A Multiversion Trajectory Data Warehouse to Handle Structure Changes

Wided Oueslati, Jalel Akaichi
Higher Institute of Management of Tunisia
widedoueslati@live.fr, jalel.akaichi@isg.rnu.tn

Abstract

The data warehouse (DW) technology was developed to integrate heterogeneous information sources for analysis purposes. Information sources are more and more autonomous and they often change their content due to perpetual transactions (data changes) and may change their structure due to continual users' requirements evolving (schema changes). Handling properly all type of changes is a must. In fact, the DW which is considered as the core component of the modern decision support systems has to be update according to different type of evolution of information sources to reflect the real world subject to analysis. . The goal of this paper is to propose a solution, based on versioning approach, able to handle structure changes in order to keep track of the DW evolution.

Keywords: *Schema evolutions, Versioning, alternative version, real version, and trajectory data warehouse evolution.*

1. Introduction

Information sources which are integrated in the DW are autonomous and they can change their schema independently of DW. Such changes must be supported when they rich the DW. In fact, the DW technology is seen as the process of good decision making since it provides necessary tools for data analysis such as the On Line Analytical Processing (OLAP). In the literature the DW evolution can be classified into three different approaches namely schema evolution [1, 2, 3, 4] and schema versioning [5, 6] when the DW is defined as a multidimensional schema (fact and dimension tables) and view adaptation and synchronization [7, 8, 9, 10, 11, 12, 13, 14, 15] when the DW is defined as a set of materialized views. The goal of this paper is to present a DW Versioning to handle structure changes and content changes, then to propose a comparative study between those works.
This paper is organized as follows. In section 2, we present different researches works related to DW evolution. In section 3, we present comparative studies between researches works cited above. In section 4, we summarize the work and we propose new perspectives that can be done in the future.

2. Related Works

In the literature, the DW evolution can be classified into three different approaches namely schema evolution [1, 2, 3, 4], schema versioning [5, 6] and view maintenance [7, 8, 9, 10, 11, 12, 13, 14, 15]. The first approach focuses on dimensions updates [1, 2], instances updates [4], facts updates and attributes updates [3]. In [1], authors proposed some operators to define dimension updates. Those operators are: Generalize operator, Specialize operator, Relate levels operator, Unrelated levels operator, Delete level operator, Add instance

operator and Delete instance operator. After defining operators to handle dimension updates, authors of [1] saw that they must handle the impact of dimension structural updates on the data cube. In fact, they proposed some data cube adaptation after the Dellevel update, Addlevel update, DelInstance update and AddInstance update by computing for each cube view an expression to maintain it.

In [2] authors proposed an extension to the work presented in [1] and defined the WareHouse Evolution System (WHES) prototype to support dimensions and cubes update. In fact, they extended the SQL language and gave birth to the Multidimensional Data definition Language (MDL). This latter allowed defining operators for to support evolution of dimensions and cubes. For dimensions update, authors defined the following operators: CreateDimension, DropDimension, RenameDimension, AddLevel, DeleteLevel, RenameLevel, AddProperty and DeleteProperty. For cube updates, authors defined the following operators: CreateCube, DropCube, RenameCube, AddMeasure, DeleteMeasure, RenameMeasure, AddAxis and DeleteAxis.

In [3], authors defined a formal description of multidimensional schemas and instances. This formal description constitutes the data model. This latter was defined as follows: a MD model м is a 6 tuple (F, L, A, gran, class, attr) where F is a finite set of fact names, L is a finite set of dimension level names, A is a finite set of attributes names, Gran is a function that associates a fact with a set of dimension level names, Class : is a relation defined on the level name, Attr is a function mapping an attribute to a given fact or to a given dimension level

After defining the data model, authors presented a set of formal evolution operations. Those latter can have an effect on the model or not. The following evolution operations have no effects on the model: Insert level, Delete level, Insert attribute, Delete attribute, Insert classification relationship, Delete classification relationship, Connect attribute to dimension level, Disconnect attribute from dimension level, Connect attribute to fact, Disconnect attribute from fact, Insert fact:, Delete fact, Insert dimension into fact and Delete dimension.

The second approach focuses on keeping trace of changes by keeping different versions of a given DW [5, 6]. In [5] authors make the difference between schema evolution and schema versioning. In fact, for them schema evolution consists in transferring old data from old schema and updating it in a new schema. However, schema versioning consists in keeping the history of all versions by temporal extension or by physical storing of different versions

The third approach focuses on maintaining a materialized view in response to data changes [7, 8, 9, and 10] or to data sources changes [10, 11, 12, 13, and 14] and sometimes to monitor the DW quality under schema evolution [15]. Research works elaborated in the context of view maintenance can be classified into the following categories:

- View adaptation [7, 8, 9, and 10]: this approach consists in adapting views to changes by adding meta data to materialized views. Those meta data contain structural updates related to materialized view.
- View synchronization (rewriting of views) [10, 11, 12, 13, and 14]: many research works were interested in this approach because it is in relation with other problems such as data integration, data warehouse modeling…

In [10] Bel presents an approach for dynamic adaptation of views related to data sources (relations sources) changes. The main idea of this work is to avoid the recomputation of views which are defined from several sources. In fact, the key idea is to compute the new view from the old one. Bel presents the view adaptation problem from two different points of views. The first point of view is from the user or from the DW designer or administrator and the second point of view is from data sources. For the first point of view, the user or the DW designer can bring into play schema changes on views (e.g. adding an attribute, delete an attribute,

modifying an attribute domain in a view schema) independently of the data sources. Then the changes of the view definition lead to recompute the materialized view. This is the so called "view adaptation". For the second point of view, the data sources (relation sources) can change their schema. This type of changes can touch the DW structural consistency since it may invalidate the materialized views. In this case the solution is to preserve the structural consistency of the DW. This kind of view adaptation is so called "the structural view maintenance". Bel investigated the view adaptation problem from both point of views citing above. Let's start with the view adaptation from point of view data sources or relation sources: the author of [10] presents the impact of schema changes of data sources on the SELECT, WHERE and FROM clauses of the view query.

EVE system [11] proposes a prototype solution to automate view definitions rewriting to solve the problem of view inflexibility. This solution has the goal to preserve the maximum number of affected view definitions by the occurrence of information sources schema changes. The EVE approach assumes that information sources are integrated in the EVE system via a wrapper which translates their models into a relational common model. They are supposed to be heterogeneous and autonomous which join, or change dynamically their capabilities such as their schema.

EVE system includes two basic modeling tools: a model permitting to user to express view definition evolution via an extended SQL called Evolvable SQL (E-SQL) [11] and a model for the description of the information sources (MISD) [11] and the relationships between them. This model of Information Sources description can be exploited for seeking a suitable substitution for the affected view definition components (attributes, relations, and conditions).

The View Knowledge Base (VKB) described by E-SQL and the Meta Knowledge Base (MKB) revealed by MISD, represent the base for any operation of view rewriting or view synchronization process.

Authors of [14] propose to design a mobile agents view synchronization system based on EVE called MAVIE. This latter has to ensure data warehouse maintenance under schema changes.

MAVIE solution decreases the synchronization time due to parallelism permitted by mobile agents and avoids the saturation of the network. The architecture of MAVIE system [14] is distributed on four entities which are the mobile MKB agent, the mobile VKB agent, the mobile detector agent and the mobile synchronizer agent. All those agents know each other via their identifier, names and sites. That fact will assure the direct communication between all mobile agents.

In [15], Quix saw that the quality of the DW is important. In fact, his approach is embedded in the Data Warehouse Quality (DWQ) framework. The aim of [9] is not to provide new techniques to maintain views but to monitor the DWQ under evolution. In fact, Quix presented many evolution operations and their impacts on quality factor. For example, the add /delete of a view and the add/delete of an attribute to/from view will affect the completeness, the correctness and the consistency of the logical schema. The rename of a view will affect the interpretability and understand ability of the view and its attributes. The change of an attribute domain will affect the interpretability of data. The add of an integrity constraint will affect the credibility and consistency of data in data store. The delete of an integrity constraint will affect the consistency of data.

3. Preliminaries

A data warehouse is a data repository which collects and maintains a large amount of data from distributed, autonomous and heterogeneous data sources.

Multidimensional modeling of a DW is usually presented by a star schema [16][17][18]. It is called a star schema because the E\R diagram of his schema resembles a star. The star schema architecture is the simplest data warehouse designing.

The star model is very popular because of its representation which is easy to understand and its performance on very large queries. In fact, this model allows browsing of specific categories, summarizing, drill-down and specifying criteria. The fact table represents the variable to be analyzed and the dimension tables represent the axe of analysis.

The snowflake schema is a variant and a refinement of the star schema, since it keeps the same primitives with normalization of dimension tables to eliminate redundancy. In fact, it represents aggregation hierarchies in the dimensions since, each attribute of a hierarchical level is putted in a dimension table. The snowflake schema [19][20][21] may improve in some cases the performance because smaller tables are joined, and is easy to maintain and increases flexibility.

Last but not least, multidimensional model is based on the idea of the constellation [22] [23]. In this kind of model, we found more than one fact and several dimensions which are shared between facts.

Basic concepts of multidimensional modeling are facts, measures, dimensions and hierarchies.

Definition 1: Fact: it represents the variable to be analyzed by decision makers. Graphically it is a table with two sections. In first section, we find the name of the fact. In the second section, we find its primary key, primary keys of all dimensions that are connected to the fact and a set of measures.

Definition 2: Measure: it represents the quantitative aspect of a fact, it is a numeric property of a fact (e.g. quantity, amount…)

Definition 3: Dimension: it represents the axe of analysis of a given fact. It is a fact property in a given domain. There are two variants of dimensions: Temporal dimension that represents Time at any granularity and Spatial dimension that represents Geography at any granularity.

Definition 4: hierarchy: it is a finite set of members in a dimension and their positions relative to one another. For example, the Geography dimension is defined with the levels Continent, Country, City. In such order, we propose the following schema of dimension Geography and its instances.

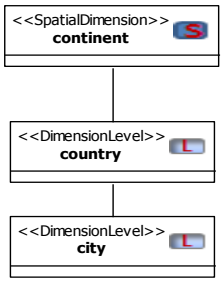

Figure 1: Geography Dimension Structure

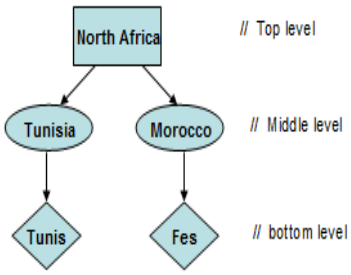

Figure 2: Geography Dimension Instance

North Africa is the continent and it represents the top level of the hierarchy, Tunisia and Morocco are countries in the North Africa continent and they represent the middle level of the hierarchy and finally Tunis and Fes are cities respectively in Tunisia and Morocco, they represent the bottom level of the hierarchy. This latter is more specific than other levels.

4. Multiversion TDW

Schema versioning approach keeps track of the modification of schema level and instance level. The goal is to reduce inconsistent OLAP results. There are two types of schema versioning; real version and alternative version.

The real version can be defined as a version that handles changes of the real world like changing geographical borders of countries.

The alternative version can be defined as a version that handles simulation purposes. It represents the so called what-if-analysis to handle virtual business scenarios.

One or several alternative versions are created from a given real version. They have the Parent/Child relationship.

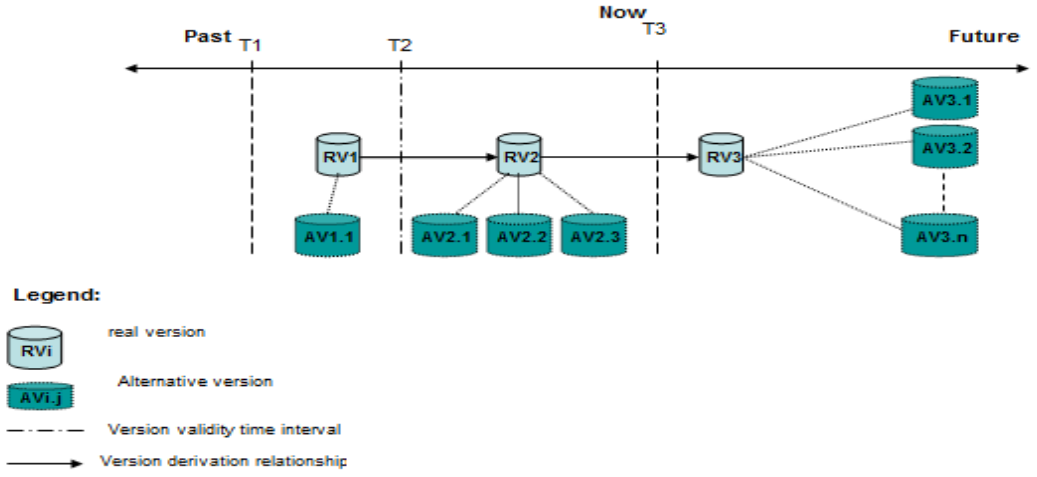

Figure 3. Schema Versionning: Real and Alternative Versions

We propose constraints that have to be fulfilled to guaranty the integrity of our multiversion trajectory data warehouse. Let **MVTDW** be a set of **TDW** versions. Let **RVi** be a real version in **MVTDW** with **i** $\in$ **{1,…, n}**. Let **AVi.j** be a child of the real version **RVi** and an alternative version in the whole **MVTDW**. Let **RVi** $\mapsto$ **AVi.j** means that **AVi.j** is a subset of **RVi**. Versions must be time stamped in order to represent their time validity. Let **VTi** be a valid time interval of **RVi** with a time of begin **tb(RVi)** and a time of end **te(RVi).** Let **VTi.j** be a valid time interval of **AVi.j** with a time of begin **tb(AVi.j)** and a time of end **te(AVi.j)** .

Let **P** be a point of time. **P** is true if **te(RVi) = tb(RVi+1)**, else **P** is false and **P** $\equiv \varnothing$.

Basic constraints related to real versions are that:

- $\forall$ **VTi, tb(RVi)** $\leq$ **te(RVi)** has to be true i.e., a real version may not end before it starts.

- **$\forall$ (RVi, RVi+1); VTi $\cap$ VTi+1 = P.**

In order to give a formal description of the integrity constraints between a real version and its alternative version, we propose the predicate IsIn as follows:

- **IsIn (RVi, AVi.j)** defines that the valid time of an alternative version AVi.j is a subset of the valid time of its parent real version **RVi**. **(RVi, AVi.j) is true if $\forall$ t $\in$ [tb(AVi.j), te(AVi.j)], t $\in$ [tb(RVi) , te(RVi)]** i.e., **$\forall$(RVi, AVi.j), RVi $\mapsto$ AVi.j** then **VTi.j $\subseteq$ VTi.**

As shown in the figure 3, the real version **RV1** and its child **AV1.1** are valid within the time interval **[T1, T2],** the real version **RV2** and its child **AV2.1**, AV2.2 and **AV2.3** are valid within the time interval **[T2, T3]** while the real version **RV3** and its child **AV3.1**, **AV3.2** and **AV3.n** are valid within the time interval **[T3, +∞]** since we don't know yet what will happen in the future.

5. Running Example

To elucidate the problem more deeply, we propose an example of a trajectory Data Warehouse (TDW) [24]. This latter stores trajectory data related to moving objects during their trajectories. Those latter are composed of trajectory sections that are composed of stops and moves.

In our running example, the TDW stores trajectory data resulting from mobile hospitals. Those latter move in their trajectories and stop in a given district to check if women living there have the breast cancer disease. Decision makers analyze trajectory data in order to know the number of breast cancer patients in a given stop that is in a given district. Patients are organized into group-patient. This latter indicates the stage of the disease assuming that the breast cancer has three stages. The trajectory fact table stores information about the number of patients, the date of diagnostic and the district where patients were diagnosed in the mobile hospital. The schema of the TDW is shown as follows:

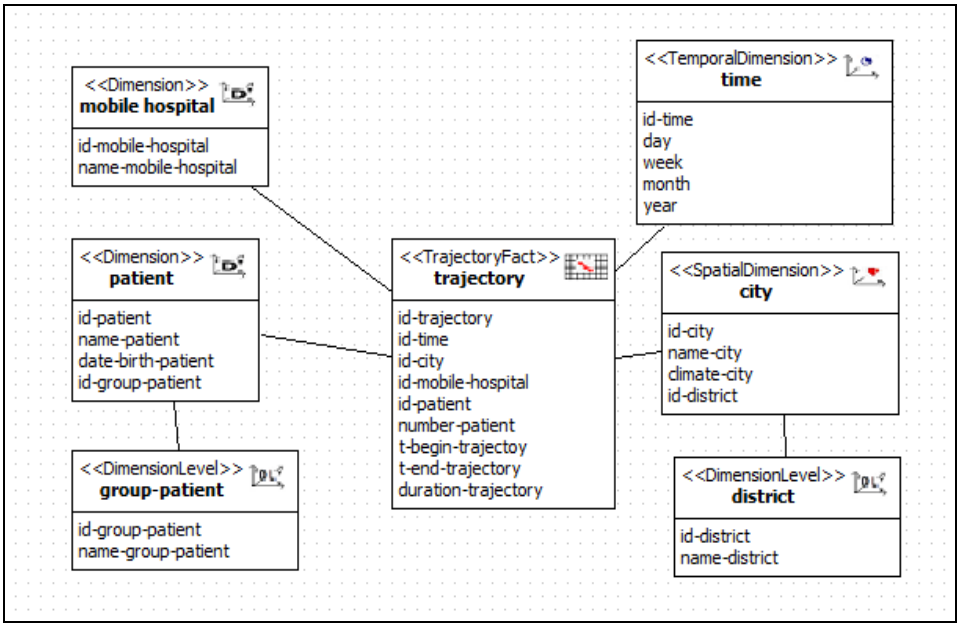

Figure 4. A Schema of the Trajectory Data Warehouse with T-UML [25]

5.1. Identification of Schema Changes and their Impacts

As we know, correct strategic decisions can keep a business alive; therefore, a TDW has to be adaptable to any changes that can happen. Those changes are due to changes of data sources in the course of time or to the changes of users' requirements over a period of time.

The goal of this section is to identify a set of TDW schema changes. This set will include known atomic changes, e.g. dimension updates, fact updates, and attribute updates. Moreover, we will propose changes that are specific to trajectories. For example, since spatial dimensions and measures can have several granularities, one can add a granularity to a dimension, or change the granularity of a dimension that is involved in a fact. In addition to atomic changes, our set of schema changes will include complex changes, e.g. merging two dimension hierarchies into one. This type of change has infrequently been addressed in the literature.

A multidimensional schema can undergo evolutions which can put back the existing schema by having impacts of variable importance on the data. For example, the evolutions of schema impacting on the table of the facts have generally important consequences on the stored data. Indeed, the volumetric of the data warehouse depends generally on the volumetric of the fact tables. So the impact on data of the fact table can be considerable. We evoke the evolutions of schema according to the importance of their impacts on the data warehouse.

OLAP systems are composed of fact tables and dimension tables forming the multidimensional structure of analysis. A lot of models consider that fact table is the dynamic part of the DW and dimension tables are static. However, in real world, axes of analysis evolve in time.

5.1.1 Dimension Updates

We define in this section a set of primitives operations related to the management of schema evolution especially the dimension updates and a set of integrity constraints to be respected for this end. Those latter will be defined as follows:

IC 1: constraint add dimension: the primary key of the added dimension must be a foreign key in the trajectory fact table (PK new D $\in$ {FK Trajectory Fact}).

IC 2: constraint drop dimension: the primary key of the removed dimension must be removed from trajectory fact table (PK removed D $\notin$ {FK Trajectory Fact}).

IC 3: constraint add attribute to a dimension: none

IC 4: constraint delete attribute from a dimension: the deleted attribute must not be the identifier i.e. the primary key of the dimension (deleted attribute $\neq$ PK dimension).

- **Add dimension**

First of all, a possible evolution is the addition of a dimension. It amounts to increase the level of detail of the fact table. In fact, facts will be more detailed, since measures in the fact table will be described by an additional dimension and will present more descriptors.

The impact on the data is considerable because it is not only a question of adding a table of dimension but also of recalculating all the data of the fact table when data sources allow to calculate measures one more time for old facts. We propose the following Add-Dimension formal definition:

Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of

the version VT then V=(TF, D, H, IC, VT). Dn is the name of the new dimension, Pk is the primary key of the new dimension and A is the set of its attributes. The result of the primitive Create Dimension gives birth to a new version V' = (TF', D', H, IC, VT) where TF'= (TF name, Measures, TF foreign key ∪ {new dimension primary key}) and D'= D∪ {Dn, Pk, A}.

After presenting the formal definition, we propose the following Add-Dimension algorithm. This latter will be applied on the basic TDW version.

Algorithm Creation-Dimension
Declare
newD: the new dimension
ID-newD: the identifier of the new dimension
PK: primary key of the Tajectory-Fact
Begin
1*: Create new Dimension structure : newD: ID-newD and Descriptors*
2*: insert (ID-newD,descriptors) into newD*
4*: Alter Tajectory-Fact (ID-newD)*
Add CONSTRAINT FOREIGN KEY
REFERENCES newD (ID-newD);
5: *Alter Tajectory-Fact*
*Modify CONSTRAINT PK ∪{ID-newD }****;***
End

- **Delete dimension**

Another possible evolution is the deletion of a dimension, which allows decreasing the level of detail of the fact table. In fact, facts will be less detailed.

The impact on the data is considerable because it is necessary to recalculate the aggregates of the fact table because the identifier of the deleted dimension would be eliminated from the fact table. Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Dn is the name of the deleted dimension, Pk is the primary key of the deleted dimension and A is the set of its attributes. The result of the primitive Delete Dimension gives birth to a new version V'= (TF', D', H, IC, VT) where TF'= (TF name, Measures, TF foreign key \ {deleted dimension primary key}) and D'= D \ {Dn, Pk, A, H}.

After presenting the formal definition, we propose the following Delete-Dimension algorithm. This latter will be applied on the basic TDW version.

Algorithm Delete-Dimension
Declare
D: the dimension to be deleted
ID-D: the identifier of the dimension to be deleted
PK: primary key
FK: foreing key
Input*: D*
Begin
1*: Alter Table Trajectory-Fact*
Modify CONSTRAINT PK= PK \{ ID-D };
2*: Alter Table Trajectory-Fact*
Drop CONSTRAINT PK= FK (*ID-D)*;
3*: Update Trajectory-Fact*
Set ID-D =NULL;

***4**: Alter Table Trajectory-Fact*
Delete COLUMN (ID-D);
***5**: Drop Table D*
End

- **Add attribute to a dimension**

A possible dimension update is to insert a new attribute in a dimension. This increases the number of descriptors of a given dimension. Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Dn is the name of the dimension, and An is the name of added attribute. The result of the primitive Add Attribute gives birth to a new version V'= (TF, D', H, IC, VT) where D'= {(Dn, A ∪ {An}, H)}.

After presenting the formal definition, we propose the following Add-Attribute algorithm. This latter will be applied on the basic TDW version.

Algorithm Add-Attribute to Dimension
Declare
D: the Dimension to which a new attribute will be added
A: the added attribute to a given dimension D
***Input**: D*
Begin
***1**: Alter Dimension D*
Add COLUMN (A);
End

- **Delete attribute from a dimension**

A possible dimension update is to delete an attribute from a dimension. This decreases the number of descriptors of a given dimension. Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Dn is the name of the dimension, and An is the name of deleted attribute. The result of the primitive delete Attribute gives birth to a new version V'= (TF, D', H, IC, VT) where D'= {(Dn, A \ {An}, H)}.

After presenting the formal definition, we propose the following Delete-Attribute algorithm. This latter will be applied on the basic TDW version.

Algorithm Delete-Attribute from Dimension
Declare
D: the Dimension from which an attribute will be deleted
A: the deleted attribute from a given dimension D
***Input**: Dimension D*
Begin
***1**: Update Dimension D*
Set A= NULL;
***2**: Alter Dimension D*
Delete COLUMN (A);
End

5.1.2 Dimension Levels Updates

The hierarchy can be seen as a directed acyclic graph where levels are vertices and links between levels are edges.

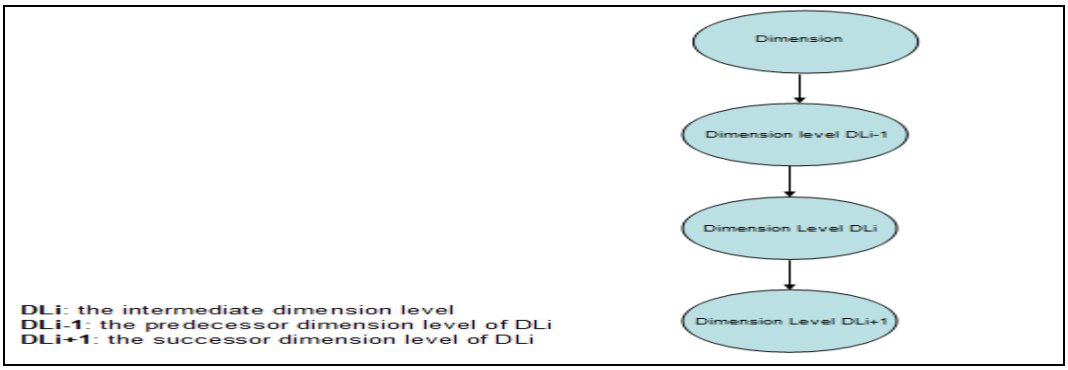

Figure 5. terminology of levels' granularities

The hierarchy is composed of a set of granularity levels. For this reason, the update of the hierarchy must be followed by the update of its levels. Creating a granularity level into a hierarchy consists on creating a level and a link that connect it to the level on which it is based on. The level can be created in the middle or at the end of a hierarchy. Creating a level in the middle of a hierarchy must have as a goal to aggregate data of the created level towards the existing upper level i.e. data of the created level are semantically aggregately towards the existing upper level. Two scenarios are possible for creating a level at the end of a hierarchy. The first one is that the level is created under the last level of an existing hierarchy and the second one is that the level is created under the dimension itself and this will give birth to a new hierarchy.

Removing a level from a hierarchy leads to the remove of the level itself and the link between it and the level above it. This remove can be performed a level that is in the middle or the end of a hierarchy.

In case of creation or removal of a level that is in the middle of hierarchy, it is necessary to propagate updates in the hierarchy. The propagation aims to define relationships between relevant levels of the updated hierarchy. In fact, in case of creation of a level in the middle of the hierarchy, propagation consists on defining the aggregation link between the created level and the upper level. However, removing a level from the hierarchy, propagation consists on defining the aggregation link between the lower and the upper level of the removed level.

We define in this section a set of algorithms related to dimension levels updates (creation and removal of a level from a hierarchy) and a set of integrity constraints to be respected for this end. Those latter will be defined as follows:

IC 5: constraint add dimension level: the primary key of the added level must be a foreign key in the dimension (Pk DL $\in$ Fk D).

IC 6: constraint drop dimension level: the primary key of the removed level must not be a foreign key in the dimension (Pk of removed DL $\notin$ Fk D).

- **Add dimension level**

Changes affecting the dimension hierarchies, such that adding new levels of granularity enrich an existing hierarchy or define a new hierarchy. For example in the dimension country containing the levels Country > District, we can introduce a new level that group District by City (Country> City > District). Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Dn is the name of the dimension, and Hn is the name of added hierarchy. The result of the primitive Add Hierarchy gives birth to a new version V'= (TF, D', H', IC, VT) where D'= {(Dn, A, H∪{Hn})} and H'={Hn, Levels, (>)} where > represents the dependency relationship between levels.

After presenting the formal definition, we propose the following Add-Dimension-Level algorithm. This latter will be applied on the basic TDW version.

Algorithm Creation-Dimension level
Declare
DLi: *the dimension level to be added to a given hierarchy*
PK: *primary key of a dimension level*
Input*: End-of-hierarchy (= true if DLi will be the end of the hierarchy)*
Begin
If *(End-of-hierarchy)*
1*: Create T able DLi (ID-DLi PK, descriptors);*
2*: Alter Table DL(i-1)*
Add COLUMN (ID-DLi);
3*: Alter Table DL(i-1)*
PK- DL(i-1)= PK- DL(i-1) ∪ {*ID-DLi*};
Else
4*: Create T able DLi (ID-DLi PK, descriptors);*
5*: Alter Table DLi*
ADD CONSTRAINT Primary Key PK-DLi (ID-DLi, ID-(i+1));
6*: Alter Table DL(i-1)*
PK- DL(i-1)= PK- DL(i-1) \ {*ID-DL(i+1)* };
7*: Update DL(i-1)*
Set ID-DL(i+1) = NULL;
8*: Alter Table DL(i-1)*
Delete ID-DL(i+1)
9*: Alter Table DL(i-1)*
Add COLUMN (ID-DLi);
10: *Alter Table DL(i-1)*
PK- DL(i-1)= PK- DL(i-1) ∪ {*ID-DLi*};
End

- **Delete dimension level**

In the case of the abolition of hierarchy level, the importance of the impact depends on the location of this level in the hierarchy. Indeed, if the level is in an intermediate position in the hierarchy, there must be consistency maintaining the necessary links in the hierarchy.). Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Dn is the name of the dimension, and Hn is the name of deleted hierarchy. The result of the primitive Delete

Hierarchy gives birth to a new version V'= (TF, D', H', IC, VT) where D'= {(Dn, A, H\ {Hn})} and H'= H \ {Hn, Levels, (>)} where > represents the dependency relationship between levels.

After presenting the formal definition, we propose the following Delete-Dimension-Level algorithm. This latter will be applied on the basic TDW version.

Algorithm Delete-Dimension level
Declare
DLi: *the dimension level to be added to a given hierarchy*
PK: *primary key of a dimension level*
Input*: Dimension level DLi, end-of-hierarchy (= true if DLi = DLj)*
Begin
If *(End-of-hierarchy)*
1*: Update DL(i-1)*
Set ID-DLi = NULL;
2*: Alter Table DL(i-1)*
PK- DL(i-1)= PK- DL(i-1) ∪ {*ID-DLi*};
3*: Alter Table DL(i-1)*
Delete ID-DL(i-1)
4*: Drop Table DLi*
Else
5*: Update DL(i-1)*
Set ID-DLi = NULL;
6*: Alter Table DL(i-1)*
PK- DL(i-1)= PK- DL(i-1) \ {*ID-DLi*};
7*: Alter Table DL(i-1)*
Delete ID-DLi
8*: Drop Table DLi*
9: *Alter Table DL(i-1)*
Add COLUMN (ID-DL(i+1));
10: *Alter Table DL(i-1)*
PK- DL(i-1)= PK- DL(i-1) ∪ {*ID-DL(i+1)*};
End

5.1.3 Fact Updates

We define in this section a set of primitives operations related to the management of schema evolution especially the fact updates and a set of integrity constraints to be respected for this end. Those latter will be defined as follows:

IC 7: constraint add a measure to a fact: the type of the new measure must be of type numeric (type new m = numeric).

IC 8: constraint delete a measure from a fact: the number of measures must be more than two so after deleting a measure must be more or equal to one (number measures \ deleted measure ≥ 1).

- **Add a measure to a fact**

Another modification which touches the fact table is the addition of a measure. This measure can be diverted from an existing measure. The impact is lesser than when we touch a dimension, because it does not to challenge existing data of the fact table. However, it requires calculating for every fact this new measure.

This latter must share exactly the same dimensions as the other measures of the fact table. If this is not the case, it should be considered creating another fact table that may share some of the existing dimensions. Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Mn is the name of the new measure. The result of the primitive Add Measure to a fact gives birth to a new version V'= (TF', D, H, IC, VT) where TF'= {(TF name, M ∪ {Mn}, D)}. After presenting the formal definition, we propose the following Add-Measure algorithm. This latter will be applied on the basic TDW version.

Algorithm Add-Measure to Trajectory-Fact
Declare
TF: the Trajectory-Fact to which a new measure will be added
M: the added measure to a given Trajectory-Fact TF
Input*: Trajectory-Fact*
begin
1*: Alter Table Trajectory-Fact TF*
Add COLUMN (M);
End

- **Delete a measure from a fact**

The removal of a measure, in turn, affects also the fact table. However, no recalculation of the fact table is needed, since it is only to remove a column. . Let V be a version of the TDW and it is composed of a trajectory fact TF, a set of dimensions D, a set of hierarchies H, a set of integrity constraints IC and a valid time of the version VT then V = (TF, D, H, IC, VT). Mn is the name of the deleted measure. The result of the primitive Delete Measure from a fact gives birth to a new version V'= (TF', D, H, IC, VT) where TF'= {(TF name, M \ {Mn}, D)}.

After presenting the formal definition, we propose the following Delete-Measure algorithm. This latter will be applied on the basic TDW version.

Algorithm Delete-Measure from Trajectory-Fact
Declare
TF*: the Trajectory-Fact from which a measure will be deleted*
M: the deleted measure from a given Trajectory-Fact TF
Input*: Trajectory-Fact*
Begin
1*: Update Table Trajectory-Fact TF*
Set M= NULL;
2*: Alter Table Trajectory-Fact TF*
Delete (M);
End

6. MVTDW Prototype

Our MVTDW is right now being implemented as a prototype. This latter is implemented with C Sharp and it supports derivation of versions from the original TDW version. This latter is stored in oracle DW. The MVTDW prototype is composed of TDW version guide located at the left and schema viewer located at the right. The schema guide allows showing the content of each TDW version. The schema viewer

allows seeing the schema of the selected TDW version. The bottom "new version derivation" can modify the TDW version while applying primitives operations described in section 5.1. Whereas, the bottom "querying versions" can express queries that address several TDW versions. To this end, we propose to extend the SQL language.

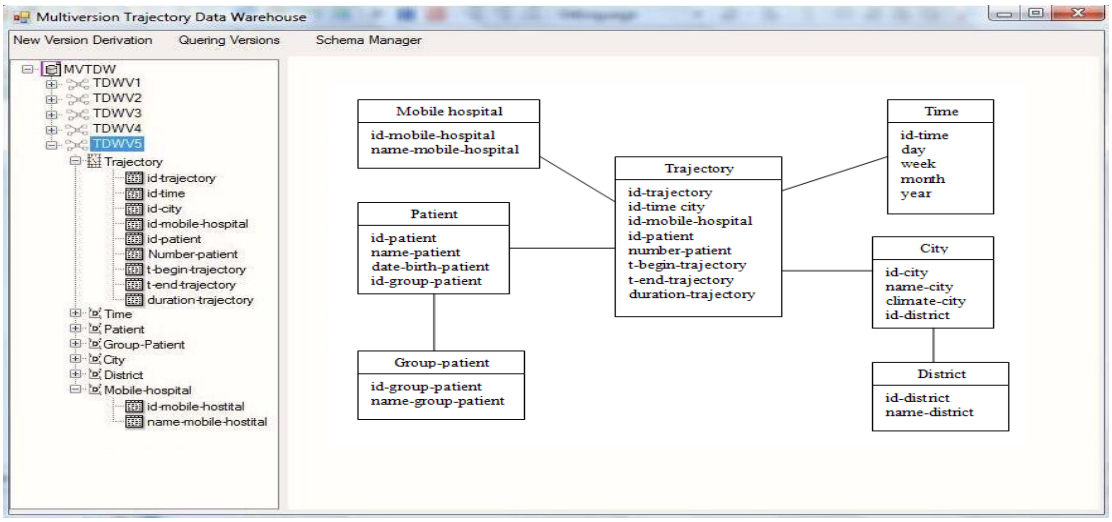

Fig12. MVTDW Prototype

7. Conclusion and Future Works

OLAP systems are composed of fact tables and dimension tables forming the multidimensional structure of analysis. A lot of models consider that fact table is the dynamic part of the DW and dimension tables are static. However, in real world, axes of analysis evolve in time. To solve this problem, we proposed a new approach that is based on versioning called MVTDW. This latter is composed of real versions and alternative versions. We defined some constraints to assure integrity between versions and some algorithms to be applied in case of schema and instance changes on the TDW versions. The user has to define which version should use to answer queries. However, data returned by queries may be in several versions of a TDW. For this reason, it is necessary to ensure data mapping functions between all versions of a given MVTDW and to extend the SQL language which can express queries that address several TDW versions. This focus will be discussed in future works.

References

[1] C A Hurtado, A O Mendelzon, and A A Vaisman. Maintaining Data Cubes under Dimension Updates. In XVth International Conference on Data Engineering (ICDE 99), Sydney, Australia, pages 346–355. IEEE Computer Society, 1999.

[2] E. Benitez-Guerrero,C. Collet, M. Adiba . THE WHES APPROACH TO DATA WAREHOUSE EVOLUTION .e-Gn osis[online], Vol.2Art.2004

[3] M Blaschka, C Sapia, and G Hofling. On Schema Evolution in Multi-dimensional Databases. In Ist International Conference on Data Ware-housing and Knowledge Discovery (DaWaK 99), Florence, Italy, volume1676 of LNCS, pages 153–164. Springer, 1999.

[4] T. Morzy, R. Wrembel, On Querying Versions of Multiversion Data Warehouse. In Proc. Int.Workshop on Data Warehousing and OLAP, DOLAP'04, Washington (USA), 2004.

[5] B Bebel, J Eder, C Koncilia, T Morzy, and R Wrembel. Creation and Management of Versions in Multiversion Data Warehouse. In XIXth ACM Symposium on Applied Computing (SAC 04), Nicosia, Cyprus, pages 717–723. ACM Press, 2004.

[6] M Body, M Miquel, Y Bedard, and A Tchounikine. A Multidimensionaland Multiversion Structure for OLAP Applications. In Vth ACM In-ternational Workshop on Data Warehousing and OLAP (DOLAP 02),McLean, Virginia, USA, pages 1–6. ACM Press, 2002.

[7] A. Gupta, I. M umick, K. Ross. Adapting Materialized Views after redefinitions SIGMOD, pages 211-222, 1995.

[8] A. Nica, E. A. Rundensteiner. View Maintenance after Vew Synchronization. International Database Engineering and application Symposium, pages 213-215, 1999.

[9] E. A. Rundensteiner, A. Nica, A. J. Lee. On Preserving Views in Evolving Environments. Proc. Fourth Int Workshop Knowledge Representation Meets Databases. Pages 131- 141. 1997.

[10] Z Bellahsene. Schema Evolution in Data Warehouses. Knowledge and Information Systems, 4(3):283–304, 2002.

[11] E. A. Rundensteiner, A. Koeller, X. Zhang, A.J. Lee, A. Nica: Evolvable View Environment EVE: A Data Warehouse System Handling Schema and Data Changes of Distributed Sources. Proceedings of the International Database Engineering and Application Symposium (IDEAS'99), Montreal, Canada, April 1999.

[12] A. Rajaraman, Y. Sagiv, J. D. Ullman. Answering Queries Using Templates With Binding Patterns. Proc. ACM Symp. Principles Database System, pages 105-112, 1995 .

[13] L. V. S. Lakshmanan, F. Sadri, I. N. Subramanian. Schema SQL a Language for Interoperability in Relational Multi-Databases Systems. Proc. 22.nd Int Conf. Very Large Databases, pages 239-250, 1996.

[14] J.Akaichi, W.Oueslati. MAVIE: A Mobile Agents View synchronization systEm. In first international conference on the applications of digital information and web technology. Ostravem pages 145-150. 2008.

[15] C. Quix. Repository Support for Data Warehouse Evolution. In Proc. of the Intl WorkshopDMDW, Heidelberg, Germany 2004.

[16] G. M. Freitas Alberto H. F. Laender M. Luiza Campos. MD2 – Getting Users Involved in the Development of Data Warehouse Application, In Proc. of the 4th International Workshop (DMDW), Toronto, Canada, pp. 3-12, 2002.

[17] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual Design of Data Warehouses from E/R Schemas. In Proc. of the 31st Hawaii International Conference on System Sciences, Vol.7, Kona, Hawaii, pp. 334-343, 1998.

[18] M. Krippendorf, and Y. Song. The Translation of Star Schema into Entity Relationship Diagrams. In Proc. of the 8th International Workshop on Database and Expert Systems Applications (DEXA), Toulouse, France, pp. 390-395, 1997.

[19] M. Levene and G. Loizou. Why is the Snowflake Schema a Good Data Warehouse Design? In Information Systems Vol. 3, N°28, pp. 225-240, 2003

[20] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses. Book, John iley & Sons Publishers, Chichester, 1998.

[21] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual design of data warehouses from E/R schemes. In Proceedings of the Hawaii International Conference on System Sciences, Hawaii, pp. 334–343, 1998.

[22] O. Teste. Towards Conceptual Multidimensional Design in Decision Support Systems. In Proc. of the 5th East-European Conference on Advances in Databases and Information Systems (ADBIS), Vilnius, Lithuania, pp. 77-88, 2001.

[23] J. Pokorný , P. Sokolowsky. "A conceptual modelling perspective for Data Warehouses", In Journal of Electronic Business Engineering, pp.666-684, 1999.

[24] W. Oueslati , J. Akaichi. "Mobile Information Collectors TrajectoryData Warehouses Design", International Journal of Managing Information Technology, pp.1-20, 2010.

[24] W. Oueslati , J. Akaichi. "A Trajectory UML Profile for Modeling Trajectory Data: A Mobile Hospital Use Case", International Journal of Advanced Research in Computer Science, pp.481-485, 2011.