document metaphor*: The EPR for a patient should be thought of as consisting of a number of documents containing information about the patient. The documents are structured, i.e., are not in free-form text. The structure of a document is *hierarchical*, with a document made up of *record items* or *record item complexes*, see Figure 2.2.

The lowest level of a document is the record item, which can be thought of as a basic element of information, e.g., the patient's name. Record item complexes can be made up of record items or record item complexes, leading to a tree-structured document. A typical record item complex would consist of lifestyle information such as weight, smoking, alcohol, and exercise, grouped together. A record item can also point to the record item or record item complex in another document, where this piece of information was originally entered, thus providing multiple views of the data. This gives the EPR the structure of a *directed, acyclic graph*. All record items have some common properties such as a reference to the patient and a reference to the original context of the data (in case of a pointer). The original context is important when information is exchanged, as new users of the information can then

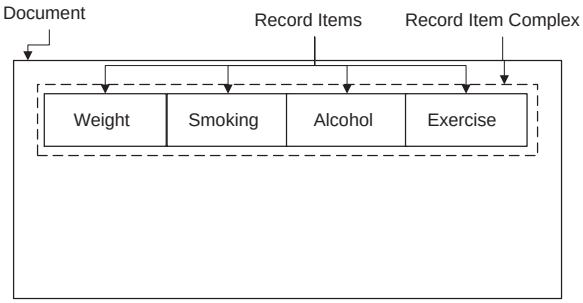

Figure 2.2: An EHCRA Document

get a complete view of the patient's situation when, e.g., a diagnosis, was chosen. The legal requirements are met by recording the time the data is included in the EPR, the status of the data, i.e., valid or invalid, and the unique id of the healthcare person entering the data. An update of the patient's weight is thus made by marking the old weight as invalid, and inserting a new valid weight, thereby keeping the full history of changes.

2.3.3 New Challenges

As mentioned above, clinical data warehousing introduces several new challenges to DW technology, compared to conventional data warehousing. We will illustrate these by referring to the example from the previous section.

Utilizing EPR Features

One very important issue is to utilize the features of the EPR optimally for building a clinical data warehouse. EPR systems in general, and EHCRA-based systems, in particular, have features that make them a very good foundation for clinical data warehouses.

The EHCRA standard is the European standard for the structure of the EPR. All EPR systems must thus be able to at least deliver data in an EHCRA-compliant format, even if they do not structure data internally according to EHCRA. Thus, utilizing the features of EHCRA in the integration of EPR data with the CDW allows for a very attractive and open solution that will work with many different EPR systems.

All versions of data in the EPR are stored along with their times of update. This gives full *transaction-time* support in EHCRA-based systems, thereby making it much easier to provide this support in the clinical DW. Often *valid time* is also attributed to the data in the EPR, providing for full *bitemporal* support [63].

The EPR is supposed to be the only (or at least the primary) tool that the clinical user is using in the daily work, so there is a great need to have access to all data, also lab results, etc., through the EPR. Thus *integration of operational data is already achieved* in the EPR, making the integration process in the DW very easy in comparison to conventional data warehousing. At a higher level in the MRI standard mentioned in Section 3.1, data from several healthcare enterprises, potentially the

whole world, is integrated in the EPR. This makes the integration in the DW of data from different locations much easier.

The EU-sponsored project Synapses [24] concerns the building of a *federated healthcare record server* that integrates a wide variety of EPR systems. This should provide access to clinical data in an EHCRA-compliant format, no matter how the actual EPR systems structure the data internally. Initially, the goal of the Synapses project is to facilitate the exchange of electronic patient records between different healthcare units, possibly using different EPR systems. However, another exciting application would be to use the Synapses server to transfer data to the CDW in a uniform way, no matter what the underlying EPR systems are, thus making the task of integrating data from different EPR systems very easy. A lot of effort could be saved, compared to accessing the proprietary EPR systems directly.

In normal operational systems, data is entered "post mortem" either automatically or by a clerk. The data is almost never used again by the person registering or entering the data, thus giving little incentive for carefully registering all the data, the correct data, and nothing but the correct data. This means that extensive data cleaning procedures must be established when the data is to be transferred to the DW [58]. In the EPR, the physician entering the data is also the primary user of the data, so entering dirty data will directly translate into problems in the daily treatment of the patients. Thus, data is quality assured continuously by the primary users. This means that the operational data is more likely to be of *high quality*, thus requiring less cleaning when being moved to the CDW. This will give the results obtained from the CDW a high level of credibility.

There are many unresolved issues in how to optimally exploit EHCRA-compliant systems as the basis for CDW's.

Complex Data Modeling Features

One of the most prominent demands is a data model for the CDW that includes more complex modeling constructs than typical multidimensional models, while not losing their obvious strengths in the area of decision support, i.e., we should not return to the full generality of the ER model.

In the multidimensional model [40], facts are in a n-1 relationship to the base elements of the dimensions, which in turn encode strict hierarchies, i.e., lower levels have n-1 relationships to upper levels. An example of this is a purchase. Exactly one product can be purchased, the product can belong to exactly one product category, etc.

But consider the case study where a patient has multiple diagnoses at the same time. The relation between patient and diagnosis is most naturally modeled as an n-n relationship, as the *same* patient may have multiple diagnoses. For instance, if we ask the question "How many patients have diagnoses A or B" we only want patients with both to be counted once. We should be able to capture this intended behavior in the schema. This is not easily possible using a conventional multidimensional model.

In multidimensional modeling [64], we have three alternatives for encoding n-n relationships: traditional dimensions, mini-dimensions, and snowflaking. Using traditional dimensions, we would enumerate all the possible combinations of diag-

noses. Having 10.000 diagnoses, this would amount to $2^{10.000}$ dimension records, making this solution practically unusable. Enumerating only the combinations actually used would still yield a very large number of dimension records. Furthermore, the dimension tables would be very wide and incomprehensible to the users. Using mini-dimensions with one dimension for each possible diagnosis would yield 10.000 dimensions, making the solution bad-performing as well as incomprehensible. Using snowflaking will not give any advantages over traditional dimensions, as the basic elements of the dimensions would the same, i.e., the possible combinations of diagnoses.

Another characteristic of clinical data is that we have many "loosely coupled" facts, e.g., the weight and smoking measurements from the case study. The values of these two measurements can change independently of each other, and a value is not always present at a given point in time, i.e., if the patient has not reported smoking habits. The measurements can be viewed in two ways, as time-variant attributes of the patient, or as separate entities that can be manipulated independently. The data model should be able to handle both treating the facts together, as if they belonged to the same entity, e.g., patient, or treating them separately.

In addition the data model should provide integrated semantic support for the demands listed in the following sections, e.g., temporal support, so that the solutions do not appear as poorly integrated "add-ons."

Advanced Temporal Support

One very important property of clinical data is the importance of temporal aspects. The same test, e.g., the HbA1c% measurement, can be made hundreds of times, so it is important to know both when the data is considered to be *valid in the real world*, and when it is *stored and changed in the database*. These temporal aspects of the data, known as *valid time* and *transaction time*, must both be supported to provide *bitemporal* support [62]. This support is for instance needed in order to "couple" different facts, e.g., smoking and weight, thereby computing "snapshots" of measurement values at specific intervals or points in time[3]. These snapshots are used to observe temporal trends in the evolution of one type of data values or in the relation between different types. In order to conduct these and other types of time studies, it is necessary to have available a strong support for time-series data, including a rich set of temporal analysis tools. It should be possible to use the above-mentioned advanced temporal concepts wherever meaningful. These advanced temporal concepts are not supported by current models.

Advanced Classification Structures

A data type of extreme importance in the clinical sector is "coded," or classified, data. One example is a diagnosis, which at the lowest level is a very precise indication of one specific medical condition. Diagnoses are then grouped repeatedly into larger, more general classes. A diagnosis is a good example of a typical OLAP "dimension,"

[3]This is possible because although the same property may be measured many times for the same patient, only one measurement value is considered to be "valid" at any given point in time.

as it characterizes the condition of the patient, it is attached to; but unlike the typical dimension, the diagnosis hierarchy is non-strict. Take for instance the diagnosis "Diabetes during pregnancy." This is in the group "Other pregnancy related diseases," but also in the group "Diabetes." This leads to an interesting requirement. If we ask for the number of patients, grouped by diagnosis at the lowest level, we naturally only want each pregnant-diabetes patient to be counted once. Then, if we "roll up" to the next level of diagnoses, we want the patients to be counted both in the pregnancy-related and the diabetes diseases groups. If we then roll up again, not considering the diagnosis dimension at all, we should again only count the patients once. Clearly the user of the DW should be able to work with the data and get the correct results, without having to worry about double-counting, etc. In the case of strict hierarchies, this feature is referred to as *summarizability* [103, 70].

Current data models do not specifically address this issue of correct aggregation in the case of non-strict hierarchies.

Another requirement related to classification structures is that they should be able to handle temporal change. Classifications change and new diagnoses and new groups come and go at a steady rate. The CDW should support this in an intelligent way, so that analysis of data across changes is handled smoothly and preferably transparently to the user. This requirement is also not handled well by current techniques.

Continuously Valued Data

Measurements and lab results, e.g., the HbA1c% measurement from the case study, are the key facts in the CDW. Unlike typical DW facts, these types of data clearly do not yield any meaning when summed. Other standard aggregation operators, such as MIN, MAX and AVG do apply, but the real demands are for more complex operations, such as standard deviation and other statistical functions. These operators are mainly used during follow-up on treatment in relation to clinical protocols, see below, or in medical research. The CDW should be able to support these advanced operations very efficiently, to supply the performance necessary to analyze large amounts of data accumulated over long periods of time. To do so, it must be investigated how pre-stored and pre-aggregated data can be used to achieve high performance. Current techniques for maintaining pre-aggregated data support only simple aggregation operators such as SUM.

Dimensionally Reduced Data

In a clinical DW, average patients might have hundreds of different facts describing their current situation, in diabetes treatment about 200 facts are recorded. There is an urgent need to be able to aggregate this massive amount of information in a useful way. In the case study, a patient has four independent indicators describing the lifestyle, i.e., levels of smoking, exercise, alcohol, and weight. These could be combined into one aggregate measure indicating the overall lifestyle of the patient. In multidimensional terms, we have reduced the previous four dimensions to just one. In a traditional OLAP world, the only way to reduce dimensionality is by projection, thereby *ignoring* all information about the omitted dimensions. The dimension re-

duction approach clearly has advantages over this, as the complexity of the data is reduced, while the essence is maintained. The clinical DW should be able to support the definition of such combination functions, and it should provide good performance for reducing/increasing the number of dimensions. The issue of pre-aggregation in connection with dimensionality reduction is also very interesting.

Integration of Very Complex Data

The clinical world is also characterized by very complex types of data. One example is the 2048 by 2048 pixel foot picture from the case study, which in multidimensional terms could be viewed as being 4.194.304 dimensional, by considering all pixels to be independent dimensions. While this clearly is an overly complex way of looking at it, it should be possible to incorporate this type of data in the CDW for data analysis purposes. The functionality should be more advanced than just allowing the raw data to be stored and retrieved, i.e., the support often associated with "blobs." Rather, it should be possible to define *feature extractors* on the raw data, e.g., pattern recognition functions for wounds, and to perform analyses on the extracted features. The extractors should be tightly integrated with the DW, allowing for addition of new and modification of existing extractors incrementally, i.e., without having to recompute every feature from scratch. Existing data warehouse techniques accommodate only with simple, structured data such as text and numbers.

Support for Clinical Protocols

The introduction of *managed care* is a very prominent current trend in the clinical world. Instead of relying solely on the judgment and knowledge of one doctor, the treatment of specific diseases is conducted according to well-defined protocols that specify the conditions and actions for using specific treatments. The protocol can be viewed as a "best practice" or an advanced set of business rules. A patient can be treated according to different protocols at different times, as shown in the case study.

There is a need to analyze the actual treatments, to investigate conformance to the protocols, outcomes, etc. As an example, we could ask what protocol provides the best treatment in terms of keeping the HbA1c% close to normal.

Ideally, the protocols would be specified formally, to allow "automatic" follow-up on treatments. That is, queries against the CDW could be generated directly from the protocols, and the results of these be used to test conformance, to adjust the protocols, etc. The CDW should have integrated support for clinical protocols, to accommodate this important part of clinical practice. Today, data warehouse systems do not have any support for advanced business rules like these.

Support for Medical Research

Medical research can take several different forms; one form that the clinical DW enables is the so-called *qualitative* research where large amounts of data is analyzed to confirm known or discover unknown trends and correlations in the data. For example, a correlation between the weight and the HbA1c% of a patient could be discovered. The discovery process in medical research would benefit enormously from having

data mining facilities integrated into the CDW. There should be a conceptually simple, fast-performing, and yet flexible way to produce the "flat" sets of data that are normally fed into data mining algorithms. The results could be used as inspiration for hypotheses that could then be tested in controlled, formal clinical studies. The integration of data mining and data warehousing and the use of a DW for research purposes are both in their infancy in today's DW products.

2.3.4 Comparison of Conventional and Clinical DW

The differences between clinical and conventional data warehouses extend into their corresponding operational systems. For conventional systems, the operational systems often consist of a wide range of poorly integrated legacy systems. In a modern clinical setup, however, almost all data is already accessible in an EPR, thus providing integration at the operational level. The EPR also has other characteristics that differ from most typical operational systems (see Table 2.1). Both types of systems have small granularity of data, but in the EPR, data is never deleted, and a full trace of all updates are maintained for legal reasons.

	Conventional	*Clinical (EPR)*
Integration	No	Yes
Granularity	Small	Small
Volatility	High	Zero
History	No	Yes

Table 2.1: Comparing Conventional and Clinical Operational Systems

If we consider the same characteristics for conventional versus clinical data warehouses, we also see some interesting trends (see Table 2.2).

	Conventional	*Clinical*
Integration	Yes (hard)	Yes (easy)
Granularity	Medium	Small (drill back)
Volatility	Low	Zero
History	Sometimes	Always

Table 2.2: Comparing Conventional and Clinical Data Warehouses

Integration of data is achieved for both types, but in the typical conventional DW, integration is difficult to achieve because data is scattered in many legacy systems. In contrast, integration of data is already achieved in the EPR, thus making it easier to build the DW. Granularity varies from small to large in a conventional DW, but in a CDW, we always need to have the operational granularity of data. This is caused by the need to "drill back" to the EPR, e.g., when encountering an interesting anomaly in the data. The physician then needs access to the full patient record to determine the exact cause. In a conventional DW only 6-10 years of data is kept, but in the clinical world it is important to see the full disease history, which might span 50 years or

	Conventional	*Clinical*
Data Model	Simple	Complex
Temporal Support	Medium	Advanced
Classifications	Simple	Advanced
Continuously Valued Data	No	Yes
Dimensionally Reduced Data	No	Yes
Very Complex Data	No	Yes
Advanced Business Rules	Maybe	Yes (Protocols)
Data Mining	Maybe	Yes (Medical Research)

Table 2.3: Characteristics of Conventional versus Clinical Data Warehouses

more, e.g., in the case of diabetes patients. It is also very important to have the full update history of the EPR, to facilitate trend analysis. This level of temporal support is not always present in business data warehouses.

2.3.5 Standardization Efforts

An area of high importance to clinical information systems is the various standardization efforts in the field of healthcare informatics.

First, the Health Level 7 (HL7) organization's work on standardization of electronic data interchange in healthcare environments [30] specifies the content of electronic messages transmitting healthcare information, by referring to a common model that specifies domain concepts and legal data values. The HL7 standard is widely used in the industry for interfacing different systems.

Second, the Object Management Group (OMG) has launched the CORBAmed initiative [84] that is aimed at providing standard interfaces to healthcare information systems based on OMG's Common Request Broker Architecture (CORBA). Several CORBAmed workgroups focus on specific areas such as clinical decision support, clinical observations, patient identification, and HL7 integration.

Third, the most recent player is Microsoft, with the "ActiveX for Healthcare" initiative [77] that is a direct competitor to CORBAmed, but is based on Microsoft's Distributed Component Object Model (DCOM) instead of CORBA.

From a clinical data warehousing perspective, these initiatives deal almost entirely with integration of clinical data in the CDW and can thus be seen as enablers of integration, whether the CDW is based on an EPR or not. These initiatives do not address the other challenges presented in the chapter.

2.4 Summary

Table 2.3 summarizes the challenging needs covered in the chapter and compares them with the characteristics of a conventional DW, i.e., a DW as it is often used in a business context. This does not imply that business or other domains do not need the advanced features presented in the chapter. Rather, the comparison shows why

conventional DW techniques fail to meet the requirements of clinical data warehousing. The investigation of these requirements are the focus of this chapter. From the comparison it is clear that the needs of a clinical DW poses some very interesting challenges for researchers and developers alike.

The data model must support advanced constructs such as many-to-many relationships between facts and dimensions. Full support for bitemporal data and analysis over time is also needed. Advanced classification structures must be provided that integrate support for non-strict hierarchies with means of handling change and time, while maintaining support for correct aggregations.

Continuously valued data must be efficiently supported, including how to perform advanced operations on them. Dimensional reduction of data in the CDW is important in order to make sense of the high-dimensional data. Very complex data such as pictures or x-rays must be available in the CDW, with facilities for doing analysis on them.

The integration of clinical protocols in the CDW is important to allow for follow-up on the treatment of patients. Support for medical research, e.g., via data mining facilities, will enable the clinical community to perform their research much more efficiently than is possible today.

We have shown that DW technology faces exciting new challenges from the area of clinical data warehousing. Clinical data warehousing provides excellent opportunities for first-class DW research that will also have applications in areas beyond the clinical world.

The challenges that are especially important to the general database research community include the following: advanced data models including temporal support, advanced classification structures, continuously valued data support, and dimensional reduction of data.

We will work on these issues in clinical applications in order to support the successful application of data warehousing in the clinical world.

Chapter 3

A Foundation for Capturing and Querying Complex Multidimensional Data

3.1 Introduction

On-Line Analytical Processing (OLAP) [22] 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 [19] 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* [116]. 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 health care IT, Kommunedata, 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. This chapter presents eleven advanced requirements that a multidimensional data model should satisfy. The requirements are illustrated using a medical case study. Twelve previously proposed data models, which are representative for the spectrum of multidimensional data models, are evaluated against the eleven requirements. No existing model satisfies more than four of these requirements. This chapter presents an extended multidimensional data model that addresses all eleven requirements.

The medical case study presented in Section 3.2 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, 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 3.2, 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 to 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 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 chapter is structured as follows. Section 3.2 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 3.3 defines the basic extended multidimensional data model, using examples from the case study for illustration, then adds support for handling time to the model. With the data structures of the model available, Section 3.4 defines the algebraic query language of the model and discusses its properties. The model's algebraic query language is closed and at least as strong as relational algebra with aggregation functions. Section 3.5 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 3.6 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 3.7 evaluates the model against the requirements. Section 3.8 discusses the use of pre-aggregated data for query evaluation involving imprecision. Section 3.9 summarizes the chapter and points to future directions. Appendices 3.10 and 3.11 describe how to use relational databases technology to implement the new model and to handle imprecise data during querying, respectively.

3.2 Motivation and Related Work

Section 3.2.1 presents a medical case study. The study illustrates the chapter's contributions and also motivates the requirements for multidimensional models, which are presented in Section 3.2.2. Section 3.2.3 relates these requirements to existing multidimensional data models. Finally Section 3.2.4 considers related work in imprecise data management.

3.2.1 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 3.1.

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 [59]. 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 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) [133]. 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

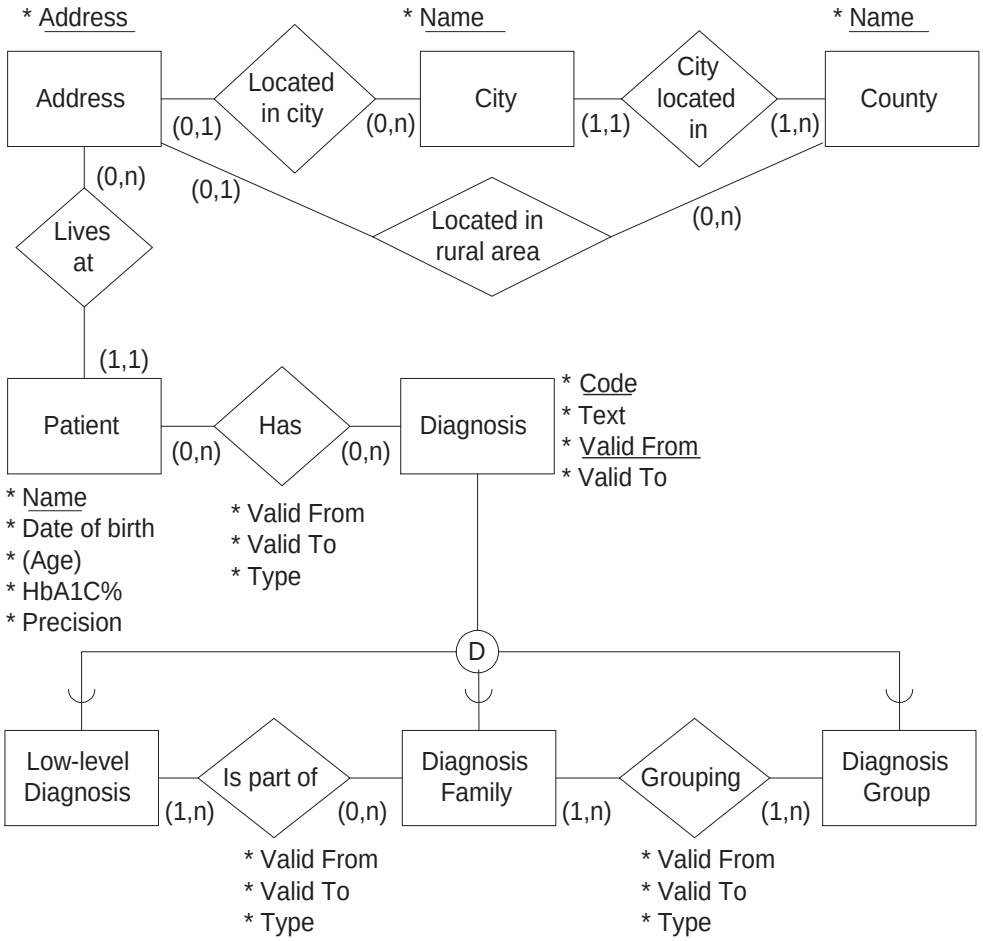

Figure 3.1: Patient Diagnosis Case Study

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 chapter, 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 chapter, 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 chapter 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 [20]. 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

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

Patient Table

ID	Address
50	21 Central Str.
51	34 Main Str.
52	123 Rural Rd.
53	1 Sandy Ds.

Address

ID	Name	CountyID
20	Sydney	30
21	Melbourne	31

City

AddressID	CityID
50	20
51	21

LocatedInCity

AddressID	Name
52	31
53	32

LocatedInRuralArea

ID	Name
30	Sydney
31	Melbourne
32	Outback

County

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	Scndr.
2	8	01/01/70	31/12/81	Primary
2	5	01/01/82	30/09/82	Scndr.
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 pregn.	01/01/80	NOW
6	O24.1	Non insulin dpdnt diabetes during pregn.	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 3.1: Data for the Case Study

was introduced. The resulting tables are shown in Table 3.1 and will be used in examples throughout the chapter.

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 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 chapter 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.

3.2.2 Requirements for Data Analysis

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

1. *Explicit hierarchies in dimensions*. The hierarchies in the dimensions should be captured explicitly by the schema. 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. *Symmetric treatment of dimensions and measures*. The data model should allow measures to be treated as dimensions and vice versa. In our example, the attribute Age for patients would typically be treated as a measure, to allow for computations such as average age, etc., but we should also be able to define an Age dimension which allows us to group the patients into age groups.

3. *Multiple hierarchies in each dimension*. 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.

4. *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, but performing average calculations on them does make sense. In the field of statistical databases, a closely related concept is *summarizability* [70, 103], which means that an aggregate result, e.g., total sales, can be computed by directly combining results from lower-level aggregations, e.g., the sales for each store.

5. *Non-strict hierarchies*. The hierarchies in a dimension are not always strict, i.e., we can have many-to-many relationships between the different levels in a dimension. In our example, the diagnosis hierarchy is not strict. The data model should be able to handle these just as well as "ordinary" strict dimensions.

6. *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.

7. *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.

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* [64, 9] is part of this problem.

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 3.2.1.

3.2.3 Existing Multidimensional Models

We proceed to evaluate twelve data models for data warehousing on the requirements just presented. We consider the models of Rafanelli & Shoshani [103], Agrawal et al. [2], Gray et al. [40], Dyreson [31], Kimball [64], Li & Wang [71], Gyssens & Lakshmanan [48], Cabbibo and Torlone [12], Datta & Thomas [26], Lehner [69], Vassiliadis [127], and Microsoft's OLE DB for OLAP standard [78]. 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 [40, 64, 48, 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 "Area," "County," and "Region" would be used to characterize the patients. The hierarchy between the attributes is not captured explicitly by the schema of the simple cubes, so the user will not be able to learn from the schema that Area rolls up to County and not the other way around. Simple cubes include Star schema designs [64].

The structured cube models [2, 71, 31, 12, 69, 127, 78] 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 [68]. The hierarchies are captured using either *grouping relations* [71], *dimension merging functions* [2], measure graphs [31], roll-up functions [12], level lattices [127], or an explicit tree-structured hierarchy as part of the cube [69, 78].

The last group of models is the *statistical object models* [103]. 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 twelve data models against our eleven requirements are seen in Table 3.2. If a model supports all aspects of a requirement, we say that the model provides *full* support, denoted by "$\surd$". 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 [103]	√	-	-	√	p	p	-	-	-	-	-
Agrawal et al. [2]	p	√	√	-	p	-	-	-	-	-	-
Dyreson [31]	√	-	√	p	-	-	-	-	-	p	p
Gray et al. [40]	-	√	√	p	-	-	-	-	-	-	-
Kimball [64]	-	-	√	p	-	-	-	-	p	-	-
Li & Wang [71]	p	-	√	p	-	-	-	-	-	-	-
Gyssens & Lakshmanan [48]	-	√	√	p	-	-	-	-	-	-	-
Cabbibo & Torlone [12]	√	-	√	p	-	-	-	-	-	-	-
Datta & Thomas [26]	-	√	√	-	p	-	-	-	-	-	-
Lehner [69]	√	-	-	√	-	-	-	-	-	-	-
Vassiliadis [127]	√	-	√	√	-	-	-	-	-	-	-
MS OLE DB for OLAP [78]	√	-	√	p	-	-	-	-	-	-	-

Table 3.2: Evaluation of the Data Models

1. *Explicit hierarchies in dimensions*: The simple cube models [40, 64, 48, 26] do not capture the hierarchies in the dimensions explicitly. Some models provide partial support by the *grouping relation* [71] and *dimension merging function* [2], but do not capture the complete hierarchy together with the cube. This is done by the remaining models [103, 31, 12, 69, 127, 78], thus capturing the full cube navigation semantics in the schema.

2. *Symmetric treatment of dimensions and measures*: Most of the models [64, 103, 31, 71, 12, 69, 127, 78] 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 [40, 2, 48, 26] do not impose this restriction. They either do not distinguish between measures and dimensions [40, 48], or they allow for the conversion of measures to dimensions and vice versa [2, 26].

3. *Multiple hierarchies in each dimension*: Some models [103, 69] 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 [40, 64, 2, 48, 71, 31, 12, 26, 127, 78] allow multiple hierarchies.

4. *Support for aggregation semantics*: Most of the models [40, 31, 64, 48, 12, 71, 78] 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 [70] 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, 26]. Two models [103, 69] 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 [127] 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.

5. *Non-strict hierarchies*: Most of the models [40, 31, 64, 48, 71, 12, 69, 127, 78] implicitly or explicitly require that hierarchies be strict. Two models [2, 26] mention briefly that non-strict hierarchies are allowed, but does not go deeper into the issues raised by allowing this, e.g., the possibility of double-counting and the use of pre-computed aggregates. The remaining model [103] investigates the possible problems with allowing non-strict hierarchies and advises against using this feature.

6. *Non-onto hierarchies*: Only one model [103] discusses the possibility of having non-onto hierarchies, but advises against using this feature. All the other models do not allow non-onto hierarchies.

7. *Non-covering hierarchies*: None of the models allow non-covering hierarchies.

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*: Only one model [64] discusses this issue, but none the proposed solutions fully support analysis across changes in the dimensions. None of the other models support analysis across changes, although one mentions that this is a very important issue [69].

10. *Handling different levels of granularity*: Dyreson [31] 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. None of the other models handle different levels of granularity in the data.

11. *Handling imprecision*: For the reasons mentioned above, the concept of incomplete data cube [31] provides partial support for imprecision in the data, as this can be handled using varying granularities. 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 1–4. Requirement 5 (non-strict hierarchies) is partially supported by three of the models, while requirement 6 (non-onto) is partially supported by only one model. Requirement 9 (handling change and time) is only partially supported by

Kimball [64]. Requirements 10 and 11 (handling different levels of granularity and imprecision) are only partially supported by Dyreson [31]. Requirements 7 and 8 are not supported by any of the models. The objective of the model presented next is to support all eleven requirements.

3.2.4 Related Work on Imprecision

The area of "imperfect information" has attracted much attention in the scientific literature [80]. We have previously compiled a bibliography on uncertainty management [32] 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 chapter. Aggregation of imprecise data has been examined in the context of both possibilistic (fuzzy) databases [108] and (to a lesser extent in) probabilistic databases [37], 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 [132], and in this chapter, we similarly use tools from statistics to handle imprecise aggregate data.

The approach presented in this chapter 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. The 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 chapter'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 [21], which allow unknown data to be captured explicitly.

Multiple imputation [106, 11] 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 [14]. 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 [136] 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 [107, 108] 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* [8] has been used extensively in temporal databases for a variety of purposes, including the handling of imprecision in the data [33]. However, aggregation of imprecise temporal data remains to be studied. In the area of multidimensional databases, only the work on *incomplete data cubes* [31] has addressed the issue of handling imprecise information. Compared to this chapter'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.

3.3 An Extended Multidimensional Data Model

In this section, our data model 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 new multidimensional model, we define the *intension*, the *extension*, and give an illustrating example.

3.3.1 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 3.2.1 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: $\perp_{Diagnosis}$ = *Low-level Diagnosis* $\sqsubset$ *Diagnosis Familiy* $\sqsubset$ *Diagnosis Group* $\sqsubset \top_{Diagnosis}$. We have that $Pred$(*Low-level Diagnosis*) = {*Diagnosis Family*}. Other examples of category types are *Age* and *Ten-year Age Group* from the Age dimension type, and *DOB* and *Year* from the DOB dimension type. Figure 3.2, 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 [69, 102], we distinguish between three types of aggregate functions: Σ, applicable to data that can be added together, ϕ, applicable to data that can be used for average calculations, and c, applicable to data that is constant, i.e., it can only be counted. Considering only the standard SQL aggregation functions, we have that Σ = {SUM, COUNT, AVG, MIN, MAX}, ϕ = {COUNT, AVG, MIN, MAX}, and c = {COUNT}. The aggregation types are ordered, $c \subset \phi \subset \Sigma$, so data with a higher aggregation type, e.g., Σ, also possess the characteristics of the lower aggregation types. For each dimension type $\mathcal{T} = (\mathcal{C}, \sqsubseteq_{\mathcal{T}})$, 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, $Aggtype$(*Low-level Diagnosis*) = c, $Aggtype(Age) = \Sigma$, $Aggtype$(*Ten-year Age Group*) = c, and $Aggtype(DOB) = \phi$.

A *dimension* D of type $\mathcal{T} = (\{\mathcal{C}_j\}, \sqsubseteq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$ 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 $\perp_{\mathcal{T}}$ 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. [40].

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 $\top_{Diagnosis} = \{\top\}$. The values in the sets refer to the *ID* field in the Diagnosis table of Table 3.1. The partial order $\sqsubseteq$ is given by the first two columns in the Grouping table in Table 3.1. 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 "Own Birth Date" and a "First Child Birth Date" dimension, both of the dimension type "Date of Birth" discussed above. We want to take the union of those two dimensions (and the categories in them) to produce a combined "Birth Date" dimension with both types of birth dates in it. This should be allowed since the dimensions and the categories have the same type. However, if we have a "Shipment Date" dimension with the same structure (days, weeks, months, quarters, years) we should not be allowed to take the union of that and the "Birth Date" dimension since they are not of the same type, meaning that the data in them does not represent the same thing and thus should not be mixed. Another example would be to subtract Own Birth Date years from First Child Birth Date years, e.g., when performing an aggregation computation on the data, to get the age of people at the time their first child is born. This should be allowed since the two categories have the same type. On the contrary, it probably does not make sense to subtract Own Birth Date years from Shipment Date years. This can be prevented by using different category types for the "Year" category in the two dimensions. The union and aggregation operators are discussed in detail in Section 3.4

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 $\top_{Diagnosis}$.

It is desirable to distinguish between the dimension values in themselves and the real-world "names" that we use for them. The names might change or the same value might have more than one name, making the name a bad choice for identifying an value. In common database terms, this is the argument for *object ids* or *surrogates*.

To support this feature, we require that a category C has one or more *representations*. A representation Rep is a bijective function $Rep : Dom(C) \leftrightarrow Dom_{Rep}$, i.e., a value of a representation uniquely identifies a single value of a category and vice versa, thus making the representation an "alternate key." We use the notation $Rep(e) = v$ to denote the mapping from dimension values to representation values.

Example 7 A diagnosis value has two representations, *Code* and *Text*. Using the ID's from the Diagnosis table to identify the values, we have $Code(3) =$ "O24$''$ 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 $\exists e_1 \in D\ ((f, e_1) \in R \wedge e_1 \sqsubseteq e)$. 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 $\bot_{Diagnosis}$, as do the existing data models. If no diagnosis is known for patient 1, we would have added the pair $(1, \top)$ to R.

A *multidimensional object* (MO) is a four-tuple $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (\mathcal{F}, \mathcal{D} = \{\mathcal{T}_i\})$ is the 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. 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 $\perp$ 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 $\top_{HbA1c\%}$. 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 3.2.

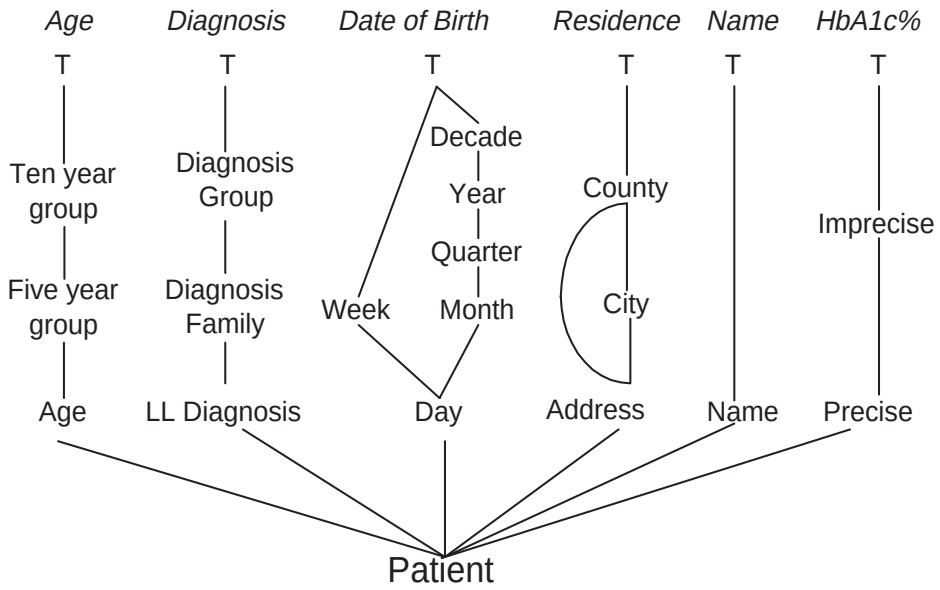

Figure 3.2: Schema of the Case Study

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 $\top_{Diagnosis}$, and a subdimension of the Age dimension that includes only *Ten-Year Group* and $\top_{Age}$. Make an MO with these two dimensions and the fact type Patient for all patients in the country. This results in an MO capturing all patients in the country together with their diagnosis groups and their ten-year age groups. Putting this MO together with the "Patient" MO from the example above, we obtain a multidimensional object family with two shared subdimensions. The shared subdimensions

could be used to investigate how diagnoses versus age groups for the patient group from the case study compare to the national average.

To 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.

3.3.2 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 [35] explicitly assume a discrete model of time). Also the temporal data types offered by the SQL standard [75] 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 [8]. 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 [63]. 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" [63]. Thus, we do not have the problem of "value-equivalent" statements [63, 119, 61], 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 that $7 \sqsubseteq_{[01/01/70-31/12/79]} 3$. For the representation, we have the relationship $Code(8)_{[01/01/70-31/12/79]} = \text{D1}$. For the fact-dimension relation, we have the relationship $(2, 3) \in_{[23/03/75-24/12/75]} R$. 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 [63]. 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.

3.3.3 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 [70]. 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, and 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 $\forall f \in F : f \rightsquigarrow e_1 \wedge f \rightsquigarrow e_2 \wedge e_1 \in C_j \wedge e_2 \in C_j \Rightarrow e_1 = e_2$[1]. The hierarchy in dimension D is *snapshot strict*, if at any given time t, the hierarchy is 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.

3.4 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.

3.4.1 The Basic Algebra

In this section the fundamental 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, thus we do not specify the values representations for the resulting MO's. The aggregation types are only changed by the aggregate formation operator, so they are not specified for the other operators. Two helper definitions are important in several of the operators.

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

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

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

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 $\mathcal{S}' = \mathcal{S}$, $F' = \{f \in F \mid \exists e_1 \in D_1, .., e_n \in D_n\ (\ p(e_1, .., e_n) \wedge f \rightsquigarrow_1 e_1 \wedge .. \wedge f \rightsquigarrow_n e_n)\}$, $D' = D$, $R' = \{R'_i\}$, and $R'_i = \{(f', e) \in R_i \mid f' \in F'\}$. Thus, we restrict the set of facts to those that are characterized by values where p evaluates to true. The fact-dimension relations are restricted accordingly, while the dimensions and the schema stay the same.

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 left in Figure 3.3.

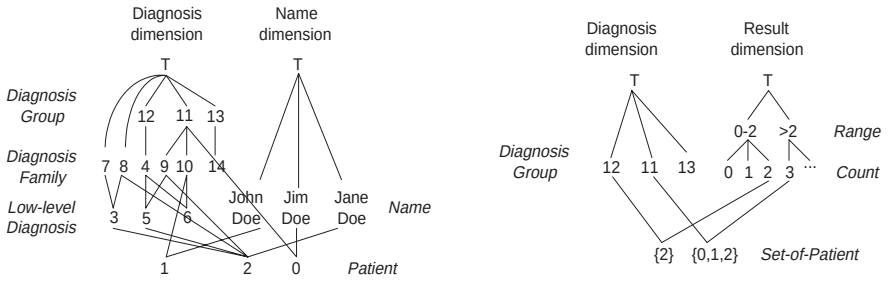

Figure 3.3: Resulting MO's for Projection and Aggregate Formation

rename: Given a multidimensional object, $M = (\mathcal{S}, F, D, R)$, and fact schema $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, such that $\mathcal{D}$ is isomorphic with $\mathcal{D}'$, we define the rename ρ as: $\rho[\mathcal{S}'](M) = M'$, where $M' = (\mathcal{S}', F, D, R)$. We see that rename just return the

contents of M with the new schema $\mathcal{S}'$, which has the same structure as the old schema $\mathcal{S}$. Rename is used to alter the names of dimensions so that dimensions with the same name, e.g., resulting from a "self-join," can be distinguished.

union: Given two n-dimensional MO's, $M_k = (\mathcal{S}_k, F_k, D_k, R_k), k = 1, 2$ such that $\mathcal{S}_1 = \mathcal{S}_2$, we define the union $\bigcup$ 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 $R'_i = \{(f', e) | f' = (f_1, f_2) \wedge f' \in F' \wedge ((i \sqsubseteq n_1 \wedge (f_1, e) \in R_{1_i}) \vee (i > n_1 \wedge (f_2, e) \in R_{2_{i-n_1}}))\}$. In words, the new fact type is the type of *pairs* of the old fact types, and the new set of dimension types is the union of the old sets. The set of facts is the subset of the cross product of the old sets of facts where the join predicate p holds. For p equal to $f_1 = f_2$, $f_1 \neq f_2$, and $true$, the operation is an *equi-join*, *non-equi-join*, and *Cartesian product*, respectively. For the instance, the set of dimensions is the set union of the old sets of dimensions, and the fact-dimension relations relates a pair to an value if one member of the pair was related to that value before.

Example 17 We want to know if any patients are registered with more than one name. We take two copies of the "Patient" MO and perform projection over the Name dimension for both. For the second copy, the Name dimension type is renamed to "*Name2*". We then perform an identity-based join of the two with the predicate $f_1 = f_2$. This gives us an MO with two dimension types, *Name* and *Name2*. The fact type is the type of pairs of patients; the set of facts is still $F = \{0, 1, 2\}$, 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 [66], we assume the existence of a *family* of aggregation functions g that take some k-dimensional subset $\{D_{i_1}, .., D_{i_k}\}$ of the n dimensions as arguments, e.g., SUM_i sums the i-th dimension and SUM_{ij} sums the i-th and j-th dimensions. We assume a function $Args(g) = \{j \mid g \text{ uses dimension } j \text{ as argument}\}$ that returns the argument dimensions of g.

Given an n-dimensional MO, M, a dimension D_{n+1} 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

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

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

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

$$D' = \{D_i', i = 1, .., n\} \cup \{D_{n+1}\}, D_i' = (C_i', \sqsubseteq_i'),$$

$$C_i' = \{C_{ij}' \in D_i \mid Type(C_{ij}') \in \mathcal{C}_i'\}, \sqsubseteq_i' = \sqsubseteq_{i|_{D_i'}}, R' = \{R_i', i = 1, .., n\} \cup \{R_{n+1}'\},$$

$$R_i' = \{(f', e_i') \mid \exists (e_1, .., e_n) \in C_1 \times .. \times C_n\ (f' = Group(e_1, .., e_n) \wedge f' \in F'$$
$$\wedge e_i = e_i')\}, \text{ and } R_{n+1}' = \cup_{(e_1,..,e_n) \in C_1 \times .. \times C_n} \{(Group(e_1, .., e_n),$$
$$g(Group(e_1, .., e_n))) \mid Group(e_1, .., e_n) \neq \emptyset\}$$

The aggregation types for the remaining parts of the argument dimensions are not changed. The aggregation types for the result dimension is given by the following rule. If g is distributive, the paths to $C_1, .., C_n$ are strict, and the hierarchies up to $C_1, .., C_n$ are 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" in the sense that they contain overlapping data, cannot be used for further aggregation. This prevents the user from getting incorrect results by accidentally "double-counting" data.

Example 18 We want to know the number of patients in each diagnosis group. To do so, we apply the aggregate-formation operator to the "Patient" MO with the *Diagnosis Group* category and the $\top$ categories from the other dimensions. The aggregate function g to be used is *set-count*, which counts the number of members in a set. The resulting MO has seven dimensions, but only the Diagnosis and Result dimensions are non-trivial, i.e., the remaining five dimensions contain only the $\top$ categories. The set of facts is still $F = \{0, 1, 2\}$. 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 on the right side of Figure 3.3. Note that each patient is only counted once for each diagnosis group, even though patient 2 has *several* diagnoses in each group.

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

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

drill-across from one MO to another is just the value-based join of the two MO's on their common dimensions.

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

SQL-like aggregation: Computation of an SQL aggregate function on an MO, grouped by a set of dimension categories, is done by first applying the aggregate formation operator to the MO with the given categories[2], and the given function. The dimensions not in the "GROUP BY" clause are then projected out.

star-join: A star-join as described in [64] is just a selection on the dimensions, usually combined with an aggregate formation with a given aggregate function on a set of category types.

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

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

Theorem 1 *The algebra is closed.*

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

Theorem 2 *The algebra is at least as powerful as the relational algebra with aggregation functions* [66].

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

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

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.

3.4.2 Handling Time in the Algebra

We will now turn our attention to how time can be handled in the algebra. Our requirements are to be able to view data as it appeared at a given point in time, in the database or in the real world, and to do analysis related to time, including analysis across times of change in the data. We note that the operators do not introduce any "value-equivalent tuples," thus the data stays coalesced. First, we consider valid-time MO's. To support the need to view data as they appeared at any given point in time in the real world, we introduce the *valid-timeslice operator* [63].

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; for a full treatment, see, e.g., the TSQL2 language [119].

The last step is to define how the basic algebra operations deals with the time attached to MO's. Neither the selection operator, the projection operator, or the rename operator change the time attached to the resulting MO's. For the union operator, time attachments for the resulting MO is computed according to the following rules[3]. $(f, e) \in_{T_1} R_{1_i} \wedge (f, e) \in_{T_2} R_{2_i} \Rightarrow (f, e) \in_{T_1 \cup T_2} R_i'$, $e_1 \sqsubseteq_{1_{T_1}} e_2 \wedge e_1 \sqsubseteq_{2_{T_2}} e_2 \Rightarrow e_1 \sqsubseteq'_{T_1 \cup T_2} e_2$, $Rep_1(e) =_{T_1} v \wedge Rep_2(e) =_{T_2} v \Rightarrow Rep'(e) =_{T_1 \cup T_2} v$, $e \in_{1_{T_1}} C_j \wedge e \in_{2_{T_2}} C_j \Rightarrow e \in'_{T_1 \cup T_2} C_j$. 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. $(f, e) \in_{T_1} R_{i_1} \wedge (f, e) \in_{T_2} R_{i_2} \wedge T_1 \setminus T_2 \neq \emptyset \Rightarrow (f, e) \in_{T_1 \setminus T_2} R_i'$, $F' = \bigcap_{i=1,..,n}\{f \mid \exists (f, e_i) \in$

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

$R'_i\ ((f, e_i) \in_{T'} R'_i \wedge T' \neq \emptyset)\}$. 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 $i > n_1$.

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.

3.5 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 granularities coarser than the "grouping" categories. The approach proposed here attempts to use the standard data model just defined to additionally capture imprecision. We assume that the chronons attached to the MOs are precise, which is reasonable as we can use, e.g., the load time in the DW, to obtain precise time. 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 chapter, as this has been covered extensively in previous research [33, 32].

3.5.1 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* 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 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 3.4.

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 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.

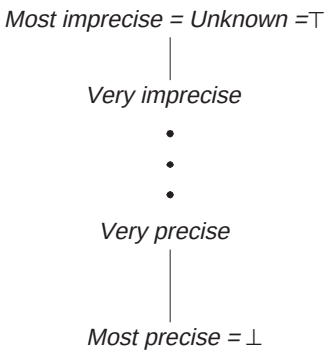

Figure 3.4: The Spectrum of Imprecision

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 diagnosis family, the alternative query would be the patient count grouped by diagnosis group.

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 [80].

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)        % use normal evaluation
else
    Q' = Alternative(Q,M)                   % suggest alternative query
    if Q' is accepted then Eval(Q',M)       % normal evaluation for alt. 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.

3.5.2 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, $Granularity_i$ 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 $\mathcal{S} = (\mathcal{F}, \mathcal{D})$, $\mathcal{D} = \{\mathcal{T}_i, i = 1, ..n\}$, $\mathcal{T}_i = (\mathcal{C}_i, \sqsubseteq_{\mathcal{T}_i})$, $\mathcal{C}_i = \{\mathcal{C}_{ij}\}$, $D = \{D_i, i = 1, .., n\}$, 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

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

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

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

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

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

Example 19 The MO from Example 9[4] has 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*, *Diagnosis Group*, $\top_{Diagnosis}$}. 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 $R_1 = \{(0, \textit{Diagnosis Group}), (1, \textit{Diagnosis Family}), (2, \textit{Diagnosis Family})\}$ and $R_2 = \{(0, \top_{HbA1c\%}), (1, Precise), (2, Imprecise)\}$. A graphical illustration of the precision MO is seen in Figure 3.5.

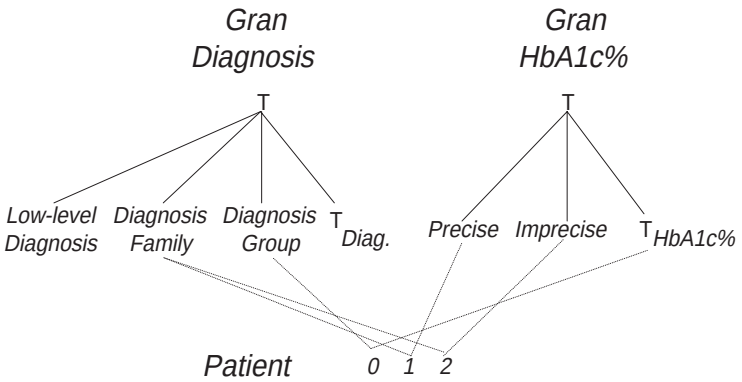

Figure 3.5: Precision MO

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 $Q_p = \alpha[G_1, .., G_n, G_{n+1}, SetCount](M_p)$, 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 $Type(C_i) \sqsubseteq_{\mathcal{T}_i} Type(C_i') \wedge \forall C_{ij}(f \in F \wedge (f, e) \in R_i \wedge e \in C_{ij} \Rightarrow Type(C_{ij}) \sqsubset_{\mathcal{T}_i} Type(C_i'))$, 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. The physician can then use this additional information to decide whether to run the alternative query or proceed with the original one.

Example 20 The physician wants to know the average HbA1c% grouped by Diagnosis Family. The query asked is $Q = \alpha[$*Diagnosis Family* $, \top_{HbA1c\%}, D_3, AVG_2](M)$, thus effectively grouping only on Diagnosis Family, as the $\top_{HbA1c\%}$ component has only one value. The testing query then becomes $Q_p = \alpha[Granularity_{Diagnosis}, \top_{GranHbA1c\%}, D_3, SetCount](M_p)$, which counts the number of facts with the different Diagnosis granularity levels. The result of Q_p, described by the fact-dimension relations, is $R_1 = \{(\{1, 2\},$ *Diagnosis Family*$), (\{0\},$ *Diagnosis Group*$)\}$, $R_2 = \{(\{1, 2\}, \top_{GranHbA1c\%}), (\{0\}, \top_{GranHbA1c\%})\}$, and $R_3 = \{\{(\{1, 2\}, 2), (\{0\}, 1)\}$. This tells us that 2 patients are diagnosed with a diagnosis family, while 1 is diagnosed with a diagnosis group diagnosis. Thus, the alternative query will be $Q = \alpha[$*Diagnosis Group*$, \top_{HbA1c\%}, D_3, AVG_2](M)$, which groups on Diagnosis Group rather than Diagnosis Family.

[4]To avoid unnecessary complexity of the examples, we consider only one diagnosis for Jane Doe in this section, namely the diagnosis "9" (Insulin Dependent Diabetes, current version).

3.6 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.

3.6.1 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 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 3.8, 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:

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

$., e'_i, .., e_n) \subseteq Group(e_1, .., e_i, .., e_n)))\}$ 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 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 $R_2 = \{(\{0, 1\}, 2), (\{0, 2\}, 2)\}$. We see that 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 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* [136]. 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 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 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 weight on all other links is 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, giving 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 $R'_1 = \{(\{0_{.8}, 2_1\}, \textit{Insulin-dependent Diabetes}), (\{0_{.2}, 1_1\}, \textit{Non-insulin-dependent Diabetes}\ )\}$ 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.

3.6.2 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 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* [106]. 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 $h : \text{CM} \times \text{CM} \mapsto \text{CM}$, 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 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 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 $CM = \mathcal{N} \times \mathcal{N}$, 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 25 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 [113]. 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* [106, 11].

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 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.

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 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 27 For 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 $(\text{Insulin-dependent diabetes}, 5.5, 0), (\text{Non-insulin-dependent Diabetes}, 7, 1)$. For the liberal answer, the result is $(\text{Insulin-dependent diabetes}, 5.8, 1), (\text{Non-insulin-dependent Diabetes}, 6.5, 1.5)$. For the weighted answer, the result is $(\text{Insulin-dependent diabetes}, 5.7, .9), (\text{Non-insulin-dependent Diabetes}, 5.6, 1.2)$.

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 $c : \bot_D \times M \mapsto D$, 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 $x \sqsubseteq v \wedge \mathit{Level}(v) = \mathit{Ceiling}(x)$, 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 3.6. 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.

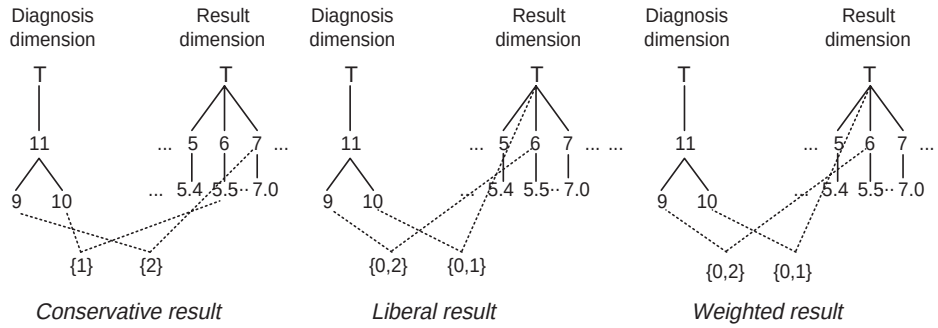

Figure 3.6: Resulting MO's for the Conservative, Liberal, and Weighted Answers

3.7 Addressing the Requirements

In this section, we discuss how our model addresses the eleven requirements presented in Section 3.2.2.

The model captures the *explicit hierarchies in dimensions* using the lattice structure of the dimension types. The structure of the case study, seen in Figure 3.2, 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 3.2 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.

An value in a dimension may have several direct parents in the model , e.g., the diagnosis "Insulin dependent diabetes during pregnancy" has both "Insulin dependent diabetes" and "Diabetes during pregnancy" as direct parents in the Diagnosis dimension. Thus, *non-strict hierarchies* in dimensions are supported. *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, we can view data as it appeared at any given point in time. By extending the partial order of a dimension, it is possible to link values that represent the "same" thing across change, e.g., the old and the new "Diabetes" diagnosis. In this way, we may obtain meaningful analysis results across changes in the data. In this respect, the model supports *handling change and time*.

The dimension values that are part of the fact-dimension relations can belong to any category in the dimension, supporting in this manner *different levels of granularity* in the data. For example, we can express that some patients are 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.

3.8 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 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 [107, 108, 14], 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 [86]. 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 [70], which intuitively means that lower-level results can be directly combined into higher-level results. It has been proven [70] that summarizability is equivalent to the hierarchies in dimensions being *strict*, *onto*, and *covering*, 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 3.5.2. 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 *Set-Count* 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 [70] 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 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 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 $\top_{Diagnosis}$ 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 unweighted.

3.9 Conclusion and Future Work

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

We present a real-world case study from the clinical world, where we track patients, their diagnoses, names, dates of birth, ages, places of residence, and their HbA1C%, an indication of the long-term blood sugar level. We use the case study to justify eleven requirements that a multidimensional data model must satisfy in order to support the complex data found in real-world applications. Requirements not handled by current models include many-to-many relationships between facts and dimensions, handling change and time, handling imprecise data, and handling different levels of granularity. Twelve previously proposed data models are evaluated according to the requirements, and it is shown that none of them satisfies more than five requirements fully or partially.

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. Especially, time is handled by adding valid time and transaction time to the basic model. We propose an algebra on the multidimensional objects from the model, and we show that it is closed and at least as strong as relational algebra with aggregation functions. The algebra is extended to handle time.

We show how to use the presented data model to capture imprecise data using the concept of *data granularity*. 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. Finally, we show how to represent the

multidimensional objects as relational tables and how to implement the imprecision handling approach using SQL, thus providing a basis for implementing the model using relational technology.

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 should be investigated how the model and query handling techniques may be efficiently implemented using special-purpose algorithms and data structures, to achieve optimal concrete complexity. Next, a notion of completeness for multidimensional algebras, similar to Codd's relational completeness would be an exciting research topic. We also believe that it is important to investigate how multidimensional models can cope with the hundreds of dimensions found in some applications. There are several future research directions related to imprecision. The investigation of the issues related to "single-value" aggregation functions such as MIN and MAX, that are not readily sensitive to weighting, in relation to data granularity is warranted. It would also be interesting to explore other means of graphically presenting imprecision in the result to facilitate the user interpretation of an imprecise result, or to possibly present the user with the data that *prevented* a given query from being precisely answerable. Also, it would be good to give precise measures for the usefulness of technique, given the available data.

3.10 Relational Representation of the Model

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

The traditional way to map a multidimensional data model to a relational database is to use a *star schema* [64], where the *fact table* contains measures and foreign keys to the *dimension tables*. However, a star schema design requires the relationships between the fact and dimension tables to be many-to-one, and that the hierarchies in the dimensions be strict. To represent many-to-many relationships between facts and dimensions, several rows in the fact table are necessary for each fact. To represent non-strict hierarchies, several rows in the dimension tables are necessary for each dimension key. These violations of the pure star schema design can lead users to get incorrect results when aggregating data, as it is easy to accidentally double-count data. Alternatively, if the users understand the potential problems, they need to employ the expensive SELECT DISTINCT clause in SQL statements to get correct results.

To avoid these problems, we use a non-standard mapping to relational tables. The basic idea for representing the dimensions is to encode the partial order on a dimension composed with a category representation, directly in one table. Thus, for each representation Rep of a category C_j in the given MO, we get a table $T_{\mathrm{C_j-Rep}}$ that encodes the composition of Rep with the partial order on the dimension. If Rep is encoded in the table $T_{\mathrm{Rep}} = (RepValue, DimensionValue)$ and the direct parent-child relationships in the partial order is encoded in the table $PO = (ParentValue, ChildValue)$ then $T_{\mathrm{C_j-Rep}} = T_{\mathrm{Rep}} \bowtie_{DimensionValue=ParentValue} PO^*$, 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 3.1 encodes the direct parent-child relationships in the Diagnosis dimension. We perform the reflexive, transitive closure of the Grouping table, thus getting all ancestor-descendent pairs in the Diagnosis partial order. This is joined with the Diagnosis table, which encodes the *Code* representation for the Diagnosis Group category. This gives us the the table $T_{\mathrm{DiagGroup-Code}}$ which can be seen in Table 3.3. 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 3.3: The $T_{\mathrm{DiagGroup-Code}}$ Table for the Example

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

Example 30 For the example above, alternative a) would maintain the join of $T_{\mathrm{DiagGroup-Code}}$ 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 $T_{\mathrm{DiagGroup-Code}}$ table. The fact table would then point to these combinations, instead of the diagnoses directly. Alternative c) would just keep the $T_{\mathrm{DiagGroup-Code}}$ and Has tables.

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

When extending the representations to capture valid/transaction time, the basic dimension-encoding mechanism still works. The encoding table is extended with columns capturing the time when the tuple is true. We take the intersection of the time periods when tables are joined, thus capturing the time period when the combined tuples are valid. The $T_{\mathrm{DiagGroup-Code}}$ table extended with time is seen in Table 3.3. Alternative a) and c) can be extended with time columns without any problems. For alternative b) we need to enumerate all combinations of dimension values *and* the associated time periods. This will probably lead to a number of combinations that is close to the number of facts, thus rendering the solution impractical.

3.11 SQL Implementation of Imprecision

This section discusses 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 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* [64]. 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 [64], 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 [105], a technique optimized for "multi-dimensional" 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, 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 [64]. 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, 43], 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 $g(\,Comb(D_1.W, .., D_m.W) * D_k.EV),$
$\quad GCF(\,Comb(D_1.W, .., D_m.W) * D_k.GCM)$
FROM $F, D_1, .., D_m$
WHERE
$\quad F.DK_1 = D_1.DK$ AND ... AND $F.DK_m = D_m.DK$ AND
$\quad F.DK_k = D_k.DK$ AND $D_k.L_A = D_k.L_D$ AND
$\quad (D_1.L_A = GL_1$ OR $(D_1.L_A > GL_1$ AND $D_1.L_A = D_1.L_D))$ AND
$\quad ....$
$\quad (D_m.L_A = GL_m$ 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.

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

Fact Table

AnsID	DesID	Ancestor	AnsLevel	DesLevel	W
9	9	Ins. dep. diab.	0	0	1
10	10	Non-ins. dep. diab.	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 3.4: Relational Implementation of the Case Study

Example 31 We implement the MO from the case study[5] 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 3.4. When using SQL to compute the weighted average of the HbA1c%, grouped by Diagnosis Family seen in Example 26, 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 26. The SQL statements are seen below.

[5]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.

```
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 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 [64] to access the aggregate tables.

Chapter 4

Extending Practical Pre-Aggregation in On-Line Analytical Processing

4.1 Introduction

On-Line Analytical Processing (OLAP) systems, which aim to ease the process of extracting useful information from large amounts of detailed transactional data, have gained widespread acceptance in traditional business applications as well as in new applications such as health care. These systems generally offer a dimensional view of data, in which measured values, termed facts, are characterized by descriptive values, drawn from a number of dimensions; and the values of a dimension are typically organized in a containment-type hierarchy. A prototypical query applies an aggregate function, such as average, to the facts characterized by specific values from the dimensions.

Fast response times are required from these systems, even for queries that aggregate large amounts of data. The perhaps most central technique used for meeting this requirement is termed *pre-aggregation*, where the results of aggregate queries are pre-computed and stored, i.e., materialized, for later use during query processing. Pre-aggregation has attracted substantial attention in the research community, where it has been investigated how to optimally use pre-aggregated data for query optimization [44, 25] and how to maintain the pre-aggregated data when base data is updated [81, 101]. Further, the latest versions of commercial RDBMS products offer query optimization based on pre-computed aggregates and automatic maintenance of the stored aggregates when base data is updated [131].

The fastest response times may be achieved when materializing aggregate results corresponding to all combinations of dimension values across all dimensions, termed *full* (or eager) pre-aggregation. However, the required storage space grows rapidly, to quickly become prohibitive, as the complexity of the application increases. This phenomenon is called *data explosion* [27, 117, 86] and occurs because the number of possible aggregation combinations grows rapidly when the number of dimensions increase, while the sparseness of the multidimensional space decreases in higher di-

mension levels, meaning that aggregates at higher levels take up nearly as much space as lower-level aggregates. In some commercial applications, full pre-aggregation takes up as much as 200 times the space of the raw data [86]. Another problem with full pre-aggregation is that it takes too long to update the materialized aggregates when base data changes.

With the goal of avoiding data explosion, research has focused on how to select the best subset of aggregation levels given space constraints [49, 46, 135, 4, 124, 112] or maintenance time constraints [47], or the best combination of aggregate data and indices [45]. This approach is commonly referred to as *practical* (or partial or semi-eager [31, 49, 130]) pre-aggregation. Commercial OLAP systems now also exist that employ practical pre-aggregation, e.g., Microsoft Decision Support Services (Plato) [79] and Informix MetaCube [56].

The premise underlying the applicability of practical pre-aggregation is that lower-level aggregates can be *re-used* to compute higher-level aggregates, known as summarizability [70]. Summarizability occurs when the mappings in the dimension hierarchies are *onto* (all paths in the hierarchy have equal lengths), *covering* (only immediate parent and child values can be related), and *strict* (each child in a hierarchy has only one parent); and when also the relationships between facts and dimensions are many-to-one and facts are always mapped to the lowest levels in the dimensions [70]. However, the data encountered in many real-world applications fail to comply with this rigid regime. This motivates the search for techniques that allow practical pre-aggregation to be used for a wider range of applications, the focus of this chapter.

Specifically, this chapters leverages research such as that cited above. It does so by showing how to transform dimension hierarchies to obtain summarizability, and by showing how to integrate the transformed hierarchies into current systems, transparently to the user, so that standard OLAP technology is re-used. Specifically, algorithms are presented that automatically transform dimension hierarchies to achieve summarizability for hierarchies that are non-onto, non-covering, and non-strict. The algorithms have low computational complexity, and are thus applicable to even very large databases. It is also described how to use the algorithms to contend with non-summarizable relationships between facts and dimensions, and it is shown how the algorithms may be modified to accommodate incremental computation, thus minimizing the maintenance cost associated with base-data updates.

To our knowledge, this work is the first to present algorithms to automatically achieve summarizability for non-covering and non-onto hierarchies. The research reported here is also the first to demonstrate techniques and algorithms for achieving summarizability in non-strict hierarchies. The integration of the techniques into current systems, transparently to the user, we believe is a novel feature. The only past research on the topic has been on how to manually, and not transparently to the user, achieve summarizability for non-covering hierarchies [103].

The next section presents a real-world clinical case study that exemplifies the non-summarizable properties of real-world applications. Section 4.3 proceeds to define the aspects of a multidimensional data model necessary for describing the new techniques, and defines also important properties related to summarizability. Sections 4.4 and 4.5 present algorithms that transform dimension hierarchies to achieve

summarizability, then apply the algorithms to fix non-summarizable relationships between facts and dimensions. Section 4.6 demonstrates how the techniques may be integrated into current systems, transparently to the user. Section 4.7 summarizes and points to topics for future research. Appendix 4.8 describes how to modify the algorithms to accommodate incremental computation.

4.2 Motivation—A Case Study

This section presents a case study that illustrates the properties of real-world dimension hierarchies. The case study concerns patients in a hospital, their associated diagnoses, and their places of residence. The data analysis goal is to investigate whether some diagnoses occur more often in some areas than in others, in which case environmental or lifestyle factors might be contributing to the disease pattern. An ER diagram illustrating the underlying data is seen in Figure 4.1.

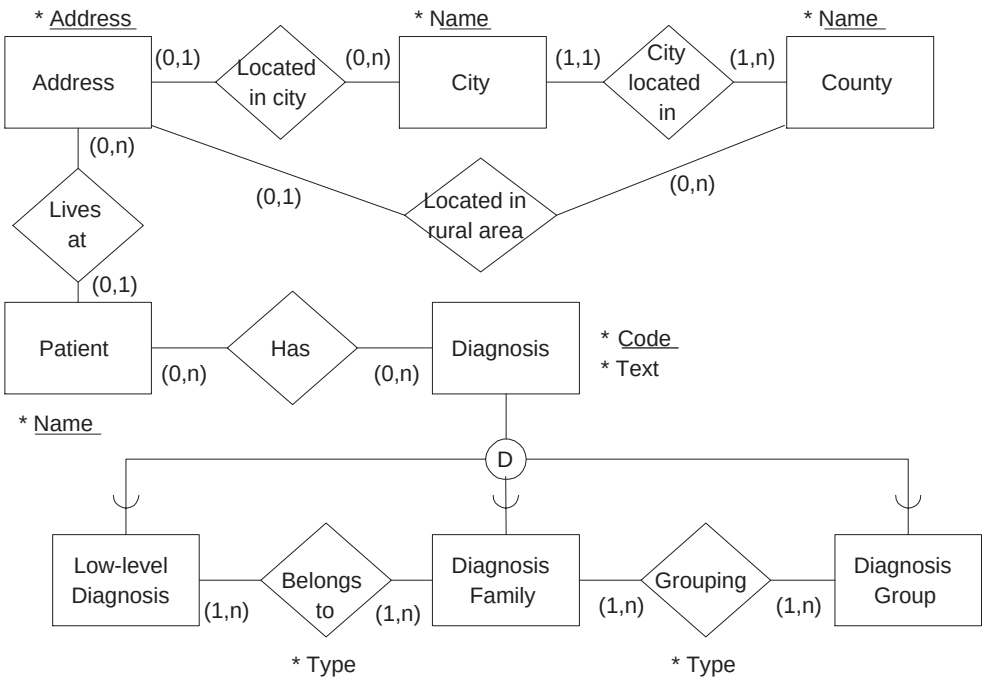

Figure 4.1: ER Schema of Case Study

The most important entities are the *patients*, for which we record the name. We always want to count the number of patients, grouped by some properties of the patients. Thus, in multidimensional terms, the patients are the *facts*, and the other, describing, entities constitute the *dimensions*.

Each patient has a number of *diagnoses*, leading to a *many-to-many* relationship between facts and the diagnosis dimension. When registering diagnoses of patients, physicians use different levels of granularity, ranging from very precise diagnoses, e.g., “Insulin dependent diabetes during pregnancy,” to more imprecise diagnoses,

e.g., "Diabetes," which cover wider ranges of patient conditions. To model this, the relationship from patient to diagnoses is to the supertype "Diagnosis," which then has three subtypes, corresponding to different levels of granularity, the *low-level diagnosis*, the *diagnosis family*, and the *diagnosis group*. Examples of these are "Insulin dependent diabetes during pregnancy," "Insulin dependent diabetes," and "Diabetes," respectively. The higher-level diagnoses are both (imprecise) diagnoses in their own right, but also serve as groups of lower-level diagnoses.

Each diagnosis has an alphanumeric code and a descriptive text, which are specified by some standard, here the World Health Organization's International Classification of Diseases (ICD-10) [133], or by the physicians themselves. Indeed, two hierarchies are captured: the standard hierarchy specified by the WHO, and the user-defined hierarchy, which is used for grouping diagnoses on an ad-hoc basis in other ways than given by the standard. The *Type* attribute on the relationships determines whether the relation between two entities is part of the standard or the user-defined hierarchy.

The hierarchy groups low-level diagnoses into *diagnosis families*, each of which consists of 2–20 related diagnoses. For example, the diagnosis "Insulin dependent diabetes during pregnancy[1]" is part of the family "Diabetes during pregnancy." In the WHO hierarchy, a low-level diagnosis belongs to exactly one diagnosis family, whereas the user-defined hierarchy does not have this restriction. Thus, a low-level diagnosis can belong to several diagnosis families, e.g., the "Insulin dependent diabetes during pregnancy" diagnosis belongs to both the "Diabetes during pregnancy" and the "Insulin dependent diabetes" family. Next, diagnosis families are grouped into *diagnosis groups*, consisting of 2–10 families, and one family may be part of several groups. For example, the family "Diabetes during pregnancy" may the part of the "Diabetes" and "Other pregnancy related diseases" groups.

In the WHO hierarchy, a family belongs to exactly one group. In the WHO hierarchy, a lower-level value belongs to exactly one higher-level value, making it *strict* and *covering*. In the user-defined hierarchy, a lower-level value may belong to zero or more higher-level values, making it *non-strict* and *non-covering*. Properties of the hierarchies will be discussed in more detail in Section 4.3.2.

We also record the addresses of the patients. If the address is located in a city, we record the *city*; otherwise, if the address is in a rural area, we record the *county* in which the address is located. A city is located in exactly one county. As not all addresses are in cities, we cannot find all addresses in a county by going through the "City located in" relationship. Thus, the mapping from addresses to cities is *non-covering* w.r.t. addresses. For cities and counties, we just record the name. Not all counties have cities in them, so the mapping from cities to counties is *into* rather than *onto*.

In order to exemplify the data, we assume a standard mapping of the ER diagram to relational tables, i.e., one table per entity and relationship type. We also assume the use of surrogate keys, named *ID*, with globally unique values. The three subtypes of the Diagnosis type are mapped to a common Diagnosis table, and because of this,

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

the "belongs to" and "grouping" relationships are mapped to a common "Grouping" table. The resulting tables with sample data are shown in Table 4.1 and will be used in examples throughout the chapter.

If we apply pre-aggregation to the data from the case study, several problems occur. For example, if the counts of patients by City are pre-computed and we use these for computing the numbers of patients by county, an incorrect result will occur. In the data, the addresses "123 Rural Road" and "1 Sandy Dunes" (one of them is the address of a patient) are not in any city, making the mapping from City to County not *covering* w.r.t. addresses.

Next, if the counts of patients by Low-Level Diagnosis are pre-computed and we use these for computing the total count of patients, an incorrect result again ensues. First, patients only with lung cancer are not counted, as lung cancer is not present at the level of Low-Level Diagnosis; the mapping from Low-Level Diagnosis to Diagnosis Family is *into*. Second, patients such as "Jim Doe" only have higher-level diagnoses and will no be counted; the fact-to-dimension mapping has *varying granularity*. Third, patients such as "Jane Doe" have several diagnoses and will be counted several times; , the relationship between facts and dimensions is *many-to-many*. Fourth, Low-Level diagnoses such as "Insulin dependent diabetes during pregnancy" are part of several diagnosis families, which may also lead to "double" counting when computing higher-level counts; the dimension hierarchy is *non-strict*.

These problems yield "non-summarizable" dimension hierarchies that severely limit the applicability of practical pre-aggregation, leaving only full pre-aggregation, requirering huge amounts of storage, or no pre-aggregation, resulting in long response time for queries.

The properties described above are found in many other real-world applications. Many-to-many relationships between facts and dimensions occur between bank customers and accounts, between companies and Standard Industry Classifications (SICs), and between students and departments [65, 70]. Non-strict dimension hierarchies occur from cities to states in a Geography dimension [103] and from weeks to months in a Time dimension. In addition, hierarchies where the change over time is captured are generally non-strict. The mapping from holidays to weeks as well as organization hierarchies of varying depth [53] offer examples of "into" mappings. Non-covering relationships exist for days-holidays-weeks and for counties-cities-states, as well as in organization hierarchies [53].

Even though many real-world cases possess the properties described above, current techniques for practical pre-aggregation require that facts are in a many-to-one relationships to dimensions and that all hierarchies are strict, onto, and covering. Thus, current techniques cannot be applied when the hierarchies has these properties.

4.3 Method Context

This section defines the aspects of a multidimensional data model that are necessary to define the techniques that enable practical pre-aggregation in applications as the one just described. The full model is described elsewhere [95]. Next, the data model context is exploited for defining properties of hierarchies relevant to the techniques.

ID	Name
1	John Doe
2	Jane Doe
3	Jim Doe

Patient

PatientID	AddressID
1	50
2	51
3	52

LivesAt

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

Address

PatientID	DiagnosisID	Type
1	9	Primary
2	5	Secondary
2	9	Primary
3	11	Primary

Has

ParentID	ChildID	Type
4	5	WHO
4	6	WHO
9	5	User-defined
10	6	User-defined
11	9	WHO
11	10	WHO
12	4	WHO
13	14	WHO

Grouping

ID	Code	Text	Type
4	O24	Diabetes during pregnancy	Family
5	O24.0	Insulin dependent diabetes during pregnancy	Low-Level
6	O24.1	Non insulin dependent diabetes during pregnancy	Low-Level
9	E10	Insulin dependent diabetes	Family
10	E11	Non insulin dependent diabetes	Family
11	E1	Diabetes	Group
12	O2	Other pregnancy related diseases	Group
13	A1	Cancer	Group
14	A11	Lung cancer	Family

Diagnosis

ID	Name
20	Sydney
21	Melbourne

City

AddressID	CityID
50	20
51	21

LocatedInCity

ID	Name
30	Sydney
31	Melbourne
32	Outback

County

ID	Name
52	31
53	32

LocatedInRuralArea

CityID	CountyID
20	30
21	31

CityLocatedIn

Table 4.1: Tables for the Case Study

The particular data model has been chosen over other multidimensional data models because it quite naturally captures the data described in the case study and because

it includes explicit concepts of dimensions and dimension hierarchies, which is very important for clearly presenting the techniques. However, the techniques are also applicable to other multidimensional or statistical data models, as will be discussed in Section 4.6.

4.3.1 A Concrete Data Model Context

For each part of the model, we define the *intension* and the *extension*, and we 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 32 In the case study from Section 4.2, *Patient* is the fact type, and *Diagnosis*, *Residence*, and *Name* are the dimension types. The intuition is that *everything* that characterizes the fact type is considered to be *dimensional*.

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 value size. The top element of the ordering corresponds to the largest possible value size, that is, there is only one value in it's extension, logically containing all other values.

We say that $\mathcal{C}_j$ *is a category type of* $\mathcal{T}$, written $\mathcal{C}_j \in \mathcal{T}$, if $\mathcal{C}_j \in \mathcal{C}$.

Example 33 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: $\bot_{Diagnosis}$ = *Low-level Diagnosis* $\sqsubset$ *Diagnosis Family* $\sqsubset$ *Diagnosis Group* $\sqsubset \top_{Diagnosis}$. Other examples of category types are *Address*, *City*, and *County*. Figure 4.2, 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 $\mathcal{T} = (\{\mathcal{C}_j\}, \sqsubseteq_\mathcal{T}, \top_\mathcal{T}, \bot_\mathcal{T})$ 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. We assume a function $Pred : C \mapsto 2^C$ that gives the set of immediate predecessors of a category C_j. Similarly, we a assume a function $Desc : C \mapsto 2^C$ that gives the set of immediate descendants of a category C_j. For both $Pred$ and $Desc$, we "count" from the category $\top_\mathcal{T}$ (of type $\top_\mathcal{T}$), so that category $\top_\mathcal{T}$ is the ultimate predecessor and category $\bot_\mathcal{T}$ (of type $\bot_\mathcal{T}$) is the ultimate descendant.

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 $e \in \cup_j C_j$.

The category of type $\perp_{\mathcal{T}}$ in dimension of type $\mathcal{T}$ contains the values with the smallest value size. The category with the largest value size, with type $\top_{\mathcal{T}}$, contains exactly one value, denoted $\top$. For all values e of the dimension D, $e \sqsubseteq \top$. Value $\top$ is similar to the *ALL* construct of Gray et al. [40]. When the context is clear, we refer to a category of type $\top_{\mathcal{T}}$ as a $\top$ category, not to be confused with the $\top$ dimension value.

Example 34 In our *Diagnosis* dimension we have the following categories, named by their type. The numbers in parentheses are the ID values from the Diagnosis table in Table 4.1. *Low-level Diagnosis* = {"Insulin dependent diabetes during pregnancy" (5), "Non insulin dependent diabetes during pregnancy" (6)}, *Diagnosis Family* = {"Diabetes during pregnancy" (4), "Insulin dependent diabetes" (9), "Non insulin dependent diabetes" (10), "Lung cancer" (14)}, *Diagnosis Group* = {"Diabetes" (11), "Other pregnancy related diseases" (12), "Cancer" (13)}, and $\top_{Diagnosis} = \{\top\}$. We have that $Pred(\textit{Low-level Diagnosis}) = \{\textit{Diagnosis Family}\}$. The partial order $\sqsubseteq$ is obtained by combining WHO and user-defined hierarchies, as given by the Grouping table in Table 4.1. 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 $D = (C = \{C_j\}, \sqsubseteq)$ 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 $\exists e_1 \in D\ ((f, e_1) \in R \wedge e_1 \sqsubseteq e)$. We require that $\forall f \in F\ (\exists e \in \cup_j C_j\ ((f, e) \in R))$; thus, all fact maps to at least one dimension value in every dimension. The $\top$ value is used to represent an unknown or missing value, as $\top$ logically contains all dimension values, and so a fact f is mapped to $\top$ if it cannot be characterized within the particular dimension.

Example 35 The fact-dimension relation R links patient facts to diagnosis dimension values as given by the Has table from the case study, so that $R = \{$("John Doe" (1), "Insulin dependent diabetes" (9)), ("Jane Doe" (2), "Insulin dependent diabetes during pregnancy" (5)), ("Jane Doe" (2), "Insulin dependent diabetes" (9)), ("Jim Doe" (3), "Diabetes" (11))$\}$. Note that facts may be related to values in higher-level categories. We do not require that e belongs to $\perp_{Diagnosis}$. For example, the fact "John Doe" (1) is related to the diagnosis "Insulin dependent diabetes" (5), which belongs to the *Diagnosis Family* category. This feature will be used later to explicitly capture the different granularities in the data. If no diagnosis was known for patient "John Doe" (1), we would have added the pair ("John Doe" (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 36 For the case study, we get a three-dimensional MO $M = (\mathcal{S}, F, D, R)$, where $\mathcal{S} = (\textit{Patient}, \{\textit{Diagnosis, Name, Residence}\})$ and $F = \{$"John Doe" (1),

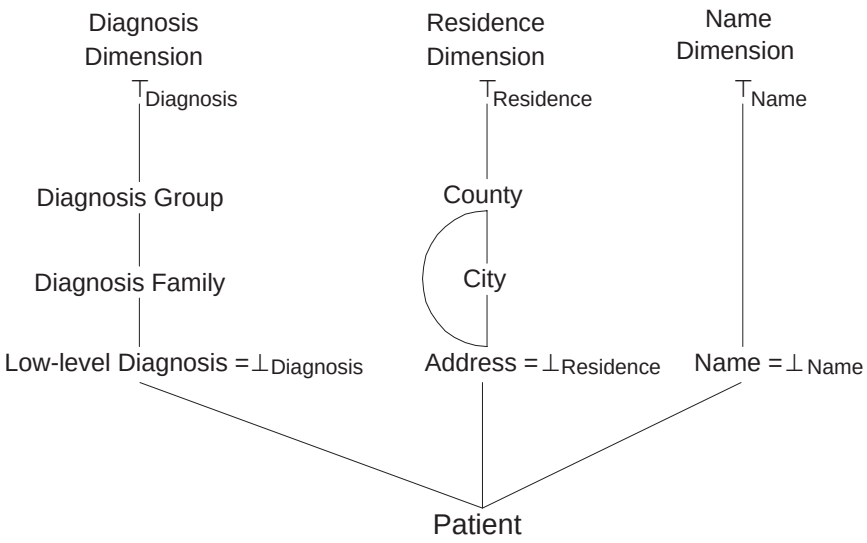

Figure 4.2: Schema of the Case Study

"Jane Doe" (2), "Jim Doe" (3)}. The definition of the diagnosis dimension and its corresponding fact-dimension relation was given in the previous examples. The Residence dimension has the categories *Address* ($= \perp_{Residence}$), *City*, *County*, and $\top_{Residence}$. The values of the categories are given by the corresponding tables in Table 4.1. The partial order is given by the relationship tables. Additionally, the only value in the $\top_{Residence}$ category is $\top$, which logically contains all the other values in the Residence dimension. The Name dimension is simple, i.e., it just has a *Name* category ($= \perp_{Name}$) and a $\top$ category. We will refer to this MO as the "Patient" MO. A graphical illustration of the schema of the "Patient" MO is seen in Figure 4.2. Because some addresses map directly to counties, County is an immediate predecessor of Address.

The facts in an MO are objects with *value-independent identity*. We can test facts for equality, but do not assume an ordering on the facts. The combination of dimensions values that characterize the facts of a fact set is *not* a "key" for the fact set. Thus, several facts may be characterized by the same combination of dimension values. But, the facts of an MO is a *set*, so an MO does not have duplicate *facts*. The model formally defines quite general concepts of dimensions and dimension hierarchies, which is ideal for the presentation of our techniques. The presented techniques are not limited by the choice of data model.

4.3.2 Hierarchy Properties

In this section important properties of MOs that relate to the use of pre-computed aggregates are defined. The properties will be used in the following sections to state exactly what problems the proposed algorithms solve. The first important concept

is *summarizability*, which intuitively means that higher-level aggregates may be obtained directly from lower-level aggregates.

Definition 5 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 argument on the left-hand side of the equation is a multiset, i.e., the same value may occur multiple times.

Summarizability is important 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 (efficiently) compute higher-level aggregates on-the-fly. Instead, we have to pre-compute the all the aggregate results of queries 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 base data is often inefficient.

It has been shown that summarizability is equivalent to the aggregate function (g) being *distributive*, all paths being *strict*, and the mappings between dimension values in the hierarchies being *covering* and *onto* [70]. These concepts are formally defined below. The definitions assume a dimension $D = (C, \sqsubseteq)$ and an MO $M = (\mathcal{S}, F, D, R)$.

Definition 6 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 *into*. If all mappings in a dimension are onto, we say that the dimension hierarchy is *onto*.

Mappings that are into typically occur when the dimension hierarchy has varying height. In the case study, there is no low-level cancer diagnosis, meaning that some parts of the hierarchy have height 2, while most have height 3. It is thus not possible to use aggregates at the Low-level Diagnosis level for computing aggregates at the two higher levels. Mappings that are into also occur often in organization hierarchies.

Definition 7 Given three categories, C_1, C_2, and 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*.

Non-covering mappings occur when some of the links between dimension values skip one or more levels and map directly to a value located higher up in the hierarchy. In the case study, this happens for the "1 Sandy Dunes" address, which maps directly to "Outback County" (there are no cities in Outback County). Thus, we cannot use aggregates at the City level for computing aggregates at the County level.

Definition 8 Given an MO $M = (\mathcal{S}, F, D, R)$, and two categories C_1 and C_2 that belong to the same dimension $D_i \in D$ such that $Type(C_1) \sqsubset Type(C_2)$, we say that the mapping from C_1 to C_2 is *covering with respect to* F, the set of facts, iff $\forall f \in F\,(\forall e_2 \in C_2\,(f \rightsquigarrow_i e_2 \Rightarrow \exists e_1 \in C_1\,(f \rightsquigarrow_i e_1 \wedge e_1 \sqsubseteq_i e_2)))$.

This case is similar to the one above, but now it is the mappings between facts and dimension values that may skip one or more levels and map facts directly to dimension values in categories above the bottom level. In the case study, the patients can map to diagnoses anywhere in the Diagnosis dimension, not just to Low-level Diagnoses. This means that we cannot use aggregates at the Low-level Diagnosis Level for computing aggregates higher up in the hierarchy.

Definition 9 Given two categories, C_1 and C_2 such that $C_2 \in Pred(C_1)$, we say that the mapping from C_1 to C_2 is *strict* iff $\forall e_1 \in C_1\,(\forall e_2, e_3 \in C_2\,(e_1 \sqsubseteq e_2 \wedge e_1 \sqsubseteq e_3 \Rightarrow e_2 = e_3))$. 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 an MO $M = (\mathcal{S}, F, D, R)$ and a category C_j in some dimension $D_i \in D$, we say that there is a *strict path* from the set of facts F to C_j iff $\forall f \in F(f \rightsquigarrow_i e_1 \wedge f \rightsquigarrow_i e_2 \wedge e_1 \in C_j \wedge e_2 \in C_j \Rightarrow e_1 = e_2)$. (Note that the paths to the $\top_{\mathcal{T}}$ categories are always strict.)

Non-strict hierarchies occur when a dimension value has multiple parents. This occurs in the Diagnosis dimension in the case study where the "Insulin dependent diabetes during pregnancy" low-level diagnosis is part of both the "Insulin Dependent Diabetes" and the "Diabetes during pregnancy" diagnosis families, which in turn both are part of the "Diabetes" diagnosis group. This means that we cannot use aggregates at the Diagnosis Family level to compute aggregates at the Diagnosis Group level, since data for "Insulin dependent diabetes during pregnancy" would then be counted twice.

Definition 10 If the dimension hierarchy for a dimension D is *onto*, *covering*, and *strict*, we say that D is *normalized*. Otherwise, it is *un-normalized*. For an MO $M = (\mathcal{S}, D, F, R)$, if all dimensions $D_i \in D$ are normalized and $\forall R_i \in R\,((f, e) \in R_i \Rightarrow e \in \perp_D)$ (, i.e., all facts map to dimension values in the bottom category), we say that M is *normalized*. Otherwise, it is *un-normalized*.

For normalized hierarchies and MOs, all mappings are summarizable, meaning that we can pre-aggregate values at any combination of dimension levels and safely re-use the pre-aggregated values to compute higher-level aggregate results. Thus, we want to normalize the dimension hierarchies and MOs for which we want to apply practical pre-aggregation.

We proceed to describe how the normalization of the dimension hierarchies and MOs used for aggregation is achieved. We first show how to perform transformations on dimension hierarchies, then later describe how the same techniques may be applied to eliminate the non-summarizable properties of fact-dimension relations.

4.4 Dimension Transformation Techniques

This section describes how dimensions can be transformed to achieve summarizability. Transforming dimensions on their own, separately from the facts, results in well-behaved dimensions that can be applied in a number of different systems or sold to third-party users. The transformation of the dimension hierarchies is a three-step operation. First, all mappings are transformed to be *covering*, by introducing extra "intermediate" values. Second, all mappings are transformed to be *onto*, by introducing "placeholder" values at lower levels for values without any children. Third, mappings are made *strict*, by "fusing" values together. The three steps are treated in separate sections. None of the algorithms introduce any non-summarizable properties, so applying each once is sufficient.

In general, the algorithms take as input a set of tables R_{C_1,C_2} that specifies the mapping from dimension values in category C_1 to values in category C_2. The input needs not contain all pairs of ancestors and descendants—only direct parent-child relationships are required. If there are non-covering mappings in the hierarchy, we have categories C, P, H such that $\{P, H\} \subseteq Pred(C)$ and $Type(P) \sqsubset Type(H)$. In this case, the input must also contain $R_{P,H}$ tables that map P values to H values.

4.4.1 Non-Covering Hierarchies

The first algorithm renders all mappings in a dimension hierarchy covering w.r.t. any category. When a dimension value is mapped *directly* to another value in a category higher than the one immediately above it in the hierarchy, a new intermediate value is inserted into the category immediately above, and the two original dimension values are linked to this new value, rather than to each other.

Example 37 In the hierarchy for the Residence dimension, two links go from Address directly to County. The address "123 Rural Road" (52) is in "Melbourne County" (31), but not in a city, and the address "1 Sandy Dunes" (53) is in "Outback County" (32), which does *not* have any cities at all. The algorithm inserts two new dimension values in the City category, **C31** and **C32**, which represent Melbourne and Outback county, respectively, and links them to their respective counties. The addresses "123 Rural Road" and "1 Sandy Dunes" are then linked to **C31** and **C32**, respectively. This occurs in the first call of procedure MakeCovering (on the Address category; the procedure is given below). When MakeCovering is called recursively on the City, County, and $\top$ categories, nothing happens, as all mappings are already covering. The transformation is illustrated graphically in Figure 4.3. The dotted lines show the "problematic" links, and the bold-face values and thick lines show the new dimension values and links.

In the algorithm, C is a *child* category, P is a *parent* category, H is a "*higher*" category, L are the non-covering *links* from C to H, and N are the "higher" dimension values in L. The $\bowtie$ operator denotes natural join. The algorithm works as follows. Given the argument category C (initially the bottom category) in line (1), the algorithms goes through all C's parent categories P (2). For each parent category P, it looks for predecessor categories H of C that are "higher" in the hierarchy than

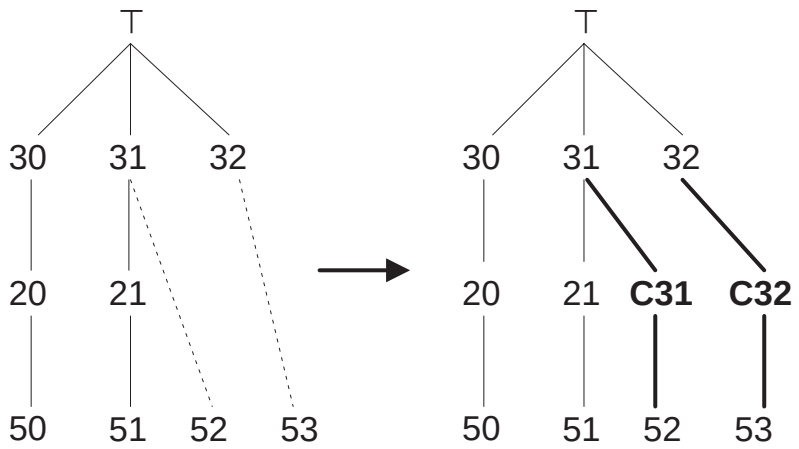

Figure 4.3: Transformations by the MakeCovering Algorithm

P (4). If such an H exist, there might be links in the mapping from C to H that are not available by going through P. Line (6) finds these "non-covered" links, L, in the mapping from C to H by "subtracting" the links that *are* available by going through P from *all* the links in the mapping from C to H. Line (7) uses L to find the dimension values N in H that participate in the "non-covered" mappings. For each value in N, line 8 inserts a corresponding marked value into P; these marked values represent the N values in P. The marked values in P are then linked to the original values in H (9) and C (10). Line (12) contains a recursive call to the algorithm P, thus fixing mappings higher up in the hierarchy. The algorithm terminates when it reaches the $\top$ category, which has no predecessors.

(1) **procedure** MakeCovering(C)
(2) **for** each $P \in Pred(C)$ **do**
(3) **begin**
(4) **for** each $H \in Pred(C)$ where $Type(H) > Type(P)$ **do**
(5) **begin**
(6) $L \leftarrow R_{C,H} \setminus \Pi_{C,H}(R_{C,P} \bowtie R_{P,H})$
(7) $N \leftarrow \Pi_H(L)$
(8) $P \leftarrow P \cup \{Mark(h) \mid h \in N\}$
(9) $R_{P,H} \leftarrow R_{P,H} \cup \{(Mark(h), h) \mid h \in N\}$
(10) $R_{C,P} \leftarrow R_{C,P} \cup \{(c, Mark(h)) \mid (c, h) \in L\}$
(11) **end**
(12) MakeCovering(P)
(13) **end**
(14) **end**

All steps in the algorithm are expressed using standard relational algebra operators. The *general* worst-case complexity of join is $\mathcal{O}(n^2)$, where n is the size of the input. However, because the input to the algorithm are hierarchy definitions, the

complexity of the join in the algorithm will only be $\mathcal{O}(n \log n)$. Thus, all the operators used can be evaluated in time $\mathcal{O}(n \log n)$, where n is the size of the input. The $Mark$ operation can be performed in $\mathcal{O}(1)$ time. The inner loop of the algorithm is evaluated at most once for each link between categories, i.e., at most $k^2/2$ times, where k is the number af categories (if all categories are directly linked to all others). Thus, the overall big-$\mathcal{O}$ complexity of the algorithm is $\mathcal{O}(k^2 n \log n)$, where k is the number of categories and n is the size of the largest participating R_{C_1,C_2} relation. The worst-case complexity will not apply very often; in most cases, the inner loop will only be evaluated at most k times.

The algorithm inserts new values into the P category to ensure that the mappings from P to higher categories are summarizable, i.e., that pre-aggregated results for P can be directly combined into higher-level aggregate results. The new values in P mean that the cost of materializing aggregate results for P is higher for the transformed hierarchy than for the original. However, if the hierarchy was not transformed to achieve summarizability, we would have to materialize aggregates for G, and perhaps also for higher level categories. At most one new value is inserted into P for every value in G, meaning that the extra cost of materializing results for P is never greater than the cost of the (otherwise necessary) materialization of results for G. This is a very unlikely worst-case scenario—in the most common cases, the extra cost for P will be much lower than the the cost of materializing results for G, and the savings will be even greater because materialization of results for higher-level categories may also be avoided.

The correctness argument for the algorithm has two aspects. First, the mappings in the hierarchy should be *covering* upon termination. Second, the algorithm should only make transformations that are semantically correct, i.e., we should get the same results when computing results with the new hierarchy as with the old. The correctness follows from Theorem 3 and 4, below. As new values are inserted in the P category, we will get aggregate values for both the new and the original values when "grouping" by P. Results for the original values will be the same as before, so the original result set is a *subset* of the result set obtained with the transformed hierarchy.

Theorem 3 Algorithm MakeCovering terminates and the hierarchy for the resulting dimension D' is covering.

Proof: By induction in the height of the dimension lattice. *Base*: The height is 0, making the statement trivially true. *Induction Step*: We assume the statement is true for dimension lattices of height n, and consider lattices of height $n + 1$. For termination, we note that there is a finite number of (P, H) pairs, all operations in the inner loop terminate, and the algorithm is called recursively on P, which is the root of a lattice of height n. For the covering property, we note that the insertion of intermediate, marked values into P means that the mapping from P to H is covering w.r.t. C. By the induction hypothesis, the mappings higher in the hierarchy are fixed by the recursive call of the algorithm.

Theorem 4 Given dimensions D and D' such that D' is the result of running MakeCovering on D, an aggregate result obtained using D is a subset of the result obtained using D'.

Proof: Follows easily from Lemma 1, next, as the inserted values are "internal" in the hierarchy.

Lemma 1 For the dimension $D' = (C', \sqsubseteq')$ resulting from applying algorithm MakeCovering to dimension $D = (C, \sqsubseteq)$, the following holds: $\forall e_1, e_2 \in D\ (e_1 \sqsubseteq' e_2 \Leftrightarrow e_1 \sqsubseteq e_2)$ (there is a path between any two original dimension values in the new dimension hierarchy iff there was a path between them in the original hierarchy).

Proof: By induction in the height of the dimension lattice. *Base*: The height is 0 making the statement trivially true. *Induction Step*: We assume the statement is true for dimension lattices of height n, and consider lattices of height $n+1$. Examing the inner loop, we see that the insertion of intermediate values into P, and the linking of values in C and H to these, only links values in C and H that were linked before. No links or values are destroyed by the inner loop. Thus, the statement is true for the links from C to P, and from C to H. By the induction hypothesis, the statement holds true for the transformations made by the recursive call on P.

We see that the original values in the hierarchy are still linked to exactly the same original values as before, as stated by Lemma 1, although new values might have been inserted in-between the original values. Thus, when evaluating a query using the transformed hierarchy, the results for the original values will be the same as when using the original hierarchy.

Assuming only the original result set is desired, results for the new values must be excluded, which is easy to accomplish. The new, "internal" values are marked with "mark=internal", whereas the original values have "mark=original". In order to exclude the new, internal values from the result set, the equivalent of an SQL HAVING clause condition of "mark=original" is introduced into the original query.

4.4.2 Non-Onto Hierarchies

The second algorithm renders all mappings in hierarchies onto, i.e., all dimension values in non-bottom categories have children. This is ensured by inserting placeholder values in lower categories to represent the childless values. These new values are marked with the original values, making it possible to map facts to the new placeholder values instead of to the original values. This makes it possible to only map facts to the bottom category.

Example 38 In the Diagnosis dimension, the "Lung cancer" diagnosis family (ID = 14) has no children. When the algorithm reaches the Diagnosis Family category, it inserts a placeholder value (**L14**) into the Low-level Diagnosis category, representing the "Lung cancer" diagnosis, and links it to the original value. Facts mapped to the "Lung cancer" value may then instead be mapped to the new placeholder value, ensuring that facts are mapped only to the Low-level Diagnosis Category. A graphical illustration of the transformation is seen in Figure 4.4. The bold-faced **L14** value is the new value inserted, and the thick line between 14 and **L14** is the new link inserted.

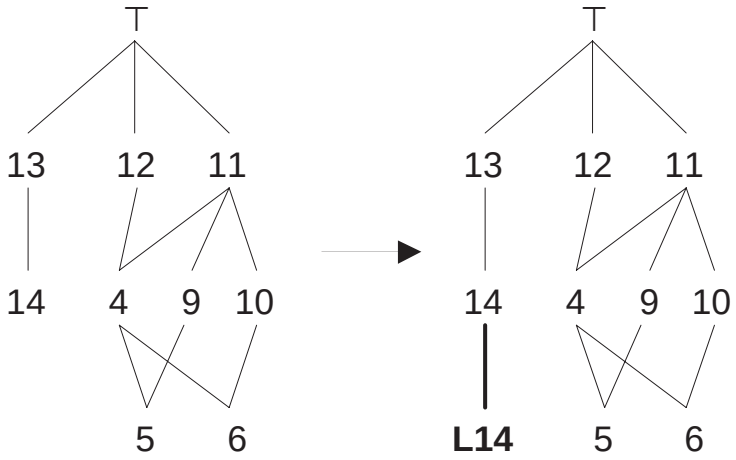

Figure 4.4: Transformations by the MakeOnto Algorithm

In the algorithm below, P is a *parent* category, C is a *child* category, and N holds the parent values with *no* children. The algorithm works as follows. Given a category P (initially the $\top$ category) in line (1), the algorithm goes through all categories C that are (immediate) descendants of P (2). For each C, line (4) finds the values N in P that have *no* children in C, by "subtracting" the values *with* children in C from the values in P. For each "childless" value in N, lines (5) and (6), respectively, insert into C a placeholder value marked with the parent value, and links the new value to the original. MakeOnto is then called recursively on C (7). The algorithms terminates when it reaches the $\bot$ category, which has no descendants.

(1) **procedure** MakeOnto(P)
(2) **for** each $C \in Desc(P)$ **do**
(3) **begin**
(4) $N \leftarrow P \setminus \Pi_P(R_{C,P})$
(5) $C \leftarrow C \cup \{Mark(p) \mid p \in N\}$
(6) $R_{C,P} \leftarrow R_{C,P} \cup \{(Mark(p), p) \mid p \in N\}$
(7) MakeOnto(C)
(8) **end**
(9) **end**

Following the reasoning in Section 4.4.1, we find that the overall big-$\mathcal{O}$ complexity is equal to $\mathcal{O}(k^2 n \log n)$, where k is the number of categories and n is the size of the largest participating R_{C_1,C_2} relation. However, the complexity will only be $\mathcal{O}(kn \log n)$ for the most common cases.

The MakeOnto algorithm inserts new values into C to ensure that the mapping from C to P is summarizable. Again, this means that the cost of materializing results for C will be higher for the transformed hierarchy than for the original. However, if the new values were not inserted, we would have to materialize results for P, and perhaps also higher categories, as well as C. At most one value is inserted in C for

every value in P, meaning that the extra cost for C is never greater than the cost of materializing results for P. As before, this is a very unrealistic scenario, as it corresponds to the case where *no* values in P have children in C. In most cases, the extra cost for C will be a small percentage of the cost of materializing results for P, and the potential savings will be even greater, because pre-aggregation for higher-level categories may be avoided.

As before, the correctness argument for the algorithm has two aspects. First, the mappings in the hierarchy should be *onto* upon termination. Second, the algorithm should only make transformations that are semantically correct. The correctness follows from Theorems 5 and 6, below. Again, the result set for the original values obtained using the original hierarchy will be a subset of the result set obtained using the transformed hierarchy. The results for the new values can be excluded from the result set by adding a HAVING clause condition.

Theorem 5 Algorithm MakeOnto terminates and the hierarchy for the resulting dimension D' is onto.

Proof: By induction in the height of the dimension lattice. *Base*: The height is 0, making the statement trivially true. *Induction Step*: We assume the statement is true for dimension lattices of height n, then consider lattices of height $n + 1$. For termination, we note that there is a finite number of descendants C for each P, that all operations in the loop terminate, and that the algorithm is called recursively on C, which is the top element in a lattice of height n. For the onto property, we note that the insertion of placeholder values into C makes the mapping from C to P onto. By the induction hypothesis, the mappings further down in the lattice are handled by the recursive call.

Theorem 6 Given dimensions D and D' such that D' is the result of applying the MakeOnto algorithm to D, an aggregate result obtained using D is a subset of the result obtained using D'.

Proof: Follows easily from the observation that "childless" dimension values are linked to new, placeholder values in lower categories in one-to-one relationships, meaning that data for childless values will still be counted exactly once in aggregate computations that use the new dimension.

4.4.3 Non-Strict Hierarchies

The third algorithm renders mappings in hierarchies strict, meaning that problems of "double-counting" will not occur. Non-strict hierarchies occur when one dimension value has several parent values.

The basic idea is to "fuse" a set of parent values into one "fused" value, then link the child value to this new value instead. The fused values are inserted into a new category in-between the child and parent categories. Data for the new fused category may safely be re-used for computation of higher-level aggregate results, as the hierarchy leading up to the new category is strict.

The fused value is also linked to the relevant parent values. This mapping is by nature non-strict, but this non-strictness is not a problem, as we prevent aggregate results for the parent category from being re-used higher up in the hierarchy. This is done by "unlinking" the parent category from its predecessor categories.

The categories higher up are instead reached through the fused category. This means that we can still get results for any original category, while being able to apply practical pre-aggregation throughout the hierarchy. In pre-aggregation terms, the "unlinking" of the parent categories means that we must prevent results for including this category from being materialized—only "safe" categories may be materialized. This should be given as a constraint to the pre-aggregation system that chooses which levels of aggregation to materialize.

We note that the algorithm does not introduce more *levels* in the hierarchy, only more categories, and that the number of "safe" categories in the result is the same as the number of original categories. This means that the complexity of the task of selecting the optimal aggregation levels to materialize is unaffected by the algorithm.

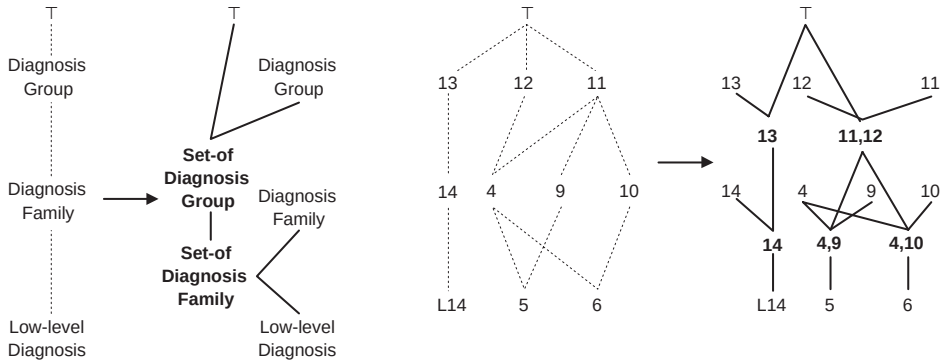

Figure 4.5: Schema and Value Transformations by the MakeStrict Algorithm.

Example 39 The result of running the algorithm on the Diagnosis dimension is seen in Figure 4.5. Because of the non-strictness in the mapping from Low-level Diagnosis to Diagnosis Family, and from Diagnosis Family to Diagnosis Group, two new category types and the corresponding categories are introduced. The third picture indicates the argument to the algorithm; and, in addition, its dotted lines indicate the links deleted by the algorithm. The fourth picture gives the result of applying the algorithm; here, the bold-face values and thick lines indicate the values and links inserted by the algorithm.

In the first call of the algorithm the three Low-level Diagnosis values—"(low-level) Lung cancer" (L14); "Insulin dependent diabetes during pregnancy" (5); and "Non insulin dependent diabetes during pregnancy" (6)—are linked to the three new fused values—"(low-level) Lung cancer" (**14**); "Diabetes during pregnancy, Insulin dependent diabetes" (**4**, **9**); and "Diabetes during pregnancy, Non insulin dependent diabetes" (**4**, **10**)—and these are in turn linked to "Lung Cancer" (14); "Diabetes during pregnancy" (4); "Insulin dependent diabetes" (9); and "Non insulin dependent

diabetes" (10). The these latter four values in the Diagnosis Family category are un-linked from their parents, as the Diagnosis Family category is "unsafe."

When called recursively on the Set-of Diagnosis Family category, the algorithm creates the new fused values "Cancer" (**13**) and "Diabetes, Other pregnancy related diseases" (**11**, **12**) in the Set-of Diagnosis Group category. These new values are linked to the values "Cancer" (13), "Diabetes" (11), and "Other pregnancy related diseases" (12) in the Diagnosis Group category, and to the $\top$ value; and the values in the Diagnosis Group category are un-linked from their parents. Note the importance of having a $\top$ value: the values not linked to $\top$ are exactly the unsafe values, for which aggregate results should not be re-used.

The algorithm assumes that all the paths in the dimension hierarchy have equal length, i.e., all direct links are from children to their immediate parents. This is ensured by the MakeCovering and MakeOnto algorithms. In the algorithm below, C is a *child* category, P is a *parent* category, G is a *grandparent* category, N is the *new* category introduced to hold the "fused" values, and $\bowtie$ denotes natural join.

(1) **procedure** MakeStrict (C)
(2) **for** each $P \in Pred(C)$ **do**
(3) **begin**
(4) **if** $(\exists e_1 \in C\ (\exists e_2, e_3 \in P\ (e_1 \sqsubseteq e_2 \wedge e_1 \sqsubseteq e_3 \wedge e_2 \neq e_3)))$
$\wedge Pred(P) \neq \emptyset$ **then**
(5) **begin**
(6) $N \leftarrow CreateCategory(2^P)$
(7) $R_{C,N} \leftarrow \{(e_1, Fuse(\{e_2 \mid (e_1, e_2) \in R_{C,P}\}))\}$
(8) $N \leftarrow \Pi_N(R_{C,N})$
(9) $R_{N,P} \leftarrow \{(e_1, e_2) \mid e_1 \in N \wedge e_2 \in Unfuse(e_1)\}$
(10) $Pred(C) \leftarrow Pred(C) \cup \{N\} \setminus \{P\}$
(11) $Pred(N) \leftarrow \{P\}$
(12) **for** each $G \in Pred(P)$ **do**
(13) **begin**
(14) $R_{N,G} \leftarrow \Pi_{N,G}(R_{N,P} \bowtie R_{P,G})$
(15) $Pred(N) \leftarrow Pred(N) \cup \{G\}$
(16) $Pred(P) \leftarrow Pred(P) \setminus \{G\}$
(17) **end**
(18) MakeStrict(N)
(19) **end**
(20) **else** MakeStrict(P)
(21) **end**
(22) **end**

The algorithm takes a category C (initially the $\bot$ category) as input. I then goes through the set of immediate parent categories P of C (line (2)). Line (4) tests if there is non-strictness in the mapping from C to P *and* if P has any parents (4). If this test fails, there is no problem as aggregate results for P can either be safely re-used or are guaranteed not be re-used; and the algorithm in then invoked recursively, in line (20).

If the test succeeds, the algorithm creates a new fused category. First, a new, empty category N with domain 2^P is created in line (6). The values inserted into this category represent *sets* of values of P. For example. the value "**1**, **2**" represents the set consisting of precisely $1, 2$. Values in C are then linked to to new, fused values, representing their particular *combination* of parents in P (7). The new values are constructed using a Fuse function, that creates a distinct value for each combination of P values and stores the corresponding P values along with it.

The resulting links are used in line (8) to insert the fused values into their category N, and an "Unfuse" function, mapping fused values from N into the corresponding P values, is used in line (9) to map the values in N to those in P. In line (10), N is included in, and P is excluded from, the sets of predecessors of C. The set of predecessors of N is set to P in line (11), meaning that the new category N resides in-between C and P in the hierarchy.

For each grandparent category G, the algorithm links values in N to values in G, in line (14), includes G in the predecessors of N, in line (15), and excludes G from the predecessors of P, in line (16), thereby also deleting the links from P to G from the hierarchy. The exclusion of the G categories from the predecessors of P means that aggregate results for P will not be re-used to compute results for the G categories.

In the end, the algorithm is called recursively on the new category, N. Note that the test for $Pred(P) \neq \emptyset$ in line (4) ensures that the mapping from N to P will not be altered, as P now has *no* predecessors.

Following the reasoning in Section 4.4.1, we find that the overall big-$\mathcal{O}$ complexity is equal to $\mathcal{O}(pnk \log n \log k)$, where p is the number of immediate parent and children categories in the dimension type lattice, n is the size of the largest mapping in the hierarchy, and k is the maximum number of values fused together. For most realistic scenarios, p and k are small constants, yielding a low $\mathcal{O}(n \log n)$ complexity for the algorithm.

The MakeStrict algorithm constructs a new category N and insert fused values in N to achieve summarizability for the mapping from N to P, and from N to G. The algorithm only inserts the fused values for the combinations that are actually present in the mapping from C to P. This means that the cost of materializing results for N is never higher than the cost of materializing results for C. This is a worst-case scenario, for the most common cases the cost of materializing results for N will be be close to the cost of materializing results for P. However, without the introduction of N, we would have to materialize results not only for P, but also for G and *all* higher-level categories. Thus, the potential savings in materialization costs are very high indeed.

Considering correctness, the mappings in the hierarchy should be *strict* upon termination, and the algorithm should only make transformations that are semantically correct. More specifically, it is acceptable that some mappings be non-strict, namely the ones from the new, fused categories to the unsafe parent categories. This is so because unsafe categories do *not* have predecessors in the resulting hierarchy, meaning that aggregate results for these categories will not be re-used.

The correctness follows from Theorems 7 and 8, below. When evaluating queries we get the same result for original values as when evaluating on the old hierarchy.

The values that are deleted by the algorithm were not linked to any facts, meaning that these values did not contribute to the results in the original hierarchy. As all the new values are inserted into new categories that are unknown to the user, the aggregate result obtained will be the same for the original and transformed hierarchy. Thus, we do not need to modify the original query.

Theorem 7 Let D' be the dimension resulting from applying algorithm MakeStrict on dimension D. Then the following hold: Algorithm MakeStrict terminates and the hierarchy for the dimension D'', obtained by removing unsafe categories from D', is strict.

Proof: By induction in the height of the dimension lattice. *Base*: The height is 0, making the statement trivially true. *Induction Step*: Assuming that the statement is true for lattices of height n, lattices of height $n + 1$ are considered. All steps in the algorithm terminate, and the algorithm is called recursively on either P (in the strict case) or N (in the non-strict case), both of which are the root of a lattice of height n, thus guaranteeing termination.

For the strictness property, there are three cases. If the mapping from C to P is already strict, this mapping is not changed, and by the induction hypothesis, the statement holds for the recursive call on P. If the mapping from C to P is non-strict, but P does not have any parents, strictness is ensured, as P is excluded from D''. If the mapping is non-strict and P has parents, the resulting mapping from C to N is strict. By the induction hypothesis, the statement holds true for the recursive call on N, as the introduction of N has not increased the height of the lattice.

Theorem 8 Given dimensions D and D' such that D' is the result of applying the MakeStrict algorithm to D, an aggregate obtained using D' is the same as that obtained using D.

Proof: Follows from Lemma 2, as all facts are mapped to values in the $\perp$ category, which is a safe category. Thus, there will be a path from a fact f to an original dimension value e iff there was one in the original hierarchy, meaning that aggregate results computed using the original and the new hierarchy will be same.

Lemma 2 For the dimension $D' = (C', \sqsubseteq')$ resulting from applying the MakeStrict algorithm to dimension $D = (C, \sqsubseteq)$, the following holds. $\forall e_1, e_2 \in D\,(e_1 \in C_1 \wedge \mathit{Safe}(C_1) \wedge e_1 \sqsubseteq' e_2 \Leftrightarrow e_1 \sqsubseteq e_2)$ (there is a path between an original dimension value in a safe category and any other original dimension value in the new dimension hierarchy iff there was a path between them in the original hierarchy).

Proof: By induction in the height of the dimension lattice. *Base*: The height of the lattice is 0, making the statement trivially true. *Induction Step*: If either the mapping from C to P is strict, or P does not have any parents, the algorithm does not change the mappings, and by the indiction hypothesis, the statement is true for the recursive call on P. Otherwise, we observe that the creation of fused values in N, and the linking of C, P, and G values to these, only links *exactly* the values in C and P, or C and G, that were linked before. Because P is not safe, the links from P to G may be deleted. By the induction hypothesis, the statement is true for the recursive call on N.

4.5 Fact-Dimension Transformation Techniques

This section explains how the set of algorithms from Section 4.4 may also be applied to the relationships between facts and dimensions, thus providing a basis for enabling practical pre-aggregation on concrete MOs that include fact data.

The basic idea is to view the set of facts F as the bottom granularity in the lattice. The input to the algorithms then consists of the facts, F, the $R_{F,C}$ tables, describing the mappings from facts to dimension values, and the C and R_{C_1,C_2} tables, describing the dimension categories and the mappings between them.

Only the covering and strictness properties are considered because for the fact-dimension relationships, a mapping between facts and dimension values that is *into* means that not all dimension values in the bottom category have associated facts, which does not affect summarizability. As before, we first apply the MakeCovering algorithm, then the MakeStrict algorithm.

The computational complexity of the algorithms will now be dominated by the size, n, of the mapping between facts and dimension values, i.e., the complexity will be $\mathcal{O}(n \log n)$ if we assume the height of the lattice and the maximum number of values fused together to be small constants. This means that the algorithms can be applied to even very large databases.

4.5.1 Mixed Granularity Mappings

The first case to consider is the one where some of the mappings are non-covering w.r.t. the facts, meaning that not all facts can be reached through these mappings and thus resulting in these facts not being accounted for in aggregate computations. This occurs when some facts are mapped *directly* to dimension values in categories higher than the $\perp$ category, i.e., the facts are mapped to values of *mixed* granularities.

We use the MakeCovering algorithm to make the mappings covering, initially calling it on F, which is now the bottom of the lattice. The algorithm makes the mappings covering w.r.t. the facts by inserting new marked values, representing the parent values, in the intermediate categories, and by linking the facts to the new values instead of the parent values. As in Section 4.4.1, the marked values keep information about their original values, so that when new fact-dimension mappings are added, the links that are supposed to go *directly* to the original parent values now instead can be set to go to the marked value in the $\perp$ category.

Example 40 In the case study, the mapping between Patients and Diagnoses is of mixed granularity: "John Doe" (1) and "Jane Doe" are both mapped to the Diagnosis Family, "Insulin dependent diabetes" (9), "Jane Doe" is additionally mapped to the Low-level Diagnosis, "Insulin dependent diabetes during pregnancy" (5), and "Jim Doe" is mapped to "Diabetes" (11), a Diagnosis Group.

In the first call of the algorithm, two new Low-level Diagnoses are inserted: "**L9**," representing "Insulin dependent diabetes," and "**L11**," representing "Diabetes"; and the facts are mapped to these instead of the original values. In the recursive call on Low-level Diagnosis, an "**F11**" value representing "Diabetes" at the Diagnosis Family level is inserted between "Diabetes" and value "**L11**."

The transformations are illustrated in Figure 4.6, where dotted lines indicate links that are deleted by the algorithm and bold-face value and thick lines indicate dimension values and links inserted by the algorithm.

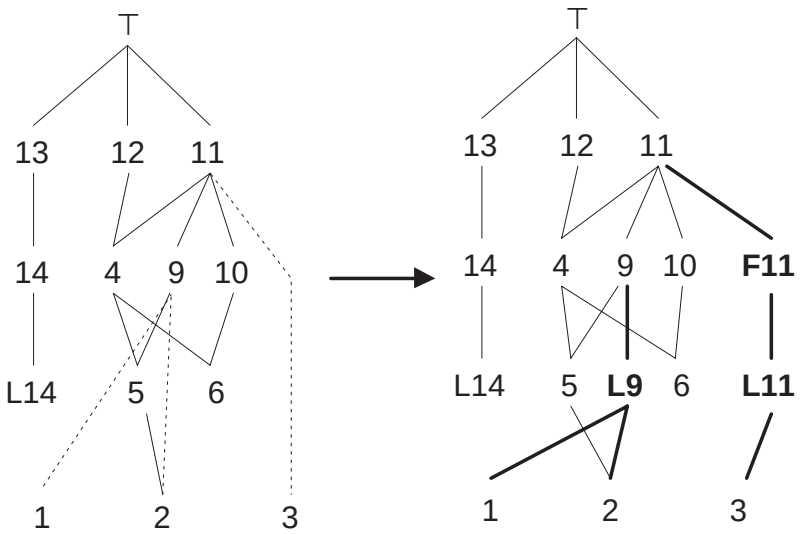

Figure 4.6: Transformations for Varying Granularities.

4.5.2 Many-To-Many Relationships

The second case occurs when relationships between facts and dimension values are many-to-many. This means that the hierarchy, with the facts as the bottom category, is non-strict, leading to possible double-counting of facts. It is enough to make the hierarchy partly strict, as described in Section 4.4.3. The MakeStrict algorithm is initially called on F, which is now the bottom of the hierarchy lattice. Because the MakeCovering algorithm has already been applied, all paths from facts to the $\top$ value have equal length, as required by the MakeStrict algorithm.

Some dimension values have no facts mapped to them, leading to an interesting side effect of the algorithm. When the algorithm fuses values and places the fused values in-between the original values, it also deletes the child-to-parent and parent-to-grandparent links. The fact-less dimension values are then left disconnected from the rest of the hierarchy, with no links to other values.

These fact-less dimension values do not contribute to any aggregate computations and are thus superfluous. To minimize the dimensions, an "Delete-unconnected" algorithm that deletes the fact-less dimension values by traversing the hierarchy starting at the facts is invoked in a postprocessing step. For a hierarchy of height k, this can be done in time $\mathcal{O}(kn \log n)$, where n is the size of the mapping between facts and dimensions. Thus, the overall computational complexity is not altered.

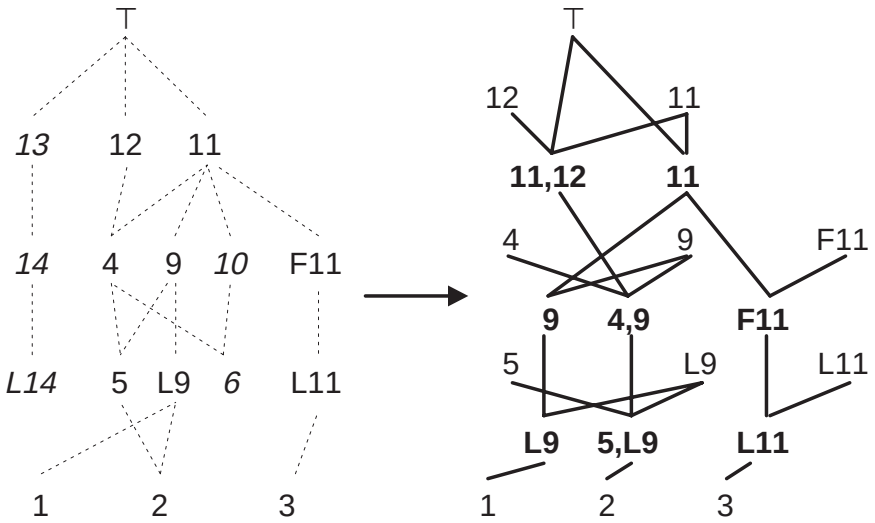

Figure 4.7: Transformations for Many-to-many Fact-Dimension Relationships

Example 41 The relationship between patients and diagnoses is many-to-many. In Example 40, the MO was transformed so that all mappings were covering, as seen in Figure 4.6; algorithm MakeStrict is applied to this MO. The final result of the application of the algorithm is seen to the right in Figure 4.7. Values in italics, e.g., *L14*, and dotted lines indicate deleted values and links. Bold-face values and thick lines denote values and links inserted by the algorithm.

Three new categories are introduced: "Set-of Low-level Diagnosis," "Set-of Diagnosis Family," and "Set-of Diagnosis Group," as non-strictness occurs at all levels. Fused values are inserted into these fused categories. For example, values "(low-level) Lung Cancer" (**L14**), "Insulin dependent diabetes during pregnancy, (low-level) Insulin dependent diabetes" (**5**, **L9**), and "(low-level) Insulin dependent diabetes" (**L9**) are inserted into the "Set-of Low-level Diagnosis" category; and the original values are linked to the new values.

Values "(low-level) Lung cancer" (**L14**), "Lung cancer" (14), "Cancer" (13), "Non insulin dependent diabetes during pregnancy" (6), and "Non insulin dependent diabetes" (10) do not characterize any facts and are deleted by "Delete-unconnected."

4.6 Architectural Context

The overall idea presented in this chapter is to take un-normalized MOs and transform them into normalized MOs that are well supported by the practical pre-aggregation techniques available in current OLAP systems. Queries are then evaluated on the transformed MOs. However, we still want the users to see only the original MOs, as they reflect the users' understanding of the domain. This prompts the need for means of handling both the original and the transformed MOs. This section explores this coexistence.

A current trend in commercial OLAP technology is the separation of the front-end presentation layer from the back-end database server. Modern OLAP applications consist of an OLAP client that handles the user interface and an OLAP server that manages the data and processes queries. The client communicates with the server using a standardized application programming interface (API), e.g., Microsoft's OLE DB for OLAP [78] or the OLAP Council's MDAPI [85]. The architecture of such a system is given to the left in Figure 4.8.

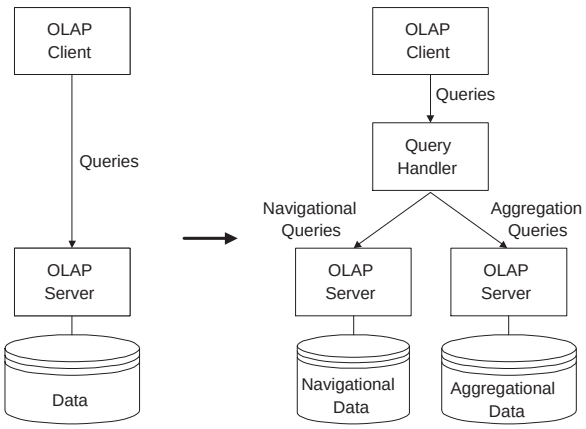

Figure 4.8: Architecture of Integration

This separation of client and server facilitates our desire to have the user see the original MO while queries are evaluated against the transformed MO. Studies have shown that queries on a data warehouse consist of 80% *navigational* queries that explore the dimension hierarchies and 20% *aggregation* queries that summarize the data at various levels of detail [64]. Examples of navigational and aggregation queries are "Show me the Low-Level Diagnoses contained in the Insulin-Dependent Diabetes Diagnosis Family" and "Show me the count of patients, grouped by Diagnosis Family," respectively. The navigational queries must be performed on the *original* MO, while the aggregation queries must be performed on the *transformed* MO. This is achieved by introducing an extra "Query Handler" component between the client and the server. The OLAP client sends a query to the query handler, the primary task of which is to determine whether the query is a navigational query (internal to a dimension) or an aggregation query (involving the facts). Navigational queries are passed to one OLAP server that handles the original (navigational) data, while aggregation queries are passed to another OLAP server that manages the transformed (aggregation) data. This extended system architecture is seen to the right in Figure 4.8.

The OLAP server for navigation data needs to support dimension hierarchies which have non-summarizable properties, a requirement not yet supported by many commercial systems today. However, relational OLAP systems using snow-flake schemas [64] are able to support this type of hierarchies, as are some other OLAP systems, e.g., Hyperion (Arbor) Essbase [53]. If the OLAP system available does not have sufficiently flexible hierarchy support, one solution is to build a special-purpose OLAP server that conforms to the given API. This task is not as daunting as it may

seem at first because only *navigational* queries need to be supported, meaning that multidimensional queries can be translated into simple SQL "lookup" queries.

We note that the only data needed to answer navigational queries is the hierarchy definitions. Thus, we only need to store the fact data (facts and fact-dimension relations, in our model) once, in the aggregational data, meaning that the overall storage requirement is only slightly larger than storing just the aggregational data. Navigational queries are evaluated on the original hierarchy definitions and do not need to be re-written by the query handler.

As described in Section 4.4, aggregation queries need to be re-written slightly by adding an extra HAVING clause condition to exclude results for the new values inserted by the transformation algorithms. This can easily be done automatically by the query handler, giving total transparency for the user. Even though the added HAVING clause conditions are only necessary for the covering and onto transformations, they can also be applied to hierarchies transformed to achieve strictness; this has no effect, but simplifies the query rewriting.

4.7 Conclusion and Future Work

Motivated by the increasing use of OLAP systems in many different applications, including in business and health care, this chapter provides transformation techniques for multidimensional databases that leverage the existing, performance-enhancing technique, known as practical, or partial or semi-eager, preaggregation, by making this technique relevant to a much wider range of real-world applications.

Current pre-aggregation techniques assume that the dimensional structures are *summarizable*. Specifically, the mappings in dimension hierarchies must be *onto*, *covering*, and *strict*; the relationships between facts and dimensions must be many-to-one, and the facts must always be mapped to the lowest categories in dimensions. The chapter presents novel transformation techniques that render dimensions with hierarchies that are non-onto, non-covering, and non-strict summarizable. The transformations have practically low computational complexity, they may be implemented using standard relational database technology, and the chapter also describes how to integrate the transformed hierarchies in current OLAP systems, transparently to the user.

The chapter also describes how to apply the transformations to the cases of non-summarizable relationships between facts and dimensions, which also occur often in real-world applications. Finally, it is shown how to modify the algorithms to incrementally maintain the transformed hierarchies when the underlying data is modified.

Several directions for future research appear promising. The current techniques render the entire dimension hierarchies summarizable; extending the techniques to consider only the parts that have been selected for preaggregation appears attractive and possible. Another direction is to take into account the different types of aggregate functions to be applied, leading to local relaxation of the summarizability requirement. For example, *max* and *min* are insensitive to duplicate values, thus relaxing summarizability.

4.8 Incremental Computation

When dimension hierarchies or fact data are updated, the transformed hierarchies must be updated correspondingly. One solution is to recompute the hierarchies using the new data. This straightforward solution is attractive when updating small dimension hierarchies that only change infrequently, or when large bulks of updates are processed. However, for massive hierarchies and frequent updates, and for updates of small parts of the hierarchies in general, it is desirable if the algorithms need only consider the *changed* parts of data, which will only be a small fraction of the total data volume. This section briefly describes how to incrementalize the algorithms.

In addition to modifying the transformed hierarchies, it is also necessary to update the actual pre-aggregated data when the underlying base data is modified. The modified hierarchies resulting from the algorithms given in this section differ only locally from the argument hierarchies. This means that the cost of updating the pre-aggregated data will not be greatly affected by the hierarchy transformations.

In the incremental algorithms, updates are modeled as deletions followed by insertions, so we consider only the latter two modification operations. We use prefix Δ_i to denote inserted values, Δ_d to denote deleted values, and Δ to denote all modifications. For example, $\Delta_i C$ denotes the values inserted into C. The category and links tables in the algorithms refer to the states *after* modifications; and when a hierarchy value is deleted, all links to that value are also assumed to be deleted in the same set of modifications. Below, we only describe how to change the algorithms to fix "problematic" hierarchy modifications, i.e., modifications that may cause the hierarchies to become non-covering, non-onto, or non-strict. In a complete incremental solution, the incremental algorithms need to be complemented by an algorithm (rather trivial) that applies the non-problematic modifications to the modified hierarhies.

	Insrt	Dlt
C	yes	no
P	no	yes
H	no	yes
$R_{C,H}$	yes	no
$R_{C,P}$	no	yes
$R_{P,H}$	no	yes

Covering

	Insrt	Dlt
C	no	yes
P	yes	no
$R_{C,P}$	no	yes

Onto

	Insrt	Dlt
C	no	yes
P	no	yes
G	no	yes
$R_{C,P}$	yes	yes
$R_{P,G}$	yes	yes

Strict

Table 4.2: Effects of Insertions and Deletions on the Covering, Onto, and Strictness Properties

4.8.1 Covering Hierarchies

Modifications may render *covering* hierarchies non-covering in several ways. The the left-most table in Table 4.2, named "Covering" and discussed next, indicates whether an insertion ("Insert") or a deletion ("Delete") on the different parts of the input to MakeCovering may render the modified hierarchy non-covering.

Problems may arise if links are inserted into $R_{C,H}$ that are not covered by insertions into $R_{C,P}$ and $R_{P,H}$, or if links are deleted in $R_{C,P}$ or $R_{P,H}$, but the corresponding C-to-H links are not deleted in $R_{C,H}$. If values are deleted in P or H, their links will be deleted too, which is handled by the case above. Values cannot be inserted into C without any links, as all values in the original hierarchy must at least be linked to the $\top$ value.

The incremental version of MakeCovering algorithm starts by finding (in line (6)) the links L from C to H that are not covered by the links from C to P and P to H. These links are used as the base for the rest of the transformation. Thus, line (6) of the algorithm becomes the following expression.

$$L \leftarrow \Delta_i R_{C,H} \cup \Pi_{C,H}(\Delta_d R_{C,P} \bowtie R_{P,H}) \cup \Pi_{C,H}(R_{C,P} \bowtie \Delta_d R_{P,H}) \\ \setminus \Pi_{C,H}(\Delta_i R_{C,P} \bowtie \Delta_i R_{P,H}) \setminus \Delta_d R_{C,H}$$

4.8.2 Onto Hierarchies

The effects on the *onto* property of insertions and deletions are outlined in the middle table in Table 4.2. Insertion of values into P, deletion of values in C, and deletion of links in $R_{C,P}$ may cause the hierarchy to become non-onto. The incremental version of the MakeOnto algorithm thus starts by finding (in line (4)) the "childless" values N from P with no children in C. As a result, line (4) of the algorithm becomes the following expression.

$$N \leftarrow \Delta_i P \cup \Pi_P(\Delta_D R_{C,P}) \setminus \Pi_P(\Delta_d P) \setminus \Pi_P(\Delta_i R_{C,P})$$

4.8.3 Strict Hierarchies

The case of maintaining the *strictness* property of hierarchies is more complicated because a new category N is introduced by the algorithm. We assume that all new categories have already been created before the incremental algorithm is used, i.e., if non-strictness is introduced in new parts of the hierarchy, we have to recompute the transformed hierarchy. The introduction of non-strictness requires major restructuring of both the hierarchy and the pre-aggregated data, so this is reasonable.

An overview of the effect on strictness of insertions and deletions in the input to algorithm MakeStrict is given in the right-most table in Table 4.2. If links are inserted into, or deleted from, $R_{C,P}$ or $R_{P,G}$, the links to N for the affected C, P, and G values must be recomputed.

Insertions into, or deletion from, C, P, or G will be accompanied by corresponding link insertions and deletions, so they are handled by the above case. The incremental MakeStrict, given below, works by finding the affected C, P, and G values, then recomputes their links to N and deletes the old links, and finally inserting the new links. As before, it is followed by a step that deletes the disconnected parts of the hierarchy.

(1) **procedure** IncrementalMakeStrict(C)
(2) **for** each $P \in Pred(C)$ such that $Pred(P) \neq \emptyset$ **do**
(3) **begin**
(4) $dC \leftarrow \Pi_C(\Delta R_{C,P})$
(5) $dR_{C,N} \leftarrow \{(c, Fuse(\{p \mid (c,p) \in dC \bowtie R_{C,P}\})\}$
(6) $dN \leftarrow \Pi_N(dR_{C,N})$
(7) $N \leftarrow N \cup dN$
(8) $R_{C,N} \leftarrow R_{C,N} \setminus \{(c,n) \mid c \in dC\} \cup dR_{C,N}$
(9) $dP \leftarrow \Pi_P(\Delta R_{C,P})$
(10) $dR_{N,P} \leftarrow \{(n,p) \mid n \in dN \wedge p \in dP \cap UnFuse(n)\}$
(11) $R_{N,P} \leftarrow R_{N,P} \setminus \{(n,p) \mid p \in dP\} \cup dR_{N,P}$
(12) **for** each $G \in Pred(P)$ **do**
(13) **begin**
(14) $dG \leftarrow \Pi_G(\Delta R_{P,G} \cup (dP \bowtie R_{P,G}))$
(15) $R_{N,G} \leftarrow R_{N,G}\,\{(n,g) \mid g \in dG\}$
$\cup \Pi_{N,G}(R_{N,P} \bowtie R_{P,G} \bowtie dG)$
(16) **end**
(17) IncrementalMakeStrict(N)
(18) **end**
(19) **end**

Chapter 5

Extending OLAP Querying To Object Databases

5.1 Introduction

On-Line Analytical Processing (OLAP) systems have become increasingly popular in many application areas, as they considerably ease the process of analyzing large amounts of enterprise data. Designed specifically with the aim of better supporting the retrieval of higher-level summary information from detail data, these systems offer substantial additional user-friendliness over general database management systems (DBMSs). The special dimensional data models employed in OLAP systems enable visual querying, as well as contribute to enable OLAP systems to offer better performance for aggregate queries than do traditional DBMSs. As another example, most OLAP systems support *automatic aggregation* [103, 70], which means that the system knows which aggregate functions to apply when retrieving different higher-level summaries.

Almost all OLAP systems are based on a *dimensional* view of data, in which measured values, termed facts, are characterized by descriptive values drawn from a number of dimensions; and the values of a dimension are typically organized in a containment-type hierarchy. While the dimensional view of data is particularly well suited for the aggregation queries performed in OLAP analysis, it also limits the abilities of OLAP systems to capture complex relationships in the data. As a result, an OLAP database only captures some of the structure available in the data from which it derives. Furthermore, it is often difficult or impossible to combine data from an OLAP system with data from other sources.

In contrast, object database (ODB) systems excel at capturing and querying general, complex data structures. These systems offer semantically rich data models and query languages that include constructs such as classes, inheritance, complex associations between classes, and path expressions. However, ODB systems do not support aggregate queries well. For example, the complex data structures tend to make it hard to formulate correct queries that aggregate the data in the ODB. Also, ODB systems are optimized to perform more general types of queries, mostly on the detail level, so the performance for aggregate queries is usually not satisfactory.

Federated database systems [115, 51, 52, 29] support the *logical* integration of autonomous database systems, without requiring data to be physically moved and while allowing the individual autonomous database systems to function as before. Federation is a flexible solution that may leverage existing technology and adapt quickly to changing information requirements. In contrast, *physical* integration of data, commonly referred to as the physical (as opposed to logical) data warehousing approach [130]. This approach has its own advantages, perhaps most significantly in terms of performance when combining data from different databases, but it is very difficult to keep the warehouse data up to date. Thus, it is often impossible or impractical to use physical data warehousing, especially if the data sources belong to different organizations. The two approaches are complimentary, in that they are appropriate under different circumstances.

When integrating data from databases based on different data models, the traditional approach has been to map all data into one common data model and federate the (logically) transformed data rather than the original data [115, 51, 29]. In this paper, we adopt an alternative approach and show how to combine data from summary databases (SDBs) and object databases using a federated database approach[1], where data is handled using the most appropriate data model and database technology: SDB systems for summary data and ODB systems for complex, general data. No attempt is made at "shoehorning" the data into one common format, which is unlikely to fit all the data.

Focus is on enabling OLAP-style queries over SDBs to also include data from ODBs without jeopardizing the benefits of OLAP queries. Specifically, aggregation safety remains enforced, meaning that incorrect or meaningless extended queries are avoided. As a first step in demonstrating the capabilities of the system, a prototypical, user-oriented query language for SDBs, termed SumQL, is defined. The concept of a *link*, which enables the connection of SDBs to ODBs in a general and flexible manner, is then integrated into SumQL along with object features, yielding an extended language, termed SumQL++.

With this language as a vehicle, it is shown how the system enables using path expressions for referencing data in SDBs in *selection criteria*. Queries over SDBs may return ODB data along with the aggregate results, i.e., the result of an OLAP query may be *decorated* with object data. Finally, SDB data may be grouped based on ODB data. All extensions are accompanied by formal definitions in terms of SumQL and the underlying object query language (the ODMG data model and OQL query language [13] are used for the ODBs). The paper's contribution is presented in terms of the SumQL and SumQL++ languages, which are defined formally in the paper and concisely capture the relevant concepts, to be self-contained and ensure precision. Other languages such as SQL [], OQL [], and MDX [] may take the place of SumQL++ once enriched with the constructs in SumQL++ that they do not already offer.

A prototype has been built [41] that supports the execution of SumQL++ queries over a federation of autonomous SDBs and ODBs.

[1] Although the paper's contributions are applicable to almost all current OLAP systems, we use the term SDB instead of OLAP DB to emphasize the focus on aggregate queries over summary data.

The arguably most related previous work concerned the system based on the *n*D-SQL language [39]. This system enables the querying of a federation of solely relational data sources, which are treated symmetrically, using *n*D-SQL. In contrast, we extend OLAP-style queries on an identified SBD to object databases with related data. Further, *n*D-SQL supports neither dimension hierarchies nor the aggregation semantics that enable safe aggregation. Other existing middleware offerings such as DataJoiner [54], Cohera [23], and Oracle Gateways [88] exhibit the same limitations, which renders the formulation of distributed OLAP queries cumbersome and errorprone in comparison to this paper's proposal.

More specifically, we believe this paper to be the first to consider the integrated querying of data from independent summary and object databases without prior physical integration, with the objective of giving OLAP users enhanced, aggregation-safe query capabilities. Surveys of OLAP data models and languages [95, 125, 128] indicate that this issue has not been addressed previously. To our knowledge, the paper is also the first to demonstrate a "multi-paradigm" (or "multi-model") federation [6, 50, 51], where one of the data models is a dedicated summary data model. Finally, the paper is the first to investigate how OLAP concepts such as summarizability and aggregation safety are influenced by federation with external data and how they may be preserved to ensure safe query results.

The remainder of the paper is structured as follows. Section 5.2 presents a real-world case study and considers the arguments for why federating summary and object databases is a good idea. Section 5.3 introduces the foundations for the SDBs and ODBs. It describes a prototypical summary data model and its high-level, user-oriented summary query language, SumQL, as well as the central concept of summarizability. It also briefly presents the Object Data Management Group (ODMG) data model and its OQL query language. Section 5.4 describes the notion of link that connects SDBs to ODBs, and Section 5.5 proceeds to describe the federated data model, which incorporates links, and its extended SumQL query language, which enables queries to access information in both SDBs and ODBs. Section 5.6 describes the prototype implementation of a system that implements the concepts and techniques presented. The last section summarizes and offers research directions. Finally, an appendix describes the formal syntax and semantics of SumQL.

5.2 Motivation

In this section, we discuss why it is a good idea to federate summary and object databases and present a real-world case study that is used for illustration throughout the paper.

5.2.1 Reasons for Federation

Many reasons exist for preferring federating SDBs with ODBs, as opposed to physically integrating these. The generic arguments for federation include leveraging existing technology, accessing the most current information, and allowing the autonomous existence of the systems being federated. These arguments also apply in this case, so we concentrate on the advantages specific to summary and object databases.

In many situations, SDBs only contain abstract summary data and do not contain the base data from which the summary data is derived, thus rendering access to external databases necessary to be able to answer certain queries. For example, summary databases provided by the Ministry of Health do not permit access to base data, because the base data is unavailable or considered too sensitive for general disclosure, e.g., diagnosis information. The same situation arises in census databases, where only high-level information is disclosed publicly.

Federating SDBs and ODBs enables a *simple and special-purpose* SDB system. An SDB needs not contain all objects, attributes, and relationships in the base database, but only the elements relevant to summary querying. This is attractive, as capturing all information in the SDB unnecessarily impedes casual use of the SDB system. Indeed, most OLAP systems that implement summary databases do not have the necessary facilities, e.g., category inheritance [69], to support this extra information. The federated approach allows the SDB to contain only the most commonly used information, providing a simple summary-level view of data, while still allowing access to relevant data that resides in the SDB. When SDB data resides in a special-purpose SDB system, we cannot use existing database middleware to access it, leading to a need for technology that enables federations of SDBs and ODBs.

It is possible to obtain *better performance* when performing summary querying in an OLAP-type system rather than in a general-purpose DBMS. The former type of system typically employs specialized, performance enhancing techniques, such as multidimensional storage and pre-aggregation. So even if all data comes from one single (non-SDB) database, it is desirable to perform summary querying in a specialized OLAP system.

Next, it is *easier to formulate summary queries* in an SDB system than in a general (relational or object) DBMS. This is because an SDB query language is designed exclusively for expressing summary queries over categories, taking advantage of, e.g., the automatic aggregation implied by the summary database semantics. Even when extending an SDB language to access object data (as we do in Section 5.5), it is easier to pose summary queries in the extended language than in a general database query language such as OQL or SQL.

An SDB system may support the formulation of summary queries that return *correct, or meaningful, query results*. When building an SDB, the data may be shaped in order to satisfy summarizability conditions [70]. Briefly, a summary query satisfies summarizability conditions if the query result is correct w.r.t. the real world. For example, summarizing the populations over cities to get summaries for states will produce incorrect results if the populations in towns and farms outside cities are not accounted for. As another example, if patients have several diseases, and we summarize over all diseases to get the total number of sick people, we will get the wrong result as some patients are counted more than once. We may enrich an SDB system with information that enables the system to ensure correctness. For example, we may specify that inventory levels should not be added across time [70] or that patient counts for diseases should not be added. In a general-purpose DBMS, no mechanisms for ensuring correct summary results are available.

The federated approach offers additional *flexibility* when the query requirements change. SDBs may be huge, and therefore rebuilding them may be time consuming.

Updates to an SDB, e.g., adding new types of information, may require a total or partial rebuild of the database. Because of the rebuild time, a rebuild of the SDB will most likely be refused (by the IS department) or postponed to the next scheduled rebuild, e.g., once a week or once a month. In contrast, a new link can be added in a matter of minutes, yielding much faster access to newly required information. This allows *rapid prototyping* of OLAP systems. In a relational DB setting, the ability to do this rapid prototyping is one of the key selling points for the Cohera federated DBMS [23].

The above reasoning suggests that in many cases, it is advantageous to logically federate existing OLAP and object databases instead of performing physical integration.

5.2.2 Case Study

The case study concerns data in three different databases, each managed by a separate organization. Each databases serves a different purpose, but the databases contain related data. A graphical illustration of the databases is seen in Figure 5.1.

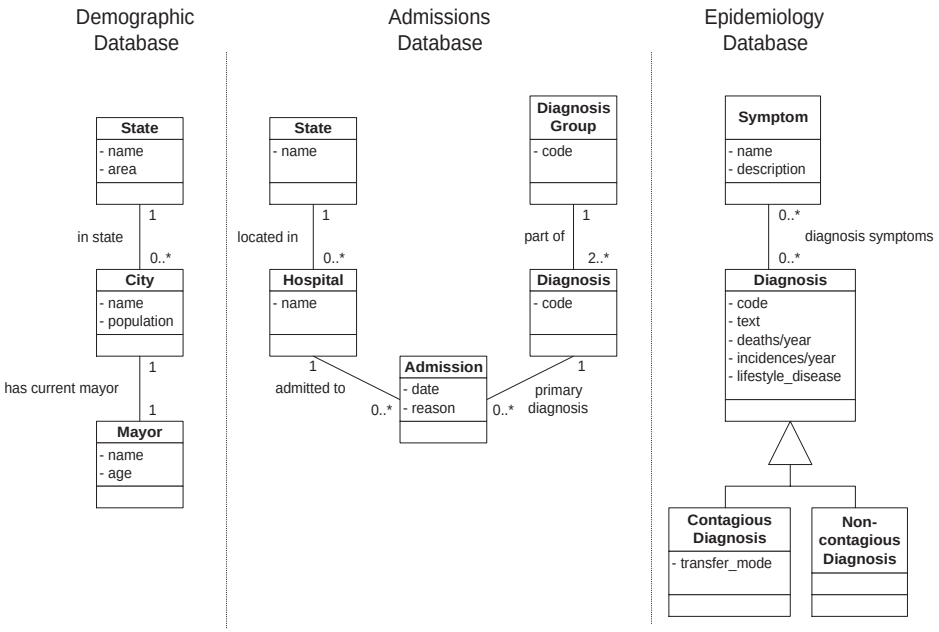

Figure 5.1: UML Schema of Case Study

The databases are modeled using the Unified Modeling Language (UML) [104]. Compound boxes denote classes. The class name is in boldface in the top part of the box, while class attributes are listed in the middle part. The bottom part is reserved for class methods, i.e., dynamic aspects of the class, but since we are only interested in the data, methods are omitted. Associations, i.e., relationships, between classes are represented by lines tagged with an association name. The cardinality of an association is shown by the numbers at the ends of the association line. Either a

single cardinality or a range of cardinalities are specified. A "*" denotes any natural number.

The *demographic database* is maintained by the Department of the Interior and offers central access to demographic data for all cities and states in the country. Data is collected for *states*, for which name and area is stored, and for *cities*, for which name and population is recorded. The database also contains information about the current *mayor* of a city. There are zero or more cities in each state, and each city has exactly one current mayor.

Next, the *admissions database* is maintained by the Department of Health and provides an overview of the admissions patterns for all hospitals nationwide. For an *admission*, the date of admission and the reason for admission, e.g., accident, are recorded. Additionally, we record which *hospital* the patient is admitted to and the primary *diagnosis* that caused the admission. For hospitals, the name and the *state* where the hospital is located are recorded. For diagnoses, we record an alphanumeric code, determined by a standard classification of diseases, e.g., the World Health Organization's International Classification of Diseases (ICD-10) [133]. The classification also determines how the diagnoses are grouped into *diagnosis groups*. Diagnosis groups consist of at least 2 related diagnoses and a diagnosis belongs to exactly one diagnosis group. For diagnosis groups, we record a alphanumeric code, determined by the classification.

The last database is an *epidemiology database* maintained by a medical school for research purposes. Data are collected from hospitals, practicing physicians, and insurance companies to obtain a rich overview of the occurrence of diseases. The database is organized around the *diagnoses* in the standard disease classification also used in the admissions database, but more information is recorded. In addition to the alphanumeric code and an additional descriptive text, the database also records the number of incidences per year, the number of deaths per year, and whether the disease is dependent on the lifestyle of the patient. The Diagnosis class has two subclasses, *Contagious Diagnosis* and *Non-contagious Diagnosis*. For contagious diagnoses, we additionally record the mode of transfer of the disease, e.g., by air. The *symptoms* of diseases are also recorded. For symptoms, we record a name and a description of the symptom.

The three databases were built and are used separately, which explains the differences in their information contents. But, we want to use them together, to include information from the demographic and epidemiology databases in queries against the admissions database. Thus, we need to provide a *logical* integration of the databases.

To obtain some example data, we assume a standard mapping of the UML schemas to relational schemas, i.e., one table per class, and relationships expressed using foreign keys. We also assume the use of surrogate keys, named *ID*, with globally unique values. Subclasses are supported by sharing of IDs with the superclass. For example, the Contagious Diagnosis subclass is represented by a separate table with the ID shared with the Diagnosis table. The tables for the demographic, admissions, and epidemiology databases are shown in Tables 5.1, 5.2, and 5.3, respectively.

ID	Name	Area
0	California	100000
1	Oregon	40000

State Table

ID	Name	Population	StateID	MayorID
10	Berkeley	140000	0	21
11	Portland	500000	1	22
12	Oakland	400000	0	20

City Table

ID	Name	Age
20	Mr. X	45
21	Ms. Y	57
22	Ms. Z	33

Mayor Table

Table 5.1: Data for the Demographic Database

ID	Day	Reason	HospitalID	DiagnosisID
30	05/23/99	Accident	40	50
31	04/12/99	F.P. referral	41	51
32	05/01/98	Specialist referral	41	52

Admission Table

ID	Name	StateID
40	Alta Bates	70
41	Portland General Hospital	71
42	Portland Kaiser	71

Hospital Table

ID	Code	GroupID
50	E10	60
51	E11	60
52	N12	61

Diagnosis Table

ID	Code	Text
60	E1	Diabetes
61	N1	Infections

DiagnosisGroup Table

ID	Name
70	California
71	Oregon

State Table

Table 5.2: Data for the Admissions Database

5.3 Federation Data Models and Query Languages

This section defines a prototypical multidimensional data model and query language used for the SDB component in the federation; and it briefly presents the data model and query language of the federation's ODB component.

The multidimensional model precisely and concisely captures core multidimensional concepts such as categories, dimensions, and automatic aggregation. As part of this, the notion of summarizability is defined. The ODB data model and query language is the ODMG data model and OQL query language.

5.3.1 Summary Data Model

The model has constructs for defining the *schema*, the *instances*, and the *aggregation properties*.

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*.

ID	Code	Text	Deaths	Incidences	Lifestyle
80	E10	Insulin dependent diabetes	50000	900000	Yes
81	E11	Non insulin dependent diabetes	20000	1500000	Yes
82	N12	Pneumonia	100000	1000000	No

Diagnosis Table

ID	TransferMode
82	Air

ContagiousDiagnosis Table

ID	Name	Description
90	Cough	The lungs of the patient ...
91	Acetone Breath	The breath of the patient ...
92	Fever	The temperature of the patient...

Symptom Table

DiagnosisID	SymptomID
80	91
81	91
82	90
82	92

Diagnosis_Symptoms Table

Table 5.3: Data for the Epidemiology Database

Example 42 In the case study we will have *Admissions* as the fact type, and *Diagnosis*, *Place*, *Reason*, and *Time* as the dimension types.

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 each member of the former's extension logically contains several 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 43 Diagnoses are contained in Diagnosis Groups. Thus, the *Diagnosis* dimension type has the following order on its category types: $\bot_{Diagnosis}$ = *Diagnosis* $\sqsubset$ *Diagnosis Group* $\sqsubset \top_{Diagnosis}$. Thus, $Pred(\textit{Diagnosis}) = \{\textit{Diagnosis Group}\}$. Other examples of category types are *Day*, *Month*, and *Year*. Figure 5.2, to be discussed in detail in Example 47, 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 $\mathcal{T} = (\{\mathcal{C}_j\}, \sqsubseteq_\mathcal{T}, \top_\mathcal{T}, \bot_\mathcal{T})$ 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.

The partial order is defined as follows. 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 a partial order $\sqsubseteq_C$ on the categories in a dimension, as given by the partial order $\sqsubseteq_{\mathcal{T}}$ 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, $e \sqsubseteq \top$. Value $\top$ is similar to the *ALL* construct of Gray et al. [40]. 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 44 The *Diagnosis* dimension has the following categories, named by their type. *Diagnosis* = $\{50, 51, 52\}$, *Diagnosis Family* = $\{60, 61\}$, and $\top_{Diagnosis} = \{\top\}$. The values in the sets refer to the *ID* fields in the Diagnosis and Diagnosis Group tables in Table 5.2. The partial order $\sqsubseteq$ is given by the GroupID field in the Diagnosis table. Additionally, the top value $\top$ is greater than, i.e., logically contains, all the other diagnosis values.

Let $C_1, .., C_n$ be categories and T a domain that includes the special value *null*. A *measure* for these categories and this domain is a function $M : C_1 \times .. \times C_n \mapsto T$. We say that M is measure for the set of dimensions $D = \{D_1, .., D_n\}$, if M is a measure for the categories $\perp_{D_1}, .., \perp_{D_n}$. Every measure M has associated with it a *default aggregate function* $f_M : T \times T \mapsto T$. The default aggregate function must be distributive. The null value is used to indicate that no data exists for a particular combination of category values. As is the case for SQL, the aggregate functions ignore null values.

Example 45 In the case study we have one measure, $TotalAdmissions$, which is the total number of admissions by Diagnosis, Place, Time, and Reason. The default aggregation function is SUM.

The measures associated with each dimension may have different aggregation properties. For different kinds of measures, different aggregate functions are meaningful. For example, it is meaningful to sum up the number of admissions; and because this data is ordered, it is also meaningful to compute the average, minimum, and maximum values. In contrast, in at least some situations, it may not be meaningful to compute the sum (over time) of measures such as the number of patients hospitalized, but it remains meaningful to compute the average, minimum, and maximum values. Next, it makes little sense to compute these aggregate values on data such as diagnoses, which do not have any ordering defined on them. Here, the only meaningful aggregation is the count of occurrences. Whether or not an aggregate function is meaningful also depends on the dimensions being aggregated over. For example, patient counts may be summed over the Place dimension, but not over the Time dimension. For additional discussion of these issues, we refer to reference [70].

By recording what aggregate functions may be meaningfully applied to what data, it is possible to support correct aggregation of data. With such information available, it is possible to either completely reject “illegal” aggregation or to warn the users that the results may not be meaningful.

Following previous research [69, 102], we distinguish between three distinct sets of aggregate functions: Σ, applicable to data that may be added together, ϕ, applicable to data that can be used in average calculations, and c, applicable to data that may only be counted.

Considering only the standard SQL aggregate functions, we have that $\Sigma = \{$SUM, COUNT, AVG, MIN, MAX$\}$, $\phi = \{$COUNT, AVG, MIN, MAX$\}$, and $c = \{$COUNT$\}$. The aggregation types are ordered, $c \subset \phi \subset \Sigma$. If a set of aggregate functions is meaningful for some data, so are the functions in lower sets.

For each measure M for a set of dimensions $D = \{D_1, .., D_n\}$, we assume a function $a_M : D \mapsto \{\Sigma, \phi, c\}$ that gives the aggregation type for each dimension. In Section 5.3.2 we further discuss issues related to correct aggregation of data.

Example 46 In the case study, $a_{TotalAdmissions}(\textit{Diagnosis}) = \Sigma$.

An n-dimensional *summary database* (SDB) is a 3-tuple $S = (\mathcal{S}, D, M)$, where $\mathcal{S}$ is the schema, $D = \{D_1, .., D_n\}$ is a set of dimensions, and $M = \{M_1, .., M_k\}$ is a set of measures for the categories $\perp_{D_1}, .., \perp_{D_n}$.

Example 47 The case study has a 4-dimensional summary database with Diagnosis, Place, Reason, and Time as dimensions. There is one measure, the $TotalAdmissions$, as described above. A graphical illustration of the SDB is seen in Figure 5.2.

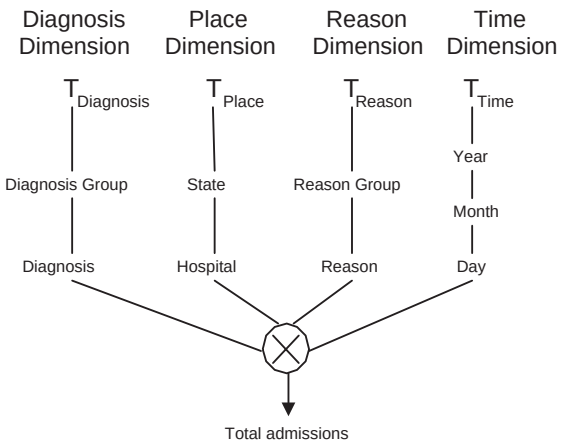

Figure 5.2: Summary Model for the Admissions Database

5.3.2 Summarizability

This section defines *summarizability*, an important property of SDBs related to the use of pre-computed aggregates. Intuitively, summarizability captures when higher-level aggregates may be obtained directly from lower-level aggregates.

Definition 11 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 argument on the left-hand side of the equation is a multiset, i.e., the same value may occur multiple times.

Summarizability is important since it is a condition for the flexible re-use of computed aggregates. Without summarizability, (pre-computed) lower-level results generally cannot be correctly combined into higher-level results. In such situations, we have to compute the higher-level results from base data, which may be computationally expensive.

It has been shown that summarizability is equivalent to the aggregate function (g) being *distributive* and the mappings between dimension values in the hierarchies being, *strict*, *covering*, and *onto* [70]. These properties are defined formally elsewhere [95, 97, 70]. Informally, summarizability requires that the dimension hierarchies take the form of balanced trees, i.e., all paths from the root have the same length (onto), links between values do not "skip" levels (covering), and all values below the root have exactly one parent (strictness). If hierarchies do not have this form, some lower-level values will either be double-counted or ignored.

Summarizability is closely related to the aggregation types defined in the previous section. We use the aggregation types to capture when it is safe to aggregate a measure over a given dimension. If we have aggregated over a non-summarizable hierarchy, e.g., a diagnosis hierarchy where one diagnosis is part of several diagnosis groups, it is not permissible to use the aggregate results for the diagnosis groups to compute the result for the entire dimension, as the same admissions will then be counted more than once. We use the aggregation types to prevent this. Problems related to summarizability also occur when we extend the queries over SDBs to include data from external ODBs, see Section 5.5 for details.

5.3.3 The Summary Query Language

The query language of the SDB component is termed SumQL and is meant to be language that makes it easy for the user to pose aggregate queries over SDBs. We have chosen to define a separate summary language rather than attempting to augment the object query language, OQL, for querying SDBs because we wish to refer explicitly to the special data structures in SDBs.

Using OQL, or some variant thereof, for querying SDBs would mean that we would have to overload some of the language constructs, re-using them with a different meaning. This is undesirable, as it confuses the meaning of statements in the language.

SumQL is reminiscent of SQL, but includes constructs that reflect SDB concepts such as measures, dimensions with hierarchically organized categories, and automatic aggregation, thus supporting naturally the expression of aggregate queries over summary databases. Using SumQL enables us to concisely and precisely define the extensions for referencing object data.

The general format of a SumQL query is displayed below and explained in the following. Symbol "+" indicates one or more occurrences and square brackets denote optional parts. The formal syntax and semantics of SumQL are given in Appendix 5.8.

SumQL query ::= SELECT measure$^+$
INTO summary_database
BY_CATEGORY category$^+$
FROM summary_database
[WHERE predicate_clause]

The SELECT clause contains a list of measures for which a result is to be computed. Unlike in SQL, aggregate functions such as SUM need not be specified; rather, the default aggregation function specified in the schema is automatically applied to aggregate the data. An INTO clause follows that specifies the SDB into which the result of the query is stored. Thus, SumQL queries take SDBs as arguments and return an SDB.

The BY_CATEGORY clause specifies the aggregation level at which the measures are to be computed. For each dimension *not* mentioned in this clause, the $\top$ category of the dimension is assumed. Effectively, all dimensions and measures not mentioned in the BY_CATEGORY and SELECT clauses are ignored.

The FROM clause specifies the SDB from which to aggregate. For simplicity, we only consider queries over one SDB, and no "drill-across" or "union" functionality is provided. However, the data model and query language can easily be extended to handle this (see [95] for an example). The optional WHERE clause specifies predicates that are applied to the SDB before aggregation occurs. The predicates can include standard constructs such as comparison operators, set operators, and string operators. These constructs are equivalent or similar to those found in SQL and OQL [13].

Example 48 The following SumQL statement computes the "Total Admissions" measure from the "admissions" SDB, aggregated to the level of Year and State, for the years after 1997. The resulting SDB is called "testdb."

SELECT TotalAdmissions INTO testdb BY_CATEGORY year, state
FROM admissions WHERE year $>$ 1997

5.3.4 The Object Model and Query Language

This section briefly reviews the object data model and query language used by the ODB component of the federation. We use the Object Data Management Group's object data model, ODMG 2.0 [13], and its associated query-language, OQL. The ODMG data model includes constructs such as object class definitions, attributes, object identifiers, set-valued attributes, reference attributes, tuple attributes, inverse attributes, inheritance structures, and object class unions. An in-depth coverage of the ODMG data model and the OQL language may be found in the literature [13].

Example 49 Data definitions for the demographic and epidemiology databases from the case study are shown in Figure 5.3 and Figure 5.4, respectively. The keyword "REQUIRED" for an attribute is similar to SQL's "NOT NULL" and means that a value must always be supplied for the attribute. Keyword "DERIVATION" indicates that the content of the attribute is derived from the content of an attribute in another

class. Keyword “isa” denotes a sub-class relationship, while “set-of” specifies a set-valued attribute with the given cardinality.

```
OBJECT CLASS State
  DESCRIPTION: "states"
  ID: name
  ATTRIBUTE name: VARCHAR(30) REQUIRED
  ATTRIBUTE area: INTEGER
  ATTRIBUTE cities
     DERIVATION: !in_state[City]

OBJECT CLASS City
  DESCRIPTION: "cities"
  ID: name
  ATTRIBUTE name: VARCHAR(30) REQUIRED
  ATTRIBUTE population: INTEGER
  ATTRIBUTE in_state: State REQUIRED
  ATTRIBUTE current_mayor: Mayor REQUIRED

OBJECT CLASS Mayor
  DESCRIPTION: "mayors"
  ID: name
  ATTRIBUTE name: VARCHAR(30) REQUIRED
  ATTRIBUTE age: INTEGER
  ATTRIBUTE city
     DERIVATION: !current_mayor[City]
```

Figure 5.3: Demographic ODB Schema.

The OQL query language has constructs such as path expressions and class selectors. Path expressions are used to navigate through *reference attributes* to other classes using dot-notation, while class selectors restrict queries to operate only on a certain subclass.

Example 50 The following query uses a path expression to select the city name only for cities where the current mayor is more than 40 years old. The path expressions navigates from cities to mayors via reference attribute “current_mayor.”

SELECT City.name FROM City WHERE City.current_mayor.age $>$ 40

Example 51 The next query navigates from symptoms to the diagnoses that exhibit those symptoms using a path expression and then applies a class selector (the square brackets) to select the attribute “transfer_mode” of the Diagnosis sub-class “Contagious Diagnosis.” Thus, only transfer modes for contagious diagnoses with the the symptom “Cough” are returned:

```
OBJECT CLASS Symptom
  DESCRIPTION: "symptoms"
  ID: name
  ATTRIBUTE name: VARCHAR(50) REQUIRED
  ATTRIBUTE description: VARCHAR(255)
  ATTRIBUTE diagnoses
     DERIVATION: !symptoms[Diagnosis]

OBJECT CLASS Diagnosis
  DESCRIPTION: "diagnoses"
  ID: code
  ATTRIBUTE code: VARCHAR(10) REQUIRED
  ATTRIBUTE text: VARCHAR(100)
  ATTRIBUTE deaths_pr_year: INTEGER
  ATTRIBUTE incidences_pr_year: INTEGER
  ATTRIBUTE lifestyle_disease: CHAR(1)
  ATTRIBUTE symptoms: set-of [0,] Symptom

OBJECT CLASS ContagiousDiagnosis isa Diagnosis
  DESCRIPTION: "Only contagious diseases"
  ATTRIBUTE transfermode: VARCHAR(30)

OBJECT CLASS NonContagiousDiagnosis isa Diagnosis
  DESCRIPTION: "Non-contagious diseases"
```

Figure 5.4: Epidemiology ODB Schema

```
SELECT Symptoms.diagnoses[ContagiousDiagnosis]transfer_mode
FROM Symptoms WHERE Symptoms.name = “Cough”
```

5.4 Linking Databases

This section defines the links that are used to connect SDBs and ODBs. As mentioned in the introduction, we use explicit links to connect the databases, rather than relying solely on implicit knowledge of relationships among the databases when formulating queries.

Explicit links are preferable for several reasons. First, even if the data in the SDB is derived from source data in an ODB, the complete mapping may be unknown because of substitutions for missing data and other types of data cleansing, interpolation, etc. Second, explicit links are needed when linking an SDB to an unrelated ODB, i.e., an ODB other than the base data from which the SDB was extracted. Third, the source data may be sensitive and thus unavailable to the SDB user. So,

we propose to explicitly link even summary data to the base data from which it was derived.

Links are logically kept separate from the federated data, but can physically be implemented as part of these databases.

Formally, a *link* L from a category C to an object class O is a relation $L = \{(c, o)\}$, where $c \in C$ and $o \in O$. All links have a *name* to distinguish them. This is because each category and even pair of category and object class may have several links.

Links may be specified in several ways. An *equivalence link* is specified by a predicate $C = O.a$, where C is a category, O is an object class, and a is an attribute of O that uniquely identifies instances of O, i.e., a is a candidate key for O in relational database terms. Equivalence links occur when a category in the SDB represents the same real-world entities as does some object class in an ODB. An *attribute link* is specified by the same type of predicate, the only exception being that a does not uniquely identify instances of O. An *enumerated link* is given by an a link relation $L = \{(c, o)\}$, where pairs of dimension values in C and object ids from class O are explicitly enumerated. Therefore multiple dimension values may be assigned to the same object. Enumerated links are typically used for linking a category in an SDB and an object class that do not represent the same real-world entities.

Example 52 In our case study, we can specify an equivalence link between the Diagnosis category in the Admissions SDB and the Diagnosis Class in the Epidemiology ODB by the predicate "Diagnosis = Diagnosis.Code," as the values of the Diagnosis category are the codes of the diagnoses. In subsequent examples, we term this link "diag_link."

Example 53 An enumerated link from the Hospital category in the SDB to the City class in the Demographic ODB may be specified by explicitly assigning hospitals to cities based on where the hospitals are located. The contents of the link relation is $L = \{$("Alta Bates","Berkeley"), ("Portland General Hospital","Portland"), ("Portland Kaiser","Portland") $\}$. We will use the name "city_link" for this link.

The *cardinality* of a link is an important property, as the cardinality may affect summarizability. The cardinality of a link $L = \{(c, o)\}$ between category C and object class O is $[1 - 1]$ if $|L| = |\pi_C(L)| = |\pi_O(L)|$, where π denotes relational projection and $|.|$ denotes relation cardinality; the cardinality of L is $[n - 1]$ if $|L| = |\pi_C(L)| > |\pi_O(L)|$; and the cardinality is $[1 - n]$ if $|L| = |\pi_O(L)| > |\pi_C(L)|$. Finally, if the cardinality of L is not $[1 - 1]$,$[1 - n]$, or $[n - 1]$, its cardinality is $[n - n]$. For some link properties, only the cardinality of the object side of a link is interesting. As a short-hand notation, we say that the cardinality of a link is $[-1]$ if the cardinality is $[1 - 1]$ or $[n - 1]$. Similarly, the cardinality of a link is $[-n]$ if the cardinality is $[1 - n]$ or $[n - n]$.

Example 54 The cardinality of link "diag_link" is $[1 - 1]$ and the cardinality of "city_link" is $[n - 1]$.

It is also necessary to capture whether some dimensions values or objects do not participate in a link. For that purpose, we define that a link $L = \{(c, o)\}$ from

category C to object class O *covers* C if $C = \pi_C(L)$. Similarly, L covers O if $O_i = \pi_O(L)$, where O_i is the set of object ids for O. If L covers both C and O, L is *complete*; otherwise, L is *incomplete*.

Example 55 The "diag_link" link is complete, while the "city_link" link covers the Hospital category, but not the City class. For example, the city of Oakland is not present in the link.

In Section 5.5 we will explore the effect of these link properties on the semantics of queries. Specifically, we shall see that incomplete links and $[-n]$ links, which are analogous to non-summarizable hierarchies, require special attention. Interestingly, an attribute link always has a link cardinality that is $[-n]$, while an equivalence link always has a $[1-1]$ cardinality.

In some situations, it is desirable to have links that are more powerful than enumerated links. For example, the database designer may want to annotate links with what may be termed metadata, e.g., the reason why the link was added, who added the link, or the time interval when the link is valid.

Such annotated links do offer additional modeling capabilities, but are nevertheless excluded. The reason is that offering a general solution along these lines—which allows general annotations, including complex object structures with set-valued attributes, references to other classes, embedded objects, etc.—would amount to the reinvention of a complete object model, an unnecessary complication.

Instead, we propose that annotations be stored in a separate ODB, and we propose to store the potentially complex link information in a separate ODB using a *link class* that represents the instances of the link. We may then create a normal link from the desired category to this link class. The link class would also be linked to the desired object class that we wanted to link to originally.

We do not consider links between ODBs, as this is supported by object database federation systems, e.g., the "OPM*QS" multidatabase system [18].

5.5 The Federated Data Model and Query Language

Having described the data models and query languages of the SDB and ODB components to be federated, as well as a minimal mechanism for linking SDBs and ODBs, the next step is to provide language facilities that enable OLAP-type queries across the entire federation. Specifically, we extend SumQL.

The federation approach presented here has the distinguishing feature that it uses the aggregation semantics of the data to provide *aggregation-safe* queries, i.e., queries that do not return results that are incorrect or meaningless to the user. This section describes how the previously defined concepts of aggregation types, summarizability, link cardinality, and link coverage combine to provide aggregation-safety for queries.

5.5.1 The Federated Data Model

The federation consists of a collection of independent components, supplemented with additional information and components that enable functioning of the federation.

Specifically, the federation consists of an SDB, a number of ODBs, and links that interrelate information in the different databases. Formally, a federation F of an SDB S and a set of ODBs $O = \{O_1, .., O_n\}$ is a three-tuple $F = (S, O, L)$, where $L = \{L_1, .., L_m\}$ is a set of links from categories in the dimensions of S to classes in $O_1, .., O_n$.

We assume only a single SDB. Permitting multiple SDBs introduces additional challenges, e.g., the matching of categories and dimensions, which are not covered here. The case of a single SDB is very useful, as typical queries to a federation naturally centers around one SDB: Typical queries concern SDB measures, grouped by SDB categories, and involving selection criteria relating to data from the ODBs; or queries retrieve ODB data along SDB data; and in some cases, it is desirable to actually *group* SDB data by categorical ODB data.

Rather than requiring that the SDB and ODB data comply with one common data model, the federation adopts a *multi-paradigm* approach [50, 6], where the data remain in their original data models. This approach has previously been advocated in programming languages, where research has been done on how to allow programs to be written that exploit imperative, object-oriented, functional, and logical programming paradigms in a single program [10].

Allowing multiple data models (or paradigms) to co-exist in the federation enables us to exploit the strengths of the different data models and query languages when managing and querying the data. In particular, the availability of multiple paradigms allows a problem solution to take advantage of the fact that certain subsets of a problem are often well suited for one solution paradigm, while other problem subsets are better suited for other paradigms.

Like the arguments to queries are federated databases, the results are also federated databases, i.e., query results may have SDB, ODB, and link components. This closure property mirrors those of the well-known relational, object, and multidimensional data models and query languages, and permits the result of one query to be used in a subsequent query. We allow the sets of ODBs and links, O and L, to be empty. Thus, an SDB in itself is a federation.

5.5.2 The SumQL++ Language

As our objective is to allow more powerful OLAP queries over SDBs by allowing the queries to include data from ODBs, we take SumQL as the outset and extend this language. The new, extended language is termed "SumQL++" as it introduces object-oriented concepts into its predecessor, akin to the C++ successor to the C programming language.

The queries we are interested in are the typical OLAP queries that select a set of measures from an SDB, grouped by a set of categories. Three extensions of SumQL are useful in this respect. First, we introduce path expressions in selection predicates, in order to integrate ODB data. Second, we introduce so-called *decorations* [40] of SumQL results, which enable ODB data to be returned along with the SumQL result. Third, SumQL is extended to enable SDB data to be grouped by data belonging to ODBs, i.e., attributes of object classes, rather than just the built-in SDB categories.

Extended Selection Predicates

The first extension of SumQL is to allow selection predicates that reference ODB data. The basic idea is to allow the use of standard OQL *path expressions*, as described in Section 5.3.4, in the category expressions in the selection predicates, using the well-known dot-notation for path expressions.

The link that is used to get to the ODB is included in the category expression. A category expression always starts with an SDB category, and is followed by an optional part consisting of the link name and a path expression. Inside the path expressions, *class selectors* may occur that restrict predicates to work on selected (sub)classes. The syntax is shown below. The square brackets in single quotes in the "class_connector" rule denote (sub)class selection and are part of the language being defined. Nonterminals not defined below are strings.

category_exp	::=	category [. link object_path attribute]
object_path	::=	class_connector | path_list
class_connector	::=	. | '[' class ']'
path_list	::=	class_connector path_element | path_list path_element
path_element	::=	reference_attribute class_connector

Example 56 We want to use the Epidemiology ODB to get the total admissions by year for only the diagnoses for which cough is a symptom. We use the "diag_link" link to do so in the following the SumQL++ statement.

SELECT TotalAdmissions INTO testdb BY_CATEGORY Year
FROM Admissions WHERE Diagnosis.diag_link.symptoms.name = "Cough"

Example 57 We use a class selector in the Epidemiology ODB to get the total admissions by year for only contagious diagnoses with the transfer mode "Air," with the following SumQL++ statement.

SELECT TotalAdmissions INTO testdb BY_CATEGORY Year FROM Admissions
WHERE Diagnosis.diag_link[ContagiousDiagnosis]transfer_mode = "Air"

To describe the semantics of this extension to SumQL, we first need some additional definitions. Given a category expression E of the form $E = C.L.OP.a$, where C is a category, L is a link, OP is a object path (as defined in the syntax above), and a is an attribute of an object class, the *cardinality* of E is defined next.

Let R be the set of attribute values resulting from the OQL query "SELECT $X.k, X.OP.a$ FROM X," where X is the class that L links to, k is the attribute that L links to in X, and OP and a are as above. Let L' be the link relation obtained by performing a natural join of L with R, i.e., $L' = L \bowtie R$, where $\bowtie$ denotes natural join. We say that L' is the link *specified by* E. The cardinality of E is defined as the link cardinality of L'.

Informally, the cardinality of a category expression is the combination of the cardinalities that we encounter as we go through the link and the subsequent (possibly set-valued) reference-attributes, i.e., going through a $[-1]$ relationship in a link or a

reference attribute does not change the running cardinality, but a $[-n]$ relationship causes the total cardinality to be $[-n]$.

Using the definitions above, and following the definitions given for links, we say that E *covers* O, *does not cover* O, *covers* C, *does not cover* C, is *complete*, and is *incomplete*, if L' *covers* O, *does not cover* O, *covers* C, *does not cover* C, is *complete*, or is *incomplete*, respectively. Above, O is the object class that a is an attribute of, i.e., the last object class reached in the category expression. We say that O is the *final class* of E. C is the category in the beginning of E. We say that C is the *starting category* of E.

Example 58 The cardinality of the category expression "Hospital.city_link.located-in.name" is $[n-1]$ as we only go through $[n-1]$ relationships and the state name is a key attribute. The cardinality of the category expression "Diagnosis.diag_link.symptoms.name" is $[n-n]$ because the "symptoms" reference attribute is set-valued.

The cardinality and covering properties of a category expression affect the meaning of a SumQL++ statement. If the cardinality is $[-1]$, the predicate will only reference one attribute value and the meaning is clear. However, if the cardinality is $[-n]$, the predicate will reference more than one attribute value, leading to several possible semantics for the query.

For example, the category predicate "Diagnosis.diag_link.symptoms.name = "Cough" " in Example 56 has a $[-n]$ cardinality. One possible interpretation of this is that *all* the referenced attribute values must match the predicate, e.g., that *all* symptoms must have name "Cough." Another interpretation is that *at least one* attribute value must satisfy the predicate, e.g., that at least one symptom has name "Cough." This is the interpretation chosen in the OQL language, and as we also think it is the most sensible to end users, we will also adopt this interpretation.

Similar problems may when a category expression E does not cover its starting category C, because L' then will be undefined for the uncovered dimension values of C. However, if we adopt our previous interpretation, that *at least one attribute value* must match the predicate, the meaning is well-defined. The values in C not covered by E will then be excluded from the selection. There are no problems if E does not cover its final class O, as L' will be defined for all the instances of O referenced by E.

Formally, the semantics of the extended SumQL++ predicates are as follows. We are given a SumQL++ query Q with a number of category predicates $P_1, .., P_N$ of the form $P_i = E_i \ POP_i \ V_i$. The $E_1, .., E_n$ are category expressions of the form $E_i = C_i.L_i.OP_i.a_i, i = 1, .., n$, where C_i is a category, L_i is a link, OP_i is a object path, and a_i is an attribute of the final class of E_i. The POP_i are the *predicate operator* parts of P_i, i.e., comparison and BETWEEN, IN, and MATCH operators. The V_i are the *value* parts of the predicates.

For each E_i, let R_i be the set of attribute values resulting from the OQL query "SELECT $X_i.k_i$ FROM X_i WHERE $OP_i.a_i \ POP_i$," where X_i is the class that L_i links to, and k_i is the attribute that L_i links to. For each predicate P_i, we now form a modified predicate $P_i' = C_i$ IN $(e_{1_i}, .., e_{k_i})$, where $\{e_{1_i}, .., e_{k_i}\} = \pi_{C_i}(L_i \bowtie R_i)$ ($\bowtie$ denotes natural join). Informally, we obtain the attribute values for the link class

for which the predicate holds, then obtain the corresponding dimensions values by joining with the link, and finally form a (pure) SumQL predicate with the resulting dimension values using the "IN" notation.

With Q' being the query obtained from Q by substituting all the P_is with the P'_is, the result of evaluating Q on a federation $F = (S, O, L)$ is the federation $F' = (S', \emptyset, \emptyset)$, where S' is the SDB resulting from evaluating Q' on S. This federation has no ODB or links components, which makes sense as the ODB data was only used to select a subset of the SDB for evaluation.

Example 59 We evaluate the query from Example 56. First we get the result of the query "SELECT Diagnosis.code FROM Diagnosis WHERE Diagnosis.symptoms-.name="Cough." The result of this is the set $R = \{$"N12"$\}$ (the code for pneumonia). We then join R with the link relation "diag_link," which is the identity relation, and project over the Diagnosis category, obtaining the dimension value "N12". We then form the pure SumQL query: "SELECT TotalAdmissions INTO testdb BY_CATEGORY Year FROM Admissions WHERE Diagnosis IN ("N12")," evaluating it over the Admissions SDB.

Decorating the Query Result

It is often desirable to display additional descriptive information along with the result of an SDB query. This is commonly referred to as *decorating* the result of the query [40]. For example, when asking for the number of admissions by hospital, it may be desirable to display the name of the city and the name of the city's mayor along with the hospital name.

This can be achieved by extending the SumQL with features for decorating the result. One possibility would be to allow category expressions with path expressions in the SELECT clause, but we advise against this as it would then be unclear which parts of the SELECT clause referred to measures and which parts referred to decorations. Instead, we extend SumQL with an optional "WITH" clause. The extended syntax is shown below.

SumQL query ::= SELECT measure$^+$
INTO summary_database
BY_CATEGORY category$^+$
[WITH expression$^+$]
FROM summary_database
[WHERE predicate_clause]

Example 60 Using this extension, we select the number of admissions by hospital, decorated with the names of the city and its mayor.

```
SELECT TotalAdmissions INTO testdb FROM Admissions
BY_CATEGORY Hospital
WITH Hospital.city_link.name, Hospital.city_link.current_mayor.name
FROM Admissions
```

It only makes sense to decorate the result with data that is correlated to the original query result, so the categories referenced in the WITH clause MUST be part of the BY_CATEGORY clause.

Formally, assume a SumQL++ query Q with category expressions $E_1, .., E_n$ in the WITH clause of the form $E_i = C_i.L_i.OP_i.a_i, i = 1, .., n$, where C_i is a category, L_i is a link, OP_i is a object path, and a_i is an attribute of the final class of E_i, the semantics is as follows. For each E_i, let R_i be the result of the OQL query "SELECT $X_i.k_i, X_i.OP_i.a_i$ FROM X_i," where X_i and k_i are the class that L_i links to and the attribute that L_i links to, respectively. Then form a new object class Z_i from the set of tuples $L_i \bowtie R_i$ using the concatenation of the category C_i and the attribute a_i as its object identifier. Let Q' denote Q, but without the WITH clause. The result of evaluating Q over a federation $F = (S, O, L)$ is the federation $F' = (S', O', L')$, where S' is the result of evaluating Q' over F, $O' = \{\{Z_i\}\}$, and $L' = \{L'_i\}$, where L'_i are attribute links specified by $C_i = Z_i.C_i$.

Thus, the decoration data is returned in the ODB and link parts of the federation and is not integrated into the result SDB. This loose coupling of decoration data and SDB data is essential in avoiding semantic problems, which might otherwise occur if the category expressions E_i do not cover the categories C_i. In this case, we just return decoration data matching a subset of the C_i, i.e., we perform an operation equivalent to an outer join. Similarly, no cardinalities for the E_is cause problems. If the cardinality of E_i is $[-n]$, e.g., for the expression "Diagnosis.diag_link.symptoms.name," the object class simply contains several objects for each C_i value, e.g., there will be two objects, with the symptom names "Cough" and "Fever," with Diagnosis value "N12" (pneumonia).

Example 61 For the query in Example 60, we get two object classes in the result, CityName with the attributes "hospital," "name," and "city∘hospital," with the latter as the object identifier, and MayorName with the attributes "hospital," "name," and "mayor∘hospital," again with the latter as the object identifier. The links have the specifications "Hospital = CityName.Hospital" and "Hospital = MayorName.Hospital."

Grouping By Object Class Attributes

The last extension is to allow the measures of an SDB to be grouped by attribute values in ODBs, enabling aggregation over hierarchies outside the SDB. This feature will be used when aggregation requirements change suddenly.

To achieve this, we allow *category expressions* instead of just categories in the BY_CATEGORY clause. The syntax of the extension is given below. The only difference from the previous syntax is that the BY_CATEGORY clause now is a list of category expressions rather than just a list of categories. Remember that a category expression is either a category or a category followed by a link, an object path, and an attribute.

SumQL query ::= SELECT measure$^+$
INTO summary_database
BY_CATEGORY expression$^+$
[WITH expression$^+$]
FROM summary_database
[WHERE predicate_clause]

Example 62 The number of admissions grouped by symptoms may be retrieved as follows.

SELECT TotalAdmissions INTO testdb
BY_CATEGORY Diagnosis.symptoms.name FROM Admissions

This type of SumQL++ queries will return SDBs where one new dimension is added for each category expression in the BY_CATEGORY clause, thereby reflecting the hierarchy specified by the category expression, and aggregation will occur over these new dimensions.

Formally, given a SumQL++ query,

$$Q = \text{“SELECT } M_1, .., M_k \text{ INTO db BY\_CATEGORY } E_1, .., E_n \text{ FROM } S \text{ WHERE } P\text{”},$$

with the category expressions in the BY_CATEGORY clause being of the form $E_i = C_i.L_i.OP_i.a_i,\ i = 1, .., n$, where C_i is a category, L_i is a link, OP_i is a object path, and a_i is an attribute of the final class of E_i, the result of Q on federation $F = (S, O, L)$ may be specified as follows.

First, let $S' = (\mathcal{S}', D', M')$ be the SDB obtained from S as follows. For each E_i, add a new dimension type to S with the category types $\top_i$, A'_i, and $\bot'_i$. Category type A'_i represents the attribute values of a_i, while category type $\bot'_i$ represents the dimension values of the bottom category in S. The ordering of the types is $\top_i > A'_i > \bot'_i$. Thus, $\mathcal{S}'$ is specified.

For each dimension type, new dimensions D'_i are added to D'. The categories of D'_i correspond to the category types. The $\top_i$ category has just the $\top$ value. If L'_i is the resulting link of E_i, category A'_i has the values given by $\pi_{a_i} L'_i$. Let R_i be the relation specified by $(e_1, e_2) \in R_i \Leftrightarrow e_1 \in \bot_i \wedge e_2 \in C_i \wedge e_1 \sqsubseteq_i e_2$, i.e., the relation specified by the partial order between $\bot_i$ values and C_i values. Let $B_i = R_i \bowtie L'_i$ ($\bowtie$ is the natural join). Then the values of the category $\bot'_i$ is the set $\pi_{\bot_i}(B_i)$. The partial order on dimension D'_i, $\sqsubseteq'_i$, is specified as follows: $e_1 \sqsubseteq'_i e_2 \Leftrightarrow e_2 = \top \vee e_1 = e_2 \vee (e_1, e_2) \in B_i$. This completes the specification of D'.

The set of measures M' is identical to the original set of measures M as the measures operate on the same categories.

The result of evaluating a SumQL query “SELECT $M_1, .., M_k$ INTO S'' BY_CATEGORY $A'_1, .., A'_n$ FROM S' WHERE P” is the federation $F' = (S'', \emptyset, \emptyset)$. The ODB and links components are empty, as the ODB data has been turned into dimensions in this result.

Example 63 For the query in Example 62 we get one new dimension type "SymptomName" with the category types "$\top_{SymptomName}$," "SymptomName," and "Diagnosis." The new "SymptomName" dimension has the categories specified by the category types. The partial order on the new dimension is given by joining the "Diagnosis," "Diagnosis_Symptom," and "Symptom" tables from Table 5.3 and then projecting on the "Code" and "Name" attributes. We note that the resulting hierarchy is non-strict, as the "Acetone Breath" symptom occurs for both "Insulin Dependent Diabetes" and "Non Insulin Dependent Diabetes."

Depending on the properties of the E_is, problems may occur in the aggregation process. If E_i does not cover C_i, some of the data in the SDB (the data characterized by the non-covered subset of C_i) will not be considered in the aggregate result. Reversely, if E_i does not cover its final class O_i, there will not be any measure data associated with the non-covered objects in O_i. This means that the result of the aggregation function will be undefined for multidimensional tuples containing the non-covered objects. To remedy these problems, we require that the E_is be *complete*.

Even when the category expressions are complete, special attention is needed to ensure summarizability. Problems may occur when the cardinality of an E_i is $[-n]$, in which case the same measure data, e.g., the same admissions, will be accounted for more than once in the overall result, e.g., for different symptoms.

This result is meaningful and correct in itself because the data belongs to several groups. However, the result should not be used for *further aggregation* as the same data may then be accounted for more than once for the same group, e.g., we may not aggregate over all symptoms to get the total number of admissions. To avoid this, we set the aggregation type for all measures to c, i.e., we *disallow* further aggregation on the data, if the cardinality of E_i is $[-n]$. If the cardinality of E_i is $[-1]$ the aggregation types are not changed.

5.5.3 Summary

Although the extensions to SumQL were described separately above, they can be used together in one SumQL++ statement. Assuming an SumQL++ statement that contains all three extensions, query evaluation proceeds as follows. First, the rules for handling grouping by object attributes are used, producing a statement without object attribute grouping. This statement is then processed using the rules for the WITH clause described in Section 5.5.2, resulting in a statement without a WITH clause, which can then be evaluated using the rules for extended selection predicates as described in Section 5.5.2. The statement produced by the extended predicate rules is a pure SumQL statement which may be evaluated following standard SumQL semantics.

5.6 Implementation

This section describes the prototype implementation of a system capable of answering SumQL++ queries over federations of an SDB and several ODBs.

5.6.1 Implementation Overview

The overall architecture of the federated system is seen in Figure 5.5. The parts of the system handling object and link data are based on the commercially available OPM tools [72, 38] that implement the Object Data Management Group's (ODMG) object data model [13] and the Object Query Language (OQL) [13] on top of a relational DBMS, in this case the Oracle8 RDBMS. In-depth descriptions of the OPM toolset exist in the literature [16, 17]. The OLAP part of the system has been implemented using both Microsoft's SQL Server OLAP Services using the Multi-Dimensional eXpressions (MDX) [78] query language (MOLAP) and the Oracle8 RDBMS using SQL as the query language (ROLAP). The graphical user interface (GUI) is implemented as Java classes running in a standard Web browser for optimal flexibility. A description of the user interface may be found in Chapter C.

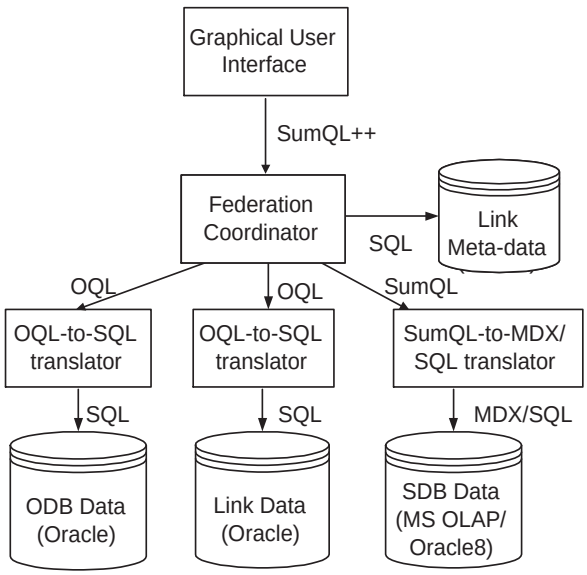

Figure 5.5: Architecture of the Federated System

The system has six major components: the GUI, the ODB systems, the link DB system, the SDB system, the federation coordinator, and the metadata database. The ODB, link DB, and SDB components are treated as independent units by the federation system; only their published interfaces are used, and no assumptions about their internal workings are made. The link component stores enumerated links and is placed in an independent "link" DB, as it cannot generally be assumed that these links may be stored in some ODB component. Should this be possible, we can choose to do so, e.g., to obtain better performance. The operation of the prototype is entirely based on federation metadata specified in the metadata database. This allows for a very flexible system that may adapt quickly to changes. For example, if a new connection to an outside ODB is desired, appropriate links just needs to be specified and stored as metadata, after which queries can start using the new ODB.

5.6.2 Representation of Metadata

We now describe the metadata for the federation system in detail. A UML diagram describing the meta-data is seen in Figure 5.6.

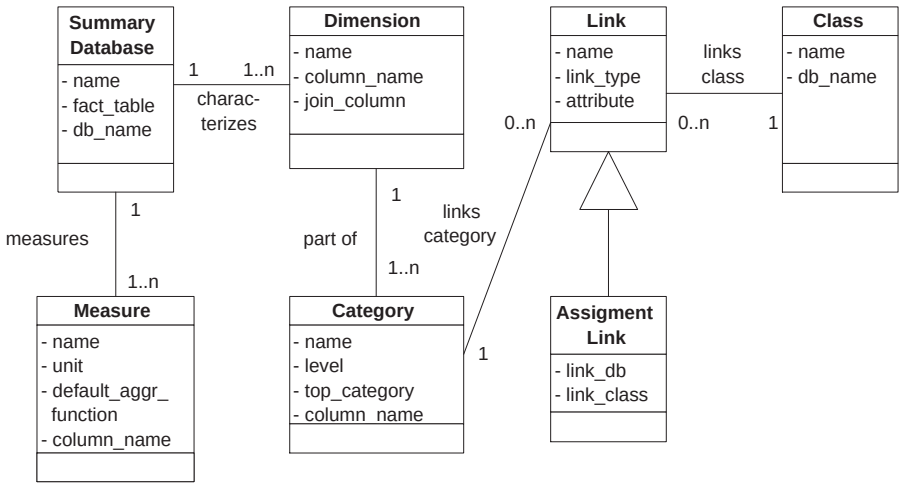

Figure 5.6: UML Schema of Federation Meta-Data

The meta-data consists of two subject areas: summary meta-data and link meta-data. For simplicity, we have chosen to implement both the summary and link meta-data in the same meta-database, but the system does not make any assumptions about this. The summary meta-data could be stored in a separate meta-database, e.g., the meta-database for an OLAP system used to implement the SDB. The summary meta-data contains information about the SDBs and their internal components such as measures, dimensions and categories, as well as information on how the SDB is mapped to the implemention of the SDB, e.g., a relational star schema. The link meta-data tracks the links in the system, their types, and the categories and object class attributes they link to. For assignment links, we also record the ODB and class used to hold the physical link relation. The meta-data for ODBs is not recorded in the meta-database, instead it is maintained by the OPM system as a part of the ODBs. Thus, we have no redundancy in the meta-data, which makes it much easier to maintain consistency when the meta-data changes.

5.6.3 Query Processing

This section describes the processing of SumQL++ queries over the federation. We start by giving an overview of the processing of queries using MS SQL Server OLAP Services for the SDB.

Queries are generated by the GUI and sent to the federation coordinator which then parses the query. Based on the content of the query, the system looks up the relevant metadata (link specifications, ODB names, etc.) in the metadata database and pro-

cesses the query according to the metadata by issuing queries to the DB components. Example 64 below explains how a particular query is processed.

Example 64 The query below selects the total admissions by diagnosis, state and year, restricted to diagnoses with “Cough” as a symptom and years later than 1997.

```
SELECT TotalAdmissions INTO testdb BY_CATEGORY Diagnosis,State,Year
FROM Admissions
WHERE (Diagnosis.diag_link.symptoms.name ="Cough") AND (Year > 1997)
```

This query is processed as follows. The Federation Coordinator (FC) parses the query and identifies the link and ODB parts of the query. Based on the link name (diag_link), the FC looks up in the metadata which ODB, object class, and attribute the link is to and the type of the link, i.e., equivalence, attribute, or enumerated. For this query, the ODB is the “Epidemiology” DB, the class is “Diagnosis,” the attribute is “code,” and the link type is “equivalence.” The object path to be followed is “.symptoms,” and the final attribute is “name.”. Based on this information, the FC forms the OQL query seen below.

```
SELECT code = @n001 FROM @n000 IN SUMDB:Diagnosis,
@n001 IN @n000.code WHERE @n000.symptoms.name = "Cough";
```

The OQL query is then executed against the Demographic ODB, giving as result the single diagnosis code “N12”. Based on the result of the OQL query, the FC now forms the SumQL query seen below, which is executed against the SDB component of the federation to obtain the final result. The reason for using the intermediate SumQL statements is to isolate the implementation of the OLAP data from the FC. As an another alternative, we have also implemented a translator into SQL statements against a relational "star schema" design.

```
SELECT TotalAdmissions INTO testdb BY_CATEGORY Diagnosis,State,Year
FROM Admissions WHERE (Diagnosis IN ( 'N12' ) AND Year > 1997)
```

The SumQL query is now translated into the MDX statement seen below and executed against the SDB managed by MS SQL Server OLAP Services.

```
SELECT [Measures].[TotalAdmissions]  ON COLUMNS, INTERSECT
(CROSSJOIN(CROSSJOIN([Diagnosis].[N12], [Place].[State].MEMBERS),
[Time].[Year].MEMBERS),CROSSJOIN(CROSSJOIN([Diagnosis].[Diagno
sis].MEMBERS,[Place].[State].MEMBERS), FILTER([Year].MEMBERS,
[Time].CURRENTMEMBER.NAME > "1997")))
ON ROWS FROM Admissions
```

This example was intended to illustrate the amount of work that a user will have to go through without the aid of the user interface and the federated translation tools. In particular, we wish to emphasize the usefulness of the OLAP-object database links to generate the combined result. Also, the users are spared the verbosity of MDX (which is hidden from them).

We now describe the query processing strategy of the prototype in detail. The pseudo-code below describes the query processing of a SumQL++ statement with predicates having path expressions in them. The "<" and "> are used to indicate substitutions.

The query processing works by examining each extended predicate separately. For each predicate, the meta-data for the link used in the predicate is retrieved. Based on the meta-data, an OQL query is constructed to retrieve the attribute values for the object side of the link for which the predicate holds. If the link is an equivalence or attribute link, the retrieved values can simply be used to form a pure SumQL predicate using the "IN" notation. If the link is an assignment link, another OQL query is constructed and executed against the link database for the particular link. This query essentially *joins* the previous result set of object attribute values with the link relation to retrieve the dimension values for which the predicate holds. These dimension values are then used to form a pure SumQL predicate using the "IN" notation.

(1) **procedure** Evaluate(Q) /* Q is a SumQL++ query
(2) $Q' \leftarrow Q$
(3) **for** each SumQL++ predicate P in Q /*in the form of <category>.<link>.<objpath> <constraint> */
(4) Get <object_db>,<class_name>,<attribute>,<type> for <link> from eta-database
(5) OQL ← " SELECT obj.<attribute>
FROM obj in <object_db>:<class_name>
WHERE <objpath> <constraints>"
(6) $R_1 \leftarrow$ Execute(OQL)
(7) **if** link.type IN $\{E, A\}$ **then** /* equivalence or attribute link */
(8) $P' \leftarrow$ "<category_name> IN ($< R_1 >$)"
(9) **else** /* Assignment link */
(10) Get <link_db> and <link_class> from meta-database
(11) LinkOQL ← " SELECT link.category
FROM link in <link_db>:<link_class>
WHERE link.<attribute> IN (<R1>)"
(12) $R_2 \leftarrow$ Execute(LinkOQL)
(13) $P' \leftarrow$ "<category_name> IN ($< R_2 >$)
(14) **end if**
(15) $Q' \leftarrow$ Replace(P with P' in Q')
(16) **end for**
(17) Execute(Q') on the SDB

5.6.4 Query Optimizations

We now describe some possible optimizations to the query processing strategy described above. These optimizations are not implemented in the prototype.

The query processing described above makes use of substituting sets of values directly into the queries using the "IN" notation. Most of the time, this will give acceptable performance, as the sets of values are quite small, usually only a few hundred values, and at most a few thousand values. However, sometimes the categories will be too large, e.g., a large customer dimension, for this approach to be feasible. If

the query performance is not satisfactory, there are two optimizations that are useful. The first is to introduce *temporary categories* into SumQL. Temporary categories are used to hold subsets of dimension values for a particular category, e.g., the values that result from an extended selection predicate. The temporary categories (implemented as MDX named sets or RDBMS tables) can then be used during the processing of SumQL statements by joining them with the dimension tables before joining with the fact table. This will often provide better performance than listing the values directly in the SumQL statement as the parsing of the SumQL statement will be faster and the underlying DBMS will be able to process the resulting MDX or SQL query more efficiently.

The other optimization takes advantage of the fact that SDBs often store pre-aggregated data to answer summary queries faster. In the standard query processing strategy, selection is always performed before aggregation. However, as is the case for RDBMS query processing, it is sometimes an advantage to re-arrangere the order of the operators, in this case we want to perform aggregation before selection. Naturally, we can only perform such re-arrangements when they are semantically correct, i.e., when the result of the query remains the same, so certain criteria must be met for the re-arrangement to be valid. In this case, we can perform aggregation before selection if the selection criteria in a dimension is applied to a category at a higher level than the category being aggregated to in that dimension. This is formally stated by the following theorem. If a selection predicate is not applied to a particular dimension, this is interpreted as the selection being "$\top_i = \top$," i.e., the selection that selects everything in the dimension. Thus, we interpret the selection predicate as being applied to the top category in this case.

Theorem 9 *Let $S = (\mathcal{S}, D, M)$ be an n-dimensional SDB such that $\mathcal{S} = \{\mathcal{T}_i\}, i = 1, .., n$, $D = \{D_i, i = 1, .., n\}$. Let $C_i, C_i', i = 1, .., n$ be categories such that $\forall i : C_i \sqsubseteq_{\mathcal{T}_i} C_i'$. Let p be a predicate such that p is applied to category C_i' in dimension i. Then $\alpha[C_1, .., C_n](\sigma(S, p)) = \sigma(\alpha[C_1, .., C_n](S), p)$.*

The performance gain of this optimization comes from the fact that the aggregation operation may just correspond to a simple lookup because the data is already pre-aggregated. Thus, this optimization is only a good idea if pre-aggregation is used. This optimization is one example of the usefulness of having a low-level, algebraic query language as well as a high-level user-oriented language such as SumQL.

5.6.5 Implementation of the SDB System

In the sections above, we have described how to use MS SQL OLAP Services and the MDX language for implementing the SDB system. However, due to the component architecture of the system where the implementation-neutral SumQL language is used to express queries over the SDB, it is quite easy to use a ROLAP system for the SDB instead. To demonstrate this, we have also implemented the SDB component as a ROLAP system, using a star schema design [64] on the Oracle DBMS. For this implementation, the SDB component receives a SumQL query which is then translated into an SQL query over the star schema, based on the meta-data for the particular

SDB. We have chosen to also provide a ROLAP SDB implementation for the prototype for several reasons. First, relational star schemas are the most commonly used implementation method for multidimensional data warehouses and data marts. Second, as the other components are also based on Oracle, it makes the overall system simpler. Third, most RDBMSs, including Oracle, do now have special optimization techniques for processing SDB queries on star schemas. These optimizations include special join algorithms and the transparent use of pre-aggregated summary tables. This means that a good SDB performance can be expected even when an RDBMS, rather than a decicated multidimensional OLAP (MOLAP) DBMS, is used for the implementation.

5.7 Conclusion and Future Work

Motivated by the increasingly widespread use of OLAP technology, we have presented the concepts and techniques underlying a prototype system that logically integrates data in OLAP databases with data from outside object databases, without requiring physical integration of the data.

Summary data is best handled using OLAP technology, while complex detail-level data structures are best handled with object database technology. The enables the handling of the data using the most appropriate data model and technology, while still allowing queries to reference data across the different databases and data models. No attempt is made to map data into one common data model, which would be sub-optimal for some of the data. To our knowledge, this is the first example of a "multi-model" federation that includes a dedicated summary data model. We also believe this study to be the first that considers the impact on core OLAP concepts, e.g., summarizability, when federating with external data. In contrast to earlier works, the approach presented here uses the aggregation semantics of data to guard against meaningless or incorrect queries.

More specifically, as a vehicle for presenting the paper's contributions, a high-level language for summary databases, SumQL, has been introduced. This has then been extended to support queries that reference data in separate object databases. The resulting language, SumQL++ embodies the concept of *links* that connect an SDB to ODBs in a general and flexible way, in addition to object-oriented concepts. SumQL++ permits *selection criteria* that reference data in the ODBs using path expressions, facilities for *decorating* the aggregate results of SDB queries with external object data, and the ability to *group* data in the SDB according to object data. We have focused on the extension of aggregate queries over SDBs to also include data from ODBs. The formal semantics of SumQL++ is given in terms of a formal multidimensional data model and the ODMG data model and OQL query language. It is possible to use other languages such as SQL, OQL, and MDX in the place of SumQL++ once these are enriched with the necessary SumQL++ constructs that they do not already offer.

Interesting research directions include extending the approach to handle federations with several SDBs, as well as the federation with XML databases, which offer less structure than object databases and thus may benefit even more from the en-

forcement of aggregation semantics by the federation. Next, it would be of interest to investigate the dynamic restructuring of the OLAP schema, enabling the use of measures as dimensions and vice versa. Yet another interesting direction would be to consider the optimization of queries over the federation. For example, it may in some situations be advantageous to perform aggregation before selection, to take advantage of OLAP techniques such as pre-aggregation.

5.8 Formal Definition of SumQL

This sections formally defines the syntax and semantics of the SumQL language.

5.8.1 Syntax of SumQL

We now list the syntax for SumQL. The following notation is used in the syntax below: lower case letters are used for variable names; upper case letters are used for keywords; | denotes 'or'; [ ] is used to designate optional expressions. To save space, we have not included definitions of strings, reals, and integers, as their definitions are obvious.

```
select_query       ::= SELECT measure_list
                       INTO summary_database
                       BY_CATEGORY category_list
                       FROM summary_database
                       [ WHERE predicate_clause ]
measure_list       ::= measure | measure_list measure
measure            ::= string
summary_database   ::= string
category_list      ::= category | category_list category
category           ::= string
predicate_clause   ::= predicate_factor
                       | predicate_clauseboolean_op predicate_element
predicate_factor   ::= predicate_element | ( predicate_clause )
boolean_op         ::= AND | OR
predicate_element  ::= category_predicate | NOT category_predicate
category_predicate ::= category_exp predicate_op value
                       | category_exp BETWEEN (value, value)
                       | category_exp IN value_list
                       | category_exp MATCH ' string '
category_exp       ::= category
predicate_op       ::= = | != | > | >= | < | <=
value              ::= integer | real | ' string '
value_list         ::= value | value_list value
```

5.8.2 Semantics of SumQL

To describe the formal semantics of SumQL, we first specify a formal algebraic query language on the multidimensional data model. The algebraic query language is rather low-level and not for end-users, but is convenient for describing semantics. Next, we specify the semantics of SumQL by translation to the algebraic language. The algebraic language presented here is not meant to be computationally complete. We only include the operators that correspond to standard OLAP functions, such as aggregation and selection, while other operators such as union are left out. This is done purposefully, to make sure that the computational power of the language will not surpass that of any commercial OLAP tool, rendering the results presented here widely applicable to commercial OLAP tools.

selection: Given an SDB $S = (\mathcal{S}, D, M)$ and a predicate p on the dimension types $\mathcal{D} = \{\mathcal{T}_i\}$, we define the selection σ as: $\sigma[p](S) = (\mathcal{S}', D', M')$, where $\mathcal{S}' = \mathcal{S}$, $D' = D$, $M' = \{M'_i, i = 1, .., k\}$, $M'_i(e_1, .., e_n) = \text{if } (p(e_1, .., e_n)) \text{ then } M_i(e_1, .., e_n) \text{ else } null$. The aggregation types are not changed by the selection operator.

Thus, the schema and the dimensions are retained, while the measures are restricted to the part of the multidimensional space where predicate p holds.

Example 65 If selection is applied to the sample SDB with the predicate $Year = 1998$, the resulting SDB has the same schema and dimensions, but the Total Admissions measure is restricted to only return non-null values for the multidimensional combinations where the days $d \sqsubseteq_{Time} 1998$ and where the original measure returned non-null values for those combinations. All other combinations return the null value.

projection: Given an SDB $S = (\mathcal{S}, D, M)$, where $\mathcal{S} = (\mathcal{F}, \mathcal{D})$, a set of measures $M_{q_1}, .., M_{q_p} \in M$, and a set of dimension types $\{\mathcal{T}_{j_1}, .., \mathcal{T}_{j_m}\} \subseteq \mathcal{D}$ such that $\mathcal{T}_i = (\{\top_{\mathcal{T}_i}\}, identity, \top_{\mathcal{T}_i}, \top_{\mathcal{T}_i})$ for $i \notin \{j_1, .., j_m\}$, we define the projection π as: $\pi[\mathcal{T}_{j_1}, .., \mathcal{T}_{j_m}, M_{q_1}, .., M_{q_p}](S) = (\mathcal{S}', D', M')$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = \mathcal{F}$, $\mathcal{D}' = \{\mathcal{T}_{j_1}, .., \mathcal{T}_{j_m}\}$, $D' = \{D_i \in D \mid Type(D_i) \in \mathcal{D}'\}$, $M' = \{M'_i, i \in \{q_1, .., q_p\}\}$, and $M'_i(e_1, .., e_m) = M_i(e'_1, .., e'_n)$, where $e'_j = \text{if } (j \in \{j_1, .., j_m\}) \text{ then } e_j \text{ else } \top$. The aggregation types are not changed by the projection operator.

Thus, we require that the dimensions left out in the projection are "simple," having only the $\top$ categories. We then keep only the dimension types specified in the projection and their corresponding dimensions. The measures are modified to take only the remaining dimensions as arguments. Only the measures specified in the projection are kept. Note that we do not have to perform any other modifications (such as aggregation) on the measures, as the requirement on the dimensions left out makes sure that the measures have well-defined results, even when the number of dimensions is reduced.

Example 66 Imagine having a version of the example SDB, S', where the Reason and Time dimensions have only the $\top$ category. This could for instance be the result of aggregating along these dimensions (see the aggregation operator below). The result of a projection $\pi[Diagnosis, Place, TotalAdmissions](S')$ is the SDB where

Reason and Time are removed from the set of dimension types and dimensions, making the SDB 2-dimensional, and the new "Total Admissions" measure gives the same values for the combination (d, p) as the old measure gave for the combination $(d, p, \top, \top)$.

aggregation: Given an SDB $S = (\mathcal{S}, D, M)$ and a set of categories $C_1, .., C_n$ such that $C_i \in D_i, i = 1, .., n$, we define aggregation α as: $\alpha[C_1, .., C_n](S) = (\mathcal{S}', D', M')$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$, $\mathcal{F}' = \mathcal{F}$, $\mathcal{D}' = \{\mathcal{T}_i', i = 1, .., n\}$, $\mathcal{T}_i' = (\mathcal{C}_i', \sqsubseteq_{\mathcal{T}_i}', \top_{\mathcal{T}_i}', \bot_{\mathcal{T}_i}')$, $\mathcal{C}_i' = \{\mathcal{C}_{ij} \in \mathcal{T}_i \mid Type(C_i) \sqsubseteq_{\mathcal{T}_i} \mathcal{C}_{ij}\}$, $\sqsubseteq_{\mathcal{T}_i}' = \sqsubseteq_{\mathcal{T}_i|_{\mathcal{C}_i'}}$, $\bot_{\mathcal{T}_i}' = Type(C_i)$, $\top_{\mathcal{T}_i}' = \top_{\mathcal{T}_i}$, $D' = \{D_i', i = 1, .., n\}$, $D_i' = (C_i', \sqsubseteq_i')$, $C_i' = \{C_{ij}' \in D_i \mid Type(C_{ij}') \in \mathcal{C}_i'\}$, $\sqsubseteq_i' = \sqsubseteq_{i|_{D_i'}}$, $M' = \{M_i, i = 1, .., k\}$, $M_i'(e_1, .., e_n) = f_{M_i}(\{M_i(e_1', .., e_n') \mid e_1' \in \bot_{D_1} \wedge .. \wedge e_n' \in \bot_{D_n} \wedge e_1' \sqsubseteq_1 e_1 \wedge .. \wedge e_n' \sqsubseteq_n e_n\})$ (the set on the right-hand side of the last equation is a multi-set, or bag).

If the hierarchies up to the grouping categories are summarizable, the aggregation types for the new dimensions are the same as for the original. If one or more of the hierarchies in the dimensions being aggregated over are not summarizable, then the aggregation types for all dimensions are set to c, as no further aggregation should be based on the data.

Example 67 On the example SDB, S, we apply the operation $\alpha[Diagnosis, Hospital, \top_{Reason}, \top_{Time}](S)$, i.e., we aggregate over all of the Reason and Time dimensions, but not over the Diagnosis and Place dimensions. This gives us the SDB described in the previous example. To make the new SDB, for each (diagnosis,hospital) combination (di, h), we find the group of (diagnosis,hospital,reason,day) combinations (di, h, r, da) such that $r \sqsubseteq_{Reason} \top_{Reason}$ and $da \sqsubseteq_{Time} \top_{Time}$, i.e., all the 4-dimensional combinations that di and h are part of. For each (di, h, r, da), we apply the "Total Admissions" measure, M, to the combination to get the corresponding measure value. We store the measure values for each (di, h) combination in their own multiset, to which we apply the default aggregation operator, SUM. The measure values for the new "Total Admissions" measure, M' for a combination (di, h) is thus $M(di, h) = \text{SUM}_{r \sqsubseteq_{Reason} \top_{Reason}, da \sqsubseteq_{Time} \top_{Time}}(\{M(di, h, r, da)\})$, i.e., the sum over all the (di, h, r, da) combinations that (di, h) is a part of. Note that the set on the right-hand side of the equation is a multi-set, or bag.

We can now give the formal semantics of a SumQL statement in terms of the algebraic query language. The semantics are as follows. Given an SDB $S = (\mathcal{S}, D, M)$, categories $C_{j_1}, .., C_{j_m}$ in dimensions $D_{j_1}, .., D_{j_m}$ with dimension types $\mathcal{T}_{j_1}, .., \mathcal{T}_{j_m}$ and measures $M_1, .., M_p$ the result of the SumQL statement: SELECT $M_1, .., M_p$ INTO S' BY CATEGORY $C_{j_1}, .., C_{j_m}$ FROM S WHERE p is: $S' = \pi[\mathcal{T}_{j_1}, .., \mathcal{T}_{j_m}, M_1, .., M_p](\alpha[C_1, .., C_n](\sigma[p](S)))$, where $C_i = \text{if}(i \in \{j_1, .., j_m\})$ then C_{j_i} else $\top_{D_i}$.

Chapter 6

Summary of Conclusions and Future Research Directions

This chapter summarizes the conclusions and future research sections in Chapters 2 through 5 and Appendices A through C.

Motivated by the increasing use of multidimensional databases for data analysis in complex application areas such as health care, this thesis has investigated several aspects of data modeling and query processing for complex multidimensional data.

Chapter 2 investigated the exciting new challenges that data warehouse and OLAP technology faces from the area of clinical data warehousing. The challenges that are especially important to the general database research community included the following: advanced data models including temporal support, advanced classification structures, continuously valued data support, dimensional reduction of data, and integration of complex data.

Chapter 3 presented eleven requirements that multidimensional data models data should support to accommodate analysis of complex multidimensional data. Twelve previously proposed data models were evaluated against the eleven requirements, and it was seen that existing models did not support many-to-many relationships between facts and dimensions, did not have built-in mechanisms for handling change and time, lacked support for imprecision, and were unable to insert data with varying granularities. Additionally, most of the models did not support irregular dimension hierarchies and aggregation semantics. Chapter 3 presented an extended multidimensional data model and algebraic query language that addressed all eleven requirements. In particular, imprecise data was handled using the common multidimensional constructs of dimension hierarchies and granularities. The presented data model and query evaluation techniques could be implemented using standard OLAP technology and techniques such as RDBMSes and pre-aggregation.

Chapter 4 investigated the practical use of pre-aggregated data over irregular OLAP hierarchies. The scope of practical pre-aggregation was significantly extended to cover a much wider range of realistic situations. Specifically, algorithms were given that transformed irregular dimension hierarchies and fact-dimension relationships, which often occur in real-world OLAP applications, into well-behaved structures that, when used by existing OLAP systems, enabled practical pre-aggregation.

The algorithms had low computational complexity and could be applied incrementally to reduce the cost of updating OLAP structures. The transformations could be made transparently to the user.

Chapter 5 presented the concepts and techniques underlying a flexible, "multi-model" federated system for extending OLAP querying to external object databases. The system eased the integration of OLAP data with complex external data considerably and allowed data to be handled using the most appropriate data model and technology: OLAP systems for dimensional data and object database systems for more complex, general data. A prototypical OLAP language was defined and extended to naturally support queries that involve data in object databases. The language permitted selection criteria that referenced object data, queries that returned combinations of OLAP and object data, and queries that grouped dimensional data according to object data. The system was designed to be aggregation-safe, in the sense that it exploited the aggregation semantics of the data to prevent incorrect or meaningless query results. A prototype implementation of the system was reported.

Appendix A surveyed the field of clinical data warehousing, both from an industrial and an academic point of view, as of Mid 1997. Seven evaluation criteria were presented and nine products and projects were evaluated against them, giving a good overview of the (by then) current state of the art. The field was still in it's infancy, but the potential for clinical benefits of the technology was large. However, the products surveyed did not address several advanced requirements for clinical use, including richer data models, temporal support, and intelligent integration of complex data.

Appendix B presented the TreeScape system that, unlike any other system known to the authors, enables the reuse of pre-computed aggregate query results for irregular dimension hierarchies, which occur frequently in practice. The system established a foundation for obtaining high query processing performance while pre-computing only limited aggregates, even when the hierarchies were irregular. This was done using the dimension transformation techniques described in Chapter 4. It was described how the transformations could be made transparent to the user using a query re-write mechanism.

Appendix C presented the OLAP++ system for federating OLAP and object databases. The system allowed users to easily pose OLAP queries that reference external object databases. This enabled very flexible and fast integration of object data in OLAP systems without the need for prior physical integration. It was shown how the user interface allowed the user to easily specify OLAP queries over the federation and how these queries were processed.

The work reported in this thesis may be continued in several directions in the future.

For the presented data model, it should be investigated how the model and query handling techniques may be efficiently implemented using special-purpose algorithms and data structures, to achieve optimal concrete complexity. Next, a notion of completeness for multidimensional algebras, similar to Codd's relational completeness would be an exciting research topic. We also believe that it is important to investigate how multidimensional models can cope with the hundreds of dimensions found in some applications.

There are several future research directions related to imprecision. The investigation of the issues related to "single-value" aggregation functions such as MIN and MAX, that are not readily sensitive to weighting, in relation to data granularity is warranted. It would also be interesting to explore other means of graphically presenting imprecision in the result to facilitate the user interpretation of an imprecise result, or to possibly present the user with the data that *prevented* a given query from being precisely answerable. Also, it would be good to give precise measures for the usefulness of technique, given the available data.

The presented normalization approach for enabling practical pre-aggregation has several promising directions for future research. The current techniques render the entire dimension hierarchies summarizable; extending the techniques to consider only the parts that have been selected for pre-aggregation appears attractive and possible. Another direction is to take into account the different types of aggregate functions to be applied, leading to local relaxation of the summarizability requirement. For example, *max* and *min* are insensitive to duplicate values, thus relaxing summarizability.

For the federation approach presented in this thesis, interesting research directions include extending the approach to handle federations with several SDBs, as well as the federation with XML databases, which offer less structure than object databases and thus may benefit even more from the enforcement of aggregation semantics by the federation. Next, it would be of interest to investigate the dynamic restructuring of the OLAP schema, enabling the use of measures as dimensions and vice versa. Yet another interesting direction would be to consider the optimization of queries over the federation. For example, it may in some situations be advantageous to perform aggregation before selection, to take advantage of OLAP techniques such as pre-aggregation.

Bibliography

[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, S. Sarawagi. Modeling Multidimensional Databases. *IBM Technical Report 1995*. Also appeared in *Proceedings of the Thirteenth International Conference on Data Engineering*, pp. 232–243, 1997.

[3] E. B. Baatz. Return on Investment - What's It Worth. *CIO Magazine* October 1, 1996

[4] E. Baralis, S. Paraboschi, and E. Teniente. Materialized View Selection in a Multidimensional Database. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pp. 156–165, 1997.

[5] 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.

[6] T. Barsalou and D. Gangopadhyay. M(DM): An Open Framework for Interoperation of Multimodel Multidatabase Systems. In *Proceedings of the Eighth International Conference on Data Engineering*, pp. 218–227, 1992.

[7] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys* 18(4):323–364, 1986.

[8] 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.

[9] R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Systematic Change Management in Dimensional Data Warehousing. In *Proceedings of the Third International Baltic Workshop on DB and IS*, pp. 27–41, 1998.

[10] T. A. Budd. *Multiparadigm Programming in Leda*. Addison-Wesley, 1995.

[11] 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.

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

[13] R. G. G. Cattell et al. (editors). *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.

[14] 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.

[15] I-M. A. Chen, V. M. Markowitz. An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools. *Information Systems*, 20(5): 393-418, 1995.

[16] I. A. Chen and V. M. Markowitz. The Object-Protocol Model (OPM) Version 4.1. Lawrence Berkeley National Laboratory Technical Report LBNL-32738 (revised), 1996.

[17] I. A. Chen, A. Kosky, V. M. Markowitz, and E. Szeto. The OPM Query Language and Translator - Version 4.1. Lawrence Berkeley National Laboratory Technical Report LBNL-38180, 1996.

[18] I. A. Chen. A. Kosky, V. M. Markowitz, and E. Szeto. OPM*QS: The Object-Protocol Model Multidatabase Query System. Lawrence Berkeley National Laboratory Technical Report LBNL-38181, 1996.

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

[20] 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.

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

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

[23] Cohera Corporation. Cohera Data Federation System. <www.cohera.-com/datasheets.html>. Current as of February 18, 1999.

[24] Commission of the European Communities, Directorate General XII. *Synapses: Federated Healthcare Record Server*. Contract no. HC 1046 (HC).

[25] S. Dar, H. V. Jagadish, A. Y. Levy, and D. Srivastava. Answering SQL Queries Using Views. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pp. 318–329, 1996.

[26] A. Datta and H. Thomas. A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases. In *Proceedings of the Seventh Annual Workshop on Information Technologies and Systems*, 1997.

[27] P. M. Deshpande, J. F. Naughton, K. Ramasamy, A. Shukla, K. Tufte, and Y. Zhao. Cubing Algorithms, Storage Estimation, and Storage and Processing Alternatives for OLAP. *IEEE Data Engineering Bulletin*, 20(1):3–11, 1997.

[28] B. A. Devlin and P. T. Murphy. An Architecture for a Business and Information System. *IBM Systems Journal*. Vol. 27, No. 1, 1988

[29] R. Domenig and K. Dittrich. An Overview and Classification of Mediated Query Systems. *ACM SIGMOD Record*, 28(3), 1999.

[30] Duke University Medical Informatics Department. Health Level 7 (HL7) WWW page. <http://www.mcis.duke.edu/standards/HL7/hl7.htm>. Current as of February 26, 2000.

[31] 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.

[32] C. E. Dyreson. A Bibliography on Uncertainty Management in Information Systems. In A. Motro and P. Smets (Eds.). *Uncertainty Management in Information Systems - From Needs to Solutions*, pp. 413–458, Kluwer Academic Publishers, 1997.

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

[34] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*, 2nd edition. Benjamin/Cummings Publishing Company, 1994.

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

[36] European Committee for Standardization (CEN). *European Prestandard - prENV 12265. Medical Informatics - Electronic Healthcare Record Architecture.*, 1995.

[37] 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.

[38] Genelogic Corporation. Genelogic Home Page. <www.genelogic.com>. Current as of February 26, 2000.

[39] F. Gingras, Laks V. S. Lakshmanan: nD-SQL: A Multi-Dimensional Language for Interoperability and OLAP. 134-145, Electronic Edition

[40] 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.

[41] J. Gu, T. B. Pedersen, and A. Shoshani. OLAP++: Powerful and Easy-to-Use Federations of OLAP and Object Databases. *Demo proposal, submitted for conference publication*, 2000.

[42] J. Gu, T. B. Pedersen, and A. Shoshani. The OLAP++ System for Federating OLAP and Object Databases. *Demo proposal, submitted for conference publication*, 2000.

[43] 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.

[44] A. Gupta, V. Harinarayan, and D. Quass. Aggregate Query Processing in Data Warehousing Environments. In *Proceedings of the Twenty-First International Conference on Very Large Data Bases*, pp. 358–369, 1995.

[45] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index Selection for OLAP. In *Proceedings of he Thirteenth International Conference on Data Engineering*, pp. 208–219, 1997.

[46] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proceedings of the Sixth International Conference on Database Theory*, pp. 98–112, 1997.

[47] H. Gupta and I.S. Mumick. Selection of Views to Materialize Under a Maintenance-Time Constraint. In *Proceedings of the Seventh International Conference on Database Theory*, pp. 453–470, 1999.

[48] M. Gyssens and L. V. S. Lakshmanan. A Foundation for Multi-Dimensional Databases. In *Proceedings of the 23rd Conference on Very Large Databases*, pp. 106–115, 1997.

[49] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing Data Cubes Efficiently. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pp. 205–216, 1996.

[50] D. K. Hsiao and M. N. Kamel. The Multimodel, Multilingual Approach to Interoperability of Multidatabase Systems. In *Proceedings of the First International Workshop on Research Issues on Data Engineering*, 1991.

[51] D. K. Hsiao. Federated Databases and Systems: Part I - A Tutorial on Their Data Sharing. *VLDB Journal*, 1(1): 127–179, 1992.

[52] D. K. Hsiao. Federated Databases and Systems: Part II - A Tutorial on Their Resource Consolidation. *VLDB Journal*, 1(2): 285–310, 1992.

[53] Hyperion Corporation. Hyperion Essbase OLAP Server. <www.hyperion-.com/downloads/essbaseolap.pdf>. Current as of February 26, 2000.

[54] IBM Corporation. DB2 DataJoiner. <www-4.ibm.com/software/data/datajoiner/>. Current as of February 24, 2000.

[55] Information Architects Corporation (now InfoMedtrics Corporation). Products/Services. <www.infomedtrics.com/html/products/index.html>. Current as of February 26, 2000.

[56] Informix Corporation. Informix MetaCube ROLAP Option Overview. <www.informix.com/informix/products/options/mcro/overview/mcbroweb .htm>. Current as of February 26, 2000.

[57] W. H. Inmon. *Building the Operational Data Store.* John Wiley & Sons, 1995.

[58] W. H. Inmon. Building the Data Warehouse, 2nd Ed. *Wiley Computer Publishing* 1996.

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

[60] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses? In *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases*, pp. 530–541, 1999.

[61] 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.

[62] C. S. Jensen and R. T. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, 21(4):311–352, March 1996.

[63] C. S. Jensen and C. E. Dyreson, (editors). A Consensus Glossary of Temporal Database Concepts - February 1998 Version. In O. Etzion, S. Jajodia, and S. Sripada (editors). *Temporal Databases: Research and Practice*. LNCS 1399, Springer-Verlag, pp. 367–405, 1998.

[64] R. Kimball. *The Data Warehouse Toolkit*. Wiley Computer Publishing, 1996.

[65] R. Kimball. Data Warehouse Architect: Help with Multi-Valued Dimension. *DBMS Magazine*, 11(9), 1998.

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

[67] Kommunedata Corporation. *GS-Open Design Specification* (in Danish). Internal document, 1996.

[68] 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.

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

[70] 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.

[71] 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.

[72] V. M. Markowitz et al. OPM Home Page. <gizmo.lbl.gov/DM_ TOOLS/DMTools.html>. Current as of February 26, 2000.

[73] MedAI Corporation. Clinical Decision Support Services. <www.medai.com/ Products_Services.html>. Current as of February 26, 2000.

[74] Medical Records Institute. Applications for Electronic Patient Record Systems. <www.medrecinst.com/caregiver/applications.shtml>. Current as of February 26, 2000.

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

[76] Meta Group Corporation. 1999 Data Warehouse Marketing Trends/Opportunities - An In-Depth Analysis of Key Market Trends. *Meta Group*, 1999.

[77] Microsoft Corporation. ActiveX for Healthcare Committee. <www.mshug .org/ahc/>. Current as of February 26, 2000.

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

[79] Microsoft Corporation. OLAP Services White Paper. <www.microsoft.com/ sql/productinfo/ olapservices.htm>. Current as of February 26, 2000.

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

[81] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pp. 100–111, 1997.

[82] L. H. Nielsen. Clinical Aspects of the Concept Electronic Record in the Danish Health Service (in Danish). Project report, Health Informatics Education, Aalborg University, Denmark, 1996.

[83] Niinimäki J, et al. Medical Data Warehouse, an Investment for Better Medical Care. *In Proceedings of Medical Informatics Europe 1996*, IOS Press 1996.

[84] Object Management Group. CORBAmed WWW page. <www.omg .org/corbamed>. Current as of April 13th, 1998.

[85] The OLAP Council. *MDAPI Specification Version 2.0*. OLAP Council Technical Document, 1998.

[86] The OLAP Report. *Database Explosion*. <www.olapreport.com/Database-Explosion.htm>. Current as of February 26, 2000.

[87] F. Olken, D. Rotem. Random Sampling from Databases - A Survey. *Statistics & Computing*, 5(1):25–42, March 1995.

[88] Oracle Corporation. Oracle Gateways <www.oracle.com/gateways>. Current as of February 24, 2000.

[89] Oracle Corporation. Clinical Computing Plc. <alliance.oracle.com/catdoc/html/p18409.htm>. Current as of February 26, 2000.

[90] Oracle Corporation. Oracle Unveils Clinical Applications. <www.oracle .com/corporate/press/html/PR011397.110413.html> Current as of February 26, 2000.

[91] Palo Alto Management Group Corporation. 1999 Business Intelligence and Data Warehousing Program Competitive Analysis Report. *Palo Alto Management Group*, 1999.

[92] T. B. Pedersen and C. S. Jensen. Clinical Data Warehousing - A Survey (short version). *Presented at the Healthcare Computing 1998 Conference.*

[93] T. B. Pedersen and C. S. Jensen. Clinical Data Warehousing - A Survey (long version, corresponds to Appendix A). In *Proceedings of the VIII Mediterannean Conference on Medical and Biological Engineering and Computing*, Section 20.3 (CDROM proceedings), 1998.

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

[95] 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 Report TR-37, <www.cs.auc.dk/TimeCenter>, 1998.

[96] 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 Statistical and Scientific Database Management*, pp. 90–101, 1999. Extended version available as Technical Report TR-99-5003, Department of Computer Science, Aalborg University, <www.cs.auc.dk/TR5003.ps>, 1999.

[97] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation for On-Line Analytical Processing. In *Proceedings of the Twenty-Fifth International Conference on Very Large Databases*, pp. 663–674, 1999. Extended version available as Technical Report TR-99-5004, Department of Computer Science, Aalborg University, <www.cs.auc.dk/XX5004.ps>, 1999.

[98] T. B. Pedersen, A. Shoshani, J. Gu, and C. S. Jensen. Extending OLAP Querying to Object Databases. *Submitted for conference publication*, 2000.

[99] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Manuscript, to be submitted for journal publication*, 2000.

[100] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. The TreeScape System: Reuse of Pre-Computed Aggregates over Irregular OLAP Hierarchies. *Demo proposal, submitted for conference publication*, 2000.

[101] D. Quass and J. Widom. On-Line Warehouse View Maintenance for Batch Updates. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pp. 393–404, 1997.

[102] 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*, pp. 264–272, 1983.

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

[104] Rational Corporation. UML 1.1 Notation Guide. <www.rational.com/uml/resources/documentation/notation/index.jtmpl>. Current as of February 26, 2000.

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

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

[107] 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.

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

[109] SAS Institute Corporation. SAS Institute Announces the SAS Pharmatechnology Process. <www.sas.com/new/preleases/050797/news1.html>. Current as of February 26, 2000.

[110] SAS Institute Corporation. Health Care. <www.sas.com/software/industry/healthcare.html>. Current as of February 26, 2000.

[111] SAS Institute Corporation. Release of PH.Interface to Oracle Clinical. <www.sas.com/software/ industry/pht/headlines/OrClinInter.html>. Current as of February 26, 2000.

[112] A. Segev and J. L. Zhao. Selective View Materialization in Data Warehousing Systems. <ftp://segev.lbl.gov/pub/LBL_DB_PUBLICATIONS/1997/aggreg-dw.ps>. Current as of February 9, 1999.

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

[114] Shared Medical Systems Corporation. NOVIUS. <www.smed.com/ solutions/products/ novius.htm>. Current as of February 26, 2000.

[115] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *Computing Surveys*, 22(3):183-236, 1990.

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

[117] A. Shukla, P. M. Deshpande, J. F. Naughton, and K. Ramasamy. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pp. 522–531, 1996.

[118] K. Skifjeld. From concept to product - experiences from the development of the DocuLive EPR system. *The British Journal of Healthcare Computing & Information Management*, March 1997.

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

[120] Stanford University Medical Informatics Department. Integrating Heterogeneous Databases. <www-smi.stanford.edu/projects/helix/hetero.html>. Current as of February 26, 2000.

[121] Stanford University Medical Informatics Department. The EON Project. <www-smi.stanford.edu/projects/eon/index.html>. Current as of February 26, 2000.

[122] Stanford University Medical Informatics Department. Tzolkin Temporal Data Management System. <www-smi.stanford.edu/projects/eon/tzolken.html>. Current as of February 26, 2000.

[123] Sybase Corporation. Success Stories: Quest Informatics. <success.sybase .com/success/docitem.stm?content_id=1000167>. Current as of February 26, 2000.

[124] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pp. 126–135, 1997.

[125] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. Wiley, 1997.

[126] Transaction Processing Council. The TPC-R Benchmark. <www.tpc.org>. Current as of February 21, 1999.

[127] 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.

[128] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. In *SIGMOD Record*, 28(4):64–69, 1999.

[129] J. Widom (editor). Special Issue on Materialized Views and Data Warehousing. *IEEE Data Engineering Bulletin*, 18(2), 1995.

[130] J. Widom. Research Problems in Data Warehousing. In *Proceedings of the Fourth International Conference on Information and Knowledge Management.*, pp. 25–30, 1995.

[131] R. Winter. Databases: Back in the OLAP game. *Intelligent Enterprise Magazine*, 1(4):60–64, 1998.

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

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

[134] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0. W3C Recommendation, 1998. <www.w3.org/TR/1998/REC-xml-19980210>. Current as of February 26, 2000.

[135] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in a data warehousing environment. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pp. 136–145, 1997.

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

Appendix A

Clinical Data Warehousing — A Survey

A.1 Introduction

The concept of data warehousing has taken the computer industry by storm in the recent years, as enterprises have realized the enormous opportunities in extracting useful information from the data "hidden" in their computer systems. The new functionality offered by data warehousing has traditionally been used in business, in areas such as retail and finance, but the technology is now increasingly being used in more "scientific" areas.

One of these areas is clinical, where clinical data about a large patient population is analyzed to perform clinical quality management and medical research [3]. The use of data warehousing in the clinical area will be driven by the *need* to manage the entire care process to stay competitive, as well as the *opportunities* for gaining new insights by actively *using* the enormous amount of patient data available. Data warehousing stands out as the only *viable* technology for realizing the full information potential in operational data. Thus, clinical data warehousing will become very important in clinical enterprises in the not-too-distant future. So far, adoption of the technology in healthcare has been hindered by lack of knowledge on what data warehousing can be used for, and what current products offer. This chapter offers the needed knowledge by describing the current state-of-the-art of products and other efforts, providing evaluation criteria for comparing clinical data warehouse systems, and looking at future directions of the area.

This work was done as part of a clinical data warehouse research project, involving university computer scientists, industry, and clinicians engaged in a joint effort to determine how the data warehouse concept should be used and extended to support the needs of clinical users.

In Section 2, we define the concept of data warehousing and provide evaluation criteria for clinical data warehouse systems. In Section 3, we describe the various efforts in the area, and list their conformance to the criteria. Section 4 discusses the relative merits of the efforts, and offers a look at the future, as well as concluding remarks.

A.2 Data Warehousing

The term "Data Warehouse" (DW) was first used by Barry Devlin [28], but it is Bill Inmon that has won the most acclaim for introducing the concept, see Figure A.1 for the basic architecture of a DW.

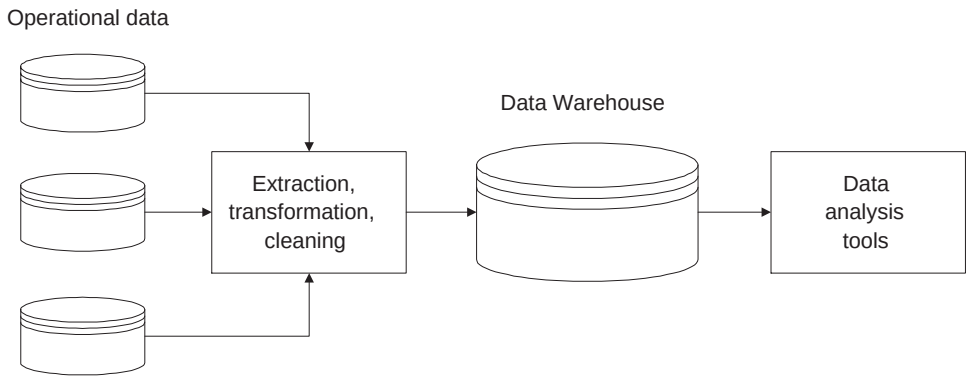

Figure A.1: Integration of Operational Data in the Data Warehouse

He defines a Data Warehouse as: "A Data Warehouse is a *subject oriented, integrated, non-volatile* and *time-variant* collection of data in support of *management's decisions*." [58] Let us have a closer look at these defining properties.

Data is organized by *subjects*, such as patients, to allow the users to work with terms from their daily life. In operational systems, data is organized to support a particular application, e.g., a laboratory system, often making data incomprehensible to humans. The subject orientation of a DW makes it much easier to understand the data.

Data is *integrated* from multiple operational systems, both by definition and content. As seen in Figure A.1, data is extracted from the operational systems, transformed into a format suited for data analysis, and "cleaned" to adjust for invalid, incompatible, or missing data. This makes the previously very hard task of combining data from different systems a lot easier for the end user.

Data is *non-volatile*, i.e., it is kept for many years and sometimes never deleted. In operational systems, data is often deleted after a few months, when they are no longer needed by the particular application. Retaining the data makes analysis over long time periods possible.

Data is *time-variant*, i.e., it always has a notion of time attached to it, and often a complete history of changes is kept. This makes analysis for trends over time possible. In operational systems, data is often not related to time, and only the newest version of the data is stored.

Data in a DW support *management's decisions*, i.e., it is optimized for data analysis, not data entry. Data is used to *understand and manage* the enterprise, both at a strategic and a tactical level. Examples of this include clinical quality management and medical research.

Compared to an ordinary clinical decision support systems (CDSS), the DW has a much broader scope. Typically CDSS's are specialized for one very specific purpose, with an emphasis on a deep level of functionality. There is a very limited possibility of asking "new" questions. On the other hand, the DW has an emphasis on the data itself, providing a possibility to combine all the kinds of data in the enterprise, and thus more easily answer unanticipated questions. Instead of just providing a set of fixed reports, an explorative way of working with the data is encouraged and supported.

To allow for comparison of the systems to be covered in Section 3, we list seven different criteria, that a full-blown clinical DW should meet in order to provide full business value for the user.

The system should be *open*, i.e., it should allow integration with systems or components from other vendors. This is very important, as systems without this ability will lock users into a proprietary, non-extensible solution, which almost surely will not support all their needs.

The system should have features for importing data from *external systems*, e.g., laboratory systems, into the DW. Without this feature, the system cannot be considered a "true" DW, as users will have to reenter data into the DW system to gain advantage of it. This will mean both extra work and more data errors, which will severely limit the usefulness of the system.

A full-blown clinical DW should support *all the types of data* important to the healthcare enterprise, including financial data (F) such as billing and contracts, demographic data (D) such as age and sex, clinical data (C) such as diagnoses and procedures, numerical data (N) such as lab results, and image data (I) such as x-rays. If all kinds of data are not supported in the DW system, the possibility of combining data to gain new knowledge will be severely limited, thus diminishing the business value of the DW.

Next, a clinical DW system should support data analysis at several levels. The lowest level is the *patient* level, where data about the individual patient can be viewed and analyzed, e.g., to find a pattern in the development of a disease for a particular patient. This level of analysis focuses on giving the particular patient the best possible treatment, and is thus important for the *practice* of care. The next level is the *group* level, where data about a group of patients, e.g., patients having a particular disease, is analyzed. One application of this is clinical quality management, where treatments and outcomes are analyzed and compared to norms in order to identify how the care process can be enhanced. This level focuses on *medical research* and *care improvement*, and is thus important from a more *scientific* point of view. The top level is that of the healthcare *enterprise*, where clinical, financial, and demographic data are combined to investigate the profitability and overall quality of the services provided. This level of analysis focuses on overall performance of the enterprise and is thus important from a *management* perspective.

We also list if the systems have any particular *advanced features* that makes them stand out from the rest such as support for drug development or predefined disease studies. These features may be very important for some users, e.g., the pharmaceutical industry, while others have different demands.

A.3 Clinical Data Warehousing Systems

We will now examine the current state-of-the-art of clinical data warehousing by describing the most important efforts we have encountered so far. The first five sections describe commercial data warehouse products, while last three describe specific clinical data warehouse projects. The list is not meant to be exhaustive, but we believe that it is representative for the current state of affairs. For all DW application areas, most of the work has been done in industry, rather than in scientific environments. Compared to using DW for business purposes, clinical data warehousing is still in its infancy, with only a few providers and no wide-spread use. It is, however, recognized as an important and emerging field presenting tough challenges [3].

	Open	Ext. Data	Datatypes	Patient	Group	Enterprise	Advanced Features
CC	No	Yes	C,N	Yes	Yes	No	Collaborative info
OC	Yes	No	C,N	Yes	Yes	No	Drug development
SAS	Yes	Yes	C,N	Yes	Yes	No	Drug development
Ma	Yes	Yes	F,C,N	Yes	Yes	No	Disease studies
IAI	Yes	Yes	F,D,C,N	Yes	Yes	Yes	Longitudinal studies
SMS	Yes	Yes	F,D,C,N	Yes	Yes	Yes	Rules Engine
QI	No	No	C,N	No	Yes	No	Very large database
TUCH	No	No	C	No	Yes	No	None
SMI	No	Yes	C,N	No	No	No	Temporal, protocols

CC: Clinical Computing; OC: Oracle Clinical; SAS: SAS Institute; Ma: MEDai; IAI: Information Architects Inc.; SMS: Shared Medical Systems; QI: Quest Informatics; TUCH: Turku University Central Hospital; SMI: Stanford Medical Informatics

Table A.1: Comparison of Clinical DW Systems

A.3.1 Oracle and Partners

As one of the major DW players, with an extensive consulting business and a great number of partners, the Oracle corporation has been involved in a lot of DW projects, including some of the clinical variety.

Clinical Computing [89], an Oracle partner, provides the di-Proton/Clinical Data Warehouse solution that is aimed at renal, i.e., kidney function, information management. The management of information such as treatment assessments, dialysis equipment checks, interdialytic vital signs, complications, procedures, medications, and infection history is supported. This enables analysis of core clinical indicators and outcomes. Interfaces to laboratory systems and dialysis machines facilitates the data collection process. An interesting feature is that collaborative care information such as nutritional recommendations, psychosocial assessments, etc., is also recorded, providing a larger picture of the patient's status. Table A.1 summarizes the evaluation of each DW effort covered in this section according to the criteria given in Section 2.

Recently, Oracle Inc. itself ventured into the clinical world with the release of Oracle Clinical [90], the first in the Oracle Pharmaceutical suite of applications. The product is meant to address the needs of large, pharmaceutical companies for man-

aging information about the extensive clinical trials that are needed to test a new drug. This involves meeting strict government regulations as well as facing hard competition. Previous, the companies had to build their own applications for each series of trials, resulting in very high costs and longer development cycles. The product is already in use at several major companies, including Boehringer IngelHeim, Genentech, and Hoffmann-La Roche. The core product will later be supplemented by products supporting adverse event handling, remote data entry, and data analysis, the objective being to provide a complete turn-key solution for the industry.

A.3.2 SAS Institute

Another major player in the DW market, SAS Institute, a long-time champion in the data analysis marketplace, has recently introduced the SAS Pharmatechnology Process [109], a clinical data warehouse framework for the specific needs of clinical research. The cornerstone in the framework is SAS/PH-Clinical, a software system for assimilating and reviewing data from clinical trials.

The product already supports the review process, and is being further developed to support all the processes involved in developing a new drug, such as laboratory analysis and pharmacokinetics. The product allows the clinical trials data to be viewed in spreadsheet or graphical form. The patient population may be subsetted based on specific attributes, test results or the treatment protocol used, and the subset may be compared to other subsets or the complete patient population. The data is usually displayed in summary form, but the user may drill down to watch the complete patient history, e.g., when an anomaly occurs. Both ad-hoc data exploration and standardized reporting is supported.

The system is not a complete clinical data warehouse in itself, but it may be used with SAS Institute's Data Warehouse Administrator to build such a solution. Indeed, a clinical data warehouse is a very central part in SAS Institute's visions for the use of information in the health care industry [110]. These include data warehousing, continuous quality improvement, outcomes management, health plan management, and utilization analysis. Especially interesting on the clinical side is the focus on using data mining and OLAP techniques to identify key clinical indicators, thus improving the quality of care.

SAS Institute has several partners in this area, including Xerox for document integration, and Oracle for integration with the Oracle Clinical software [111]. It is possible to map views defined in Oracle Clinical into the study definitions that are required by SAS/PH-Clinical in a reasonably straightforward way, transferring both data and metadata from Oracle Clinical. This allows the rich data analysis features offered by the SAS System to be used for clinical data analysis.

A.3.3 MEDai

MEDai is a company focused on utilizing Artificial Intelligence (AI) techniques for health care data analysis. They provide the Clinical Decision Support System (CDSS) [73], which is a data warehouse/clinical data repository with powerful analytical capabilities. The CDSS system can extract data from existing hospital systems to pro-

vide both clinical and financial data. It allows for comparison of performance to norms, both at the facility and physician level. It also provides outcome analysis and has facilities for the development of treatment protocols. AI techniques are used for severity/risk adjustment for the patients, and data mining and drill down capabilities allow for data exploration. A distinguishing feature is the more than 20 predefined "disease studies" for data analysis, covering more than 80 percent of normal admissions. These include pneumonia, cesarean sections, chest pain/coronary artery disease, HIV, asthma, diabetes mellitus, and migraine. This makes it possible to immediately perform data analysis of the most common diseases.

A.3.4 Information Architects Inc.

In a data warehouse, different types of data are integrated to get a complete view of an enterprise. This is what Information Architects Inc. (IAI) makes possible with it's "healthcare information warehouse" product [55]. In this system, administrative, financial, and clinical data are integrated to provide a foundation for measuring both cost and value of the services delivered in the healthcare delivery process. The data warehouse consists of more than 200 tables, with integrated desktop reporting and server-side tools. It supports quality-of-care reporting, provider comparisons, outcome analysis using advanced statistics, and longitudinal health studies. Categories of care and treatment groups are supported for summary information, with associated norms for comparison of performance. The system is highly scalable, ranging from PC servers to Massively Parallel Processor (MPP) machines.

A.3.5 Shared Medical Systems

Shared Medical Systems (SMS) have a long history in the healthcare informatics business. Their product line Novius.ihn [114] is aimed at the Integrated Health Network (IHN) market, and has several interesting features. It has an integrated DW that standardizes, stores, and manages demographic, financial, and clinical data from across the IHN. A common vocabulary engine allows for the definition of terms and relationships, which can then be used for the definition of clinical protocols. The rules engine allows for transformation and abstraction of data. Relational data structures supporting analysis over time and aggregation are provided. The product has an integrated management solutions component for strategic and tactical analysis. A quality management component providing study definition, indicator derivation, and statistical analysis is available as an option. The DW is well integrated with the wide range of clinical operational system that SMS offers and has support for receiving data in many formats, including EDI. This provides for easy acquisition of quality DW data.

A.3.6 Quest Informatics

The clinical data warehouse run by Quest Informatics [123] may well be the currently largest clinical data repository. Each week, 20 million new test results are loaded, and the system is predicted to break the terabyte (1000 GB) barrier in the near future, making it a very large DW by any standard. The system is used by Quest Informatics

to turn the vast amount of lab results received from Quest Diagnostics Incorporated into valuable knowledge. This information is then employed by the users of Quest to improve their healthcare delivery. This is done by offering summary data, including comparisons between the users' patients and a standard patient population, thus identifying the broad areas of improvement. The user can then drill down to more detailed levels, getting the specific information about where improvements may be possible. The demands to the system are very tough; it must be very flexible to meet changing customer requirements, while still being able to perform effectively on the vast amounts of data.

A.3.7 Turku University Central Hospital

One of the only European efforts in the area is reported by Turku University Central Hospital in Finland [83]. The system integrates data from several hospital and laboratory information systems to provide a broad view of clinical data suitable for research. The system is comprised of a data transportation tool, a data warehouse database, and a proprietary front-end query tool. The DW has primarily been used in two studies, one on drug-laboratory interference and one on drug-drug interactions. The first study is concerned with the interference of drug prescription with the thyrotropin (TSH) test and determines how often drugs affecting the TSH test are prescribed. In fact, 11.6% of drug prescriptions turned out to interfere with the TSH test. The second study concerned drug interactions for nephrological patients, and the results were used in developing a drug interaction reminder system. An interesting issue in clinical data warehousing is noted, namely that data protection is very important due to the sensitive nature of clinical data, as opposed to typical business data warehouses. The project concludes that the presence of a clinical data warehouse is very facilitating for clinical research, as it provides an easy way to get precise answers to questions that otherwise often would not even have been asked due to the difficulties in getting to the data.

A.3.8 Stanford Medical Informatics

The efforts done at the Department of Medical Informatics at Stanford University are highly relevant to clinical data warehousing, although their approach to clinical data analysis is not strictly in the DW category.

An important aspect in any DW is integration of heterogeneous source data. This problem has been treated by the TransFER [120] project, which has developed a method for a query on a single, integrated, global schema to be translated to queries against the relevant local schemas, combining the results in the end. However, unlike the data warehouse approach, data in the global schema is not materialized, but only kept in the local databases.

Support for clinical treatment protocols is very important in clinical information systems. The EON project [121] has studied how to formally represent and reason about clinical protocols. A domain model of clinical concepts, suitable for protocol-based care has been developed, along with appropriate problem-solving methods. Representation and reasoning about *time* plays a very central role in clinical systems,

and as part of the EON project, the Tzolkin Temporal Data Management System [122] has been developed to allow temporal abstractions of raw clinical data, i.e., identify the time periods when certain clinical generalizations are true.

A.4 Discussion and Summary

In this section, we discuss the relative merits of the products and projects covered in Section 3, look at future prospects, and offer concluding remarks.

The DW at Turku University solves a specific problem, but does not have general applicability, and is thus not interesting for the general customer. Clinical Computing offers a good system, but it is limited to the renal domain. Quest Informatics has an impressive system in terms of data volume, but it is still a proprietary system designed specially for the company. Oracle Clinical and SAS/PH-Clinical both are feature-rich products targeted at an entire industry. Especially the SAS product seems to offer a high level of functionality. Their drawback is that they are not optimized for ordinary clinical functions, but rather for the highly specialized process of pharmaceutical drug development. The work at Stanford Medical Informatics is very interesting and touches on many of the core issues in clinical data warehousing. However, data warehousing as such is not treated, and no products are offered. It is the systems from MEDai, IAI, and SMS that are the most interesting. MEDai employs advanced AI techniques for data analysis, including predefined disease studies. SMS offers the widest range of clinical operational systems, and thus achieves very good integration with operational data, along with advanced functionality such as support for protocols and rules. IAI perhaps has the most "true DW" offering, with their integration of administrative, financial, and clinical information, supported by integrated analysis tools. Selecting one system out of these three must be based on the particular needs of the customer.

The future of clinical data warehousing looks very bright indeed, provided that the systems of tomorrow can fulfill the real needs of clinicians for data analysis, most of which are not well supported by current commercial products. The needs include a richer data model for capturing more of the semantics of the data, advanced temporal support to allow for analysis of data that change over time, support for advanced queries on continuously valued data, e.g., advanced statistics, and intelligent integration of very complex data, e.g., x-rays, in the DW for analysis purposes. Support for reducing the complexity of the data while still maintaining the essence is also very much needed, as well as means of handling the advanced classification structures employed in medicine. The concepts of clinical treatment protocols and medical research should also be tightly integrated into the clinical DW.

To summarize, we have introduced the concept of a data warehouse and described how it is used in a clinical setting. We have described the efforts in the area of clinical data warehousing and seen what products might be interesting for the general clinical customer. We think that the use of data warehouses in clinical settings will explode in the coming years, as systems mature and the clinicians realize the potential of using their data for quality improvement and research. We are currently working

to meet some of the important challenges to data warehousing provided by clinical applications.

Appendix B

The TreeScape System: Reuse of Pre-Computed Aggregates over Irregular OLAP Hierarchies

B.1 Introduction

In order to improve query performance, modern On-Line Analytical Processing (OLAP) systems use a technique known as *practical pre-aggregation*, where *select* combinations of aggregate queries are materialized and re-used when computing other aggregates; full pre-aggregation, where all combinations of aggregates are materialized, is infeasible, as it typically causes a blowup in storage requirements of 200–500 times the size of the raw data [86, 117]. Normally, practical pre-aggregation requires the dimension hierarchies to be regular, i.e., to be balanced trees, but this is quite often not the case in real-world systems.

The TreeScape system presented here enables practical pre-aggregation even for irregular hierarchies, based on techniques described previously by the authors [97]. We show how to achieve practical pre-aggregation through transformations of the dimensions and how the transformations can be accomplished transparently to the user. The system enables the achievement of fast query response while saving huge amounts of storage compared to current OLAP systems and techniques. The prototype implementation of TreeScape demonstrates that these benefits may be achieved with standard technology. While this demonstration uses a particular RDBMS, it's ODBC driver, and particular relational OLAP tool, TreeScape is not dependent on any specific suite of products[1], making the solution flexible and useful.

B.2 Normalizing Hierarchies

We use a small case study concerning patients and their diagnoses for illustrating the workings of the system. Diagnoses have three different levels of precision, depending

[1]The solution assumes that an ODBC interface is available for the RDBMS, a requirement that is met for all commercial RDBMSs.

on how accurate a patient's condition can be described. The most precise diagnoses are *low-level diagnoses*, which are grouped into *diagnosis families*, which, in turn, are grouped into diagnosis groups. The example data consists of 9 diagnoses and their hierarchical relationship, along with patient counts. The data can be seen in Table B.1 and to the left in Figure B.1.

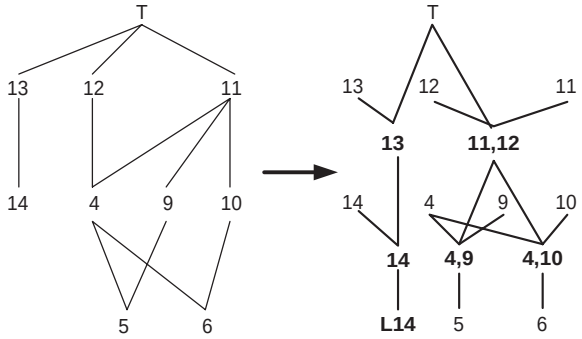

Figure B.1: Dimension Transformations

ID	Text	Type
4	Diabetes during pregnancy	Family
5	Insulin dependent diabetes during pregnancy	Low-Level
6	Non insulin dependent diabetes during pregnancy	Low-Level
9	Insulin dependent diabetes	Family
10	Non insulin dependent diabetes	Family
11	Diabetes	Group
12	Pregnancy related	Group
13	Cancer	Group
14	Lung cancer	Family

Diagnosis

ParentID	ChildID
4	5
4	6
9	5
10	6
11	9
11	10
12	4
13	14

Grouping

DiagID	Count
5	1

Patient

Table B.1: Case Study Tables

The hierarchy is irregular. For example, it is unbalanced because the diagnosis "Lung cancer" (14) has no low-level diagnoses associated with it. The hierarchy is non-strict because, e.g., diagnosis 4 ("Diabetes during pregnancy") has several par-

ents. As a result, problems occur when pre-aggregated data is used, e.g., at the low-level diagnosis level, to compute results such as the number of patients per diagnosis group. The problems include double-counting data and not counting data that should be counted, which leads to incorrect results.

A solution of the problems with pre-aggregation is to render the hierarchies well-behaved by *normalizing* them. Informally, the normalization process introduces new *placeholder* values where the hierarchy is unbalanced, and introduces *fused* values that represent *sets of* parent values when child values have multiple parents. The result of normalizing the hierarchy described above is seen to the right in Figure B.1. For example, value "L14" representing "Lung Cancer" at the low-level diagnosis level, and value "4,9" representing the set of diagnoses $\{4, 9\}$ are introduced by the normalization. In the figure, all boldface values and links have been added by the normalization process. The normalization technique is described in detail elsewhere [97].

The normalized hierarchy supports practical pre-aggregation. For example, it is possible to store counts of patients at the low-level diagnosis level, and then re-use these to compute the counts for diagnosis families and diagnosis groups. With the example data (one patient with diagnosis 5), this will only require the storage of the one value versus six values being required for *full* pre-aggregation (one value for low-level diagnosis 5, two values for diagnosis families 4 and 9, two values for diagnosis groups 11 and 12, and one value for $\top$, which represents the total for all diagnoses).

The example is somewhat indicative of the storage savings achieved within a single dimension. When several dimensions are combined, the total space saved (with respect to full pre-aggregation) is the product of the savings in each dimension, resulting in savings factors of 100 or more in practice. The savings occur because of *multidimensional sparseness* [86, 117], the phenomenon of the multidimensional space being very sparse for the lower levels in the dimensions, while quickly becoming more dense at higher levels. The query response time using the normalization approach will not be quite as fast as using full pre-aggregation, but will most likely be comparable, i.e., within an order of magnitude. This is much faster than computing the results from the base data, as would be required with *no* pre-aggregation.

B.3 System Architecture

While the hierarchy transformations enable practical pre-aggregation, they also have the undesired side-effect of introducing new values into the hierarchies that are of little meaning to the users. Thus, the transformations should remain invisible to the users. This is achieved by working with two versions of each user-specified hierarchy and by using a query rewrite mechanism. This is described in detail in Section B.4. The overall system architecture is seen in Figure B.2.

The ROLAP client tool, in this case the ROLAP tool Synchrony, which originated from Kimball's Startracker tool [64], makes SQL requests to the ROLAP database, in this case the Oracle8 RDBMS, using the ODBC standard. We have implemented a special, query-transforming ODBC driver (QTOD) that, based on case-specific metadata, transforms the SQL requests into requests that hide the transformations from

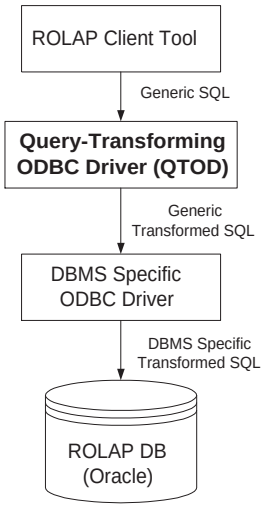

Figure B.2: System Architecture

the users, returning the query results that the user would expect based on the original hierarchies. A transformed request is submitted to the OLAP DB using an RDBMS-specific ODBC driver. The QTOD component is common to all RDBMSs, so Oracle8 may be replaced by another RDBMS such as IBM DB2, Informix, or MS SQL Server. Another ROLAP tool may also be used, making the solution quite general and flexible.

We have chosen to base the prototype on an RDBMS (Oracle8) since RDBMSs are the most commonly used platform for Data Warehouse and OLAP applications. Additionally, the major RDBMSs now, like dedicated multidimensional DBMSes (MDDBs), use pre-aggregated data for faster query responses [131]. However, the approach could also be implemented using multidimensional technology, e.g., based on the Microsoft OLE DB for OLAP standard [78].

The transformation algorithms are implemented in Oracle's PL/SQL programming language. The transformations are relatively fast, taking at most a few minutes, even for large dimensions. Once the dimension hierarchies have been transformed, the QTOD transforms queries and results between the original and transformed hierarchies. The QTOD is a thin layer and adds very little overhead to queries. It is implemented using GNU Flex++/Bison++ scanner/parser generators and the MS Visual C++ compiler.

B.4 Implementation Specifics

Studies have shown that queries on a data warehouse consist of 80% *navigational* queries, which explore the dimension hierarchies, and 20% *aggregation* queries, which aggregate the data at various levels of detail [64]. These two types of queries are treated differently to give the user the illusion that the dimension hierarchies have their original form.

The multidimensional data is captured in a *star schema* [64]. The dimension table for the Diagnosis dimension is given in Table B.2, which has one column for the low-level diagnosis ID in addition to columns for the textual descriptions of low-level diagnoses, diagnosis families, and diagnosis groups.

DiagID	Lowlevel	Family	Group
5	Ins. dpndnt. diab. during pregn.	Diab. during pregn.	Diab.
5	Ins. dpndnt. diab. during pregn.	Diab. during pregn.	Pregn. related
5	Ins. dpndnt. diab. during pregn.	Ins. dpndnt. diab.	Diab.
6	Non Ins. dpndnt. diab. during pregn.	Diab. during pregn.	Diab.
6	Non Ins. dpndnt. diab. during pregn.	Diab. during pregn.	Pregn. related
6	Non Ins. dpndnt. diab. during pregn.	Non Ins. dpndnt. diab.	Diab.
100	!Lowlevel!Lung Cancer	Lung cancer	Cancer

Table B.2: DDiagnosis Dimension Table

The hierarchy captured in the table is *partially normalized*, i.e., placeholder values have been introduced to balance the hierarchy (but it remains non-strict). Specifically, the "!Lowlevel!Lung Cancer" placeholder value has been inserted into the Low-level Diagnosis level. We prefix such values with a "!" and their level to indicate that they are inserted by the transformation process. Note the multiple occurrences of lower-level values caused by the non-strictness of the hierarchy. This is the table that will be used for user navigation in the hierarchy. Its name is prefixed with a "D" to distinguish it from another "Diagnosis" dimension table (described below), to be used for aggregation queries.

We now describe how to achieve transformation transparency for *navigational queries*. The query below retrieves all low-level diagnosis names.

```
SELECT DISTINCT Lowlevel
FROM Diagnosis
```

Navigational queries issued by ROLAP tools have exactly this format. The query is transformed by the QTOD into the query below, which operates against the table DDiagnosis. The transformed query returns the result seen in Table B.3.

```
SELECT DISTINCT Lowlevel
FROM DDiagnosis
WHERE Lowlevel NOT LIKE '!%'
```

Lowlevel
Insulin dependent diabetes during pregnancy
Non insulin dependent diabetes during pregnancy

Table B.3: Navigational Query Result

Due to the use of `DISTINCT` as a quantifier, duplicates are not returned. The `NOT LIKE` predicate removes the placeholder values inserted into the hierarchy to

balance it, which in this case is the value "!Lowlevel!Lung Cancer." As desired, the result is unaffected by the translations.

For *aggregation queries*, it is also possible to achieve transformation transparency, although this is more difficult. For dimensions with non-strictness, a special dimension table is introduced that holds only the part of the normalized hierarchy that does *not* contain non-strictness. In the normalized hierarchy to the right in Figure B.1, this part is the Low-level Diagnosis category and the two special categories introduced by the normalization process to hold *sets of diagnosis families* and *sets of diagnosis groups*, respectively. This part of the hierarchy is implemented in the Diagnosis dimension table seen in Table B.4.

DiagID	Lowlevel	Family	Group
1000020	!Low-level Diagnosis!Lung cancer	14	13
5	Insulin dpndnt. diabetes during pregn.	4,9	11,12
6	Non insulin dpndnt. diabetes during pregn.	4,10	11,12

Diagnosis

Group	SGroup
Cancer	13
Diabetes	11,12
Pregnancy Related	11,12

SGroup

Table B.4: Dimension and Group Tables for Aggregation

The "Lowlevel" column contains the normal textual diagnosis description, whereas the special "Family" and "Group" columns contain comma-separated ordered lists of the IDs of the sets of values that are represented by the column values. For example, value "4,9" represents the set $\{4, 9\}$.

We need to capture the remaining part of the hierarchy, which consists of non-strict mappings from a "set-of-X" category to the "X" category, e.g., the mapping of the "set-of-Diagnosis Group" category to the "Diagnosis Group" category to the right in Figure B.1, which maps $\{13\}$ to 13 (Cancer) and $\{11, 12\}$ to 11 (Diabetes) and 12 (Pregnancy Related). This is done by introducing a special table for each such mapping, named by the category prefixed with an "S" (for Set-of). For example, for the Diagnosis Group category, table "SGroup" in Table B.4 maps sets of diagnosis groups to the individual diagnosis groups in the sets. The "Group" column represents the diagnosis group, while the "SGroup" column represents the associated set of diagnosis groups.

With these tables available, it is possible to obtain transformation transparency for aggregation queries. A ROLAP aggregation query has the format of the query below that computes the number of patients per diagnosis group.

```
SELECT Diagnosis.Group, SUM(Patient.Count)
FROM Diagnosis,Patient
WHERE Diagnosis.DiagID=Patient.DiagID
GROUP BY Diagnosis.Group
```

This is transformed into the more complex query given next.

```
SELECT SGroup.Group, SUM(QQQQQQQ.Count)
FROM Sgroup,
 (SELECT Diagnosis.Group,
         SUM(Patient.Count) AS Count
  FROM Diagnosis,Patient
  WHERE Diagnosis.DiagID=Patient.DiagID
  GROUP BY Diagnosis.Group) QQQQQQQ
WHERE  QQQQQQQ.Group=SGroup.SGroup AND
       SGroup.SGroup NOT LIKE '!%'
GROUP BY SGroup.Sgroup
```

The transformed aggregation query has two parts. The nested table expression computes the number of patients per *set of diagnosis group*, making this available via correlation name QQQQQQQ. This part of the hierarchy is a balanced tree, so the RDBMS can safely use pre-aggregated data for optimizing the query performance. The result of the nested table expression is used in the outer query, which aggregates the last part of the way up to the diagnosis groups using the "SGroup" table. The outer query also filters out any placeholder values inserted by the normalization process (prefixed with a "!"). As a result, the client OLAP tool will retrieve the expected result.

Good query performance without the use of excessive storage for pre-aggregated data is obtained by using practical pre-aggregation for the "nice" part of the hierarchy captured in the "Diagnosis" dimension table. The query transformation exemplified here can be performed for all ROLAP aggregation queries, making the solution quite general.

B.5 Demonstration

Based on concrete data from a real-world case study, the demonstration will initially show snapshots that illustrate the hierarchy normalization process. Next, query processing will be demonstrated by means of concrete navigational and aggregation queries. This includes a description of how the queries are transformed to hide the hierarchy transformations from the user, as well as the evaluation of the queries on concrete data. Finally, the demonstration will compare the query execution times for the queries and the amount of storage required for pre-aggregated data with the two alternatives to our approach, namely *no* pre-aggregation, which gives very long query response times, and *full* pre-aggregation, which requires unrealistically large amounts of storage for pre-aggregated data.

Supporting material in the form of slides and posters will be used in the demonstration.

Appendix C

OLAP++: Powerful and Easy-to-Use Federations of OLAP and Object Databases

C.1 Introduction

On-Line Analytical Processing (OLAP) systems provide good performance and ease-of-use when retrieving summary information from very large amounts of data. However, the complex structures and relationships inherent in related non-summary data are not handled well by OLAP systems. In contrast, object database systems are built to handle such complexity, but do not support summary querying well.

This chapter presents OLAP++, a flexible, federated system that enables OLAP users to exploit simultaneously the features of OLAP and object database systems. In a previous paper [98], we have defined a comprehensive framework for handling federations of OLAP and object databases, including the SumQL++ language that allows OLAP systems to naturally support queries that refer to and retrieve data from object databases. The OLAP++ system allows data to be handled using the most appropriate data model and technology: OLAP systems for summary data and object database systems for the more complex, general data. Also, the need for physical integration of data is reduced considerably. We present a case study based on the Transaction Processing Council (TPC) TPC-R benchmark [126]. The system is implemented in C++ on top of the Object Protocol Model (OPM) system [15] and the Microsoft SQL Server OLAP Services system [78].

C.2 Federations of OLAP and Object Databases

OLAP systems use a multidimensional view of data that typically categorizes data as being measurable facts (measures) or dimensions, which are mostly textual and characterize the facts. Dimensions are structured using categories (levels) that correspond to the required levels of detail. Object systems use the familiar concepts of classes, attributes, and relationships between classes. A federation between an OLAP and

an object database is defined by specifying a link between a category in the OLAP database and a class in the object database.

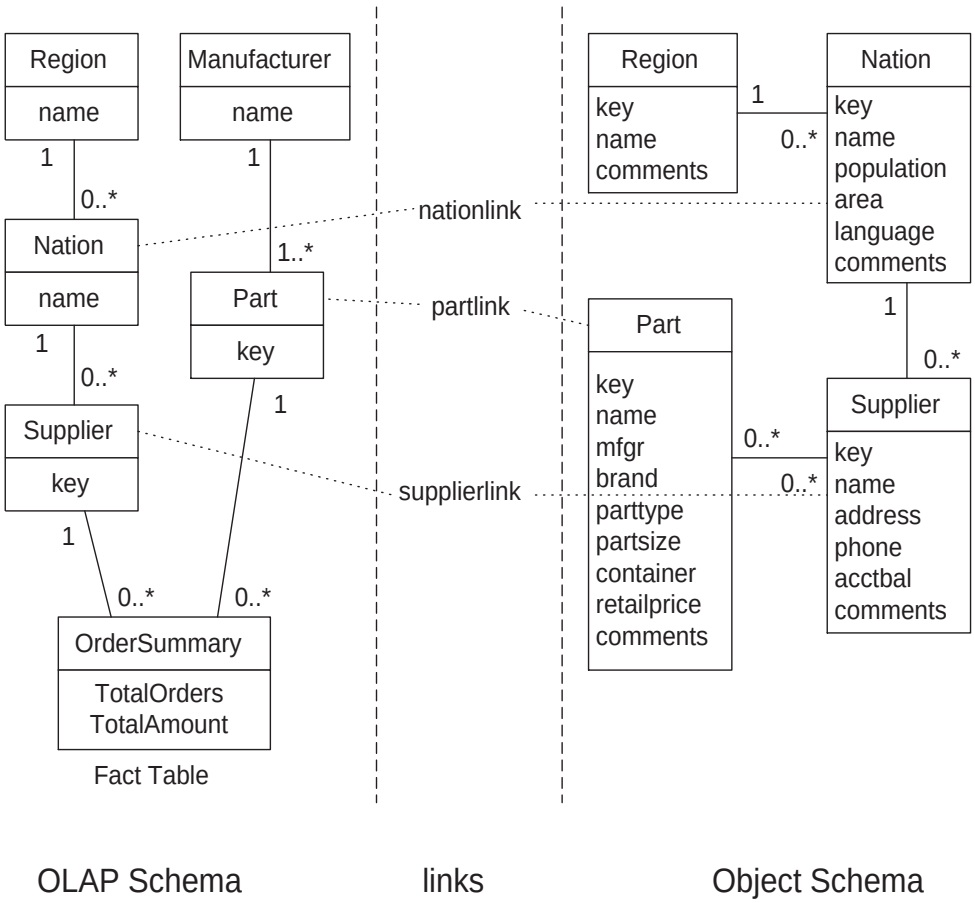

Figure C.1: Schema of the Federation

Figure C.1 shows an example schema of a federation in UML notation. The schema is based on the TPC-R benchmark [126], but has been divided into an OLAP part and an object part. The measured facts in the OLAP schema are the total number of orders and the total cost amount for the orders. The facts are characterized by a Supplier dimension and a Manufacturer dimension. The Supplier dimension has Customer, Nation, and Region categories that allow the facts to be summarized to the required level of detail. The Manufacturer dimension has the categories Part and Manufacturer. The object part of the schema has Region, Nation, Supplier, and Part classes and relationships between them. Link nationlink connects the Nation category in the OLAP part to the Nation class in the object part as indicated by the dotted lines. Links supplierlink and partlink connect the Supplier category and class, and the Part category and class, respectively. Below is an example SumQL++ query for the schema.

```
SELECT TotalAmount INTO testdb
BY_CATEGORY Manufacturer, Nation
FROM OrderSummary
WHERE (Region = "ASIA") AND
 Nation.nationlink.[Nation].population > 100,000,000
```

The above query gets the total cost amount for the two-dimensional cross product of nation and manufacturer where the nations have populations beyond 100 million and are in the Asian region. This query uses the link "nationlink" to go from the OLAP schema to the object schema. The class name in the square brackets is optional and is only specified here to indicate the class reached by going through the link.

C.3 System Architecture

The overall architecture of the federated system is seen in Figure C.2. The object part of the system is based on the OPM tools [15] that implement the Object Data Management Group's (ODMG) object data model [13] and the Object Query Language (OQL) [13] on top of a relational DBMS, in this case the ORACLE RDBMS. The OLAP part of the system is based on Microsoft's SQL Server OLAP Services using the Multi-Dimensional eXpressions (MDX) [78] query language. The GUI is implemented as Java classes running in a standard Web browser for optimal flexibility.

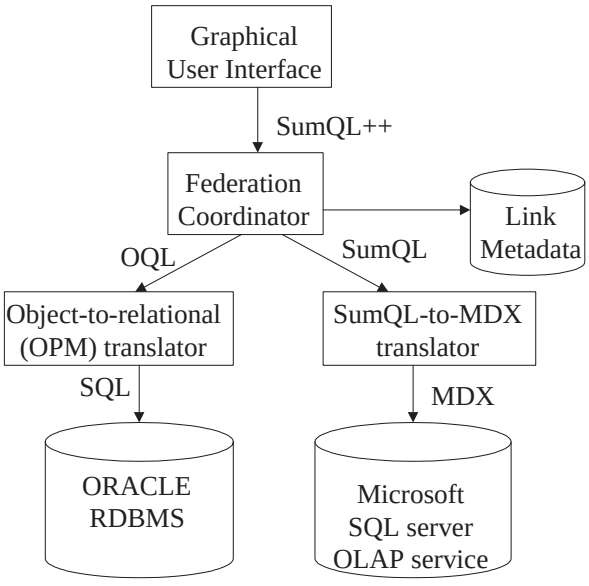

Figure C.2: Architecture of the Federated System

When a SumQL++ query is received by the Federation Coordinator (FC), it is first parsed to identify the measures, categories, links, classes and attributes referenced in the query. Based on this, the FC then queries the metadata to get information about which databases the object data and the OLAP data reside in and which categories are linked to which classes.

Based on the object parts of the query, the FC then sends OQL queries to the object databases to retrieve the data for which the particular conditions hold true. This data is then put into a "pure" SumQL statement, i.e., without object references, as a list of category values. This SumQL statement is then sent to the OLAP database layer to retrieve the desired measures, grouped by the requested categories. The SumQL statement is translated into MDX by a separate layer, the "SumQL-to-MDX translator", and the data returned from OLAP Services is returned to the FC.

The reason for using the intermediate SumQL statements is to isolate the implementation of the OLAP data from the FC. As an another alternative, we have also implemented a translator into SQL statements against a relational "star schema" design.

The system offers good query performance even for large databases while making it possible to integrate existing OLAP data with external data in object databases in a flexible way that can adapt quickly to changing query needs.

C.4 The Demonstration

The demonstration will show the specification of, and query processing for, specific queries on a large TPC-R-based database. First, the use of the system will be demonstrated. Second, we will describe the details of query processing in the system. In the demonstration, we will also show how new federations can be specified "on-the-fly" and used immediately. Supporting material in the form of slides and posters will be used in the demonstration.

C.4.1 User Interface

The web screen interface shown in Figure C.3 shows how the user perceives the specification of a SumQL++ query. Figure C.3 shows the selection of the summary measure "TotalAmount". This is followed by the section with the category attributes "Manufacturer" and "Nation". Note that each category can be selected from a "category hierarchy". In the figure, "Nation" was selected from the "Region-Nation-Supplier" category hierarchy. The order of the category grouping can be specified in this screen as well by switching the dimension positions.

Figure C.4 shows the specification of query conditions. Initially, each dimension is shown with its categories and links to the object database. If a category is selected, a category condition can be entered. In the figure, Region= "ASIA" was selected. If a link is clicked on, then the attributes of the object linked to are shown. The user can select an attribute to specify a condition. In the figure, the condition "population > 100 Million" was selected through the "nationlink". The result of the above selections is a concise SumQL++ query (the same query as the example in Section C.2), as shown next.

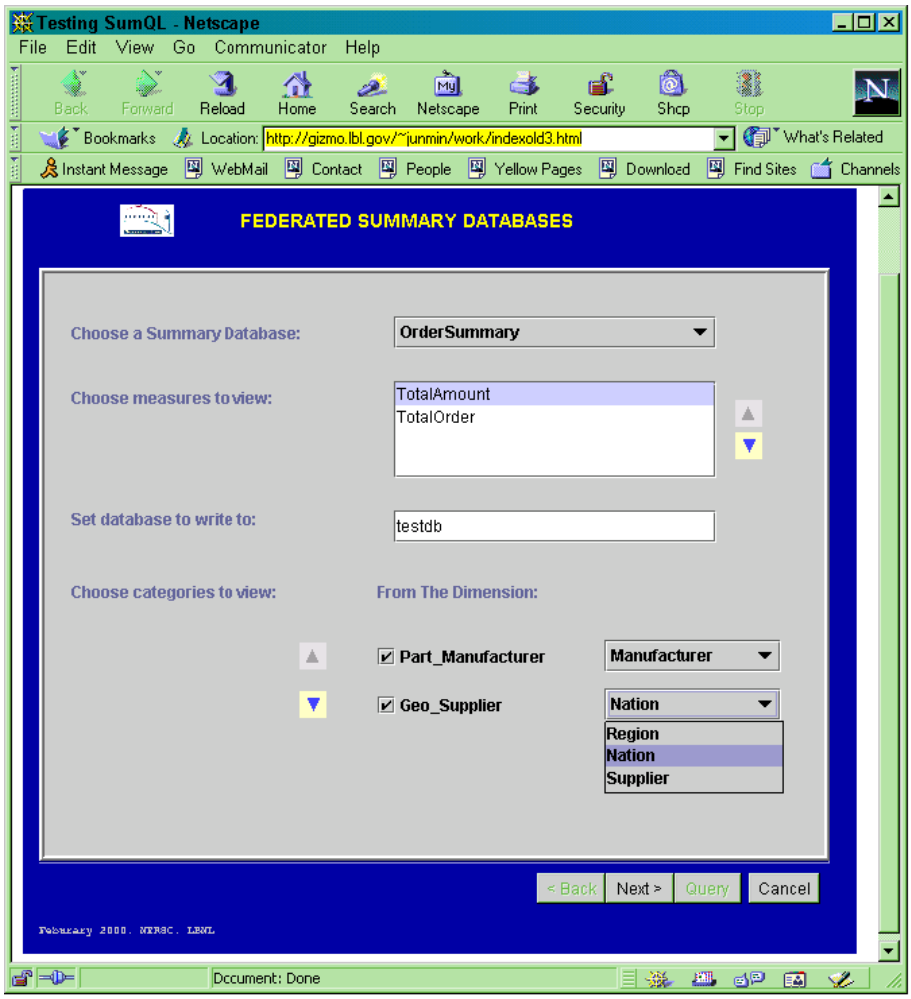

Figure C.3: Selection of Measures and Category Attributes

```
SELECT TotalAmount INTO testdb
BY_CATEGORY Manufacturer, Nation
FROM OrderSummary
WHERE (Region = "ASIA") AND
      (Nation.nationlink.population > 100000000)
```

The result of this query is then displayed on the user's screen, as shown in Figure C.5.

C.4.2 Query Processing

We now proceed to describe the steps in the query processing in more detail. After the query is generated, the system parses the query to determine the OLAP and object parts. For the example above the result of the parsing is:

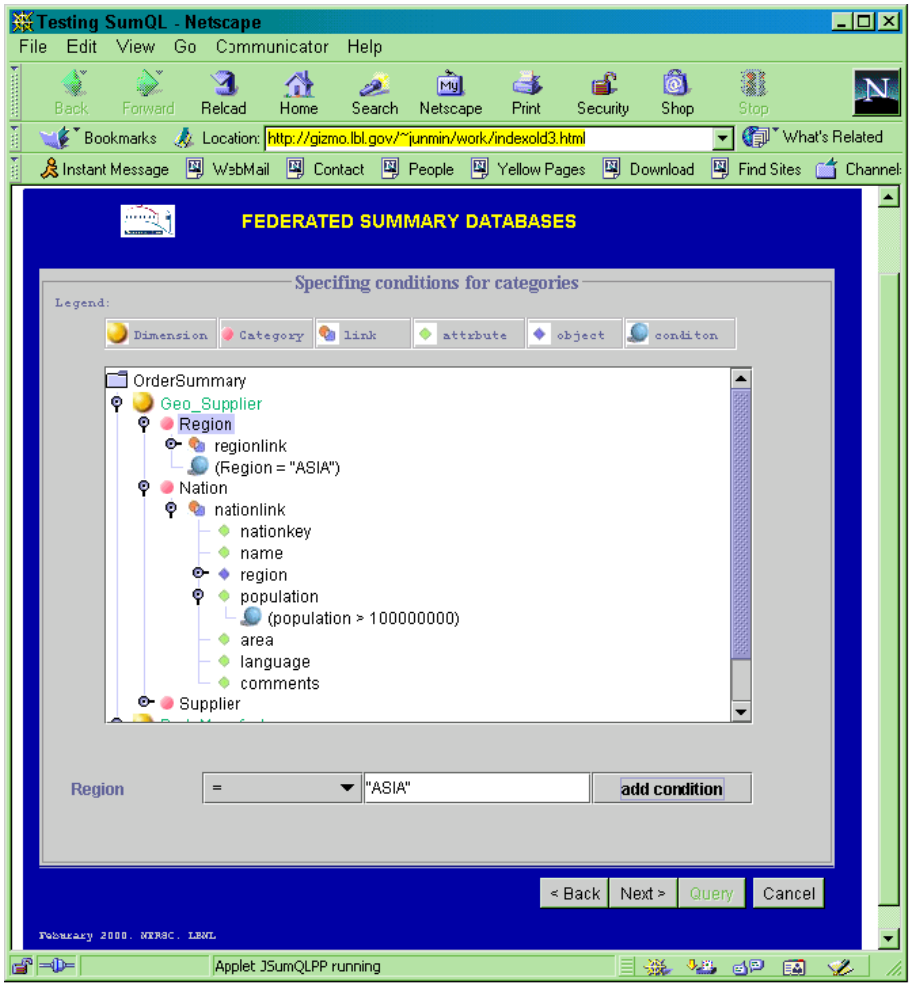

Figure C.4: Specification of Conditions

SELECT `TotalAmount` **INTO** `testdb`
BY_CATEGORY `Manufacturer, Nation`
FROM `OrderSummary`
[AND]
predicate: CATEGORY = Region
no object path
——-> = "ASIA"
predicate: CATEGORY = Nation
LINK = nationlink
PATH = .
ATTR = population
——-> > 100000000

Manufacture	Nation	TotalAmount
Manufacturer#1	CHINA	$1,828,836,362.73
	INDIA	$1,878,339,813.97
	INDONESIA	$1,833,735,510.30
	JAPAN	$1,708,892,850.49
Manufacturer#2	CHINA	$1,854,693,740.10
	INDIA	$1,899,630,808.35
	INDONESIA	$1,800,984,069.04
	JAPAN	$1,686,441,585.08
Manufacturer#3	CHINA	$1,847,335,838.07
	INDIA	$1,932,734,790.64
	INDONESIA	$1,834,270,926.06
	JAPAN	$1,702,617,297.30
Manufacturer#4	CHINA	$1,817,791,499.44
	INDIA	$1,838,696,860.36
	INDONESIA	$1,838,743,137.86
	JAPAN	$1,698,211,032.43
	CHINA	$1,848,958,788.86

Figure C.5: Query Result

Each link predicate is then evaluated by the object system. For example, the following OQL query is passed to the object DB system to find the nations with a population of more than 100 million:

```
SELECT name = @n001
FROM @n000 IN tpcr:NATION, @n001 IN @n000.name
WHERE @n000.population > 100000000;
```

After the results are returned, they are used in the OLAP part of the system to generate the following SumQL query that retrieves the desired data.

```
SELECT TotalAmount INTO testdb
BY_CATEGORY Manufacturer, Nation
FROM OrderSummary
WHERE (Region = 'ASIA' AND
       Nation IN( 'BRAZIL', 'INDIA',
                  'INDONESIA', 'JAPAN',
                  'CHINA', 'RUSSIA','UNITED STATES' ))
```

This, in turn, gets translated into MDX as follows.

```
SELECT {[Measures].[L ExtendedPrice] } ON COLUMNS,
INTERSECT
(CROSSJOIN([Part_Manufacturer].[P Mfgr].MEMBERS,
DESCENDANTS([R Region Name].[ASIA],[N Nation Name],
SELF)),
CROSSJOIN([Part_Manufacturer].[P Mfgr].MEMBERS,
{[N Nation Name].[BRAZIL],[N Nation Name].[INDIA],
[N Nation Name].[INDONESIA],[N Nation Name].[JAPAN],
[N Nation Name].[CHINA],[N Nation Name].[RUSSIA],
[N Nation Name].[UNITED STATES]}))
ON ROWS
FROM OrderSummary
```

The result is then stored in the Oracle database "testdb," to make it available for further processing, and converted to HTML for presentation to the user.

This section was intended to illustrate the amount of work that a user will have to go through without the aid of the user interface and the federated translation tools. In particular, we wish to emphasize the usefulness of the OLAP-object database links to generate the combined result. Also, the users are spared the verbosity of MDX (which is hidden from them). It is optional to display the concise SumQL++ expression to the user, as a way to verify the correctness of the query.

We do not describe the specification of new links in this chapter. However, this will be shown at the demonstration.

Appendix D

Summary in Danish

Denne afhandling omhandler datamodellering og forespørgselsudførsel for komplekse multidimensionelle data. Multidimensionelle data er blevet genstand for megen interesse indenfor både den akademiske verden og erhvervslivet i de senere år, på grund af populariteten af data warehousing og On-Line Analytical Processing (OLAP) applikationer.

Et applikationsområde hvor komplekse multidimensionelle data er almindelige, er indenfor medicinsk informatik, et område der kan drage stor nytte af funktionaliteten der tilbydes af data warehousing og OLAP. Imidlertid stiller kliniske applikationers specielle natur nye og anderledes krav til data warehousing og OLAP teknologierne, i forhold til kravene for konventionelle data warehousing applikationer. Denne afhandling præsenterer et antal spændende, nye forskningsudfordringer forårsaget af kliniske applikationer, som databaseforskningen kan arbejde på. Disse inkluderer behovet for datamodelleringsegenskaber for komplekse data, avanceret temporal understøttelse, avancerede klassifikationsstrukturer, data med kontinuerte værdier, dimensionelt reducerede data, og integration af komplekse data.

OLAP systemer bruger typisk multidimensionelle datamodeller til at strukturere deres data. Denne afhandling identificerer elleve modelleringskrav for multidimensionelle datamodeller. Disse krav er udledt af en realistisk vurdering af komplekse data fra virkelige applikationer. En inspektion af tolv multidimensionelle datamodeller afslører mangler i at opfylde nogle af kravene. Eksisterende modeller understøtter ikke mange-til-mange relationer mellem facts og dimensioner, har ikke indbyggede mekanismer for at håndtere forandring og tid, mangler understøttelse af upræcise data og er ikke i stand til at håndtere data med varierende granularitet. Ydermere understøttede de fleste af modellerne ikke irregulære dimensionshierarkier og aggregeringssemantik. Denne afhandling definerer en udvidet multidimensionel datamodel og et algebraisk spørgesprog, som adresserer alle elleve krav. Denne model genbruger de almindelige multidimensionelle begreber dimensionshierarki og granulariteter til at beskrive upræcise data. For forespørgsler der ikke kan besvares præcist p.g.a. upræcise data, foreslås teknikker der tager hensyn til upræcision i gruppering af data, i den efterfølgende aggregeringsberegning, og i præsentationen af det upræcise result til brugen. Ydermere bliver der foreslået alternative forespørgsler der er upåvirkede af upræcisionen. Datamodellen og de præsenterede forespørgselsevalueringsteknikker kan implementeres v.h.a. relationel databaseteknologi. Metoden er

også i stand til at udnytte multidimensionelle teknikker til forespørgselsudførelse såsom præaggregering. Dette giver en praktisk løsning med lave ekstraomkostninger for beregninger.

Præaggregering, hvor resultatet af aggregeringsforespørgsler materialiseres til senere brug, er en essentiel teknik for at sikre tilfredsstillende svartid under dataanalyse. Fuld præaggregering, hvor alle kombinationer af aggregater bliver materialiseret, er ikke brugbart i praksis. I stedet for bruger moderne OLAP systemer praktisk præaggregering, hvor kun udvalgte kombinationer af aggregater materialiseres og disse genbruges til effektivt at beregne andre aggregater. Imidlertid er dette genbrug af aggregater afhængig af at dimensionshierarkierne og relationerne mellem facts og dimensioner opfylder strenge krav. Dette begrænser anvendelsesområdet for praktisk præaggregering alvorligt. Denne afhandling udvider anvendelsesområdet for praktisk præaggregering betydeligt, så det dækker en meget større mængde af realistiske situationer. Specifikt præsenteres algoritmer der transformerer "irregulære" dimensionshierarkier and fact-dimensionsrelationer, som ofte forekommer i virkelige OLAP applikationer, til regulære strukturer der kan bruges af eksisterende OLAP systemer til at muliggøre praktisk præaggregering. Algoritmerne har lav beregningskompleksitet og kan anvendes inkrementelt for at reducere omkostningerne ved at opdatere OLAP strukturer. Transformationerne kan gøres transparente for brugeren. En prototypeimplementation af teknikkerne er beskrevet.

OLAP systemer giver god ydelse og er lette at bruge for forespørgsler der aggregerer store mængder af data. Imidlertid bliver de komplekse strukturer og relationer som ofte findes i data i ikke-standard applikationer ikke understøttet godt af OLAP systemer. I modsætning hertil er objektdatabasesystemer bygget til at håndtere denne kompleksitet, men understøtter ikke OLAP forespørgsler godt. Denne afhandling præsenterer begreberne og teknikkerne der understøtter et fleksibelt, "multimodel" føderationssystem der gør OLAP brugere i stand til udnytte egenskaberne ved OLAP og objekt systemer samtidigt. Systemet tillader data at blive håndteret v.h.a. den mest velegnede teknologi: OLAP systemer for dimensionelle data og objektdatabasesystemer for mere komplekse, generelle data. Som en platform for at demonstrere systemets evner defineres et prototypisk OLAP sprog der udvides til naturligt at understøtte forespørgsler der involverer data i objektdatabaser. Sproget tillader selektionskriterier der refererer objektdata, forespørgsler der returnerer kombinationer af OLAP og objekt-data og forespørgsler der grupperer data i.h.t. objektdata. Systemet er designet til at være aggregeringssikkert, forstået sådan at det udnytter aggregeringssemantikken for data til at forhindre forkerte eller meningsløse forespørgselsresultater. Disse egenskaber kan også integreres i eksisterende sprog. En prototypeimplementation af systemet er beskrevet.