BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
Vol. 54, No. 1, 2006

# Formal approach to modelling a multiversion data warehouse

B. BĘBEL*, Z. KRÓLIKOWSKI, and R. WREMBEL

Institute of Computing Science, Poznań University of Technology, 2 Piotrowo St., 60-965 Poznań, Poland

**Abstract.** A data warehouse (DW) is a large centralized database that stores data integrated from multiple, usually heterogeneous external data sources (EDSs). DW content is processed by so called On-Line Analytical Processing applications, that analyze business trends, discover anomalies and hidden dependencies between data. These applications are part of decision support systems. EDSs constantly change their content and often change their structures. These changes have to be propagated into a DW, causing its evolution. The propagation of content changes is implemented by means of materialized views. Whereas the propagation of structural changes is mainly based on temporal extensions and schema evolution, that limits the application of these techniques. Our approach to handling the evolution of a DW is based on schema and data versioning. This mechanism is the core of, so called, a multiversion data warehouse. A multiversion DW is composed of the set of its versions. A single DW version is in turn composed of a schema version and the set of data described by this schema version. Every DW version stores a DW state which is valid within a certain time period. In this paper we present: (1) a formal model of a multiversion data warehouse, (2) the set of operators with their formal semantics that support a DW evolution, (3) the impact analysis of the operators on DW data and user analytical queries. The presented formal model was a basis for implementing a multiversion DW prototype system.

**Key words:** schema evolution, data evolution, schema versioning, data versioning, multiversion data warehouse, formal model.

## 1. Introduction

A data warehouse (DW) is a large database (often of terabytes size) that integrates data from various external data sources (EDSs). EDSs are implemented as databases as well as various storage systems (e.g. spreadsheets, legacy systems, flat files, XML files). They store production data collected during normal functioning of an enterprise. These production data are loaded, integrated, augmented with summaries in a DW for the purpose of detail analysis from various perspectives. Data are analyzed by, so called, On-Line Analytical Processing (OLAP) queries aiming at: discovering trends (e.g. sale of products), patterns of behaviour and anomalies (e.g. credit card usage) as well as finding hidden dependencies between data (e.g. market basket analysis, suggested buying). The findings are then applied in real business. Data warehouse and OLAP technologies are important components of decision support systems.

The process of good decision making often requires forecasting future business behaviour, 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 data warehouse 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 a DW structure and content.

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 [1,2]. 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 normal daily work with the support of information systems. On the contrary, schema changes in EDSs are caused by: changes of a 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 [3] and the history of data changes is supported by applying temporal extensions e.g. [4]. Whereas EDSs schema changes are often handled by applying schema evolution, e.g. [5,6] and temporal versioning techniques [7–9]. 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 [10], 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 this paper we contribute by presenting a formal model of a multiversion data warehouse and a formal semantics of operators modifying the structure do a DW schema and dimensions. The presented model and operators were a basis

*e-mail: bartosz.bebel@cs.put.poznan.pl

for implementing a multiversion DW prototype system.

The rest of this paper is organized as follows. Section 2 presents basic definitions concerning a multidimensional data model. Section 3 discusses related approaches to handling dynamics of a DW. Section 4 informally overviews our concept of a mutliversion data warehouse and Section 5 presents its formal model. Formal semantic of operators modifying a MVDW are discussed in Section 6. Finally, Section 7 summarizes the paper.

## 2. Basic definitions

A DW takes advantage of a multidimensional data model [11–13] 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).

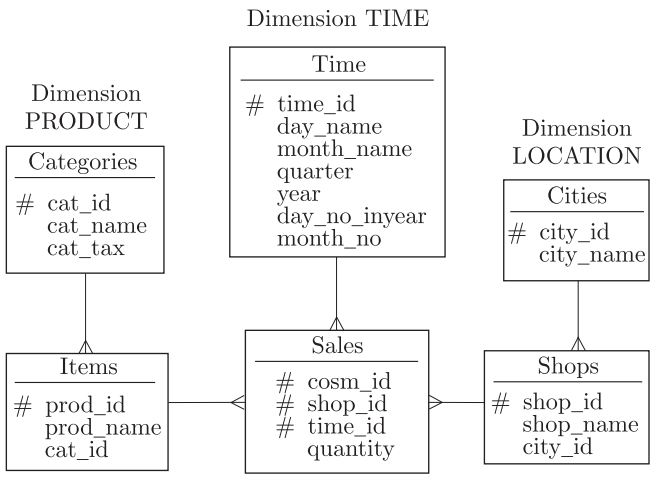


Fig. 1. An example DW schema on sale of products

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. Cities, Shops, Categories, Items, and Time. In a relational implementation, a level is implemented as a table, called a dimension level table.

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: 'Deep t-shirt' and 'Yves Rocher 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: {'Deep t-shirt' $\rightarrow$ 'clothes', 'Yves Rocher shampoo' $\rightarrow$ 'cosmetics'}, where $\rightarrow$ is the hierarchical assignment of a lower level instance to an upper level instance.

## 3. Related work

The support of evolution of schema and data turned up to be required in the applications of object-oriented databases to Computer Aided Design, Engineering, and Manufacturing systems. The problem was intensively investigated and resulted in the development of various approaches and prototypes, [14–18], to list only a few of them. These and many other approaches were proposed for versioning complex objects stored in a database of moderate size. On the contrary, in data warehouse systems objects being versioned have very simple structure (several fact or dimension tables) but the size of a database is much larger. Therefore, the versioning mechanisms mentioned above are not suitable for versioning traditional (relational) data warehouses. The approaches to the management of changes in a DW can be classified as: (1) schema and data evolution: [5,6,19–22], (2) temporal and versioning extensions [2,4,7–9,13,23–29]. The approaches in the first category support only one DW schema and its instance. In a consequence, any structural modification requires data conversions that, in turn, results in the loss of historical DW states. In the approaches from the second category, in [4,7–9,28], changes to the structure of dimension instances are time-stamped in order to create temporal versions. The approaches are suitable for representing historical versions of data, but not schemas. The paper by [13] addresses also the problem of dimension updates and focuses on consistency criteria that every dimension has to fulfill. It gives an overview how the criteria can be applied to a temporal DW only.

In [2,25] 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 [26,27,30] 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, [23,24] 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 [31], 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.

In [29] 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.

## 4. Multiversion data warehouse – overview

A multiversion data warehouse (MVDW) is composed of the ordered 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, changing legislations, opening/closing shops. Real versions are linearly ordered by the time they are valid within. Alternative versions are created for simulation purposes, as part of a 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 graph is the first real DW version.

Every real as well as alternative version is valid within a certain period of time. Version validity is represent by two timestamps, i.e. begin validity time and end validity time, that are associated with every version (cf. [10] for details).

A schema version is composed of several elements, whose informal description is as follows. Each level can have many versions, which belong to the set of levels versions. A level version is assigned to a given dimension version as a part of this dimension's hierarchy. All versions of dimensions form the set of dimensions versions. Facts also can have multiple versions. Versions of a facts are elements of the set of fact table versions. Facts versions are assigned to given dimensions versions. The structure of levels versions as well as fact versions consists of attributes, which form the set of attributes. Attributes are not versioned, i.e. every attribute can be assigned to only one dimension or fact.

A DW instance version is composed of records versions assigned to a level version or to a fact table version. All records versions belong to the set of records versions.

Figure 2 schematically shows relationships between the discussed schema and instance versions elements.

## 5. Multiversion data warehouse – formal model

As stated in Section 4, a multiversion data warehouse, denoted as $\boldsymbol{MVDW}$, is composed of the set of its versions, denoted as $\boldsymbol{DWV}$. A single DW version is composed of a schema version ($DWSV_i$) and an instance version ($DWSV_i$). A schema version includes several schema components, whereas an instance version includes several instance components. Subsequent sections provide formal definitions of the model components.

**5.1. Model components.** Formal definitions of a schema and an instance version use multiple components that are defined in this section.

**Set of versions identifiers.** Each DW version is unambiguously identified by unique identifier $dwv\_id$ that is an element from the set of all DW version identifiers $\boldsymbol{VID} = \{dwv\_id_1, \ldots, dwv\_id_n\}$, where $dwv\_id_i$ is an identifier of a i-th DW version.

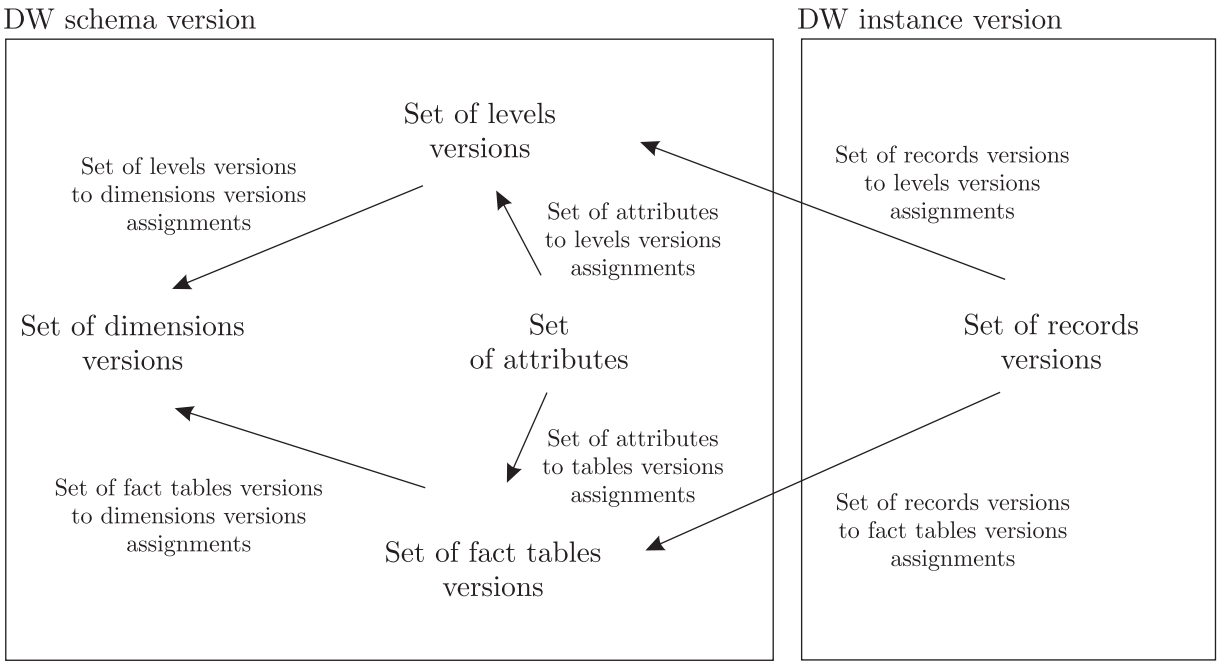


Fig. 2. Relationships among schema and instance version elements

**Multiversion level.** A multiversion level is the element of a dimension structure. It may consist of multiple versions. One level version may exist in several DW versions and in this case the level version is a shared one. A multiversion level is defined as triple $L_i^{MV} = \langle mvl\_id_i, mvl\_name_i, \boldsymbol{LV}_i \rangle$, where: $mvl\_id_i$ is a multiversion level identifier, $mvl\_name_i$ is a multiversion level name, $\boldsymbol{LV}_i = \{LV_1, \ldots, LV_m\}$ is the set of level versions, which is the subset of the set $\boldsymbol{LV}$ of all level versions. A level version is defined as a pair $LV_j = \langle lv\_id_j, \boldsymbol{VID}_j \rangle$, where: $lv\_id_j$ denotes a level version identifier and $\boldsymbol{VID}_j \subseteq \boldsymbol{VID}$ is the set of identifiers of these DW versions, in which version $LV_j$ of level $L_i^{MV}$ exists. $\boldsymbol{L^{MV}} = \{L_1^{MV}, \ldots, L_n^{MV}\}$ is the set of all multiversion levels.

**Multiversion dimension.** A multiversion dimension may consist of multiple versions. One dimension version may exist in several MVDW versions and in this case the dimension version is a shared one. A multiversion dimension is defined as triple $D_i^{MV} = \langle mvd\_id_i, mvd\_name_i, \boldsymbol{DV}_i \rangle$, where: $mvd\_id_i$ is a multiversion dimension identifier, $mvd\_name_i$ is a multiversion dimension name, $\boldsymbol{DV}_i = \{DV_1, \ldots, DV_m\}$ is the set of dimension versions, which is the subset of the set $\boldsymbol{DV}$ of all dimensions versions. A dimension version is defined as quadruple $DV_j = \langle dv\_id_j, \boldsymbol{LV}_j, \rhd_j, \boldsymbol{VID}_j \rangle$, where: $dv\_id_j$ is a dimension version identifier, $\boldsymbol{LV}_j \subseteq \boldsymbol{LV}$ is the set of levels versions, which belong to a dimension version hierarchy; a pair $(\boldsymbol{LV}_j, \rhd_j)$ is a lattice, which describes dimension version hierarchy, with distinguished elements: bottom element (bottom level) and implicit top element (denoted as 'All') (cf. [33]); $\rhd_j \subseteq \boldsymbol{LV} \times \boldsymbol{LV}$ is a relation over the set of levels versions in a dimension hierarchy; its transitive and reflexive closure $\rhd_j^*$ defines a partial order over the set of level versions: if $LV_k$ and $LV_l \in \boldsymbol{LV}_j$ and $LV_l \rhd^* LV_k$, then there is no such $LV_o$, that $LV_l \rhd^* LV_o \rhd^* LV_k$; $\boldsymbol{VID}_j \subseteq \boldsymbol{VID}$ is the set of version identifiers, in which version $DV_j$ of multiversion dimension $D_i^{MV}$ exists. For a given level version $LV_k$, function $SubLevels : \boldsymbol{LV} \rightarrow 2^{\boldsymbol{LV}}$ returns the set of all levels versions that are located in a hierarchy of a dimension version, in a specified DW version, in a subtree whose root is level version $LV_k$. For a given level version $LV_k$ function $SuperLevels : \boldsymbol{LV} \rightarrow 2^{\boldsymbol{LV}}$ returns the set of all levels versions that are direct parents of $LV_k$, in hierarchy of dimension version, in a specified DW version. $\boldsymbol{D^{MV}} = \{D_1^{MV}, \ldots, D_n^{MV}\}$ denotes the set of all multiversion dimensions.

**Multiversion fact.** A multiversion fact may consist of several versions, each of which may be shared by multiple DW versions. A multiversion fact is defined as a triple $F_i^{MV} = \langle mvf\_id_i, mvf\_name_i, \boldsymbol{FV}_i \rangle$, where: $mvf\_id_i$ is a multiversion fact identifier, $mvf\_name_i$ is a multiversion fact name, $\boldsymbol{FV}_i = \{FV_1, \ldots, FV_m\}$ is the set of all versions of a multiversion fact, which is the subset of a set $\boldsymbol{FV}$ of all multiversion facts versions. A multiversion fact version is defined as a pair $FV_j = \langle fv\_id_j, \boldsymbol{VID}_j \rangle$, where: $fv\_id_j$ is an identifier of a fact version, $\boldsymbol{VID}_j \subseteq \boldsymbol{VID}$ is the set of DW versions identifiers, in which a given fact version exists. $\boldsymbol{F^{MV}} = \{F_1^{MV}, \ldots, F_n^{MV}\}$ denotes the set of all multiversion facts.

**Set of attributes.** $\boldsymbol{A} = \{A_1, A_2, \ldots, A_n\}$ is the set of attributes belonging to levels or facts. Each element of this set is defined as a triple $A_i = \langle a\_id_i, a\_name_i, a\_type_i \rangle$ where: $a\_id_i$ is an attribute identifier, $a\_name_i$ is an attribute name, and $a\_type_i$ is an attribute type. The model does not support versioning attributes.

**Attribute to level version assignment.** Function $AtrLevel : \boldsymbol{A} \rightarrow \boldsymbol{LV}$ defines the assignment of an attribute to a level version in a given DW version.

**Attribute to fact version assignment.** Function $AtrFact : \boldsymbol{A} \rightarrow \boldsymbol{FV}$ defines the assignment of an attribute to a fact version in a given DW version.

**Fact version to level version assignment.** Function $FactLevel : \boldsymbol{FV} \rightarrow 2^{\boldsymbol{LV}}$ defines the assignment of a specified fact version to the set of levels versions in a given DW version. For a given fact version $FV_i \in \boldsymbol{FV}$, the function returns the set of base levels versions within dimension hierarchies, connected to fact version $FV_i$.

**Multiversion record.** A multiversion record implements an instance of a multiversion level or a record of a multiversion fact. A Multiversion record is defined as a pair $R_i^{MV} = \langle mvr\_id_i, \boldsymbol{RV}_i \rangle$, where: $mvr\_id_i$ is a multiversion record identifier, $\boldsymbol{RV}_i = \{RV_1, \ldots, RV_n\}$ is the set of record versions that is a subset of set $\boldsymbol{RV}$ of all records versions. A record version is defined as a triple $RV_j = \langle rv\_id_j, rv\_value_j, \boldsymbol{VID}_j \rangle$, where: $rv\_id_j$ is a record version identifier, $rv\_value_j \in \boldsymbol{VAL}$ is the value of a record in the set $\boldsymbol{VAL}$ of values of all multiversion records, $\boldsymbol{VID}_j \subseteq \boldsymbol{VID}$ is the set of DW version identifiers, in which a given record version exists. For a given record version, function $RecValue : \boldsymbol{RV} \rightarrow \boldsymbol{VAL}$ returns its value in a specified DW version. For a given record version $RV_k$ being an instance of level version $LV_l$, function $SuperInst : \boldsymbol{RV} \rightarrow 2^{\boldsymbol{RV}}$ returns the set of record versions, each of which is the instance of direct parent levels of level $LV_l$, to which instance $RV_k$ is classified in a specified DW version. For a given record version $RV_k$ being the instance of level version $LV_l$, function $SubInst : \boldsymbol{RV} \rightarrow 2^{\boldsymbol{RV}}$ returns the set of records versions, being the instances of direct child levels of level $LV_l$, which are classified to instance $RV_k$ in a specified DW version. $\boldsymbol{R}^{MV} = \{R_1^{MV}, \ldots, R_n^{MV}\}$ denotes the set of all multiversion records.

**Record version to level version assignment.** Function $RecLevel : \boldsymbol{RV} \rightarrow \boldsymbol{LV}$ defines the assignment of a record version to a level version in a specified DW version.

**Record version to fact version assignment.** Function $RecFact : \boldsymbol{RV} \rightarrow \boldsymbol{FV}$ defines the assignment of a record version to fact version in a specified DW version.

**5.2. Multiversion data warehouse.** Formally, a MVDW is defined as follows:

$$\boldsymbol{MVDW} = \langle dw\_id, dw\_name, \boldsymbol{DWV}, \blacktriangle, \boldsymbol{CM} \rangle \quad (1)$$

where:

– $dw\_id$ and $dw\_name$ represent MVDW identifier and MVDW name, respectively;

– $\boldsymbol{DWV}$ is the set of data warehouse versions, each of which consists of a schema version and an instance version;
– ▲ is the set of parent-child relationships between DW versions;
– $\boldsymbol{CM}$ is the set of conversions methods, which accomplish transformations between adjacent instance versions; conversion methods are necessary for integrating results of queries addressing several DW versions, cf. [32].

**5.3. Data warehouse version.** A data warehouse version $DWV_i \in \boldsymbol{DWV}$ is formally defined as follows:

$$DWV_i = \langle dwv\_id_i, dwv\_name_i, DWSV_i, DWIV_i \rangle \quad (2)$$

where:

– $dwv\_id_i$ and $dwv\_name_i$ represent a DW version identifier and a DW version name, respectively;
– $DWSV_i$ is a schema version;
– $DWIV_i$ is an instance version.

**5.4. Schema version.** A schema version, denoted as $DWSV_i$, $i = 1, \ldots, n$, describes the structure of data in a DW version $DWV_i$, within DW version validity period, cf. Section 4. Its formal definition is as follows:

$$DWSV_i = \langle\ dwv\_id_i, \boldsymbol{DV}_i, \boldsymbol{LV}_i, \boldsymbol{FV}_i, \boldsymbol{A}_i, AtrLevel_i, AtrFact_i, FactLevel_i \rangle \quad (3)$$

where:

– $dwv\_id_i$ is an identifier of version $DWV_i$ whose structure is described by schema version $DWSV_i$;
– $\boldsymbol{DV}_i \subseteq \boldsymbol{DV}$ is the set of dimension versions which exist in schema version $DWSV_i$;
– $\boldsymbol{LV}_i \subseteq \boldsymbol{LV}$ is the set of level versions, which exist in schema version $DWSV_i$;
– $\boldsymbol{FV}_i \subseteq \boldsymbol{FV}$ is the set of fact versions, which exist in schema version $DWSV_i$;
– $\boldsymbol{A}_i \subseteq \boldsymbol{A}$ is the set of attributes belonging to level and fact versions in schema version $DWSV_i$;
– function $AtrLevel_i : \boldsymbol{A}_i \rightarrow \boldsymbol{LV}_i$ assigns an attribute to a level version in schema version $DWSV_i$;
– function $AtrFact_i : \boldsymbol{A}_i \rightarrow \boldsymbol{FV}_i$ assigns an attribute to fact version in schema version $DWSV_i$;
– function $FactLevel_i : \boldsymbol{FV}_i \rightarrow 2^{\boldsymbol{LV}_i}$ connects a fact version to a level version in schema version $DWSV_i$.

**5.5. Instance version.** An instance version, denoted as $DWIV_i, i = 1, \ldots, n$, represents the set of data consistent with its schema version $DWSV_i$. The formal definition of an instance version is as follows:

$$DWIV_i = \langle dwv\_id_i, \boldsymbol{RV}_i, RecLevel_i, RecFact_i \rangle \quad (4)$$

– $dwv\_id_i$ is the identifier of DW version $DWV_i$ to which instance version $DWIV_i$ belongs;
– $\boldsymbol{RV}_i \subseteq \boldsymbol{RV}$ is the set of records versions, forming instance version $DWIV_i$;
– function $RecLevel_i : \boldsymbol{RV}_i \rightarrow \boldsymbol{LV}_i$ assigns records versions to levels versions in instance version $DWIV_i$;
– function $RecFact_i : \boldsymbol{RV}_i \rightarrow \boldsymbol{FV}_i$ assigns records versions to facts versions in instance version $DWIV_i$.

## 6. MVDW operators

We distinguish two groups of operators that modify the structure of a data warehouse, namely:

– operators that have an impact on a DW schema, further called schema change operators;
– operators that have an impact on the structure of a dimension instance, further called dimension instance structure change operators.

All operators address a particular version of a data warehouse. They are formally described in this section by: their meaning, the set of input arguments, constraints that have to be fulfilled before and after applying a given operator, the set of changes to a DW schema version and its instance, as well as additional comments.

**6.1. Schema change operators.** The following 15 operators describe the evolution of a DW schema.

**Creating a new dimension.**

Meaning: The operator creates a new dimension in the schema of a specified DW version; created dimension has no hierarchy.
Input: DW version $DWV_i$; name $dim\_name$ of a new dimension.
Constraints: There is no dimension in the schema of DW version $DWV_i$ having the same name as $dim\_name$.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. A new multiversion dimension $D^{MV}_{new}$ with name $dim\_name$ is created. The dimension has only one version $DV_{new}$.

Instance changes. None, since the newly created dimension has no levels.

Comments: The operator creates a new dimension in a given schema version; the dimension has no hierarchy hence no instances and no connections to facts. Changes made by the operator do not require adaptation of either DW instance version or analytical queries. In a consequence, there is no need to derive a new DW version before the operator is applied.

**Creating a new level.**

Meaning: The operator creates a new level with a given set of attributes in a specified schema version; the created level does not belong to any dimensions hierarchy.
Input: DW version $DWV_i$, level name $lev\_name$, the set of attributes $\boldsymbol{A}_{new}$ for a newly created level.
Constraints: There is no level in a schema version $DWV_i$ having the same name as $lev\_name$.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. A new multiversion level $L^{MV}_{new}$ with name $lev\_name$ is created, the level has only one version $LV_{new}$; its structure is set up by the set of attributes $\boldsymbol{A}_{new}$; $\boldsymbol{A}_{new}$ is added to the set $\boldsymbol{A}_i$ of version attributes; the attributes from $\boldsymbol{A}_{new}$ are assigned to level version $LV_{new}$ by function $AtrLevel_i$.

Instance changes. None, since the newly created level has no instances.

Comments: Since new level has no instances, changes made by the operator, do not require adaptation either or an instance version or analytical queries. In a consequence there is no need to derive a new DW version before an operator is applied.

**Connecting a level into a dimension hierarchy.**

Meaning: The operator connects a given level into a hierarchy of a given dimension, in a specified DW version. The level being connected can already be connected to other hierarchies of the same dimension (the case of a dimension with multiple hierarchies).

Input: DW version $DWV_i$; version $DV_k$ of a multiversion dimension $D^{MV}_l$ where level version $LV_j$ is to be connected to, position of the level in the hierarchy is described by two sets: the set $\boldsymbol{LV}_{top} \subseteq \boldsymbol{LV}_k$ of direct parent levels of $LV_j$ and the set $\boldsymbol{LV}_{bottom} \subseteq \boldsymbol{LV}_k$ of direct child levels of $LV_j$.

Constraints: None.

Output: DW version $DWV'_i$ with the following changes:

Schema changes. Level $LV_j$ is added to the set $\boldsymbol{LV}_k$ of levels belonging to the hierarchy of dimension version $DV_k$ (unless level $LV_j$ is already the part of another hierarchy in dimension version $DV_k$). A partial order $\rhd_k$ over the set of $\boldsymbol{LV}_k$ is modified in the following way: level version $LV_j$ becomes the child level of all levels from set $\boldsymbol{LV}_{top}$ and becomes the parent level for all levels from set $\boldsymbol{LV}_{bottom}$. If $\boldsymbol{LV}_{bottom} = \emptyset$ (i.e. level $LV_j$ becomes a new bottom level) it is necessary to drop an old assignment of a bottom level and to create an assignment of level $LV_j$ to facts; this assignment is created by function $FactLevel_i$.

Instance changes. The described operator requires classification of all instances of level version $LV_j$ to some instances of levels from set $\boldsymbol{LV}_{top}$ unless $LV_j$ becomes a new top level of a hierarchy; classification of all instances of levels from set $\boldsymbol{LV}_{bottom}$ to some instances of level $LV_j$ unless $LV_j$ becomes a new bottom level of a hierarchy (in both cases, instances that are to be classified should be specified by MVDW administrator); fact records adaptation to a changed dimension hierarchy.

Comments: Changes to a dimension structure done by the described operator, imply changes to a version instance. Adaptations of: (1) instances of a level being connected to a hierarchy, (2) instances of levels which are already in a hierarchy, and (3) records of fact tables may cause some major changes in DW version instance, even potential data loss. Analytical queries may also require modifications. Moreover, the results of queries, obtained from adapted data, may result in incorrect business decisions. This leads us to the conclusion that the derivation of a new DW version is required before applying the discussed operator.

**Disconnecting a level from a dimension.**

Meaning: The operator disconnects a given level from a hierarchy of a given dimension, in a specified DW version. If a level being disconnected belonged to one hierarchy, the level becomes an isolated one.

Input: DW version $DWV_i$; level version $LV_j$ of a multiversion level $L^{MV}_m$ being disconnected from a hierarchy of version $DV_k$ of a multiversion dimension $D^{MV}_l$, $LV_j \in \boldsymbol{LV}_k$.

Constraints: If the level being disconnected is a base level in a dimension hierarchy, it should not be associated with any fact in its schema version.

Output: DW version $DWV'_i$ with the following changes:

Schema changes. If level version $LV_j$ is part of one hierarchy of dimension version $DV_k$, then version $LV_j$ is removed from the set $\boldsymbol{LV}_k$ of level versions in all hierarchies of dimension version $DV_k$; partial order $\rhd_k$ on the set of level versions $\boldsymbol{LV}_k$ is modified, a new order describes dimension hierarchy in which all child levels of $LV_j$ are connected to some parent levels of $LV_j$ (unless $LV_j$ was a base level in a hierarchy). If $LV_j$ was a top level, then all its child levels are connected to implicit element "All".

Instance changes. If disconnected level $LV_j$ was not a base level, then all instances of child levels of $LV_j$ should be reclassified to instances of $LV_j$ parent levels (unless $LV_j$ was a top level of a hierarchy); fact records should also be adapted to a modified dimension hierarchy.

Comments: Changes to a dimension structure done by the operator imply the following changes to instances of dimension version $DV_k$: (1) reclassification of level instances and (2) adaptation of fact data. These changes may lead to data loss and incorrect results or interpretations of analytical queries. Therefore, a new DW version should be derived before applying the described operator.

**Removing a dimension.**

Meaning: The operator removes a given dimension from a schema of a specified DW version.

Input: DW version $DWV_i$, version $DV_j$ of a multiversion dimension $D^{MV}_k$, which is being removed from a schema of $DWV_i$.

Constraints: No hierarchy in the dimension being removed.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. Removing version $DV_j$ of multiversion dimension $D_k^{MV}$ from the set $\boldsymbol{DV}_i$ of all dimensions.

Instance changes. None, since the dimension being removed has no hierarchy, hence it has no data.

Comments: There is no need to derive a new DW version before applying the described operator since the dimension does not have a hierarchy. Consequently, the dimension has neither instances nor associations to fact tables. A dimension removal does not influence either dimension instances or fact records or user queries.

**Removing a level.**

Meaning: The operator removes a given level, that is disconnected from hierarchies of all dimensions, in a specified DW version.
Input: DW version $DWV_i$; version $LV_j \in \boldsymbol{LV}_i$ of multiversion level $L_k^{MV}$, which is to be removed from $DWV_i$.
Constraints: Level version $LV_j$ is not part of a hierarchy in any dimensions in $DWV_i$.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. Removing level version $LV_j$ from the set $\boldsymbol{LV}_k$ of all versions of multiversion level $L_k^{MV}$; removing the assignments of attributes to level $LV_j$ by modifying function $AtrLevel_i$; removing all attributes previously assigned to level $LV_j$ from $\boldsymbol{A}_i$ (the set of attributes).

Instance changes. Removing instances of level version $LV_j$ by modifying function $RecLevel_i$.

Comments: There is no need to derive a new DW version before the described operator is applied. A level being removed is not part of any dimension hierarchy. Consequently, there is no association between a level and a fact. A level removal does not influence either dimension instances or fact records or user queries.

**Creating a new attribute for a level.**

Meaning: The operator creates a new attribute in a schema of a given level in a specified DW version.
Input: DW version $DWV_i$; version $LV_j$ of a multiversion level $L_m^{MV}$; attribute $A_{new}$ defined as $\langle a\_id_{new}, a\_name_{new}, a\_type_{new}\rangle$ being created in the schema of level version $LV_j$.
Constraints: There is no attribute in the schema of level version $LV_j$ having the same name as attribute $a\_name_{new}$ being created.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. Adding a new attribute $A_{new}$ to the set $\boldsymbol{A}_i$ of attributes; creating an assignment of attribute $A_{new}$ to level version $LV_j$ by function $AtrLevel_i$.

Instance changes. Possible adaptation of instances of level version $LV_j$, by assigning values (user defined, default, derived) to a newly created attribute for the instances of level version $LV_j$.

Comments: There is no need to adapt dimension instances after the operator has been applied to a schema version. However, changes introduced by the operator can have impact on results of analytical queries. For example, let us assume that a new attribute was added to a level in a hierarchy of a dimension. The dimension has already assigned a non-empty fact table. Now, users can analyze fact data from the perspective of the newly added attribute. The obtained results can be correct or not, depending on the semantics of this attribute and the way it was created, e.g. the attribute may not correctly describe facts which existed before the attribute was added. To avoid this dilemma it is safe to derive a new DW version before applying the operator.

**Removing an attribute from a level.**

Meaning: The operator removes a given attribute from the set of attributes of a given level, in a specified DW version.
Input: DW version $DWV_i$; version $LV_j$ of multiversion level $L_m^{MV}$; attribute $A_{del}$ being removed from the set of attributes of level version $LV_j$.
Constraints: None.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. Removing the assignment of an attribute $A_{del}$ to level version $LV_j$ by modifying function $AtrLevel_i$; removing attribute $A_{del}$ from the set $\boldsymbol{A}_i$ of attributes.

Instance changes. The modification of values of records versions assigned to level version $LV_j$; the modification consists in deleting values of a removed attribute.

Comments: It is necessary to derive a new DW version before applying the operator. Removing a level attribute causes level data loss. Moreover, analytical queries have to be reformulated.

**Changing the domain of level attribute or fact attribute.**

Meaning: The operator changes the domain of a given attribute in a specified DW version. The attribute is part of a level version or fact version.
Input: DW version $DWV_i$; attribute $A_j \in \boldsymbol{A}_i$ whose domain is being modified to a new one - $a\_type_{new}$.
Constraints: None.
Output: DW version $DWV_i'$ with the following changes:

Schema changes. Attribute $A_j'$ whose domain has been modified to a new domain $a\_type_{new}$.

Instance changes. Adaptation of record versions; the form of adaptation depends on the form of an attribute domain change.

Comments: A decision whether to derive a new DW version before the operator is applied depends on a character of a domain modification. If a domain modification does not require an attribute values adaptation (for example: maximal attribute length is modified from 15 to 20 characters) and it does not cause data loss, the derivation of a new DW version is not necessary. In other cases, a DW administrator may decide to derive a new DW version and apply the operator there, in order to prevent from data loss.

**Creating a new fact.**

Meaning: The operator creates a new fact with its attributes; there is no association between a new fact and any dimension levels.

Input: DW version $DWV_i$; fact name $f\_name$; the set $\boldsymbol{A}_{new} = \{A_1, \ldots, A_k\}$ of attributes of a new fact.

Constraints: No fact exists in schema version $DWV_i$ having the same name as $f\_name$.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. Creating new multiversion fact $F_{new}^{MV}$ and adding it to the set $\boldsymbol{F^{MV}}$ of multiversion facts; the newly created multiversion fact has only one version $FV_{new}$, which is the new element of set $\boldsymbol{FV}_i$ of fact versions in $DWV_i'$; adding the set $\boldsymbol{A}_{new}$ of new fact attributes to the set $\boldsymbol{A}_i$ of attributes in schema version $DWV_i'$; creating assignments between new fact version $FV_{new}$ and attributes from set $\boldsymbol{A}_{new}$, by modifying function $AtrFact_i$.

Instance changes. None, new fact has no instances.

Comments: It is not necessary to derive a new DW version before the operator is applied. Since a new fact has no instances and it is not associated with any dimension. In a consequence, there is no need to perform any adaptation. The modification does not influence analytical queries either.

**Creating a new attribute for a fact.**

Meaning: The operator creates a new measure attribute for a given fact, in a specified DW version.

Input: DW version $DWV_i$; version $FV_j$ of a multiversion fact $F_m^{MV}$; attribute $A_{new}$ defined as a triple $\langle a\_id_{new}, a\_name_{new}, a\_type_{new}\rangle$, that is being added to the set of attributes of fact version $FV_j$.

Constraints: No attribute exists in the set of attributes of fact version $FV_j$ having the same name as $a\_name_{new}$.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. Adding attribute $A_{new}$ to the set $\boldsymbol{A}_i$ of attributes of DW version $DWV_i'$; assigning attribute $A_{new}$ to fact version $FV_j$ by modifying function $AtrFact_i$.

Instance changes. Possible fact version $FV_j$ data adaptation consisting in assigning values (user defined, default, derived) to attribute $A_{new}$.

Comments: Adding a new measure to a fact may cause incorrect results of analytical queries. As an example, let us consider fact table Sales storing product sales data in the first and second quarter of 2004. At the begining of a third quarter of 2004 an attribute ClaimsNumber has been added to the schema of fact table Sales. The registration of customers claims starts from a third quarter of 2004. All values of an attribute ClaimsNumber in fact records, which describe sales in first and second quarters of 2004 were set to 0. Let's assume that a user analyzes the total number of claims in months of 2004. In the period from January until June the number of claims equals to 0. Whereas in July, August 2004 etc. the number of claims appears as greater than 0. In a consequence, a user may conclude that the quality of products sold in the second half of 2004 became worse than products sold in the first half of 2004. The conclusion is evidently false. This example motivates a need for deriving a new DW version before the discussed operator is applied.

**Removing an attribute from a fact table.**

Meaning: The operator removes a measure attribute from a given fact, in a specified DW version.

Input: DW version $DWV_i$; version $FV_j$ of multiversion fact $F_m^{MV}$; attribute $A_{del}$ being removed from the set of attributes of fact version $FV_j$.

Constraints: None.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. Removing an association between attribute $A_{del}$ and fact version $FV_j$ by modifying function $AtrFact_i$; removing attribute $A_{del}$ from the set $\boldsymbol{A}_i$ of attributes in $DWV_i'$.

Instance changes. Adaptation of fact instances, assigned to fact version $FV_j$, by deleting values of the removed attribute.

Comments: Applying the operator to a schema version causes data loss and it requires reformulation of analytical queries. In a consequence, the operator should be applied to a new DW version.

**Creating an association between a fact and a level.**

Meaning: The operator creates an association between a given version of a fact and a given version of a base level in dimension hierarchy, in a specified DW version.

Input: DW version $DWV_i$, version $FV_j$ of multiversion fact $F_k^{MV}$, version $LV_l$ of multiversion level $L_m^{MV}$; $LV_l$ is a base level in the hierarchy of version $DV_n$ of multiversion dimension $D_o^{MV}$.

Constraints: $LV_l$ is a base level in a hierarchy of dimension version $DV_n$.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. Creating an association between fact version $FV_j$ and level version $LV_l$, by modifying function $FactLevel_i$.

Instance changes. Possible adaptation of fact instances assigned to fact version $FV_j$.

Comments: There are two following cases concerning the operator: (1) when a fact table is empty, no adaptation of its instances is required; (2) when a fact table stores data, it is necessary to assign each fact record to its level instance. Sometimes it requires decreasing the level of fact data aggregation. If it is not possible to assign a fact record to a level instance (e.g. a level has no instances or there is no proper level instance to assign to), fact records have to be removed or assigned to a specially created level instance. Since data loss during the process of adaption may happen and the results of user analytical queries can be influenced, a new DW version has to be created before applying the operator.

**Removing an association between a fact and a level.**

Meaning: The operator removes an association between a given fact and a given level in a specified DW version.

Input: DW version $DWV_i$; version $FV_j$ of a multiversion fact $F_k^{MV}$; version $LV_l$ of a multiversion level $L_m^{MV}$, which is a base level in a hierarchy of version $DV_n$ of multiversion dimension $D_o^{MV}$.

Constraints: None.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. Removing an association between fact version $FV_j$ and level version $LV_l$ by modifying function $FactLevel_i$.

Instance changes. Possible adaption of fact instances assigned to fact version $FV_j$.

Comments: Removing an association between a fact and a level requires only a fact table records adaptation (if a fact table is not empty). One of possible adaptations is increasing a level of fact data aggregation. DW users also lose one of the perspectives for data analysis. These reasons motivate a necessity for a new DW version derivation before an operator is applied.

**Removing a fact.**

Meaning: The operator removes a given fact from a specified DW version.

Input: DW version $DWV_i$; version $FV_j$ of multiversion fact $F_k^{MV}$.

Constraints: Version $FV_j$ being removed is not connected to any version of a multiversion levels in $DWV_i$.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. Removing version $FV_j$ from the set $\boldsymbol{FV}_k$ of versions of multiversion fact $F_k^{MV}$; removing assignments between fact version $FV_j$ and its attributes, by modifying function $AtrFact_i$; removing attributes previously assigned to fact version $FV_j$, from the set $\boldsymbol{A}_i$ of attributes in $DWV_i'$.

Instance changes. Removing assignments between fact version $FV_j$ and its instances, by modifying function $RecFact_i$; removing fact instances of $FV_j$.

Comments: It is not necessary to derive a new DW version before the operator is applied since a fact being removed was previously disconnected from all levels.

**6.2. Dimension instance structure change operators.** The following 5 operators describe the evolution of dimensions instances.

**Inserting a new level instance.**

Meaning: The operator inserts a new instance to the set of instances of a given level, in a specified DW version.

Input: DW version $DWV_i$; version $LV_j$ of multiversion level $L_m^{MV}$; value $rv\_value_{new}$ of new level $LV_j$ instance.

Constraints: If a level an instance is inserted into, is not a top level in a dimension hierarchy, then each parent level of $LV_j$ should have an instance, to which an instance of $LV_j$ can be classified.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. None.

Instance changes. Inserting new multiversion record $R_{new}^{MV}$ into the set $\boldsymbol{R}^{MV}$ of multiversion records; the newly created record has one version $RV_{new}$, added to the set $\boldsymbol{RV}_i$ of record versions, in a DW version $DWV_i'$; record value $rv\_value_{new}$ has been added to the set $\boldsymbol{VAL}$ of record values; creating an assignment between record version $RV_{new}$ and a level version $LV_j$, by modifying function $RecLevel_i$. If $LV_j$ is not a top level version in a dimension hierarchy, then a new instance should be classified to explicitly chosen instances of parent levels of $LV_j$. There is no need to adapt fact instances.

Comments: Although there is no need to adapt either level or fact instances, it is necessary to derive a new DW version before applying the operator, as a new level instance can change the results of analytical queries. In a consequence, the obtained results can be wrongly interpreted if users do not have a proper information on changes made to dimension instances.

**Deleting a level instance.**

Meaning: The operator deletes a given instance from the set of instances of a given level, in a specified DW version.

Input: DW version $DWV_i$; record version $RV_{del}$ being deleted; $RV_{del}$ is defined as a triple $\langle rv\_id_{del}, rv\_value_{del}, \boldsymbol{VID}_{del}\rangle$, which implements the instance of version $LV_k$ of a multiversion level $L_m^{MV}$.

Constraints: None.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. None.

Instance changes. Removing an assignment between record version $RV_{del}$ and level version $LV_k$, by modifying function $RecLevel_i$; deleting record version $RV_{del}$ from the set $\boldsymbol{RV}$ of record versions, in DW version $DWV_i'$; deleting record

$rv\_value_{del}$ from the set $\boldsymbol{VAL}$ of records; the following adaptations should take place: (1) instances of child levels of $LV_k$, which were classified to a deleted instance, should be reclassified to other instances of level $LV_k$ or should be deleted; (2) fact instances, which are connected, either directly (when $LV_k$ is a base level) or indirectly (when $LV_k$ is a top level or a level inside a hierarchy), should be connected to other instances of levels or should deleted.

Comments: Deleting a level instance can cause massive adaptations of dimension instances as well as fact instances. Such adaptations can lead to data loss and also can change results of analytical queries. For these reasons, a new DW version should be derived before applying the operator.

**Reclassifying a level instance.**

Meaning: The operator changes the parent of a given child level instance into another parent level instance. Both parent instances of an instance being reclassified (i.e. the one before reclassification and the one after reclassification) are the instances of the same parent level.

Input: DW version $DWV_i$; record version $RV_j$ (the instance of level version $LV_k$ belonging to multiversion level $L_l^{MV}$); $RV_j$ is classified to a record version $RV_{old}$ (the instance of level version $LV_m$); record version $RV_{new}$ (the instance of level version $LV_m$); instance $RV_{new}$ will be the new parent instance of instance $RV_j$; level version $LV_m$ is the parent of $LV_k$.

Constraints: Value $rv\_value_j$ of record version $RV_j$ should allow its reclassification to record version $RV_{new}$.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. None.

Instance changes. Modification of record version value $rv\_value_j$, which classifies record version $RV_j$ to record version $RV_{new}$.

Comments: Instance changes caused by applying the described operator do not lead to any data loss (neither dimension or fact instances). However, they may change the results of analytical queries. If users are not provided with an information on changes in the structure of a dimension instance then interpretations of obtained query results may be wrong. For these reasons, a new DW version should be derived before applying the operator.

**Merging n instances of a level into a new instance.**

Meaning: The operator merges n instances $\{RV_1, \ldots, RV_n\}$ of a given level $LV_j$ into the new instance $RV_{new}$ of the same level. Let $RV_{parent}$ denote the parent instance of $\{RV_1, \ldots, RV_n\}$. If $LV_j$ is not a top level, then: (1) $\{RV_1, \ldots, RV_n\}$ have to be classified to the same instance of their parent level, i.e. $RV_{parent}$, and (2) $RV_{new}$ will be classified also to $RV_{parent}$.

Input: DW version $DWV_i$; the set $\boldsymbol{RV}_{merge} = \{RV_1, \ldots, RV_n\}$ of record versions, i.e. the instances of level version $LV_j$, being merged; the instances are classified to a record version $RV_p$, i.e. the instance of level version $LV_o$; $LV_o$ is the parent level of level version $LV_j$; value $rv\_value_{new}$ of the new instance of level $LV_j$ (an instance, to which instances from set $\boldsymbol{RV}_{merge}$ will be merged).

Constraints: If merged instances are not the instances of a top level, then they have to be classified to the same parent instance. Value $rv\_value_{new}$ of a record, which implements a level instance after merging, should allow its classification to instance $RV_{parent}$. If merged instances are not the instances of a base level, then values of their child instances should allow their classification to a new instance (the one whose value is represented by $rv\_value_{new}$).

Output: DW version $DWV_i'$ with the following changes:

Schema changes. None.

Instance changes. Creating new multiversion record $R_{new}^{MV}$ in the set $\boldsymbol{R^{MV}}$ of multiversion records. A new record has one version $RV_{new}$, which is the element of the set $\boldsymbol{RV}_i$ of record versions in $DWIV_i'$. Record value $rv\_value_{new}$ is added to the set $\boldsymbol{VAL}$ of record values in $DWIV_i'$. Modification of function $RecLevel_i$, which: (1) creates an assignment of record version $RV_{new}$ to level version $LV_j$, and (2) deletes the assignments of record versions in set $\boldsymbol{RV}_{merge}$ to level version $LV_j$. Classification of the instances of $LV_j$ (previously classified to instances from set $\boldsymbol{RV}_{merge}$) to a new instance, implemented by record version $RV_{new}$. Removing from set $\boldsymbol{RV}$ elements of set $\boldsymbol{RV}_{merge}$. Removing values of merged instances, $rv\_value_k, k = 1, \ldots, n$, from set $\boldsymbol{VAL}$. If level version $LV_j$ is a bottom level of dimension hierarchy, it is necessary to adapt fact data, connected to the instances being merged. The form of the adaptation can be either (1) reconnecting fact data to instance $RV_{new}$ after the merge operation or (2) deleting fact data, connected to the merged instances (instances from set $\boldsymbol{RV}_{merge}$).

Comments: The described operator does not require adaptation of dimension instances, however, in some cases, its application can lead to fact table data adaptation. This adaptation can cause data loss. In all cases, applying the operator changes the results of analytical queries. If users are not provided with an information on changes in the structure of a dimension instance then interpretations of obtained query results may be wrong. For these reasons, a new DW version should be derived before applying the operator.

**Splitting a level instance into n new instances.**

Meaning: The operator splits a given level instance $RV_{old}$ into n new instances $\{RV_1, \ldots, RV_n\}$ of the same level $LV_j$,

in a specified DW version. Let $RV_{parent}$ denote the parent instance of $RV_{old}$. If $RV_{old}$ is not an instance of a top level, then $\{RV_1, \ldots, RV_n\}$ have to be classified to $RV_{parent}$. Instances of child levels, which were previously classified to $RV_{old}$, will be classified to one of the instances $\{RV_1, \ldots, RV_n\}$, chosen by a DW administrator.

Input: DW version $DWV_i$; record version $RV_{old}$, i.e. the instance of level version $LV_j$; record version $RV_{old}$ is classified to a record version $RV_{parent}$, i.e. the instance of level version $LV_o$; $LV_o$ is the parent level of $LV_j$; instance $RV_{old}$ will be split into $\{RV_1, \ldots, RV_n\}$, each of which will be implemented by records whose values are given in set $\boldsymbol{VAL}_{new} = \{rv\_value_1, \ldots, rv\_value_n\}$; element $rv\_value_c \in \boldsymbol{VAL}_{new}$ is the value of a record, which will implement an instance to which all instances, previously classified to $RV_{old}$, will be reclassified.

Constraints: None.

Output: DW version $DWV_i'$ with the following changes:

Schema changes. None.

Instance changes. Creating $n$ multiversion records $\{R_1^{MV}, \ldots, R_n^{MV}\}$ in set $\boldsymbol{R^{MV}}$. Every record $\{R_1^{MV}, \ldots, R_n^{MV}\}$ has only one version. The versions form set $\boldsymbol{RV}_{new} = \{RV_1, \ldots, RV_n\}$, which is added to the set $\boldsymbol{RV}_i$ of record versions in $DWV_i'$. Values of records, stored in set $\boldsymbol{VAL}_{new}$, are added to the set $\boldsymbol{VAL}$ of record values. Modification of function $RecLevel_i$, which : (1) assigns record versions from set $\boldsymbol{RV}_{new}$ to level version $LV_j$ as its new instances, and (2) removes the assignment of record version $RV_k$ from $LV_j$. The classification of the instances of child levels $LV_j$ (previously classified to instance $RV_k$) to instance $RV_c \in \boldsymbol{RV}_{new}$. Removing record version $RV_k$ from the set $\boldsymbol{RV}$ of record versions. Removing value $rv\_value_k$ of $RV_k$ from the set $\boldsymbol{VAL}$ of record values. If $LV_j$ is a bottom level in dimension hierarchy, it is necessary to adapt fact data connected to instance $RV_k$ being split. The form of an adaptation can be either (1) reconnecting fact data to the instances after the split operation or (2) deleting fact data, previously connected to a split instance.

Comments: The split operator performs an operation opposite to the merge operator, but the consequences of both operations are the same. Consequently, the application of the split operator should be performed in a new DW version.

## 7. Summary

Handling evolution of data warehouses is currently becoming an important research field [34,35]. On the one hand, research issues in this field are mainly focusing on temporal extensions, that limit their use. On the other hand, commercial data warehouse systems (e.g. Oracle10g, Sybase IQ, MS SQLServer, IBM DB2) are not able to store and manage more than one DW state at the same time.

Our approach to this problem is based on a multiversion data warehouse that is composed of the set of its versions. A DW version represents the structure and content of a DW at a certain time period. A DW version can be used for incorporating structural changes in external data sources as well as changes to a DW schema resulting from changing user requirements. Moreover, DW versions can be applied to creating alternative business scenarios and predicting future. DW versions can also store historical data from certain time periods, and in this case they offer the functionality of temporal data warehouses.

In this paper we presented a formal model of a mutliversion data warehouse. We identified and analyzed possible schema changes and dimension instance changes applicable to a MVDW. Every such a change was in depth analyzed with respect to: its meaning, input and output parameters, the impact on fact and level instances. To the best of our knowledge, it is the first formal approach to describing the evolution of data warehouses. The discussed model was the basis for developing a prototype MVDW system, cf. [32].

## References

[1] J. Roddick, "A survey of schema versioning issues for database systems", *Information and Software Technology* 37 (7), 383–393 (1996).

[2] E. Rundensteiner, A. Koeller, and X. Zhang, "Maintaining data warehouses over changing information sources", *Communications of the ACM* 43 (6), 57–62 (2000).

[3] A. Gupta and I.S. Mumick (eds.), *Materialized Views: Techniques, Implementations, and Applications*, The MIT Press, ISBN 0-262-57122-6, 1999.

[4] P. Chamoni and S. Stock, "Temporal structures in data warehousing", *Proc. DaWaK99*, 353–358 (1999).

[5] M. Blaschka, C. Sapia, and G. Hofling, "On schema evolution in multidimensional databases", *Proc. DaWak99 Conference*, 153–164 (1999).

[6] C.E. Kaas, T.B. Pedersen, and B.D. Rasmussen, "Schema evolution for stars and snowflakes", *Proc. Intern. Conf. on Enterprise Information Systems (ICEIS2004)*, 425–433 (2004).

[7] J. Eder and C. Koncilia, "Changes of dimension data in temporal data warehouses", *Proc. DaWaK Conference*, 284–293 (2001).

[8] J. Eder, C. Koncilia, and T. Morzy, "The COMET metamodel for temporal data warehouses", *Proc.* $14^{th}$ *CAISE02 Conference*, 83–99 (2002).

[9] A.O. Mendelzon and A.A. Vaisman, "Temporal queries in OLAP", *Proc. VLDB Conference*, 242–253 (2000).

[10] B. Bębel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel, "Creation and management of versions in multiversion data warehouse", *Proc. ACM Symposium on Applied Computing (SAC'2004)*, 717–723 (2004).

[11] M. Gyssens and L.V.S. Lakshmanan, "A foundation for multi-dimensional databases", *Proc.* $23^{rd}$ *VLDB Conference*, 106–115 (1997).

[12] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis, *Fundamentals of Data Warehouses*, Springer-Verlag, 2003.

[13] C. Letz, E.T. Henn, and G. Vossen, "Consistency in data ware-

house dimensions", *Proc. Intern. Database Engineering and Applications Symposium (IDEAS'02)*, 224–232 (2002).

[14] R. Agrawal, S. Buroff, N. Gehani, and D. Shasha, "Object versioning in Ode", *Proc. ICDE Conference*, 446–455 (1991).

[15] M. Ahmed-Nacer and J. Estublier, "Schema evolution in software engineering", *Databases – A new Approach in ADELE environment, Computers and Artificial Intelligence* 19, 183–203 (2000).

[16] W. Cellary and G. Jomier, "Consistency of versions in object-oriented databases", *Proc. VLDB Conference*, 432–441 (1990).

[17] S. Gançarski and G. Jomier, "A framework for programming multiversion databases", *Data Knowledge Engineering* 36(1), 29–53 (2001).

[18] W. Kim and H. Chou, "Versions of schema for object-oriented databases", *Proc. VLDB Conference*, 148–159 (1988).

[19] C.A. Hurtado, A.O. Mendelzon, and A.A. Vaisman, "Maintaining data cubes under dimension updates", *Proc. ICDE Conference*, 346–355 (1999).

[20] C.A. Hurtado, A.O. Mendelzon, and A.A Vaisman, "Updating OLAP dimensions", *Proc. DOLAP Conference*, 60–66 (1999).

[21] A. Koeller, E. Rundensteiner, and N. Hachem, "Integrating the rewriting and ranking phases of view synchronization", *Proc. DOLAP98 Workshop*, 60–65 (1998).

[22] A.A. Vaisman, A.O. Mendelzon, W. Ruaro, and S.G. Cymerman, "Supporting dimension updates in an OLAP server", *Proc. CAISE02 Conference*, 67–82 (2002).

[23] M. Body, M. Miquel, Y. Bédard, and A. Tchounikine, "A multidimensional and multiversion structure for OLAP applications", *Proc. DOLAP'2002 Conf.*, 1–6 (2002).

[24] M. Body, M. Miquel, Y. Bédard, and A. Tchounikine, "Handling evolutions in multidimensional structures", *Proc. ICDE 2003 Conf.*, 581 (2003).

[25] J. Chen, S. Chen, and E. Rundensteiner, "A transactional model for data warehouse maintenance", *Proc. ER*, 247–262 (2002).

[26] H.G. Kang and C.W. Chung, "Exploiting versions for on–line data warehouse maintenance in MOLAP servers", *Proc. VLDB Conference*, 742–753 (2002).

[27] S. Kulkarni and M. Mohania, "Concurrent maintenance of views using multiple versions", *Proc. Intern. Database Engineering and Application Symposium*, 254–259 (1999).

[28] L. Schlesinger, A. Bauer, W. Lehner, G. Ediberidze, and M. Gutzman, "Efficienlty synchronizing multidimensional schema data", *Proc. DOLAP*, Atlanta, 69–76 (2001).

[29] A. Balmin, T. Papadimitriou, and Y. Papakonstanitnou, "Hypothetical queries in an OLAP environment", *Proc. VLDB Conf.*, 220–231 (2000).

[30] D. Quass and J. Widom, "On-line warehouse view maintenance", *Proc. SIGMOD Conference*, 393–404 (1997).

[31] M. Golfarelli, J. Lechtenbörger, S. Rizzi, and G. Vossen, "Schema versioning in data warehouses", *ER Workshops 2004* LNCS 3289, 415–428 (2004).

[32] T. Morzy and R. Wrembel, "On querying versions of multiversion data warehouse", *Proc. $7^{th}$ ACM Int. Workshop on Data Warehousing and OLAP (DOLAP 2004)*, Washington, 92–101 (2004).

[33] J. Lechtenbörger and G. Vossen, "Multidimensional normal forms for data warehouse design", *Information Systems* 28(5), 415–434 (2003).

[34] S. Rizzi, "Open problems in data warehousing: 8 years later", *The Keynote Speech at the DMDW Conference*, Berlin, 2003.

[35] The panel discussion at the *DOLAP Conference*, Washington, 2004.