TRANSACTION CONCEPTS FOR SUPPORTING CHANGES IN DATA WAREHOUSES[*]

Bartosz Bębel, Zbyszko Królikowski, Tadeusz Morzy, Robert Wrembel
Institute of Computing Science, Poznań University of Technology, Poznań, Poland

Keywords: data warehouse, materialized view refreshing, schema modification, data and schema versioning, transaction

Abstract: A data warehouse (DW) provides information from external data sources for analytical processing, decision-making, and data mining tools. External data sources are autonomous, i.e. they change over time, independently of a DW. Therefore, the structure and content of a DW has to be periodically synchronized with its external data sources. This synchronization concerns DW data as well as schema. Concurrent work of synchronizing processes and user queries may result in various anomalies. In order to tackle this problem we propose to apply a multiversion data warehouse and an advanced transaction mechanism to a DW synchronization.

1 INTRODUCTION

A data warehouse (DW) integrates autonomous and heterogeneous external data sources (EDSs) in order to provide the information for analytical processing, decision-making, and data mining tools. The operational data, produced by OLTP (On-Line Transaction Processing) applications are periodically loaded into a DW, previously being cleaned, integrated, and often summarized. Then the data are processed by OLAP (On-Line Analytical Processing) applications in order to discover trends, anomalies, patterns of behavior, to predict future, and support pertinent business decisions. The subjects of analysis are called *facts* and they are described by *dimensions* that set the context of facts.

External data sources are autonomous, i.e. they change over time, independently of a DW. We distinguish three types of data source modifications, which imply changes in a DW, i.e. fact data changes, dimension data changes, and schema changes. *Fact data changes* represent modification of data, which are sources for facts in data warehouse. *Dimension data changes* represent modifications of data, which are sources for data warehouse dimensions. A content modification in the EDS, e.g., inserting a new record, may lead to a modification of a dimension structure in a DW. For instance, inserting a new product in an EDS may lead to modification of the structure of the *Products* dimension in a DW. So far the dimension information has been treated as static one. But in real life the changes to this kind of information are common. *Schema changes* represent changes to the schema of EDSs, e.g. adding an attribute to a table, dropping an attribute, adding a new table.

As data sources change, warehouse data become obsolete and therefore, the structure and content of a DW have to be periodically refreshed. An efficient refreshing a DW under EDSs content changes is one of the basic problems in data warehouse research area. A DW needs to be informed when some changes appeared in EDSs data. This functionality is the most often provided by EDS wrapper, which observes data source state and sends notifications to a DW when changes take place. Having received such a notification, a DW may begin its refreshing process.

Data in a DW are stored in materialized views, which are usually defined by SPJ (selection-projection-join) query over tables in EDSs. There are two basic techniques of refreshing a materialized view, namely full refreshing and incremental refreshing. The first technique consists in re-computing a materialized from scratch. Whereas an incremental refreshing consists in finding and

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

BÄŹbel B., Królikowski Z., Morzy T. and Wrembel R. (2004).
TRANSACTION CONCEPTS FOR SUPPORTING CHANGES IN DATA WAREHOUSES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 290-297
DOI: 10.5220/0002636702900297
Copyright © SciTePress

applying to a materialized view only these changes in source tables that appeared since the last refresh. However, the result of this process can be incorrect because of concurrent source tables updates. Many algorithms proposed so far try to solve this problem using wide range of mechanisms, e.g. (Zhuge, Garcia-Molina, Hammer, Widom 1995, Zhuge, Garcia-Molina, Wiener 1996).

Another problem with a DW refreshing is caused by concurrent work of refreshing processes and users running analytical queries. Typical way of data analysis, performed by a DW user, consists of series of long-lasting, complicated queries, built with join, group-by, sort, and aggregate operations, reading large amounts of data. The problem is how to synchronize DW users' work with the process of refreshing a DW? Users' queries need to see a consistent state of data, but concurrent data warehouse refreshing can violate this requirement. The refreshing techniques proposed so far, called commonly *on-line warehouse maintenance*, use multiversion concurrency control algorithms (Quass, Widom, 1997). The limitation of these algorithms is that they do not support the mechanism of transaction with its atomicity, consistency, isolation, and durability properties. In a consequence, a DW user may see various data anomalies while a DW is being refreshed. The only algorithm for transactional DW refresh was proposed by (Chen, Chen, Rundensteiner 2000), however proposed solution does not cover all aspects of DW dynamics (for example dimension updates).

1.1 Basic Definitions

The most popular data model of DWs is based on multidimensional data cubes, where *measures* – the instances of *facts*, i.e. subjects of analysis, are described in terms of *dimensions*. Examples of measures include: number of items sold, income, turnover, etc. Typical examples of dimensions are *Time*, *Geography*, *Products*, etc. A value of a measure in n-dimensional cube is referenced by n-dimensional vector, where each element corresponds to an element of a dimension. Dimensions are usually organized in hierarchies. An example of a hierarchical dimension is *Geography*, with *Countries* at the top, that are composed of *Regions*, that in turn are composed of *Cities*, i.e. *Cities -> Regions -> Countries*. A schema object in a hierarchy is called a *level*. Values in every level are called *dimension members*.

Multidimensional cubes can be implemented either in *MOLAP* (multidimensional OLAP) or in *ROLAP* (relational OLAP) servers. In the former case, a cube is stored in multidimensional array. In the latter case, a cube is implemented as the set of relational tables, some of them represent dimensions, and are called *dimension tables*, while others store values of measures, and are called *fact tables*.

In the rest of this paper, we will use the definitions of a DW schema and a DW instance. A *schema* of a DW is composed of the set of all dimensions, dimension levels, dimension members and facts. Whereas an *instance* of a DW consists of measures, i.e. cell values stored in fact tables.

1.2 Motivating Examples

Changes to the content and structure of EDSs, as well as concurrent work of user queries and refreshing processes may lead to serious DW anomalies.

In order to illustrate these anomalies, let us assume the existence of three data sources: DS_1 with table *Categories* (storing categories of products), DS_2 with table *Products* (storing product descriptions), and DS_3 with table *Sales* (storing records about sale of products). A DW integrates these three sources. Its schema is composed of two dimensions, namely *Category* and *Time*. The latter is organized hierarchically as follows: *Month -> Day*. Fact table *Daily_Sales* stores information about total sales of products in every day of a year. The *Daily_Sales* fact table is a materialized view, whose query joins records from *Products* at DS_2 and *Sales* at DS_3, and groups sale records by the category of product and the day of sale. Second materialized view, i.e. *Monthly_Sales*, aggregates daily sales in the *Time* dimension.

Example 1 – incorrect refreshing a materialized view

Let us assume that transaction T_1 at data source DS_2 inserts a new product "soap" of category "Hygiene" into table *Product*. The data source notifies a DW. In a consequence, the DW sends its maintenance query Q_1 to DS_3 in order to retrieve a delta, i.e. new sales for product "soap" from category "Hygiene". This delta will refresh materialized view *Daily_Sales*. During the transfer of Q_1 to DS_3, another transaction, T_2 at DS_3, inserts into table *Sales* a record describing the sale of product "soap". Next, T_2 commits and notifies the DW. When Q_1 arrives at DS_3, it retrieves a record, which didn't exist when maintenance query Q_1 was sent. The delta is sent back to the DW and it refreshes *Daily_Sales*. In a meantime, the DW receives notification from DS_3, sends maintenance query Q_2 to DS_2 and receives the delta, which is **the same record** as the one retrieved by Q_1. The materialized view *Daily_Sales* will be refreshed twice with the same data, and its content will become incorrect. This situation is called *duplication anomaly* and occurs during concurrent data sources updates. Additional, overflow records that make materialized view incorrect are called an *error term* (Zhuge, Garcia-Molina, Hammer, Widom 1995, Zhuge, Garcia-Molina, Wiener 1996).

Example 2 – inconsistent content of dependent materialized views

Let us assume that a DW user begins its OLAP session for analyzing monthly sales of given categories (he/she queries the *Monthly_Sales* materialized view). During his/her work, some products were sold, so the *Sales* table at data source DS_3 was updated. This change triggers refreshing the *Daily_Sales* fact table. As *Monthly_Sales* depends on *Daily_Sales* it should also be refreshed. If refreshing *Daily_Sales* and *Monthly_Sales* is not realized as a transaction, a user querying *Monthly_Sales* and drilling down to *Daily_Sales* will see inconsistent data (Zhuge, Garcia-Molina, Wiener 1997).

Example 3 – wrong interpretation of results

Let us assume that the 1st March, 2003 category "Hygiene" was merged into category "Cosmetics". In a consequence, each product from table *Products*, which previously belonged to category "Hygiene", belongs to category "Cosmetics". Let us take a look at DW user's analysis. If a user retrieves information about monthly sales of products from category "Cosmetics" he/she will observe that in March the total sale of "Cosmetics" grows rapidly. A user can then draw wrong conclusions, not knowing that category "Hygiene" was merged into "Cosmetics". This problem was caused by data sources updates, which resulted in dimension change in a DW.

Example 4 – refreshing under concurrent fact and dimension updates

This example shows potential problems when concurrent fact and dimension updates occur. Let us review the following sequence of transactions at data sources. T_1 at DS_3 inserts a record about sale of product "soap" from category "Hygiene". Before T_1 commits, T_2 at DS_2 changes the category of "soap" from "Hygiene" to "Cosmetics". Next, T_1 and T_2 commit. Then T_3 at DS_3 inserts sales of "soap"; this time "soap" belongs to "Cosmetics". Now the problem appears how a DW should be refreshed? Without a suitable concurrency control mechanism, all sales of "soap" will refresh the *Daily_Sales* for category "Cosmetics", which is incorrect since the sales, committed by T_1, were for product of category "Hygiene".

Example 5 – broken query

A serious problem with refreshing a DW may appear if during the execution of transaction T_1 at data source DS_2 the schema change occurs at DS_3 (for example one attribute of table *Sales* is dropped). A maintenance query sent by a DW to DS_3, as a reaction for notification received from DS_2, can not be executed since it is incorrect because of a schema change at DS_3. Refreshing process cannot be completed. DW schema has to be rebuilt in order to reflect schema changes at data sources.

Observations. The examples presented above lead us to two following observations. Firstly, *incorrect refreshing of a materialized view* (Example 1) and *inconsistent content of dependent materialized views* (Example 2) anomalies are caused by the lack of transaction mechanism applied to DW refreshing. Moreover, in order to alleviate the *incorrect refreshing of a materialized view* anomaly, concurrent changes in EDSs have to be serialized. Secondly, *wrong interpretation of results* (Example 3), *refreshing under concurrent fact and dimension* (Example 4), and *broken query* (Example 5) anomalies are caused by such changes in EDSs that have impact on a DW schema. In order to handle this kind and other kinds of schema changes, a DW has to: either (1) dynamically adapt its structure and transform existing data to a new structure, or (2) use versioning mechanism of schema and data, what we propose.

1.3 Contribution

Our approach to the problem of maintaining a DW under changes of schemas and contents of EDSs is based on explicit versioning the whole DW (i.e. schema and data) (reference removed for blind reviewing). Changes into a DW structure and data are reflected in a new, explicitly derived, version of a DW.

Maintaining versions of the whole DW allows us on the one hand, to run queries that span multiple versions and compare various factors computed in those versions, and on the other hand, to create and manage alternative virtual business scenarios.

Moreover, in order to assure consistent view of a DW for a user, while the DW is being refreshed, and to assure correctness of the whole DW refreshing process we propose two types of transactions (1) a fact transaction and (2) a schema and dimension structure transaction. A *fact transaction* is responsible for incremental refreshing DW fact tables to reflect updates of underlying EDSs' data. Whereas a s*chema and dimension transaction* is applied to: (1) modifying a DW schema when EDSs' schemas change, and (2) to modifying a DW dimension structures when EDSs' data reflected in dimensions change.

Paper organization. The rest of this paper is organized as follows. Section 2 overviews related work in the area of DW maintenance under schema and data changes in EDSs. Section 3 briefly presents our concept a multiversion data warehouse. Section 4 discusses our concept of transactions in a DW. Section 5 presents the metamodel of our multiversion data warehouse. Finally, Section 6 concludes the paper.

2 RELATED WORK

The existing approaches to propagating changes from EDSs to a DW can be classified into two categories: (1) data refreshing and (2) handling changes in a DW schema.

The solutions in the first category incrementally refresh DW fact table using different mechanism for avoiding duplication anomaly. The *ECA* algorithm (Zhuge, Garcia-Molina, Hammer, Widom, 1995), removes an error term by applying so called compensating queries. Two extensions of the basic *ECA* algorithm, namely ECA^K and ECA^L, are able to process some data source modifications locally at DW, i.e. without sending maintenance queries to EDSs. The same idea is used in the *Sweep* algorithm (Agrawal, El Abbadi, Singh, Yurek, 1997). Next solution, i.e. the *Strobe* algorithm (Zhuge, Garcia-Molina, Wiener, 1996), stores the list of EDSs' updates reported to DW during maintenance query execution. This list, called an action list, is used for compensating an error term. (Mostefaoui, Raynal, Roy, Agrawal, 2002) propose an architecture where EDSs form a ring. The process of finding delta caused by EDSs' updates is based on exchanging a token among EDSs. None of above solutions uses transactional refreshing a DW. In a consequence the atomicity and isolation of the refreshing process cannot be guaranteed. Moreover, the above approaches focus on only those changes in EDSs' data that do not have any impact on a DW schema.

The only transactional solution to the problem of incremental DW refresh is, to the best of our knowledge, (Chen, Chen, Rundensteiner 2000). The authors propose a special purpose transaction, called *DWMS_Transaction*, which covers the whole process of a DW fact table refreshing. The *DWMS_Transaction* has been defined as a sequence of two transactions, namely local EDS update transaction and its corresponding DW maintenance transaction. The main contribution of the reported work is an observation that the anomaly during the process of incremental refreshing can be mapped into the problem of guaranteeing the serializability of *DWMS_Transactions*. The authors point out that *DWMS_Transaction* is rather conceptual than a real transaction mechanism, which is the potential solution's weakness. However, even such conceptual model of transaction allows to reformulate a maintenance anomaly problem to well-known "read dirty data" problem. The compensation techniques are no longer required. The solution also deals with schema changes, but does not tackle the problem of data warehouse dimension structure changes and concurrent DW users' sessions.

The support for handling changes in a DW schema was studied in the two following categories: (1) schema and data evolution, (2) temporal and versioning extensions. The approaches in the first category (Koeller, 1998), (Blaschka, 1999), (Hurtado, 1999a), (Hurtado, 1999b) support only one DW schema and its instance. When a change is applied to a schema all data described by the schema must be converted, that incurs high maintenance costs.

In the approaches from the second category, in (Eder, Koncilia, 2001), (Eder, Konicilia, Morzy, 2002), (Chamoni, Stock, 1999), (Mendelzon, Vaisman, 2000) changes are time stamped in order to create temporal versions. However, the last two approaches expose their inability to express and process queries that span or compare several temporal versions of data. On the contrary, the model and prototype of a temporal DW presented in (Eder, Koncilia, 2001), (Eder, Koncilia, Morzy, 2002) support queries for a particular temporal version of a DW or queries that span several versions. In the latter case, conversion functions must be applied, as data in temporal versions are virtual.

In (Kang, Chung, 2002), (Kulkarni, Mohania, 1999), (Quass, Widom, 1997), (Rundensteiner, Koeller, Zhang 2000) implicit versioning in a DW was proposed. In all of the four approaches, versions are used for avoiding conflicts and mutual locking between OLAP queries and transactions refreshing a DW. Versions are implicitly created and removed by the system, which is a drawback of these approaches. On the contrary, (Bellahsene, 1998) proposes permanent user defined versions of views in order to simulate changes in a DW schema. However, the approach supports only simple changes in source tables and it does not deal either with typical multidimensional schemas or evolution of facts or dimensions. Also (Body et al., 2002) supports permanent time stamped versions of data. The proposed mechanism, however, uses one central fact table for storing all versions of data. In a consequence, the set of schema changes that may be applied to a DW is limited, and only changes of dimensions' structure are supported.

3 MULTIVERSION DATA WAREHOUSE

This section informally overviews our concept of a multiversion DW. Its formal description was presented in Morzy, Wrembel, 2003 (reference removed for blind reviewing).

In order to be able to manage changes in a DW schema we developed the model of a DW with versioning capabilities. In our approach, changes to a schema may be applied to a new version of a DW. This version, called a child version, is derived from a previous version, called a parent version. Versions of a DW form a *version derivation graph*. Each node of this graph represents one version, whereas edges represent *derived–from* relationships between two consecutive versions. In our approach, a version derivation graph is a DAG.

A *multiversion data warehouse* (MVDW) is composed of the set of its versions. Every version of a MVDW is in turn composed of a schema version and an instance version. The latter stores the set of data consistent with its schema version.

In our approach we distinguish two following kinds of DW versions: real versions and alternative versions. A *real version* reflects changes in the real world. Real versions are created in order to keep up with the changes in a real business environment, like for example: changing organizational structure of a company, changing geographical borders of regions, creating and closing shops, changing prices/taxes of products. Real versions are linearly ordered by the time they are valid within.

The purpose of maintaining *alternative versions* is twofold. Firstly, an alternative version is created from a real version in order to support the what-if analysis, i.e. it is used for simulation purposes. Several alternative versions may be created from the same real versions. Secondly, such a version is created in order to simulate changes in the structure of a DW schema. The purpose of such versions is mainly the optimization of a DW structure and system tuning. A DW administrator may create an alternative version that would have a simple star schema instead of an original snowflake schema, and then test the system performance using new data structures.

Every version is valid within certain period of time. In order to check a version validity, every real and alternative DW version has associated, so called valid time, represented by two timestamps, i.e. begin valid time (BVT) and end valid time (EVT).

4 TRANSACTION CONCEPT IN DATA WAREHOUSE

In order to assure the correctness of a DW refreshing process we propose two types of transactions: (1) a fact transaction as well as (2) a schema and dimension transaction.

4.1 Fact transaction

A *fact transaction* is responsible for incrementally refreshing a DW fact tables. This transaction begins after updates at one of the EDSs were committed. The following problems may occur during the refreshing process:

1. Computed delta may be wrong because of an incorrect refreshing a materialized view (cf. Example 1).
2. The communication channels between EDSs and DW may fail during refreshing. As a result, a DW would not be able to finish its refreshing.
3. If EDSs are extensively used, there may be started many refreshing processes. A DW should use some techniques to assure the proper execution of these concurrent processes.
4. A refreshing process can conflict with a user analytical sessions.

The problems mentioned above can be solved by applying to a refreshing process the mechanism of transaction. Our solution is based on the algorithm proposed in (Chen, Chen, Rundensteiner 2000) where the basic functionality of each EDS's wrapper is extended to support data versions generated by EDS's updates. Thus, maintenance queries are answered using appropriate data versions. This mechanism eliminates an incorrect refreshing a materialized view (the first problem). The wrapper can still answer the maintenance queries even if its data source is unavailable (the second problem). The proper execution of concurrent refreshing transactions should be arranged by transaction scheduler, serial or parallel (the third problem). The isolation of refreshing transactions and DW user sessions should be achieved by applying a multiversion concurrency control algorithm where user sessions read an "old" version of data while the "new" version is created at the moment by a refreshing transaction (the fourth problem).

As an extension to the above algorithm we propose to introduce a parallelism inside the fact refreshing transaction. A delta construction process can be easily made parallel – many concurrent sub-transactions inside a main transaction concurrently build separated parts of delta, then a process coordinator joins the partial results into a final delta.

However, this extension imposes advanced transaction models.

4.2 Schema and dimension structure transaction

A schema and dimension structure transaction is responsible for refreshing a DW schema and dimensions structures. The necessity to change a DW schema or/and dimension structures may be caused by two types of events: (1) changes at EDSs and (2) explicit changes made by DW users.

In the first case either an EDS schema is changed (for example – an attribute is dropped from a table) or data updates, which are the sources for DW dimensions data, appeared. A DW is notified of these changes and starts a process to adapt its schema and/or dimension structure. In the second case a DW user (administrator) makes decision to change the schema or the dimension structure. In both cases the administrator decides if the changes are applied to the current DW version or are applied to a new version (a real or an alternative one).

Each change to a DW schema and dimensions structure should also be reflected in the metadata describing the DW.

Changes in a schema or dimension structure can also cause the transformations of fact table data, e.g. removing a specified dimension from DW schema. In a consequence, fact data have to be transformed to a new structure.

We can now define the following steps of schema and dimension structure transaction.

1. Creation of a new DW version, which depends on the administrator's decision.
2. Application of schema and/or dimensions structure changes in the specified version of a DW (the current or a new one).
3. Modification of DW metadata reflecting the changes.
4. Transformation of fact table data.

Some of the above steps can be made parallel, e.g. step 3 and 4. The implementation of operations within a given step can also be executed in parallel, e.g. modifications of separated dimensions, the transformations of fact table data.

We argue that handling DW dynamics can not be based on the concept of standard OLTP transaction since:

1. Standard transaction was designed for great number of very short user interactions with a database. A DW refreshing process and user queries are much longer than standard OLPT activities. Moreover, the concurrency control mechanisms of OLTP systems (for example locking the resources) are inappropriate for a DW since they reduce the degree of concurrency.
2. Standard transaction has a flat structure – it cannot be divided into set of sub-transactions executed concurrently.

For these reasons advanced transaction models should be applied in data warehouses. There are three such models (Barghouti, Kaiser 1994): (1) nested transactions, (2) multilevel transactions, and (3) sagas. A *nested transaction* is a composition of the set of subtransactions, each subtransaction can itself be a nested one. Only the top-level nested transaction is visible to other transactions and it appears as a normal atomic transaction. Sub-transactions inside the top-level transaction are run concurrently and their actions should be synchronized by an internal concurrency control mechanism. A *multilevel transaction* has similar structure to a nested one, but it has predefined number of levels and different concurrency control mechanism can be used for each level. *Saga* is a multilevel transaction limited to only two levels. Partial results of subtransactions inside one saga are visible to other sagas.

Another interesting solution is the possibility of dynamically restructuring running transactions (Barghouti, Kaiser 1994). This model allows changing the execution of transaction as the reaction for modification of user requirements, e.g. splitting one transaction into several new transactions or merging several transactions into a new one.

5 METAMODEL OF MULTIVERSION DW

The advanced models of transactions are currently implemented in our prototype multiversion DW management system.

The metamodel of our MVDW is shown in Figure 1. The diagram presents data dictionary tables used for representing a multiversion schema of a DW as well as mappings between fact tables, dimension tables, and attributes in adjacent DW versions. The *Versions* table stores the information about existing DW versions. Every DW version is composed of fact tables (dictionary tables *Facts* and *Versions_Facts*) and dimensions (dictionary tables *Dimensions* and *Dimensions_Versions*).

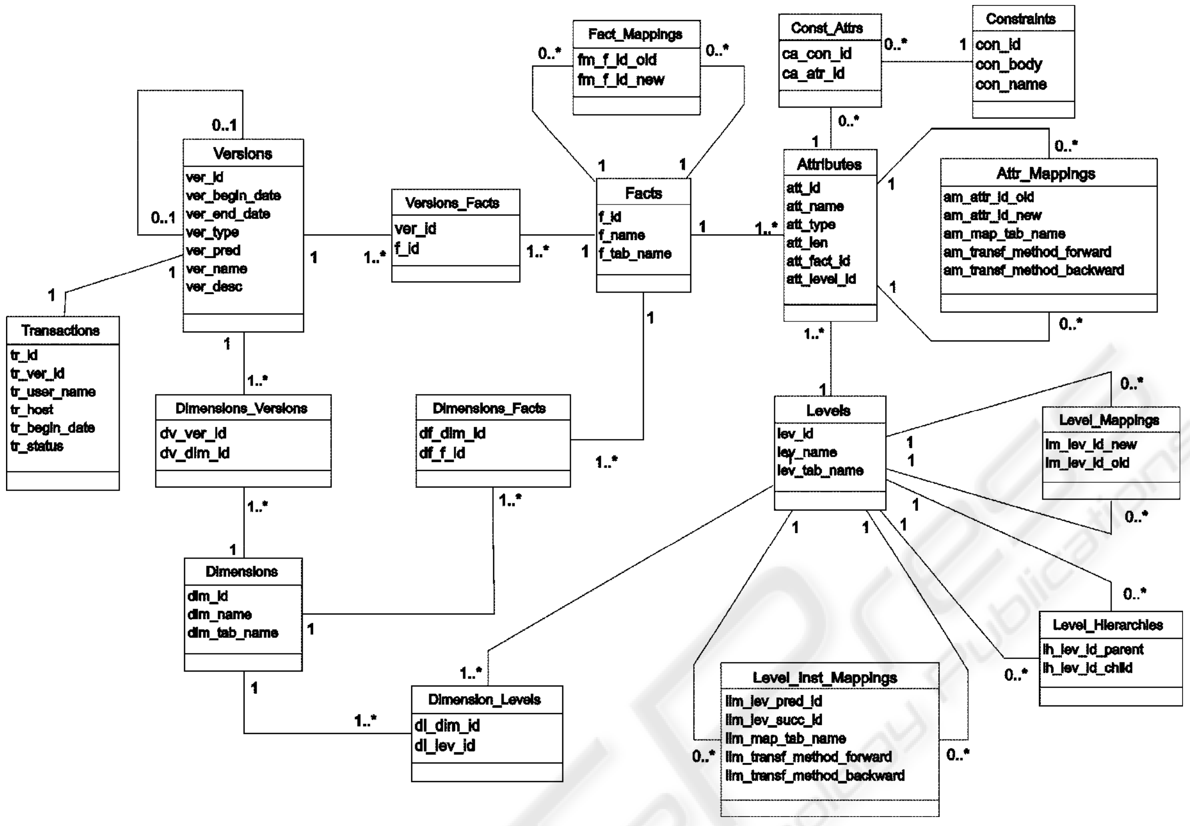

Figure 1: The metamodel of MVDW

Dimensions, in turn, have levels, represented by dictionary tables *Levels* and *Dimension_Levels*. The association between fact tables and dimensions is represented by table *Dimensions_Facts*. The *Fact_Mappings* data dictionary table is used for storing mappings between a given fact table in DW version V_i and the same fact table in version V_j, derived from V_i.

Every fact and level table has the set of its attributes that are stored in *Attributes*. Every attribute may have integrity constraints defined (dictionary tables *Constraints* and *Const_Attrs*). Table *Attr_Mappings* is used for storing mappings between an attribute existing in DW version V_i and the same attribute in adjacent version V_j. The *am_transf_method_forward* is used for storing the name of a transformation program between continuous values of an attribute in a previous version V_i to values of this attribute in version V_j, derived from V_i. Backward transformation program name is stored in *am_transf_method_backward*. Transformation programs are implemented as procedures stored in a database. Discrete values of attributes are mapped in a separate table, whose name is pointed by *map_tab_name*. The schema of this mapping table is composed of three attributes: *attr_id*, *attribute_value_from*, and *attribute_value_to*. The first one stores the identifier of an attribute whose value is mapped. An *attribute_value_from* stores the original value in version V_i, whereas *attribute_value_to* stores the value as required in version V_j.

The *Level_Mappings* table represents mappings between levels in consecutive DW versions. *Level_Inst_Mappings* represents mappings between dimension members in case of structural changes in dimensions, for example, splitting a faculty, merging several shops into one, changing the name of a product. The meaning of *lim_map_tab_name*, *lim_transf_method_forward*, and *lim_transf_method_backward* is the same as respective attributes *am_map_tab_name*, *am_transf_method_forward*, and *am_transf_method_backward* of *Attr_Mappings*.

The *Transactions* table stores the information about transactions used for creating new versions of a DW. Since a DW version may be committed or under derivation, *Transactions* store also the status of every DW version.

6 CONCLUSIONS

In this paper we tackled the problem of synchronizing a DW content and schema with respect to changes in EDSs. We analyzed various anomalies that can appear in a DW during a refreshing process. These anomalies are the results of lacking transaction mechanisms in refreshing.

In our approach, handling the changes in EDSs is done by means of: (1) DW versions and (2) advanced transaction mechanisms. Currently, we are implementing our concepts in a multiversion DW management system based, which is implemented in Java. Data and metadata are stored in an Oracle9i database.

Future work will focus on:

- comparing advanced transaction models (nested, multilevel, and saga) in a DW environment;
- the analysis and development of inter– and intra–version integrity constraints;
- the development of a query language able to span, work on, and compare data from multiple versions of a data warehouse;
- the development of new data structures for efficient storing and indexing multiversion data and their experimental evaluation.

REFERENCES

Agrawal D., El Abbadi A., Singh A., Yurek T., Efficient View Maintenance at Data Warehouse. Proc. of the 1997 ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.

Barghouti N.S., Kaiser G.E., Concurrency Control in Advanced Database Applications. Department of Computer Science, Columbia University, New York, USA, 1994.

Bellahsene, Z. (1998). View Adaptation in Data Warehousing Systems. *Proc. of the DEXA Conf.*

Blaschka, M. Sapia, C., Hofling, G. (1999). On Schema Evolution in Multidimensional Databases. *Proc. of the DaWak99 Conference*, Italy.

Chamoni, P., Stock, S. (1999). Temporal Structures in Data Warehousing. *Proc. of the Data Warehousing and Knowledge Discovery DaWaK*, Italy.

Chen J., Chen S., Rundensteiner E., TxnWrap: A Transactional Approach to Data Warehouse Maintenance. Worcester Polytechnic Institute Technical Report, 2000. (Revisited on February 2002).

Eder, J., Koncilia, C. (2001). Changes of Dimension Data in Temporal Data Warehouses. *Proc. of the DaWak 2001 Conference*, Germany.

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

Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A. (1999a). Maintaining Data Cubes under Dimension Updates. *Proc. of the ICDE Conference*, Australia.

Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A. (1999b). Updating OLAP Dimensions. *Proc. of the DOLAP Conference*.

Kang, H.G., Chung, C.W.: (2002). Exploiting Versions for On–line Data Warehouse Maintenance in MOLAP Servers. *Proc. of the VLDB Conference*, China.

Kulkarni, S., Mohania, M. (1999). Concurrent Maintenance of Views Using Multiple Versions. *Proc. of the Intern. Database Engineering and Application Symposium*.

Mendelzon, A.O., Vaisman, A.A (2000). Temporal Queries in OLAP. *Proc. of the VLDB Conference*, Egypt.

Morzy T., Wrembel R. (2003). Modeling a Multiversion Data Warehouse: a Formal Approach. *Proc. of the 5th International Conference on Enterprise Information Systems (ICEIS)*, Angers, France.

Mostefaoui A., Raynal M., Roy M., Agrawal D., El Abbadi A., The Lord of the Rings: Efficient Maintenance of Views at Data Warehouse. Proc. of 16th International Conference Distributed Computing, DISC 2002, Toulouse, France, October 28-30, 2002.

Quass, D., Widom, J. (1997). On–Line Warehouse View Maintenance. *Proc. of the SIGMOD Conference*.

Yang, J., Widom, J. (2000). Temporal View Self–Maintenance. *Proc. of the EDBT Conference*, Germany.

Rundensteiner E., Koeller A., and Zhang X.: Maintaining Data Warehouses over Changing Information Sources. *Communications of the ACM*, vol. 43, No. 6, 2000.

Zhuge Y., Garcia-Molina H., Hammer J., Widom J., View Maintenance in a Warehousing Environment. Proc. of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.

Zhuge Y., Garcia-Molina H., Wiener J., The Strobe Algorithms for Multi-Source Warehouse Consistency. Proc. of the 4th International Conference on Parallel and Distributed Information Systems, December 18-20, 1996, Miami Beach, Florida, USA.