Constructing OLAP Cubes Based on Queries

Tapio Niemi
Department of Computer and information Sciences
FIN-33014 University of Tampere
Finland
+358 32156595
tapio@cs.uta.fi

Jyrki Nummenmaa
Department of Computer and Information Sciences
FIN-33014 University of Tampere
Finland
+358 405277999
jyrki@cs.uta.fi

Peter Thanisch
Department of Computer Science, University of Edinburgh
Edinburgh, EH9 3JZ
Scotland
+44 7968401525
pt@dcs.ed.ac.uk

ABSTRACT

An On-Line Analytical Processing (OLAP) user often follows a train of thought, posing a sequence of related queries against the data warehouse. Although their details are not known in advance, the general form of those queries is apparent beforehand. Thus, the user can outline the relevant portion of the data posing generalised queries against a cube representing the data warehouse.

Since existing OLAP design methods are not suitable for non-professionals, we present a technique that automates cube design given the data warehouse, functional dependency information, and sample OLAP queries expressed in the general form. The method constructs complete but minimal cubes with low risks related to sparsity and incorrect aggregations. After the user has given queries, the system will suggest a cube design. The user can accept it or improve it by giving more queries. The method is also suitable for improving existing cubes using respective real MDX queries.

Categories and Subject Descriptors

H.2.1 **[Database Management]:** Logical Design – *data models, normal forms, schema and subschema*

General Terms

Management, Design

Keywords

Logical OLAP design, MDX queries, data warehousing.

1. INTRODUCTION

On-Line Analytical Processing (OLAP) is a method to support decision making in situations where raw data on measures such as sales or profit needs to be analysed at different levels of statistical aggregation. In OLAP, queries are made against multidimensional cubes, called OLAP cubes.

The design of a cube is based on knowledge of the application area and the types of queries the users are expected to pose. Since constructing an OLAP cube can be a difficult task for the end user, it is often seen as the duty of the data warehouse administrator. This has led to researchers and vendors regarding the OLAP cube as a static storage structure for data warehouse data. This is problematic since the user often wants to make new kinds of queries, which may also need new OLAP cubes. The same cube is not always practical for different analysis tasks, since the structure of the cube has a notable effect on efficiency and ease of posing queries. Moreover, even professionals may have difficulties in cube design since generally accepted logical design methods do not exist so far. In general, the structure of the resulting OLAP schema should be constructed by taking into account the information about the dependencies among the concepts and other meta knowledge on the concepts and the application area, and the query requirements.

We believe that the user should be able to design customised cubes which are the best suitable for his/her current analysing tasks. This is important since the analysis tasks or the data in the data warehouse can change almost daily and the cube design, of course, should follow these changes. Moreover, some users may also utilise planning methods in which a cube is designed for a hypothetical situation, for example to analyse a new organisation structure beforehand. It is also worth mentioning that business intelligence people work in multi-user client/server environments where there can be major advantages to having the cube on the desktop machine rather than on an occasionally slow server.

In our approach, we assume that a new OLAP cube is constructed for new analysis purposes. For example, one user might have detected some interesting facts in a subcube of an existing cube. The user might then pass on this information to a specialist who needs to examine the data located in the subcube in greater detail. We also assume that actual data warehouse data is stored independently and that it is available for populating OLAP cubes. Our aim is to reduce the level of technical knowledge that a user needs in order to construct an OLAP cube. We combine the cube design and query construction. There is a natural connection between OLAP cubes and queries, since it is meaningful to think that an OLAP query returns an OLAP cube. An ideal cube contains all information and only the information that is relevant to the user's queries. In addition, the structure of the cube should be such that, for example, aggregations are not misleading and the level of sparsity is low. The method has 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, e.g. 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. Figure l illustrates the method.

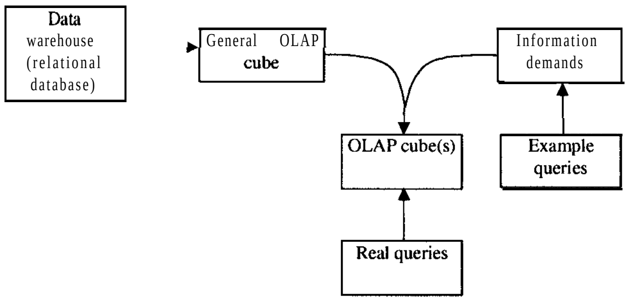

Figure 1. The basis of the method

The paper is organised as follows: In Section 2 we study related work. The aims of logical OLAP design and a method how the user can present the information requirements are studied in Section 3. In Section 4, we show how an OLAP cube can be constructed based on the users' information requirements. Finally, Section 7 contains conclusions and future work.

2. RELATED WORK

Most of the work in OLAP has been concentrated on efficient implementation. In some works also the OLAP design has been studied but the methods presented may be too complicated and time consuming for non-professionals.

Cheung et al. have studied requirement-based cube design [3]. Their aim is to find such data cubes to be materialised that maximises query performance. Thus, the method can be seen as an optimisation method rather than a logical cube design method. The optimisation is based on queries that users are assumed to pose to the system. In the method, each user query is represented as a data cube and the aim of the optimisation process is to modify the cube set (e.g. by merging cubes) in order to find a better set compared to query performance.

Selfridge et al. divide business data analysis into data exploration and data analysis [13]. In data exploration the aim is to find a relevant subset of data to analyse, while in analysis the aim is to find an answer to a business question. Our method is based on a similar kind of idea: choosing data for the OLAP cube is analogous to data exploration and studying the cube to data analysis.

Zurek and Sinnwell list some non-standard requirements for data warehouses [14]. Realignment of a data warehouse schema is needed quite often. Especially, dimension hierarchies can change, even regularly. Their second point is that also detailed level data is sometimes needed, while, in general, data loaded into a data warehouse is summarised. It is often assumed that a data warehouse is quite static and updated, for example, once a day or week. However, in many situations this is not enough and the warehouse should be much more up to date. Further, in some business planning situations hypothetical planning data is loaded into a data warehouse. This data can be changed during the iterative planning process thus efficient updating is necessary. Furthermore, in some OLAP applications, such as enterprise resource planning, the design of the cube itself may need to be changed on a daily basis.

Sapia studies modelling and predicting query behaviour in OLAP systems [1 l]. He presents a formal model and a graphical notation for studying interaction patterns in OLAP systems. The knowledge about the user behaviour can be applied in logical and physical OLAP design and also to make query processing more effective by predicting queries. The importance of the logical schema in OLAP is much greater than in traditional database systems, since the logical OLAP schema is usually used as a user interface. Therefore, the logical schema determines the queries that can be posed.

The entity/relationship model (ER) [2] has been applied to cube design. For example Sapia et al. [12] extend the ER model by adding three primitives into it in order to allow expressing the multidimensional structure of data: a fact relationship set, a dimension level set, and a classification set. However, the method does not offer much help for the design process but the designer has to know what kind of structure is suitable for the current application. Further, the model does not have features to help to achieve the desirable properties of the OLAP cube (summarizability, efficiency).

Golfarelli et al. [5] present a semi-automatic modelling method (called Dimensional Fact model) based on an existing ER schema. It can automatically find possible dimension hierarchies and allows the user to modify them. Summarizability is taken into account; measure attributes are classified additive, semi-additive, and non-additive. Measures are supposed to be atomic. To ensure completeness in aggregations, a many to one mapping has to be hold between levels in a dimension. The algorithm for building dimensions may help to get orthogonal dimensions, but this is not studied in the paper. However, some relationships may be lost while constructing dimensions, and they may cause problems with orthogonality.

A method by Cabibbo and Torlone [I] is based on a multidimensional data model, called MD, whose main parts are a dimension and a fact table. The method has some similarities to the method of Golfarelli et al. They both start from the existing ER schema of an operational database.

Lehner et al. [6] present normal forms for multidimensional databases. The normal forms are designed to help in ensuring summarizability and controlling sparsity. The dimensional normal form requires that there exists a functional dependency between the hierarchical levels in a dimension, and in the multidimensional normal form additionally no functional dependency is allowed between different dimensions. The multidimensional normal form helps to avoid problems with incorrect aggregation and sparsity but the normal form itself may sometimes be impossible to achieve in practise, because in many applications it is not always possible to find totally independent dimensions. The orthogonality of dimensions is studied only with respect to unary functional dependencies, and summarizability is not understood in as large a sense as in the paper by Lenz and Shoshani [7].

Niemi et al. have studied normal forms for OLAP cubes [9]. The aim is to reduce structural sparsity caused by functional dependencies between dimensions. The authors have studied how different kinds of dependency sets affect sparsity. Moreover, synthesis and decomposition methods have been presented to produce normalised cubes.

3. EXPRESSING INFORMATION NEEDS

In many OLAP applications, the user does not exactly know in advance what data is going to prove to be relevant to answering their questions. For example, they may intend to explore the data trying to find causalities. A data warehouse usually contains too much information for one cube and all this information is seldom relevant for the user. A method is needed to limit the size of the resulting OLAP cube for both user oriented and technical reasons.

3.1 Formal OLAP Model

We use the relational model [4] to represent the queries and the data stored in a data warehouse. We have to formalise our notion of the OLAP cube. The cube consists of measures and dimensions. The measure is the value under analysis. The dimensions are coordinates for the measure in the hypercube, i.e. the dimension values determinate the measure value. It is not necessary to differentiate measure and other dimensions in the logical model; we can handle the measure just as a dimension. The dimension can have a hierarchical structure consisting of different levels. This enables the user to study data at different levels of details by drilling down or rolling up in the hierarchy.

Our presentation is based on relational database theory and we assume that the reader is familiar with the basics of it.

Definition 1 A dimension schema D is a set of attributes. We assume that there exists a single attribute key $K \in D$, which is called a dimension key for D.

If D is a dimension schema then a relation over d is a dimension, that is, a dimension is an instance of a dimension schema. It is generally assumed that each dimension schema D_i is chosen in such a way that there exists a single-attribute key K_i, although theoretically this would not need to be so (it is always possible to construct an artificial key). The functional dependencies among attributes in a dimension form a hierarchy (we assume antisymmetry) such that the dimension key attribute is on the first level of the hierarchy, attributes directly determined by the key are on the second level, etc. By the dimension level we mean an abstraction level in a dimension hierarchy. For example, if the items on the lowest level are shops, they can first rollup to the town level and then to the country level. The attributes representing the levels of dimensions are called dimension level attributes. In the example above, the dimension level attributes are town and country.

Definition 2 Let $D_1, \ldots, D_n$ be dimension schemata, $K_1, \ldots, K_n$ their keys, and M a set of attributes called measures. The OLAP cube schema is a relation schema $H = D_1 \cup D_2 \cup \cdots \cup D_n \cup M$, and $\{K_1, \ldots, K_n\}$ is a superkey for H. An OLAP cube is a relation over the schema H. Further, an OLAP schema is a set of OLAP cube schemata.

3.2 Logical Design Goals

We adopt three criteria with which we assess the quality of an OLAP cube design: completeness, correct aggregations, and minimal sparsity. In this section they will be studied in more detail.

Completeness and Minimalism. Cabibbo and Torlone [1] 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. On the other hand, the cube should contain only relevant information for the user. The avoidance of overlylarge cubes is also important for the effectiveness of the system.

Correct Aggregations. Lenz and Shoshani have studied summarizability, i.e. correctness of aggregations, in OLAP [7]. They give three necessary conditions for summarizability, and they also assume that these conditions are sufficient. The conditions are:

1. disjointness of categories in hierarchies,
2. completeness in hierarchies, and
3. correct use of measure (summary) attributes with statistical functions.

Disjointness requires that attributes in dimensions form disjoint subsets (categories) over the elements on a level. Completeness in hierarchies means that all elements occur in one of the dimensions and every element is assigned to some category on the level above it in the hierarchy. Correct use of measure attributes with statistical functions 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. Knowing the type of an attribute we can conclude which statistical functions can be applied to the attribute.

Minimal Sparsity. Sparsity, in general, can refer to two different characteristics of the cube design.

1. The proportion of cube cells containing zeros or nulls because there is no measure value for the combination of attribute values that collectively constitute the coordinates of the cell, and
2. The ratio of the raw data values to the number of aggregation values, a function of the hyper-surface area of the cube.

The first kind of sparsity is affected by, for example, decisions about whether to make an attribute an aggregation level within a dimension, or a dimension in its own right. The second kind of sparsity is affected by design decisions such as whether to split one dimension into two dimensions. This type of sparsity increases if the number of dimensions increases, because adding dimensions increases the hyper-surface area of the cube. We are primarily interested in the first kind of sparsity, as the following definition shows.

Definition 3 The sparsity of an OLAP cube is the ratio: (the number of empty cells) / (the total number of cells).

This way, sparsity is one if all cells are empty and zero if all cells are occupied. Sparsity clearly depends on actual data. However, functional dependencies may imply potential sparsity.

Definition 4 The ***structural sparsity*** of an OLAP cube is the amount of sparsity implied from the functional dependencies.

3.3 Graphical Formalism

In our examples, we use a schema drawing technique simplified from the one presented by Niemi et al. [IO]. Text boxes represent concepts or attributes, and arrows functional dependencies between the attributes. Transitive dependencies are not drawn. The method is chosen, because it enables us to show the dependency information explicitly. An example of the graphical formalism is shown in Figure 2. The measure-id attribute, i.e. the attribute that represents and identities the event whose properties are under analysis, is drawn with a bold rectangle. When analysing dependencies for cube design the dependencies related to the measure-id attribute are not relevant. Therefore those dependencies are drawn with a dashed line.

3.4 Queries

The user's information needs are expressed as a set of general queries against a base cube representing the data warehouse. This does not mean that we know the actual queries beforehand. We only assume that a general form of queries and the attributes of the data warehouse to which the queries refer are known. We suppose that an OLAP cube or a set of cubes can be seen as an answer to this query set. Therefore it is meaningful to design the OLAP cube using query information.

3.4. I Multidimensional Expressions MDX

Multidimensional Expressions (MDX) is a declarative query language for multidimensional databases designed by Microsoft [8]. Because of lack of space, we give only a brief introduction to the syntax of MDX. 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 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. The cube specification is not relevant in this work, because we implicitly assume that all queries are posed to one base cube. Thus, we ignore the FROM clause in the rest of this paper. In Example 2 we can see a simple MDX query.

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. Since we use MDX in specifying information for the OLAP cube, we are not interested in tabular presentations and this feature of MDX can be omitted. The axis or dimensions are specified by giving the set of members. MDX has several ways to do 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, which give additional information on dimension members. For example, the product can have its colour as a member property. Member properties can be expressed as own dimensions 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.

3.4.2 Constructing Dimensions and Their Hierarchies for Base Cube

To be able to present OLAP queries we must have a general OLAP cube. We construct the cube such that the closure of each attribute in a data warehouse forms a dimension. The attribute itself is a key for the dimension and other attributes in its closure (i.e. the attributes that the attribute functionally determines) form the dimension hierarchy. We do not distinguish between measures and other dimensions since MDX handles them in a similar way. Moreover, for simplicity, we ignore property attributes, too.

Definition 5 Let assume that each attribute A_i of a data warehouse forms a dimension schema D_i such that $D_i = A_i^+$. Clearly, A_i is a key for Dr. A ***base OLAP cube schema*** used in queries is a relation schema H= $D_1 \cup D_2 \cup \cdots \cup$ D,,, where D_1, D_2, ... D_n are the dimensions.

Example 1 We have a schema about a company selling products in Figure 2.

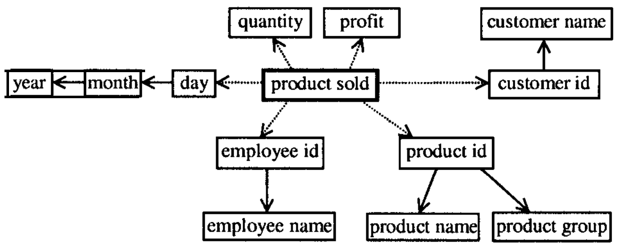

Figure 2. A company example

According to Definition 5, we will get the following dimensions: customer id, customer name; customer name; product id, product name, product group; product name; product group; employee id, employee name; employee name; day, month, year; monthyear; year; quantioy;f i t .

The keys of the dimension schemata are underlined. We assume that each dimension has a unary key. Dimension level attributes and property attributes are not separated but in the final implementation property attributes can be taken into account, if the used implementation system supports them.

Example 2 An MDX query related to the schema in Figure 2.

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

The query in Example 2 corresponds to a three-dimensional cube where profit is the measure and day, product group, and customer id are the dimensions.

4. CONSTRUCTING CUBES

To construct an OLAP cube we need to detect the measure(s) and the dimensions from the set of example MDX queries. A cube (or a set of cubes) is called ***complete*** if all given queries can be answered and ***minimal*** if the cube, or cubes, contain no data other than that needed to answer the queries and their subqueries. By a *subquery* we mean such a query whose answer is included in the answer of another query in the given query set. Our aim, of course, is to construct both complete and minimal cubes.

4.1 Similarity of Queries

The user's information requirements are presented as a set of MDX queries posed against the virtual cube representing the contents of the whole data warehouse. For constructing a logical OLAP schema, we need only the set of dimension attributes used in the SELECT clause and the measure in the WHERE clause of the MDX query. However, we do not use the measure attributes while forming the logical structure for the cube, the measure attributes are only included as dimensions in the final cube schema. The information in the FILTER and slicing attributes in the WHERE clauses are needed when populating the cube, but it is out of our scope in this work. Our aim is to find a schema for an OLAP cube or a set of cubes that represent answers to the given query set. We have three possibilities of how many cubes to construct:

1. All queries represent one OLAP cube.
   - Too large a cube is undesirable for technical reasons and it can also be confusing for the user.
2. Each query represents a cube of its own.
   - This restricts analysing power and may lead to redundancy in the use of the storage space.
3. 'Similar' queries represent a common cube.
   - This is the best possibility but "similarity" of queries can be difficult to judge and a small set of example queries can lead to too many cubes.

The similarity of queries can be defined in many different ways. We assume that queries are similar if they share a dimension directly or transitively and if they operate on the same hierarchy level. However, in this phase we distinct the measure attributes and dimension attributes, i.e. the queries are not treated similar if they only have the same measure. Since the size of a cube increases if we needed to add more dimensions, it is often more economical to construct different cubes in cases there the measure is the only connecting thing between queries. Moreover, some often-used dimensions having a special role, like time or geography, may be ignored when constructing equivalence classes. For example, the time dimension is used in the most of the queries but it does not imply that the queries have anything else in common. The following algorithm groups queries to equivalence classes, for which OLAP cubes will be constructed. The normalisation process can still construct more cubes than equivalence classes.

Algorithm 1 Dividing MDX queries into equivalence classes

```
Input:  A base OLAP cube schema H, a set Q of MDX queries
        against H, and a set of functional dependencies over
        the attributes in H.
Output: A set of equivalence classes of queries in Q.

1. For each query $Q_i$ in Q, place attributes of the SELECT
   clause to a set $X_i$. Ignore too general dimensions, e.g. time or
   geography.
2. For each set $X_i$, based on the functional dependency
   information, construct the set $Y_i$ of dimension keys of
   dimensions of H, such that $Y_i$ contains a dimension key K if
   there exists such an attribute $A \in X_i$ that $A \in K^+$.
3. Construct equivalence classes for the queries as follows:
     Two queries Q and Q' belong to the same equivalence
     class E if we can form a sequence of existing queries
     $< Q_0 = Q, Q_1, \ldots. Q_n, Q' = Q_{n+1} >$ such that $Y_i \cap Y_{i+1} \neq 0$,
     $0 \leq i \leq n$, where $Y_i$ denotes the dimension key set of the
     query $Q_i$.
4. (An additional phase to improve efficiency for analysing less
   detailed cubes):
   For each equivalence class $E_i$:
     For each query Q in $E_i$:
       If there is a query $Q' \in E_i$ with the same dimensions as
       Q but some dimensions of Q' are in more detailed
       levels than in Q, then construct a new equivalence
       class E' as follows: $E' = E_i - \{Q'\}$.
5. Output the set of equivalence classes obtained.
```

The algorithm places queries in the same equivalence class if they directly or transitively have common dimensions. These equivalence classes can be further divided to improve performance of less detailed queries by constructing a different cube for a less detailed query set. This cube will be smaller and more efficient to use.

Example 3 A user wants to analyse how different variables affect profit and quantity. This can be represented, for example, by the following three MDX queries against the base cube presented in Example 1:

- Q_1: SELECT day, product group
  WHERE profit, year.[2000]
- Q_2: SELECT employee id, customer id
  WHERE profit
- Q_3: SELECT year, product id
  WHERE quantity

In the first phase of Algorithm 1 the attributes in the select clause are found. We get sets X_1 = (day, product group), X_2 = {employee id, customer id), and X_3 = (year, product id). The corresponding dimension key sets are: Y_1 = (day, product id), Y_2 = [employee id, customer id), and Y_3 = (day, product id). From these dimension key sets two equivalence classes can be formed: $E_1 = \{Q_1, Q_3\}$ and $E_2 = \{Q_2\}$. Moreover, E_1 can be further divided into two classes $E_1' = \{Q_1\}$ and $E_1'' = \{Q_3\}$ according to

Step 4 in the algorithm, because Q_1 and Q_2 operates in different levels of details in the both their common dimensions.

4.2 Normal Forms

The result of logical design can be evaluated by comparing it against some normal forms [6, 9]. The normal forms will be something like relational normal forms: a particular normal form has some desirable properties related to the correctness of aggregation, sparsity, etc.

OLAP normal forms can be classified into two categories, according to the nature of the properties that they ensure. Multidimensional normal forms are related to dependencies between dimensions and dimensional normal forms concern intra-dimensional dependencies [6]. A dimension is in dimensional normal form if there is one terminal attribute and functional dependencies between attributes on different hierarchy levels, i.e. there is a single attribute key for the dimension. A data cube is in dimensional normal form if all its dimensions are in dimensional normal form. Further, a data cube is in multidimensional normal form if it is in dimensional normal form and there are no FDs between dimensions. The aim of the dimensional normal form is to guarantee completeness in aggregations while the multidimensional normal form is mostly used in controlling sparsity of the OLAP cubes.

A new normal form (which we call here the non-sparse normal form) for controlling structural sparsity of OLAP cubes has been defined by Niemi et al. [9]. The non-sparse normal form takes into account non-unary functional dependencies and dependencies having intersecting right-hand sides. The schema in Figure 3 is in multidimensional normal form but not in non-sparse normal form because of right-hand side intersecting dependencies customer → office and employee → office.

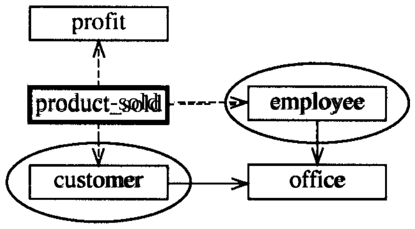

Figure 3. Both customer and employee determines office

4.3 Constructing Normalised Cube

The final OLAP schema is constructed using the method of Niemi et al. [9]. The method applies the functional dependency information in construction of OLAP cubes. The resulting cube or the set of cubes will be in non-sparse normal form that reduces sparsity and incorrect aggregations. The normalisation algorithm works as follows:

I. *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 an input. Multiple cubes are constructed by placing conflicting dimensions in different cubes. There is a conflict between two dimensions if there is a functional dependency having attributes (on the left or right hand side) from both dimensions. 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 but it is better to allow a sparse result and to stop the process earlier. The algorithm can also handle measure attributes. If the measure attributes are given as an input, dimensions will be constructed for them.

The following algorithm constructs an OLAP schema using a set of MDX queries as input.

Algorithm 2 Constructing an OLAP schema from a set of MDX queries.

Input: A base OLAP cube schema H, a set Q of MDX queries against H, and a set of functional dependencies over the attributes in H.

Output: A set of OLAP cube schemata.

```
1. Divide the queries into equivalence classes using Algorithm
   1.
2. For each equivalence class:
   2.1 Set the closures (according to the functional
       dependencies) of the attributes (both dimension and
       measure) of all queries in the equivalence class to the
       initial dimensions of the algorithm.
   2.2 Apply the normalisation algorithm by Niemi et al.
       described above.
3. Output the resulting cube schema(s). It is also possible to
   compute the structural sparsity information and output also it
   to the user.
```

If the user is not satisfied with the result of the algorithm, he or she can manually change the cubes or dimensions.

Example 4 We continue with Example 3. If we do not use the last additional phase of Algorithm 1, we have two equivalence classes of queries: $E_1=\{Q_1, Q_3\}$ and $E_2=\{Q_2\}$ with attributes (day, month, year, product id, product name, product group, profit, quantity] and (employee id, employee name, customer id, customer name, profit]. The both equivalence classes are given to the normalisation algorithm separately. For the first class the algorithm returns an OLAP cube with dimensions (day, month, year), (product id, product name, product group], (profit}, and (quantity], and for the second class (employee id, employee name], (customer id, customer name), and [profit]. The both resulting cubes are in non-sparse normal form having zero as their structural sparsity.

Choosing which hierarchy levels are taken into the cube dimensions is not always so obvious. In the current method the more detailed levels are not included but the more general levels are. The reason is to keep the cube as small as possible, since additional more genera1 levels require storage space only if the precalculated aggregation values are stored. However, quite often the user would also like to do drill down operations from the level that is chosen in the example queries. Thus, in practice, it would be useful to ask the user for every dimension whether the more detailed levels should be included in the final cube.

4.4 Discussion about Properties of Resulting OLAP Schema

We presented three design goals: 1) completeness and minimalism, 2) correctness of aggregations, and 3) minimal sparsity. In this section we discuss how our methods help to achieve those goals.

1. The completeness is guaranteed by taking into account all dimensions given in MDX queries. Of course, the resulting cubes are complete only related to the example query set, which may not totally corresponds the user's information requirements. The resulting OLAP schema is minimal related to the number of dimensions, since only dimensions mentioned in the queries are taken into the OLAP schema.
2. The dimensional normal form helps achieving correct aggregations by guaranteeing complete and disjoint hierarchies because of the functional dependencies between hierarchy levels in a dimension. In addition to this, the user should pay attention to the correct use of statistical aggregation functions.
3. Minimal amount of structural sparsity is achieved by the non-sparse normal form, if the cubes can be fully normalised. Sometimes this could lead to one dimensional cubes, which is not sensible, but the normalisation has to be stopped earlier.

5. CONCLUSIONS AND FUTURE WORK

We have presented a method to construct OLAP cubes based on example queries. The user can start cube construction by posing queries against a general OLAP cube, which represents the information stored into the data warehouse. Another possibility to use the method is to improve the structure of existing cubes based on information about posed queries. The method does not only optimise cubes for efficiency but also ease of use.

It is also possible to make this kind of system capable to learn. For example, the ***time*** is often wanted to be a dimension and it may be that the *cities* are classified according to the *country,* etc. Further, if the user has always kept some concept as a dimension, she/he probably wants it to be a dimension in this time, too. In an organisation, "collaborative learning" can be applied: for example, the cubes constructed by another user can be suggested first, etc.

6. REFERENCES

[1] [Cabibbo, L. and Torlone, R.: A logical approach to multidimensional databases, *6th International Conference on Extending Database Technology (EDBT98, Valencia, Spain),* G. Alonso (eds.), LNCS, Vol. 1377, Springer, 1998.

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

[3] Cheung, D., Zhou, B., Kao, B., Lu, H., Lam, T., and Ting, H.: Requirement-based data cube schema design, *Proceedings of the Eighth International Conference on Information and Knowledge Management,* ACM, 1999.

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

[5] M. Golfarelli, D. Maio, and S. Rizzi: Conceptual design of data warehouses from E/R schemes, *Proceedings of the 31st Hawaii International Conference on System Sciences,* 1998.

[6] 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),* Capri, Italy, 1998.

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

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

[9] Niemi, T., Nummenmaa, J., and Thanisch, P.: Functional dependencies in controlling sparsity of OLAP cubes, *Data Warehousing and Knowledge Discovery, Second International Conference, Da WaK* 2000, Y. Kambayashi et al. (eds.), LNCS, Vol. 1874, Springer, 2000.

[IO] Niemi, T., Nummenmaa, J., and Thanisch, P.: Applying dependency theory to conceptual modelling, *Topics in Conceptual Analysis and Modeling,* Czech Academy of Sciences' Publishing House Filosofia, Prague, 2000.

[I 1] Sapia, C.: On modelling and predicting query behavior in OLAP systems, *Proceedings of the International Workshop on Design and Management of Data Warehouses, S.* Gatziu et al. (eds.), Heidelberg, Germany, 1999.

[12] Sapia, C., Blaschka, M., Höfling, G., and Dinter, B.: Extending the E/R model for the multidimensional paradigm, *Advances in Database Technologies,* Y. Kambayashi et al. (eds.), LNCS 1552, Springer, 1998.

[13] Selfridge, P., Srivastava, D., and Wilson, L.: IDEA: Interactive data exploration and Analysis, *ACM-SIGMOD International Conference on Management of Data,* 1996.

[14] Zurek, T. and Sinnwell, M.: Data warehousing has more colours than just black & white, *VLDB99, Proceedings of 25th International Conference on Very Large Data Bases,* M. Atkinson et al. (eds.), Morgan Kaufmann, 1999.