Proceedings of the VLDWH Workshop 2002 (Dexa 2002), Aix-en-Provence, IEEE No PRO1668, ISBN 0-7695-1668-8, pp 817-821

Representing Temporal Data in Non-Temporal OLAP Systems

Johann Eder
University of Klagenfurt
Dep. of Informatics-Systems
eder@isys.uni-klu.ac.at

Christian Koncilia
University of Klagenfurt
Dep. of Informatics-Systems
koncilia@isys.uni-klu.ac.at

Abstract

Multidimensional data warehouses and OLAP systems do not provide adequate means for dealing with changes in dimension data, changes appearing frequently in dynamic application areas as current business systems. As data warehouses and OLAP tools serve as decision support systems they have to reflect such changes. Temporal data warehouses propose sophisticated modelling tools for covering any changes in dimension data but cannot be used with current OLAP systems. In this paper we show some techniques to incorporate temporal dimension data into multidimensional OLAP systems. In particular we show how to superimpose conventional multidimensional data warehouses with temporal master data, enabling queries to span multiple periods and to return correct answers.

1 Introduction

A data warehouse (DWH) is an integrated repository for data stemming from several heterogenous database systems [3] This source data may be structured, semi-structured or unstructured. They are mainly used for 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 frequently found in business data warehouses are time, organizational structure (divisions, departments, etc.), space (cities, regions, countries) and product data.

Data warehouses are well prepared to deal with modifications in transaction data, e.g. the changing values of the fact *Turnover* over time can be covered by introducing a dimension *Time*. Surprisingly, data warehouses are not well prepared for changes of the structure of dimensions in spite of their requirement for serving as long term memory. Furthermore, they cannot adequately represent changes in units, e.g., from ATS to EURO, changes in formulas computing derived data, or schema changes.

For an example: In a data warehouse that stores information about diseases for all European countries along the dimensions *Time*, *Geography*, *Diagnosis* and *Facts* At least two dimensions changed over time:

Geography: Reunification of Germany in 1990 or the independence of Slovenia in 1991 have to be represented.

Diagnoses: Diagnoses for patients are represented with the "International Statistical Classification of Diseases and Related Health Problems" (ICD) code. However, codes for diagnoses changed from ICD Version 9 to ICD Version 10. E.g. the code for "malignant neoplasm of stomach" has changed from 151 in ICD-9 to $C16$ in ICD-10; diagnoses were regrouped, e.g. "transient cerebral ischaemic attacks" has moved from "Diseases of the circulatory system" to "Diseases of the nervous system" [6]; etc.

How can we get correct results for queries like "did diabetes increase over the last 5 years" or "number of patients with cancer in Germany over the last 25 years"? Ignorance of the above changes will lead to incorrect results.

In [4] we present an architecture for a *Temporal Data Warehouse* that enables users to get correct results for queries spanning multiple periods by introducing time stamps to represent the valid time of objects.

Contribution. In this paper, we discuss approaches to map temporal data warehouse structures to current non-temporal multidimensional OLAP systems superimposing dimensions and aggregation hierarchies such that current non-temporal OLAP technology can also be used for processing of changing dimension data.

Related Work. We first presented the problem and a concept for solution for simple data warehouses by transformation functions in [3]. Other approaches for temporal data warehouses are [9, 1, 8, 2]. To our best knowledge, only [8] deals with both schema and instance modifications. However, the approach proposed in [8] supports only schema/instance evolution and no versioning. Furthermore, in contrast to our temporal data warehouse approach [3, 4] none of the papers mentioned supports a mechanism to introduce relationships between instances in different structural versions, i.e., transformation functions.

2 Temporal Data in a Temporal Warehouse

Our concept called *COMET* proposed in [3] and [4] extends the well known data warehouse approach with aspects of temporal databases and schema versioning. The changes we have to cope with are not only schema changes, but also changes in the dimension data (also called master data). The dimension *Time* ensures to keep track of the history of transaction data, i.e., measures. Nevertheless, for correct query results after modifications of dimension data we have to track modifications of these data [3], and therefore, we extend data warehouses with [3]:

- **Temporal extension**: dimension data has to be time stamped in order to represent their *valid time*. The *valid time* represents the time when a "fact is true in the modeled reality" [7].
- **Structure versions**: by providing time stamps for dimension data our system is able to cope with different versions of structure.
- **Transformation functions**: Our system supports functions to transform data from one structure or schema version into another.

All dimension members, i.e., instances of dimensions, and all hierarchical links between these dimension members have to be time stamped with 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 $T_e \geq T_s$. According to [7] a time interval "is the time between two instants" and it "may be represented by a set of contiguous granules". Furthermore, we timestamp all schema definitions, i.e. dimensions, categories and their hierarchical relations, in order to keep track of all modifications of the data warehouse schema [4].

If we represent all time stamps of all modifications within our data warehouse on a linear time axis the interval between two succeeding time stamps on this axis represents a structure version. Therefore, within a structure version the structure of dimension data on both the schema level and on the instance level is stable.

The data returned by a query may originate in several (different) structure versions. Transformation functions make correct analysis over changes in the dimension data and dimension structure possible.

3 Temporal Data in OLAP Systems

The approach discussed in section 2 stores temporal data in a temporal data warehouse model. However, frequently the need arises that temporal data should be represented in a non-temporal data warehouse, i.e. a non-temporal OLAP system.

In this section we will discuss how to deal with temporal data in a non-temporal OLAP system, like Oracle Express or Cognos PowerPlay. In particular, we will discuss how to store temporal data in Hyperion Essbase Version 6.0 (`http://www.hyperion.com`), a popular and widely-used multidimensional database management system (without focusing on it's specific nuts and bolts).

3.1 Modifications of Master Data

In [3] we identified (beside the basic operations `INSERT` and `DELETE`) the following types of modifications:

Move: The hierarchical position, i.e. the parent of a dimension member changes. For instance, in our running example "transient cerebral ischaemic attacks" (ICD-10 code $G45$) has moved from "Diseases of the circulatory system" (ICD-9 code $390-459$) to "Diseases of the nervous system" (ICD-10 code $G00 - G99$).

Change: The key of a dimension member changes. For instance the code for "malignant neoplasm of stomach" has changed from 151 in ICD-9 to $C16$ in ICD-10.

Split: A dimension member splits up into several other dimension members. For instance, the ICD-9 code for "Chemotherapy" $V58.1$ split up into $Z51.1$ "Chemotherapy session for neoplasm" and $Z51.2$ "Other Chemotherapy". Another example is the code 175 that split up into ten different ICD-10 codes, namely $C50.0$ to $C50.9$.

Merge: Several dimension members merge into one dimension member. In our running example, the ICD-9 codes 2350 and 2351 merged into the ICD-10 code $D37.0$.

3.2 Representing Versions in Unique Dimensions

The technically simplest and most naive way to represent temporal master-data in non-temporal OLAP systems would be to represent each structure version for each dimension in a unique dimension. However, the volume consumed by a data cube grows proportional to the number of elements, i.e. dimension members stored in the cube. Hence, this approach would lead to enormous cubes. Furthermore, navigating through the data stored in the cube would not be intuitive and easy for the end-user. Moreover, it is only possible to compare the different versions of a dimension member by manually comparing the various versions.

3.3 Extending the Scenario Dimension

Scenario dimensions are frequently found in multidimensional databases. Typically, they represent different versions of transaction data, i.e. of measures by introducing dimension members like "Actual Value", "Planned Value" and "Budget".

However, the scenario dimension could also serve as a dimension to distinguish between different versions of master data. This could be accomplished by introducing a dimension member "Structure Version $[T_S, T_E]$" for each structure version that should be represented in the data cube. In contrast to the first approach mentioned this approach requires only small additional resources.

For our running example, we could represent the fact that FRG and GDR reunited in 1990 by creating two dimension members "Structure Version $[Start, 1989]$" and "Structure Version $[1990, Now]$" in the scenario dimension. Furthermore, we have to create a new dimension member in the dimension *Geography*, namely "Germany".

The main drawback of this approach is that structural changes of dimensions members are not obvious for the user. For instance, all three dimension members FRG, GDR and $Germany$ would reside in the dimension *Geography* without any obviously hints for the user that these dimension members are in any relation.

Nevertheless, the user is able to detect the implicitly stored valid time by combining the corresponding dimension member and the dimension members representing the structure versions. If the cell selected by this combination comprises a value not equal to `NULL` then the dimension member is valid in the selected structure version.

3.4 Attribute Dimension

The following approach uses a new feature supported by Hyperion Essbase, namely *Attribute Dimensions*. An attribute dimension is a dimensions that contains members that describe characteristics of members of another dimension.

Usually, it describes characteristics such as weight, color or size. All members of an attribute dimension do have the same type. This type may be `Boolean`, `Text`, `Numeric` or `Date`. Each dimension may have several attribute dimensions and each attribute dimension has to be assigned to exactly one dimension.

Attribute dimensions may be used to represent the valid time of members of other dimensions. The administrator may introduce an attribute dimension of type `Text` whose members look like "$[1997 - 1999]$", representing the begin and the end of the valid time as a string. He could also introduce two attribute dimensions of type `Date`, where one attribute dimension represents the begin of the valid time and the other represents the end of the valid time.

Figure 1 shows how to represent valid time using attribute dimensions. In this example, the attribute "$[1997 - Now]$" is assigned to the dimension member "Camera" representing that it is valid from 1997 until now. "VCR" is valid from 1997 until 2001 and "Television" is valid from 2002 until now.

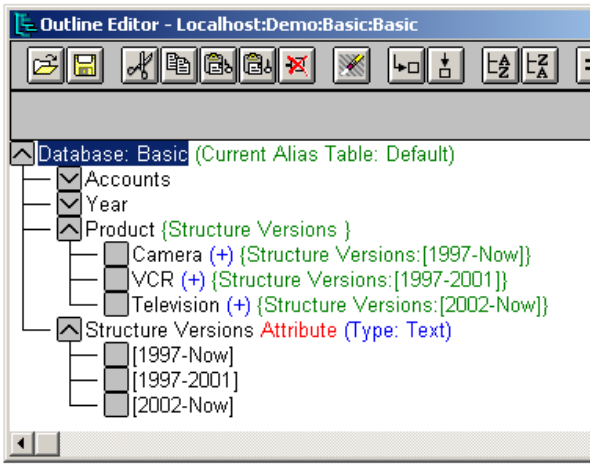

Figure 1. Using Attribute Dimensions

The user may now easily analyze the data stored in the multidimensional database using this attribute dimensions. He may select all members regardless of their valid time, or he may select only those members which are assigned to a specific attribute dimension, i.e. members that are valid in a specific structure version.

The main drawback of this approach is, that only dimensions marked as "sparse" can have attributes, i.e. assigned attribute dimensions. Hence, if all dimensions change over time, all dimensions have to be sparse. This increases the resources needed to store the database and decreases the performance.

Furthermore, each attribute dimension may only be assigned to one dimension. Hence, for each dimension whose dimension members change over time a single attribute dimension has to be created. This leads of course to additional expenses regarding the administration of the cube.

Another drawback is, that modifications of the key of a dimension member cannot be treated correctly, i.e. their is no obvious relation for the user between the two versions of such a dimension member. Hence, it is only possible to compare the different versions of a dimension member whose key changed by manually comparing the various versions.

3.5 Prefix/Suffix Notation

The main idea of this approach is to extend the keys of dimension members with a prefix or suffix representing the valid time of the dimension member. Usually, the keys used in Hyperion Essbase for dimension members are their names. For sake of readability, only dimension members that were subject of modifications get time stamped.

According to [7] data models for temporal database man-

agement systems "may represent a time line by a sequence of non-decomposable, consecutive time intervals of identical duration." These intervals are named chronons. A dimension *Time* is usually a part of multidimensional database applications. As this dimension defines the finest level on which modifications of transaction data are being stored, the chronon for the time stamp should be equal to the chronon of the dimension Time.

The chronon and format of the time stamps should be uniform within the whole cube, i.e., within the database. The format of the time stamp should comprise both, the start and the end of the valid time.

Within Hyperion Essbase, the suffix/prefix may be stored 1) within the dimension member name, 2) within the alias of the dimension member, 3) as a unique User Defined Attribute (UDA) or 4) as a combination of the possibilities mentioned. In Hyperion Essbase, the name of a dimension member has to be unique throughout the whole cube. Hence, the suffix/prefix should be stored at least within the dimension member name. This leads to unique names. Storing the suffix/prefix within the alias or within an UDA increases the opportunities for analyzing data.

To enable easy multi-period comparisons even after modifications of dimension members, we propose an implementation for this approach where modifications of dimension members are stored in a subordinate hierarchical level. In other words, we insert a new, artificial hierarchical level above the level of the dimension member which was subject of changes.

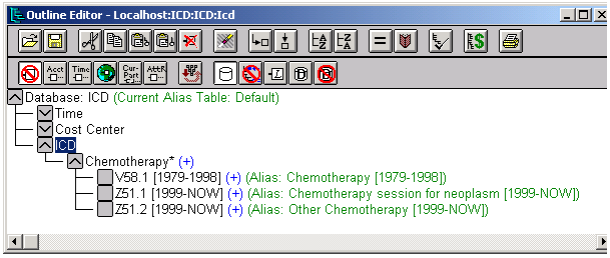

Figure 2. Incorporating a Subordinate Level

Figure 2 shows an example of such a subordinate level. In this example, we introduced a new hierarchy level by inserting a new dimension member named "Chemotherapy *". We added this asterisk to inform the user that this hierarchy is an artificial hierarchy. The children of this member are both, the old and the new version of the corresponding dimension member.

The main advantage of this approach is that it allows multi-period comparisons on the level of the artificial hierarchy for all types of modifications mentioned, except *MOVE*.

For a modification of the type *MOVE*, no artificial hierarchy level is necessary. Figure 3 shows how to represent

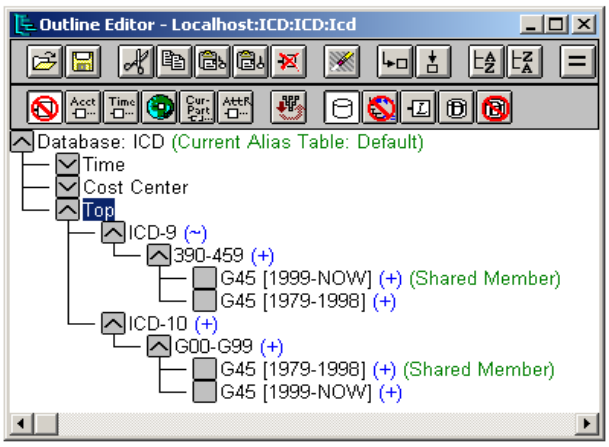

Figure 3. A *Move* in Hyperion Essbase

the example mentioned ($G45$ has moved from $390-459$ in ICD-9 to $G00-G99$ in ICD-10). In order to avoid to double the values on the top level, correct consolidation functions have to be given. Hence, we have to exclude the ICD-9 path from consolidation. In Hyperion Essbase, this can be done by using the consolidation function ~ (tilde), where ~ means "exclude member from consolidation". In order to allow comparisons on the upper levels, the corresponding member has to inserted as a *shared member*. A shared member is a member with multiple parents. In this example, the member $G45$ is a child of both $390-459$ and $G00-G99$.

3.6 Folding Modifications into the Consolidation Hierarchy

We will now discuss the last and most powerful approach to represent temporal data in non-temporal, multidimensional database management systems. The main idea of this approach is that we "fold" modifications of dimension members into the consolidation hierarchy. This can be done, by creating a new hierarchical level within each dimension that was subject of modifications for each structure version.

The main advantage of this approach is that it enables the user to create multi-period comparisons for all types of modifications. Furthermore, it requires only a small amount of additional resources. The main drawback is, that the resulting structure is quite complex, and not easy to understand for the user. Therefore, without training, the user would be swamped with this approach.

CHANGE: A changing key can easily be represented by creating a dimension member with the new key whose child is the dimension member with the old key. Figure 4c) shows how to represent that the key for dimension member $C1$ changed from $C1$ to $C2$.

SPLIT: We are able to represent a dimension member that splits into several new dimension members by creating

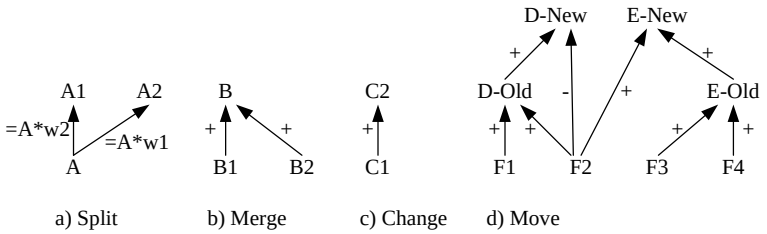

Figure 4. Folding Different Types of Modifications into the Consolidation Hierarchy

the new members as parents of the old dimension member. In order to allow multi-period comparisons we need to split the data stored for the old element. This can either be done by applying a weighting factor and dividing the stored values according to the given weighting factor, or (if it is not possible to find a weighting factor) by aggregating all the old data into one of the new dimension members.

Figure 4a) shows how to represent that A split into $A1$ and $A2$. Please note that the consolidation function is not a simple aggregation or subtraction, but a formula that allows to divide the data stored for A correctly into $A1$ and $A2$ (within Essbase, this could be done by using so called *Calc-Scripts*). In this example $w1$ and $w2$ are the given weighting factors.

MERGE: Merging several dimension members into one dimension member can be represented by creating the new dimension member whose children are the old dimension members. The new data will be loaded into the cells referenced by the dimension member B. The old data loaded for $B1$ and $B2$ will be aggregated into B. Hence, multi-period comparisons are possible. Figure 4b) shows how to represent that the dimension members $B1$ and $B2$ merged into B.

MOVE: Some complex modifications are necessary in order to represent that a dimension member has been moved from one parent to another. Figure 4d) gives an example of how to represent that $F2$ moved from D to E. In this example, D had the children $F1$ and $F2$, and E had the children $F3$ and $F4$. At a given timepoint $F2$ moved from D to E. The outcome of this is that D now only has one child $F1$ and E now has three children, $F2$, $F3$ and $F4$.

We represent this behavior by introducing an old and a new version for both parent dimension members, namely D and E. This versions are tagged $D-New$, $D-Old$, $E-New$ and $E-Old$. The member $F2$ is a child of $D-Old$ and $E-New$. In order to avoid that the values stored in the cells referenced by $F2$ are aggregated into $D-New$ (via $D-Old$), we have to subtract those values from $D-New$.

INSERT & DELETE: After a new dimension member was inserted or an old member was deleted multi-period comparisons for these elements are not possible, simply because no data exists before the insert or after the delete. Hence, such newly created or deleted dimension members can be simply inserted into the corresponding dimension.

4 Conclusions

In this paper we discussed how to represent changes of master data in data warehouses and in OLAP systems. We briefly discussed how to represent such changes in our temporal data warehouse approach, called *COMET*.

Frequently the need arises to represent temporal behavior of master data within existing, non-temporal data warehouses. We proposed five different approaches to represent the various types of modifications to master data, e.g. *Move*, *Split* and *Merge* in non-temporal OLAP systems and compare these approaches mentioned with respect to resource consumption and the support for multi-period comparisons.

References

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

[2] P. Chamoni and S. Stock. Temporal Structures in Data Warehousing. In *Proc. of the DaWak Conference*, Italy, 1999.

[3] J. Eder and C. Koncilia. Changes of Dimension Data in Temporal Data Warehouses. In *Proc. of the DaWak Conference*, Munich, Germany, 2001.

[4] J. Eder, C. Koncilia, and T. Morzy. The COMET Metamodel for Temporal Data Warehouses. In *Proc. of the 14th Int. Conference on Advanced Information Systems Engineering (CAISE'02)*, Toronto, Canada, 2002.

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

[6] General Register Office for Scotland. Annual Report of the Registrar General for Scotland. 2000. URL: http://www.gro-scotland.gov.uk/.

[7] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*. Springer-Verlag, 1998. in [5].

[8] A. Vaisman. *Updates, View Maintenance and Time Management in Multidimensional Databases*. Universidad de Buenos Aires, 2001. Ph.D. Thesis.

[9] J. Yang. *Temporal Data Warehousing*. Stanford University, June 2001. Ph.D. Thesis.