Managing Multiple Real and Simulation Business Scenarios by Means of a Multiversion Data Warehouse

Bartosz Bębel, Zbyszko Królikowski, and Robert Wrembel

Poznań University of Technology, Institute of Computing Science
Poznań, Poland
{bbebel,zkrolikowski,rwrembel}@cs.put.poznan.pl

Abstract. This paper addresses problems of the evolution of data warehouse schema and dimensions. In order to handle the evolution, we apply a multiversion data warehouse (MVDW). In this paper we discuss real world cases illustrating a DW evolution and show how to apply the MVDW in order to handle the cases.

1 Introduction

A data warehouse (DW) is a large database that integrates data from various external data sources (EDSs). Data integrated in a DW are analyzed by the so called On-Line Analytical Processing (OLAP) applications for the purpose of: discovering trends, patterns of behavior and anomalies as well as finding hidden dependencies between data. The process of decision making often requires forecasting future business behavior, based on present and past data as well as on assumptions made by decision makers. This kind of data processing is called a 'what-if' analysis. In this analysis, a decision maker simulates in a DW changes in the real world, creates virtual alternative business scenarios, and explores them with OLAP queries.

Existing DW technologies offer functionalities for managing data warehouses of static (time invariant) structures. In practice, however, changes to a DW structure are frequent. They are typically caused by changes in the structure of EDSs, new user requirements, changes in environments (legislation, administrative structure) where a business is run, needs to simulate alternative business scenarios. As a consequence, a DW structure evolves. Existing solutions in this field focus on schema and data evolution, simulation, temporal techniques, and versioning. Schema evolution approaches (e.g. [6]) maintain only one (current) DW state. The simulation approaches (e.g. [5]) simulate or screen the evolution by means of various tools and data structures. Temporal techniques (e.g. [9, 18, 14]) use timestamps on modified data in order to create temporal versions. In versioning approaches (e.g. [7, 16]), a DW evolution is managed partially by means of schema versions and partially by data versions. These approaches allow to manage a DW evolution partially since they do not offer a clear separation between different DW states. Moreover, they do not offer tools for modeling alternative business scenarios.

In our approach, we apply a multiversion data warehouse (MVDW) to the management of data warehouses that evolve in time. In the MVDW a schema and dimension evolution is represented by the sequence of persistent DW versions. A DW version corresponds either to the real world state or to a simulation scenario. The MVDW supports a correct representation of a DW evolution, the 'what-if' analysis, and the separation of different DW states.

In this paper we discuss real world cases of a DW evolution and show how to handle the cases by means of the MVDW. The work is based on our previous achievements in designing and implementing the MVDW [4, 25].

Section 2 outlines basic definitions used in this paper. Section 3 presents real cases of a DW evolution. Section 4 overviews our approach to handling a DW evolution and Section 5 shows how our approach can handle real cases. Section 6 discusses related work and Section 7 summarizes the paper.

2 Basic Definitions

The organization of data being analyzed conforms to the *multidimensional data model* [17] with *facts* representing elementary information being analyzed. A fact contains numerical features, called *measures*, e.g. quantity, income, turnover, price that quantify the fact and allow to compare different facts. Values of measures depend on a context set up by *dimensions*, e.g. *Time*, *Location*, *Product*. Dimensions usually form hierarchies, composed of *levels*. Values in every level are called *level instances*. Hierarchically assigned instances of levels in dimension D_i, where the hierarchy of level instances is set up by the hierarchy of levels, set up the *dimension instance* of D_i.

An example of a hierarchical dimension is *Geography* (cf. Figure 1a), with level *Countries* at the top, intermediate level *Regions*, and bottom level *Cities*. An example instance of dimension *Geography* is shown in Figure 1b.

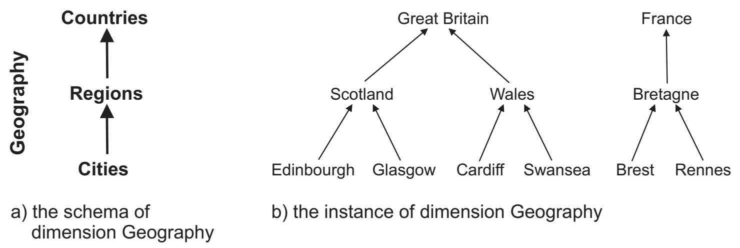


Fig. 1. An example dimension schema and its instance

The multidimensional data model can be implemented either in MOLAP (multidimensional OLAP) servers or in ROLAP (relational OLAP) servers. In the first case, data are stored in n-dimensional arrays. In the second case, data are stored relational tables. Basic ROLAP schemas are organized as star or snowflake structures [10]. Based on these basic schemas, one can build their variations, namely a fact constellation schema or a star-flake schema.

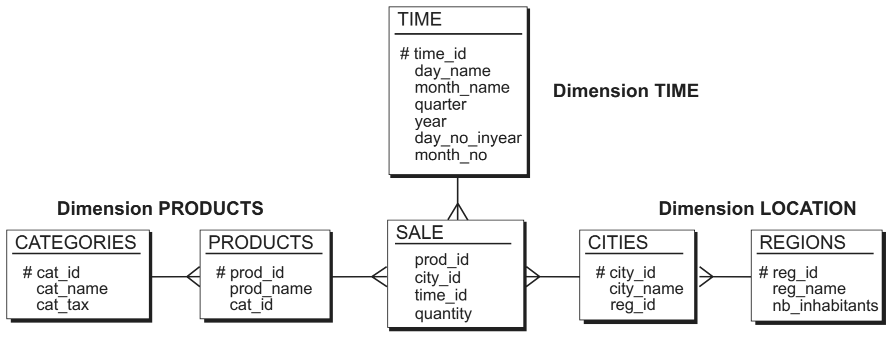


Fig. 2. An example ROLAP schema

An example star-flake DW schema is shown in Figure 2. It is used for analyzing sales of products by locations in various periods of time. It is composed of three following dimensions: *TIME* - with a denormalized level table *Time*, *LOCATION* - with normalized level tables *Cities* and *Regions*, *PRODUCTS* - with normalized level tables *Products* and *Categories*.

Without any loss of generality, in the reminder of this paper we will focus our discussion on the ROLAP implementation.

3 Motivating Examples

In this section we present real world examples illustrating the need of changes to a DW schema and structures of dimension instances.

3.1 Example 1: Schema Changes

This example comes from gambling machines business [12]. Let us assume that until time t_1 tax from gambling machines has been collected per every single machine. A simplified schema of a DW used for income analysis from this business is shown in Figure 3a. Since time t_2 tax has been collected per location, regardless the number of machines installed there. A simplified schema of a DW for the new scenario, valid from t_2, is shown in Figure 3b.

As we can observe in this example, the way of collecting taxes has impact on the schema of a DW. Until t_1 measures (*turnover* and *tax*) are analyzed in the context of time and machines, whereas from t_2 measures are analyzed in the context of time and locations.

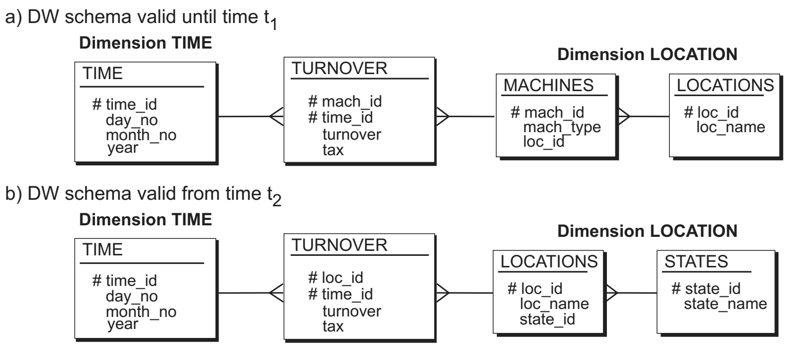


Fig. 3. Example schemas of a DW for gambling machines business

3.2 Example 2: Dimension Instance Structural Changes

The two examples presented here come from Poland and both illustrate cases where the structure of dimension instances changed. The first case concerns changing administrative division of a country. The second one concerns reclassification of building materials from one tax category to another one.

Changing administrative division of a country.
The administrative division of Poland until 1998 included 49 regions. In 1999, the number of regions was reduced to 19. Some regions retained their old names, but their borders where substantially changed. Let us now assume a data warehouse allowing to analyze monthly product sales per product category, per region, as shown in Figure 2.

Example instances of the *Location* dimension before and after 1999 are shown in Figure 4. R_i and NR_i denote a region name before 1999 and after 1999, respectively. C_i denotes a city name. Note that some regions (e.g. R_1 and R_2) retained their old names after this administrative division change, but since 1999 they include more cities than in 1998.

If we compared the yearly gross sales of bathroom equipment products (the *Categories* level) per region, we would observe sales increase in regions R_1 and R_2 in 1999 as compared to 1998. This increase would be caused by the changes in the structure of the *Location* dimension instances rather than by a real sales increase.

Reclassification of product to categories.
This real scenario took place in Poland after joining the EU in May 2004. Before joining the EU, building materials were sold with 7% VAT. After joining the EU, VAT was increased to 22%. The changes made to the *Products* dimension instances are schematically shown in Figure 5.

Let us assume that in our DW (cf. Figure 2) we compute and compare a gross sales of pine boards in consecutive months of 2004. In the results, we observe

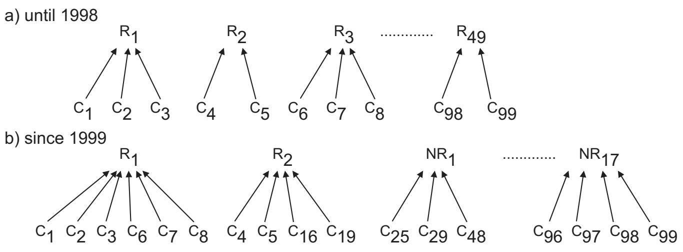

a) until 1998
R1
R2
R3
R49
C1 C2 C3 C4 C5 C6 C7 C8 C98 C99
b) since 1999
R1
R2
NR1
NR17
C1 C2 C3 C6 C7 C8 C4 C5 C16 C19 C25 C29 C48 C96 C97 C98 C99

Fig. 4. A schematic view on changes to the structure of the *Location* dimension instances

a remarkable increase in gross sales between April and May. In practice, this increase is mainly caused by VAT increase rather that by higher sales.

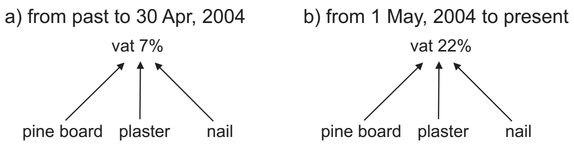


Fig. 5. A schematic view on changes to the structure of the *Products* dimension instances

3.3 Example 3: the What-if Analysis

In order to lower taxes, companies apply depreciation of their fixed assets. In Poland there are two different types of the depreciation, namely a linear one and a nonlinear one. A chosen depreciation type yields certain depreciation rate resulting in higher or lower taxes paid in a given year. In order to chose the depreciation type that most suits a company, multiple business simulation scenarios have to be created, one scenario for one depreciation type. Then in every scenario, appropriate depreciation type is applied and the results are compared.

This kind of simulation functionality was highly required by an existing Polish company, as part of its data warehouse system.

4 Multiversion Data Warehouse

The **multiversion data warehouse** is composed of the sequence of persistent versions [4]. A **DW version** is in turn composed of a schema version and an instance version. A **schema version** describes the structure of a DW within a given time period, whereas an **instance version** represents the set of data described by its schema version.

We distinguish two types of DW versions, namely real and alternative ones. *Real versions* are created in order to keep up with changes in a real business environment. Real versions are linearly ordered by the time they are valid within. *Alternative versions* are created for simulation purposes, as part of the 'what-if' analysis. Such versions represent virtual business scenarios. All DW versions are connected by version derivation relationships, forming a version derivation graph. The root of this tree is the first real version. Every DW version (a real and an alternative one) is valid within certain period of time represented by two timestamps, i.e. begin validity time and end validity time, that are used in queries.

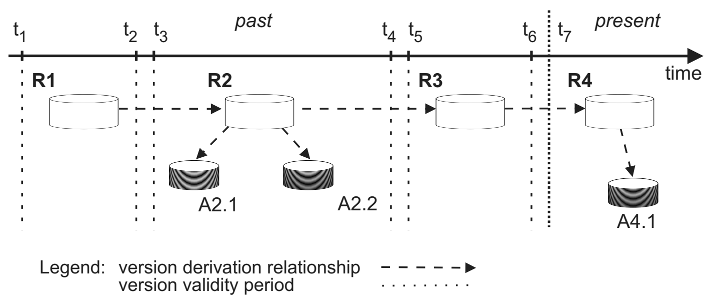


Fig. 6. A schematic view on a multiversion data warehouse, composed of real and alternative versions

Figure 6 schematically shows real and alternative versions. $R1$ is an initial real version of a DW. Based on $R1$, new real version $R2$ was created. Similarly, $R3$ was derived from $R2$, etc. $A2.1$ and $A2.2$ are alternative versions derived from $R2$; $A4.1$ is an alternative version derived from $R4$. $R1$ is valid from time t_1 (begin validity time) until t_2 (end validity time). Versions $R2$, $A2.1$, and $A2.2$ are valid from t_3 until t_4, etc.

5 Applying the MVDW to Real World Cases

In this section we demonstrate how the MVDW can be applied to real world cases discussed in Section 3. We illustrate the cases with our prototype system. It is written in Java and Oracle PL/SQL languages, whereas data and metadata are stored in an Oracle9i/10g database.

5.1 Schema Changes

The new way of collecting taxes from the gambling business, discussed in Section 3.1, is handled by the MVDW. Each way of collecting taxes is represented by a separate DW version. Thus, the MVDW is composed of two following versions: *RV1 (tax-machine)* and *RV2 (tax-location)*. The first one is valid from past to time t_1 and stores taxes collected in an old way. The second one is valid from time t_2 until present and stores taxes collected in a new way. In such MVDW, a user can address in a query either the first or the second DW version and then he/she can compare the query results.

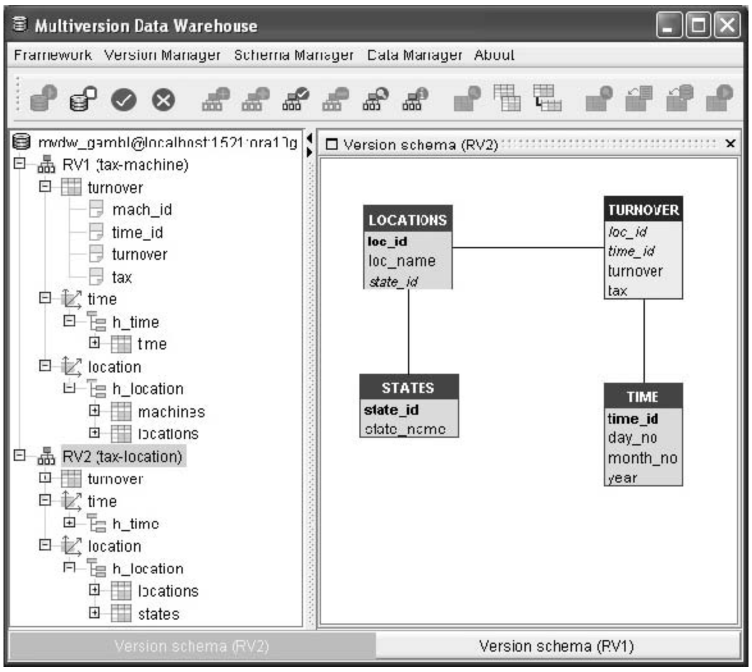
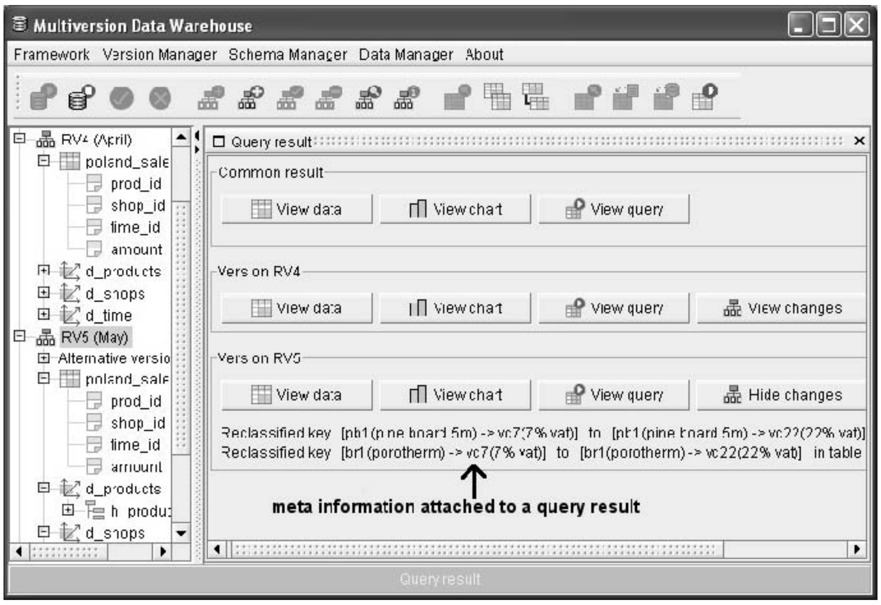


Fig. 7. An example of handling schema changes for the gambling business by the MVDW

An example window of our prototype system that manages these two DW versions is shown in Figure 7. A user interface is composed of two main panels. The left hand side panel - an *object navigator* allows browsing through versions of the MVDW. The schema (tables and their attributes, dimensions, hierarchies) of every DW version can be explored there. The left hand side panel - a *visualizer* is used for visualizing a schema of a selected DW version and for visualizing query results.

5.2 Dimension Instance Structural Changes

The changes in the structure of dimension instances, discussed in Section 3.2 are handled in separate versions. In the example describing the reclassification of products to categories, sales records until the 1st of May 2004 are stored in DW version *RV4 (April)*, whereas sales records collected from the 1st of May 2004 are stored in *RV5 (May)*, as shown in Figure 8.



Reclassified key [pb1(pine board 5m) -> vc7(7% vat)] to [pb1(pine board 5m) -> vc22(22% vat)]
Reclassified key [br1(porotherm) -> vc7(7% vat)] to [br1(porotherm) -> vc22(22% vat)] in table

Fig. 8. An example of handling changes in the structure of dimension instances

A user can query both DW versions at once by means of a multiversion query language that we have developed (cf. [25]). Such a query will be parsed into two partial queries - one for version *RV4 (April)* and one for version *RV5 (May)*. The partial queries are executed in their proper DW versions and their results are presented to a user. Additionally, the results are annotated with meta-information describing changes that were made to the queried DW versions. In our example from Figure 8, the meta-information describes changes made to the instance of the *Product* dimension. Thus, an analyst can easily recognize that in *RV5 (May)* the structure of instances of dimension *Products* changed as compared to *RV4 (April)*. Moreover, he/she can figure out what kinds of changes were applied and whether they may have impact on the results of this analysis.

5.3 The What-if Analysis

The requirement of multiple simulation business scenarios discussed in Section 3.3 can be fulfilled in our prototype by applying alternative DW versions.

In this scenario, every alternative version is derived from the same original real version leaving original data intact. An alternative version stores data being computed by a selected depreciation model and it stores the results of this computation. Thus, a user can apply different depreciation models to different alternative versions, and then he/she can compare their results. It is worth noticing that original data remain unchanged in a real version. The alternative versions may be persistent or transient.

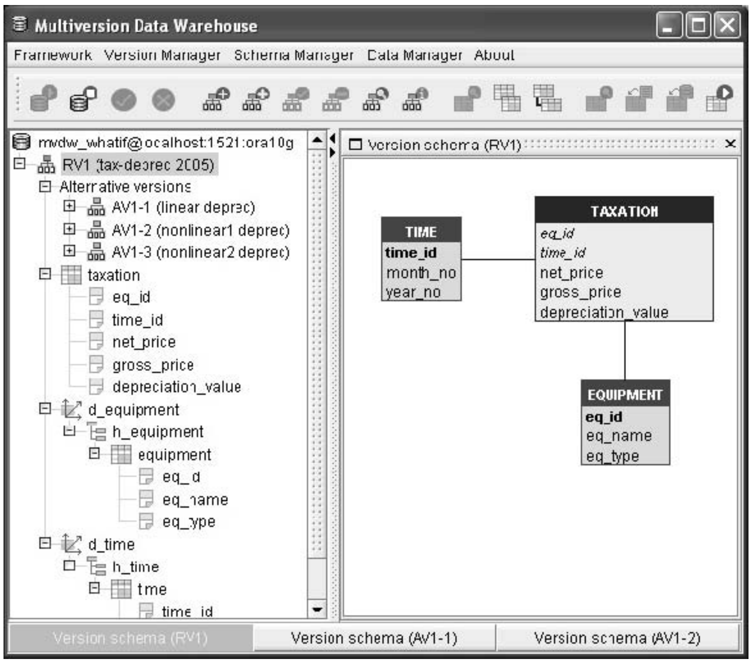


Fig. 9. An example of handling alternative business scenarios for the 'what-if' analysis

A simple setting for the discussed scenario in our MVDW is shown in Figure 9. The MVDW is composed of one real version named *RV1 (tax-deprec 2005)*. Three different alternative versions were derived from *RV1 (tax-deprec 2005)*, namely *AV1-1 (linear deprec)* storing data for linear depreciation, *AV1-2 (nonlinear1 deprec)* and *AV1-3 (nonlinear2 deprec)* storing data for two different nonlinear depreciation models.

6 Related Work

The support of schema and data evolution turned up to be required mainly in the applications of object-oriented databases to CAD and CASE systems. Various approaches and prototypes have been developed in this area, e.g. [1, 2, 8, 15, 21]. These and many other approaches have been proposed for versioning complex objects stored in a database of moderate size and they are inappropriate for managing the evolution in data warehouses. DWs have different data characteristics and processing characteristics. They store simple data (several tables) but data volumes are huge.

The solutions to handling changes/evolution in DWs can be categorized as follows: (1) *schema and data evolution* [6, 20, 18, 19], (2) *simulation* [3, 5], (3) *temporal extensions* [9, 13, 14, 23, 27, 22, 26, 24], and (4) *versioning extensions* [7, 16].

Schema evolution approaches maintain one DW schema and the set of data that evolve in time. Schema modifications (e.g. dropping an attribute, changing the length or domain of an attribute) require data conversions and, as a consequence, historical DW states are lost. Modifications of the structure of dimension instances is implemented by simple updates of attribute values. This also causes that old values are lost.

Simulation approaches use virtual data structures to simulate or to screen DW evolution. In the approach proposed in [5] virtual DW structure, called scenario, is constructed for hypothetical queries, for the purpose of the 'what-if' analysis. Then, the system using substitution and query rewriting techniques transforms a hypothetical query into an equivalent query that is run on a real DW. As this technique computes new values of data for every hypothetical query based on virtual structures, performance problems will appear for large DWs. The approach proposed in [3] simulates changes in a DW schema by means of views. The approach supports only simple changes in source tables (add, drop, modify an attribute) and it does not deal either with typical multidimensional schemas or the evolution of facts or dimensions.

Temporal extensions use timestamps on modified data in order to create temporal versions [9, 23, 26, 24]. Some of the approaches focus mainly on handling changes in the structure of dimension instances, reclassifying a lower level instance to a new upper level one, merging or splitting level instances [13, 14, 27, 22]. The concept presented in [13, 14] supports transformations of fact instance as a consequence of dimension instances' changes. To this end, system conversion methods are applied. The methods are expressed as matrices defining recalculations of facts. In [27], a time-stamped history of changes to dimension instances is stored in an additional data structure. The paper by [22] additionally proposes consistency criteria that every evolving dimension has to fulfill. It gives an overview how the criteria can be applied to a temporal DW only. All the discussed approaches from this category are suitable for representing historical versions of data, but not versions of a schema.

In *versioning extensions*, a DW evolution is managed partially by means of schema versions and partially by data versions. The versioning mechanism presented in [7] supports explicit, time-stamped persistent versions of data. This mechanism uses one central fact table for storing all versions of data. As a consequence, only changes to dimensions' structures and dimensions' instances structures are supported. In [16] an explicit DW schema versioning mechanism is presented. A new persistent schema version is created for handling schema changes. The approach supports only four basic schema modification operators, namely adding/deleting an attribute as well as adding/deleting a functional dependency. A persistent schema version requires a population with data, but this issue is not addressed in the paper. The approaches from this category solve the DW evolution problem partially. Firstly, they do not offer a clear separation between different DW states. Secondly, the approaches do not support modeling alternative, hypothetical DW states required for the 'what-if' analysis.

Commercial DW systems existing on the market (e.g. Oracle9i/10g, Oracle Express Server, IBM DB2 UDB, SybaseIQ, Computer Associates CleverPath OLAP, NCR Teradata, Hyperion Essbase OLAP Server, SAP Business Warehouse, MS SQL Server2000) do not offer mechanisms for managing multiple DW states.

7 Summary

Managing the evolution of DWs has been recognized as one of the most important research and technological issues in the field of data warehousing. The evolution concerns not only the structures of dimensions and dimension instances but also a DW schema. The evolution results among others from changes to external data sources, new user requirements, and simulation requirements.

In this paper we presented real cases illustrating the need of a DW evolution. Then we discussed our solution to this problem. The solution is based on the Multiversion Data Warehouse approach that consists in representing a schema and dimension evolution by the sequence of persistent DW versions [4]. A DW version corresponds either to the real world state (representing the content of EDSs within a given time period) or to a simulation scenario applied to the 'what-if' analysis. The MVDW approach provides a correct representation of a DW evolution and separates different DW states. It also supports modeling and managing alternative business scenarios. At the end of the paper we showed how our approach can be applied for handling real cases discussed in Section 3.

Current work focuses on developing index structures for multiversion data and on experimentally evaluating these structures. Future work will concentrate on extending the MVDW approach to handling changes in the Extraction-Translation-Loading processes.

References

1. Agrawal R., Buroff S., Gehani N., Shasha D.: Object Versioning in Ode. Proc. of ICDE, 1991
2. Ahmed-Nacer M., Estublier J.: Schema Evolution in Software Engineering Databases - A new Approach in ADELE environment. In Computers and Artificial Intelligence, 19, pp. 183-203, 2000

3. Bellahsene, Z.: View Adaptation in Data Warehousing Systems. Proc. of DEXA, 1998
4. Będel B., Eder J., Koncilia Ch., Morzy T., Wrembel R.: Creation and Management of Versions in Multiversion Data Warehouse. Proc. of ACM SAC, 2004
5. Balmin, A., Papadimitriou, T., Papakonstanitnou, Y.: Hypothetical Queries in an OLAP Environment. Proc. of VLDB, 2000
6. Blaschka M., Sapia C., Höfling G.: On Schema Evolution in Multidimensional Databases. Proc. of DaWaK, 1999
7. Body, M., Miquel, M., Bédard, Y., Tchounikine A.: A Multidimensional and Multiversion Structure for OLAP Applications. Proc. of ACM DOLAP, 2002
8. Cellary W., Jomier G.: Consistency of Versions in Object-Oriented Databases. Proc. of VLDB, 1990
9. Chamoni, P., Stock, S.: Temporal Structures in Data Warehousing. Proc. of DaWaK, 1999
10. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. SIGMOD Record, 26, 1997
11. Cui Y., Widom J.: Lineage Tracing for General Data Warehouse Transformations. Proc. of VLDB, 2001
12. Czejdo B., Messa K., Morzy T., Putonti C.: Design of Data Warehouses with Dynamically Changing Data Sources. Proc. of the Southern Conference on Computing, 2000
13. Eder, J., Koncilia, C.: Changes of Dimension Data in Temporal Data Warehouses. Proc. of DaWaK, 2001
14. Eder, J., Koncilia, C., Morzy, T.: The COMET Metamodel for Temporal Data Warehouses. Proc. of CAISE, 2002
15. Gançarski S., Jomier G.: A framework for programming multiversion databases. Data Knowledge Engineering, 36(1), pp. 29-53, 2001
16. Golfarelli M., Lechtenbörger J., Rizzi S., Vossen G.: Schema Versioning in Data Warehouses. Proc. of ER Workshops, 2004, LNCS 3289
17. Gyssens M., Lakshmanan L.V.S.: A Foundation for Multi-Dimensional Databases. Proc. of VLDB, 1997
18. Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Maintaining Data Cubes under Dimension Updates. Proc. of ICDE, 1999
19. Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Updating OLAP Dimensions. Proc. of ACM DOLAP, 1999
20. Kaas Ch.K., Pedersen T.B., Rasmussen B.D.: Schema Evolution for Stars and Snowflakes. Proc. of ICEIS, 2004
21. Kim W., Chou H.: Versions of Schema for Object-Oriented Databases. Proc. of VLDB, 1988
22. Letz C., Henn E.T., Vossen G.: Consistency in Data Warehouse Dimensions. Proc. of IDEAS, 2002
23. Mendelzon, A.O., Vaisman, A.A.: Temporal Queries in OLAP. Proc. of VLDB, 2000
24. Microsoft ImmortalDB. Retrieved November 25, 2005 from http://research.microsoft.com/db/ImmortalDB/
25. Morzy T., Wrembel R.: On Querying Versions of Multiversion Data Warehouse. Proc. of ACM DOLAP, 2004
26. Salzberg B., Jiang L., Lomet D., Barrena M., Shan J., Kanoulas E.: A Framework for Access Methods for Versioned Data. Proc. of EDBT, 2004
27. Schlesinger L., Bauer A., Lehner W., Ediberidze G., Gutzman M.: Efficienlty Synchronizing Multidimensional Schema Data. Proc. of ACM DOLAP, 2001