Creation and Management of Versions in Multiversion Data Warehouse [1]

Bartosz Bębel
Institute of Computing Science
Poznań University of Technology
Poznań, Poland
bartosz.bebel@cs.put.poznan.pl

Johann Eder
Department of Informatics-Systems
University of Klagenfurt
Klagenfurt, Austria
eder@isys.uni-klu.ac.at

Christian Koncilia
Department of Informatics-Systems
University of Klagenfurt
Klagenfurt, Austria
koncilia@isys.uni-klu.ac.at

Tadeusz Morzy
Institute of Computing Science
Poznań University of Technology
Poznań, Poland
+48 61 6652370
tadeusz.morzy@cs.put.poznan.pl

Robert Wrembel
Institute of Computing Science
Poznań University of Technology
Poznań, Poland
+48 61 6652529
robert.wrembel@cs.put.poznan.pl

ABSTRACT
A data warehouse (DW) provides an information for analytical processing, decision making, and data mining tools. On the one hand, the structure and content of a data warehouse reflects a real world, i.e. data stored in a DW come from real production systems. On the other hand, a DW and its tools may be used for predicting trends and simulating a virtual business scenarios. This activity is often called the what-if analysis. Traditional DW systems have static structure of their schemas and relationships between data, and therefore they are not able to support any dynamics in their structure and content. For these purposes, multiversion data warehouses seem to be very promising. In this paper we present a concept and an ongoing implementation of a multiversion data warehouse that is capable of handling changes in the structure of its schema as well as simulating alternative business scenarios.

Categories and Subject Descriptors
H.2. [**Database Management**]: Logical Design – *data models.*

General Terms
Design, Experimentation, Performance.

Keywords
data warehouse, versioning, integrity constraints.

1. INTRODUCTION
A data warehouse (DW) integrates autonomous and heterogeneous external data sources (EDSs) in order to provide an information for analytical processing, decision making, and data mining tools [11]. For a few recent years users of data warehousing systems have been paying a special attention to the analysis of operational data produced by OLTP (On-Line Transaction Processing) applications, in order to discover trends, anomalies, and patterns of behavior. Different analytical tools enable the analysts to make better decisions.

An important consequence of the autonomy of EDSs is that they may evolve in time independently of each other and independently of a data warehouse. The changes in EDSs can be categorized as: (1) content changes, i.e. insert/update/delete records, and (2) schema changes, i.e. add/modify/drop an attribute or a table that are very common as reported in [19, 20]. Both types of changes may lead to schema changes in a data warehouse. For instance, adding a new attribute in one of the EDS may require adding this attribute to the DW schema, if one would like to analyze values of this attribute in the warehouse. Furthermore, even a content modification in the EDS, e.g., inserting a new record, may lead to a schema modification in the DW. For instance, inserting a new product in an EDS may lead to modification of the structure of the Products dimension in the DW.

Most of the research done so far with respect to the data warehouse maintenance has focused on providing transactional incremental DW refreshing under content changes of EDSs. However, changes in the content of EDSs as well as changes in the structure of EDSs may lead to schema changes of a DW. This issue has not received much attention so far [7, 8, 19].

A naive approach to tackle the problem of schema changes is to isolate the changes from a data warehouse. Isolation can be accomplished by the middleware level. This technique may be

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

applicable to a limited period of time only, since evolution of EDS schemas will lead to inconsistencies between EDSs and their descriptions at a data warehouse level. In a consequence, it may restrict the usage of existing client analytical queries and reports or the analyses may produce obsolete results. A more advanced approach to tackle the problem of schema changes is to ensure the correct propagation of these changes to the data warehouse definition, i.e. the structure and content of a DW must be correctly adjusted. The data warehouse schema adjustments can be done in two different ways, namely schema evolution [3] and schema versioning [16, 18].

The first approach consists in updating the schema and transferring the data from an old schema into a new schema. Only the current version of the schema is present. In contrast, the second approach keeps track of the history of all versions of a schema. Versioning can be done implicit by temporal extension or explicit by physically storing different versions of data.

The process of good decision making often requires forecasting future business behavior, based on present data and assumptions made by decision makers. This kind of data processing is called the **what-if analysis**. In this analysis, a decision maker simulates changes in the real world, creates a virtual possible scenarios, and explores them with OLAP queries. To this end, a data warehouse must provide means of creating various data warehouse alternatives, represented by different data warehouse versions.

In our project we propose two different kinds of versions: (1) *real versions*, which handle changes made to EDSs, and (2) *alternative versions*, which handle changes made by a user directly in a data warehouse for the purpose of applying the what-if analysis. Real versions represent previous states of EDS. These previous states can be represented as a linear sequence of different real versions as depicted in Figure 1. However, as future is not yet known, alternative versions necessary for the what-if analysis have to be represented by a branching time model as shown in Figure 1. Notice that sometimes the user/administrator of a DW may be interested in preserving old alternative versions. Consequently, our approach also allows to manage different (branching) alternative versions in the past.

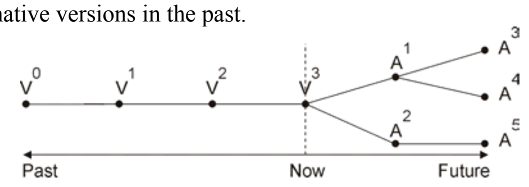

Figure 1. Representation of real versions (V^i) and alternative versions (A^j)

To illustrate the issues mentioned above, let us consider an example of a police data warehouse, storing information about committed violations and tickets given to drivers, in given locations (cities located in provinces) at given periods of time. Violations are organized into severity groups that define minimum and maximum fines allowed for violations. Let us assume that as a result of legislative changes, the borders of provinces changed causing that some cities that had previously belonged to one province, were moved to another one. This is an example of a change in the real world that has an impact on a data warehouse schema. In order to handle this change, a new real data warehouse version should be created. Assuming that a certain percent of fines paid in a city feeds the budget of that city, the police may investigate how the city budget would increase if they moved a violation from one group to a group of more severe violations. This is an example of the real world simulation (the what-if analysis) and it should be handled by an alternative data warehouse schema version.

DW systems and OLAP tools existing on the market support neither managing changes of a data warehouse structure nor the what-if analysis functionality. [22] is an example of a software tool that is able to deal with some evolution issues. However, it is limited to some basic operations and is not able to cope with complex operations as presented in [7]. Furthermore, the approach presented in [22] does not support storing several alternative data warehouse versions for the what-if analysis. Therefore, there is evidently a need to develop techniques of management of schema changes in data warehouse systems, techniques of managing alternative "versions" of the same data warehouse, and build such systems.

Our contribution. Our approach to the problem of maintaining a DW under changes of schemas and contents of EDSs is based on explicit versioning the whole data warehouse (i.e. schema and data). Changes into a data warehouse structure and data are reflected in a new, explicitly derived, version of a DW. The model of a multiversion data warehouse that we developed allows modeling alternative DW versions. The set of data originating from one version, can be persistently stored in another version.

Maintaining real and alternative versions of the whole data warehouse allows us on the one hand, to run queries that span multiple versions and compare various factors computed in those versions, and on the other hand, to create and manage alternative virtual business scenarios required for the what-if analysis.

We implemented the mentioned approach as a prototype software using Visual C++. Data and metadata are stored in an Oracle database.

Paper organization. The rest of this paper is organized as follows. Section 2 presents basic definitions in the field of data warehouse technology. Section 3 presents a leading DW example. Section 4 overviews existing approaches to DW evolution. Section 5 discusses our concept of a multiversion data warehouse, discusses types of data warehouse versions and their properties, presents time integrity constraints defined for DW versions, as well as sketches our data sharing technique. Section 6 briefly presents our prototype multiversion data warehouse system. Finally, Section 7 summarizes and concludes the paper.

2. BASIC DEFINITIONS

The most popular architecture of data warehouses are multidimensional data cubes, where **measures** which are instances of **facts**, i.e. subjects of analysis, that are described in terms of hierarchically organized **dimensions**. Examples of measures include: number of items sold, income, turnover, etc. Typical examples of dimensions are Time, Geography, Products, etc. A value of a measure in a n-dimensional cube is referenced by a n-dimensional vector, where each element corresponds to an element of a dimension.

Dimensions are usually organized in hierarchies. An example of a hierarchical dimension is *Geography*, with *Countries* at the top, that are composed of *Regions*, that in turn are composed of *Cities*. A schema object in a hierarchy is called a **level**. Values in every level are called **dimension members**.

Multidimensional cubes can be implemented either in **MOLAP** (multidimensional OLAP) servers or in **ROLAP** (relational OLAP) servers. In the former case, a cube is stored in multidimensional array. In the latter case, a cube is implemented as the set of relational tables, some of them represent dimensions, and are called **dimension tables**, while others store values of measures, and are called **fact tables**. In the paper, we will focus our discussion on ROLAP implementation of a DW, but our concept can also be used in a MOLAP implementation.

In the rest of this paper, we will use the definitions of a **data warehouse schema** and a **data warehouse instance** as presented in [7] and [8]. Hence, the schema of a data warehouse is the set of all defined dimensions, dimension levels, dimension members, hierarchical relations between dimension members and dimension levels, and facts. The instances of a data warehouse are the measures, i.e., the cell values stored in fact tables.

Therefore, we refer to each modification of a dimension, dimension level, dimension member or fact as schema evolution and schema versioning respectively.

3. MOTIVATING EXAMPLE

Let us consider the police data warehouse, mentioned in the introduction. This DW stores data about violations committed by drivers. Violations are inspected in various locations and at a certain time. The police analyzes those data in order to find out how many violations were committed in given cities at certain periods of time. Cities where drivers are inspected are grouped into administrative regions, whereas violations are organized into groups.

The schema of the police DW is shown in Figure 2. The schema is composed of the three following dimensions: **Locations**, **Violations**, and **Time**. The **Locations** dimension is composed of two levels, namely: **Regions** and **Cities**. The **Violations** dimension is also composed of two levels: **Violation_Groups** and **Violations**. Every dimension member of **Violation_Groups** defines a minimum and maximum fine that can be given for a given violation (attributes *min_fine* and *max_fine*, respectively). The **Time** dimension has one level. Records in the Violation_Groups, Violations, Regions, Cities, and Time tables are called **dimension members**.

The **Inspected_Violations** fact table stores the information about: number of violations committed (attribute *nb_violations*), total fine paid by drivers for these violations (*total_fine*), and date the violations were inspected (*time_id*). Let us further assume that the tables store the following data.

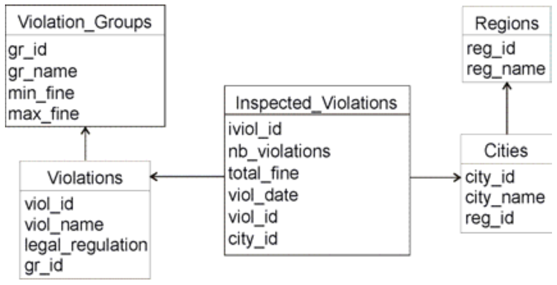

Fig.2. An example schema of the police DW

```
select * from violation_groups;
GR_ID GR_NAME MIN_FINE MAX_FINE
----- ------- -------- --------
    1 Group A       10      100
    2 Group B      100      350

select viol_id, viol_name, gr_id from violations;
VIOL_ID VIOL_NAME         GR_ID
------- ------------ ----------
      1 Violation 1           1
      2 Violation 2           1
      3 Violation 3           2

select * from cities;            select * from regions;
CITY_ID CITY_NAME REG_ID         REG_ID REG_NAME
------- --------- ------         ------ --------
      1 Poznań         1              1 Region A
      2 Warsaw         2              2 Region B
      3 Konin          1

select * from inspected_violations;
TIME_ID    VIOL_ID    CITY_ID NB_OF_VIOL TOTAL_FINE
------- ---------- ---------- ---------- ----------
      1          1          1         10        650
      1          2          1         25        900
      1          3          2         15       3200
      1          1          3         20       2000
```

Examples of user analytical queries that run on the DW may include: (1) compute sum of fines ever paid and (2) compute sum of fines paid in every city.

The data warehouse describes the real world that is likely to be changing. In order to capture these kinds of changes real DW versions are needed. The police may also want to simulate various operational scenarios. To this end, alternative DW versions are needed.

3.1 Real Data Warehouse Version

The real world represented in a DW may change. In our example, changing the borders of regions may result in cites being moved from one region to another. Such a change has an impact on the analytical results received from a data warehouse.

Let us assume a query computing a sum of fines per region and its result:

```
REG_NAME             SUM(IVI.TOTAL_FINE)
-------------------- -------------------
Region A                            3550
Region B                            3200
```

After moving city “Konin” from “Region A” to “Region B” results of the same query are different, e.g.:

```
REG_NAME             SUM(IVI.TOTAL_FINE)
-------------------- -------------------
Region A                            1550
Region B                            5200
```

In order to handle this kind of changes as well as process and analyze data properly we need a new version of a data warehouse, describing the real world after changes. In this case an old DW version would store data before an administrative-territorial change, and a new DW version would store data after that change.

3.2 Alternative Data Warehouse Version

A DW may also be used for simulating various operational/business scenarios. For example, assuming that a certain percent of fines paid by drivers in a city feeds the local budget, the police may investigate how the budget would increase if they moved a violation from the group of ordinary violations to a group of more severe ones. Let us assume that in this simulation scenario “Violation 2” was moved from “Group A” to “Group B”.

In order to create such a simulating environment, a data warehouse must be able to create alternative versions of schema and data as well as to manage these versions. The change must be applied to the structure of the **Violations** dimension, consisting in assigning a given violation to a new group of violations (a violation's foreign key update).

Next, a police decision maker may assume an increase in fines paid by drivers. This assumption is based on an observation that the higher the maximum fine allowed for a violation, the higher the tickets on average. In such a simulating scenario, a new version of fact data will also be created from the previous version. The changes to fact data require computing new values of *total_fine* for every affected record in **Inspected_Violations**. In our example, only record `<1, 2, 1, 25, 900>` is affected, as it describes violations of type "Violation 2". More precisely, the value of *total_fine* must be increased according to a new range of values defined in "Group B". Thus, we need a kind of conversion function that would compute new values of facts based on original data. In this scenario, the conversion function is written by a data warehouse administrator. New values of the *total_fine* attribute may be computed assuming that coefficient (*total_fine/nb_violations)/max_fine* remains constant after changing group assignment.

Having created a new simulation version of a DW, and having converted the fact data, a decision maker may compare the real situation with a hypothetical situation. In both cases the total sum of fines as well as the sum of fines paid in each city is computed, as shown below.

REAL CASE

```
SUM(IVI.TOTAL_FINE)       CITY_NAME SUM(IVI.TOTAL_FINE)
-------------------       --------- -------------------
               6750       Konin                    2000
                          Poznań                   1550
                          Warsaw                   3200
```

SIMULATION CASE

```
SUM(IVI.TOTAL_FINE)       CITY_NAME   SUM(IVI.TOTAL_FINE)
-------------------       ----------- -------------------
               9000       Konin                      2000
                          Poznań                     3800
                          Warsaw                     3200
```

In the simulation case city "Poznań" would substantially increase its budget.

As we have shown in the above examples, data warehouse versions are useful for handling changes in the real world as well as for simulating various business scenarios. Real data warehouse versions are applied in the first case, whereas alternative data warehouse versions are applied in the second case. Both kinds of versions are orthogonal.

4. 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 Manufacture systems. The problem was intensively investigated and resulted in the development of various approaches and prototypes, e.g. [23, 24, 25, 26, 27]. These and many other approaches were proposed for versioning complex objects for a moderated size of the whole set of data. In data warehouse systems objects being versioned have very simple structure (several fact or dimension tables). The size of a database is however very large. Therefore, those versioning mechanisms are not suitable for data warehouse versioning.

The approaches to the management of changes in a data warehouse can be classified into the two following categories that support: (1) schema and data evolution: [3, 9, 10, 13], (2) temporal and versioning extensions [5, 7, 8, 15, 1, 2, 4, 12, 14, 17, 19]. The approaches in the first category support only one data warehouse schema and its instance. When a change is applied to a schema all data described by the schema must be converted, that incurs high maintenance costs.

In the approaches from the second category, in [5, 7, 8, 15] changes are time stamped in order to create temporal versions. However, [5] and [15] expose their inability to express and process queries that span or compare several temporal versions of data. On the contrary, the model and prototype of a temporal data warehouse presented in [7, 8] support queries for a particular temporal version of a DW or queries that span several versions. In the latter case, conversion functions must be applied, as data in temporal versions are virtual.

In [12, 14, 17, 19] implicit versioning in a data warehouse was proposed. In all of the four approaches, versions are used for avoiding conflicts and mutual locking between OLAP queries and transactions refreshing a data warehouse. As versions are implicitly created and managed by the system, these mechanisms can not be used in the what–if analysis. The same drawback applies to the previously discussed temporal data warehouses that can manage only consecutive versions linearly ordered by time.

On the contrary, [2] proposes permanent user defined versions of views in order to simulate changes in a data warehouse schema. However, the approach supports only simple changes in source tables and it does not deal either with typical multidimensional schemas or evolution of facts or dimensions. Also [4] supports permanent time stamped versions of data. The proposed mechanism, however, uses one central fact table for storing all versions of data. In a consequence, the set of schema changes that may be applied to a data warehouse is limited, and only changes of dimensions' structure are supported.

An approach supporting the what–if analysis was presented in [1]. It may be considered as a kind of virtual versioning. A hypothetical query is executed on a virtual structure, called scenario. Then, the system using substitution and query rewriting techniques transforms the hypothetical query into an equivalent query that is run on a real data warehouse. As this technique computes new values of data for every hypothetical query, based on virtual modifications, the performance problems will appear for large warehouses.

5. MODEL OF A MULTIVERSION DATA WAREHOUSE

In order to be able to manage changes in a data warehouse schema a model of a data warehouse with versioning capabilities was developed in [16]. In our approach, changes to a schema may be applied to a new version of a data warehouse. This version, called a child version, is **explicitly derived** by a DW administrator from any previous version, called a parent version. A **multiversion data warehouse** (MVDW) is composed of the set of its versions. Every version of a MVDW is in turn composed of a schema version and an instance version. The latter stores the set of data consistent with its schema version. Versions of a data warehouse form a **version derivation graph**. Each node of this graph represents one version, whereas edges represent *derived–from* relationships between two consecutive versions. In our approach, a version derivation graph is a DAG.

5.1 Versions of a Data Warehouse

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

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

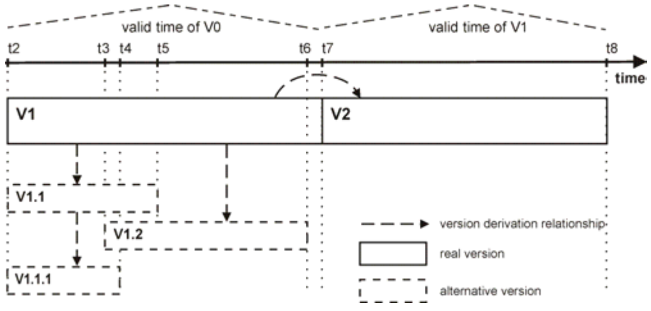

Fig.3. An example of a set of DW versions

Figure 3 schematically shows real versions and alternative versions. V0 is an initial real version of a DW. Based on V0, a new real version V1 was created and in this version the **Regions** level was added, new dimension members were inserted to this level, and existing cities were classified according to regions. Next, an alternative version V1.1 was derived from V1. In V1.1, "Violation 2" was moved from "Group A" to "Group B", as discussed in Section 3. Then, alternative version V1.1.1 was derived from V1.1 and this newly derived version "City 2" was moved from "Region North" to "Region South". V1 is also the parent version for another alternative version V1.2, which simulates the case when minimum fine of violation group "Group A" was increased by 10%. Note that the data warehouse schemas of V1.1, V1.2, and V1.1.1 are the same and are identical with their parent version V1.

5.2 Time Constraints on Versions

Every version is valid within certain period of time. In order to check a version validity, every real and alternative DW version has associated, so called valid time, represented by two timestamps, i.e. beginning valid time (BVT) and ending valid time (EVT). For example, real version V0 (from Figure 3) is valid within time t0 – BVT and t1 – EVT, whereas V1 is valid within t2 and t3. Alternative versions V1.1, V1.2, and V1.1.1 are valid within the same time period as its parent real version. Below we formally define version valid time constraints.

Let **DWV** be the set of data warehouse versions and V_i be a version in **DWV**. Let ➔ be the version derivation dependency between a parent V_m and a direct child version V_n.

Let **T** represent any real time, t_i be time in **T**. Let **VT** be a set of valid times and $VT_i = <t_k, t_l>$ where VT_i in **VT** and $\{t_k, t_l\}$ in **T**.

TC1: Real Versions Valid Time Constraint

Real versions are linearly ordered by their valid time. The ending valid time of a parent real version may be the beginning valid time of its child real version, or the valid time periods of parent and child real DW versions are disjoint.

Let $VT_m = <t_a, t_b>$ be a validity time of data warehouse version V_m and $VT_n = <t_c, t_d>$ be a validity time of version V_n.

$\forall (V_m, V_n)$ in **DWV**: V_m ➔ V_n then $VT_m \cap VT_n = \emptyset$ or $t_b = t_c$.

This constraint is illustrated in Figure 4. The ending valid time of real versions V1, i.e. t_7 is the beginning valid time of real version V2, where V2 is the child of V1.

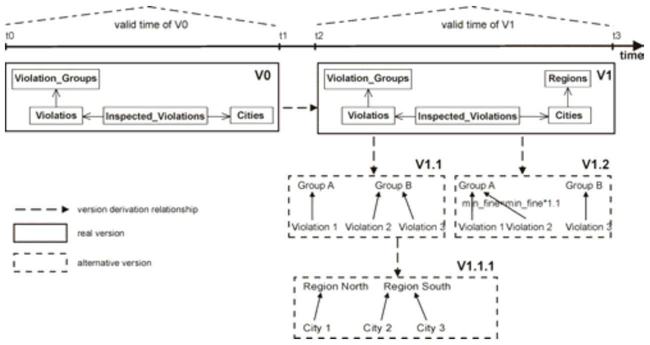

Fig.4. Real and alternative versions and their valid time constraint

TC2: Real–Alternative Versions Valid Time Constraint

A valid time of any alternative DW version is within the valid time of its parent real version.

Let $VT_r = <t_a, t_b>$ be a valid time of real version V_r and $VT_a = <t_c, t_d>$ be a validity time of alternative version V_a.

$\forall (V_r, V_a)$ in **DWV**: V_r ➔ V_a then $VT_a \subseteq VT_r$

This constraint is illustrated in Figure 4. Two alternative versions V1.1 and V1.2 were derived from real version V1. Valid time of V1.1 is $<t_2, t_5>$ and valid time of V1.2 is $<t_3, t_6>$, and both valid times are within the scope of valid time of V1.

TC3: Alternative Parent–Child Versions Valid Time Constraint

A valid time of a child alternative DW version is within a valid time of its direct parent version.

Let $VT_y = <t_a, t_b>$ be a valid time of alternative version V_y and $VT_z = <t_c, t_d>$ be a valid time of alternative version V_z.

$\forall (V_y, V_z)$ in **DWV**: V_y ➔ V_z then $VT_z \subseteq VT_y$

This constraint is illustrated in Figure 4. A child alternative version V1.1.1 was derived from its parent alternative version V1.1. Valid time of V1.1.1 is $<t_2, t_4>$ and valid time of V1.1 is $<t_2, t_5>$, where $t_5>t_4$.

From this constraint we can deduce that valid times of alternative DW versions at the same level of the version derivation hierarchy may overlap. It allow to represent alternative versions valid at the same time, while constraints TC2 and TC3 hold.

Let $VT_y = <t_a, t_b>$ be a valid time of alternative version V_y, $VT_z = <t_c, t_d>$ be a valid time of alternative version V_z, and V_x be a parent alternative version, where $V_x \Rightarrow V_y$ and $V_x \Rightarrow V_z$.

$\forall$ (V_y, V_z) in **DWV**: ($VT_y \cap VT_z \neq \emptyset$ or $VT_y \cap VT_z = \emptyset$) and (TC2 and TC3 hold).

According to the above observation, two alternative versions at the same level of derivation hierarchy are presented, i.e. V1.1 and V1.2. The valid time of V1.1 is $<t_2, t_5>$ whereas the valid time of V1.2 is $<t_3, t_6>$. In this case valid times of both versions overlap at $<t_3, t_5>$, additionally, constraint TC2 holds as valid times of both alternative child versions are within the valid time of their parent real version V1.

5.3 Data Sharing between Versions of a Data Warehouse

A naive approach to dealing with versions of data consists in storing a physical copy of data in every DW version. As the size of data warehouses is of terabytes, this approach is not suitable. Therefore, in our prototype system we are implementing data sharing technique, that is sketched in this section. This technique consists in physically storing in a given DW version only those data that were changed in a given version or are new. Other data, common to a parent and its child versions are stored only in the parent version and are shared by its child versions.

For the data sharing purpose every record, in a fact or a level table, has attached the information about all DW versions this record belongs to. At the implementation level, the information about all versions a given record belongs to is represented in the set of bitmaps, where one bitmap represents one DW version. The number of bits in a bitmap equals to the number of records in a given table. The i^{th} bit in a bitmap, describing version V_m, is set to 1 if the i^{th} record in a table, in DW version V_m, exists in this version. Otherwise the bit is set to 0.

As an simplified example illustrating our data sharing technique let us consider the content of the *Inspected_Violations* table (from Section 3), as shown below. Initially this table exists in version V1. Let us further assume that the alternative version V1.1 was derived from V1, as discussed earlier in the paper. The change in V1.1 concerned moving "Violation 2" from "Group A" to "Group B". In this case, original records from V1 are shared also by V1.1. To this end a new bitmap describing version V1.1 is added to the table, as shown below.

```
Inspected_Violations (version V1)

TIME_ID VIOL_ID CITY_ID NB_OF_VIOL TOTAL_FINE    V1  V1.1
------- ------- ------- ---------- ---------- ----- -----
      1       1       1         10        650     1     1
      1       2       1         25        900     1     1
      1       3       2         15       3200     1     1
      1       1       3         20       2000     1     1
```

Let us also assume that another real DW version, i.e. V2, was derived from V1 and a new dimension **Policemen** was added. After such a change records from the *Inspected_Violations (version V1)* table can not be shared by V2. In a consequence, a new *Inspected_Violations (version V2)* table is created at the implementation level for storing records loaded into version V2.

On the one hand, physical sharing reduces storage overhead, but on the other hand it slows down query processing. This is a trade off between disk storage and good query performance. In the process of a DW tuning (in order to increase query performance), a DW administrator may give up the physical sharing of data between, say parent version V_m and child version V_n, and may decide to create the physical copy of data in version V_n. To this end, in our prototype we are implementing a `dump version` operation.

In order to reduce the number of existing alternative versions, by default our system automatically removes all alternative DW versions derived from version V_m when a new real version is derived from V_m. It is because those alternative versions become obsolete when a new real DW version is created. A decision maker can however mark any alternative version persistent, preventing it from removal while deriving a new real version.

6. PROTOTYPE MULTIVERSION DATA WAREHOUSE

The model of a multiversion data warehouse presented in Section 5 is currently being implemented as a prototype multiversion data warehouse management system. The beta version of this prototype was implemented in Visual C++, whereas data and metadata are stored in an Oracle database. The main management window of our software is shown in Figure 5. It is composed of a *version navigator*, located at the left hand side and *schema viewer*, located at the right hand side. The schema viewer allows to among others: (1) inspect the version derivation graph, (2) see the content of every version, and (3) administrate versions. Whereas the schema viewer graphically presents the schema of a selected version.

In the current implementation, a version of a data warehouse schema can be modified by means of 12 operations [21]: creating a new dimension, removing a dimension, creating a new level, connecting a level into a dimension, disconnecting a level from a dimension, removing a level from a schema, creating a new attribute for a level, removing an attribute from a level, creating a new fact table, creating a new attribute for a fact table, creating an association between a fact table and a dimension, removing an attribute from a fact table, removing an association between a fact table and a dimension, removing a fact table from a schema.

In addition to the above 12 schema modification operations, our prototype supports 3 operations that change the structure of dimensions: (1) inserting a new dimension member into a given level, (2) deleting a dimension member from a given level, (3) changing the association of a sublevel dimension member to another super–level member.

7. SUMMARY, CONCLUSIONS AND FUTURE WORK

Commercial DW systems existing on the market have static structure of their schemas and relationships between data. In a consequence, they are not well suited for handling of any changes that occurs in the real world. A novel approach to this problem is based on a multiversion data warehouse.

In this paper we presented the concept of a multiversion data warehouse, types of versions needed in such a warehouse, and we defined inter-version time integrity constraints. A unique feature of our model of a multiversion DW is its ability to represent **alternative versions** of a data warehouse (required for the what-if analysis) as well as physical separation of different DW versions. We predict that on the one hand, queries spanning several versions will run faster than in other approaches, discussed in Section 4, as DW versions are physically stored, thus, no dynamic data conversions are required. But on the other hand, the complexity of DW metamodel reflecting versions and sharing common elements as well as data will incur time overhead for processing queries spanning several versions. Our concept is currently being implemented as a prototype software. The first beta version of our prototype supports the management of versions of a data warehouse schema. Current work focuses on physical sharing of data between several DW versions. Future work will concentrate on: (1) developing a multiversion query language capable of processing data from several DW versions, (2) developing new mechanisms of indexing multiversion data, (3) developing a model of transactions for a multiversion DW, (4) experimental evaluation of schema and data management techniques as well as efficiency of processing queries addressing several DW versions.

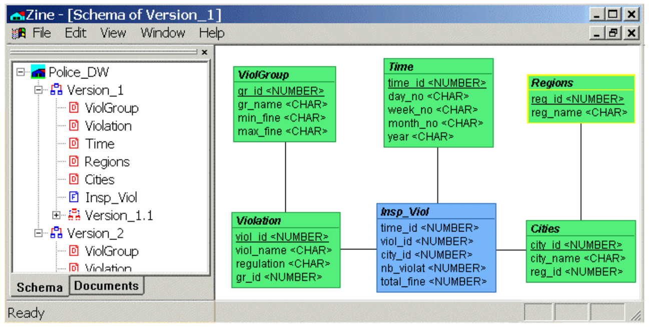

Fig.5. The main management window of the prototype multiversion data warehouse system

8. REFERENCES

[1] Balmin, A., Papadimitriou, T., Papakonstanitnou, Y.: Hypothetical Queries in an OLAP Environment. Proc. of the VLDB Conf., Egypt, 2000

[2] Bellahsene, Z.: View Adaptation in Data Warehousing Systems. Proc. of the DEXA Conf., 1998

[3] Blaschka, M. Sapia, C., Hofling, G.: On Schema Evolution in Multidimensional Databases. Proc. of the DaWak99 Conference, Italy, 1999

[4] Body, M., Miquel, M., Bédard, Y., Tchounikine A.: A Multidimensional and Multiversion Structure for OLAP Applications. Proc. of the DOLAP'2002 Conf., USA, 2002

[5] Chamoni, P., Stock, S.: Temporal Structures in Data Warehousing. Proc. of the Data Warehousing and Knowledge Discovery DaWaK, Italy, 1999

[6] Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. SIGMOD Record, 26, 1997

[7] Eder, J., Koncilia, C.: Changes of Dimension Data in Temporal Data Warehouses. Proc. of the DaWak 2001 Conference, Germany, 2001

[8] Eder, J., Koncilia, C., Morzy, T.: The COMET Metamodel for Temporal Data Warehouses. Proc. of the CAISE'02 Conference, Canada, 2002

[9] Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Maintaining Data Cubes under Dimension Updates. Proc. of the ICDE Conference, Australia, 1999

[10] Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Updating OLAP Dimensions. Proc. of the DOLAP Workshop, 1999

[11] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer-Verlag, 2000, ISBN 3-540-65365-1

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

[13] Koeller, A., Rundensteiner, E.A., Hachem, N.: Integrating the Rewriting and Ranking Phases of View Synchronization. Proc. of the DOLAP Workshop, USA, 1998

[14] Kulkarni, S., Mohania, M.: Concurrent Maintenance of Views Using Multiple Versions. Proc. of the Intern. Database Engineering and Application Symposium, 1999

[15] Mendelzon, A.O., Vaisman, A.A.: Temporal Queries in OLAP. Proc. of the VLDB Conference, Egypt, 2000

[16] Morzy, T., Wrembel, R.: Modeling a Multiversion Data Warehouse: A Formal Approach. Proc. of the Int. Conf. on Enterprise Information Systems - ICESI'2003, France, 2003

[17] Quass, D., Widom, J.: On–Line Warehouse View Maintenance. Proc. of the SIGMOD Conference, 1997

[18] Roddick J.: A Survey of Schema Versioning Issues for Database Systems. In Information and Software Technology, volume 37(7):383-393, 1996

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

[20] Sjøberg D.: Quantifying Schema Evolution. Information Software Technology 35, 1, 35-54, 1993

[21] Wrembel, R. Bębel B.: Schema Management in a Multiversion Data Warehouse. Submitted to the First Special Interest Symposium on Data Warehousing and Data Mining, Germany, July, 2003

[22] SAP America, Inc. and SAP AG. Data Modelling with BW - ASAP for BW Accelerator. 1998. http://www.sap.com

[23] Agrawal, R., Buroff, S., Gehani, N., Shasha, D. (1991). Object Versioning in Ode. Proc. of the ICDE Conference

[24] Ahmed-Nacer M., Estublier J.: Schema Evolution in Software Engineering. In: Databases – A new Approach in ADELE environment. Computers and Artificial Intelligence, 19:183-203, 2000

[25] Bielikova M., Navrat P.: Modelling Versioned Hypertext Documents. Proc. of the ECOOP´98, SCM-8 Symposium, Belgium, 1998

[26] Cellary, W., Jomier, G. (1990). Consistency of Versions in Object-Oriented Databases. Proc. of the VLDB Conference

[27] Kim, W., Chou, H. (1998). Versions of Schema for Object-Oriented Databases. Proc. of the VLDB Conference