Metadata Management in a Multiversion Data Warehouse*

Robert Wrembel and Bartosz Bębel

Institute of Computing Science, Poznań University of Technology, Poznań, Poland
{Robert.Wrembel, Bartosz.Bebel}@cs.put.poznan.pl

Abstract. A data warehouse (DW) is supplied with data that come from external data sources (EDSs) that are production systems. EDSs, which are usually autonomous, often change not only their contents but also their structures. The evolution of external data sources has to be reflected in a DW that uses the sources. Traditional DW systems offer a limited support for handling dynamics in their structures and contents. A promising approach to this problem is based on a multiversion data warehouse (MVDW). In such a DW, every DW version includes a schema version and data consistent with its schema version. A DW version may represent a real state at certain period of time, after the evolution of EDSs or changed user requirements or the evolution of the real world. A DW version may also represent a given business scenario that is created for simulation purposes. In order to appropriately synchronize a MVDW content and structure with EDSs as well as to analyze multiversion data, a MVDW has to manage metadata. Metadata describing a MVDW are much more complex than in traditional DWs. In our approach and prototype MVDW system, a metaschema provides data structures that support: (1) monitoring EDSs with respect to content and structural changes, (2) automatic generation of processes monitoring EDSs, (3) applying the discovered EDS changes to a selected, DW version, (4) describing the structure of every DW version, (5) querying multiple DW versions of interest at the same time, (6) presenting and comparing multiversion query results.

1 Introduction

A data warehouse (DW) is a large database (often of terabytes size) that integrates data from various external data sources (EDSs). A DW content includes historical, summarized, and current data. Data warehouses are important components of decision support systems. Data integrated in a DW are analyzed by, so called, On-Line Analytical Processing (OLAP) applications for the purpose of: discovering trends (e.g. sale of products) patterns of behavior and anomalies (e.g. credit card usage) as well as finding hidden dependencies between data (e.g. market basket analysis, suggested buying).

* This work is partially supported by the grant no. 4 T11C 019 23 from the Polish State Committee for Scientific Research (KBN), Poland.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1347 – 1364, 2005.
© Springer-Verlag Berlin Heidelberg 2005

The process of good decision making often requires forecasting future business behavior, based on present and historical 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 possible business scenarios, and explores them with OLAP queries. To this end, a DW must provide means for creating and managing various DW alternatives, that often requires changes to the DW structure.

An inherent feature of external data sources is their autonomy, i.e. they may evolve in time independently of each other and independently of a DW that integrates them [40, 41]. The changes have an impact on the structure and content of a DW. The evolution of EDSs can be characterized by: **content changes**, i.e. insert/update/delete data, and **schema changes**, i.e. add/modify/drop a data structure or its property. Content changes result from user activities that perform their day-to-day work with the support of information systems. On the contrary, schema changes in EDSs are caused by: changes of the real world being represented in EDSs (e.g. changing borders of countries, changing administrative structure of organizations, changing legislations), new user requirements (e.g. storing new kinds of data), new versions of software being installed, and system tuning activities.

The consequence of content and schema changes at EDSs is that a DW built on the EDSs becomes obsolete and needs to be synchronized. Content changes are monitored and propagated to a DW often by means of materialized views [20]. and the history of data changes is supported by applying temporal extensions e.g. [12]. Whereas EDSs schema changes are often handled by applying schema evolution, e.g. [10, 26] and temporal versioning techniques [17, 18, 33]. In schema evolution approaches historical DW states are lost as there is only one DW schema that is being modified. In temporal versioning approaches only historical versions of data are maintained whereas schema modifications are difficult to handle.

In our approach, we propose a multiversion data warehouse (MVDW) as a framework for: (1) handling content and schema changes in EDSs, (2) simulating and managing alternative business scenarios, and predicting future business trends (a what-if analysis). A MVDW is composed of persistent versions, each of which describes a DW schema and content in a given time period.

In order to support the lifecycle of a DW, from its initial loading by ETL processes, periodical refreshing, to OLAP processing and query optimization, a DW has to provide metadata. Metadata are defined as data about a DW. They are used for improving a DW management and exploitation. There are two basic types of metadata, namely business and technical ones. **Business metadata** include among others: dictionaries, thesauri, business concepts and terminology, predefined queries, and report definitions. They are mainly used by end-users. **Technical metadata** include among others: a DW schema description and the definitions of its elements, physical storage information, access rights, statistics for a query optimizer, ETL process descriptions, and data transformation rules [45].

In the case of a multiversion data warehouse, metadata are much more complex than in traditional DWs and have to provide additional information. Industry standard metamodels, i.e. Open Information Model [36] and Common Warehouse Metamodel [15] as well as research contributions, e.g. [24, 37] have not yet considered the incorporation of metadata supporting either schema and data evolution or schema and data versioning.

This paper's focus and **contribution** includes the development of metaschemas for the support of: (1) detecting structural and content changes in EDSs and propagating them into a MVDW, (2) automatic generation of software monitoring EDSs, based on metadata, (3) managing versions of schemas and data in a MVDW, (4) executing queries that address several DW versions, (5) presenting and comparing multiversion query results. Based on the developed metamodel, a prototype MVDW system was implemented in Java and Oracle PL/SQL language, whereas data and metadata are stored in an Oracle10g database. To the best of our knowledge, it is the first approach and implemented system managing multiple, persistent, and separate DW versions as well as supporting the analysis of multiversion data.

The rest of this paper is organized as follows. Section 2 presents basic definitions in the field of DW technology. Section 3 discusses existing approaches to handling changes in the structure and content of a DW as well as approaches to metadata management. Section 4 overviews our concept of a multiversion DW and presents its metaschema. Section 5 presents the mechanism of detecting changes in EDSs and its underlying metamodel. Finally, Section 6 summarizes the paper.

2 Basic Definitions

A DW takes advantage of a multidimensional data model [21, 24, 29] with **facts** representing elementary information being the subject of analysis. A fact contains numerical features, called **measures**, that quantify the fact and allow to compare different facts. Values of measures depend on a context set up by **dimensions**. Examples of measures include: quantity, income, turnover, etc., whereas typical examples of dimensions include *Time*, *Location*, *Product*, etc. (cf. Fig. 1). In a relational implementation, a fact is implemented as a table, called a **fact table**, e.g. *Sales* in Fig. 1.

Dimensions usually form **hierarchies**. Examples of hierarchical dimensions are: (1) *Location*, with *Cities* at the top and *Shops* at the bottom, (2) *Product*, with *Categories* and *Items* (cf. Fig. 1). A schema object in a dimension hierarchy is called a **level**, e.g. *Shops*, *Cities*, *Categories*, *Items*, and *Time*. In a relational implementation, a level is implemented as a table, called a **dimension level table**.

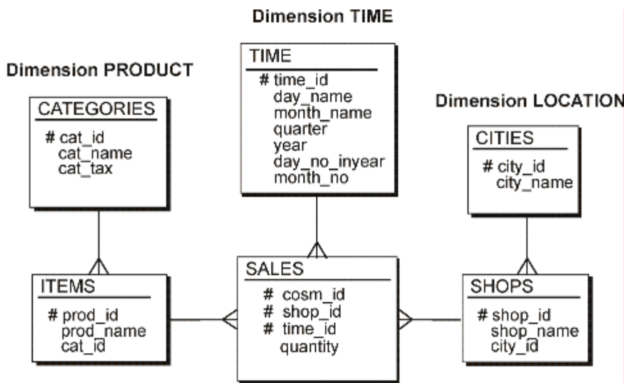

Fig. 1. An example DW schema on sale of products

A dimension hierarchy specifies the way measures are aggregated. A lower level of a dimension rolls-up to an upper level, yielding more aggregated data. Values in every level are called **level instances**. Example instances of level *Items* may include: *'t-shirt'* and *'shampoo'*, whereas instances of level *Categories* may include: *'clothes'* and *'cosmetics'*. The **dimension instance** of dimension D_i is composed of hierarchically assigned instances of levels in D_i, where the hierarchy of level instances is set up by the hierarchy of levels. Example instances of dimension *Product* include: {*'t-shirt'* → *'clothes'*, *'shampoo'* → *'cosmetics'*}, where → is the hierarchical assignment of a lower level instance to an upper level instance.

3 Related Work

The problem of schema changes appeared in mediated and federated database systems that were used for interconnecting heterogeneous data sources, e.g. [8, 9, 13, 30, 43]. A lot of research have been carried out in order to handle schema changes and propagate them into a global schema, e.g. [1, 6, 7, 31, 32]. Handling schema changes in EDSs and propagating them into a DW is partially based on that solutions. However, DW systems have different characteristics requiring new approaches to this problem. First of all, a final DW schema is usually totally different form EDSs schemas. Second of all, a DW stores persistent elementary data as well as data aggregated at many levels that have to be transformed after a DW schema updates.

The approaches to the management of changes in a DW can be classified as: (1) schema and data evolution: [10, 22, 23, 26, 27, 44], (2) temporal and versioning extensions [3, 11, 12, 14, 17, 18, 25, 28, 29, 33, 41].

The approaches in the first category support only one DW schema and its instance. In a consequence, many structural modifications require data conversions that results in the loss of historical DW states, e.g. dropping an attribute, changing an attribute data type, length or domain.

In the approaches from the second category, in [12, 17, 18, 33] changes to a DW schema are time-stamped in order to create temporal versions. The approaches are suitable for representing historical versions of data, but not schemas.

In [14, 41] data versions are used to avoid duplication anomaly during DW refreshing process. The work also sketches the concept of handling changes in an EDS structure. However, a clear solution was not presented on how to apply the changes to DW fact and dimension tables. Moreover, changes to the structure of dimensions as well as dimension instances were not taken into consideration.

In [25, 28, 38] implicit system created versions of data are used for avoiding conflicts and mutual locking between OLAP queries and transactions refreshing a DW.

On the contrary, [11] supports explicit, time-stamped versions of data. The proposed mechanism, however, uses one central fact table for storing all versions of data. In a consequence, only changes to dimension and dimension instance structures are supported. In [19] a 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. However, this issue is only mentioned in the paper. [42] addresses the problem of handling changes only in the structure of a

dimension instances. To this end, a time-stamped history of changes to dimension instances is stored in an additional data structure. The paper by [29] addresses the same problem and proposes consistency criteria that every dimension has to fulfill. It gives an overview how the criteria can be applied to a temporal DW only.

In [3] a virtual versioning mechanism was presented. A virtual DW structure is constructed for hypothetical queries simulating business scenarios. As this technique computes new values of data for every hypothetical query based on virtual structures, performance problems will appear for large DWs.

In order to handle schema and data evolution as well as versioning and in order to allow querying such evolving DW systems, the set of well defined metadata is required. From the approaches discussed above, only [18] presents a metamodel for a temporal DW. Additionally, [37] discusses and presents high level metamodel for handling and assuring data quality in a DW. The author only mentions the need for a metamodel supporting DW evolution, without giving any solutions.

The need for metadata describing multiple areas of a DW system design, development, deployment, and usage as well as the need for data exchange between different heterogeneous systems resulted in two industrial metadata standards, namely the *Open Information Model* (OIM) [24, 36, 45] and the *Common Warehouse Metadata* (CWM) [15, 24, 45], developed by multiple industry vendors and software providers. **OIM** was developed by the **Meta Data Coalition** (MDC) for the support of all phases of an information system development. OIM is based on UML, XML, and SQL92. It includes the following models: (1) object-oriented analysis and design, (2) object and component development life-cycles, (3) business engineering, (4) knowledge management tool, and (5) database and data warehousing model, including: database and multidimensional schema elements, data transformations, non-relational source elements, report definitions. OIM is supported among others by Microsoft, Brio Technologies, Informatica, and SAS Institute.

On the contrary, **CWM** was developed by the **Object Management Group** (OMG) for the support of integrating DW systems and business intelligence tools. The standard is based on XML, CORBA IDL, MOF, and SQL99. It includes the following models: (1) foundation of concepts and structures, (2) warehouse deployment, (3) relational interface to data, (4) record-oriented structures, (5) multidimensional database representation, (6) XML types and associations, (7) type transformations, (8) OLAP constructs, (9) warehouse process flows, (10) warehouse day-to-day operations. CWM is supported among others by IBM, Oracle, and Hyperion.

In 2000, the standard developed by MDC was integrated into the standard developed by OMG. Since then, the integrated standard is developed under OMG [46, 47].

None of the discussed standards, however, includes models supporting detection and propagation of changes from an EDS to a DW, or models supporting schema and data evolution in a DW. Consequently, they do not provide support for temporal or cross-version queries. Whereas our approach and implemented prototype system supports handling the evolution of DW schema and data by applying versioning mechanism. Moreover, a user can query multiple DW version, analyze, and compare the query results. In order to support these functionalities the system has to manage various metadata that are described by the metaschema that we have developed.

4 Multiversion Data Warehouse

This section informally overviews our concept of a multiversion DW, presents its metaschema, and outlines the approach to querying multiple DW versions. Formal description of a MVDW was presented in [34].

4.1 Basic Concepts

A **multiversion data warehouse** (MVDW) is composed of the set of its versions. 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, like for example: changing organizational structure of a company, changing geographical borders of regions, changing prices/taxes of products. 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. Fig. 2 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*, and *A4.1* is an alternative version derived from *R4*.

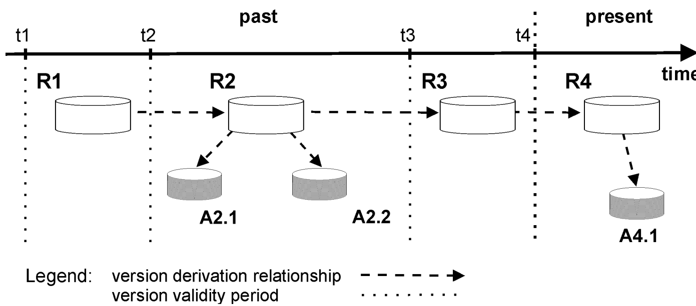

Fig. 2. An example real and alternative versions derivation graph and version validity times

Every DW version is valid within certain period of time represented by two timestamps, i.e. **begin validity time** (BVT) and **end validity time** (EVT) [5]. For example, real version *R1* (from Fig. 2) is valid within time *t1* (BVT) and *t2* (EVT), *R2* is valid within *t2* and *t3*, whereas *R4* is valid from *t4* until present. Alternative versions *A2.1*, *A2.2*, and *A4.1* are valid within the same time period as the real versions they were derived from.

A schema version, after being derived, is modified by means of operations that have an impact on a DW schema - further called schema change operations, as well as by operations that have an impact on the structure of a dimension instance - further

called dimension instance structure change operations. **Schema change operations** include among others: adding a new attribute to a level, dropping an attribute from a level, creating a new fact table, associating a given fact table with a given dimension, renaming a table, creating a new level table with a given structure, including a super-level table into its sub-level table, and creating a super-level table based on its sub-level table. The last three operations are applicable to snowflake schemas.

Dimension instance structure change operations include among others: inserting a new level instance into a given level, deleting an instance of a level, changing the association of a sub-level instance to another super-level instance, merging several instances of a given level into one instance of the same level, and splitting a given level instance into multiple instances of the same level. The full list of schema and dimension instance structure change operations with their formal semantics, their application to a MVDW, and their outcomes can be found in [4]. Their presentation is out of scope of this paper.

4.2 MVDW Metaschema

The model of a MVDW is composed of multiversion dimensions and multiversion facts. A multiversion dimension is composed of dimension versions. A dimension version is in turn composed of level versions that form hierarchies. A multiversion fact is composed of fact versions. A fact version is associated with several dimension versions. This association represents a cube version. A fact version and a dimension version can be shared by several DW versions.

The overall metaschema of our prototype MVDW is shown in Fig. 3. It is designed in the Oracle notation [2] where: a dashed line means a not mandatory foreign key, a solid line means a mandatory foreign key, a line end split into three means a relationship of cardinality many, whereas a simple line end means a relationship of cardinality one.

The *Versions* dictionary table stores the information about all existing DW versions, i.e. version identifier, name, begin and end validity times, status (whether a version is committed or under development), type (a real or an alternative one), parent-child (derivation) dependencies between versions. The meta information about fact versions is stored in the *Fact_Versions* dictionary table, i.e. fact identifier, name, an identifier of a multiversion fact a given fact belongs to, fact implementation name, DW version identifier a given fact belongs to, the identifier of a transaction that created a given fact. The meta information about dimension versions is stored in *Dim_Versions*, i.e. dimension version identifier, name, an identifier of a multiversion dimension a given dimension belongs to, DW version identifier a given dimension belongs to, the identifier of a transaction that created a given dimension.

The description of versions of hierarchies and their assignments to a given dimension version are stored in *Hier_Versions* and *Dim_Hier_Versions*, respectively. Versions of hierarchies are composed of level versions, whose descriptions are stored in *Lev_Versions*, i.e. level identifier, name, an identifier of a multiversion level a given level belongs to, implementation name, DW version identifier a given level belongs to, the identifier of a transaction that created a given level. Level versions are components of versions of level hierarchies. These associations are stored in *Hier_Elements*.

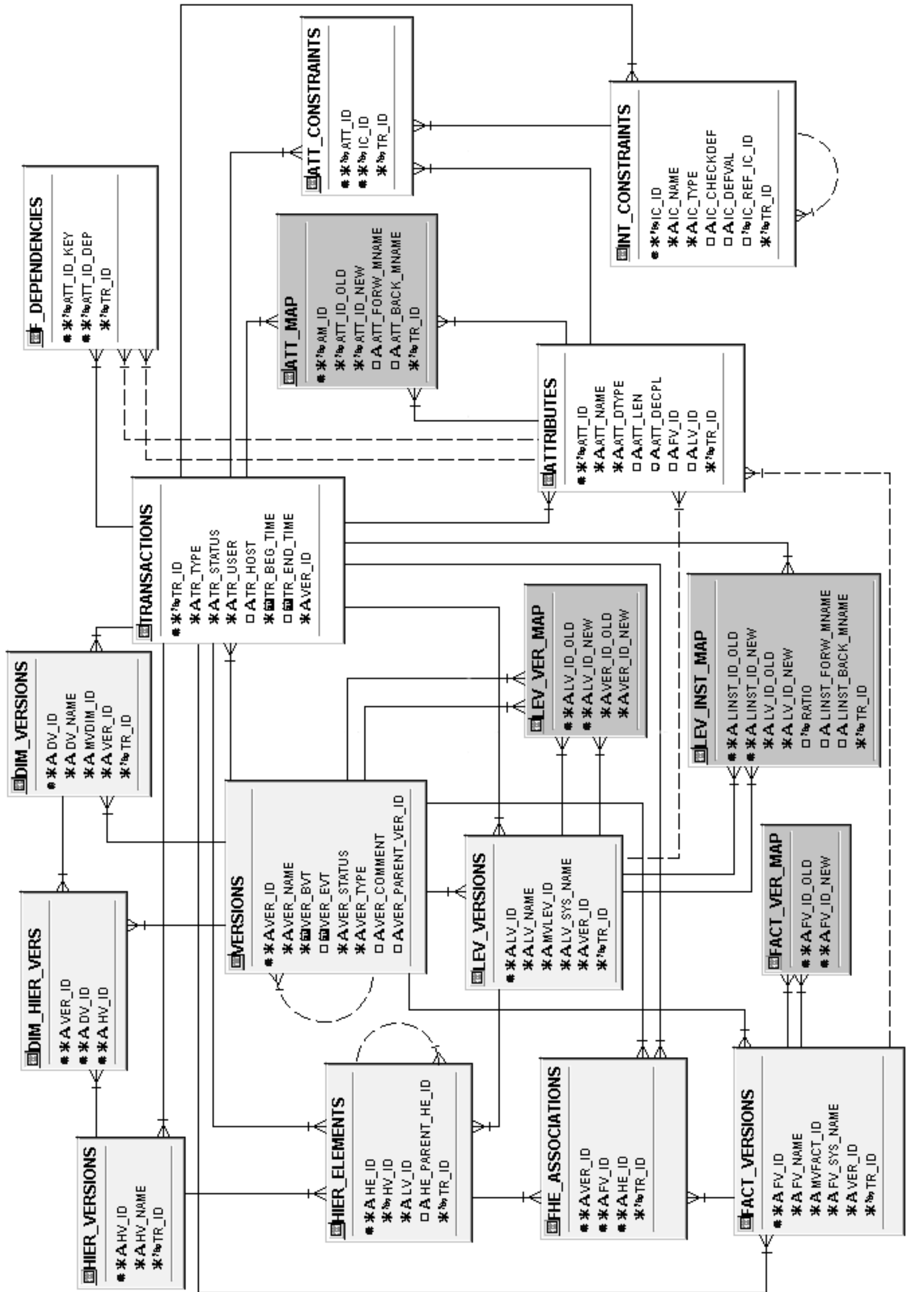

Fig. 3. The metaschema of our prototype MVDW

Fact versions are associated with dimension versions via level versions. The associations are stored in *FHE_Associations*. Every record in this metatable contains an identifier of a fact version, and identifier of the version of a hierarchy element (an association with the lowest level in a level hierarchy), an identifier of a DW version this association is valid in, and an identifier of a transaction that created this association.

Every fact version and level version includes the set of its attributes, that are stored in the *Attributes* dictionary table. Notice that attributes are not versioned in order to simplify the model. In a consequence, a single attribute can't be shared by multiple DW versions. Integrity constraints defined for attributes as well as for fact and level tables are stored in *Att_Constraints* and *Int_Constraints*. Functional dependencies between attributes in level versions are stored in *F_Dependencies*.

Table *Att_Map* is used for storing mappings between an attribute existing in DW version V_o and a corresponding attribute in a child version V_p. This kind of mappings are necessary in order to track attribute definition changes between versions, i.e. changing attribute name, data type, length, and integrity constraints. Some changes in attribute domain between two consecutive DW versions, say V_o and V_p (e.g. changing university grading scale from the Austrian one to the Polish one) will require data transformations, if the data stored in V_o and V_p are to be comparable. To this end, forward and backward conversion methods have to be provided. Their names are registered in *Att_Map* as the values of *Att_Forw_Mname* and *Att_Back_Mname*, respectively. In our prototype system, conversion methods are implemented as Oracle PL/SQL functions. The input argument of such a function is the name of an attribute whose value is being converted and the output is the converted value. Conversion methods are implemented by a DW administrator and they are registered in the metaschema by a dedicated application.

The *Fact_Ver_Map* dictionary table is used for storing mappings between a given fact table in DW version V_o and a corresponding fact table in version V_p, directly derived from V_o. This kind of mappings are necessary in order to track fact table definition changes between versions, i.e. changing table name or splitting a table.

The purpose of *Lev_Ver_Map* is to track changes of level tables between versions, i.e. changing table name, including a super-level table into its sub-level table, creating a super-level table based on its sub-level table, cf. [4].

As outlined in Section 4.1, the instances of level versions can be modified by changing associations to super-level instances as well as by merging and splitting them. Operations of this type result in new dimension instance structures. In order to allow querying multiple DW versions under such modifications, the system has to map level instances in version V_o into their corresponding instances that were modified in version V_p. To this end, the *Lev_Inst_Map* data dictionary table is used.

Example 1. In order to illustrate the idea and usage of mapping tables, let us consider a DW schema from Fig. 1 and let us assume that initially, in a real version from February (R^{FEB}) to March (R^{MAR}) there existed 3 shops, namely *ShopA*, *ShopB*, and *ShopC* that were represented by appropriate instances of the *Location* dimension. In April, a new DW version was created, namely R^{APR} in order to represent a new reality where *ShopA* and *ShopB* were merged into one shop - *ShopAB*. This change was reflected in the *Location* dimension instances. To this end, two following records were inserted to the *Lev_Inst_Map* dictionary table:

```
<id_ShopA,id_ShopAB,id_ShopsR^MAR,id_ShopsR^APR,
                                 100,null,null,id_tr>
<id_ShopB,id_ShopAB,id_ShopsR^MAR,id_ShopsR^APR,
                                 100,null,null,id_tr>
```

The first and the second value in the above records represents an identifier of *ShopA* and *ShopAB*, respectively. The third and fourth value represents the *Shops* level table identifier in version R^{MAR} and R^{APR}, respectively. The fifth value stores the merging/splitting ratio. In our example, the ratio equals to 100, meaning that the whole *ShopA* and *ShopB* constitute *ShopAB*.

For more advanced splitting or merging operations it will be necessary to provide a backward and forward transformation methods for converting facts from an old to a new DW version. If such methods are explicitly implemented and provided by a DW administrator, then their names are registered as the values of the sixth and seventh attribute. The last attribute stores transaction identifier of a transaction that carried out the modifications. □

The prototype MVDW is managed in a transactional manner and the *Transactions* dictionary table stores the information about transactions used for creating DW versions and modifying them.

4.3 Metadata Visualization - MVDW User Interface

A MVDW administrator manages the structure and content of a MVDW via a graphical application, implemented in Java. Its main management window is shown in Fig. 4. It is composed of the *version navigator*, located at the left hand side and the *schema viewer*, located at the right hand side. Both visualize the content of the MVDW metaschema.

The main functionality of the application includes:

- the derivation of a new (real or alternative) version of a data warehouse schema and its instance;
- the modification of a schema version and dimension instance structures, by means of operations outlined in Section 4.1;
- loading data from EDSs into a selected DW version (any ODBC data sources, sources accessible via a gateway, or text files can be used);
- visualizing the schema of a selected DW version;
- visualizing a schema version derivation graph;
- querying multiple DW versions and presenting query results.

4.4 Metadata in Multi-version Queries

The content of a MVDW can be queried either by a query that addresses a single version - further called a **single-version query** (SVQ) or by addressing multiple versions - further called a **multi-version query** (MVQ).

In a MVDW, data of user interest are usually distributed among several versions and a user may not be aware of the location of particular data. Moreover, DW versions being addressed in multi-version queries may differ with respect to their schemas. For these reasons, querying a MVDW is challenging and requires intensive usage of metadata.

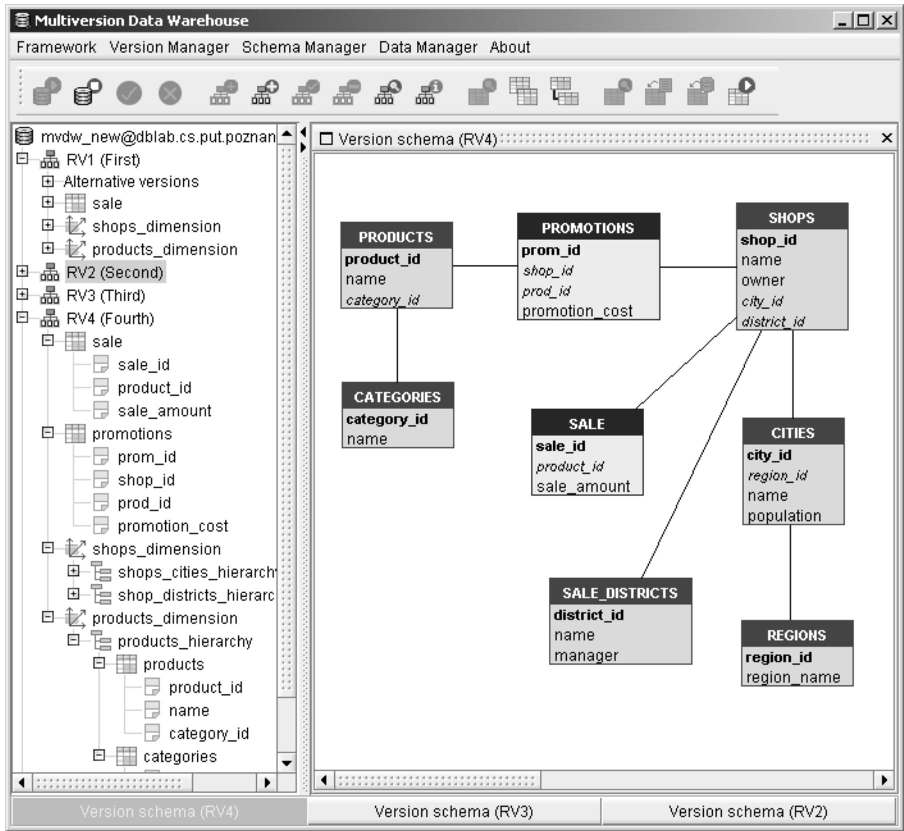

Fig. 4. The application for managing a MVDW

For the purpose of querying a MVDW, a traditional SQL `select` command has to be extended. To this end, we proposed clauses that allow querying: (1) a single DW version, that can be either a real or an alternative one, (2) the set of real DW versions, (3) the set of alternative DW versions. By including clauses (2) and (3) in one query, a user can query real and alternative versions at once. The detail description of the clauses as well as a user interface for specifying multi-version queries and visualizing their results is discussed in [35].

A user's multi-version query is processed by the MVQ parser and executor in the following steps.

1. **Constructing the set of DW versions**
   The set S^V of versions that is to be addressed in a MVQ is constructed by the parser by using version *begin validity time* and *end validity time* (cf. Section 4.1), which are stored in the *Versions* dictionary table.
2. **Decomposing MVQ**
   Next, for every DW version in S^V, the parser constructs an appropriate single-version query. In this process, the differences in version schemas are taken into consideration. If some tables and attributes changed names between versions, then

appropriate names are found in data dictionary tables and are used in SVQs. If an attribute is missing in DW versions V_i, V_j, V_k, then the attribute is excluded from single-version queries addressing V_i, V_j, V_k. The data dictionary tables used in this step include among others: *Fact_Versions*, *Dim_Versions*, *Hier_Verisions*, *Dim_Hier_Versions*, *Hier_Elements*, *FHE_Associations*, *Lev_Versions*, *Fact_Ver_Map*, *Lev_Ver_Map*, *Attributes*, *Att_Map*.

3. **Executing SVQs**
   Every single version query constructed in step 2) is next executed in its own DW version. Then, the result set of every SVQs is returned to a user and presented separately. Additionally, every result set is annotated with:
   – an information about a DW version the result was obtained from,
   – a meta information about schema and dimension instance changes between adjacent DW versions being addressed by a MVQ. The metadata information attached to SVQ result allows to analyze and interpret the obtained data appropriately.
4. **Integrating SVQ results**
   Result sets of single-version queries, obtained in step 3), may be in some cases integrated into one common data set. This set is represented in a DW version specified by a user (the current real version by default). This integration will be possible if a MVQ addresses attributes that are present in all versions of interest and if there exist transformation methods between adjacent DW versions (if needed). For example, it will not be possible to integrate the results of a MVQ addressing DW version V_o and V_p, computing the sum of products sold (`select sum(amount) ...`), if in version V_o attribute *amount* exists and in version V_p the attribute was dropped.
   While integrating result sets the following dictionary tables are used among others: *Fact_Versions*, *Fact_Ver_Map*, *Lev_Versions*, *Lev_Ver_Map*, *Attributes*, *Att_Map*, *Lev_Inst_Map*.

Example 2. In order to illustrate annotating result sets of SVQs with meta information let us consider a DW schema from Fig. 1 and let us assume that initially, in a real version from February R^{FEB}, there existed 3 shops, namely *ShopA*, *ShopB*, and *ShopC*. These shops were selling *Ytong bricks* with *7%* of VAT. Let us assume that in March, *Ytong bricks* were reclassified to *22%* VAT category (which is a real case of Poland after joining the European Union). This reclassification was reflected in a new real DW version R^{MAR}.

Now we may consider a user's MVQ that addresses DW versions from February till March and computes gross total sale of products. The query is decomposed into two partial queries: one for R^{FEB} and one for R^{MAR}. After executing the corresponding SVQs in their proper versions, the result set of SVQ addressing version R^{MAR} is augmented and returned to a user with meta information describing changes in the structure of the *Product* dimension instance between R^{FEB} and R^{MAR}, as follows:

```
Dimension PRODUCT: Level PRODUCTS:
          change association:
          Ytong bricks(vat 7% → vat 22%)
```

This way a sale analyst will know that a gross sale increase form February to March was at least partially caused by VAT increase. □

5 Detecting Changes in EDSs

For each external data source supplying a MVDW we define the set of events being monitored and the set of actions associated with every event.

5.1 Events and Actions

We distinguish two types of events, namely: structure events and data events. A **structure event** signalizes changes in an EDS's structure, that include: adding an attribute, modifying the name or domain of an attribute, dropping an attribute, adding a new data structure (table, class), dropping a data structure, changing the name of a data structure. A **data event** signalizes changes in an EDS's content, that include: adding, deleting, or modifying a data item. The set of events being monitored at EDSs is explicitly defined by a DW administrator and stored in so called **mapping metaschema**, cf. Section 5.2.

For every event in the set, a DW administrator explicitly defines one or more ordered **actions** to be performed in a particular DW version. We distinguish two kinds of actions, namely messages and operations. **Messages** represent actions that can not be automatically applied to a DW version, e.g. adding an attribute to an existing data structure at an EDS, creating a new data structure. These events may not necessarily require DW version modification if a new object is not going to store any information of user's interest. Messages are used for notifying a DW administrator about certain source events. Being notified by a message, an administrator can manually define and apply appropriate actions into a selected DW version. **Operations** are generated for events whose outcomes can be automatically applied to a DW version, e.g. the insertion, update, and deletion of a record, the modification of an attribute domain or name, the change of a data structure name. The associations between events and actions is stored in the mapping metaschema.

From the implementation point of view, operations are represented by SQL DML and DDL statements or stored procedures addressing an indicated DW version. The executable codes of operations and bodies of messages are automatically generated by monitors, cf. Section 5.3.

5.2 Mapping Metaschema

The structure of the mapping metaschema is shown in Fig. 5 (represented in the Oracle notation). The *SRC_SOURCES* dictionary table stores descriptions of external data sources. Information about EDSs data structures whose changes are to be monitored, are registered in two dictionary tables: *SRC_OBJECTS* and *SRC_ATTRIBUTES*. All monitored events at EDSs are stored in *SRC_EVENTS*.

DW_AC_SRC_EV_MAPPINGS stores mappings between events detected at EDSs and their associated actions that are to be executed in a given DW version. Action definitions, i.e. an action type and a data warehouse object an action is to be performed on, are stored in *DW_ACTIONS*. Data warehouse object descriptions (i.e. fact and dimension level tables, dimensions and hierarchies) are stored in the *DW_OBJECTS* and *DW_ATTRIBUTES* dictionary tables. Values taken from EDSs may need transformations before being loaded into a DW version (e.g. conversion of

GBP into Euro). Expressions that transform/compute values of attributes are stored in the *DW_ATTR_EXPRESSIONS*.

A DW administrator defines the content of the mapping metaschema (i.e. mappings between events and actions) by means of a graphical application, called the *metaschema manager* written in Java. The mapping metaschema in stored in an Oracle10g database.

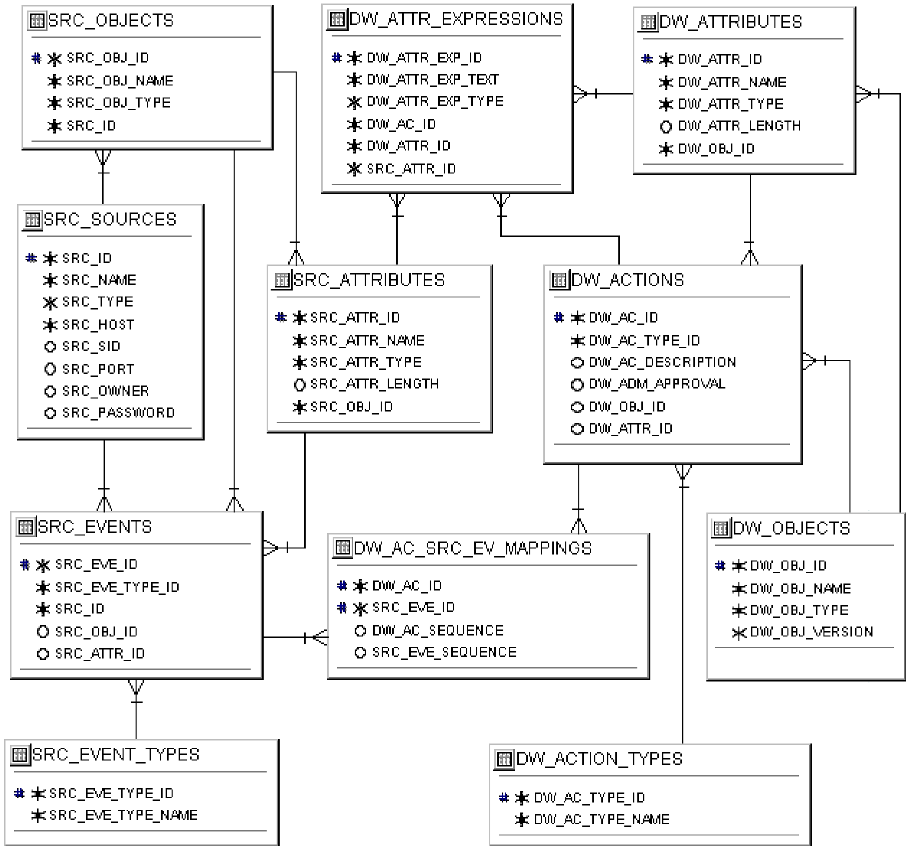

Fig. 5. The structure of the mapping metaschema

5.3 Automatic Generation of Monitors

External data sources are connected to a MVDW in a standard way via software called monitors. Each EDS has its own dedicated **monitor** that is responsible for detecting the occurrences of predefined events at that source. For every EDS, the code of its monitor is automatically generated by a software module called the **monitor generator**, based on the content of the mapping metaschema. In the current prototype system, monitors are implemented in the Oracle PL/SQL language as stored packages and triggers detecting defined events.

After installing monitors at EDSs, they generate executable codes of operations and bodies of messages in response to EDS events. Generated actions are stored in a special data structure called the **DW update register**. Every action is described by its type (message or DML statement), its content (e.g. SQL statement or stored procedure) addressing particular objects in a particular DW version, and its sequence that represents the order of action executions. When an administrator decides to refresh a DW version, he/she selects actions for execution and runs a dedicated process, called the **warehouse refresher**, that reads actions stored in the DW update register and applies them to a specified DW version.

6 Summary and Conclusions

Handling changes in external data sources and applying them appropriately into a DW became one of the important research and technological issues [16, 39]. Structural changes applied inappropriately to a DW schema may result in wrong analytical results. Research prototypes and solutions to this problem are mainly based on temporal extensions that limit their use. Commercial DW systems existing on the market (e.g. Oracle10g, Oracle Express Server, IBM DB2, SybaseIQ, Ingres DecisionBase OLAP Server, NCR Teradata, Hyperion Essbase OLAP Server, SAP Business Warehouse, MS SQL Server2000) do not offer mechanisms for managing multiple DW states.

Our approach to this problem is based on a multiversion data warehouse, where a DW version represents the structure and content of a DW within a certain time period. Managing multiple persistent versions of a DW allows to:

- store history of real world changes without loss of any information,
- create alternative business scenarios for simulation purposes,
- query multiple DW states and compare query results.

A fully functional DW system needs managing metadata in order to support the full lifecycle of a system. In the case of a multiversion data warehouse, metadata are much more complex than in traditional DWs and have to provide additional information, among others on: the structure and content of every DW version, a trace of schema and dimension instance changes applied to every DW version. Industry standard metamodels, i.e. Open Information Model and Common Warehouse Metamodel as well as research contributions have not yet considered the incorporation of metadata supporting either schema and data evolution or schema and data versioning.

In this paper we contributed by:

1. The development of a MVDW metamodel that is capable of: (1) managing versions of schemas and data in a MVDW, (2) executing queries that address several DW versions, and (3) presenting, comparing, and interpreting multiversion query results.
2. The framework for detecting changes in external data sources and propagating them into a MVDW, with the functionality of: (1) automatic generation of software monitoring EDSs, based on metadata, (2) automatic generation of actions in response to EDSs events.

Based on the developed metamodels, a prototype MVDW system was implemented in Java and Oracle PL/SQL language, whereas data and metadata are stored in an Oracle10g database.

In the current implementation, monitors are automatically generated for data sources implemented on Oracle databases. In future we plan to extend automatic generation of monitors for other database systems including: IBM DB2, Sybase Adaptive Server Enterprise, and MS SQL Server as well as for non-database sources including: text and XML files. Future work will also focus on extending our metamodels in order to handle data quality issues in a MVDW. Another interesting task is to extend the accepted industry standard CWM so that it is suitable for describing a multiversion DW.

References

1. Andany J., Leonard M., Palisser C.: Management of schema evolution in databases. Prof. of VLDB, 1991
2. Barker R.: Case*Method: Entity Relationship Modelling Addison-Wesley, 1990, ISBN 0201416964
3. Balmin, A., Papadimitriou, T., Papakonstanitnou, Y.: Hypothetical Queries in an OLAP Environment. Proc. of VLDB, Egypt, 2000
4. Bębel B.: Transactional Refreshing of Data Warehouses. PhD thesis, Poznań University of Technology, Institute of Computing Science, 2005
5. Bębel B., Eder J., Konicilia C., Morzy T., Wrembel R.: Creation and Management of Versions in Multiversion Data Warehouse. Proc. of ACM SAC, Cyprus, 2004
6. Bellahsene Z.: View Mechanism for Schema Evolution in Object-Oriented DBMS. Proc. of BNCOD, 1996
7. Benatallah B.: A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases. Proc. of ER, 1999
8. Bouguettaya A., Benatallah B., Elmargamid A.: Interconnecting Heterogeneous Information Systems, Kluwer Academic Publishers, 1998
9. Elmagarmid A., Rusinkiewicz M., Sheth A.: Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann Publishers, 1999
10. Blaschka, M. Sapia, C., Hofling, G.: On Schema Evolution in Multidimensional Databases. Proc. of DaWak99, Italy, 1999
11. Body, M., Miquel, M., Bédard, Y., Tchounikine A.: A Multidimensional and Multiversion Structure for OLAP Applications. Proc. of DOLAP, USA, 2002
12. Chamoni, P., Stock, S.: Temporal Structures in Data Warehousing. Proc. of DaWaK99, Italy, 1999
13. Chawathe S.S., Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J.D., Widom J.: The TSIMMIS project: Integration of heterogeneous information sources. Proc. of IPS, Japan, 1994
14. Chen J., Chen S., Rundensteiner E.: A Transactional Model for Data Warehouse Maintenance, Proc of ER, Finland, 2002
15. Object Management Group. Common Warehouse Metamodel Specification, v1.1. http://www.omg.org/cgi-bin/doc?formal/03-03-02
16. Panel discussion on "Future trends in Data Warehousing and OLAP" at DOLAP2004, ACM DOLAP, USA, 2004

17. Eder, J., Koncilia, C.: Changes of Dimension Data in Temporal Data Warehouses. Proc. of DaWaK, Germany, 2001
18. Eder, J., Koncilia, C., Morzy, T.: The COMET Metamodel for Temporal Data Warehouses. Proc. of CAISE, Canada, 2002
19. Golfarelli M., Lechtenbörger J., Rizzi S., Vossen G.: Schema Versioning in Data Warehouses. ER Workshops 2004, LNCS 3289
20. Gupta A., Mumick I.S. (eds.): Materialized Views: Techniques, Implementations, and Applications. The MIT Press, 1999, ISBN 0-262-57122-6
21. Gyssens M., Lakshmanan L.V.S.: A Foundation for Multi-Dimensional Databases. Proc. of the 23rd VLDB Conference, Grece, 1997
22. Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Maintaining Data Cubes under Dimension Updates. Proc. of ICDE, Australia, 1999
23. Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Updating OLAP Dimensions. Proc. of DOLAP, 1999
24. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer-Verlag, 2003, ISBN 3-540-42089-4
25. Kang, H.G., Chung, C.W.: Exploiting Versions for On–line Data Warehouse Maintenance in MOLAP Servers. Proc. of VLDB, China, 2002
26. Kaas Ch.K., Pedersen T.B., Rasmussen B.D.: Schema Evolution for Stars and Snowflakes. Proc. of ICEIS, Portugal, 2004
27. Koeller, A., Rundensteiner, E.A., Hachem, N.: Integrating the Rewriting and Ranking Phases of View Synchronization. Proc. of DOLAP, USA, 1998
28. Kulkarni, S., Mohania, M.: Concurrent Maintenance of Views Using Multiple Versions. Proc. of IDEAS, 1999
29. Letz C., Henn E.T., Vossen G.: Consistency in Data Warehouse Dimensions. Proc. of IDEAS, 2002
30. Levy A., Rajaraman A., Ordille J.: Querying heterogeneous information sources using source descriptions. Proc. of VLDB, 1996
31. McBrien P., Poulovassilis A.: Automatic Migration and Wrapping of Database Applications - a Schema Transformation Approach. Proc. of ER, 1999
32. McBrien P., Poulovassilis A.: Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. Proc. of CAiSE, 2002
33. Mendelzon, A.O., Vaisman, A.A.: Temporal Queries in OLAP. Proc. of VLDB, Egypt, 2000
34. Morzy, T., Wrembel, R.: Modeling a Multiversion Data Warehouse: A Formal Approach. Proc. of ICEIS, France, 2003
35. Morzy T., Wrembel R.: On Querying Versions of Multiversion Data Warehouse. Proc. ACM DOLAP, USA, 2004
36. Meta Data Coalition. Open Information Model. http://www.MDCinfo.com
37. Quix C.: Repository Support for Data Warehouse Evolution. Proc. of DMDW'99
38. Quass, D., Widom, J.: On–Line Warehouse View Maintenance. Proc. of SIGMOD, 1997
39. Rizzi S.: Open Problems in Data Warehousing: 8 Years Later. Keynote speech at DOLAP2003, ACM DOLAP, USA, 2003
40. Roddick J.: A Survey of Schema Versioning Issues for Database Systems. In Information and Software Technology, volume 37(7):383-393, 1996
41. Rundensteiner E., Koeller A., and Zhang X.: Maintaining Data Warehouses over Changing Information Sources. Communications of the ACM, vol. 43, No. 6, 2000
42. Schlesinger L., Bauer A., Lehner W., Ediberidze G., Gutzman M.: Efficienlty Synchronizing Multidimensional Schema Data. Proc. of DOLAP, Atlanta, USA, 2001

43. Templeton M., Henley H., Maros E., van Buer D.J.: InterVisio: Dealing with the complexity of federated database access. The VLDB Journal, 4(2), 1995
44. Vaisman A.A., Mendelzon A.O., Ruaro W., Cymerman S.G.: Supporting Dimension Updates in an OLAP Server. Proc. of CAISE02 Conference, Canada, 2002
45. Vetterli T., Vaduva A., Staudt M.: Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metadata. SIGMOD Record, vol. 29, No. 3, Sept. 2000
46. http://xml.coverpages.org/OMG-MDC-20000925.html
47. http://www.omg.org/news/releases/pr2000/2000-09-25a.htm