Evolution of Dimension Data in Temporal Data Warehouses

Johann Eder and Christian Koncilia

University of Klagenfurt
Dep. of Informatics-Systems
{eder,koncilia}@isys.uni-klu.ac.at

Abstract. Multi-dimensional analysis is one of the most important applications of data warehouses, giving the possibility to aggregate and compare data along dimensions relevant in the application domain. Typically time is one of the dimensions we find in data warehouses allowing comparisons of different periods. The instances of dimensions, however, change over time – countries unite and separate, products emerge and vanish, organizational structures evolve. In current data warehouse technology these changes cannot be represented adequately since all dimensions are (implicitly) considered as orthogonal, putting heavy restrictions on the validity of OLAP queries spanning several periods.
We propose an extension of the multi-dimensional data model employed in data warehouses allowing to cope correctly with changes in dimension data: a temporal multi-dimensional data model allows the registration of temporal versions of dimension data. Mappings are provided to transfer data between different temporal versions and enable the system to correctly answer queries spanning multiple periods and thus different versions of dimension data.

Keywords: Data Warehouses, OLAP, Temporal Databases, Structure Versions

1 Introduction and Motivation

Data warehouses or data marts are integrated materialized views over several data sources which can be conventionally structured or semi-structured data and are frequently heterogeneous. The most important usage of data warehouses is On-Line Analytical Processing (OLAP) typically using a multi-dimensional view of the data. OLAP tools then allow to aggregate and compare data along dimensions relevant to the application domain. Typical dimensions found frequently in business data warehouses are time, organizational structure (divisions, departments, etc.), space (cities, regions, countries) and product data.

This multi-dimensional view provides long term data that can be analyzed along the time axis, whereas most OLTP systems only supply snapshots of data at one point of time. Available OLAP systems are therefore prepared to deal with changing values, e. g. , changing profit or turnover. Surprisingly, they are not able to deal with modifications in dimensions, e. g. , if a new branch or division is established, although time is usually explicitly represented as a dimension in data warehouses. Neither did this problem attract much research so far, rare examples are [4, 15].

The reason for this disturbing property of current data warehouse technology is the implicitly underlying assumptions that the dimensions are orthogonal. Orthogonality with respect to the dimension time means the other dimensions ought to be time-invariant. This silent assumptions inhibits the proper treatment of changes in dimension data.

Naturally, it is vital for the correctness of results of OLAP queries that modifications of dimension data is correctly taken into account. E.g. when the economic figures of European countries over the last 20 years are compared on a country level, it is essential to be aware of the re-unification of Germany, the separation of Czechoslovakia, etc. Business structures and even structures in public administration are nowadays subject to highly dynamic changes. Comparisons of data over several periods, computation of trends, computation of benchmark values from data of previous periods have the necessity to correctly and adequately treat changes in dimension data. Otherwise we face meaningless figures and wrong conclusions triggering bad decisions. From our experience we could cite too much such cases.

A brief example: From March 2000 onwards a division A is split into two divisions A_1 and A_2. If we want to analyze all months of the year 2000, we will, for division A, only have the data for January and February, whereas from March onwards we will only have the data for divisions A_1 and A_2. It is therefore taken for granted that the user of the analysis is in possession of an adequate knowledge of the domain and that he/she knows that the divisions A, A_1 and A_2 are related and how they are related. We propose to represent such relationships in the data warehouse and use this information for answering queries. In the example just given we could define that it is possible to represent the turnover of the division A_1 for the periods before March 2000 as a function $turnover(A_1, period) = 30\%\ of\ turnover(A, period)$. We could also show that for all periods from March 2000 onwards the number of employees $M\#$ of the division A corresponds to the function $M\#(A, period) = M\#(A_1, period) + M\#(A_2, period)$. Using such functions, it is possible to assure that a successful analysis can be made even though there might be changes in the structure.

The following extensions to a data warehousing system are therefore necessary:

- **Temporal extension**: dimension data has to be time stamped in order to represent their valid time.
- **Structure versions**: by providing time stamps for dimension data the need arises that our system is able to cope with different versions of structures. We call such a version of structure *structure version*.
- **Transformation functions**: Our system has to support functions to transform data from one structure version into another.

This paper is structured in the following way: Chapter 2 defines a multidimensional system and extends this definition to a temporal multidimensional

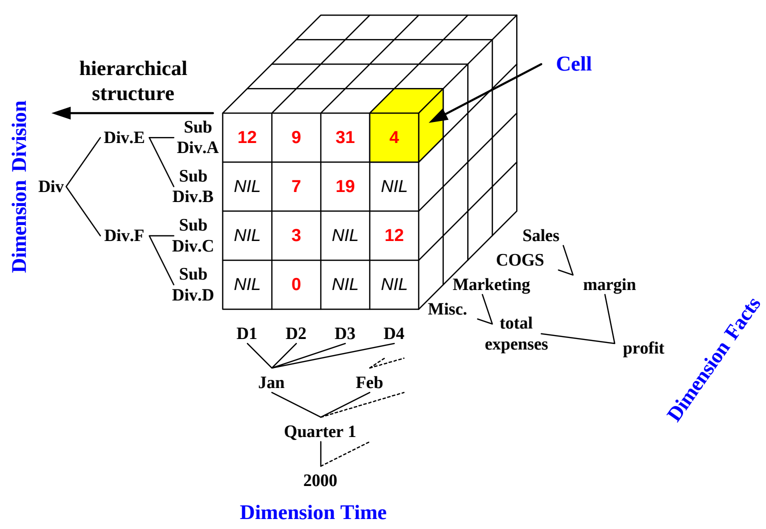

Fig. 1. An example for a multidimensional cube

system. Chapter 3 defines the term structure version. Chapter 4 shows the various modifications of dimension data that may occur, and describes the necessary basic operations to modify dimension data. Chapter 5 details the functions necessary in order to analyse data despite discontinuities of structure and introduces the term of a *transformation matrix*. Chapter 6 shows how to answer queries in such a system. Chapter 7 describes related work. In Chap. 8 we provide a characterization of the further work and a summary.

2 Temporal Multidimensional Systems

A multidimensional view on data consists of a set of dimensions. These dimensions define an n-dimensional data cube [16, 12]. Usually, a data cube is defined by a dimension *Time*, a dimension *Facts* and by several dimensions describing the managerial structures such as divisions, products, or branches.

Figure 1 shows an example for a data cube with three dimensions "Facts", "Time" and "Divisions". A dimension is a set of dimension members and their hierarchical structure. For example *"Profit"*, *"Margin"* and *"Sales"* are dimension members of the dimension "Products" and are in the hierarchical relation $Profit \rightarrow Margin \rightarrow Sales$. The hierarchical structure of all dimensions defines all possible consolidation paths, i. e. , it defines all possible aggregation and disaggregation paths.

A cell in such an n-dimensional data cube contains a value and is referenced by a vector [11]. For example, the tagged cell with the value 4 in the cube shown in Fig. 1 can be referenced by the vector $\nu = (Sub\ Div.\ A, D4, Misc.)$

Until now we have intuitively defined a multidimensional system, or a data warehouse respectively. We will now extend this description to define a temporal data warehouse that supports valid time relations. Therefore, we have to introduce:

- **Chronons**: Usually one dimension describes the factor time and represents the valid time in the system [4]. The finest dimension member within this dimension time defines the chronon Q. The time axis defined through the dimension time is a series of chronons. A chronon is defined as "a non-decomposable time interval of some fixed, minimal duration" [10].
- **Time Intervals**: Dimensions are a set of dimension members (=nodes) and a set of hierarchical links between these dimension members (=edges) such that the resulting graph is a tree representing the hierarchical structure of the dimension members. In order to introduce valid time in such a system all nodes and all edges must have a time interval $[T_s, T_e]$ representing the valid time where T_s is the beginning of the valid time, T_e is the end of the valid time and $T_e \geq T_s$. In P. Chamoni's and S. Stock's paper [4], only edges obtain a time interval. However, an additional time interval for nodes is necessary in order to deal with the modification of attributes of dimension members.

So far we described a temporal multidimensional system intuitively. We will now give a formal description of a temporal multidimensional system.

A temporal multidimensional system consists of:

i.) A number of dimensions $N+1$.

ii.) A set of dimensions $\mathcal{D} = \{D_1,\ ...,\ D_N, F\}$ where F is the dimension describing the required facts and D_i are all other dimensions including a time dimension if required.

iii.) A number of dimension members M.

iv.) A set of dimension members $\mathcal{DM} = \mathcal{DM}_{D_1} \cup ... \cup \mathcal{DM}_{D_N} \cup \mathcal{DM}_F = \{DM_1,\ ...,\ DM_M\}$ where $\mathcal{DM}_F$ is the set of all facts, $\mathcal{DM}_{D_i}$ is the set of all dimension members which belong to dimension D_i. A dimension member DM_i is defined as $DM_i =< DM_{id},\ Key,\ D_i,\ \mathcal{UDA},\ [T_s, T_e] >$. DM_{id} is a unique identifier for each dimension member that cannot be changed (similar to $O_{id's}$ in object-oriented database systems). $[T_s, T_e]$ represents the valid time of the dimension member. D_i is the dimension identifier to which the dimension member belongs. Key is a user defined key (e.g., the number of a product) which is unique within D_i for each timepoint $T_s \leq T \leq T_e$. $\mathcal{UDA}$ is a set of user defined attributes (e.g., the name and/or color of a product).

v.) A set of hierarchical assignments $\mathcal{H} = \{H_1, ..., H_O\}$ where $H_i =< DM^C_{id},\ DM^P_{id},\ Level,\ [T_s, T_e] >$. DM^C_{id} is the identifier of an dimension member, DM^P_{id} is the dimension member identifier of the parent of DM^C_{id} or $\emptyset$ if the dimension member is a top-level dimension member. $Level$ is a value $0...L$ where L is the number of layers and $Level$ is the level of DM^C_{id}. All "leaves" (dimension members without successors) are at level 0. $[T_s, T_e]$ is the time stamp representing the valid time for the relation between DM^C_{id} and DM^P_{id}. No dimension member may be its own parent/child and cycles within $\mathcal{H}$ are not admissible.

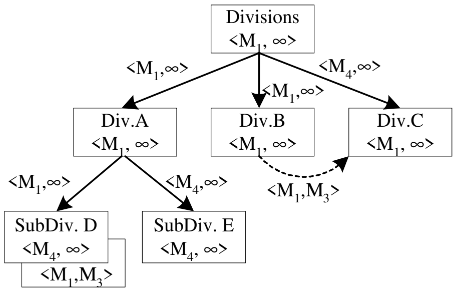

Fig. 2. A Dimension *Divisions* with time stamps

vi.) A function $cval : (DM_{D_1}, ..., DM_{D_N}, DM_F) \rightarrow value$ which uniquely assigns a value to each vector $(DM_{D_1}, ..., DM_{D_N}, DM_F)$ where $(DM_{D_1}, ..., DM_{D_N}, DM_F) \in \mathcal{DM}_{D_1} \times ... \times \mathcal{DM}_{D_N} \times \mathcal{DM}_F$. Therefore, a cube (or n-cube) C is defined by this function $cval$. The domain of this cube $dom(C)$ is the set of all cell references. The range of this cube $ran(C)$ are all cell values.

3 Structure Versions

Temporal projection and temporal selection as defined in the *Consensus Glossary of Temporal Database Concepts* [10] allows us to create what we call a *Structure Version* (SV) out of a temporal data warehouse.

Intuitively, we can say that a structure version is a view on a multidimensional structure that is valid for a given time interval $[T_s, T_e]$. All dimension members and all hierarchical relations within this multidimensional structure are also valid for the given time interval. In other words: within one structure version there cannot exist different versions of a dimension member or a hierarchical relation. Vice versa each modification of a dimension member or a hierarchical relation leads to a new structure version, if a structure version for the given time interval does not already exist.

Formally, each structure version is a 4-tuple $< SV_{id}, T, \{\mathcal{DM}_{D_1,SV_{id}}, ..., \mathcal{DM}_{D_N,SV_{id}}, \mathcal{DM}_{F,SV_{id}}\}, \mathcal{H}_{SV_{id}} >$ where SV_{id} is a unique identifier and T represent the valid time of that structure version as a time interval $[T_s, T_e]$. $\mathcal{DM}_{D_i,SV_{id}}$ $(\mathcal{DM}_{D_i,SV_{id}} \subseteq \mathcal{DM}_{D_i})$ is a set of all dimension members which belong to dimension D_i and which are valid at each timepoint P with $T_s \leq P \leq T_e$. $\mathcal{DM}_{F,SV_{id}}$ $(\mathcal{DM}_{F,SV_{id}} \subseteq \mathcal{DM}_F)$ is the set of all facts which are valid at each timepoint P with $T_s \leq P \leq T_e$. $\mathcal{H}_{SV_{id}}$ $(\mathcal{H}_{SV_{id}} \subseteq \mathcal{H})$ is a set of hierarchical assignments valid at each timepoint P with $T_s \leq P \leq T_e$.

On a conceptual level (of course not on an implementation level) we can say that for each structure version SV there exists a corresponding cube. This cube has the same valid time interval as SV.

Figure 2 shows an example for the consolidation tree of the dimension "Divisions" including time intervals. Each node and each edge in this figure has a time interval $[T_s, T_e]$. In this example an attribute of "$SubDiv.D$" was modified at M_4, a new subdivision "$SubDiv.E$" was introduced at M_4 and $Div.C$ was a subdivision of $Div.B$ from M_1 until M_3 (pictured by the dotted line). Two structure versions can be identified in this example:

i.) $< SV_1, [M_1, M_3], \{\{Divisions, Div.A, Div.B, Div.C, SubDiv.D\}, \{Sales\}\}, \{Div.A \rightarrow Divisions, SubDiv.D \rightarrow Div.A, ...\} >$

ii.) $< SV_2, [M_4, \infty], \{\{Divisions, Div.A, Div.B, Div.C, SubDiv.D, SubDiv.E\}, \{Sales\}\}, \{Div.A \rightarrow Divisions, SubDiv.D \rightarrow Div.A, SubDiv.E \rightarrow Div.A, ...\} >$.

In this example we have two different structure versions SV_1 and SV_2. SV_1 and all given dimension members ($Divisions$, $Div.A$, $Div.B$, ...) and hierarchical assignments ($Div.A \rightarrow Divisions, ...$) are valid from M_1 to M_3. SV_2 and all given dimension members and hierarchical assignments are valid from M_4 to ∞, i. e., until now.

4 Structural Changes

After having formally defined the concept of a temporal data warehouse, we will now present the three basic operations `INSERT`, `UPDATE` and `DELETE` to modify the structure of a temporal data warehouse.

The given chronon Q defines the granularity of data in the data warehouse regarding the dimension time. Hence, we would not gain any additional information from considering structural changes within one chronon. Therefore, we only have to capture structural changes with the granularity defined through the chronon.

Furthermore we define a data warehouse (DWH) as a non-empty, finite set of structure versions DWH $= \{SV_1, ..., SV_n\}$. As described above each structure version SV_i is a 4-tuple $< SV_{id}, T, \mathcal{DM}_{SV_{id}}, \mathcal{H}_{SV_{id}} >$. For this data warehouse we define that it must be a consecutive sequence of tuples $< SV_{id}, T_i, \mathcal{DM}_{SV_{id}}, \mathcal{H}_{SV_{id}} >$ where $T_i = [T_{i,s}, T_{i,e}]$ in such a way that $T_{i,s} = T_{(i-1),e} + Q$.

Dimension data can be modified with the three basic operations `INSERT`, `DELETE` and `UPDATE`. These operations are defined as follows:

Insert: The operation `INSERT` inserts a new dimension member and is defined as `INSERT(`DM`, `T_s`)`. DM is the new dimension member and is a tuple $< Key, D_i, \mathcal{UDA}, DM_{id}^P >$. T_s defines that DM is valid for the time interval $[T_s, \infty]$. A unique DM_{id} is assigned to the new element. Key, D_i, $\mathcal{UDA}$ and DM_{id}^P are defined as described in Sect. 2.

Update: The operation `UPDATE` updates an existing dimension member and is defined as `UPDATE(`Key`,`D_i`, `DM'`, `T_s`)`. Key and D_i define the dimension member to be updated. DM' is a tuple $< Key, \mathcal{UDA}, DM_{id}^P >$. This tuple defines the new value(s).
An `UPDATE` operation translates into two actions: Modify the existing dimension member, setting the ending time to $T_s - Q$, and insert a new dimension member, setting the valid time time interval to $[T_s, \infty]$.

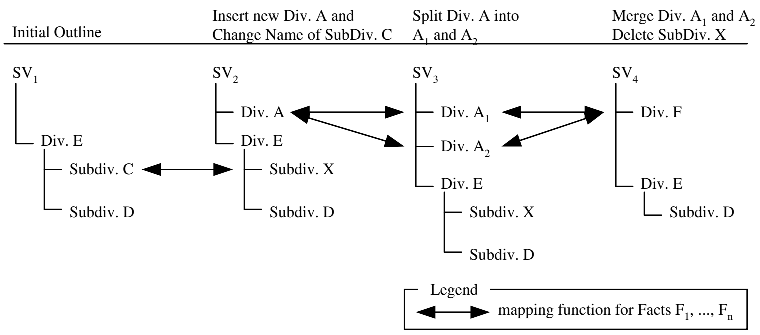

Fig. 3. An example for structural changes and mapping functions

Delete: The operation `DELETE` is defined as `DELETE(`DM`,` T_e`)`. This operation translates into an update operation which sets the ending time of the dimension member DM to T_e. DM is defined as a tuple $< Key, D_i >$.

If no structure version exists that starts with the timestamp of the corresponding `INSERT`, `UPDATE` or `DELETE` operation a new structure version is generated.

Using these basic operations we can modify the structural dimensions of the multidimensional cube. We distinguish the following modifications:

i.) SPLIT: One dimension member splits into n dimension members. E. g., Fig. 3 shows a split operation between the structure versions SV_2 and SV_3 where a division *"Div.A"* splits up into two division *"Div.*A_1*"* and *"Div.*A_2*"*. We would need one delete operation (*"Div.A"*) and two inserts (*"Div.*A_1*"* and *"Div.*A_2*"*) to cope with this.

ii.) MERGE: n dimension members are merged into one dimension member. A merge is the opposite of a split, i. e. a split in one direction of time is always a merge in the opposite direction of time. Consider for the example given above that these modifications occur at the timepoint T. For each analysis that requires data from a timepoint before T for the structure version which is valid at timepoint T we would call these modifications "split". For each analysis that requires data from timepoint T for a structure version valid before timepoint T these modifications would be called "merge".

iii.) CHANGE: An attribute of a dimension member changes, for example if the product number (Key) or the name of a department (user defined attribute) changes. Such a modification can be done by using the update operation.

iv.) MOVE: Modify the hierarchical position of a dimension member, e. g., if a product P no longer belongs to product group G_A but to product group G_B. This can be done by changing the DM_{id}^P (parent ID) of the corresponding dimension member with an update operation.

v.) NEW-MEMBER: Insert a new dimension member, e. g., if a new product becomes part of the product spectrum. This modification can be done by using an insert operation.

vi.) DELETE-MEMBER: Delete a dimension member, e. g., if a branch disbands. Just as a merge and a split are related depending on the direction of time, this is also applicable for the new-member and delete-member relation. In opposite to a new-member operation there is a relation between the deleted dimension member and the following structure version. Consider for the structure given in Fig. 1 that we delete the dimension member *"Subdivision B"* at timepoint T. If for the structure version valid at timepoint T we would request data from a timepoint before T we still could get valid data by simply subtracting the data for *"Subdivision B"* from *"Division E"*.

5 Mappings between Structure Versions

We will now extend the temporal model of a data warehouse presented in chapter 2 with the definition of mapping functions between structure versions.

A structure version is a view on a temporal data warehouse valid for a given time period $[T_s, T_e]$. We distinguish between the structure of a structure version (the set of all valid dimension members of the structure version together with their hierarchies) and the data of a structure version (the cube defined by mapping the structure to the value domain).

For answering queries on the data warehouse the user always has to define which structure version should be used. The data returned by the query can, however, originate in several (different) temporal versions of the cube. Therefore, it is necessary to provide transformation functions mapping data from one structure version to a different structure version.

In the rest of the paper we make the following assumptions: Relations between different structure versions depend on the contemplated fact. For sake of simplicity and understandability we only consider the cube for a single fact. Furthermore, the cell values of upper-level dimension members are always computed from their subordinate lower level dimension members. Therefore, without loss of generality, we do not consider the upper levels here and assume that the dimensions are flat. Or, in other terms: before we transform the data we select the cube of the dimension members at level-0 and transform only this subset of cell values and compute the upper-levels of the resulting cube bottom-up as usual.

5.1 Definition of Inter-Structure Relationships

Mapping functions are employed to map data (cell values) for particular numeric facts and a particular dimension member from one structure version into another using a weighting factor (a "percentage").

Therefore, we provide an operation $MapF$ which is defined as $MapF(SV_j, SV_k, DM_{id}, DM'_{id}, \{M^1_{id}, ..., M^n_{id}\}, w)$ where:

- SV_j and SV_k are different structure versions.
- DM_{id} and DM'_{id} are unique IDs for dimension members for which $DM_{id} \in SV_j$ and $DM'_{id} \in SV_k$ is true. DM_{id} and DM'_{id} must be dimension members of the same dimension.

- $\{M_{id}^1, ..., M_{id}^n\}$ is a non-empty, finite set of fact IDs and $\exists f : f \in F \wedge f_{id} = M_{id}^i$.
- w is the weighting factor to map data from one structure version into another.

We implicitly introduce a mapping function for each dimension member which does not change from one structure version into another with a weighting factor $w = 1$.

Mapping functions may be applied to map data between contiguous (e. g., $SV_1 \leftrightarrow SV_2$) or non contiguous (e. g., $SV_1 \leftrightarrow SV_3$) structure versions. Two structure versions SV_i and SV_k are contiguous if $T_{s,i} = T_{e,k} + Q$ or if $T_{s,k} = T_{e,i} + Q$.

For a split or a merge operation we need several mapping functions. e. g., if department A splits up into A_1, A_2 and A_3 we would need three functions to map data from A to A_n and three functions to map data from A_n to A. We do not restrict the user regarding the weighting factor w. This means that the sum of all weighting factors for all functions $A \to A_n$ (split) does not have to be 1, i. e., 100%. Vice versa not all weighting factors of the functions $A_1 \to A, ..., A_n \to A$ (merge) need to be 1. This allows to represent the effects of structural changes like: Product *"Personal Computer"* is not longer sold as is but separated as *"Monitor"* and *"Desktop"*. The combined price for both products is higher than the price for the "Personal Computer" was.

The example given in Fig. 3 shows several structural changes in a dimension *"Divisions"*, e. g., *"Subdiv.C"* was renamed to *"Subdiv.X"* from SV_1 to SV_2, *"Div.A"* split up into *"Div.A_1"* and *"Div.A_2"* from SV_2 to SV_3 and so on. This example would result in the following mapping functions for the fact "Turnover":

1.) $MapF(SV_1, SV_2, Div.C, Div.X, \{\text{Turnover}\}, 1)$
2.) $MapF(SV_2, SV_1, Div.X, Div.C, \{\text{Turnover}\}, 1)$
3.) $MapF(SV_2, SV_3, Div.A, Div.A_1, \{\text{Turnover}\}, 0.3)$
4.) $MapF(SV_2, SV_3, Div.A, Div.A_2, \{\text{Turnover}\}, 0.7)$
5.) $MapF(SV_3, SV_2, Div.A_1, Div.A, \{\text{Turnover}\}, 1)$
6.) $MapF(SV_3, SV_2, Div.A_2, Div.A, \{\text{Turnover}\}, 1)$
7.) and so on...

This set of functions for instance defines that for the fact "Turnover" the division A_1 in SV_3 corresponds to 30% of the division A in SV_2 (see function 3.). Or vice versa that the division A in SV_2 is equal to the sum of A_1 and A_2 in SV_3 (see functions 5. and 6.).

5.2 Transformation Matrices

On a conceptual level we can represent each multidimensional cube and the relationships between dimension members of different structure versions as matrices.

Let SV_i be a structure version with N dimensions. Each dimension D_N consists of a set $\mathcal{DM}_N^{L0}$ which represents all Level-0 dimension members of that dimension. We can represent this structure version as a $\mathcal{DM}_1^{L0} \times \mathcal{DM}_2^{L0} \times \ldots \times$

$\mathcal{DM}_N^{L0}$ matrix. For example a structure version for a 1-dimensional cube results in a vector whereas a structure version for a 3-dimensional cube results in a matrix which elements are vectors.

Let SV_1 and SV_2 be two structure versions. We define a transformation matrix $T_{SV_1,SV_2,D_i,F}$ for each dimension D_i and each fact F. Where $T(d_i, d_j)$ is a number representing the weighting factor for mapping a fact F of dimension member d_i of structure version SV_1 to a fact of dimension member d_j of structure version SV_2.

These transformation matrices are merely another way of representing the information contained in the $MapF$ Relation described above. We want to emphasize that the construction of these matrices is a conceptual view on the transformation. Any meaningful implementation will take into account that these matrices are usually extremely sparse and will not implement the matrices in a naive way.

Example: Consider a cube C representing the structure defined through a structure version SV_1 with the dimensions $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$ (a_i and b_j are dimension members). We represent the cell values for a specific fact in this cube as a matrix. Therefore, a value in this matrix represents a cell value in the given 2-dimensional cube.

$$C = \begin{array}{c} \\ b_1 \\ b_2 \\ b_3 \end{array} \begin{array}{c} \begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \\ \begin{pmatrix} 3 & 7 & 5 \\ 10 & 8 & 6 \\ 20 & 13 & 5 \end{pmatrix} \end{array}$$

As mentioned above we need one transformation matrix for each dimension D_i to map data from structure version SV_1 into structure version SV_2. In the following example we split the dimension member a_1 into a_{11} and a_{12} and we merge b_1 and b_2 into b_{12}. The functions between SV_1 and SV_2 for a fact "$Fact$" are defined by the following operations:

1.) $MapF(SV_1, SV_2, a_1, a_{11}, Fact, 0.3)$
2.) $MapF(SV_1, SV_2, a_1, a_{12}, Fact, 0.7)$
3.) $MapF(SV_1, SV_2, b_1, b_{12}, Fact, 1)$
4.) $MapF(SV_1, SV_2, b_2, b_{12}, Fact, 1)$

To represent these functions we define two transformation matrices. T_A for dimension A, and T_B for dimension B:

$$T_A = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{cccc} a_{11} & a_{12} & a_2 & a_3 \end{array} \\ \begin{pmatrix} 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

$$T_B = \begin{array}{c} \\ b_{12} \\ b_3 \end{array} \begin{array}{c} \begin{array}{ccc} b_1 & b_2 & b_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

5.3 Transformation of Warehouse Data

The goal of the transformation is to map the warehouse data (cube) of a certain structure version SV_1 to the structure of a different structure version SV_2.

We first define a function to transform the cube in one dimension:

$f_{SV_1,SV_2,D,F}$ transforms the values of fact F of structure version SV_1 to the structure version SV_2 in the dimension D as follows:

$$f_{SV_1,SV_2,D,F}(C_{D=j}) = C'_{D=i} \text{ with} \qquad (1)$$
$$C'_{D=i} = \sum_{j \in DM_{D,SV_1}} T_{SV_1,SV_2,D,F}(i,j) * C_{D=j} \text{ for all } i \in DM_{D,SV_2}$$

where C is a cube with the dimension members of SV_1 in dimension D and C' is the transformed cube where all values in the cube have been transformed to the members of the dimension D in the structure version SV_2 according to the transformation matrix T. $C_{D=j}$ is the (n-1) dimensional sub-cube of an n-dimensional cube associated with the member j in dimension D.

It is easy to see, that transforming a cube in dimension D_x first, and then in dimension D_y yields the same result as the transformation in the reverse sequence:

$$\begin{aligned} f_{D_y}(f_{D_x}(C)) &= C' \text{ with} \qquad (2)\\ &= \forall i,k : C'_{D_x=i,D_y=k} \\ &= \sum_j T_{D_x}(i,j) * \sum_l T_{D_y}(k,l) * C_{D_y=l,D_x=j} \\ &= \sum_l T_{D_y}(k,l) * \sum_j T_{D_x}(i,j) * C_{D_y=l,D_x=j} \\ &= f_{D_x}(f_{D_y}(C)) \end{aligned}$$

The transformation of a fact F in a cube C from structure version SV_1 to structure version SV_2 is now defined as a sequence of functions successively transforming the cube in all dimensions D_i:

$$f_{SV_1,SV_2,F} = f_{SV_1,SV_2,D_1,F}(f_{SV_1,SV_2,D_2,F}(\ldots f_{SV_1,SV_2,D_n,F}(C_{SV_1})\ldots)) \qquad (3)$$

As seen from the observation above the result does not depend on the sequence of transformation used. Again, we emphasize that this is the specification of a transformation function, and the actual implementation will efficiently make use of the sparseness of the involved matrices, etc.

Example: By using the defined transformation functions we are now able to transform data from SV_1 into SV_2. The cube C and the transformation matrices T_A and T_B are given in the example in Sect. 5.2.

$$C' = f_{SV_1,SV_2,D_A,F}(f_{SV_1,SV_2,D_B,F}(C))$$

$$= \begin{array}{c} \\ b_{12} \\ b_3 \end{array} \begin{array}{c} \begin{array}{cccc} a_{11} & a_{12} & a_2 & a_3 \end{array} \\ \begin{pmatrix} 3.9 & 9.1 & 15 & 11 \\ 6 & 14 & 13 & 5 \end{pmatrix} \end{array}$$

The matrix C' represents the cube with the structure defined through structure version SV_2 and the values of structure version SV_1.

6 Queries

When a user issues a query within such a system, he/she has to define a timepoint T_q. This timepoint specifies a certain base structure version where $T_s \leq T_q \leq T_e$ and $[T_s, T_e]$ defines the valid time interval of the base structure version.

This base structure version determines which structure has to be used for the analysis. In most cases this will be the current structure version. However, in some cases it will be of interest to use an "older" structure version. Suppose the structure versions given in Fig. 3 are valid for the following time periods and the chronon is a month:

Table 1. Valid time periods

Version	T_s	T_e
SV_1	Jan. 1998	Mar. 1998
SV_2	Apr. 1998	Jan. 1999
SV_3	Feb. 1999	Dec. 1999
SV_4	Jan. 2000	∞

We might assume the user chooses SV_4 as his or her base structure version and requests data for March 2000 and March 1999 for his analysis. In this case the system needs functions to map data which is valid for the structure version SV_3 into the structure version SV_4.

The same analysis however could also be made with SV_3 as base structure version. For this query the system needs functions to map data from SV_4 to SV_3.

We distinguish the following cases when answering queries:

(1) One structure version: A query which has to access only one structure version can be easily answered. There is no need to consider functions.

(2) Two contiguous structure versions: In this case the system has to examine for all required dimension members and all required facts whether or not there exists a mapping function $MapF(SV_1, SV_2, DM_{id1}, DM_{id2}, \{F^1_{id}, ...,$

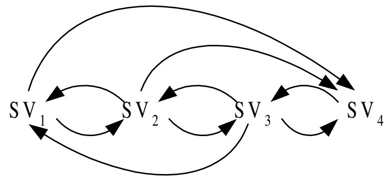

Fig. 4. Mapping functions between several structure versions (arrows are defined mapping functions)

$F_{id}^n\}$, $w)$. If no mapping function of this type exists, the result of the query is 'not defined' for the correlative cells. If such a function exists, the query can be answered.

(3) Two non-contiguous structure versions: The system has to examine for all required dimension members and all required facts whether or not there exists a mapping function $MapF(SV_1, SV_2, D_{id1}, D_{id2}, \{F_{id}^1, ..., F_{id}^n\}, w)$. If no mapping function exists, the transitive closure of all mapping functions has to be examined. In the case that there does not even exist a function in the transitive closure, the result of the query is 'not defined' for the correlative cells. If a function exists, the query can be answered.

(4) More then two structure versions: As always two schemes are considered, this case can be solved in analogy to case (3), i. e., case (2). If for the example given in Fig. 4, data from SV_1 has to be mapped to the structure version SV_3, this can be done by considering the functions $SV_1 \rightarrow SV_2$ and $SV_2 \rightarrow SV_3$.

For each query stated, the systems checks which structure versions are necessary to answer the query. E. g., if the user defines SV_4 as his base structure version and the valid time intervals are defined as shown in Tab. 1, the structure versions SV_4, SV_2 and SV_1 are necessary to answer the query "return costs for all divisions for January 1999 and January 1998". For each fact the system examines whether there exist a function to map data from SV_1 to SV_4 and from SV_2 to SV_4.

As shown in Fig. 4 functions can map data between contiguous or non contiguous structure versions. To enable the user to track changes within schemes step by step, there have to be at least functions between all successive structure versions. E.g. for the structure versions defined in Tab. 1, the following functions have to be defined: $SV_1 \leftrightarrow SV_2$, $SV_2 \leftrightarrow SV_3$ and $SV_3 \leftrightarrow SV_4$. Additionally, functions for non contiguous structure versions can be defined or derived to optimize performance.

7 Related Work

In contrast to temporal databases, which have been well studied, e. g., [3, 7, 8], few approaches are known in literature for temporal data warehouses, e. g., [4, 15]. The same applies for schema evolution of databases, e. g. , [13, 5] vs. schema evolution of data warehouses, e. g. , [2].

[4] present which extensions are necessary in order to extend a data warehouse to cover temporal aspects, in particular they keep track of the history of hierarchical assignements. In contrast to this approach we deal also with modifications of dimension members and provide relations between dimension members which are necessary to relieve the user from his/her duty to know the relation between two dimension members.

[2, 1] deal with schema evolution and schema versioning for data warehouse systems. They show how changes of the conceptual schema can be automatically transfered into the logical and internal schema. However, these papers do not deal with the inevitable consequences arising out of these scheme evolutions like misinterpretations of analyzes.

C. Hurtado, A. Mendelzon and A. Vaisman [9] have proposed a formal description of *"complex operators"* to delete, insert and/or update dimensions at a structural and at an instance level. However, they do not cope with the problem regarding the availability of data and they do not deal with the fact that several versions of dimensions may be valid for different time intervals, i. e. a temporal data warehouse or a temporal multidimensional system respectively.

[14] have proposed a formal definition of a temporal OLAP system and a temporal query language, called TOLAP. However, they do not support transformation of data between structural versions and their system is not able to cope with changes in the time and fact dimensions.

Another approach to deal with changes within dimension data was proposed in a white paper by *SAP America* [15]. They extend their schema with time stamps to enable the user to analyze data for different scenarios. However, this approach is limited to some basic operations on dimension data (e. g. , insert/delete a dimension member; change the "parent" of dimension member) and does not deal with weighting factors.

8 Conclusion

We presented a novel approach for representing changes in dimension data of multi-dimensional data warehouses, by introducing temporal extension, structure-versioning and transformation functions. This representation can then be used to pose queries (analysis) against the structure valid at a given point in time and correctly admit data from other periods into the computation of the result.

This effort is necessary as changes in these data have the combined characteristics of temporal databases and schema evolution, as these dimension data serve in multi-dimensional systems as data as well as schema elements. Our approach thus overcomes the implicit orthogonality assumption underlying multi-dimensional data warehouses.

The transformation function we propose here can only be seen as a first step and will be elaborated in the future. The simple transformation matrices however proved themselves surprisingly powerful. We were able to represent several cases of structural changes with these data (at least approximatively). Changes which can be covered by our model comprise:

- Changes in the organizational structure of enterprises, and of the regional structure of distribution systems.
- Changes of Units, like actually the changes from ATS to EURO.
- Changes in product portfolios.
- Changes in the way economic figures like unemployment rate, consumer price index, etc. are computed.

We also expect that the model we prosed here improves the correctness of interpretation of answers to OLAP queries and relieves the user from the need to have detailed knowledge about the change history of dimension data. In particular, our approach provides for multi-period comparisons of facts which currently requires stability in dimension data.

References

[1] M. Blaschka. FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems. In *Proc. of 6th Doctoral Consortium*, Germany, 1999.

[2] M. Blaschka, C. Sapia, and G. Höfling. On Schema Evolution in Multidimensional Databases. In *Proc. of the DaWak99 Conference*, Florence, Italy, 1999.

[3] M. Böhlen. Temporal Database System Implementations. *SIGMOD*, 24(4), 1995.

[4] P. Chamoni and S. Stock. Temporal Structures in Data Warehousing. In *Data Warehousing and Knowledge Discovery (DaWaK) 1999*, pages 353–358, Italy, 1999.

[5] S. M. Clamen. Schema Evolution and Integration. In *Distributed and Parallel Databases: An International Journal*, pages 2(1):101–126.

[6] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practise*. Number LNCS 1399. Springer-Verlag, 1998.

[7] Goralwalla, Tansel, and Zsu. Experimenting with Temporal Relational Databases. *ACM*, CIKM95, 1995.

[8] Gregersen and Jensen. Temporal Entity-Relationship Models - a Survey. *Time-Center*, 1997.

[9] C. Hurtado, A. Mendelzon, and A. Vaisman. Updating OLAP Dimensions. In *Proc. of the ACM Second Int. Workshop on Data warehousing and OLAP*, USA, 1999.

[10] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*, pages 367–405. Springer-Verlag, 1998. in [EJS98].

[11] A. Kurz. *Data Warehousing - Enabling Technology.* MITP-Verlag, Bonn, 1 edition, 1999.

[12] C. Li and X. Wang. A Data Model for Supporting On-Line Analytical Processing. *ACM*, CIKM 96, 1996.

[13] C. Liu, S. Chang, and P. Chrysanthis. Database Schema Evolution using EVER Diagrams. In *Proceedings of the Workshop on Advanced Visual Interfaces*, pages 123–132, 1994.

[14] A. Mendelzon and A. Vaisman. Temporal Queries in OLAP. In *Proc. of the 26th VLDB Conference, Egypt*, 2000.

[15] SAP America, Inc. and SAP AG. Data Modelling with BW - ASAP for BW Accelerator. 1998. White Paper: URL: http://www.sap.com.

[16] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. In *SIGMOD Record 28*, 1999.