# A Logical Model for Multiversion Data Warehouses

Waqas Ahmed[1,⋆], Esteban Zimányi[1], and Robert Wrembel[2,⋆⋆]

[1] Dept. of Computer & Decision Engineering (CoDE),
Université Libre de Bruxelles, Belgium
{waqas.ahmed,ezimanyi}@ulb.ac.be
[2] Institute of Computing Science,
Poznań University of Technology, Poland
robert.wrembel@cs.put.poznan.pl

**Abstract.** Data warehouse systems integrate data from heterogeneous sources. These sources are autonomous in nature and change independently of a data warehouse. Owing to changes in data sources, the content and the schema of a data warehouse may need to be changed for accurate decision making. Slowly changing dimensions and temporal data warehouses are the available solutions to manage changes in the content of the data warehouse. Multiversion data warehouses are capable of managing changes in the content and the structure simultaneously however, they are relatively complex and not easy to implement. In this paper, we present a logical model of a multiversion data warehouse which is capable of handling schema changes independently of changes in the content. We also introduce a new hybrid table version approach to implement the multiversion data warehouse.

## 1 Introduction

A data warehouse (DW) is a repository of historical, subject-oriented, and heterogeneous data that is integrated from external data sources (EDSs). An inherent feature of EDSs is that they are not static and their schema may change as a result of adaptation of new technologies, changes in the modeled reality, or changes in the business requirements. As a result of schema changes in EDSs, DWs may become obsolete and thus need to be redesigned. Often, after applying schema changes, users demand to preserve the existing content and the schema in a DW. The Multiversion data warehouse (MVDW) is an available solution to manage the schema changes in a DW. This solution is based on the schema versioning approach where every change produces a new DW version and the old content and the schema are also kept available.

The content of a DW changes as a result of periodic loading of new data into it. One particular scenario that requires consideration is changes in the states of

⋆ This research is funded by the Erasmus Mundus Joint Doctorate IT4BI-DC.
⋆⋆ The Polish National Science Center (NCN), grant No. 2011/01/B/ST6/05169.

L. Bellatreche and M.K. Mohania (Eds.): DaWaK 2014, LNCS 8646, pp. 23–34, 2014.

existing dimension members. For analysis purposes, it may be important for the user to keep the history of changes in the states of members. Slowly changing dimensions [8] and temporal data warehouses [7] are the solutions to manage changes in the content of a DW. These solutions maintain the content history by associating timestamps with the values. Though these solutions partially solve the problem of managing DW content changes, they are unable to separate different DW states that describe different real world scenarios.

Most of the available MVDW proposals try to solve the issues of content and schema changes simultaneously which makes it complicated to understand and implement these proposals. Some solutions for MVDW present metamodels to maintain metadata supporting the life cycle of a DW. These metamodels store DW versions in separate physical structures. Creation and maintenance of these separate data structures makes a MVDW complex and may also negatively impact the query performance. If the user only needs to manage the schema changes in a DW then a simplified model can serve this purpose.

We envision a MVDW in which content and structure change functionality can be implemented as independent plug-ins. In this paper we propose a model of the MVDW that supports structural changes in dimensions and facts. The proposed model is relatively simple and helps to identify the desired features of schema versioning. We also introduce a new hybrid table version (HTV) approach to implement the MVDW. This paper is organized as follows: Section 2 discusses the approaches related to handling the evolution of DWs. Section 3 presents a running example and defines the requirements for the MVDW. Section 4 presents a model of the MVDW while Sect. 5 proposes an approach to implement it. Finally, Sect. 6 concludes the paper by providing a summary and considerations for future research.

## 2 Related Work

In [7], the proposals for managing DW evolutions are classified into three broad categories: schema modification, schema evolution, and schema versioning. The systems that allow *schema modification* support changes in a data source schema but as a result of these changes, the existing data may become unavailable. *Schema evolution* is supported when a system has capability of accommodating schema changes while preserving existing data. *Schema versioning* is a mechanism through which systems store data using multiple schema versions.

Most of the research related to schema versioning deals with the issues of managing schema and content changes at the same time and therefore, presents solutions to manage schema versioning for temporal databases. Brahmia et al. [3] presented one such schema version management mechanism for multi-temporal databases. The proposed approach however, does not provide a generic model to manage schema versioning and is only specific to relational databases. The model presented in [5] also supports changes in both the content and structure of data. This model supports structural changes to the dimension members only and does not explain how to map the members from one version to another.

In [1], the authors presented a formal model of a MVDW. This model supports the real schema versions, which represent the changes in the real world, and also provides the capability of deriving alternative schema versions. The alternative schema versions can be used for what-if analysis . In [12], the authors presented a logical model for the implementation of a MVDW and discussed various constraints to maintain data integrity across DW versions. They used the work presented in [13] to query data from multiple versions of a DW. [10] is a prototype implementation which manages schema versions by creating graph based metamodels. An augmented schema [6] is also created to perform data transformations among different schema versions.

The aforementioned approaches to MVDWs track the history of content and schema changes and can query multiple DW versions but they maintain DW versions in separate structures. The creation and maintenance of these structures is relatively complex. Further, the approaches to handling DW evolution either manage changes in the content only [9], changes in the schema only [2], or changes in the content and schema simultaneously [12]. Data warehouse versioning approaches support both changes in the content and the schema at the same time but none of the existing versioning approaches deals with issues of schema and content evolution independently of each other.

Three approaches may be used to convert a database structure after applying schema changes: single table version (STV), multiple table version (MTV), and partial multiple table version (PMTV) [11]. A disadvantage of the STV approach is the null space overhead: null values are introduced as a result of both the addition and the deletion of attributes. There is no null space overhead when using the MTV approach as the deleted attributes are dropped from the new version. However, multiple table versions increase the maintenance overhead. Also, running a query that spans multiple table versions requires data adaptations and table joins which negatively affect the query performance. The PMTV approach also requires joins to construct the complete schema of the table. Joins are costly operations and may have a negative impact on the system performance.

## 3 Multiversion Data Warehouses

A multiversion data warehouse is a collection of its versions. Each *DW version* is composed of a schema version and an instance version. A *schema version* describes the structure of the data within a time period. The *instance version* represents the data that is stored using a particular schema version. We assume that at a given time instant, only one DW version is current and used to store data. A new version can be derived by applying changes to the current version only. To represent the period for which a version was used to store data, each DW version is assigned a closed-open time interval [9], represented by begin application time (BAT) and end application time (EAT). The EAT for the current version is set to *UC* (until-changed). It is possible to create alternative schema versions [12] using the model presented in Sect. 4 but for simplicity's sake, we do not consider the branching versioning model. Figure 1 shows an example of multiple DW versions.

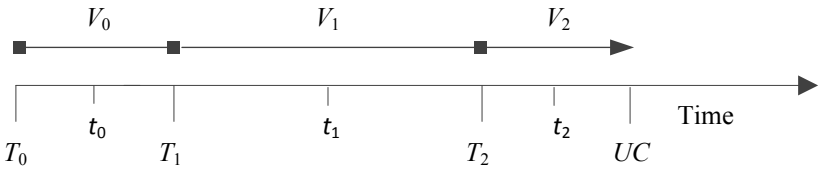
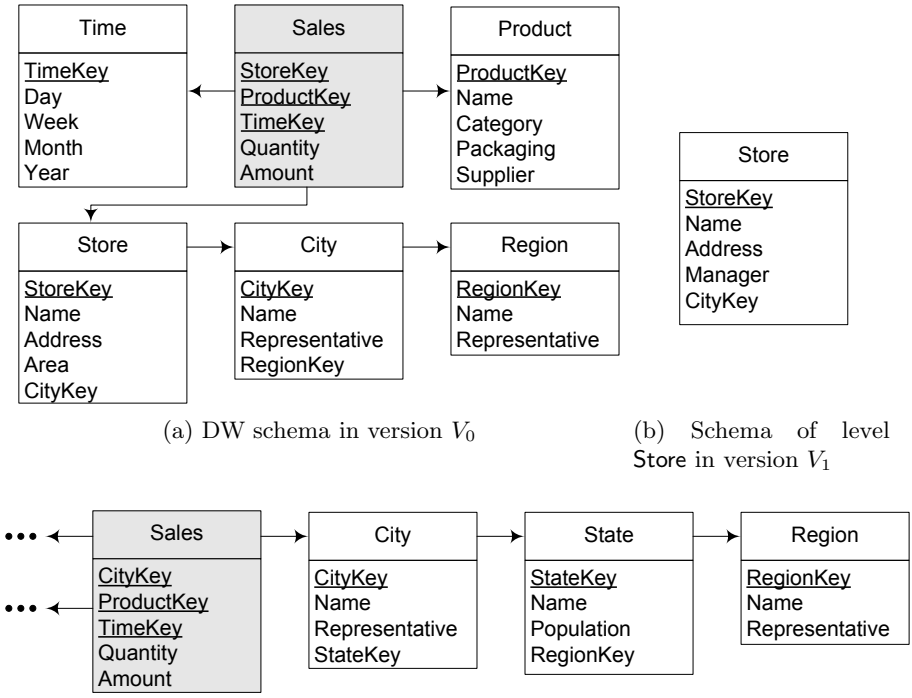



**Fig. 1.** Multiple versions of a data warehouse

Suppose that the initial version $V_0$ was created at time $T_0$. The BAT and EAT for $V_0$ are set to $T_0$ and $UC$, respectively. Figure 2a shows the DW schema in version $V_0$ which contains three dimensions: Time, Product, and Geography. The Geography dimension has a hierarchy, which consists of three levels: Store, City, and Region. Fact Sales relates these dimensions and contains measures Quantity and Amount. The granularity of dimension Geography is at level Store. Consider that the following schema changes were applied to the DW. (1) At time $T_1$, attribute Manager was added to and attribute Area was deleted from level Store. As a result, version $V_1[T_1, UC)$ was created and the EAT of $V_0$ was changed from $UC$ to $T_1$. Figure 2b shows the schema of level Store in $V_1$. (2) At time $T_2$, level Store was deleted and new level State was added between levels City and Region. This schema modification changed the granularity of dimension Geography and thenceforth all fact members were assigned to a city. Moreover, level City rolled-up to level State. Since $V_1$ was the current version at that time, version $V_2[T_2, UC)$ was derived from $V_1$ and the EAT of $V_1$ was changed from $UC$ to $T1$. Figure 2c shows the schema of $V_2$.

Changes in dimensions and/or facts create a new DW version. The possible changes to a dimension include (1) adding a new attribute to a level, (2) deleting an attribute from a level, (3) changing the domain of an attribute, (4) adding a level to a hierarchy, and (5) deleting a level from a hierarchy. The schema of a fact changes when (1) a new dimension is added, (2) a dimension is deleted, (3) a measure is added, (4) a measure is deleted, and (5) the domain of a measure is changed. In certain cases, a schema change in the dimension also requires changes in the fact. For example, adding a new level into a hierarchy at the lowest granularity will also require a new attribute to be added to the fact which will link the new level to the fact. Similarly, in case of removing the lowest level from a hierarchy, an attribute is removed from the fact to unlink the deleted level and a new attribute is added into it to link the next level of the hierarchy. Thus, effects of adding and deleting the lowest level of a hierarchy on the DW schema are similar to adding and deleting a new dimension to the DW.

It is worth mentioning that since our versioning model deals with the structural changes and not with the temporal evolution of facts and dimension members, at any instant $t$, all facts and dimension members in the MVDW are valid. To summarize the requirements of a multiversion data warehouse, we can say that (1) *in a MVDW, new data is loaded using current version only*; (2) *a MVDW must retain all the data loaded into it throughout its lifespan*; and (3) *all the data stored in the MVDW must be viewable using any MVDW schema version.*




(a) DW schema in version $V_0$

(b) Schema of level Store in version $V_1$

(c) DW schema in version $V_2$

**Fig. 2.** Multiple schema versions of the running example

The first requirement is straightforward. We elaborate the second and third requirements with the help of examples. The initial state of level Store in version $V_0$ at an instant $t_0$ in $[T_0, T_1)$ is shown in Fig. 3a. As a result of the first schema change, version $V_1$ is derived from $V_0$. Figure 3b shows the schema of level Store in $V_1$. The shaded column represents the deleted attribute. The second requirement is that when deriving version $V_1$ from $V_0$, for the existing store members, the DW must retain the values of the deleted attribute Area. Suppose that the user decides to add a new store member s4 at an instant $t_1$ in $[T_1,T_2)$. The third requirement states that this newly added member must also be available in version $V_0$. So, when accessing the store members using version $V_0$, the user should be able to access all the members stored in $V_0$ and $V_1$. As attribute Manager does not exist in $V_0$, it is not available in $V_0$ at $t_1$. This situation is shown in Fig. 3c where shaded and crossed out cells represent the unavailable attribute. Similarly, attribute Area does not exist in $V_1$ therefore, the value of attribute Area for s4 is not available in $V_0$.

As a result of the second schema change, version $V_2$ is derived from $V_1$. In $V_2$ both the dimensions and the fact are affected. Figure 3d shows the state of fact Sales in $V_0$. After second schema change, the fact members are assigned to level City and that is why attribute StoreKey is deleted and CityKey is added to the new version of fact Sales. Since in previous versions level Store rolled-up to level

| Store Key | Name | Address | Area | City Key |
|---|---|---|---|---|
| s1 | Store1 | ABC | 20 | c1 |
| s2 | Store2 | DEF | 30 | c1 |
| s3 | Store3 | HIJ | 50 | c2 |

(a) Store at $t_0$ in version $V_0$

| Store Key | Name | Address | ~~Area~~ | Manager | City Key |
|---|---|---|---|---|---|
| s1 | Store1 | ABC | ~~20~~ | | c1 |
| s2 | Store2 | DEF | ~~30~~ | | c1 |
| s3 | Store3 | HIJ | ~~50~~ | | c2 |
| s4 | Store4 | KLM | | John | c2 |

(b) Store at $t_1$ in version $V_1$

| Store Key | Name | Address | Area | ~~Manager~~ | City Key |
|---|---|---|---|---|---|
| s1 | Store1 | ABC | 20 | | c1 |
| s2 | Store2 | DEF | 30 | | c1 |
| s3 | Store3 | HIJ | 50 | | c2 |
| s4 | Store4 | KLM | | ~~John~~ | c2 |

(c) Store at $t_1$ in version $V_0$

| Store Key | Product Key | Time Key | Quantity | Amount |
|---|---|---|---|---|
| s1 | p1 | t1 | 5 | 20 |
| s2 | p1 | t1 | 3 | 15 |
| s3 | p2 | t2 | 2 | 18 |
| s4 | p2 | t3 | 3 | 9 |

(d) Sales at $t_0$ in version $V_0$

| ~~Store Key~~ | City Key | Product Key | Time Key | Quantity | Amount |
|---|---|---|---|---|---|
| ~~s1~~ | | ~~p1~~ | ~~t1~~ | ~~5~~ | ~~20~~ |
| ~~s2~~ | | ~~p1~~ | ~~t1~~ | ~~3~~ | ~~15~~ |
| | c1 | p1 | t1 | 8 | 35 |
| ~~s3~~ | c2 | p2 | t2 | 2 | 18 |
| ~~s4~~ | c2 | p2 | t3 | 3 | 9 |
| | c3 | p1 | t4 | 2 | 25 |

(e) Sales at $t_2$ in version $V_2$

| Store Key | ~~City Key~~ | Product Key | Time Key | Quantity | Amount |
|---|---|---|---|---|---|
| s1 | | p1 | t1 | 5 | 20 |
| s2 | | p1 | t1 | 3 | 15 |
| | ~~c1~~ | ~~p1~~ | ~~t1~~ | ~~8~~ | ~~35~~ |
| s3 | ~~c2~~ | p2 | t2 | 2 | 18 |
| s4 | ~~c2~~ | p2 | t3 | 3 | 9 |
| | ~~c3~~ | p1 | t4 | 2 | 25 |

(f) Sales at $t_2$ in version $V_0$

| City Key | Name | Representative | RegionKey |
|---|---|---|---|
| c1 | Brussels | John | r1 |
| c2 | Liège | Doe | r2 |
| c3 | Charleroi | Ahmed | r2 |

(g) City at $t_0$ in version $V_0$

| City Key | Name | Representative | State Key | ~~Region Key~~ |
|---|---|---|---|---|
| c1 | Brussels | John | e1 | ~~r1~~ |
| c2 | Liège | Doe | e2 | ~~r2~~ |
| c3 | Charleroi | Ahmed | e3 | ~~r2~~ |

(h) City at $t_2$ in version $V_2$

| City Key | Name | Representative | ~~State Key~~ | Region Key |
|---|---|---|---|---|
| c1 | Brussels | John | ~~e1~~ | r1 |
| c2 | Liège | Doe | ~~e2~~ | r2 |
| c3 | Charleroi | Ahmed | ~~e3~~ | r2 |

(i) City at $t_2$ in version $V_0$

**Fig. 3.** Contents of the DW in multiple versions of the running example

City, it is possible to obtain the values of CityKey for the existing fact members. While doing so, the fact members belonging to the stores that are located in the same city should be combined provided that the other key attributes are the same. One such example is shown in Fig. 3e where the first two fact members belong to stores s1 and s2. These stores are located in the same city c1 and the values of the other dimensions are the same, that is why the fact members

associated to these stores are combined and represented as a single member in the new version of fact Sales. If the user tries to access fact Sales in $V_0$ at an instant $t_2$ in $[T_1, T_2)$, the value of StoreKey for the last fact member will not be available as this attribute is not present in the new version of fact Sales. This situation is depicted in Fig. 3f.

In $V_2$, a new level State is also added between levels City and Region. Figure 3g shows level City in $V_0$. Figure 3h shows that the effect of adding a level into the DW schema is similar to adding attribute StateKey to level City and deleting attribute RegionKey from it. If the members of level City are accessed in $V_0$ at time $t_2$ then the information about the regions will not be available. Figure 3i depicts this scenario.

Suppose at some point the user decides to add a new dimension Supplier to the DW. This addition will require a new version of fact Sales because the new facts must be linked to Supplier as well. It is worth mentioning that the supplier information will not be available for the existing facts and they will roll-up to unknown supplier.

## 4 A Multiversion Data Warehouse Model

In this section, we first introduce the formal definition of a data warehouse model and then we extend the definition for a multiversion data warehouse.

**Definition 1 (Multidimensional schema).** Multidimensional schema $\mathcal{S}$ has a name and is composed of (1) the set of dimensions $\mathcal{D} = \{D_1, \ldots, D_n\}$ and (2) the set of facts $\mathcal{F} = \{F_1, \ldots, F_n\}$.

Dimension $D_i \in \mathcal{D}$, $i = 1, \ldots, n$, has a name and is composed of (1) the set of levels $\mathcal{L} = \{L_1, \ldots, L_n, All\}$, (2) aggregation relation $\mathcal{R}$, and (3) the set of hierarchies $\mathcal{H} = \{H_1, \ldots, H_n\}$. Level $L_1 \in \mathcal{L}$ is called the *base level* of the dimension and every dimension has a unique level *All*. The dimension names are unique in $\mathcal{D}$.

Level $L_j \in \mathcal{L}$, $j = 1, \ldots, n$, is defined by its schema $L_j(A_1 : T_1, \ldots, A_n : T_n)$, where $L_j$ is the level name and it is unique in $\mathcal{L}$. Each attribute $A_k$, $k = 1, \ldots, n$, is defined over domain $T_k$ and attribute name $A_k$ is unique in $L_j$. Level $All \in D_i$ does not have any attribute.

Aggregation relation $\mathcal{R}$ is a partial order binary relation on $\mathcal{L} \in D_i, i = 1, \ldots, n$, and contains ordered pairs of form $\langle L_p, L_q \rangle$, where $L_p$ and $L_q$ are levels belonging to $\mathcal{L}$. $\mathcal{R}^*$ denotes the transitive closure of $\mathcal{R}$ such that if $\langle L_1, L_2 \rangle$ and $\langle L_2, L_3 \rangle$ also belong to $\mathcal{R}$, then $\langle L_1, L_3 \rangle$ belongs to $\mathcal{R}^*$. Any level $L_j \in \mathcal{L}$ is, directly or transitively, reachable in $\mathcal{R}^*$ from the base level and any level $L_j \in \mathcal{L}$ reaches in $\mathcal{R}^*$, directly or transitively, top level *All*.

Hierarchy $H_m \in \mathcal{H} \subseteq \mathcal{R}^*$, $m = 1, \ldots, n$, and has a unique name. Each hierarchy $H_m \in \mathcal{H}$ begins from the base level and has top level *All*.

Fact $F_i \in \mathcal{F}$, $i = 1, \ldots, n$, is defined by its schema $F_i(R_1 : L_1, \ldots, R_m : L_m, M_1 : T_1, \ldots, M_n : T_n)$, where $R_s$, $s = 1, \ldots, m$, is a role name, $L_s$ is a base level of a dimension $D_i \in \mathcal{D}$ and each measure $M_t$, $t = 1, \ldots, n$, is defined over domain $T_t$. Role name $R_s$ and measure name $M_t$ are unique in $F_i$. □

**Definition 2 (Multidimensional instance).** Multidimensional instance $I$ is composed of dimension instance and fact instance.

An instance of dimension $D_i \in \mathcal{D}$ is as follows: For each level $L_j \in D_i$, the set of members $M_{L_j} = \{m_1, \ldots, m_n\}$ where member $m_k \in M_{L_j}$, $k = 1, \ldots, n$, is uniquely identifiable. Level *All* has a special member *all*. For each $L_j \in D_i$ with schema $L_j(A_1{:}T_1, \ldots, A_n{:}T_n)$, a subset of $M_{L_j} \times T_1 \times \ldots \times T_n$. For each pair of level names $\langle L_p, L_q \rangle$ in $\mathcal{R} \in D_i$, a partial function $Roll\_up_{L_p}^{L_q}$ from $M_{L_p}$ to $M_{L_q}$.

An instance of fact $F_i$, which is defined by its schema $F_i(R_1 : L_1, \ldots, R_m : L_m, M_1{:}T_1, \ldots, M_n{:}T_n)$, is a subset of $M_{L_1} \times \ldots \times M_{L_m} \times T_1 \times \ldots \times T_n$. □

**Definition 3 (Multiversion multidimensional schema).** *Multiversion multidimensional schema $\mathcal{S}_{mv}$ has a name and is composed of (1) the set of multiversion dimensions $\mathcal{D}^v = \{\mathcal{D}_1^v, \ldots, \mathcal{D}_n^v\}$, (2) the set of multiversion facts $\mathcal{F}^v = \{\mathcal{F}_1^v, \ldots, \mathcal{F}_n^v\}$, and (3) the set of schema versions $\mathcal{S}^v = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$.*

*Multiversion dimension $\mathcal{D}_i^v \in \mathcal{D}^v, i = 0, \ldots, n$, defines the set $\{D_i^{v_1}, \ldots, D_i^{v_n}\}$ of versions of dimension $D_i$. Dimension version $D_i^{v_j} \in \mathcal{D}_i^v, j = 0, \ldots, n$, is a dimension as defined in Def. 1. The dimension names for all $D_i^{v_j} \in \mathcal{D}_i^v$ are the same.*

*Multiversion fact $\mathcal{F}_m^v \in \mathcal{F}^v, m = 1, \ldots, n$, defines the set $\{F_m^{v_1}, \ldots, F_m^{v_n}\}$ of versions of fact $F_m$. Fact version $F_m^{v_j}, j = 0, \ldots, n$, is a fact as defined in Def. 1. The fact names for all $F_m^{v_j} \in \mathcal{F}_m^v$ are the same.*

*Schema version $\mathcal{S}_l \in \mathcal{S}^v$, $l = 0, \ldots, n$, has an associated time interval $T_l = [B_l, E_l)$ and is a multidimensional schema as defined in Def. 1, that is, it is composed of a set of dimensions $\mathcal{D} = \{D_1, \ldots, D_n\}$ and a set of facts $\mathcal{F} = \{F_1, \ldots, F_n\}$, where each $D_i \in \mathcal{D}$ is a dimension version $D_i^{v_j} \in \mathcal{D}_i^v$ and each $F_m \in \mathcal{F}$ is a fact version $F_m^{v_j} \in \mathcal{F}_m^v$. Only one version of dimension $\mathcal{D}_i^v$ and fact $\mathcal{F}_m^v$ can exist in $\mathcal{D}$ and in $\mathcal{F}$, respectively. The time intervals associated to all schema versions in $\mathcal{S}^v$ are disjoint, contiguous, and their union cover the time interval since the creation of the first version until now.* □

*Example 1.* The initial schema of the multiversion data warehouse in Fig. 2a can be represented as follows:

$\mathcal{D}^v = \{\mathcal{D}_G^v, \mathcal{D}_T^v, \mathcal{D}_P^v\}$, $\mathcal{F}^v = \{\mathcal{F}_S^v\}$, $\mathcal{S}^v = \{\mathcal{S}_0\}$, where $G$, $T$, $P$, and $S$ denote Geography, Time, Product, and Sales, respectively.

$\mathcal{D}_G^v = \{\mathcal{D}_G^{v0}\}$, $\mathcal{D}_T^v = \{\mathcal{D}_T^{v0}\}$, $\mathcal{D}_P^v = \{\mathcal{D}_P^{v0}\}$, $\mathcal{F}_S^v = \{\mathcal{F}_S^{v0}\}$, and $\mathcal{S}_0 = \{\{\mathcal{D}_G^{v0}, \mathcal{D}_T^{v0}, \mathcal{D}_P^{v0}\}, \{\mathcal{F}_S^{v0}\}\}$. For brevity, we omit the schema definitions of the levels and the fact.

As a result of the first schema change, a new version of the Geography dimension is derived from the previous version and the other dimensions and the fact remain unchanged. Thus, the MVDW schema is modified as follows: $\mathcal{D}_G^v = \{\mathcal{D}_G^{v0}, \mathcal{D}_G^{v1}\}$, $\mathcal{S}^v = \{\mathcal{S}_0, \mathcal{S}_1\}$, $\mathcal{S}_1 = \{\{\mathcal{D}_G^{v1}, \mathcal{D}_T^{v0}, \mathcal{D}_P^{v0}\}, \{\mathcal{F}_S^{v0}\}\}$.

Finally, new versions of the dimension Geography and fact Sales are derived as a result of the second schema change. The resulting schema of the MVDW is modified as follows: $\mathcal{D}_G^v = \{\mathcal{D}_G^{v0}, \mathcal{D}_G^{v1}, \mathcal{D}_G^{v2}\}$, $\mathcal{F}_S^v = \{\mathcal{F}_S^{v0}, \mathcal{F}_S^{v1}\}$, $\mathcal{S}^v = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2\}$, $\mathcal{S}_2 = \{\{\mathcal{D}_G^{v2}, \mathcal{D}_T^{v0}, \mathcal{D}_P^{v0}\}, \{\mathcal{F}_S^{v1}\}\}$. □

**Definition 4 (Multiversion multidimensional global schema).** The multiversion multidimensional global schema is a multidimensional schema as defined in Def. 1 and it is constructed as follows:

In the global schema of dimension $D_i \in \mathcal{D}$s, $\mathcal{L} \in D_i$ is the union of all the levels existing in all versions of $D_i$; the schema of level $L_i \in \mathcal{L}$ is the union of all the attributes of $L_i$ from all versions of $D_i$ in which $L_i$ is present; aggregation relation $\mathcal{R}$ is the union of the aggregation relations from all version of $D_i$, and $\mathcal{H} \in D_i$ is empty. Since the global schema is for system use only, there is no need to maintain hierarchies in it.

The global schema of fact $F_i \in \mathcal{F}$ consists of the set of base levels $\mathcal{B}$ and the set of measures $\mathcal{C}$, where $\mathcal{B}$ is the union of all the base levels in all versions of $F_i$ and $C$ is the union of all the measures in all versions of $F_i$. □

The global schema, defined in Def. 4, is a traditional multidimensional schema and its instance is obtained by using Def. 2. A multiversion multidimensional instance is actually an instance of the global schema. Figure 3b, including the shaded column Area, shows the global instance of level Store. The MVDW also contains a transformation function $\mathcal{T}(\mathcal{S}_i)$ which transforms the global instance into the instance of schema $\mathcal{S}_i \in \mathcal{S}^v$, as defined in Def. 3. This transformation function can be implemented as view definitions. Figure 6b shows how the global instances of Store can be transformed into the instance of Store in version $V_1$.

## 5 Implementation of the Multiversion DW

We discuss next how the single table version (STV) and the multiple table version (MTV) approaches can be used to implement the MVDW. Then, we present a new hybrid table versioning approach to implement the MVDW.

In the STV approach, the newly added attributes are appended to the existing ones and the deleted attributes are not dropped from the table. A default or null value is stored for the deleted or the unavailable attributes. Figure 4 shows the effect of the schema changes on the relational implementation of a DW that uses the STV approach. Figure 4a shows the state of the DW after the first schema change where attribute Manager is added and Area is deleted from table Store. Records s1, s2, and s3 have null values for attribute Manager because its value is unknown for these records. As attribute Area has been deleted, all newly added records such as s4, will have null values for it. These null values may incur a space overhead in case of huge amount of data. Some DBMSs partially resolve the issue of null space overhead by offering specific features but the implementation of these features has its own limitations[1].

In the MTV approach, each change in the schema of a table produces a new version of the table. Figure 5a shows the new version of table Store which is created as a result of the first schema change. This version includes the newly added attribute Manager and excludes the deleted attribute Area. The new version of table Store results in a new version of table Sales because Sales uses attribute

[1] http://technet.microsoft.com/en-us/library/cc280604.aspx

| Store Key | Name | Address | Area | City | Manager |
|---|---|---|---|---|---|
| s1 | Store1 | ABC | 20 | c1 | null |
| s2 | Store2 | DEF | 30 | c1 | null |
| s3 | Store3 | HIJ | 50 | c2 | null |
| s4 | Store4 | KLM | null | c2 | John |

(a) Store in $V_2$

| Store Key | City Key | Product Key | Time Key | Quantity | Amount |
|---|---|---|---|---|---|
| s1 | c1 | p1 | t1 | 5 | 20 |
| s2 | c2 | p1 | t1 | 3 | 15 |
| s3 | c2 | p2 | t2 | 2 | 18 |
| s4 | c2 | p2 | t3 | 3 | 9 |
| null | c3 | p1 | t4 | 2 | 25 |

(b) Sales in $V_2$

| City Key | Name | Representative | State Key | Region Key |
|---|---|---|---|---|
| c1 | Brussels | John | s1 | r1 |
| c2 | Liège | Doe | s2 | r2 |
| c3 | Charleroi | Ahmed | s3 | r2 |

(c) City in $V_2$

**Fig. 4.** State of the data warehouse using the STV approach

| Store Key | Name | Address | City | Manager |
|---|---|---|---|---|
| s4 | Store4 | KLM | c2 | John |

(a) Store in $V_2$

| City Key | Product Key | Time Key | Quantity | Amount |
|---|---|---|---|---|
| c3 | p1 | t4 | 2 | 25 |

(b) Sales in $V_2$

| City Key | Name | Representative | State Key |
|---|---|---|---|
| c1 | Brussels | John | s1 |
| c2 | Liège | Doe | s2 |
| c3 | Charleroi | Ahmed | s3 |

(c) City in $V_2$

**Fig. 5.** State of the data warehouse using the MTV approach

StoreKey of Store as a foreign key. A new foreign key constraint is required to associate the new version of Store with fact table Sales. This is possible by creating a new version of table Sales and using StoreKey attribute of the new version of Store in it. As a result of the second schema change, level Store is deleted from dimension Geography and a version of table Sales is created because henceforth, the facts are assigned to level City. Figures 5b and 5c show tables Sales and City created as a result of the second schema change.

An advantage of the MTV approach over the STV one is that it does not require the null values to be stored for the dropped columns thus it prevents the storage space overhead. The disadvantages of this approach are that the data belonging to a table can be accessed either by creating materialized views or performing joins. The materialized views introduce the problem of view maintenance whereas, depending upon the data size in the DW and the number of existing versions, the join operations may become a performance overhead.

We propose a new Hybrid Table Version (HTV) approach for implementing the MVDW. Usually, the dimension tables in a DW have fewer records as compared to the number of records in the fact tables. We propose for changes in dimension schema, a single table version for each dimension level throughout

```
SELECT StoreKey, Name,
       Address, Area, City
FROM   Store
```

(a) Store using $V_0$

```
SELECT StoreKey, Name, Address,
       Manager, City
FROM   Store
```

(b) Store using $V_1$

```
SELECT StoreKey, TimeKey,
       ProductKey, Quantity,
       Amount
FROM   Sales
```

(c) Sales using $V_0$

```
SELECT    CityKey, TimeKey, ProductKey, SUM(Quantity) AS Quantity, SUM(Amount) AS Amount
FROM      Sales_V0 S, Store T
WHERE     Sales.StoreKey = Store.StoreKey
GROUP BY  City, TimeKey, ProductKey
UNION
SELECT    CityKey, TimeKey, ProductKey, Quantity, Amount
FROM      Sales_V1
```

(d) Sales using $V_2$

**Fig. 6.** Derivation of version instances from the global instance using views

the lifespan of the DW. This table version is defined as the union of all the attributes that have ever been defined for the dimension level. If the attributes are added or deleted from the level, they are treated in the same way as they are treated in the STV approach. Since, the data is loaded more frequently into fact tables and they contain more records than dimension tables, it is more advantageous to create a new table versions for every change in the schema of the facts. We are aware that the creation of a new structure for every fact version may negatively impact the query performance but indexing techniques [4] can be used to address the issue of efficiency. The HTV approach avoids the null space overhead in case of fact tables and limits the number of joins by managing dimension versions in a single table. In this way, the HTV approach combines the advantages of both the STV and MTV approaches. For brevity, we do not show the the state of the DW after schema changes using HTV approach but the state of levels Store and City can be envisioned as shown in Figs. 4a and 4c, respectively and the fact Sales is represented by Figs. 3d and 5b.

The data from the multiple versions of the MVDW can be accessed by defining a set of views. Whenever a new version of a dimension or fact is created, a view definition is also created to access the existing members using this newly created version. To access the new members using the existing versions, existing view definitions need to be modified. For example, the members of level Store can be accessed in versions $V_0$ and $V_1$ using the views defined in Figs. 6a and 6b, respectively. Similarly, the views defined in Figs. 6c and 6d return the fact members in versions $V_0$ and $V_2$, respectively. As a result of second schema change, the granularity of dimension Geography was changed. The view in Fig. 6d aggregates the existing sales facts to display them at the granularity of level City.

## 6 Conclusions

In this paper, we presented (1) a logical model of a multiversion data warehouse (MVDW), and (2) the hybrid table version (HTV) approach to implement

the MVDW. This approach combines the benefits of both the single table version (STV) and the multiple table version (MTV) approaches and creates new table versions only for the fact tables. As future work, we plan to combine our approach with temporal data warehouses so that the history of both the changes in the structure and content of the DW can be maintained. We plan to extend the presented model in such a way that the functionality of schema and content changes can be implemented as independent plug-ins. We are also working on the experimental evaluation of the HTV approach and its impact on query performance. Further, we have plans to develop a query language and data structures for MVDWs.

## References

1. Bebel, B., Eder, J., Koncilia, C., Morzy, T., Wrembel, R.: Creation and management of versions in multiversion data warehouse. In: Proc. of ACM SAC, pp. 717–723. ACM (2004)
2. Blaschka, M., Sapia, C., Höfling, G.: On schema evolution in multidimensional databases. In: Mohania, M., Tjoa, A.M. (eds.) DaWaK 1999. LNCS, vol. 1676, pp. 153–164. Springer, Heidelberg (1999)
3. Brahmia, Z., Mkaouar, M., Chakhar, S., Bouaziz, R.: Efficient management of schema versioning in multi-temporal databases. International Arab Journal of Information Technology 9(6), 544–552 (2012)
4. Chmiel, J.: Indexing multiversion data warehouse: From ROWID-Based multiversion join index to bitmap-based multiversion join index. In: Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Novickis, L. (eds.) ADBIS 2009. LNCS, vol. 5968, pp. 71–78. Springer, Heidelberg (2010)
5. Eder, J., Koncilia, C., Morzy, T.: The COMET metamodel for temporal data warehouses. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 83–99. Springer, Heidelberg (2002)
6. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. Data & Knowledge Engineering 59(2), 435–459 (2006)
7. Golfarelli, M., Rizzi, S.: A survey on temporal data warehousing. International Journal of Data Warehousing and Mining 5(1), 1–17 (2009)
8. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd edn. John Wiley & Sons (2013)
9. Malinowski, E., Zimányi, E.: A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models. Data & Knowledge Engineering 64(1), 101–133 (2008)
10. Rizzi, S., Golfarelli, M.: X-time: Schema versioning and cross-version querying in data warehouses. In: Proc. of ICDE, pp. 1471–1472. IEEE (2007)
11. Wei, H.-C., Elmasri, R.: Schema versioning and database conversion techniques for bi-temporal databases. Annals of Mathematics and Artificial Intelligence 30(1-4), 23–52 (2000)
12. Wrembel, R., Bębel, B.: Metadata management in a multiversion data warehouse. In: Spaccapietra, S., et al. (eds.) Journal on Data Semantics VIII. LNCS, vol. 4380, pp. 118–157. Springer, Heidelberg (2007)
13. Wrembel, R., Morzy, T.: Managing and querying versions of multiversion data warehouse. In: Ioannidis, Y., et al. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 1121–1124. Springer, Heidelberg (2006)