Integrated vision of federated data warehouses

Michel Schneider[1]

[1] LIMOS, Blaise Pascal University, Complexe des Cézeaux
63173 Aubière, France
schneider@isima.fr

Abstract. The notion of Federated Data Warehouse Architecture was suggested for various reasons: bigger autonomy of divisions in an organization, better adaptation to user needs, bigger efficiency of treatment. However this architecture puts the problem of cooperation between its constituents. Because of the heterogeneities which can exist, the simultaneous use of constituents can be made difficult. In this paper we propose a process to build an integrated vision. This integrated vision permits the simultaneous exploitation of the constituents. It is based on a generic model to describe the structure of data warehouses. A relational implementation is suggested.

Keywords: federation, data warehouse, generic model, integrated view, relational implementation

1 Introduction

A federated data warehouse architecture consists in a set of data warehouses which operate in a semi-autonomous way. These warehouses are generally organized separately and scattered geographically. But they can cooperate to contribute to a common objective. A federated architecture is generally suggested for ensuring a better correspondence with the structure of the company and to supply the necessary autonomy for the decentralized units. It agrees particularly when these units are installed in different countries. Every warehouse can then respect more easily the constraints of the country and the customs of the local partners (customers, suppliers, …). But too much autonomy risks to make cooperation between warehouses more difficult. It is necessary to maintain a minimum of cohesion, particularly as regards the nature of facts and the organization of the dimensions of analysis. For example, if one wishes at the level of the company to make analyses on sales, it is necessary that the various federated warehouses adopt some common rules for characterizing a sale (quantity sold in the day for a product, for a customer, for a shop) and for organizing the analysis criteria which are going to appear in dimensions (category of a product, address of a customer, …). The autonomy granted to the administrator of a warehouse can lead to different schemas based on not comparable elements.

This notion is relatively recent and there are few works on the subject. Main publications are relative to the interest which this kind of architecture can present for the management of a company [2,7]. One notes also investigations on the architecture itself, on its constituents and their functions [3,6,8]. The main principles are similar to those of federated data bases [10]. A hierarchic organisation is generally recommended. The importance of knowledge management is also underlined [5].

Our objective in this paper is to study the integration problems which are posed by a set of federated warehouses. We will suppose that these warehouses share common elements relative to facts and to dimensions. It is a question then of determining how to recognize these elements and how to use them to build a multi-dimensional structure which corresponds to the largest common schema. The interest of this common schema can be situated in two directions. The first one corresponds to the notion of mediation: OLAP requests are expressed on the common schema and then rewritten for the initial structures of the warehouses. Every warehouse is then interrogated separately. The results are integrated. The other one corresponds to the notion of view. The common schema is then defined by a view which can be directly interrogated. This view can be virtual or materialized. We will study this last way and we will suggest a relational approach to put it into practice.

To formalize the study we will use the generic model for warehouses which we presented in [9]. We will see indeed that this model is well suited to put in evidence the common schema.

The paper is organized as follows: sections 2 presents our generic model for data warehouses, section 3 explains the construction of the integrated view, section 4 concludes and gives some perspectives.

2 Our model for data warehouse structures

2.1 Fact type

A fact is used to record measures or states concerning an event or a situation. Measures and states can be analysed through different criteria organized in dimensions.

A fact type has the following structure :

fact_name[(fact_key), (list_of_reference_attributes), (list_of_fact_attributes)]

where

- fact_name is the name of the type;
- fact_key is a list of attribute names; the concatenation of these attributes identifies each instance of the type;
- list_of_reference_attributes is a list of attribute names; each attribute references a member in a dimension or another fact instance;
- list_of_fact_attributes is a list of attribute names; each attribute is a measure for the fact.

Each fact attribute can be analysed along each of the referenced dimensions. Analysis is achieved through the computing of aggregate functions on the values of this attribute.

Example 1. As an example let us consider the following fact type for memorizing the sales in a set of stores.

Sales[(ticket_number, product_key), (time_key, product_key, store_key),
(price_per_unit, quantity)]

The key is (ticket_number, product_key). This means that there is an instance of Sales for each different product of a ticket. There are three dimension references : time_key, product_key, store_key. There are two fact attributes : price_per_unit, quantity. The fact attributes can be analysed through aggregate operations by using the three dimensions.

There may be no fact attribute; in this case a fact records the occurrence of an event or a situation. In such cases, analysis consists in counting occurrences satisfying a certain number of conditions.

For the needs of an application, it is possible to introduce different fact types sharing certain dimensions and having references between them.

Two dimensions are independent if there is no relationship between a member of the first and a member of the second.

A dimension is degenerated in a fact type if its reference attribute is replaced by a value attribute. In other words the analysis is achieved by direct use of the values of this attribute.

2.2 Member type

Member of a dimension. The different criteria which are needed to conduct analysis along a dimension are introduced through members. A member is a specific attribute (or a group of attributes as we will see later) taking its values on a well defined domain. For example, the dimension TIME can include members such as DAY, MONTH, YEAR, … . Analysing a fact attribute A along a member M means that we are interested in computing aggregate functions on the values of A for any grouping defined by the values of M. In the paper we will also use the notation M_{ij} for the j-th member of i-th dimension.

Organization of members. Members of a dimension are generally organized into hierarchies which are conceptual representations of the hierarchies of their occurrences. Hierarchies in dimensions is a very useful concept that can be used to impose constraints on member values and to guide the analysis. Hierarchies of occurrences result from various relationships which can exist in the real world: categorization, membership of a subset, mereology.

We will model these hierarchies according to a hierarchical relationship (HR) which links a child member M_{ij} (i.e. week) to a parent member M_{ik} (i.e. year) and we will use the notation $M_{ij} \rightarrow M_{ik}$. For the following we consider only situations where a child occurrence is linked to a unique parent occurrence in a type. However, a child occurrence, as in case (b) or (c), can have several parent occurrences but each of different types. We will also suppose that HR is reflexive, antisymmetric and transitive. This kind of relationship covers the great majority of real situations [1]. Existence of this HR is very important since it means that the members of a dimension can be organized into levels and correct aggregation of fact attribute values along levels can be guaranteed.

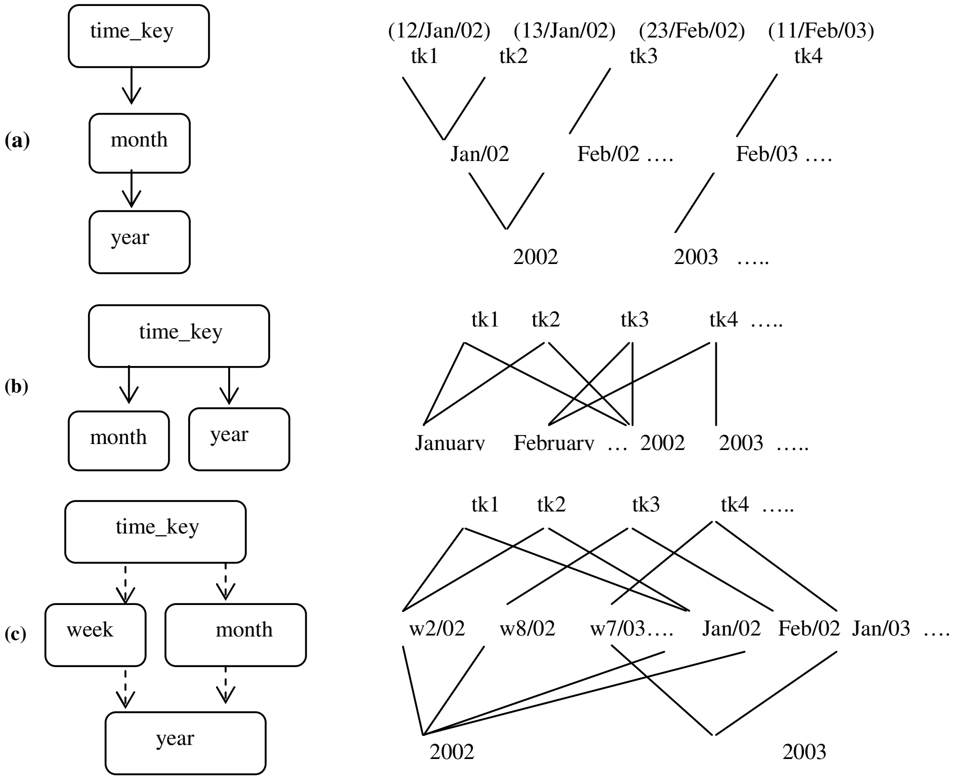
(a)
time_key
month
year
(12/Jan/02) (13/Jan/02) (23/Feb/02) (11/Feb/03)
tk1 tk2 tk3 tk4
Jan/02 Feb/02 …. Feb/03 ….
2002 2003 …..
(b)
time_key
month
year
tk1 tk2 tk3 tk4 …..
Januarv Februarv … 2002 2003 …..
(c)
time_key
week
month
year
tk1 tk2 tk3 tk4 …..
w2/02 w8/02 w7/03…. Jan/02 Feb/02 Jan/03 ….
2002 2003

Fig. 1. Different hierarchies in dimensions.

Example 2. Figure 1 illustrates different cases of hierarchies we can encounter into dimensions. In case (a) the dimension has a unique hierarchy. Case (b) depicts a similar situation with two separate hierarchies. In case (c) there exist two alternative (and exclusive) hierarchies. We mark them by using dotted arrows.

Aggregation levels in a well-formed dimension. Members in a dimension can be distributed into levels. Each level represents a level of aggregation. Each time we follow a directed edge, the level increases (by one or more depending on the used path). This action corresponds to a ROLLUP operation (corresponding to the semantics of the HR) and the opposite operation to a DRILL DOWN. Starting from the reference to a dimension D in a fact type F, we can then roll up in the hierarchy of dimension D by following a path of the cover graph of D.

Entries and roots in a dimension. Each member of a dimension can be an entry for this dimension i.e. can be referenced from a fact type. This possibility is very important since it means that dimensions between several fact types can be shared in various ways. In particular, it is possible to reference a dimension at different levels of granularity. A dimension can have several roots. A root represents a standard entry in a dimension. For the three dimensions in figure 1, there is a single root.

Property attributes in a dimension. As in other studies [4], we consider property attributes in a dimension which is used to describe the members. A property attribute is linked to its member through a functional dependence, but does not introduce a new member and a new level of aggregation. For example a member *town* in a dimension may have property attributes such as population, administrative position, … . Such an attribute can be used in the selection predicates of requests to filter certain groups.

Member type. We now define the notion of member type, which incorporates the different features presented above. A member type has the following structure:

member_name[(member_key), (list_of_reference_attributes),
(list_of_property_attributes)]

where

- member_name is the name of the type;
- member_key is a list of attribute names; the concatenation of these attributes identifies each instance of the type;
- list_of_reference_attributes is a list of attribute names where each attribute is a reference to the successors of the member instance in the cover graph of the dimension; alternatives are represented by using a sub-list in nested parentheses;
- list_of_property_attributes is a list of attribute names where each attribute is a property for the member.

Only the member_key is mandatory.

Example 3. Using this model, the representation of the members of the dimension represented in figure 1(c) is the following:

time_root[(time_key), ((week_key, month_key)), ()]
week[(week_key), (year_key), (week_type)]
month[(month_key), (year_key), ()]
year[(year_key), (), (year_type)]

Note that the two reference attributes week_key and month_key are represented in a sublist since they are the origin of two alternative paths.

Here is an occurrence of the week type :

week [(w1_03), (2003), (holiday)].

The property attribute holiday can take the value yes or no.

2.3 Modelling different warehouse structures

Fact types and member types can be interconnected in order to model various warehouse structures.

First, a fact can directly reference any member of a dimension. Usually a dimension is referenced through one of its roots (as we saw above, a dimension can have several roots). But it is also interesting and useful to have references to members other than the roots. This means that a dimension can be used by different facts with different granularities. For example, a fact can directly reference *town* in the *customer* dimension and another can directly reference *region* in the same dimension. This second reference corresponds to a coarser granule of analysis than the first.

Moreover, a fact F_1 can reference any other fact F_2. This means that a fact attribute of F_1 can be analysed by using the key of F_2 (acting as the grouping attribute of a normal member) and also by using the dimensions referenced by F_2.

Figure 2 illustrates the typical structures we want to model. Case (a) corresponds to the simple case, also known as star structure, where there is a unique fact type F_1 and several separate dimensions D_1, D_2, … . Cases (b) and (c) correspond to the notion of facts of fact. Cases (d), (e) and (f) correspond to the sharing of a dimension.

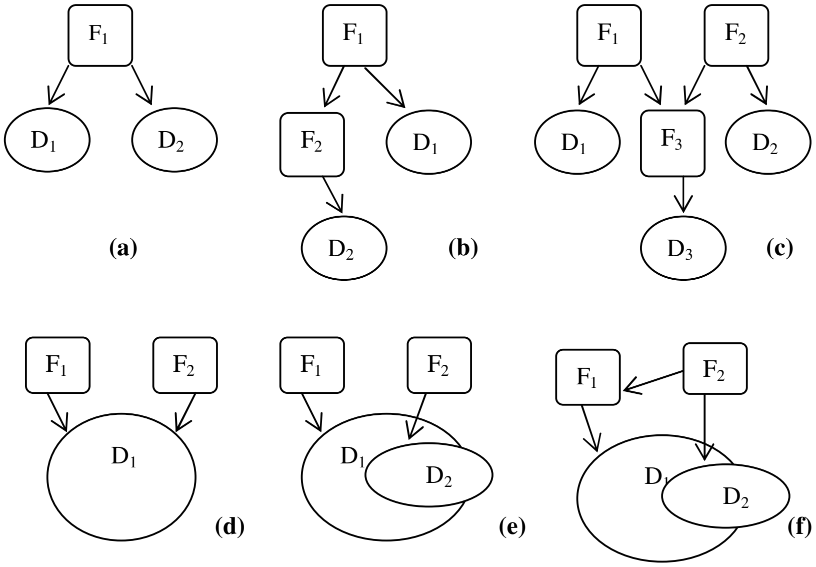

Fig. 2. Typical warehouse structures.

In figure 3, we have represented a simple case based on a star-snowflake structure. In this case, there is a unique fact type, each dimension has a unique root; each reference in the fact type points towards the root of a dimension. Note that our model does not differentiate star structures from snowflake structures. The difference will appear with the mapping towards the relational model.

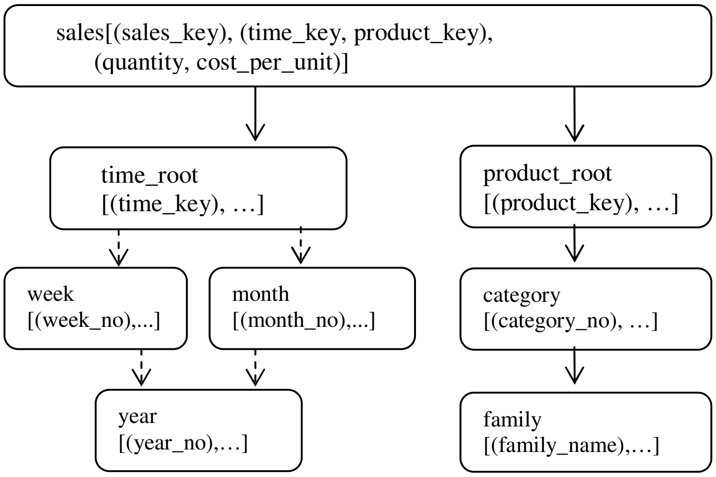
sales[(sales_key), (time_key, product_key), (quantity, cost_per_unit)]
time_root [(time_key), …]
product_root [(product_key), …]
week [(week_no),...]
month [(month_no),...]
category [(category_no), …]
year [(year_no),…]
family [(family_name),…]

Fig. 3. A star-snowflake structure.

2.4 Mapping to the relational model

One way to implement warehouses is to use relational DBMS. So it is necessary to be able to map our well-formed structures in accordance with the relational model. In this section we provide a certain number of guidelines for this mapping.

Relational mapping with split dimensions. This solution, which is straightforward, consists in mapping each type P_i into a table T_i. The key of P_i becomes the primary key of T_i. References between types are implemented via foreign keys. This solution offers a simple way to memorize precalculated aggregates by adding supplementary attributes in element types. Its drawback is well-known: navigating through the hierarchy necessitates many joins which can burden the performances. For a structure like the one described in figure 3, this solution leads to the relational snowflake warehouse structure as represented below (primary keys are marked in bold, foreign keys in italics).

sales(**sales_key**, *time_key*, *product_key*, quantity, cost_per_unit)
time_root(**time_key**, *week_no*, *month_no*, …)
week(**week_no**, *year_no*, …)
month(**month_no**, *year_no*, …)
year(**year-no**, …)
product_root(**product_key**, *category_no*,…)
category(**category_no**, *family_name, …*)
family(**family_name**, …)

Relational mapping with regrouped dimensions. This solution is only possible when the DWG has a unique root. First the root type is mapped into a specific table. Then we create a number of tables equal to the number of references in the root type.

All the elements which can be reached from one reference are grouped in the same table.

Hybrid relational mapping. The previous mapping is not possible when an element can be reached from different roots. This is because this element acts as an entry and must be the key of a table in order to install the references correctly. The hybrid mapping thus consists in inserting each element entry into a specific table. Elements which are accessible only from one entry can be stored in the table of this entry. Others must be stored in separate tables.

3 Integrated view on federated data warehouses

3.1 Illustrating the problem through an example

Let us suppose that the structure of figure 3 is used by an agency (agency A) of a company. Let us consider now another structure (figure 4) used by another agency (agency B) of the same company. For the B agency, sales are booked monthly. In other words, quantity in the type sales_per_month represents the quantity of a product sold over the referenced month. If the products referenced in the two structures are the same, one can wish at a more global level of the company (sales department) used simultaneously the two structures. It would be so interesting to be able to have an integrated vision of these two structures.

The integration of these two structures supposes the identification of the following problems:

-The similarity of facts to be analyzed (here quantity of sold products)

-The different granularities of facts in the two structures (in the structure A the granularity of quantity is the day, while in the structure B the granularity of quantity is the month; it will so be necessary to make a preliminary aggregation in the structure A before being able to make the integration)

-The similarity of some members in the dimensions.

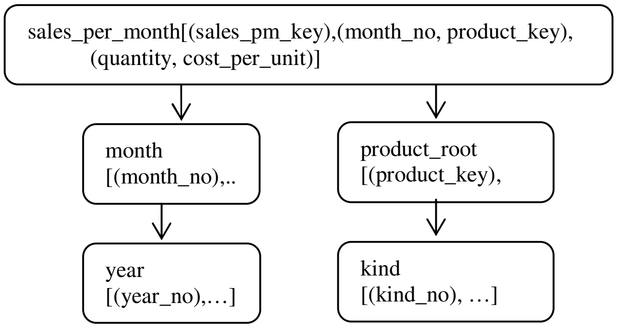
sales_per_month[(sales_pm_key),(month_no, product_key),
(quantity, cost_per_unit)]
month
[(month_no),..
product_root
[(product_key),
year
[(year_no),…]
kind
[(kind_no), …]

Fig. 4. Another data warehouse structure for sales

3.2 Principle of a process to construct an integrated view

In this study we will consider only star-snowflake structures.

An attribute α of a type (fact or member) of the structure A is said to be a synonym of an attribute β of the structure B if the semantic of these two attributes are similar and if their value domains are identical. The automatic determination of this synonymy puts various problems. We suppose that the naming of the types and the attributes respects an ontology of the domain in order to facilitate this treatment.

We propose then a process to build an integrated vision R of two structures A and B of data warehouses.

Stage 1: Introduce into the fact type of R the fact attributes of A which have a synonym in B. If there is no attribute satisfying this condition then structure R is empty and the process ends.

Stage 2: Consider the key attribute of the first member DA_{11} of the first dimension DA_1 of A. Search in B a member DB_{kj} of a dimension DB_k which is synonym to DA_{11}. Introduce then into the first dimension of R, a first member type with this key attribute. Repeat as long as possible by pursuing the search for synonyms on the two dimensions DA_{11} and DB_{kj}. Every time when a synonym is found, connect the new member with the member previously created in R for this dimension. If for the dimension DA_{11} of A there is no synonymic member in B, then this dimension does not appear in R. Repeat then on all the dimensions of A.

Stage 3: For each fact attribute of R determine its granularities with regard to the two fact attributes which are synonym in A and B. Determine so the necessary aggregations to be operated on these fact attributes in A and B (note that this treatment can be made only when the multidimensional structure of R is established according to stage 2).

Stage 4: In the fact type of R, merge the instances of the fact types in A and B after having made the aggregation operations detected in stage 3.

Stage 5: Project the instances of each hierarchy in a dimension of A and B on the corresponding hierarchy in R. Merge these instances in R. Corresponding instances coming from A and B must be identical (for example the same value of product_key in A and B must be connected with the same value of category_no and kind_no). If this condition is not satisfied make the necessary corrections before merging.

Stage 6: Connect each instance of the fact type in R to the appropriate instances of the dimension roots of R.

Remark 1: The coherence condition of stage 4 can, in some cases, induces difficult problems, in particular when numerous errors occur. These cases can result from an inappropriate treatment of the synonymies. It is possible also that the two structures A and B and their instances do not share a common approach for modelling facts and dimensions. It will be then preferable to cancel the integration process and to reconsider the organization of structures A and B.

Remark 2: The integration of more than two structures can be made recursively by integrating the two first structures, then by integrating the result with the third structure and so on.

3.3 Application of the process to the example

For the two structures of figures 3 and 4 we can establish the following synonyms:

(1) A.time_key synonym B.time_key
(2) A.month_no synonym B.month_no
(3) A.year_no synonym B.year_no
(4) A.product_key synonym B.product_key
(5) A.category_no synonym B.kind_no
(6) A.quantity synonym B.quantity
(7) A.cost_per_unit synonym B.cost_per_unit

According to these synonyms, the integrated view R (figure 5) has the same structure as the one of figure 4. For the names in R, we can use those coming from A, or those coming from B, or a mix. We can also introduce new names according to choices coming from the user.

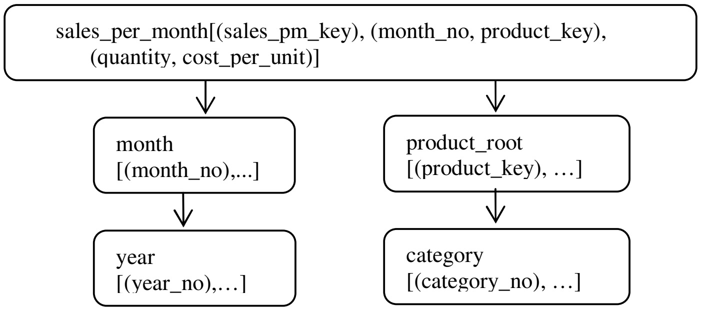

Fig. 5. The integrated view

It appears, on the basis of this multidimensional structure, that it is necessary to operate an aggregation on the tuples coming from structure A. We have to make a grouping for every month. For the attribute quantity, the aggregation is made with the function SUM. For the attribute cost_per_unit, it is necessary to calculate an average value over the month by taking into account the sold quantities. These operations are formalized by the following rules :

(8) replace A.quantity by SUM(A.quantity) GROUP BY (A.month_no)
(9) replace A.cost_per_unit by SUM(A.quantity * A.cost_per_unit)
/SUM(A.quantity) GROUP By (A.month_no)

3.4 Implementation of an integrated view

When a relational implementation is possible, an integrated view can be easily constructed by using a virtual SQL view. We illustrate this solution for the integrated view of figure 5.

We suppose that the structures of figures 3 and 4 are implemented according to the relational mapping with split dimensions. The mapping of the structure in figure 3 has been already given in section 2.4. The mapping of the structure in figure 4 is the following :

sales_per_month(**sales_pm_key**, *month_no*, *product_key*, quantity, cost_per_unit)
month(**month_no**, *year_no*, …)
year(**year-no**, …)
product_root(**product_key**, *category_no*,…)
kind(**kind_no**, …)

The SQL statement which constructs the virtual integrated view is thus the following:

```
    CREATE VIEW sales_per_month(month_no, year_no, product_key, category_no, quantity,
cost_per_unit) AS
    (SELECT A_month.month_no, A_year.year_no, A_product_root.product_key,
A_category.category_no, …, sum(quantity), sum(cost_per_unit*quantity)/sum(quantity)
FROM A_sales, A.time_root, A_month, A_year, A_product_root, A_category WHERE
A_sales.time_key = A_time_root.time_key and A_time_root.month_no = A_month.month_no
and A_month.year_no = A_year.year_no and A_sales.product_key =
A_product_root.product_key and A_product_root.category_no = A_category.category_no
GROUP BY A_month.month_no
    UNION
    SELECT B_month.month_no, B_year.year_no, B_sales_per_month.product_key,
B;kind_no, …, quantity, cost_per_unit FROM B_sales_per_month, B_month, B_year,
B_product_root, B_kind WHERE B_sales_per_month.month_no = B_month.month_no and
B_month.year_no = B_year.year_no and B_sales_per_month.product_key =
B_product_root.product_key and B_product_root.kind_no = A_kind.kind_no)
```

Note that rules (1) to (9) provides all the information to generate automatically this statement. We have not considered the generation of a primary key which necessitates a more elaborated code.

Using this view a final user can now make different analyses on the fact attribute quantity which integrates the quantity per month for each product sold into the two agencies. In the view, we can have only one tuple for a product and a month coming from one of the two agencies or two tuples coming from the two agencies.

4 Conclusion

In this paper we studied the integration problems which puts a set of federated warehouses and we suggested a process to build an integrated view on these warehouses.

We defined the integrated view as the largest common schema. The instances of the view result then from the merging of the instances of the initial warehouses. The instances of hierarchies which are common to several warehouses must then be similar. This definition allows in particular to avoid null values in the instances of the hierarchies of dimensions. The analyses which are led on the view can then be completely interpreted.

When the warehouses are represented with a relational mapping, we showed how the view can be easily built by a unique SQL statement.

This integrated view can include numerous tuples and treatment of OLAP queries can induce performance problems. It would be then important to envisage the materialization of the view and to install mechanisms able to accelerate the execution of queries (partitioning, indices, …).

With the exception of the preliminary stage for determining the synonyms, the stages of our construction process can be automated. It would now be necessary to design effective algorithms for each of these stages and to study their complexity.

Other definitions for the integrated view would be possible if one permits null values for instances in the dimension hierarchies. For the examples of figures 3 and 4 one could include the member type "family" in the dimension "product". It is clear that some instances of "category" resulting from the structure B could not be associated to an instance of "family". This will extend the analyses capacity for the integrated view, but problems of interpretation for results of requests could arise.

References

1. Abello, A., Samos, J., Saltor, F.: Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model. Proc. of Intl Workshop on Design and Management of Data Warehouses (DMDW'2001), Interlaken, Switzerland, June 4, 2001.
2. Devlin, B., O'Connell, B.: Information Integration : New Capabilities in Data Warehousing for the on Demand Business. White paper, IBM, January 2005.
3. Ferguson, M.: Data Warehouse : Architecture Methodologies and Technologie. Technology Transfer, March 2001.
4. Hùsemann, B., Lechtenbörger, J., Vossen, G.: Conceptual Data Warehouse Design. Proc. of Intl Workshop on Design and Management of Data Warehouses (DMDW'2000), Stockholm, Sweden, June 5-6, 2000.
5. Kerschberg, L.: Knowledge Management in Heterogeneous Data Warehouse Environments. DaWaK, pp 1-10, 2001.
6. Jindal, R., Acharya, A.: Federated Data Warehouse Architecture. White paper, Wipro Technologies, 2003.
7. Raden, N.: Data Warehousing for Actuaries. Hired Brains Inc, May 2004.
8. Saltor, F., Oliva, M., Abello, A., Samos, J.: Building Data Warehouse Schemas from Federated Information Systems. 17th CODATA Conference, 2002.
9. Schneider, M.: Towards a generic model for data warehouses. Proceedings of the International Conference on Industrial Engineering and Systems Management Conference (IESM), Mai 2005.
10. Sheth, A.P., Larson, J.A.: Federated Database Systems for Management of Distributed Heterogeneous and Autonomous Databases. ACM Computing Surveys, 22(3), pp 183-236, 1990.