Tapio Niemi

Methods for Logical OLAP Design

ACADEMIC DISSERTATION

To be presented, with the permission of the Faculty of Information Sciences of the University of Tampere, for public discussion in the Paavo Koli Auditorium of the University on December 15th, 2001, at 12 noon.

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES
UNIVERSITY OF TAMPERE

A-2001-9

TAMPERE 2001

Supervisor: Professor Jyrki Nummenmaa
Department of Computer and Information Sciences
University of Tampere
Finland

Opponent: Professor Heikki Mannila
Laboratory of Computer and Information Science
Helsinki University of Technology
Finland

Reviewers: Professor Heikki Mannila
Laboratory of Computer and Information Science
Helsinki University of Technology
Finland

Doctor Thomas Zürek
SAP Portals
Walldorf
Germany

Department of Computer and Information Sciences
FIN-33014 UNIVERSITY OF TAMPERE
Finland

Electronic dissertation

Acta Electronica Universitatis Tamperensis 153
ISBN 951-44-5264-X
ISSN 1456-954X
http://acta.uta.fi

ISBN 951-44-5230-5
ISSN 1457-2060

Tampereen yliopistopaino Oy
Tampere 2001

Abstract

As the amount of information is increasing all the time, information modelling and analysis have become essential areas in information management. Storing and retrieving data have earlier been the main functions in databases but the importance of deeper understanding of data has increased during the recent years. The nature of data has also become more complex. Therefore, powerful modelling methods are needed for the data. In addition, the methods have to be user-friendly since the users of data oriented applications are not often database-professionals. The general aim of this work is to develop methods to make database design and management easier and more efficient for all users, both database professionals and people with no prior experience with databases.

We have studied methods to support logical design of OLAP (On-Line-Analytical Processing) cubes. The methods give a good basis for implementing software tools that partly automate the logical design process. These methods are needed since it is commonly noticed that logical design of OLAP cubes is a complex process that requires good knowledge on both application area and databases. Good logical structure of an OLAP cube is important, because a bad design can lead, for example, to an extremely sparse cube and to such a need for storage space that is not possible to achieve in practice. As a solution, we give a method for estimating the structural sparsity of OLAP cubes and a normal form to reduce sparsity risks. Moreover, synthesis and decomposition algorithms for producing normalised OLAP cubes are developed. Hierarchical dimensions, which enable the user to analyse data on different levels of aggregation, are essential for OLAP. Hierarchies can arise from the attribute hierarchy (e.g. day, month, year) or from the relationship between the instances of two attributes (e.g. employee, manager). We study what kinds of hierarchy structures are desirable with respect to correct aggregations and efficient calculations. To represent logical OLAP schemata, a dependency-based modelling method has been developed. This method enables the user to describe concepts and their relationship to each other explicitly. The OLAP cube should be complete and minimal with respect to the user's queries. To define what data should be taken into account when constructing an OLAP cube, we give two methods based on query information. One applies the intensional concept theory and a query method based on it. The other uses MDX queries that the user poses against a base cube representing the contents of the data warehouse. If real queries are available, they can be used as input.

Acknowledgements

I especially want to thank my supervisor Professor Jyrki Nummenmaa for introducing OLAP as a promising research area as well as his help during this research. I am also very grateful to him for introducing Dr Peter Thanisch to me. I also thank Professor Heikki Mannila for his comments on my manuscript.

I am very grateful to Dr Peter Thanish for inspiriting discussions especially during my visit at the University of Edinburgh at the starting point of this research. I also want to thank him for many constructing comments and ideas during this whole process. He has also helped me with the English language.

I want to express my gratitude to Professor Hannu Kangassalo for guiding me to the area of data management. He has always helped me with any kinds of problems during my studies.

I thank my colleagues and co-students at the Department of Computer and Information Sciences in the University of Tampere. They all have been very friendly and have always had time to answer my questions or comment my work. Especially I want to mention Isto Aho, Anne Aula, Marko Junkkari, Dr Erkki Mäkinen, Marko Niinimäki, and Kati Viikki. I am also grateful to Liisa Kurki-Suonio for her help with my English language. I also want to thank all my friends for their support and offering me change to relax in the middle of 'boring' academic work.

I am grateful to Tampere Graduate School in Information Science and Engineering (TISE) and the University of Tampere for their financial support.

Finally, I warmly thank my parents and my sister for their encouragement and support during my studies.

Tampere, November 2001
Tapio Niemi

Contents

List of Publications

This thesis is based on the following research papers:

I Niemi, Tapio and Nummenmaa, Jyrki: A query method based on intensional concept definition, *Frontiers in Artificial Intelligence and Applications: Information Modelling and Knowledge Bases XII*, H. Jaakkola and H. Kangassalo (eds.), Vol. 67, pages 92-106, IOS Press, 2001.

II Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Applying dependency theory to conceptual modelling, *Topics in Conceptual Analysis and Modeling*, O. Majer (ed.), pages 271-290, Czech Academy of Sciences' Publishing House Filosofia, Prague, 2000.

III Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Normalising OLAP Cubes for Controlling Sparsity, 2001, submitted to Data & Knowledge Engineering. (An earlier version of this work is: Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Functional dependencies in controlling sparsity of OLAP cubes, *Data Warehousing and Knowledge Discovery, Second International Conference, DaWaK 2000*, Y. Kambayashi et al. (eds.), Lecture Notes in Computer Science, Vol. 1874, pages 199-209, Springer, 2000.)

IV Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Logical multidimensional database design for ragged and unbalanced aggregation hierarchies, *The Proceedings of the International Workshop on Design and Management of Data Warehouses, DMDW'2001*, D. Theodoratos et al. (eds.), pages 7-1 - 7-8, 2001.

V Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Applying intensional concept theory to OLAP design and queries, *The Proceedings of the 11th European-Japanese Conference on Information Modelling and Knowledge Bases,* H. Kangassalo et al. (eds.) pages 142-153, 2001.

VI Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Constructing OLAP cubes based on queries, *DOLAP 2001, ACM Fourth International Workshop on Data Warehousing and OLAP*, J. Hammer (ed.), pages 9-15, ACM, 2001.

1 Introduction

Until recently, research in the database field has concentrated more in storing and retrieving methods than information modelling and analysis of the data. However, two new trends can be recognised. Firstly, information systems are a part of one's everyday life in modern society and they are needed to accomplish many common tasks. This implies that more user-friendly data management methods are needed. Secondly, recent systems are able to store large amounts of data, which sets new requirements for the analysis methods.

The primary aim of this work is to provide methods for improving logical design and querying of multidimensional databases used in *On-Line Analytical Processing* (OLAP). The term "On-Line Analytical Processing" was introduced by Codd et al. in 1993 [CCS93], but the idea itself and even implementations of the idea date back to the 1960's. OLAP is a methodology for analysing data stored in large data warehouses using multidimensional structures called OLAP cubes. Especially, OLAP is useful in situations where raw data on measures such as sales or profit needs to be analysed at different levels of statistical aggregation. The OLAP methodology is often used in business environments but it is usable in many other fields, too. The idea in OLAP is to represent the database as a multidimensional cube with hierarchical dimensions (see Figure 1 as an example). This structure enables users to analyse the data from different viewpoints and on different levels of details. For example, using the cube in Figure 1 the user can try to find out why certain owner-car combinations have higher repair cost than the others. First the user can study the repair costs of some owner-car combination on the year level and then drill-down to the month and finally to the day level. From the day level the user can roll-up back to the month level and/or change the owner-car combination at hand.

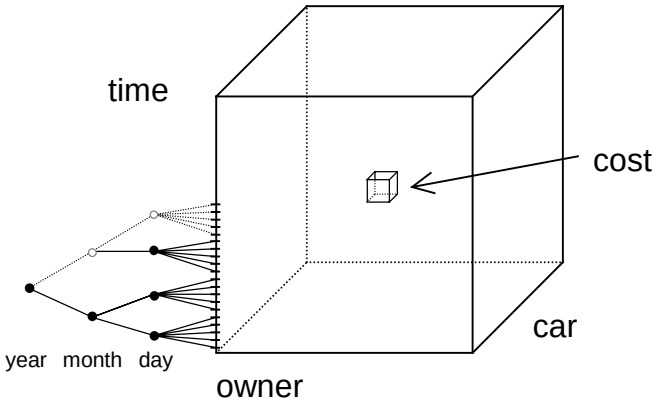

Figure 1. A three dimensional OLAP cube about repair costs of cars

In the analysing process described above, the structure of the OLAP cube is essential. To help users in logical design of OLAP cube schemata we provide automated design methods. These methods help the user to design better OLAP schemata with respect to efficiency, correctness of calculations, and ease of use. Since OLAP users, like database users in general, are mostly non-professionals in the database field, these methods should be intuitive and easy to use and yet have great expressive power.

Secondly, in the field of conceptual modelling, the aim of this work is to study conceptual modelling methods in order to improve their usability and expressiveness and to provide a query language operating purely on the conceptual level. In conceptual modelling, there is a lack of intuitive and formal design methods. Graphical methods, like the Entity-Relationship model [Che76], are practical but suffer from the lack of exact formalism. In addition, using most of these methods does not help the user to think about concepts and their relationships very deeply. This can sometimes lead to over-simplification of the application area. The purpose of the use also affects the required properties of the modelling method. For example, in OLAP modelling, presenting the dependency information is important. Further, the common general-purpose query languages are often too complicated for occasional use since users need to learn them beforehand. In such applications as the so-called information kiosks, this is seldom possible. The same kinds of problems occur when non-professionals in database field, for example, business-oriented people, analyse data using OLAP methods.

There are several reasons why efficient design methods for OLAP cubes are needed. In OLAP, the queries are made against multidimensional OLAP cubes. The OLAP cube has often been seen as a static storage structure for data warehouse data and the cube design has been based on the knowledge of the application area. The types of queries

that the users are expected to pose should be also taken into account. It is a common practice to update the data in the OLAP cube quite seldom, for example once a week. However, in many applications, the OLAP cube has to be much more up-to-date. In practice, a company may often change its own internal organisation. Thus, realignment of data warehouse schemata is needed quite often in order to keep the model faithful to reality. Especially, dimensions and their hierarchies can change even regularly [HMV99, ZuSi99]. Some users may want to speculate about the effects of, for example, changing the way their company has arranged its organisation. Also, there may be requirements to analyse data in ways, which were not anticipated at the time when the OLAP cubes were designed. Thus, the users may want to change the organisation of the cube even if the actual data have not changed. For these reasons, the design of OLAP cubes cannot only be the responsibility of database professionals, but the end users must also be able to design customised cubes which are suitable for their current analysis tasks.

OLAP design differs quite a lot from the design of operational databases. The main goal in operational databases is usually to ensure efficient transaction processing for simple updates and queries. Thus, for example, avoiding redundancy in data storage is very important, because, in addition to the need for extra storage space, redundancy also imply that updates may need to be performed in several places. The OLAP databases are mostly used for large queries and updated quite seldom. Therefore, avoiding redundancy in data storage is not very important. Instead, redundancy can often increase query performance. OLAP queries are often large aggregations of data. The OLAP cube can be represented as a (hyper) cube. Figure 1 illustrates a three-dimensional cube having time, geography, and product as dimensions and profit as a measure. The cube structure also leads to a problem with missing measure values. It is common that every combination of dimension values has no measure value. This is not a bad problem when storing data but can be a serious problem while storing pre-calculated aggregation values. Thus, controlling sparsity is one of the most important design goals in OLAP. Functional dependencies or other constraints can imply sparsity but sparsity highly depends on the actual data. This makes controlling sparsity difficult in logical design, when actual data is not available or the volume of data is too high.

Although there is some research done about the design goals for OLAP cubes, this research is far from complete and many important design issues still require further attention. To date, the research has concentrated on physical design issues such as indexing and clustering. Some work is done about the conceptual level description of

OLAP applications using, for example, ER diagrams [Che76]. Nevertheless, logical OLAP design has received little attention. The star schema approach is the most used, but it is not strictly on the logical level and there is no proper design methodology for it [Dat00]. Logical cube design, however, has an essential role in the efficiency and the usability of the OLAP system. Especially, controlling sparsity, that is, the relative proportion of empty cells in a cube, is necessary since sparse raw data with many dimensions and precalculated aggregation values can expand the need of the storage space too much to be handled by increasing the storage capacity [Ped00]. Estimating and reducing sparsity are complicated tasks but they can be automated quite well.

The relational database theory [Cod70] is the main formalism used in this work. Especially, the functional dependencies [Cod72a] have an essential role in our modelling methods for OLAP cubes. To design logical database schemata, especially logical OLAP schemata, we give a dependency based modelling method that explicitly shows dependencies between concepts. For logical OLAP design itself, we have further developed normal forms for OLAP cubes. The *non-sparse normal* form guarantees that no *structural sparsity* may exist in a cube. Both synthesis and decomposition algorithms are given to produce OLAP cubes in certain normal forms. Finally, the dimension hierarchies are classified to attribute hierarchies and data hierarchies, and the properties of these different hierarchy structures are studied. If a hierarchy is transitively anti-closed, then no redundant aggregation paths can exist. A balanced and non-ragged hierarchy having consistent level identifiers (i.e. the level identifiers totally correspond to the actual levels of the nodes) guarantees complete aggregations on each level in a hierarchy. As mentioned earlier, in OLAP, the analysis is performed by posing queries against an OLAP cube. The structure of the OLAP cube is important for successful analysis but the cube must also contain all the relevant information. It is also desirable that the cube contains only the relevant information for current analysis requirements. We have developed two methods for applying the query information in order to construct complete but not too large OLAP cubes. The methods are based on the intensional query method [NiNu01] and the use of MDX queries [Mic98].

The rest of this work is organised in the following way. Section 2 contains a review of logical OLAP design that is the main contribution of this work. In Section 3, the conclusions are given. Section 4 contains a brief introduction to the publications. The publications itself are the last part of this work.

2 OLAP Design

OLAP cube design can be divided into three phases: conceptual modelling, logical design, and physical design. The borders between them are not very clear, however, and the phases are different from their counterparts in traditional database design. Conceptual modelling includes describing and analysing the concepts and their relationships. The result of the conceptual modelling is a conceptual schema, which explicitly represents concepts and their relationships. The starting point of conceptual modelling varies. It may be a pre-existing data warehouse, an operational database, or it is possible to start designing the cube first and then design and implement the data warehouse for that cube. Figure 2 illustrates the OLAP and data warehouse design processes as we treat them in this work.

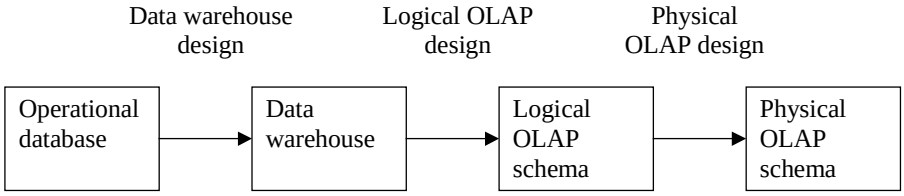

Figure 2. OLAP design process

Next, some essential terms are explained. A *database* is a collection of data that is stored in a computer more-or-less permanently and will be used in the application system of an organisation [Ull80, Dat00]. In the database context, data are meaningful facts that can be represent and stored by the computer. There are several different logical database models, for example relational and object oriented. Databases can be classified according to their use, too. An *operational database* is a database used to store and retrieve data in operational real-time applications. *On-Line Transaction Processing* (OLTP) is the most important task that is why updating, adding, and simple querying must be efficient. The database should be optimised to avoid update anomalies and redundancy. To avoid these problems several normal forms and other design principles are developed for relational databases [Cod72a, Cod74, Fag77, Num95]. A *data warehouse* is a database containing data for some analysis purposes. The data are often copied from operational databases to the data warehouse. Usually some operations

are performed to the data. For example, the data can be consolidated or summarised. A data warehouse is seldom as up-to-date as an operational database, because the data are imported, for example, once a week. Since the update operations are not usually as critical in data warehouses as in operational databases, data warehouses can be optimised for queries. Queries can be large and they often concern summarised data. *On-Line Analytical Processing* (OLAP) techniques are used to achieve more efficiency and expressive power [CCS93]. Data warehouses often contain data from several sources and also historical data for long periods, hence the size of the OLAP cube representing the data warehouse can be extremely large. A technique to speed up calculations is to pre-calculate aggregation data, but this also increases the need for the storage space.

Formally, the *OLAP cube* is a relation over the relation schema $C = D_1 \cup D_2 \cup ... \cup D_n \cup \{M\}$, where each D_k is a set of attributes called a dimension schema and M is the measure attribute. If D is a dimension schema, then a relation d over D is called a *dimension*. The OLAP cube can be seen as a conventional relation having some special constraints. The most important of these is that the dimensions must be as coordinates for the measure value. Table 1 illustrates an OLAP cubes as a relation.

Table 1. An OLAP cube as a relation

DIMENSIONS ATTRIBUTES					
dimension 1	dimension 2	dimension 3			MEASURE
Owner	Car	Day	Month	Year	Cost
Jones	ABC-124	12-1-2000	1-2000	2000	1500
Smith	DFG-125	10-2-2001	2-2001	2001	2000
Smith	QWE-456	5-4-2001	4-2001	2001	400
Tailor	RTY-789	17-4-2001	4-2001	2001	800

2.1 Logical OLAP Design Goals

Logical design starts from the conceptual schema and its result will be an OLAP schema which is a description of cubes with dimensions and their hierarchies. What we call logical design is sometimes referred to as conceptual design. Logical design is needed, because there are various different ways in which the users may want to arrange the OLAP cube, depending on the nature of the queries they intend to pose. Different designs vary significantly in terms of efficiency and practicality. For example, for a

given set of raw data, a cube can be extremely sparse in the sense that many of the data items are missing, or zero, because of the nature of the data. Some other designs can be problematic for query formulation as they may, for instance, produce incorrect aggregations.

Logical design should be independent from the implementation method chosen for the cube. In contrast to traditional database design, designing logical database schemata is not included in logical OLAP design but physical design. The aim of physical design is to find an efficient implementation for the desired data cube. Physical design also contains optimisation of the chosen storage structure, for example building indexes. Relationally implemented OLAP system is called ROLAP (Relational OLAP). The ROLAP system uses a relational database for storing the data. The relational database schema can have an unnormalised star structure or a normalised snowflake structure. To improve the efficiency of the system, multidimensional storage methods have been implemented. This is called MOLAP (Multidimensional OLAP). The Hybrid OLAP is a combination of the both storage methods. Its aim is to use the multidimensional storage method when it is more suitable than the relational one and vice versa.

The aim of logical design is to produce a ‘good’ schema for an OLAP cube or a set of cubes. A desirable OLAP cube should be such that the user’s queries can be:

1) constructed easily: The system should be easy to use and the user should be able to construct complex queries easily.
2) answered correctly: The user gets a correct and relevant answer to the query.
3) evaluated efficiently: The query can be evaluated quickly using minimal amount of resources.

As a general rule, it can be said that a cube should have as many dimensions as possible and as little sparsity as possible. Increasing dimensions increases the expressive power but also the need for the storage space. Moreover, if the dimensions are not orthogonal, the resulting cube will be sparse. These are studied in more detail in the following subsections.

2.1.1 Controlling Sparsity

The sparsity of an OLAP cube is the ratio (the number of empty cells) / (the number of all cells). Thus, sparsity is one if all cells are empty and zero if all cells are occupied. Naturally, sparsity depends on actual data but functional dependencies may imply *structural sparsity*. For example, if we have city and zip code in different dimensions and no two cities can share the same zip code. Figure 3 illustrates a two-dimensional

table of cars and their owners with a functional dependency *car* $\rightarrow$ *owner*. The measure attribute is the price of the car. The dependency *car* $\rightarrow$ *owner* implies sparsity, because a car can have only one owner. The sparsity of the table, that is, an OLAP cube, is 8/12 = 2/3.

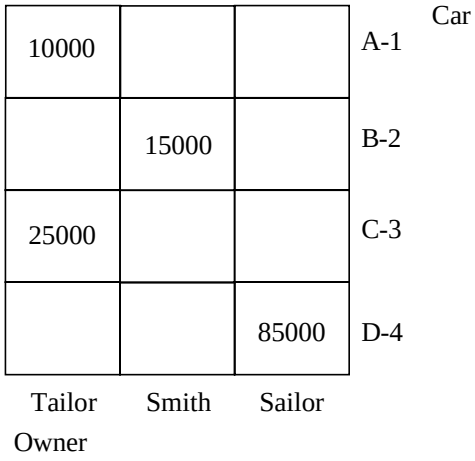

Figure 3. A functional dependency Car→Owner implies sparsity

It is not straightforward to give the exact amount of structural sparsity, since it highly depends on data. Instead, we can say what kinds of functional dependencies imply structural sparsity. First, we give two definitions. Let $C = D_1 \cup D_2 \cup ... \cup D_n, \cup M$ be an OLAP cube schema, where $D_1...D_n$ are dimension schemata and M the set of measure attribute, and let $K_1,K_2,...,K_n$ be the respective dimension key attributes. A dimension is said to be redundant if $(\{K_1,\ldots,K_n\}\backslash\{K_i\}) \rightarrow K_i$. Further, we say that C *contains key-RHS-intersection*, if there exists an attribute $A \in C$, and two sets of attributes $X \subseteq \{K_1,K_2,...,K_n\}$ and $Y \subseteq \{K_1,K_2,...,K_n\}$, such that $X \neq Y$, $X \mapsto A$ and $Y \mapsto A$, where $\mapsto$ indicates a full dependency. The following theorem indicates when structural sparsity exists. The proof can be found in [NNT01a].

Theorem 1 Let $C = D_1 \cup D_2 \cup ... \cup D_n, \cup M$ be an OLAP cube schema, where $D_1...D_n$ are dimension schemata and M the set of measure attribute, and let $K_1,K_2,...,K_n$ be the respective dimension key attributes. Let c be an OLAP cube over C, and assume that for each $B \in C$, $|v(B)| > 1$ in c. Then c is structurally sparse if and only if (i) C has a redundant dimension, or (ii) C contains RHS-intersection.

Storing empty cells is not meaningful hence efficient techniques for storing sparse data exist. The solution for the sparsity problem, however, is not that straightforward, since the need for the storage space can increase more than exponentially in situations

with sparse raw data and many pre-calculated aggregation values [Ped00]. As an example, in Figure 4 there are only three cells of original data but we need seven cells to store aggregated values although no hierarchies exist.

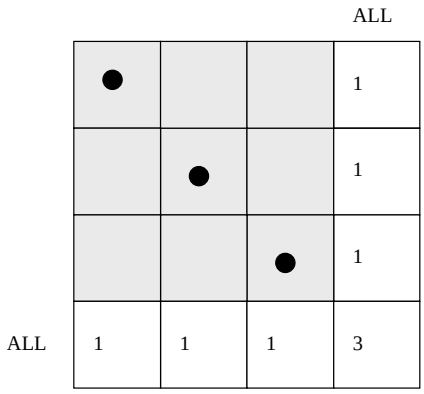

Figure 4. Pre-calculating aggregation values rapidly increase need of storage space

This kind of database explosion cannot be avoided using efficient storage methods for sparse data, because the real amount of data increases due to pre-calculated aggregation information. Solutions for this problem are to ensure that cubes storing the raw data are not sparse, cubes do not have too many dimensions, and not too many aggregations are pre-calculated.

Example 1 Consider a cube with three dimensions: `employee`, `customer`, and `product`. The measure is the quantity sold of each product. With no FDs, the structural sparsity is zero. However, if we know that one employee sells only one product, we get a FD `employee` $\rightarrow$ `product`. Now, the cube is necessarily sparse (assuming that there are more than one product), because most `product-employee` combinations will have an empty value. If we have 10 employees, 20 customers, and 5 different products, then the size of the cube is 1000 cells. We can slice the cube according to employees and get 10 slices. Each slice can have at most 20 occupied cells since an employee can sell only one product but possibly to all customers. Further, the total number of occupied cells in the whole cube is at most 10x20=200, while the total number of cells is 1000, thus structural sparsity is 80%. More generally, the structural sparsity is 1-1 / |v(`product`)|.

As a general-level rule of thumb, it can be said that creating more dimensions tends to increase structural sparsity.

2.1.2 Completeness

Cabibbo and Torlone [CaTo98] mean by completeness that all information stored in operational databases is incorporated in the OLAP cube. We restrict the concept of completeness to whether all information needed to answer the user's queries is incorporated in the OLAP cube. It is difficult to give general definition for completeness hence we define attribute-completeness, instead. Let U be the set of desired attributes for dimensions, given as input for logical OLAP cube design. We say that an OLAP cube schema C is *attribute-complete*, if each attribute of U is contained in some dimension schema. On the other hand, the cube should contain only relevant information for the user. Not to have too large cubes is also important for the effectiveness of the system.

As an example, suppose that we have such candidate dimensions D_1, …, D_n that the structural sparsity of a cube C would be 0. Suppose further that we leave out one of the dimensions, say D_k. Since the structural sparsity of C with D_k included would be 0, the underlying data warehouse would structurally have valid values for cells in the cube with dimensions D_1, …, D_n. This way, leaving out the dimension D_k will imply that the OLAP cube is not complete.

As a general-level rule of thumb, we can say that creating less dimensions tends to lead to non-completeness. This is opposite to controlling sparsity, hence, in this sense, sparsity and completeness may be conflicting design goals.

2.1.3 Summarizability

Correct summarizability means that the user does not get incorrect or misleading aggregated values in OLAP queries. There are three necessary conditions for summarizability [LeSh97]:

1. *Disjointness of category attributes*. The attributes in dimensions form disjoint subsets over the elements on a level. For example, assume that some shoes belong to both summer and winter collections. Now, then the user summarises shoes in both summer and winter collections, these particular shoes will be summarised twice, because they roll-up to two categories on the 'season' level.
2. *Completeness in hierarchies*. All elements occur in one of the dimensions and every element is assigned to some category on the level above it in the hierarchy. For example, then users summarise sales values on the 'state' level, the sales from Washington D.C. are lost, because Washington D.C. is not in a state. This implies that the hierarchy is not complete.

3. *Correct use of measure (summary) attributes with statistical functions*. This is the most complex of these requirements, since it depends on the type of the attribute and the type of the statistical function. Measure attributes can be classified into three different types, which are flow, stock, and value-per-unit. By knowing the type of an attribute, we can conclude which statistical functions can be applied to the attribute. For example, it is meaningful to calculate the sum of daily sales but not the sum of daily inventories. On the other hand, the mean is a relevant measure for both sales and inventories.

Figure 5 illustrates the first two items. The leaf level represents customer, the middle level customers in the same country, and the highest level all customers. If we operate on the country level or on the 'all' level, we obviously want that each customer is to be included exactly once.

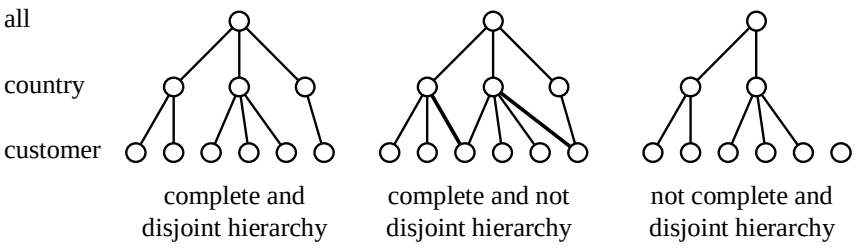

Figure 5. Examples of dimension hierarchies

Without any prior knowledge about the semantics of the attributes, we can force the completeness and disjointness conditions by requiring a functional dependency from a lower level to a higher level in the hierarchy.

2.2 Logical OLAP Design Based on Dependencies

In this section we first introduce a dependency based modelling method (Paper II) [NNT00], then OLAP normal forms and normalisation algorithms (Paper III) [NNT01a] are studied. The last section discusses dependencies in OLAP hierarchies (Paper IV) [NNT01b].

2.2.1 Representing Logical OLAP Schema

When using dependency based modelling methods, it is also important to represent dependencies in the conceptual schema. Our modelling method has two types of primitives: concepts and dependencies between concepts. A concept is an abstract entity that represents information. In this model, the intension of the concept, that is, its

information content, consists of all dependencies affecting the concept. The extension of the concept is the set of instances actualising the intension of the concept. The dependencies on the conceptual level are called conceptual or intensional dependencies, while dependencies among instances are extensional dependencies.

A universal relation with null values is used as the abstract model for the application area. The universal relation contains all concepts as its attributes. Values of attributes can be thought of as identifiers of concepts, however, we do not assume the existence of real object identifiers. Although the universal relation is not a realistic choice for the database schema, it gives a clear theoretical base by simplifying definitions of dependencies and derivations of interference rules. The idea is that one row represents only one dependency between instances. Attributes not participating in the dependency have null values in the row. A null value means that the value does not exist. A dependency is valid only if all attributes in the dependency have non-null values. In addition, we assume that unknown-value nulls do not exist in the application area.

In our model, conceptual dependencies are relationships among concepts. The set of all dependencies is a binary relation among the set of concepts. More specialised dependencies are sub-relations of it. The dependency relation is reflexive and transitive. A dependency on the conceptual level denotes that if there is a database instance, then the instance must obey the dependency.

The dependencies used in this work can be classified into four subtypes: functional (FD), key (KD), is-a, and structure (SD) dependencies. A key dependency is a symmetric functional dependency. The idea of the structure dependency comes from the hierarchical structure of the conceptual schema. The dependency denotes a connection between concepts, which cannot be expressed by functional or inclusion dependencies. The functional structure dependency is used to cover non-unary functional dependencies. Dependencies of the conceptual model have a one-to-one mapping into the dependencies of the relational model:

1. The conceptual functional dependency corresponds to the functional dependency.
2. The key dependency to the symmetric functional dependency.
3. The is-a dependency to the inclusion dependency.
4. The structure dependency denotes other kinds of dependencies.

The dependency information can be represented by a graphical schema. The dependency types are presented in Figure 6. A simple arrow always denotes a functional dependency, a two head arrow a key dependency, a triangle head arrow an is-a dependency, and an arrow with a diamond and a bold arrow head the structure

dependency. If an arrow has a diamond ending and the head of the arrow is simple, it denotes a functional structure dependency.

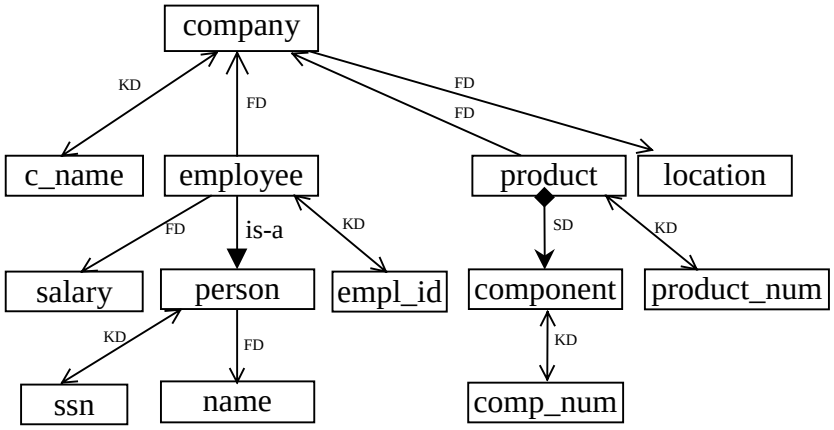

Figure 6. Example on conceptual schema of a company

We get the following dependencies from the schema in Figure 6:

Functional dependencies:
company→location
employee→company, salary
product→company
person→name

Key dependencies:
c_name↔company
product_num↔product
ssn↔person
comp_num↔component
empl_id→employee

Is-a dependencies:
employee is-a person

Structure dependencies:
product SD component

In our OLAP examples, we use a schema drawing technique simplified from the one presented above. Concepts or attributes are represented by text boxes and functional dependencies between attributes by arrows. A two-head arrow denotes a symmetric functional dependency, that is, the key dependency. Transitive dependencies are not drawn. Dimensions are presented by drawing an ellipse around the attributes of the dimension. The measure-id attribute is drawn with a bold rectangle. By the measure-id attribute we mean the attribute that represents and identifies the object whose properties or values are analysed. Attributes representing those particular values are called measure attributes. The measure-id attribute functionally determines all other attributes in the schema but the measure attribute does not. This is clear since, for example, for

one selling event there has to be exactly one product that is sold, one profit got from the product, one customer who bought the product, etc. The same profit can, of course, be related to many different selling events. When analysing the dependencies for cube design, the dependencies related to the measure-id attribute are not relevant. Therefore those dependencies are drawn with a dashed line. The actual value to measure is drawn by a bold rectangle. In some works measures are treated as dimensions. The measure still has a different role in OLAP thus we prefer to handle it separately from real dimensions in the graphical schema.

2.2.2 OLAP Normal Forms

Relational normal forms are hardly suitable for OLAP cubes, because of different goals in operational and OLAP databases. The main goal of the relational normal forms is to avoid update anomalies, while in OLAP the efficiency of queries is the most important issue. More specifically, the goals of OLAP normal forms are to ensure both minimal amount of sparsity and correctness of aggregations, but we mostly concentrate on controlling sparsity in this work.

In what follows, we study different kinds of dependency sets and their properties in the spirit of Nummenmaa [Num95]. It is known that certain types of dependency sets lead to bad relational database schemata. For example, it is impossible for certain dependency sets to construct an acyclic, functional dependency preserving, and lossless relational database schema in Boyce-Codd [Cod74] normal form. Now, we apply some of these ideas in OLAP design. The analogy to OLAP design is that for all dependency sets it is impossible to construct a non-sparse OLAP cube without loosing in expressiveness.

The problematic dependency combinations in traditional database design relate to three situations:

1. intersecting right hand sides (RHS-intersection),
2. intersecting nonredundant left hand sides (LHS-intersection), and
3. splitting, which is defined as follows. A left-hand side X splits a left hand side Y, if $Y \cap X^{\rightarrow} \neq \emptyset$ and we do not have $X \rightarrow Y$.

Respectively, we can find four types of problematic dependency sets for OLAP cubes:

1. functional dependencies between two dimensions,
2. functional dependencies between three or more dimensions,
3. RHS-intersecting dependencies between dimensions, and
4. dependencies having participating attributes from different dimensions.

Following Lehner et al., a general principle in OLAP design is that dimensions should be independent (*multidimensional normal form*), and inside a dimension there should be a single attribute key for the dimension (*dimensional normal form*) [LAW98]. It is also desirable that there exists only one measure value for each combination of dimension values similarly as the first normal form of relational database theory [Cod70]. While the aim of the dimensional normal form is to guarantee completeness in aggregations, the multidimensional normal form is mostly used in controlling sparsity of the OLAP cubes.

Our non-sparse normal form is an extension to Lehner's multidimensional normal form since the non-sparse normal form covers non-unary and RHS-intersectiong dependencies, too. The schemata in Figure 7 and in Figure 8 are in Lehner's multidimensional normal form, but they are not in our non-sparse normal form because of the dependency `employee product` $\rightarrow$ `country` in Figure 7 and the RHS-intersecting dependencies `customer` $\rightarrow$ `office` and `employee` $\rightarrow$ `office` in Figure 8.

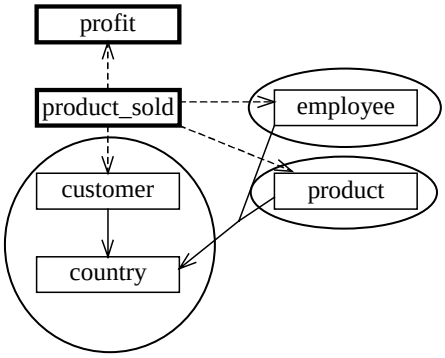

Figure 7. A non-unary dependency between dimensions `customer`, `employee`, and `product`

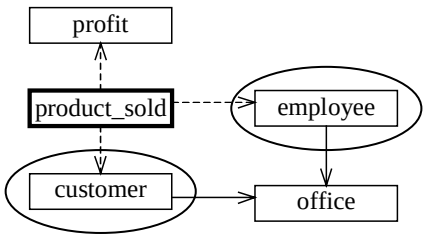

Figure 8. RHS-intersecting dependencies between dimensions `customer` and `employee`

2.2.3 Producing Normalised OLAP Schemata

Two different situations can be recognised in logical OLAP design. In the first one, the dimensions are known, and they will be decomposed based on FDs. In the second one,

only a conceptual schema with the dependency information exists and the dimensions need to be built. In the first case, it is natural to use a method based on relational decomposition and in the second case a method based on relational synthesis.

Our synthesis algorithm [NNT01a] first forms equivalence classes of attributes based on functional dependencies. Two attributes belong to the same equivalence class, if they participate in the same dependency. An attribute participates in a dependency if it exist in the left or the right hand side of the dependency. An equivalence class can form a dimension if it is in dimensional normal form. If only a composite key exists, the algorithm constructs an artificial attribute to represent the key.

Example 2 Figure 9 illustrates example input and output for Algorithm 1. The measure value is `profit` and potential dimensions are `time`, `customer`, and `product_lot`. `Employee` cannot be a dimension of its own, because the `product_lot` functionally determines it. We want to measure the profit of sold products, i.e. the `product_sold` is the concept to measure and `profit` the actual value to measure.

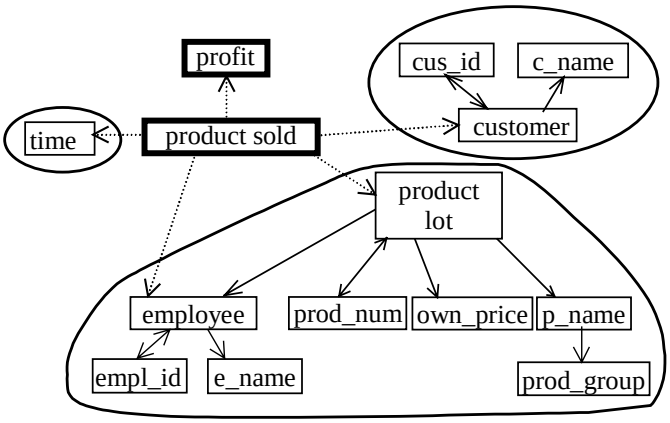

Figure 9. An OLAP schema of a company

Our second algorithm, the decomposition [NNT01a], works as follows:

1. *Dimension decomposition.* The dimensions are first constructed based on the idea that there has to be a single attribute key for a dimension. This is ensured by placing attributes A and B in different dimensions if there does not exist an attribute that functionally determines both A and B.
2. *Cube decomposition.* The dimensions constructed in Phase 1 are given as input. If there is a conflict against the desirable normal form between two dimensions, multiple cubes are constructed by placing the conflicting dimensions in different cubes. The process is repeated until all cubes are free from conflicts.

The extreme case is that the cubes will become one-dimensional. This, of course, is not desirable and it is better to allow a sparse result and to stop the process earlier.

Example 3 Normalising the schema in Figure 8 gives the OLAP schema illustrated in Figure 10. In the schema the measure is `product_sold`. Possible dimensions are: `product`, `employee`, `customer`, and `office`. Further, the relevant dependencies are: `customer` $\rightarrow$ `office`, and `employee` $\rightarrow$ `office`.

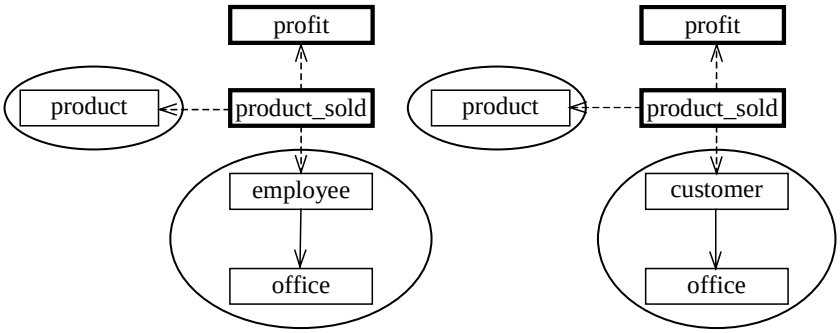

Figure 10. A normalised OLAP schema

2.2.4 OLAP Hierarchies

Hierarchical dimensions are an essential part of OLAP. The hierarchical structure allows the user to study facts in different levels of details. Good hierarchical structures ensure correct and efficient calculation of aggregation functions: consistent and coherent structure of hierarchies decreases logical errors while efficiency of calculation is increased, for example, by decreasing redundancy in rollup paths. Logical modelling methods of these hierarchies differ from the methods used in database design. Thus, dependencies used in database design are not enough for modelling OLAP hierarchies. We have studied which kinds of dependencies would be needed to get aggregation hierarchies more desirable for OLAP. In addition to the functional dependencies, three new dependency classes are given for data hierarchies. The anti-closure dependency prevents calculating redundant aggregation values by disallowing “shortcuts” in hierarchies, while non-raggedness and balance dependencies ensure complete aggregations on each level in a hierarchy.

Aggregation hierarchies can be divided into attribute hierarchies and data hierarchies. In attribute hierarchies, the attribute labelling a column in a dimension table will correspond to a level in the aggregation hierarchy of the corresponding cube dimension, and data values occurring in that column of the dimension table will become the members of the corresponding level. The data hierarchy arises when there is a

relationship between two attributes, like employee-manager. Attributes do not determine the hierarchy levels but the actual data do. In addition, a data hierarchy schema can contain an additional attribute used as a level identifier, for example a job grade. Table 2 and Figure 11 illustrate this.

Table 2. An example relation on management hierarchy

EMPID	MGRID	GRADE
1	NULL	A
2	1	B
3	1	B
4	1	C
5	2	C
5	1	C

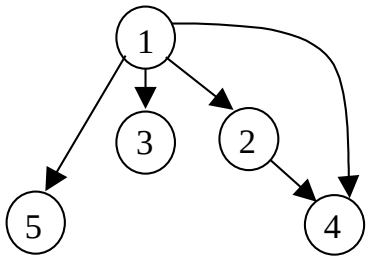

Figure 11. Directed acyclic graph representing the relation in Table 2

For the OLAP use, we have discovered six types of hierarchies that can be classified according to their generality. Figure 12 presents the hierarchy of these hierarchies. The most general is the acyclic digraph and the strictest is the balanced and non-ragged tree.

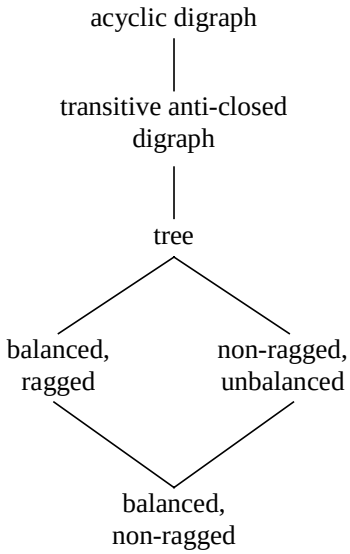

Figure 12. Hierarchy of dimension hierarchies

As Figure 12 indicates, we have six different kinds of hierarchies that have the following properties related to OLAP hierarchies:

(1) *Acyclic*. This is the most general class. It allows the day-month-quarter-year; day-year structure, where the day-year could be called a "direct shortcut". A direct shortcut means redundant aggregation paths. (Figure 11)

(2) *Transitive anti-closed digraph*. This is an acyclic graph with no direct shortcuts. If no direct shortcuts exist, then no redundant aggregation paths are possible. (Figure 13)

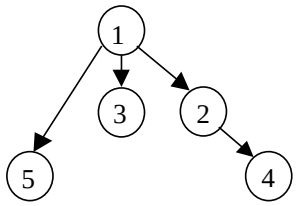

Figure 13. A transitive anti-closed graph

(3) *Tree*. Unique aggregation paths are guaranteed, that is, there is only one path from each node to the root node.

(4) *Unbalanced but non-ragged or balanced but ragged*. The available levels of aggregation may not be equal. (Figure 14 and Figure 15)

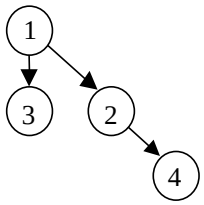

Figure 14. Unbalanced but non-ragged

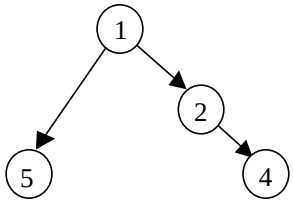

Figure 15. Ragged but balanced tree

(5) *Balanced and non-ragged*. Equal levels of aggregation are available. (Figure 16)

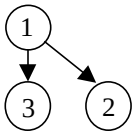

Figure 16. A non-ragged and balanced tree

In general, it can be said that a stricter hierarchy makes it easier to get correct aggregations. When a data hierarchy is balanced and non-ragged with consistent level identifiers, aggregations are complete on every level in the hierarchy. By consistent level identifiers we mean that the identifiers correspond to the actual level of a node.

2.3 Logical OLAP Design Based on Queries

We give two methods to define the data to be taken into the OLAP cube. The first method is based on the intensional concept theory (Paper V) [NNT01c] and the second on MDX queries (Paper VI) [NNT01d]. The aim of both methods is to enable the end user to produce easily good OLAP schemata with respect to sparsity, summarizabily, and completeness.

In this approach, it is assumed that a new OLAP cube is constructed for new analysis purposes, because one cube cannot be good for all kinds of queries. The ideal cube contains all information and only the information that is relevant for the user's queries. The methods require that actual data warehouse data are stored independently and the data are available for populating OLAP cubes. The aim is to reduce the level of technical knowledge that a user needs in order to construct an OLAP cube. It is meaningful to think that an OLAP query returns an OLAP cube. This gives a natural reason to combine the cube design and query construction.

2.3.1 Method Based on Intensional Query Method

The first method uses the algebraic representation of the intensional concept theory with a graphical conceptual schema [Kan93, Kau67, Nie00, NiNu01] and the information about the functional dependencies between the attributes of the data warehouse [NNT01a].

The algebraic representation of the intensional concept theory [Nie00] is based on the concept operations. In Figure 17, a conceptual schema and some resulting concepts of concept operations are shown. The most important concept operations, namely the intensional product (denoted by $\odot$), and its dual operation, the intensional sum (denoted by $\oplus$) are binary operations in the set of concepts. In terms of lattice theory, the product is the greatest lower bound and the sum is the least upper bound [DaPr90]. Therefore, the intensional sum $a \oplus b$ is the least concept containing all concepts which a or b or both contain, while the intensional product $a \odot b$ contains all concepts contained in both a and b. We assume that the concept system has a lattice structure, that is, the sum and product exist for all pairs of concepts.

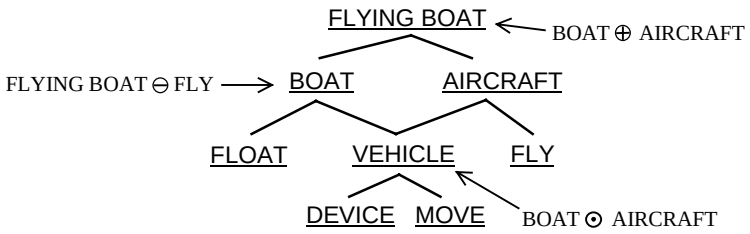

Figure 17. A Conceptual schema and the resulting concepts of some intensional concept operations.

(The concept above intensionally contains the concept below it.)

The basic relationship between concepts is the intensional containment relation that is defined as follows: a concept a intensionally contains a concept b, denoted by $a \succcurlyeq b$, if and only if $a \oplus b = a$. The intensional containment is an abstract relation that can be illustrated in a following way: if a concept a intensionally contains a concept b, then the intension of b is a part of the intension of a, and b can be called a characteristic of a [Kau67, Kan93].

The intensional query method [NiNu01] is based on intensional concept definition. While in traditional conceptual query systems the user generally defines a query using the existing concepts without actually forming new concept definitions in the conceptual schema, in our approach the query construction is equal to defining new concepts. This way, the query formulations also increase the conceptual information.

When using the method, the user can choose an existing concept or form a new concept by using the concept operations. These new concepts can be further used to construct new queries, namely new concepts. The approach is very strongly user oriented and produces declarative queries. In the respective database perspective, the expressive power of the approach is that of the project-join queries. In most cases, this is adequate when defining information for constructing an OLAP cube.

The OLAP design method based on the intensional query method has the following steps:

1. A high level conceptual schema of the data warehouse is shown to the user.
2. The user uses the intensional concept operations to construct the concept that represents the information of the OLAP cube under design.
3. The system constructs a normalised OLAP schema that contains the basic concepts contained in the given concepts as dimensions or in dimension hierarchies.
4. An OLAP cube will be shown to the user who can accept it or change it. For example, some attributes can be added or removed.
5. The final OLAP cube is presented as a graphical conceptual schema to the user who can pose queries against it using the same intensional query method.

2.3.2 Method Based on MDX Queries

The second method uses MDX (Multidimensional Expressions) queries. MDX is a declarative query language for multidimensional databases designed by Microsoft [Mic98]. The MDX query produces an OLAP cube (actually, a tabular presentation of a cube) given dimension specifications as input. A simple MDX select statement has the following form:

SELECT <axis specification> [, <axis specification>, ...]
FROM <cube specification>
WHERE <slicer specification>

Each axis specification describes how to produce one of the dimensions (axes) in the result. This includes specifying the point in the dimension hierarchy at which aggregation should be performed. The slicer specification allows the user to eliminate unwanted dimensions from the answer. In Example 4, a simple MDX query concerning sales data can be seen. The query is used in analysing how day, product group, and customer affect the profit got from products during the year 2000.

Example 4 An MDX query concerning a sales cube.

```
SELECT day, product group, customer id
FROM sales
WHERE profit, year.[2000]
```

In the select statement, the axes are associated with rows, columns, pages, etc. This facilitates presenting the result of a query in a tabular form. The axis or dimensions are specified by giving a set of members. MDX has several ways of performing it. The basic possibilities are to list the members or use MEMBERS, CHILDREN, or DESCENDANTS functions. An example about other features is the FILTER command that can be used to filter out member values of a dimension. The MDX language also contains a WITH clause but we ignore it in this work. The WITH clause can be used to construct member properties to give additional information on dimension members. For example, the product can have its colour as a member property. Member properties can be expressed as dimensions of their own or dimension levels but it is not efficient.

If a dimension is not mentioned in the axis definition, it is assumed to be a slicer dimension and the cube will be sliced according to the default member of the dimension. The slicer dimensions with the members to slice can also be given explicitly using the WHERE statement. For example, the user may be interested only in the sales of a particular year. The measure values to be viewed are also selected using the WHERE statement.

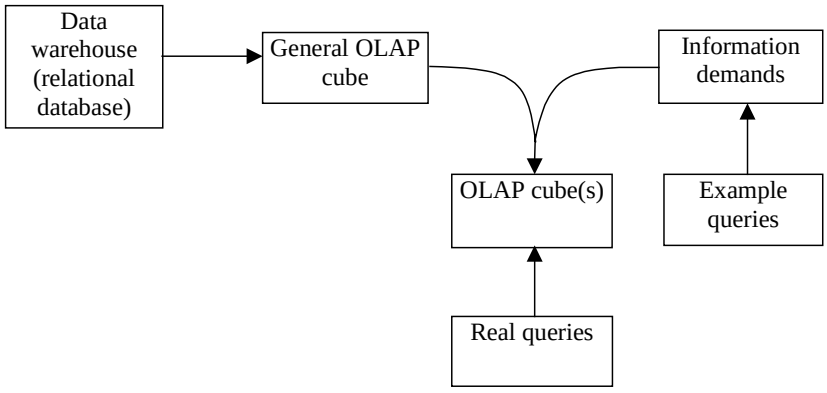

Figure 18. The basis of the method

In the OLAP design method, the idea is to construct equivalence classes of MDX queries and then construct an OLAP cube for each equivalence class. Two queries belong in the same equivalence class if they share attributes directly or transitively. Additionally, it can be required that the queries operate on the same hierarchy level. Figure 18 illustrates the method having the following steps:

1. The data warehouse is presented as a general base cube to the user.
2. The user constructs queries using the general data cube.
3. The system constructs an OLAP cube that corresponds to user's queries. The cube construction method is based on the use of functional dependency information.
4. An OLAP cube will be shown to the user and it can be accepted, modified, or rejected. The modifications can be done by giving more queries or changing the cube directly, for example, some attributes can be added or removed.
5. The system analyses the changes and informs the user about the goodness of the cube.

The resulting cube will be used in actual data analysis.

3 Conclusions

We have studied methods for improving logical OLAP design. The presented methods give a good basis for implementing software tools that can partly automate the OLAP cube construction process. We have recognised three aims for OLAP design: minimal amount of sparsity, completeness, and summarizability. However, the design process is not straightforward, because reducing sparsity and maintaining completeness may be contradictory goals.

Our methods are based on the relational dependency theory and the intensional concept theory. To improve logical design of OLAP cubes, we have developed a method to estimate the structural sparsity of OLAP cubes implied from functional dependencies. For controlling structural sparsity of OLAP cubes, a new OLAP normal form has been defined based on the idea that if an OLAP cube schema is in non-sparse normal form, it is known that its structural sparsity is zero. To produce normalised OLAP cubes synthesis and decomposition methods have been developed. The methods return a normalised OLAP schema given attributes and functional dependencies as input. The normalisation algorithms constructs dimensions with their hierarchies based on functional dependencies. Since the structure of dimension hierarchies can be complex, the attribute and data hierarchies have been identified and different kinds of hierarchical structures have been studied. Moreover, new dependencies for hierarchical OLAP dimensions have been developed.

To enable all users, despite of their level of expertise in OLAP, to design desirable OLAP cubes, we have given two design methods based on the use of query information. The first method, based on the intensional concept theory, combines database modelling and querying. The method operates on the conceptual level using the concepts of the application area in a graphical conceptual schema. This makes the method both intuitive and easy to use. The second method uses MDX queries posed against a general base cube representing the contents of a data warehouse. The user defines his/her information requirements by constructing MDX queries as examples of the information requirements. The method is also capable to use real queries when the user wants to improve existing cubes.

4 Overview of Publications

This thesis consists of six publications in the field of conceptual data management. The first two publications are background studies for the last four studies of logical OLAP design.

The first paper, “A query method based on intensional concept definition”, contains an application of the intensional concept theory for query purposes. In the intensional concept theory, the concepts are treated as algebraic objects to which algebraic operations can be applied [Kau67, Nie00]. The intensional query method, presented in the paper, combines conceptual modelling and query construction. The idea is that the user can use the same methodology in both of them. In this way, defining new queries also increases the intensional information in the application area. As a basis for data storage, we use the relational database model [Cod70], which also gives a clear and well-studied base to the data manipulation. The user can accomplish all tasks from database design to queries on the conceptual level, since the system takes care of all technicalities.

The second paper, “Applying dependency theory to conceptual modelling”, presents a modelling method based on the dependency information among concepts. A problem in many modelling techniques has been the lack of a formal theory. The basic elements of the models are entities and relationships between them. However, relationships do not have exact meaning, partly because they are usually described using natural language. We will give a formal basis for our modelling method by applying the well-studied relational dependency theory. The use of formal dependencies makes it possible to derive inference rules that enable us to derive new dependencies or new concepts. This is a clear benefit compared to many conventional modelling methods. The dependency based conceptual schema is also easy to transform into a database schema or use as an input for the OLAP design algorithms.

The third paper, “Normalising OLAP Cubes for Controlling Sparsity”, presents how functional dependencies can be applied in logical OLAP design. The main aim is controlling sparsity. New normal forms for OLAP cubes are defined and synthesis and decomposition algorithms to produce normalised cubes are given.

The fourth paper, “Logical multidimensional database design for ragged and unbalanced aggregation hierarchies”, contains a study on modelling hierarchical OLAP dimensions. Hierarchical dimensions have an essential role in OLAP and badly structured dimensions can easily lead to incorrect aggregations or redundancy in calculations.

In the fifth paper, “Applying intensional concept theory to OLAP design and queries”, the intensional query method is applied to logical OLAP design and OLAP queries. When using the presented method, the user can employ a graphical conceptual schema to define the concepts taking into the OLAP cube schema. After that the system generates a normalised OLAP schema and shows it to the user using the same graphical formalism. Then the user can define queries using the graphical schema that follows the same formalism as the original conceptual schema. The method is intuitive and does not require learning complex query languages.

In the sixth paper, “Constructing OLAP cubes based on queries”, the use of MDX queries in designing OLAP cubes is studied. The method is based on the use of example queries against a data warehouse that is represented by a general OLAP cube. If real queries are available, they can be used to optimise existing cubes.

References

[CaTo98] Cabibbo, L. and Torlone, R.: A logical approach to multidimensional databases, 6th International Conference on Extending Database Technology (EDBT'98), G. Alonso (eds.), LNCS, Vol. 1377, pages 183-197, Springer, 1998.

[CCS93] Codd, E., Codd, S., and Salley, C.: Providing OLAP to user-analysts: An IT mandate. Technical report, 1993. Available at http://www.hyperion.com/whitepaper/whitepaperreq.cfm

[Che76] Chen, P. P.: The Entity-Relationship model: Toward a Unified View of Data, *ACM Transactions on Database Systems*, 1(1), pages 9-36, 1976.

[Cod70] Codd, E. F.: A relational model for large shared data banks, *Communications of the ACM*, 13(6), pages 377-387, 1970.

[Cod72a] Codd, E. F.: Further Normalization of the Data Base Relational Model, *Data Base Systems, Courant Computer Science Symposia Series 6,* R. Rustin (ed.), pages 33-64, Prentice-Hall, 1972.

[Cod74] Codd, E. F.: Recent Investigations into Relational Data Base Systems, *Proceedings of IFIP Congress*, Stockholm, pages 1017-1021, IFIP, North-Holland, 1974.

[DaPr90] Davey, B. A. and Priestley, H. A.: *Introduction to Lattices and Order*, Cambridge University Press, 1990

[Dat00] Date, C. J.: *An Introduction to Database Systems*, 7th edition, Addison-Wesley, 2000.

[Fag77] Fagin, R.: Multivalued dependencies and a new normal form for relational databases, *ACM Transactions on Database Systems*, 2(3), pages 262-278, 1977.

[HMV99] Hurtado, C., Mendelson, A., and Vaisman A.: Maintaining data cubes under dimension updates, *Proceedings of the 15th International Conference on Data Engineering*, pages 346-355, IEEE Computer Society Press, 1999.

[Kan93] Kangassalo, H.: COMIC: A system and methodology for conceptual modelling and information construction, Data & Knowledge Engineering, 9:287-319, 1993.

[Kau67] Kauppi, R.: *Einführung in die Theorie der Begriffssysteme*, Acta Universitatis Tamperensis, Ser. A, Vol. 15, University of Tampere, Tampere, 1967.

[LAW98] Lehner, W., Albrecht, J., and Wedekind, H.: Normal forms for multidimensional databases, *Proceedings of the 10th International Conference on Scientific and Statistical Data Management (SSDBM'98)*, M. Rafanelli and M. Jarke (eds.), pages 63-72, 1998.

[LeSh97] Lenz, H.-J. and Shoshani, A.: Summarizability in OLAP and Statistical Data Bases, Ninth International Conference On Scientific And Statistical Database Management (SSDBM), pages 132-143, 1997.

[Mic98] *Microsoft OLE DB for OLAP Programmer's Reference*, Microsoft Corporation, 1998.

[Nie00] Niemi, T.: New approaches to intensional concept theory, *Information Modelling and Knowledge Bases XI*, E. Kawaguchi et al. (eds.), IOS Press, 2000.

[NNT00] Niemi, T., Nummenmaa, J., and Thanisch, P.: Applying dependency theory to conceptual modelling, *Topics in Conceptual Analysis and Modeling*, O. Majer (ed.), pages 271-290, Czech Academy of Sciences' Publishing House Filosofia, Prague, 2000.

[NiNu01] Niemi, T. and Nummenmaa, J.: A query method based on intensional concept definition, *Frontiers in Artificial Intelligence and Applications: Information Modelling and Knowledge Bases XII*, H. Jaakkola and H. Kangassalo (eds.), Vol. 67, pages 92-106, IOS Press, 2001.

[NNT01a] Niemi, T., Nummenmaa, J., and Thanisch, P.: Normalising OLAP Cubes for Controlling Sparsity, 2001, submitted to Data & Knowledge Engineering. (An earlier version of this work is: Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Functional dependencies in controlling sparsity of OLAP cubes, *Data Warehousing and Knowledge Discovery, Second International Conference, DaWaK 2000*, Y. Kambayashi et al. (eds.), Lecture Notes in Computer Science, Vol. 1874, pages 199-209, Springer, 2000.)

[NNT01b] Niemi, T., Nummenmaa, J., and Thanisch, P.: Logical multidimensional database design for ragged and unbalanced aggregation hierarchies, *The Proceedings of the International Workshop on Design and Management of Data Warehouses, DMDW'2001*, D. Theodoratos et al. (eds.), pages 7-1 - 7-8, 2001.

[NNT01c] Niemi, T., Nummenmaa, J., and Thanisch, P.: Applying intensional concept theory to OLAP design and queries, *The Proceedings of The 11th European-Japanese Conference on Information Modelling and Knowledge Bases*, H. Kangassalo et al. (eds.) pages 142-153, 2001.

[NNT01d] Niemi, T., Nummenmaa, J., and Thanisch, P.: Constructing OLAP cubes based on queries, *DOLAP 2001, ACM Fourth International Workshop on Data Warehousing and OLAP*, J. Hammer (ed.), pages9-15, ACM, 2001.

[Num95] Nummenmaa, J.: *Designing Desirable Database Schemes*, Department of Computer Science, University of Tampere, A-1995-1, Tampere, 1995. (Ph.D. Thesis)

[Ped00] Pendse, N.: Database Explosion, *The OLAP Report*, 2000. Available at http://www.olapreport.com/DatabaseExplosion.htm.

[Ull80] Ullman, J.: *Principles of Database Systems, Computer Science Press*, 1980.

[ZuSi99] Zurek, T. and Sinnwell, M.: Data warehousing has more colours than just black & white, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, M. Atkinson et al. (eds.), pages 726-729, Morgan Kaufmann, 1999.

PAPER I

Niemi, Tapio and Nummenmaa, Jyrki: A query method based on intensional concept definition, Frontiers in Artificial Intelligence and Applications: Information Modelling and Knowledge Bases XII, H. Jaakkola and H. Kangassalo (eds.), Vol. 67, pages 92-106, IOS Press, 2001.

PAPER II

Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Applying dependency theory to conceptual modelling, Topics in Conceptual Analysis and Modeling, O. Majer (ed.), pages 271-290, Czech Academy of Sciences' Publishing House Filosofia, Prague, 2000.

PAPER III

Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Normalising OLAP Cubes for Controlling Sparsity, 2001, submitted to Data & Knowledge Engineering. (An earlier version of this work is: Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Functional dependencies in controlling sparsity of OLAP cubes, Data Warehousing and Knowledge Discovery, Second International Conference, DaWaK 2000, Y. Kambayashi et al. (eds.), Lecture Notes in Computer Science, Vol. 1874, pages 199-209, Springer, 2000.)

PAPER IV

Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Logical multidimensional database design for ragged and unbalanced aggregation hierarchies, The Proceedings of the International Workshop on Design and Management of Data Warehouses, DMDW'2001, D. Theodoratos et al. (eds.), pages 7-1 - 7-8, 2001.

PAPER V

Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Applying intensional concept theory to OLAP design and queries, The Proceedings of The 11th European-Japanese Conference on Information Modelling and Knowledge Bases, H. Kangassalo et al. (eds.) pages 142-153, 2001.

PAPER VI

Niemi, Tapio, Nummenmaa, Jyrki, and Thanisch, Peter: Constructing OLAP cubes based on queries, *DOLAP 2001, ACM Fourth International Workshop on Data Warehousing and OLAP*, J. Hammer (ed.), pages 9-15, ACM, 2001.