A User-driven Data Warehouse Evolution Approach for Concurrent Personalized Analysis Needs

Fadila Bentayeb, Cécile Favre and Omar Boussaid

10th November 2006

Abstract

Data warehouses store aggregated data issued from different sources to meet users' analysis needs for decision support. The nature of the work of users implies that their requirements are often changing and do not reach a final state. Therefore, a data warehouse cannot be designed in one step, usually it evolves over the time. In this paper, we propose a user-driven approach that enables a data warehouse schema update. It consists in integrating the users' knowledge in the data warehouse modeling to allow new analysis possibilities. More precisely, we consider the specific users' knowledge, which defines new aggregated data, under the form of "if-then" rules that we call aggregation rules. These rules are used to dynamically create new granularity levels in dimension hierarchies, following an automatic and concurrent way. Our approach is composed of four phases: (1) users' knowledge acquisition, (2) knowledge integration, (3) data warehouse schema update, and (4) on-line analysis. To support our approach, we define a Rule-based Data Warehouse (*R-DW*) model composed of two parts: one "fixed" part and one "evolving" part. The fixed part corresponds to the initial data warehouse schema, whose purpose is to provide an answer to global analysis needs. The evolving part is defined by means of aggregation rules, which allow personalized analyses. To validate our approach, we developed a prototype called WEDrik (data Warehouse Evolution Driven by Knowledge), in which the R-DW model is implemented within the Oracle 10g DBMS. We also present how to achieve our approach by proposing a model dedicated to the management of the data warehouse schema evolution and the updates' algorithms. Furthermore, we applied our approach on banking data of the French bank LCL-Le Crédit Lyonnais and we illustrate our purpose with the LCL case study.

1 Introduction

Data warehouses are complex systems which store aggregated data issued from different sources, to meet users' analysis needs for decision support. This technology emerged in companies but quickly became a large research domain [24]. In a data warehouse, data are organized in a multidimensional way. The objective is to analyse *facts* through *measures*, according to *dimensions* which can be organized in *hierarchies*, representing different *granularity levels* of information.

Due to the role of the data warehouses in the daily business work of an enterprise, the requirements for the design and the implementation are dynamic and subjective. Therefore, data warehouse design is a continuous process which has to reflect the changing environment of a data warehouse, i.e. the data warehouse schema must evolve in reaction to the enterprise's evolution.

In the data warehouse schema, changes may happen or be required in many different situations. Data sources are often autonomous and generally have an existence and purpose beyond that of supporting the data warehouse itself. An important consequence of the autonomy of data sources is the fact that those sources may change without being controlled from the data warehouse administrator. Therefore, the data warehouse must be adapted to any changes which occur in the underlying data sources, e.g. changes of the schemata. Beside these changes on the source level, the analysts (users) often change their requirements. Indeed, the nature of the work of users implies that their requirements are dynamic and subjective, thus they do not reach a final state.

As an example, let us consider the case of the French bank LCL-Le Crédit Lyonnais[1]. The LCL data warehouse provides an answer to the global analysis needs, in other words, the analysis needs common to the whole of the users. However, since users work in different departments, they may have individual analysis requirements according to their own objectives, witch emerge and grow in time.

Therefore, a data warehouse schema cannot be designed in one step, usually it evolves over the time. We focus here on the evolution induced by emergent

[1]This work was supported by the French bank LCL-Le Crédit Lyonnais, and especially the Rhône-Alpes Auvergne direction

users' analysis needs.

Thus, in this paper, we propose a global approach for a data warehouse schema update, which takes into account users' own needs. This provides a real time evolution of the analysis possibilities of the data warehouse to cope with concurrent and personalized analysis needs. Our key idea consists in generating new analysis axes based on users' knowledge by dynamically extending existing dimension hierarchies or creating new ones. More precisely, we provide here an original solution to define new granularity levels inside the data warehouse schema. Note that to create a new hierarchy, we just have to repeat the same process several times since a hierarchy is a set of linked successive granularity levels.

Our global approach is composed of four phases: (1) users' knowledge acquisition, (2) knowledge integration, (3) data warehouse schema update, and (4) on-line analysis. In our approach, we consider a specific users' knowledge, which provides new aggregated data. The acquisition phase of our approach thus consists in defining this users' knowledge under the form of "if-then" rules that we call aggregation rules. We propose an algorithm to check the validity of these rules, before integrating them in the data warehouse model. To achieve this integration phase, rules are transformed into mapping tables. Then, the evolution phase allows to update the underlying data warehouse schema. We also define algorithms which create a new level in the dimension hierarchy. Furthermore, we propose a model to manage the incremental evolution of the data warehouse. This model is also exploited to provide users with the updated data warehouse schema, since the schema evolves continuously. The last phase of our approach, namely on-line analysis, consists in applying decisionnal queries onto the updated data warehouse schema.

This approach is supported by our proposed data warehouse model based on aggregation rules, named *R-DW* (*Rule-based Data Warehouse*) [13]. Our model is composed of a "fixed" part, corresponding to an initial data warehouse schema that meets the global and known analysis needs; and an "evolving" part, defined by aggregation rules, which express the new needs and create new granularity levels in the current data warehouse. Thus the *R-DW* model allows users to directly integrate their analysis needs into the data warehouse model, without the administrator's intervention. To achieve this objective, we propose a complete formalization of the *R-DW* model, which expands our previous formalization [13] along two main axes. First, we introduce constraints on rules, to ensure the consistency of the data warehouse induced by these rules. Second, we formally define the concurrent aspect, by introducing the concept of granularity level version.

To validate our approach, we developed a prototype named WEDriK (data Warehouse Evolution Driven by Knowledge), where the R-DW model is implemented within the Oracle 10g DBMS, and applied our approach on banking data of the French bank LCL.

LCL is a large company with many employees who need to carry out analyses for decision support. Since users' analysis needs depend on their own objectives and their own experience, these analysis needs sometimes may be concurrent.

The design of integrated concurrent engineering platforms has received much attention, because competing firms strives for shorter design delays and lower costs. Concurrent engineering allows for parallel design, thus leads to shorter design-to-market delays. It however requires advanced coordination and integration capabilities [9]. Concurrent engineering, also known as simultaneous engineering, is finally a non-linear product or project design approach during which all phases operate at the same time. Our approach allows to take into account the concurrent analysis needs since users are directly involve in the data warehouse evolution process.

Thus, users are able to carry out personalized analyses, based on their own knowledge, for the ultimate objective of extracting interesting information. To achieve this objective, On-Line Analytical Processing (OLAP) technology could be useful since it provides tools to summarize, consolidate, and view data so that users are widely able to explore multidimensional data, navigate through hierarchical levels of dimensions [11]. In addition, data mining could be useful too, since it provides tools for automatically discovering hidden knowledge from large data sets, and a data warehouse precisely gathers a large amount of data [14]. Thus, in the recent years, many studies addressed the issue of coupling data warehousing and data mining techniques so that they can benefit from each other's advances [31]. But this paper do not address this issue, even if it is quite interesting.

The remainder of this paper is organized as follows. First, we discuss the state of the art regarding schema evolution in data warehouses in Section 2.

Then, we motivate the need for a user-driven data warehouse schema evolution through an example in Section 3. We detail our global approach for data warehouse evolution in Section 4. Section 5 is devoted to the formalization of our *R-DW* model, on which is based our approach. We also present implementation elements in Section 6, detailing the evolution management model and the updating algorithms. In Section 7, we show, through a running example extracted from the LCL bank case study, how to exploit our approach for analysis purposes. We finally conclude this paper and provide future research directions in Section 8.

2 Related work

Data sources and analysis needs constitute the two essential input parameters in the design of a data warehouse schema. Indeed, current data warehouse development methods can fall within two basic groups: data-driven [20], or analysis needs-driven [22] approaches.

Data-driven approach ignores the needs of data warehouse users a priori. More precisely, it consists in taking the data sources as input to obtain the data warehouse model as output. For example, Golfarelli et al. propose, in [16], a semi-automated methodology to build a dimensional data warehouse model from the pre-existing E/R schemes that represent operational databases. The drawback of this method is to suppose that the generated model meets the analysis needs. However, it is not always the case.

Analysis needs-driven approach consists in defining the data warehouse schema according to analysis needs. Thus users are interviewed to gather their analysis needs before building the data warehouse, this implies user acceptance. However, this does not guarantee the longevity of the model, since the users involved in the design process may change. Besides, it is difficult to gather the analysis needs of the whole of the users in an exhaustive way, and it is more difficult to envisage the future ones.

A new hybrid approach has emerged combining the two precedent ones. It consist in mapping the candidate schemata generated from data sources with the users' analysis needs [8, 27, 30].

When designing a data warehouse schema, involving users is crucial. What about the data warehouse schema evolution ? According to the literature, schema evolution in data warehouses can take the form of schema updating or temporal modeling.

The first approach consists in transforming the data warehouse schema [4, 5, 18, 19]. These works propose to enrich data warehouses with adapted evolution operators that allow an evolution of the schema. In [18, 19], authors model dimensions as an acyclic graph, where nodes represent dimension attributes and arcs represent hierarchical links between these attributes. Then they propose evolution operators under the form of algebraic functions, which modify the graph structure. In [4, 5], the authors proposed elementary operators, which can be combined to make the schema evolve. In the two cases, only one schema is supported and the trace of the evolutions is not preserved.

On the contrary, the second approach keeps track of the schema evolution, by using temporal validity labels. These labels are affixed either on dimension instances [6], or on aggregation links [25], or on schema versions [1, 7, 26]. Let us detail these different methods. In [25], the authors propose a temporal multidimensional model and TOLAP, a query language supporting it, accounting for dimension updates and schema evolution. Dimension elements are timestamped at the schema or instance level (or both) in order to keep track of the updates that occur. The regular star schema treats all dimensions, one of them being the Time dimension, equally and assumes them to be independent. In [6], authors propose a temporal star schema, where time is not a separate, independent dimension table, but is a dimension of all tables and is represented via one or more time-valued attributes in all tables. Another promising approach to handling changes in data warehouse structure and content is based on a multiversion data warehouse [1, 7, 26]. In such a data warehouse, each version describes a schema and data at certain period of time. In order to appropriately analyze multiversion data, an extension to a traditional SQL language is required.

Another research direction that focuses on data warehouse schema updates has emerged. It is founded on the hypothesis that a data warehouse is a set of materialized views. Then, when a change occurs in a data source, it is necessary to maintain views by propagating this change [2, 32, 33].

Both of these approaches do not directly involve users in the data warehouse evolution process, and thus constitute a solution rather for a data sources evolution than for users' analysis needs evolution. Indeed, once the data warehouse is created, the users can only carry out analyses provided by the model. Thus these solutions make the data warehouse schema evolve, without taking into account new analysis needs driven by users' knowledge. To the best of our knowledge, no work focused before on this problem.

Furthermore, the personalization of the use of a data warehouse becomes crucial to extend this use to the most of users. It is a research perspective, which emerges in the data warehouse community. Works in this domain are particularly focused on the visualization of data, based on users' preferences modeling and exploitation with the profile concept [3]. It consists in refining queries to show a part of data, which meets user's preferences. Our approach does not consider the personalization of analyses through their visualization by selecting information. On contrary, we add supplement information to personalize analyses themselves, and not only their visualization.

To achieve a personalization, whatever it is, the data warehouse should be more flexible. Works aimed at bringing flexibility within the data warehouse use mostly rule-based languages. Firstly, to define the data warehouse schema in a flexible way, two approaches are possible: using rules either to express the analysis needs [21] or the knowledge about data warehouse construction [29]. Secondly, the data warehouse administrator could use rules to define some integrity constraints in order to ensure the consistency of the data and the analyses as a consequence. In [10, 15], the expressed integrity constraints use the data semantic to manage data inconsistency within the analysis. Thirdly, in order to make the analysis more flexible, a rule-based language has been developed in [12] to manage exceptions during the aggregation process. This language allows to intentionally express redefinitions of aggregation hierarchies.

Thus, rule-based languages allow flexibility within data warehouses in different works. We want to introduce such a flexibility in the analysis evolution process. However, the flexibility of the analysis evolution depends on the flexibility of the schema evolution, in particular dimension hierarchies updates. The works presented previously constitute answers to the problem of dimensions evolution, when this latter is oriented by the evolution of data themselves. In these works, the intervention of the administrator is needed to implement these solutions. With our approach, we bring a solution to take into account the emergence of new analysis needs directed by the expression of users' knowledge, without the administrator intervention, thus in a concurrent way.

3 Motivating example

To illustrate our approach throughout this paper, we use a case study defined by the French bank LCL. LCL is a large company, where the data warehouse users work in different departments, and thus have different points of view. Then, they need specific analyses, which depend on their own knowledge and their own objectives.

Let us take the example of the annual Net Banking Income (NBI). The NBI is the profit obtained from the management of customers account. It is a measure observed according to several dimensions: `CUSTOMER`, `AGENCY` and `YEAR`

(Figure 1).

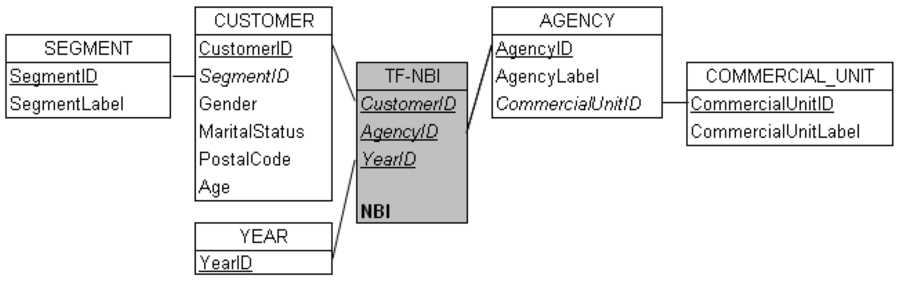


Figure 1: Data warehouse model for the NBI analysis

Concerning the `CUSTOMER` dimension, it is possible to aggregate data according to the `SEGMENT` level, as shown in the dimension schema (Figure 2a). Indeed, it is useful to identify customers according to their profit potential i.e. their segment, to carry out effective marketing campaigns.

Let us take the case of a data warehouse user, who is specialized on targeting customers for products. This user needs to obtain the NBI analysis according to the age of customers. For this end, he needs to separate customers into three age classes: less than 30 years old, between 30 and 60 years old and more than 60 years old, according to its own purposes. The existing data warehouse could not provide such an analysis, since this analysis need was not envisaged before.

The solution we propose is to dynamically add a new level in the data warehouse schema defined by the user, i.e. `Age Class` level, in the `CUSTOMER` dimension hierarchy (Figure 2b), to allow analyses according the age class.

Note that, analysis needs could be concurrent since two users could need to analyse data according to age classes defined in different way. For example,

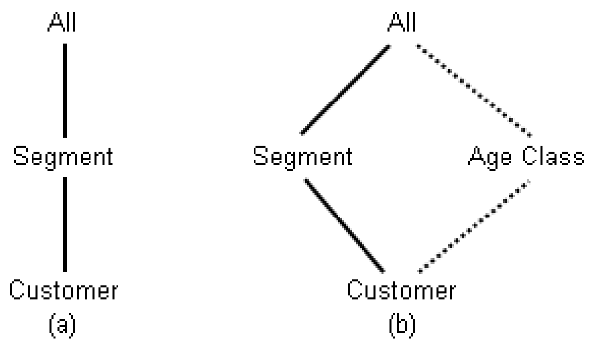


Figure 2: `CUSTOMER` dimension schemata: (a) initial, (b) with the new `Age Class` level

another user perhaps needs to distinguish minor (less than 18 years old) and major (more than 18 years old) customers.

4 A user-driven data warehouse schema evolution approach

In this section, we present a global approach for integrating new analysis needs into the existing data warehouse, to achieve our user-driven data warehouse schema evolution (Figure 3). The first phase of our approach is the *acquisition* of the users' knowledge under the form of "if-then" rules. Then, in the second phase, these rules are integrated within the DBMS (DataBase Management System), which implements the current data warehouse. This *integration* consists in transforming "if-then" rules into mapping tables. Then the *evolution* phase is realized by creating a new granularity level inside an existing hierarchy or defining a new one. Finally, the *analysis* phase could be carry out on the updated data warehouse model. It is an incremental evolution process, since each time new analysis needs may. The aim of this section is to focus on the first three phases.

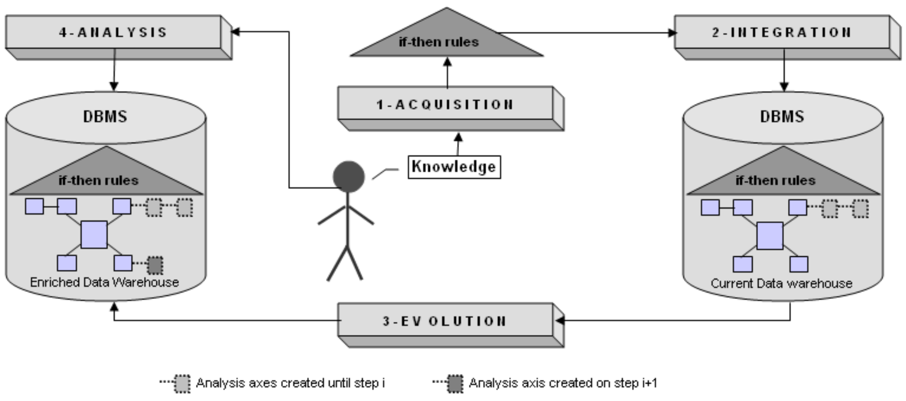


Figure 3: Data warehouse user-driven evolution process

4.1 Users' knowledge acquisition

In our approach, we consider a specific users' knowledge, which determines the new aggregated data to integrate into the underlying data warehouse. More precisely, our idea consists in involving users in the data warehouse schema evolution, to create new granularity levels according to their own knowledge. Thus, for the remainder of the paper, we define the concept of "Aggregation Data Knowledge" (ADK), which defines how to aggregate data from a given level to another one to be created.

The ADK is represented in the form of "if-then" rules. These rules have the advantage of being very intelligible for users since they model information explicitly [17], and are well adapted to define the aggregation link between two successive granularity levels. The *if*-clause contains conditions on the attributes of the lower level. The *then*-clause contains the definition of a higher level, and more precisely the values of the attributes, which characterize the new level to create.

For example, let us consider that one user needs to define the level `AGENCY TYPE` from the lower level `AGENCY`, since he knows that there is three types of agencies: some agencies host only student accounts, some others deal with foreigners, and other agencies are said to be classical. The following rules define the level `AGENCY TYPE`, by defining the values of the attributes `AgencyTypeLabel` and `AgencyTypeCode`, which characterize this level. The conditions expressed in the *if*-clause concerns the attribute Agency_ID of the level `AGENCY`.

(R1) *if* AgencyID $\in$ {'01903','01905','02256'}

then AgencyTypeCode='STU' and AgencyTypeLabel='student'

(R2) *if* AgencyID = {'01929'}

then AgencyTypeCode='FOR' and AgencyTypeLabel='foreigner'

(R3) *if* AgencyID $\notin$ {'01903','01905','02256','01929'}

then AgencyTypeCode='CLA' and AgencyTypeLabel='classical'

Thus, an "if-then" rule allows to define different attributes for the created level. In our example, the attributes `AgencyTypeCode` and `AgencyTypeLabel` are defined for the level `AGENCY TYPE`. However, for the sake of simplicity, we define just one attribute per level in the remainder of the paper, even if it is possible to express other dimension attributes in the "then" clause.

Classically, in a dimension hierarchy, an aggregation link exists between one instance in the lower level and one instance in the higher level. However, in the literature, different types of aggregation links can compose the dimension hierarchies to model real situations [23, 28]. For example, if we carry out an analysis of the sales according to LCL salesmen who work in agencies, the dimension hierarchies for the sales analysis present two levels: salesman and agency. However, it is possible that a salesman works in more than one agency. It requires then a new strategy to aggregate data. Indeed, in this case, if we aggregate data according to the agency level, one sale could be counted several times.

To avoid this problem, in our approach, we deal with "classical" hierarchy, where an instance in a lower level corresponds to exactly one instance in the higher level, and where each instance in the lower level is represented in the higher level. Thus, aggregation rules define a partition of the instances in the lower level, and each class of this partition is associated to one instance of the created level (Figure 4). In this case, we have to check the validity of the rules expressed by users according to this definition of a level, and propose in Section 6 an algorithm to perform this task.

4.2 Knowledge integration

After the acquisition of the ADK under the form of "if-then" rules, these rules have to be integrated within the DBMS, which implements the current data

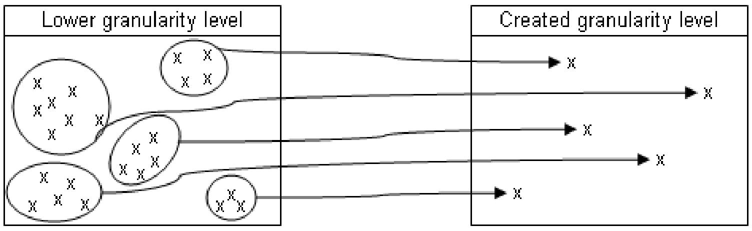


Figure 4: Aggregation rules mechanism

warehouse. This integration consists in transforming "if-then" rules into mapping tables and storing them into the DBMS, by means of relational structures.

For a given set of rules which define one new granularity level, we associate one mapping table, which contains the conditions expressed on attribute(s) of the lower level, the corresponding value(s) for the attribute(s) in the created level, and some others useful information.

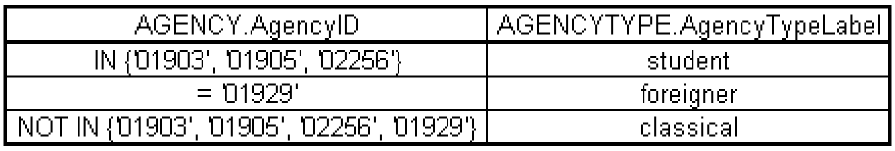
Let us take the example of the different types of agencies. Let us consider only the attribute `AgencyTypeLabel` to characterize the new level, with the following rules.

(R1) *if* AgencyID ∈ {'01903','01905','02256'} *then* AgencyTypeLabel='student'

(R2) *if* AgencyID = {'01929'} *then* AgencyTypeLabel='foreigner'

(R3) *if* AgencyID ∉ {'01903','01905','02256','01929'} *then* AgencyTypeLabel='classical'

The clauses of these rules are exploited to build the corresponding mapping table (Figure 5)



AGENCY.AgencyID	AGENCYTYPE.AgencyTypeLabel
IN {'01903', '01905', '02256'}	student
= '01929'	foreigner
NOT IN {'01903', '01905', '02256', '01929'}	classical

Figure 5: Mapping table example

4.3 Data warehouse schema evolution

The data warehouse schema evolution consists in updating the current schema by creating a new granularity level in a dimension hierarchy. The created level can be added at the end of a hierarchy or inserted between two existing ones, we thus speak of adding and inserting a level, respectively. In the two cases, the rules have to determine the aggregation link between the lower and the created levels. But, in the second case, it is also necessary to determine the aggregation link between the inserted level and the existing higher level in an automatic way.

A user makes the choice to insert a level between two existing ones only if it is possible to semantically aggregate data from the inserted level to the existing higher level. For instance, let us consider the `AGENCY` dimension schema of Figure 6(a). In this case, it is possible to aggregate data according to `Commercial Unit` that is a set of agencies, according to their geographical localization. Let us suppose that we create a new level called `Agencies Group`, which is also a set of agencies, according to their geographical localization, but smaller than a commercial unit. `Agencies Group` level could be inserted between the `AGENCY` and `Commercial Unit` levels, where a commercial unit can be seen as a set of `Agencies Groups` (Figure 6(b)). Now, let us suppose that we add a new level that aggregates data according to the "size" of agencies: small, medium and large. It is semantically impossible to create an aggregation link between the `Agency Size` and the existing level `Commercial Unit`. In this case, the user cannot insert the created level between `Agency` and `Commercial Unit`, but he can create it to define a new hierarchy (Figure 6(c)).

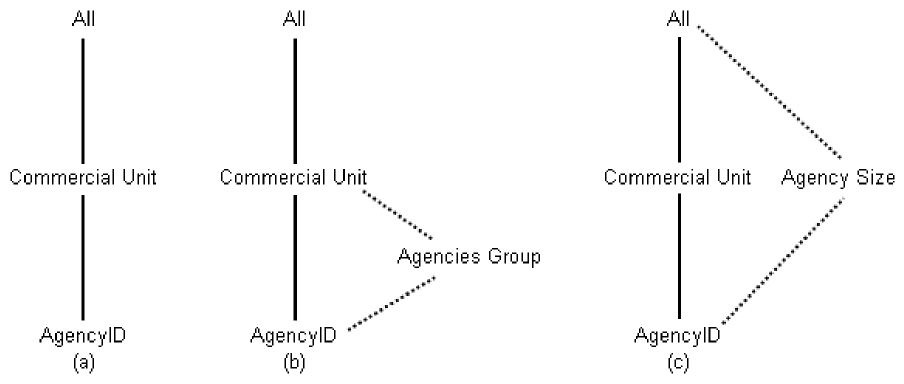


Figure 6: `AGENCY` dimension schemata

For each type of evolution, we propose, in Section 6, an algorithm which creates a new relational table and its necessary link(s) with existing tables.

To achieve these two types of evolution, we propose the two corresponding algorithms in Section 6 which create new relational tables and their link with existing tables. We although propose a model to manage the evolution of the data warehouse, which gathers all the information about the data warehouse model. Moreover it is exploited to provide users with the updated data warehouse schema, since the schema evolves continuously. We also detail this aspect in Section 6.

5 The R-DW model

In this section, we present our R-DW model, on which is based our global approach for schema update in data warehouse.

Our proposed *R-DW* model is composed of two parts: one "fixed" part and one "evolving" part. The fixed part is composed of a fact table and its dimensions, whose purpose is to provide an answer to global analysis needs. The evolving part is defined by aggregation rules, which generate dynamically new granularity levels in dimension hierarchies. These aggregation rules are defined by users.

5.1 R-DW formalization

We represent the Rule-based Data Warehouse model *R-DW* by the following triplet: $R\text{-}DW = (\mathcal{F}, \mathcal{E}, \mathcal{U})$ where $\mathcal{F}$ is the fixed part, $\mathcal{E}$ the evolving part and $\mathcal{U}$ the universe of *R-DW*.

Definition 1. *Universe of the data warehouse.* Given $R\text{-}DW = (\mathcal{F}, \mathcal{E}, \mathcal{U})$; the *universe of the data warehouse* $\mathcal{U}$ is a set of attributes, such as: $\mathcal{U} = \mathcal{U}_1 \cup \mathcal{U}_2$

where $\mathcal{U}_1 = \{B_\alpha, 1 \leq \alpha \leq z\}$ is the set of z predefined attributes (in the fixed part $\mathcal{F}$) and $\mathcal{U}_2 = \{C_\beta, \beta \geq 1\}$ is the set of generated attributes (defined in the evolving part $\mathcal{E}$).

Example 1. $AgencyID \in \mathcal{U}_1$; $AgencyTypeLabel \in \mathcal{U}_2$

Definition 2. *Fixed part of R-DW.* Given *R-DW* = $(\mathcal{F},\mathcal{E},\mathcal{U})$; the *fixed part of R-DW* is represented by: $\mathcal{F} =< F,\mathcal{D} >$ where F is a fact table and $\mathcal{D} = \{D_s, 1 \leq s \leq t\}$ is the set of t first level dimensions directly linked with the fact table F. We assume that these dimensions are independent.

Example 2. In the Figure 1, $<TF\_NBI,\{AGENCY, YEAR, CUSTOMER\}>$ is the fixed part of the *R-DW* for the NBI analysis.

Definition 3. *Dimension hierarchy and granularity level.* Given *R-DW* = $(< F,\mathcal{D} >,\mathcal{E},\mathcal{U})$; $D_s.H_k, k \geq 1$ is a *hierarchy of the dimension* D_s, $D_s \in \mathcal{D}$.

The dimension hierarchy $D_s.H_k$ is composed of a set of w ordered granularity levels noted L_i:

$D_s.H_k = \{L_1, L_2, \ldots, L_i, \ldots, L_w, w \geq 1\}$ with $L_1 \prec L_2 \prec \cdots \prec L_i \prec \cdots \prec L_w$, where $\prec$ express the total order on the L_i.

The granularity level L_i of the hierarchy H_k of the dimension D_s is noted $D_s.H_k.L_i$ or L_i^{sk}. The granularity levels could be defined with attributes called generated attributes.

Example 3. According to the schema of the Figure 2(b), we have:

$D_{CUSTOMER}.H_1$={CUSTOMER, SEGMENT} ; $D_{CUSTOMER}.H_2$={CUSTOMER, AGE CLASS}

$L_2^{CUSTOMER\ 1}$=SEGMENT ; $L_2^{CUSTOMER\ 2}$=AGE CLASS

Definition 4. *Generated attribute.* Given *R-DW* = $(< F,\mathcal{D} >,\mathcal{E},\mathcal{U})$; given L_i^{sk} the created granularity level; given $\mathcal{U}_2$ the set of attributes defined in the evolving part $\mathcal{E}$ of *R-DW*; a *generated attribute* $A \in \mathcal{U}_2$ characterizes the granularity level L_i^{sk} and is noted $L_i^{sk}.A$.

Example 4. In the example of the Section **??** concerning the definition of the age classes, we have $L_2^{CUSTOMER\ 2}.A = AgeClassLabel$

Remark: For the sake of simplicity, we suppose that each generated granularity level of dimension hierarchy is represented by only one generated attribute, even if it is possible to generate more than one attribute per granularity level.

Definition 5. *Aggregation rule.* An *aggregation rule* defines the aggregation link, which exists between two successive granularity levels in a dimension hierarchy. It is based on a set $\mathcal{T}$ of n rule terms noted RT_m, such as:

$$\mathcal{T} = \{RT_m, 1 \leq m \leq n\} = \{U\ op\ \{set|val\}\}$$

where U is an attribute of the universe $\mathcal{U}$; *op* is a relational operator ($=$, $<$, $>$, $\leq$, $\geq$, $\neq$, ...), or an ensemblist operator ($\in$, $\notin$, ...) ; *set* is a set of values and *val* is a given value.

Example 5a. $RT_1\ :\ AgencyID \in \{$'01903','01905','02256'$\}$; $RT_2\ :\ YearID <$ 2001 ; $RT_3\ :\ Gender =$ 'F'

An aggregation rule is an "*if-then*" rule. The conclusion of the rule defines the value of the generated attribute. The premise of the rule is based on a composition of conjunctions or disjunctions of these rule terms:

$$r_{ij} :\ if\ RT_1\ (and|or)\ RT_2\ ...\ (and|or)\ RT_n\ then\ L_i^{sk}.A = val_{ij}$$

where $val_{ij} \in Dom_{L_i^{sk}.A}$ the definition domain of the attribute $L_i^{sk}.A$.

Example 5b. The following rules define the values of the attribute `AgeClassLabel`, which characterizes the granularity level `AGE CLASS`:

r_{11} : *if Age* $<$ 30 *then AgeClassLabel* = 'less than 30 years old'

r_{12} : *if Age between* 30 *and* 60 *then AgeClassLabel* = 'between 30 and 60 years old'

r_{13} : *if Age* $>$ 60 *then AgeClassLabel* = 'more than 60 years old'

Example 5c. The following rule defines the value `'married women'` of the attribute `PersonGroupLabel` according to attributes `MaritalStatus` and `Gender` of `CUSTOMER` table:

if MaritalStatus = 'Married' *and Gender* = 'F' *then PersonGroupLabel* = 'married women'

Thus aggregation rules create or enrich a dimension hierarchy by defining the values of the generated attribute according to a condition (Example 6b.) or a composition of conditions (Example 6c.).

Definition 6. *Evolving part of R-DW.* Given *R-DW* = $(\mathcal{F}, \mathcal{E}, \mathcal{U})$; the *evolving part of R-DW* is represented by $\mathcal{E} = \{(L_i^{sk}.A, \mathcal{R}_{i.})\}$ where $\mathcal{R}_{i.} = \{r_{ij}, 1 \leq i \leq w, 1 \leq j \leq u\}$ is a set of u aggregation rules defining the values of the generated attribute $L_i^{sk}.A$, which characterizes the granularity level L_i of the hierarchy H_k of dimension D_s. Thus, $\mathcal{E}$ represents the set of w granularity levels of hierarchy H_k of dimension D_s and their associated rules.

Example 6. If we consider the creation of `AGE CLASS` and `AGENCY TYPE` levels, we have the following evolving part:

$$\mathcal{E} = \left\{ \begin{array}{l} \left(L_2^{\text{CUSTOMER 2}}.A\ , \left\{ \begin{array}{l} r_{11}\text{: if Age} \leq 30 \text{ then AgeClassLabel='less than 30 years old'} \\ r_{12}\text{: if } 30 < \text{Age} \leq 60 \text{ then AgeClassLabel='between 30 and 60 years old'} \\ r_{13}\text{ :if Age} > 60 \text{ then AgeClassLabel='more than 60 years old'} \end{array} \right\} \right) \\ \left(L_2^{\text{AGENCY 2}}.A\ , \left\{ \begin{array}{l} r_{21}\text{: if AgencyID} \in \{\text{'01903','01905','02256'}\} \text{ then AgencyTypeLabel='student'} \\ r_{22}\text{: if AgencyID='01929' then AgencyTypeLabel='foreigner'} \\ r_{23}\text{: if AgencyID} \notin \{\text{'01903','01905','02256','01929'}\} \text{ then AgencyTypeLabel='classical'} \end{array} \right\} \right) \end{array} \right\}$$

5.2 Concurrent concept formalization

We deal about concurrent needs when two users want to create a new granularity level, based on the same level and attributes with different conditions on these attributes. To take into account the concurrent aspect, we have to introduce the concept of version of granularity level. Note that we just consider here versions, which depend on users. We do not deal with temporal versions.

Definition 7. *Version of granularity level.*

Let be L_i^{sk} the granularity level L_i of the hierarchy H_k of the dimension D_s, which could present different versions defined by different users. A version of this level is noted :

$$L_i^{sk}(vc), \quad c \geq 1$$

where vc denotes the version number.

If there is only one version, we keep the first notation L_i^{sk}.

Example 7. $L_2^{CUSTOMER\ 2}$ corresponds to the `AGE CLASS` level of the dimension `CUSTOMER`. If two users define this level in two different ways, we have to note $L_2^{CUSTOMER\ 2}(v1)$ and $L_2^{CUSTOMER\ 2}(v2)$.

5.3 Rules' constraints formalization

When the ADK is integrated into the data warehouse, the induced data warehouse schema must remain coherent. Aggregation rules must define a partition of the instances of the lower level, where each class will correspond to an instance of the created granularity level. We define in the following two properties that should be satisfied by the defined aggregation rules.

Property 1.

Let be $L_i^{sk}.A$ the generated attribute that characterizes the granularity level L_i of the hierarchy H_k of the dimension D_s.

Given $\mathcal{R}_{i.} = \{r_{ij}, 1 \leq i \leq w, 1 \leq j \leq u\}$ the set of u aggregation rules defining the values of the generated attribute $L_i^{sk}.A$.

Let be $body(r_{ij})$ the *if*-clause of the rule r_{ij}.

Conditions expressed in the *if*-clauses of rules that define the values of a given generated attribute must be mutually disjointed:

$$\forall i, \forall p, q \; such \; as \; p < q, p \in [1, u-1], q \in [2, u],$$

$$body(r_{ip}) \bigcap body(r_{iq}) = \emptyset$$

Property 2.

Let be $L_i^{sk}.A$ the generated attribute that characterizes the granularity level L_i of the hierarchy H_k of the dimension D_s.

Given $\mathcal{R}_{i.} = \{r_{ij}, 1 \leq i \leq w, 1 \leq j \leq u\}$ the set of u aggregation rules defining the values of the generated attribute $L_i^{sk}.A$.

Given val_{ij} the value of $L_i^{sk}.A$ defined in the rule r_{ij}.

Given $Dom_{L_i^{sk}.A}$ the definition domain of $L_i^{sk}.A$.

The domain $Dom_{L_i^{sk}.A}$ has to be covered by the rules, which define the attribute $L_i^{sk}.A$:

$$\forall i, \bigcup_{j=1}^{u} val_{ij} = Dom_{L_i^{sk}.A}$$

6 Implementation

To validate our approach, we have developed a prototype named WEDriK (data Warehouse Evolution Driven by Knowledge), which is developed within the Oracle 10g DBMS. In this section, we give some necessary elements to implement our approach. First, we focus on the model used to manage the data warehouse schema update. Then we detail the different algorithms used to achieve the data warehouse schema evolution.

6.1 Management model for the data warehouse evolution

To manage the schema update of a data warehouse according to our approach, we propose the UML model presented in Figure 7.

A data warehouse is defined as a set of tables. The '`DATA WAREHOUSE`' and '`TABLE`' classes are linked with a composition link. These tables could be dimension or fact tables. Thus a generalization link connects '`FACT`' and '`DIMENSION`' classes to the '`TABLE`' class.

Each dimension table has a '`DIMENSION PRIMARY ID`' and various '`DIMENSION ATTRIBUTES`'. Moreover, fact tables present one or more '`MEASURES`' and a '`PRIMARY ID`', which is a aggregation of foreign keys referencing '`DIMENSION PRIMARY ID`'.

There is an association between '`LEVEL`' and '`DIMENSION`' called '`HIERARCHY`', to represent the fact that each dimension table corresponds to one or more granularity levels that constitute a dimension hierarchy.

Note that each level could be generated by a set of aggregation rules. This particularity is represented by the '`AGGREGATION RULE`' class linked to the '`LEVEL`' class with the label 'Generation'.

This model is not only exploited to manage the data warehouse schema evolution. It is also used to provide users with the updated data warehouse schema. This is very important since the schema evolves continuously. It allows users to know on what they can base their own needs. We choose to represent this visualization under the form of an XML (eXtensible Markup Language) document. Indeed XML allows the users to navigate through the hierarchies of the model and describes correctly the analysis possibilities.

6.2 Data warehouse schema update algorithms

To achieve the data warehouse schema update, we have to consider different algorithms, whose sequence is represented in the Figure 8.

The starting point of the algorithms sequence is a set of rules expressed by a user to create a granularity level. First an algorithm is used to check the validity of the rules (Algorithm 1). If the rules are not correct, the user, has to modify them. This process is repeated until the rules are correct. If they are correct, the addition at the end of a hierarchy or the insertion between two existing successive levels of a new level is made according to the Algorithm 2 (addition) or the Algorithm 3 (insertion), by generating the corresponding queries. When the model of the Section 6.1 is implemented to represent the update of a given

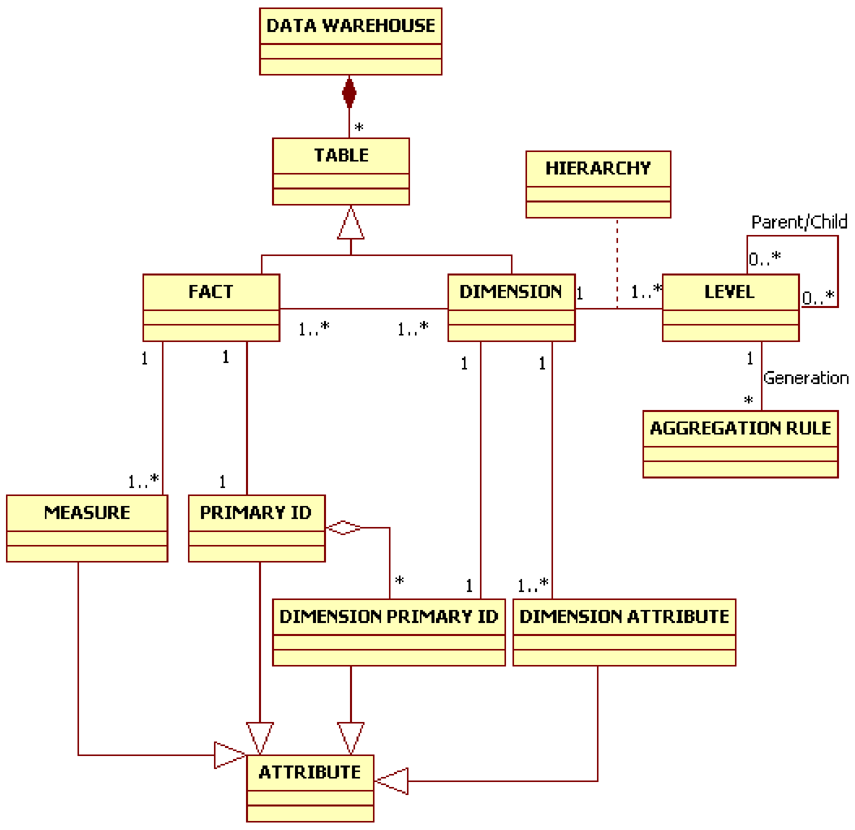


Figure 7: UML management model for the data warehouse evolution

model, the ‘LEVEL’ and ‘AGGREGATION RULE’ classes have to be instanciated. Next we detail the three algorithms.

Constraints checking algorithm. As regards the constraints due to the concept of partition, we check the validity of the rules by exploiting data with queries, according to the Algorithm 1. Let us consider the set of rules R, which determines a new granularity level L' from the lower granularity level L. This

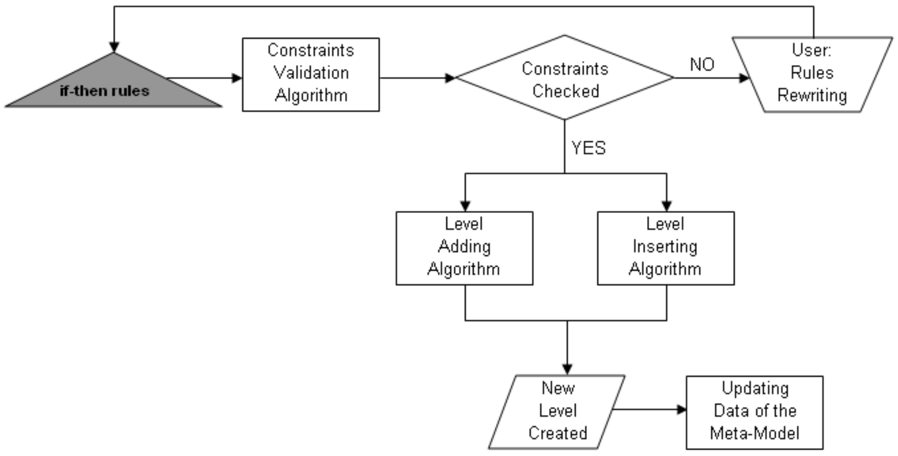


Figure 8: Algorithms sequence

set of rules must satisfy two constraints in order to respect the definition of a partition. First, the conditions expressed in the *if*-clause of the different rules have to be mutually disjointed. Second, each instance of the lower level has its corresponding value in the created level.

For each rule $r \in R$, we write the corresponding query $q \in Q$, which contains in the $WHERE$ clause the *if*-clause of the rule. Each query provides a set of instances of the level L. First, we check that the intersection of all sets considered by pairs is empty. Second, we check that the union of these sets corresponds to the whole instances of the level L.

Level adding algorithm. To create a new granularity level L', from the existing level L, we have to implement an algorithm that allows the addition of a granularity level at the end of a hierarchy, taking into account a set of aggregation rules (Algorithm 2). It consists in creating the corresponding table

Algorithm 1 Checking constraints

```
Require: existing level $L$, set of rules $R = \{r_j, 1 \leq v\}$, body($r_j$) the premise of the rule $r_j$, Tab
    an array
Ensure: Intersection_constraint_checked,Union_constraint_checked
    {Initialization}
 1: Intersection_constraint_checked=true
 2: Union_constraint_checked=true
    {Queries writing}
 3: for j = 1 to v do
 4:     Tab[j]='SELECT * FROM L where body($r_j$)'
 5: end for
    {Checking the intersection of sets induced by rules considered by pair is empty}
 6: for j = 1 to v-1 do
 7:     Q=Tab[j] INTERSECT Tab[j+1]
 8:     if $Q \neq \emptyset$ then
 9:         Intersection_constraint_checked=false
10:     end if
11: end for
    {Checking the union of sets induced by rules corresponds to the level L}
12: Q=Tab[1]
13: for j = 2 to v do
14:     Q=Q UNION Tab[j]
15: end for
16: if $Q \neq$SELECT * FROM L then
17:     Union_constraint_checked=false
18: end if
19: return Intersection_constraint_checked
20: return Union_constraint_checked
```

L' and inserting values by exploiting the rules, and then in linking the new table L' with the existing one L.

Algorithm 2 Adding a new granularity level

```
Require: existing level $L$, generated attribute $A$, $type(L'.A)$ the data type of $L'.A$ , set of rules
    $R = \{r_j, 1 \leq v\}$, body($r_j$) the premise of the rule $r_j$
Ensure: created level $L'$
    {Creating the new granularity level $L'$ and making the aggregation link between $L$ and $L'$}
 1: CREATE TABLE $L'$ ($L'.A$ $type(L'.A)$ AS PRIMARY KEY);
 2: ALTER TABLE $L$ ADD ($A$ $type(L'.A)$);
 3: for all ($r_j \in R$) do
 4:     INSERT INTO $L'$ VALUES ($val$)
 5:     UPDATE $L$ SET $A = val$ WHERE $body(r_j)$
 6: end for
```

Level inserting algorithm. To insert a new granularity level L' between two existing ones ($L1$ and $L2$), we have to implement the Algorithm 3 that allows the insertion of a granularity level between two existing ones, taking into account a set of aggregation rules, which only express the link between the lower level and the created level. We have to implement the same steps as for an addition. Then we have to automatically establish the aggregation link between L' and $L2$, by inferring it according to the one that exists between $L1$ and $L2$. Once this link is established, if the link between $L1$ and $L2$ does not make sense anymore, we can delete it.

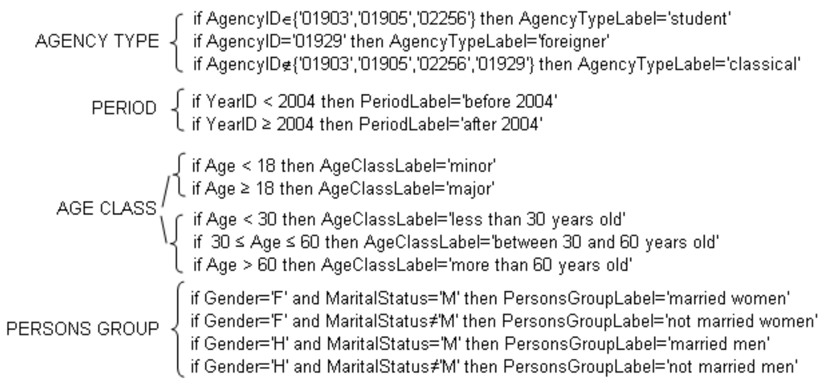
Algorithm 3 Inserting a new granularity level

```
Require: existing lower level $L1$, existing higher level $L2$, attribute linking $L1$ with $L2$ $B$,
    $type(L2.B)$ the data type of $L2.B$, generated attribute $A$, $type(L'.A)$ the data type of $L'.A$ ,
    set of rules $R = \{r_j, 1 \leq v\}$, body($r_j$) the premise of the rule $r_j$
Ensure: inserted level $L'$
    {Creating the new granularity level $L'$ and making the aggregation link between $L1$ and $L'$}
 1: CREATE TABLE $L'$ ($L'.A$ $type(L'.A)$ AS PRIMARY KEY);
 2: ALTER TABLE $L1$ ADD ($A$ $type(L'.A)$);
 3: for all ($r_j \in R$) do
 4:     INSERT INTO $L'$ VALUES ($val$)
 5:     UPDATE $L1$ SET $A = val$ WHERE $body(r_j)$
 6: end for
    {Linking $L'$ with $L2$}
 7: ALTER TABLE $L'$ ADD ($B$ $type(L2.B)$);
 8: UPDATE L' SET B=(SELECT DISTINCT B FROM L1 WHERE $L1.A$=$L'.A$);
```

7 LCL bank case study: a running example

In this section we provide some examples of analysis needs that have emerged in the French bank LCL, and how they meet answers through a running example. We applied our approach on the banking data concerning the NBI. Different analysis needs have emerge. First, since LCL has been bought by another bank in 2004, one manager of LCL needs to compare the NBI before and after 2004. Furthermore, one manager of student accounts needs to obtain analysis according to the type of agencies, since some agencies host only student accounts. In addition, different users need to determine different age classes, to obtain their own analyses. Lastly, the manager who is specialized on targeting customers for products needs to obtain the NBI analysis according to group of customers on the basis of their gender and marital status.

To obtain these analyses, users expressed the rules (Figure 9) used to make evolve the data warehouse schema to allow new analyses by considering new granularity levels such as agency type, age class, period...



AGENCY TYPE
- if AgencyID∈{'01903','01905','02256'} then AgencyTypeLabel='student'
- if AgencyID='01929' then AgencyTypeLabel='foreigner'
- if AgencyID∉{'01903','01905','02256','01929'} then AgencyTypeLabel='classical'

PERIOD
- if YearID < 2004 then PeriodLabel='before 2004'
- if YearID ≥ 2004 then PeriodLabel='after 2004'

AGE CLASS
- if Age < 18 then AgeClassLabel='minor'
- if Age ≥ 18 then AgeClassLabel='major'

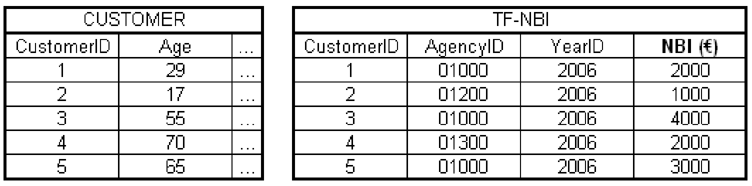
- if Age < 30 then AgeClassLabel='less than 30 years old'
- if 30 ≤ Age ≤ 60 then AgeClassLabel='between 30 and 60 years old'
- if Age > 60 then AgeClassLabel='more than 60 years old'

PERSONS GROUP
- if Gender='F' and MaritalStatus='M' then PersonsGroupLabel='married women'
- if Gender='F' and MaritalStatus≠'M' then PersonsGroupLabel='not married women'
- if Gender='H' and MaritalStatus='M' then PersonsGroupLabel='married men'
- if Gender='H' and MaritalStatus≠'M' then PersonsGroupLabel='not married men'

Figure 9: Aggregation rules expressed by users for the NBI analysis

In order to illustrate the usage of WEDriK, let us focus on the motivating example of the Section 3 where users need to build age classes starting from the ages of the table `CUSTOMER`, in a concurrent way. What we seek to do is providing an example to show step by step how to achieve these analyses.

Let us consider the samples of the tables `CUSTOMER` and `TF-NBI` (Figure 10).



CUSTOMER		
CustomerID	Age	...
1	29	...
2	17	...
3	55	...
4	70	...
5	65	...

TF-NBI			
CustomerID	AgencyID	YearID	**NBI (€)**
1	01000	2006	2000
2	01200	2006	1000
3	01000	2006	4000
4	01300	2006	2000
5	01000	2006	3000

Figure 10: Samples of tables

The following rules have been defined by two users during the *acquisition phase*:

User 1:

$(R1)\ if\ Age < 18\ then\ AgeClassLabel\ =$ 'minor'

$(R2)\ if\ Age \geq 18\ then\ AgeClassLabel\ =$ 'major'

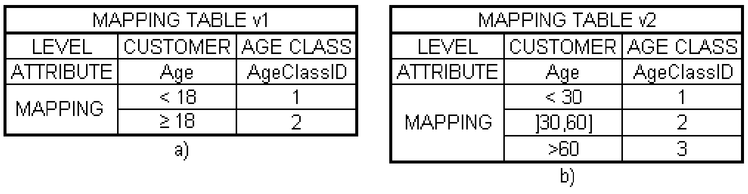
User 2:

$(R3)\ if\ Age \leq 30\ then\ AgeClassLabel\ =$ 'less than 30 years old'

$(R4)\ if\ 30 < Age \leq 60\ then\ AgeClassLabel\ =$ 'between 30 and 60 years old'

$(R5)\ if\ Age > 60\ then\ AgeClassLabel =$ 'more than 60 years old'

The *integration phase* exploits these rules to generate two mapping tables (Figure 11).



MAPPING TABLE v1		
LEVEL	CUSTOMER	AGE CLASS
ATTRIBUTE	Age	AgeClassID
MAPPING	< 18	1
	≥ 18	2

a)

MAPPING TABLE v2		
LEVEL	CUSTOMER	AGE CLASS
ATTRIBUTE	Age	AgeClassID
MAPPING	< 30	1
	]30,60]	2
	>60	3

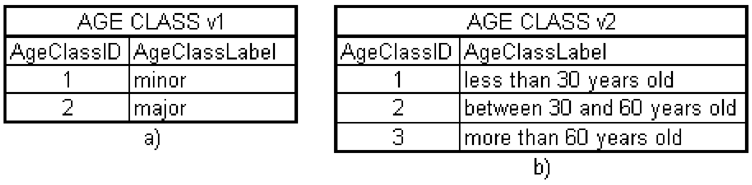
b)

Figure 11: Mapping tables for the `AGE CLASS` level

Then the *evolution phase* allows to create two versions of the `AGE CLASS` granularity level: `AGE CLASS_v1` and `AGE CLASS_v2` tables (Figure 12).

The `CUSTOMER` table is updated to be linked with the two versions of `AGE CLASS` level, adding `AgeClassv1` and `AgeClassv2` attributes (Figure 13).

Finally, the *analysis phase* allows to exploit the model with new analysis axes. Usually, in an OLAP environment, queries require the computation of



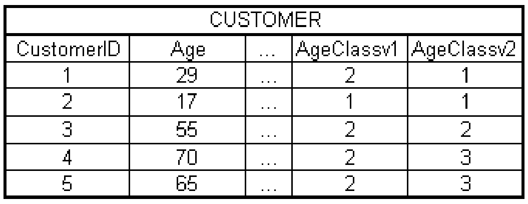
AGE CLASS v1	
AgeClassID	AgeClassLabel
1	minor
2	major

a)

AGE CLASS v2	
AgeClassID	AgeClassLabel
1	less than 30 years old
2	between 30 and 60 years old
3	more than 60 years old

b)

Figure 12: Two versions of the `AGE CLASS` level



CUSTOMER				
CustomerID	Age	...	AgeClassv1	AgeClassv2
1	29	...	2	1
2	17	...	1	1
3	55	...	2	2
4	70	...	2	3
5	65	...	2	3

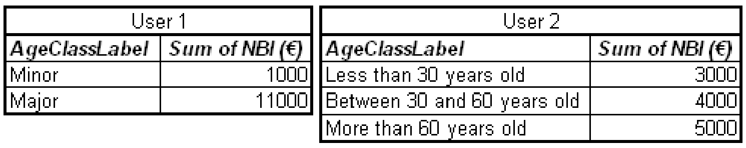
Figure 13: Customer table including links with versions of `AGE CLASS` level

aggregates over base fact tables. In the presence of dimensions with hierarchies of levels, queries computing aggregates over various dimension levels are often required. Indeed, given a data warehouse schema, the analysis process allows to summarize data by using (1) aggregation operators such as `SUM` and (2) `GROUP BY` clauses. In our case, each user needs its own analysis, more precisely the sum of NBI according to the age classes he has defined. The corresponding queries are:

User 1:
```
SELECT AgeClassLabel, SUM(NBI)
FROM TF-NBI, CUSTOMER, AGE CLASS v1
WHERE TF-NBI.CustomerID=CUSTOMER.CustomerID
AND CUSTOMER.AgeClassv1=AGECLASSv1.AgeClassID ;
GROUP BY AgeClassLabel ;
```

User 2:
```
SELECT AgeClassLabel, SUM(NBI)
FROM TF-NBI, CUSTOMER, AGE CLASS v2
WHERE TF-NBI.CustomerID=CUSTOMER.CustomerID
AND CUSTOMER.AgeClassv2=AGECLASSv2.AgeClassID ;
GROUP BY AgeClassLabel ;
```

The results are provided in Figure 14.



User 1

AgeClassLabel	*Sum of NBI (€)*
Minor	1000
Major	11000

User 2

AgeClassLabel	*Sum of NBI (€)*
Less than 30 years old	3000
Between 30 and 60 years old	4000
More than 60 years old	5000

Figure 14: Analysis results for NBI according age classes

8 Conclusion and future work

We aimed in this paper at involving users in the data warehouse evolution process to provide an answer to their own analysis needs with a global approach. This approach is based on three key ideas. First, a data warehouse allows to meet global analysis needs. Second, users' own knowledge induce new analysis needs. Three, integrating users' analysis needs and knowledge is necessary to make the data warehouse schema evolve. For this end, we defined the ADK concept, which represents the specific users' knowledge, which aggregates data from one granularity level to another. The ADK is represented in the form of "if-then" rules and is used to dynamically create new granularity levels. Thus, we enrich data warehouse with new analysis possibilities by integrating individual users' ADK into the data warehouse. It increases the interaction between the users and the decision support system. This approach is composed of four phases: (1) users' knowledge acquisition, (2) knowledge integration, (3) data warehouse schema update, and (4) on-line analysis. Our approach is supported by a data warehouse model based on "if-then" rules (*R-DW*), where rules allow us to introduce users' knowledge into the data warehouse for new analysis purposes. The *R-DW* model presents the advantage of evolving incrementally according to the user's knowledge, answering to emergent individual analysis needs, without administrator's intervention, in a concurrent way. We proposed a formalization of the *R-DW* model to address the question of how to involve users in the data warehouse schema update. Moreover, we developed an implementation that follows our proposals to validate our approach. Our prototype is implemented within the Oracle 10g DBMS and we validated our approach through the LCL case study.

To the best of our knowledge, the idea of user-defined evolution of dimensions (based on analysis needs) is novel; there are still many practical aspects to be explored, since our paper provides a formal foundation and a working implementation to be used as a basis for future work. Thus, the perspectives opened by this study are numerous. First of all, we have to finalize our prototype to insert a level and apply the necessary propagation according to the proposed algorithm. Moreover, we have to consider other evolution operations such as the deletion of a level. When the deletion concerns a level which is between two others, we have to study the necessary updates to maintain the data warehouse coherent. Concerning the rules' expression, we precise that rules are used for determining the aggregation link between instances. We think that it would be interesting to define meta-rules that define this link in a structural way. Moreover, we intend to study the performance of our approach in terms of storage space, response time and algorithms complexity. Then, we would like to use non supervised learning methods on dimension data to determine the classes of the partition to build higher granularity levels. Besides, some future work consist in studying how individual users' needs evolve in time, and thus we have to consider different strategies of rules updating and temporal versioning. Furthermore, we have to consider that an analysis need could necessitate the definition of a new dimension, which would modify the fact table. It could also imply an evolution of measures, such as the definition of derived measures based on existing ones. Finally, we would like to extend our approach on a conceptual level. The key idea is to be able to define a data warehouse schema by taking into account both data sources and analysis needs. And then, the objective is (1) to define a way to express a change in data sources or in analysis needs, (2) to evaluate the impact of this change and (3) to carry out the required evolutions in the data warehouse. Thus the approach presented in this paper would be a part of a more global evolution approach.

9 Acknowledgments

The authors would like to thank Jean-François Gaucher, Baptiste Jouaneton, Moufida Khoualdia, Julie Malaret and Ony Rakotoarivelo for their contribution to WEDriK implementation; as well as this paper's reviewers.

References

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

[2] Z Bellahsene, *Schema Evolution in Data Warehouses*, Knowledge and Information Systems **4** (2002), no. 3, 283–304.

[3] L Bellatreche, A Giacometti, P Marcel, H Mouloudi, and D Laurent, *A Personalization Framework for OLAP Queries*, VIIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 05), Bremen, Germany, ACM Press, 2005, pp. 9–18.

[4] M Blaschka, *FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems*, 6th Doctoral Consortium, 1999.

[5] M Blaschka, C Sapia, and G Höfling, *On Schema Evolution in Multidimensional Databases*, Ist International Conference on Data Warehousing

and Knowledge Discovery (DaWaK 99), Florence, Italy, LNCS, vol. 1676, Springer, 1999, pp. 153–164.

[6] R Bliujute, S Saltenis, G Slivinskas, and C Jensen, *Systematic Change Management in Dimensional Data Warehousing*, IIIrd International Baltic Workshop on Databases and Information Systems, Riga, Latvia, 1998, pp. 27–41.

[7] M Body, M Miquel, Y Bédard, and A Tchounikine, *A Multidimensional and Multiversion Structure for OLAP Applications*, Vth ACM International Workshop on Data Warehousing and OLAP (DOLAP 02), McLean, Virginia, USA, 2002, pp. 1–6.

[8] A Bonifati, F Cattaneo, S Ceri, A Fuggetta, and S Paraboschi, *Designing Data Marts for Data Warehouses*, ACM Transactions on Software Engineering and Methodology **10** (2001), no. 4, 452–483.

[9] D R Brown, M R Cutkosky, and J M Tenenbaum, *Next-Cut: A Second Generation Framework for Concurrent Engineering*, MIT-JSME Workshop, 1989, pp. 8–25.

[10] F Carpani and R Ruggia, *An Integrity Constraints Language for a Conceptual Multidimensional Data Model*, XIIIrd International Conference on Software Engineering Knowledge Engineering (SEKE 01), Buenos Aires, Argentina, 2001.

[11] S Chaudhuri and U Dayal, *An Overview of Data Warehousing and OLAP Technology*, SIGMOD Record **26** (1997), no. 1, 65–74.

[12] M M Espil and A A Vaisman, *Efficient Intensional Redefinition of Aggregation Hierarchies in Multidimensional Databases*, IVth ACM International Workshop on Data Warehousing and OLAP (DOLAP 01), Atlanta, Georgia, USA, 2001.

[13] C Favre, F Bentayeb, and O Boussaid, *A Knowledge-driven Data Warehouse Model for Analysis Evolution*, XIIIth International Conference on Concurrent Engineering: Research and Applications (CE 06), Antibes, France, Frontiers in Artificial Intelligence and Applications, vol. 143, IOS Press, 2006, pp. 271–278.

[14] U M Fayyad, G Piatetsky-Shapiro, and P Smyth, *Knowledge discovery and data mining: Towards a unifying framework*, IInd International Conference on Knowledge Discovery and Data Mining (KDD 96), Portland, Oregon, USA, AAAI Press, 1996, pp. 82–88.

[15] F Ghozzi, F Ravat, O Teste, and G Zurfluh, *Constraints and Multidimensional Databases*, Vth International Conference on Enterprise Information Systems (ICEIS 03), Angers, France, 2003, pp. 104–111.

[16] M Golfarelli, D Maio, and S Rizzi, *Conceptual Design of Data Warehouses from E/R Schemes*, XXXIst Annual Hawaii International Conference on System Sciences (HICSS 98), Big Island, Hawaii, USA, vol. 7, 1998, pp. 334–343.

[17] J H Holland, K J Holyoak, R E Nisbett, and P R Thagard, *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, 1986.

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

[19] C A Hurtado, A O Mendelzon, and AA Vaisman, *Updating OLAP Dimensions*, IInd ACM International Workshop on Data Warehousing and OLAP (DOLAP 99), Kansas City, Missouri, USA, ACM Press, 1999, pp. 60–66.

[20] W H Inmon, *Building the Data Warehouse*, Third ed., John Wiley & Sons, 2002.

[21] H J Kim, T H Lee, S G Lee, and J Chun, *Automated Data Warehousing for Rule-Based CRM Systems*, XIVth Australasian Database Conference on Database Technologies, Adelaide, Australia, 2003, pp. 67–73.

[22] R Kimball, *The Data Warehouse Toolkit*, John Wiley & Sons, 1996.

[23] E Malinowski and E Zimányi, *OLAP Hierarchies: A Conceptual Perspective*, XVIth International Conference on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia, LNCS, vol. 3084, Springer, 2004, pp. 477–491.

[24] A O Mendelzon, C A Hurtado, and D Lemire, *Data Warehousing and OLAP: A Research-Oriented Bibliography*, Available at http://www.daniel-lemire.com/OLAP/, 2006.

[25] A O Mendelzon and A A Vaisman, *Temporal Queries in OLAP*, XXVIth International Conference on Very Large Data Bases (VLDB 00), Cairo, Egypt, Morgan Kaufmann, 2000, pp. 242–253.

[26] T Morzy and R Wrembel, *On Querying Versions of Multiversion Data Warehouse*, VIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 04), Washington, District of Columbia, USA, ACM Press, 2004, pp. 92–101.

[27] A Nabli, A Soussi, J Feki, H Ben-Abdallah, and F Gargouri, *Towards an Automatic Data Mart Design*, VIIth International Conference on Enterprise Information Systems (ICEIS 05), Miami, Florida, USA, 2005, pp. 226–231.

[28] T B Pedersen, C S Jensen, and C E Dyreson, *A Foundation for Capturing and Querying Complex Multidimensional Data*, Information Systems **26** (2001), no. 5, 383–423.

[29] V Peralta, A Illarze, and R Ruggia, *On the Applicability of Rules to Automate Data Warehouse Logical Design*, XVth Conference on Advanced Information Systems Engineering (CAiSE 03), Klagenfurt/Velden, Austria, CEUR Workshop Proceedings, vol. 74, CEUR-WS.org, 2003.

[30] C Phipps and K C Davis, *Automating Data Warehouse Conceptual Schema Design and Evaluation*, IVth International Workshop on Design and Management of Data Warehouses (DMDW 02), Toronto, Canada, CEUR Workshop Proceedings, vol. 58, CEUR-WS.org, 2002, pp. 23–32.

[31] R Missaoui, G Jatteau, A Boujenou and S Naouali, *Towards Integrating Data Warehousing with Data Mining Techniques*, Data Warehouses and OLAP: Concepts, Architectures and Solutions, Idea Group, 2006.

[32] X Zhang, L Ding, and E A Rundensteiner, *Parallel Multisource View Maintenance*, The VLDB Journal **13** (2004), no. 1, 22–48.

[33] Y Zhuge, H Garcia-Molina, and J L Wiener, *Consistency Algorithms for Multi-Source Warehouse View Maintenance*, Distributed and Parallel Databases **6** (1998), no. 1, 7–40.