A general model for the design of data warehouses

M. Schneider

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

A general model for the design of data warehouses

Michel Schneider

LIMOS, Blaise Pascal University, Complexe des Cézeaux
63173 Aubière, France
Cemagref, 24 Avenue des Landais, 63172 Aubière Cedex, France
schneider@isima.fr
tel : 33 4 73 40 50 09 Fax : 33 4 40 50 01

Abstract : Design and implementation of data warehouses remain delicate tasks which are led by experts. Nevertheless it would be interesting to allow the users to define and to build themselves their system through a simple and flexible process. In particular, in the field of production systems, exists the need to integrate data from various sources and to analyze them in order to extract knowledge for optimizing these systems. Traditionally the design of a data warehouse is based on an adequate representation of facts on one hand and dimensions of analysis on the other hand. We show in this article that an unified representation can be envisaged and that the problem comes down to that of the choice of a hierarchy of criteria adapted to the necessities of analysis. Our proposition leans on a graphic representation which offers a visual help to the user.

Keywords : multidimensional, data warehouse, general model, relational mapping

1 Introduction

Basically, the schema of a data warehouse lies on two kinds of elements : facts and dimensions. Facts are used to memorize measures about situations or events. Dimensions are used to analyze these measures, particularly through aggregation operations (counting, summation, average, …). To fix the ideas let us consider the analysis of the sales in a shop according to the product type and to the month in the year. Each sale of a product is a fact. One can characterize it by a quantity. One can calculate an aggregation function on the quantities of several facts. For example one can make the sum of quantities sold for the product type "mineral water" during January in each of the last three years (2001, 2002, 2003). Product type is a criterion (one said also a member) of the dimension Product. Month and Year are criteria of the dimension Time. A quantity is so connected both with a type of product and with a month of one year. This type of connection concerns the organization of facts with regard to dimensions. On the other hand a month is connected to one year. This type of connection concerns the organization of members within a dimension. The possibilities of fact analysis depend on these two forms of connection and so on the schema of the data warehouse.

Many studies have been devoted to the modelling of dimensions. The objective is to find an organization which corresponds to the analysis operations and which provides strict control over how the aggregations can be made. In particular it is important to avoid double-counting or summation of non-additive data. Many authors recommend organizing the members of a given dimension into hierarchies with which the aggregation paths can be explicitly defined. In [23], hierarchies are defined by means of a containment function. In [14], the organization of a dimension results from the functional dependences which exist between its members, and multi-dimensional normal forms are defined. The work of [13] extends this framework by considering specialization in dimensions. It is shown that there is a natural correspondence between optional dimension levels and attribute occurring in sub-classes. Normal forms are proposed which allow constructing a class hierarchy of dimension levels which guarantees summarizability. In [9], the functional dependences are also used to design the dimensions and to relate facts to dimensions. In [1], relationships between levels in a hierarchy are apprehended through the Part-Whole semantics. In [27], dimensions are organized around the notion of a dimension path which is a set of drilling relationships. The model is centred on a parent-child (one to many) relationship type. A drilling

relationship describes how the members of a children level can be grouped into sets that correspond to members of the parent level. In [24], a dimension is viewed as a lattice and two functions anc (ancestor) and desc (descendant) are used to perform the roll up and the drill down operations. The work of [22] proposes an extended multidimensional data model which is also based on a lattice structure, and which provides non-strict hierarchies (i.e. many to many relationships between the different levels in a dimension). The work of [21] introduces different data hierarchies (balanced and non-balanced, ragged and non-ragged) and studies their influence on rollup paths and their incidence on the expressive power of the OLAP cube. In [19] solutions for handling heterogeneous hierarchies and mixed-granularity hierarchies are suggested. Heterogeneity results from the existence of different sub-classes of a same class, each sub-class having its own attributes and aggregation levels. Mixed-granularity occurs when sub-classes are both end-instances of a dimension and serve as aggregation levels. In [15] are established necessary conditions for summarizability in multidimensional structures.

Modelling of facts and their relationships has not received so much attention. Facts are generally considered in a simple fashion which consists in relating a fact with the roots of the dimensions. However, there is a need for considering more sophisticated structures where the same set of dimensions are connected to different fact types and where several fact types are inter-connected. The model described in [22] permits some possibilities in this direction. The YAM model [3] allows the usage of semantics O-O relationships between different star structures.

Apart from these studies it is important to note various propositions [4, 7, 9, 18] for cubic models where the primary objective is the definition of an algebra for multidimensional analysis. Expressiveness of the algebra is the main topic of these works.

Problem arises to compare the possibilities of these various models and algebras. Several authors worked in this direction. The work of [5] defines requirements for OLAP applications and compares four multidimensional models. No model meets all the requirements. This makes a combination of different models desirable. The work of [2] suggests a framework to classify and compare multidimensional models. Sixteen different models of different levels (conceptual, logical, physical) are studying. It appears that conceptual models offer the possibility of representing much more semantics, but they do not incorporate an algebra for manipulating the data.

Others works must also be mentioned. In [10] is suggested an environment which is able to generate the implementation of a star or snowflake data warehouse from a conceptual schema. The generation process takes into account the limitations of the OLAP target system (Cognos Powerplay or Informix Metacube). In [8], a solution is proposed to derive multidimensional structures from E/R schemas. The work of [20] suggests also a method for developing dimensional models from E/R schemas. Different options for the resulting schema can be choose (flat, star, snowflake, constellation). The work of [26] addresses the problem of integrating the data from heterogeneous data bases and storing it into the repository of the data warehouse. In this work, the data warehouse is seen as a set of materialized views. So the problem becomes a problem of view selection. Different algorithms are proposed and compared for solving it. In [16, 17] are suggested normal forms for the star and the snowflake relational models. The work of [25] extends the conventional data warehouse architecture with analysis rules, which mimic the work of an analyst during decision making. Analysis rules extend the notion of ECA (Event-Condition-Action) rule with mechanisms to analyze data multidimensionally and to make decisions. In [11] is proposed a multi-dimension algebra which strengths the traditional multi-dimensional analysis of quantitative data and extends it to qualitative data (like knowledge). Mechanisms are suggested to efficiently explore the data in a vertical direction (father and sons) or in a horizontal direction (brothers).

Our objective in this paper is to propose a model which can be used to apprehend the modelling of facts and dimension members in a unified way. It permits so to share the dimensions in various ways and to describe different relationships between fact types. Using this model, we will also define the notion of well-formed data warehouse structures. Such structures have desirable properties for applications. We suggest a graph representation for such structures which can help the users in designing and querying a data warehouse.

The paper is organized as follows: sections 2 and 3 respectively present the modelling of facts and the modelling of dimensions; section 4 presents our unified model for facts and members; section 5 presents the typical structures we want to model and defines the notion of well-formed

structures; section 6 shows the ability of our model to describe realistic cases; section 7 discusses relational mappings; sections 8 illustrates SQL queries on the relational form of well-formed structures; section 9 discusses some improvements of our model and concludes.

2 Modelling facts

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

A fact type has the following structure :

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

where

- fact_name is the name of the type ;
- F is a mark for the fact type ;
- fact_key is a list of attribute names ; the concatenation of the values of these attributes identifies each instance of the type ;
- list_of_reference_attributes is a list of attribute names ; each attribute has a value which is either an atomic value (degenerated case) or a reference to a member instance in a dimension or a reference to another fact instance ;
- list_of_fact_attributes is a list of attribute names ; each attribute is a measure for the fact type (such an attribute will also be called a *measure attribute*).

The set of referenced dimensions comprises the dimensions which are directly referenced through the list_of_reference_attributes, but also the dimensions which are indirectly referenced through other facts.

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

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.

Definition 1 (well-formed fact type). A fact type is well-formed if each referenced dimension is either degenerated or points to a legal entry in a dimension (a legal entry, as it is defined further, is a key of any member in a dimension).

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

Sales[(F), (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 references to dimensions: time_key, product_key, store_key. There are two fact attributes: price_per_unit, quantity. The fact attributes can be analyzed through aggregate operations by using the three dimensions. Since each of the three references points to a root in a dimension, this fact type is well formed.

Example 2. Consider now the same fact type but with another reference to the dimension product (the key category_key of the member category).

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

Each of the three references points to the key of a member in a dimension. So, this fact type is well formed.

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

Example 3. Consider again the fact type Sales and another reference to the dimension product (weight of a product).

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

Since product_weight is not a legal entry in a dimension, this fact type is not well-formed. We will see in section 3.6 that such an attribute is a property attribute which cannot be used to organize the multi-dimensional analysis.

In some models, it is imposed that measure attributes are independent from each others. This will clarify their analysis. The independence can be expressed by using functional dependencies: it must not exist any functional dependence between measure attributes. To be general, we do not impose a priori such a constraint.

3 Modelling dimensions

3.1 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 in section 3.6) 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 the i-th dimension.

3.2 Hierarchical organization of members

Members of a dimension are generally organized in a hierarchy which is a conceptual representation of the hierarchies of their occurrences.

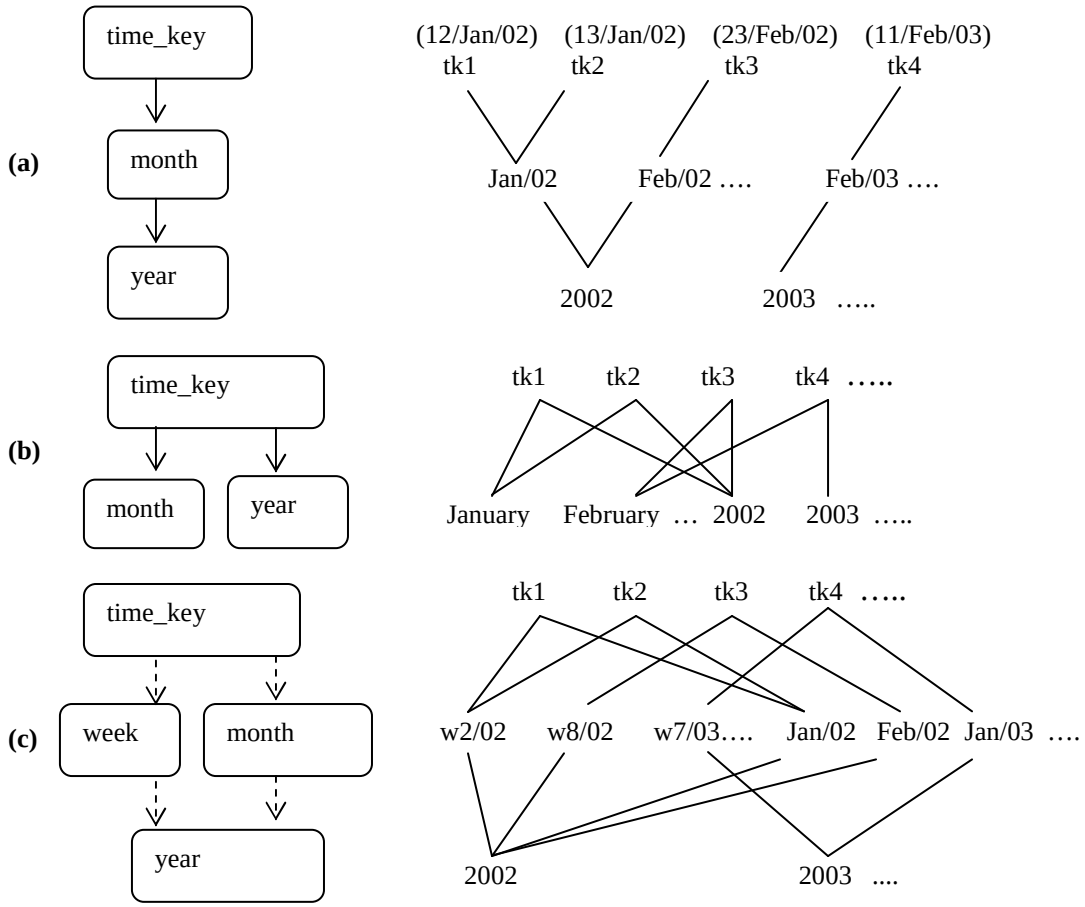

Fig. 1. Different types and data hierarchies in dimensions

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

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. Figure 1 illustrates typical situations which can occur.

Cases (a) and (b) represents the same members but organized differently. In case (a) there are hierarchical relationships between time_key and month and between month and year. Time_key, for example, is a date which encodes the day, the month and the year; the month has values such as February/02 which identifies each month from all the months of the total period. In these conditions, the amount of sales for all the months of all the years, is obtained with a group_by just on the month. In case (b) month and year are both hierarchically dependent of time_key but are independent of each other (month, for example, is a value such as January identifying a month independently from the value of year). With such a structure we can make groupings by using only values of month, or only values of year, or both values of month and year. The expression of the previous query would involve a group_by on month+year. Case (c) represents a hierarchy where the two paths are alternative. They share the same root type (time_key) and the same leaf type (year). Starting from time_key, groupings are possible by using either the values of week or either the values of month (but not both). This configuration has precise semantics: for a given occurrence of time_key, whether the week path or the month path is used, one always obtains the same occurrence of year. This means that, using either the first path or the second path, we obtain the same result when continuing the aggregations with the year element. In this case we said that alternatives paths satisfy the *path coherence constraint*. We use dotted arrows to represent alternative paths.

We will model these different cases 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. 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.

3.3 Cover graph of a dimension

For the following, we suppose that HR is anti-symmetric and transitive. To justify these properties we can refer to the notion of part-whole relationship. This kind of relationship is often appropriate to apprehend real situations [1]. There exist different semantics for part-whole-relationships and most of them verify these two properties. Thanks to the transitivity property we introduce the notion of cover graph of a dimension. The idea behind the notion of cover graph is to remove in a dimension graph all redundant directed edges and paths.

Definition 2 (cover graph of a dimension). Consider the directed graph defined by the HR between the members of a dimension. A cover graph of this dimension is a minimal sub-graph obtained by removing all the directed edges resulting from the transitivity property.

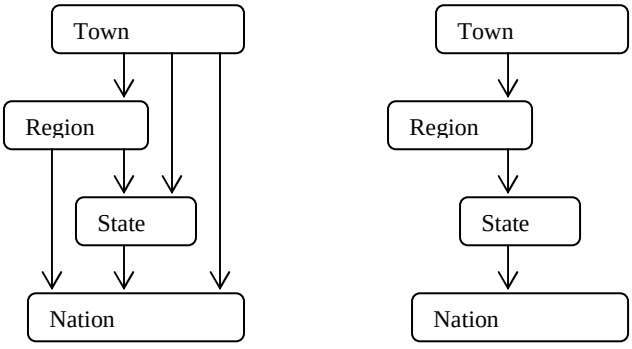

Fig. 2. A graph of a dimension and its cover graph.

Example 4. Figure 2 illustrates the case of a dimension graph with all the edges obtained by transitive closure and the corresponding cover graph.

In our dimension graph, we do not introduce the member type "ALL" which represents the upper level of aggregation. This member specifies that all the occurrences of a fact type which point to a same dimension can be aggregate together. However, this upper aggregation will always be possible with our model.

3.4 Well-formed dimension

Definition 3 (well-formed dimension). A dimension is well-formed relatively to a cover graph when this graph has a unique connected component and is acyclic and when the path coherence constraint is satisfied for alternative paths.

Restricting the cover graph to a unique component is very important in practice. If the graph comprises, for example, two components, the dimension must be divided into two distinct dimensions. We do not impose that the cover graph has a unique root. In real situations we can encounter different dimensions which share a common part.

Example 5. Let us consider the dimensions illustrated in figure 1. They are all well-formed since their cover graphs have a unique component and are acyclic (in case (c) it is supposed that the path coherence constraint is satisfied).

Example 6. In figure 3 we illustrate a dimension graph with two roots. In this case it appears that two dimensions which are priori different (supplier dimension and customer dimension) share a same part.

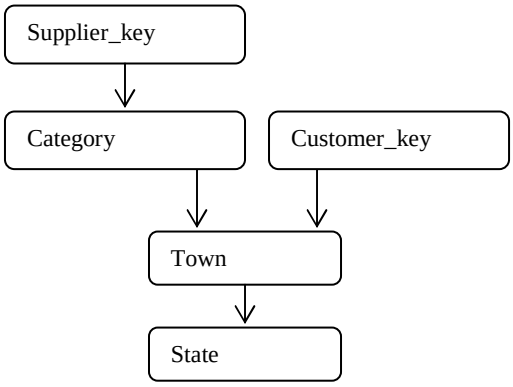

Fig. 3. A dimension graph with two roots

Remark (relaxing the transitivity property for the HR). Definition 2 can be easily extended to the more general case where the transitivity property is not satisfied by the HR. In this case, the cover graph is replaced in the definition by the initial graph of the dimension or by an equivalent non redundant graph. A non-redundant graph is obtained by removing directed edges which can be derived by using some rule.

3.5 Aggregation levels in a well-formed dimension

Since the cover graph of a well-formed dimension is acyclic, it is possible to distribute its members 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.

3.6 Property attributes in a dimension

As in other studies [12], we consider property attributes in a dimension. Such attributes are 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, … . These attributes are not interesting to specify groupings but they can be useful in the selection predicates of queries to filter certain groups.

3.7 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[(M), (member_key), (list_of_reference_attributes), (list_of_property_attributes)]

where

- member_name is the name of the type;
- M is a mark for the member 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 7. Using this model, the representation of the members of the dimension represented in figure 1(c) is the following:

time_root[(M), (time_key), ((week_key, month_key)), ()]
week[(M), (week_key), (year_key), (week_type)]
month[(M), (month_key), (year_key), ()]
year[(M), (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 [(M), (w1_03), (2003), (holiday)].

3.8 Entries and roots in a dimension

Each member_key of a member in a dimension can be an entry for this dimension i.e. can be referenced from a fact type or from another member type. This possibility is very important since it means that dimensions can be shared between several fact types in various ways. In particular, it is possible to reference a dimension at different levels of granularity. A root represents a standard entry in a dimension. The cover graph can have several roots. For the three dimensions in figure 1, there is a single root. However, definition 3 authorizes several roots.

3.9 Handling many-to-many relationships in a dimension

In certain situations, a many-to-many relationship exists between two members of a dimension hierarchy. For example a Project can be connected with several Groups (possibly zero) and conversely a Group can collect several Projects (at least one). Within the framework of an analysis of purchases (purchase is so a fact type), a dimension can incorporate projects. Problem arises so how to elaborate the hierarchy with the Projects and the Groups. One supposes that a purchase for a project depends on the group to which this project belongs. A simple solution consists in two separated hierarchies, a one around a member Project and the other one around a member Group (Figure 4(a)). It is then possible to make aggregations, either on the projects separately, or on the groups separately, or simultaneously on projects and groups by crossing the two dimensions. Another solution consists in a single dimension with a member Project-Group which points towards members Project and Group (Figure 4(b)). This solution offers the same possibilities of aggregation as the previous one but it materializes the relationship between the Projects and the

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

Groups through the instances of the member Project-Group. It becomes then possible to materialize aggregates at the level of Project-Group. In this second solution, if a project is isolated (it does not belong to any group), it is possible as in [19] to introduce an artificial instance for Group.

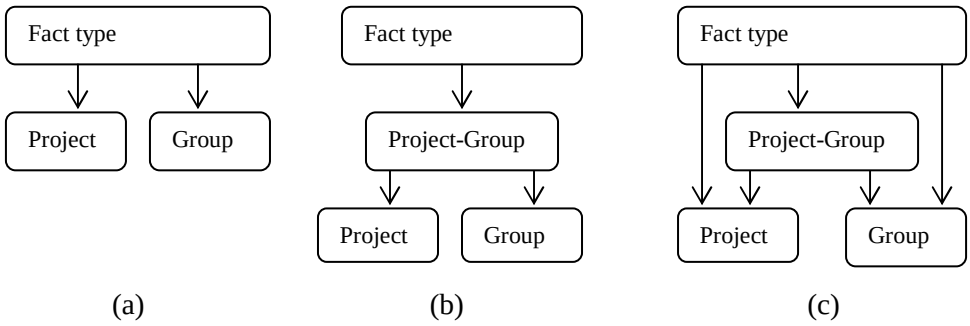

Fig. 4. Handling a many-to-many relationship in a dimension

For the graph of figure 4(b), the HR is transitive. One can so deduct from it the graph of figure 4(c). This graph means that one can directly aggregate instances of the fact type either for every Project, or for every Group. The graph of figure 4(b) is a cover graph according to definition 2.

4 A unified way for modelling elements (facts or members)

A fact type has a very similar structure to that of a member type. Moreover, property attributes of a member can be seen as fact attributes and can be analyzed along the successors of this member acting as roots of partial dimensions. For example, suppose that in a dimension we have a member town with a property attribute *population* which references the member region. One can analyze *population* by using region : one can calculate aggregates on *population* with groupings on *region_name*.

An element is a type which has the following structure :

element_name[(T), (element_key), (list_of_references), (list_of_specific_attributes)]

where

- element_name is the name of the type;
- T is a mark for the type (F or M);
- element_key identifies each instance of the type;
- list_of_references is composed of sub-lists of attribute names; each attribute references another element ; these references determine the multidimensional structure of the data warehouse;
- list_of_specific_attributes is a list of attribute names; each attribute is a measure for a fact or a property for a dimension member.

The list of references determines the dimensions along which the element can be analyzed. Each specific attribute can be analyzed along each of these dimensions. Analysis is achieved through the computing of aggregate functions on the values of this attribute. The aggregation is specified through the values of the chosen members in the dimensions. Recall that the notion of sub-list is used to mark alternative paths.

5 Well-formed structures

5.1 Various structures for data warehouses

In this section we explain how with our unified model fact types and member types can be interconnected in order to model various data 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 in our model 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.

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

Moreover, a fact F_1 can reference any other fact F_2. This type of reference is necessary to model certain situations (see section 6). This means that a fact attribute of F_1 can be analyzed 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.

To formalise the interconnection between facts and dimensions, we thus suggest extending the HR relationship of section 3 to the representation of the associations between fact types and the associations between fact types and member types. This gives a very uniform model since fact types and member types are considered equally. To maintain a traditional vision of the data warehouses, we also ensure that the members of a dimension cannot reference facts.

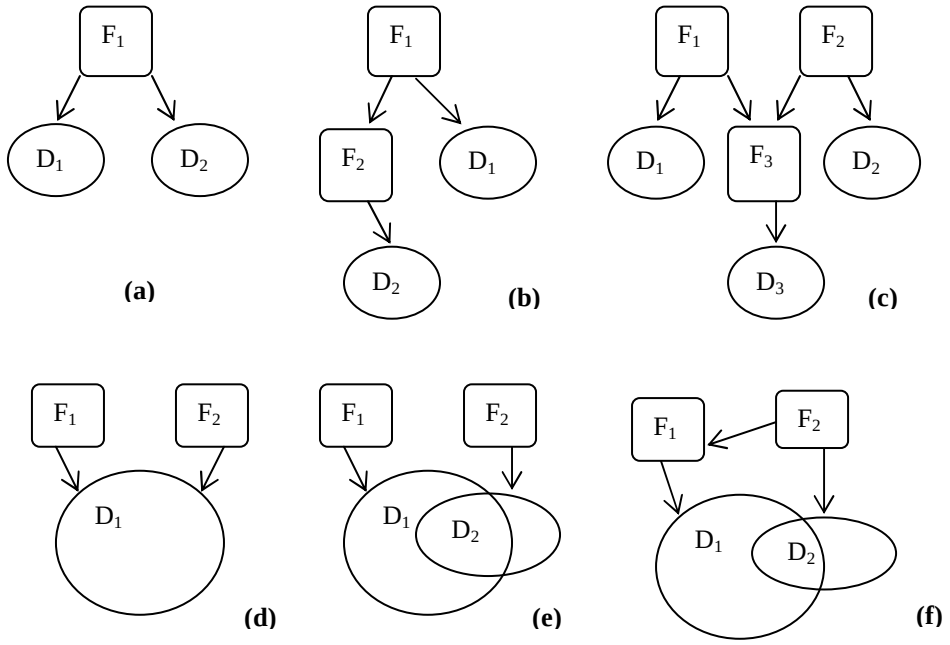

Fig. 5. Typical data warehouse structures

Figure 5 illustrates the typical structures we want to model. Case (a) corresponds to the simple case, also known as star-snowflake 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. Case (d) corresponds to the sharing of a whole dimension. In cases (e) and (f) only the members which are at the intersection of D_1 and D_2 are shared. An illustration of case (e) is given further in Fig. 7.

5.2 Data Warehouse Graph (DWG) and well-formed structures

To represent data warehouse structures, we suggest using a graph representation called DWG (data warehouse graph). It consists in representing each type by a node containing the main information about this type, and representing each reference by a directed edge. Alternative paths in dimensions are represented by dotted lines.

We are now able to introduce the notion of well-formed structures.

Definition 4 (well-formed data warehouse structures). A data warehouse structure is well-formed when the following conditions are satisfied :

1) the fact types are well-formed ;
2) the dimensions are well-formed ;
3) a fact type is a root or all its predecessors in the graph are fact types ;
4) the DWG has a unique component and is acyclic.

5.3 Cut in the DWG

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

We introduce now the notion of cut in the DWG in order to characterise the legal groupings when making an analysis.

Definition 5 (Cut of a sub-graph in a DWG). Let SG be a sub-graph of a well-formed DWG with root R. A cut of SG is a sub-set S={e_1, e_2, …, e_k) of edges of SG such that two different edges e_i, e_j of S does not appear in a same path or in two alternative paths.

Property 1. Each cut of a sub-graph of a well-formed DWG defines a legal grouping (i.e. the elements pointed by the edges of the cut can be legally combined to make an aggregation on the occurrences of the root of the sub-graph).

This important property permits to use the DWG in order to situate the different possibilities of aggregations. This ability is illustrated in the next section.

6 Illustrating the modelling of realistic cases with well-formed structures

In this section we show how different realistic cases can be described with well-formed structures.

6.1 Star and snowflake structures

We have a star or snowflake structure when :

- 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.

Our model does not differentiate star structures from snowflake structures. The difference will appear with the mapping towards the relational model (see section 7). The DWG of a star-snowflake structure is represented in figure 6. This representation is well-formed. Such a representation can be very useful to a user for formulating queries. Facts are clearly differentiated from members, reference to dimensions are shown explicitly, analysis criteria appear immediately.

We can illustrate on this structure several cuts as defined previously. For example (e_1, e_2, e_6) is a cut, also (e_1, e_5, e_6). So groupings on (e_1, e_2, e_6) or (e_1, e_5, e_6) are legal. But (e_3, e_4, e_5) or (e_2, e_5, e_6) are not cuts. So groupings on (e_3, e_4, e_5) or (e_2, e_5, e_6) are not legal.

It is clear that any subset of a cut is also a cut.

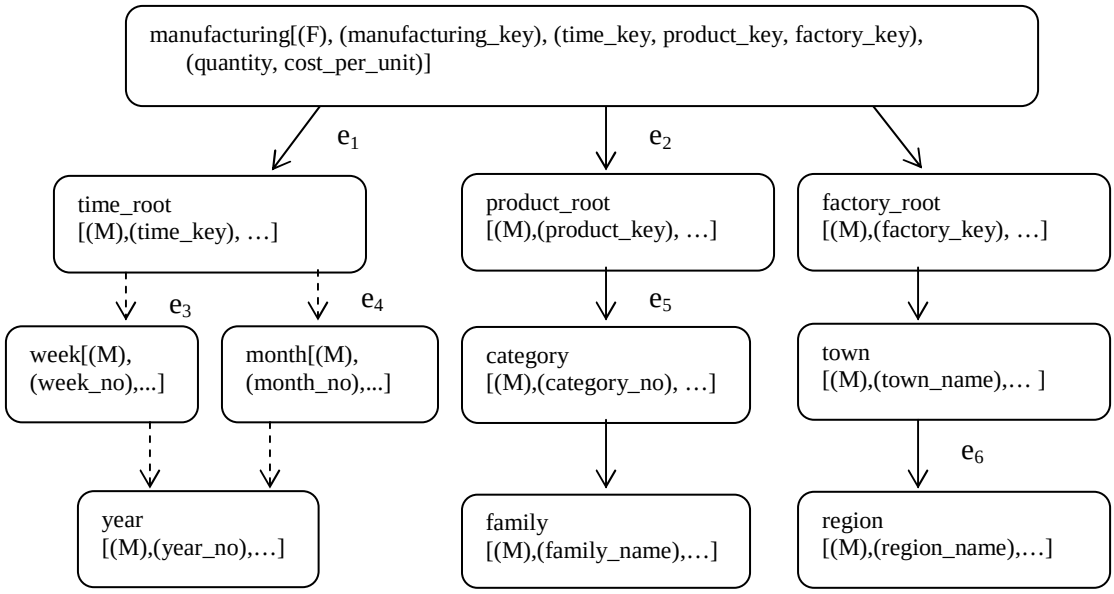

Fig. 6. The DWG of a star-snowflake structure

6.2 Constellation structure (sharing of a dimension)

The constellation structure appears when:

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

- there are at least two different root types (each root acts as a fact type);
- a fact type does not reference another fact type.

Note that in such a structure, there exists at least one element in the hierarchy which can be reached from the two roots.

Using the notion of DWG, figure 7 shows an example with the fact type *manufacturing* from figure 6 and a new fact type *stock* memorizing stock facts. *Stock* has a reference to the member *town* of the dimension *factory*. So, the dimension *factory* is shared partly between the two fact types. The two other dimensions are completely shared.

The DWG clearly shows how the two fact types can be exploited separately or simultaneously. We can explore the graph from one of its two roots and use it as a single rooted graph. We can also simultaneously exploit the two fact types. For example, to the node *town*, one can associate different aggregates from the *stock* occurrences and use them for the analysis of the *manufacturing* facts, or vice-versa.

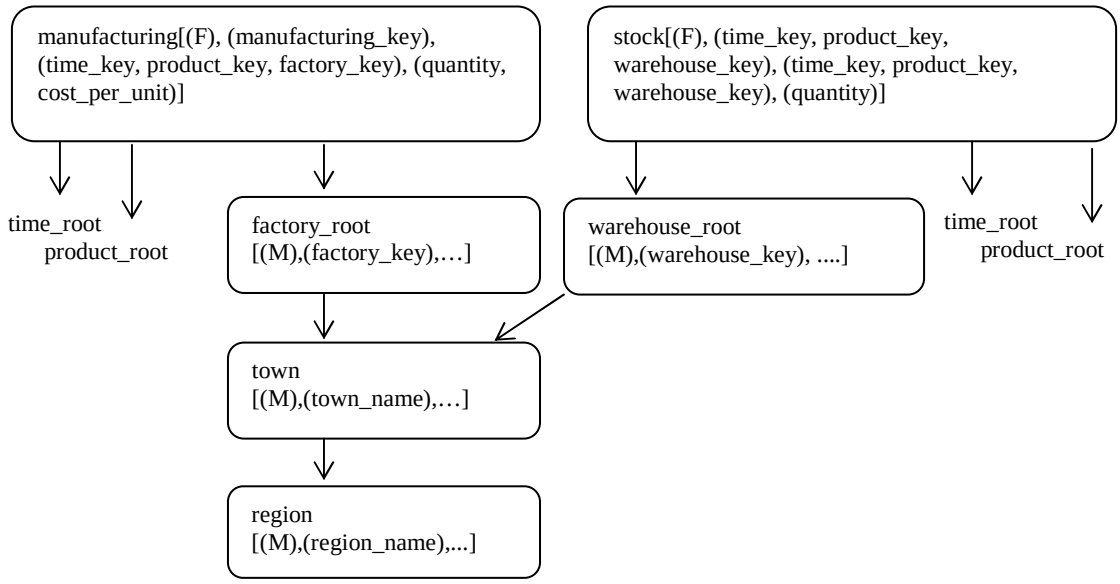

Fig. 7. The DWG for a constellation data warehouse

6.3 Facts of fact with a unique root

Sometimes one fact type, called primary fact type, can be characterized by several other fact types, called secondary fact types. Let us consider for example the case of a manufacturing of a product. It is characterized by primary fact attributes such as quantity, cost, ... An instance of manufacturing is created each time a product is manufactured. A manufacturing is in fact composed of several operations on machines. Secondary fact attributes are associated to each operation such as the duration of the operation with a given machine. It is not adequate to memorize these secondary facts in the primary type (there is a many-to-many association between operations and manufacturing). One solution consists in placing them in a secondary fact type referencing the primary fact type. Our model caters for the description of such a solution. It consists in specifying two different fact types *manufacturing* and *operation*, and installing in *operation* a reference to *manufacturing* (figure 8). It should also be noted that *operation* has a normal reference to the root of the *machine* dimension. There is an instance of *operation* for each different operation in a *manufacturing*. Note that a serial number is used in the primary key of the *operation* type to mark the order of an operation in the sequence of a manufacturing. So an operation can occur several times in a manufacturing. The key of the operation type is composed of the operation name, the serial number, the manufacturing key. For example a welding operation can be associated to the manufacturing M100, a first time with serial number 2, and a second time with serial number 4. This means that this operation occurs in position 2 and in position 4 in the sequence of operations for the manufacturing M100. So two instances with respective keys (welding, 2, M100) and (welding, 4, M100) appear for the operation type. With this representation

it is possible to analyze the duration of an operation by using the serial number (for example : what is the average ratio between the duration of a welding operation and the total duration of a manufacturing when the welding occurs in position 2) or not (for example : what is the average ratio between the duration of the welding operations and the total duration of a manufacturing).

For the secondary fact type, the primary fact type acts as a multi-dimension. So, all the dimensions of the primary type can be also used as dimensions of the secondary type. This clearly appears in the DWG of the global structure (figure 8). For example, quantity can be analyzed using the *time* and *product* dimensions and *duration* can be analyzed using *machine* dimension but also *time* and *product* dimensions. The *operation* fact can also be analyzed using *manufacturing_key* alone acting as a degenerated dimension. For example we can calculate the average duration of an operation per manufacturing.

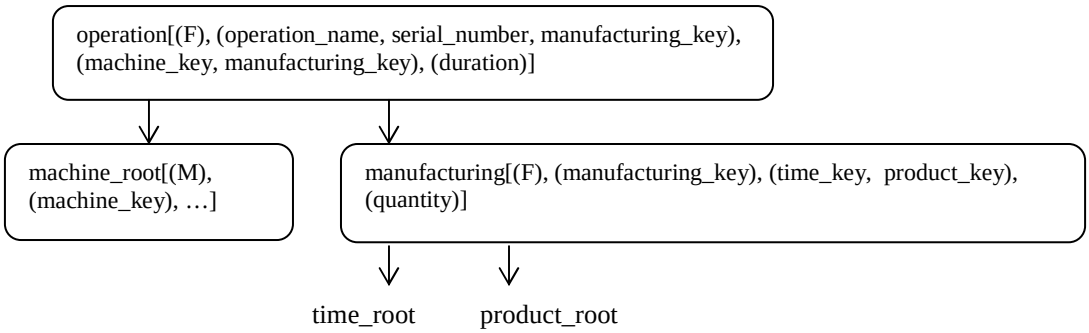

Fig. 8. Modelling facts of fact with a unique root

6.4 Facts of fact with several roots

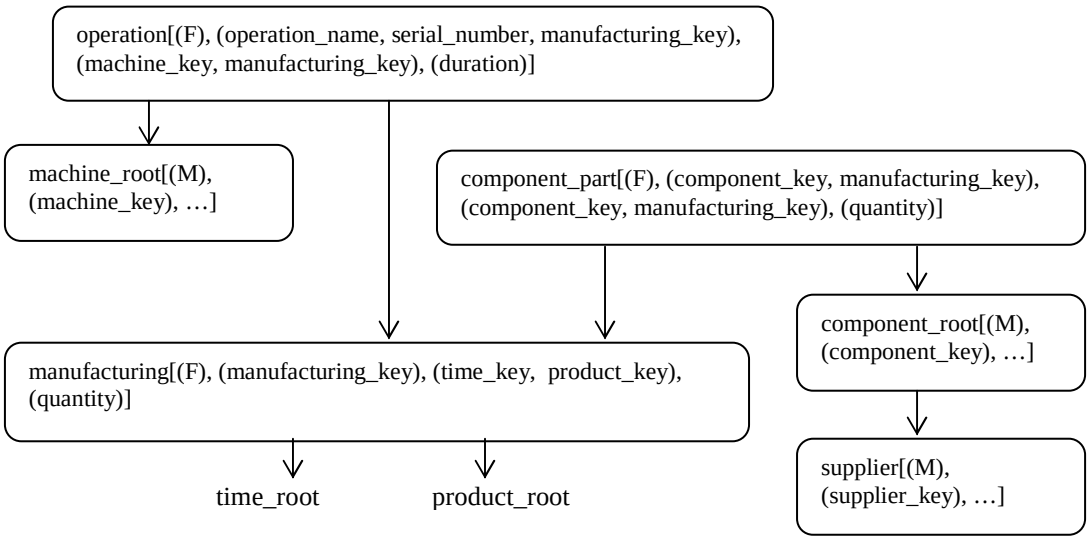

Fig. 9. Modelling facts of fact with two roots

For the previous example we wish now to analyze the component parts which are needed for a manufacturing and which are necessary to buy in advance to suppliers outside the company. For a given manufacturing, several component parts are involved. We introduce so another secondary fact type *component_part* which references the *manufacturing* type. The measure attribute *quantity* of the *component_part* type can be so analyzed not only according to the component dimension (for example: which is the supplier who have supplied the largest number of component parts) but also according to *manufacturing* (for example: how much a manufacturing requires of component parts on average) and of the dimensions *product* and *time* (for example: total number of component parts which were furnished by a given supplier for each month in year 2005).

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

7 Mapping to the relational model

One way to implement data 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. In the framework of [6], our proposal is related to relational approaches.

7.1 Relational mapping with split dimension

This solution, which is straightforward, consists in mapping each type P_i (fact type or member type) 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 6, this solution leads to the relational snowflake data warehouse structure as represented below (primary keys are marked in bold, foreign keys in italics).

manufacturing(**manufacturing_key**, *time_key*, *product_key*, *factory_key*, quantity, cost_per_unit)
time_root(**time_key**, *A1$week_no*, *A1$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**, …)
factory_root(**factory_key**, *town_name*, …)
town(**town_name**, *region_name*, ...)
region(**region_name**, …)

Note that the two references in town for the first (and unique) alternative are marked with the special prefix A1$. If a second configuration of alternative paths would exist in the same dimension, we can use the special prefix A2$, and so on.

7.2 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. For a structure like the one described in figure 6, this solution leads to the relational star data warehouse structure as represented below.

manufacturing(**manufacturing_key**, *time_key*, *product_key*, *factory_key*, quantity, cost_per_unit)
time_root(**time_key**, …, A1$week_no, …, A1$month_no, …, year_no, …)
product_root(**product_key**, …, category_no,…, family_name, …)
factory_root(**factory_key**, …, town_name, …, region_name, …)

With this mapping the structure of the dimension hierarchies is not represented. It is embedded in the data.

7.3 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.

For the structure of figure 7, this mapping gives the following tables :

manufacturing(**manufacturing_key**, *time_key*, *product_key*, factory_key, ..., *town_name*, quantity, cost_per_unit)
stock(***time_key***, ***product_key*****, warehouse_key**, ..., *town_name*, quantity)
time_root(**time_key**, …)
product_root(**product_key**, …)

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

town(**town_name**,…, region_name, ...)

Note that the attributes of the element types *factory_root*, *warehouse_root*, *region* have been encapsulated into the tables *manufacturing*, *stock*, *town* respectively.

7.4 Performances of theses mappings

These various mappings do not offer the same performances as regards the execution of queries by the relational engine. The most efficient mapping is the one that requires a minimum of joins that is the second mapping. But this mapping, as we underlined it, does not explicitly represent relationships between the dimension members. It is more difficult for the end user to separate the legal groupings and the not legal groupings. The first mapping well represents these relations but it is the least efficient. In practice it will so be necessary to choose a hybrid mapping.

It is important to notice that almost every relational DBMS offer now functions to calculate the day, the month, the year from any date. As a consequence it is useless to represent the corresponding members in the time dimension which can be so considerably simplified. In a majority of cases the time dimension can be degenerated and represented only by an attribute of type DATE in the fact tables.

7.5 Automatic generation of theses mappings

Indeed the descriptions of the different types (facts and members) contain all the necessary information for generating the relational schema. It is not difficult to design an algorithm which ensures this generation. The problem is how to produce the descriptions of the different types from a DWG. We can imagine a software component with a graphical interface which permits a user to draw the DWG. This component must control at each step the properties and the constraints of the DWG and must verify that the structure is well-formed. It is this constituent which has to offer a function for generating the descriptions of the types. Such a constituent would be very useful to allow the users to be involved in the design of the data warehouse. But its elaboration is not so easy and represents a main objective of our future work.

8 Illustrating SQL queries on the relational forms of well-formed structures

In this section we show how the DWG can help the design of aggregation queries on the relational forms defined previously.

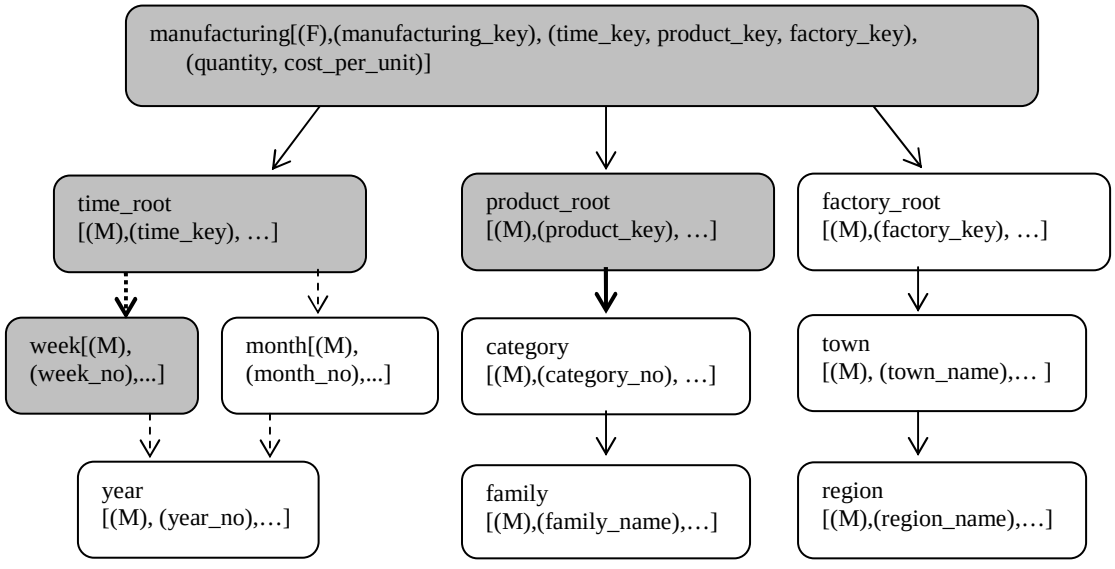

Fig. 10. The marking of the DWG for illustrating Query 1

8. 1 Query 1 (a simple aggregation query on a star or snowflake structure)

For the data warehouse represented in figure 6 we want to obtain the total quantity manufactured for each category of products and each week of the year 2004.

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

This query corresponds to the cut represented in Figure 10 (edges of the cut are marked in bold). The DWG permits to identify immediately the different elements involved in the query (in grey in the figure). Note that we do not select the year element and the category element since the attributes we need are included as foreign keys in the week table and in the product_root table.

For the relational snowflake representation the corresponding SQL query is the following :

```
Select sum(quantity) from manufacturing m, time_root t, product_root p, time_root t, week w
where m.time_key=t.time_key and m.product_key=p.product_key and
 t.A1$week_no=w.week_no and w.year_no=2004
group by t.A1$week_no, p.category.no;
```

For the relational star representation the query is slightly shorter (since the week table is embedded into the time_root table) :

```
Select sum(quantity) from manufacturing m, time_root t, product_root p
where m.time_key=t.time_key and m.product_key=p.product_key and t.year_no=2004
group by t.A1$week_no, p.category_no;
```

8.2 Query 2 (using results analysis coming from another fact when a same dimension is shared)

For the data warehouse represented in Figure 7 we want to elaborate the total quantity of product 40 manufactured in the towns where there exists a stock greater than 200 for this product on the 12 December 2004.

Also in this case the DWG can be very useful to help in identifying the different elements involved in the query (see figure 11). We use the hybrid relational mapping of section 7 for expressing the SQL query.

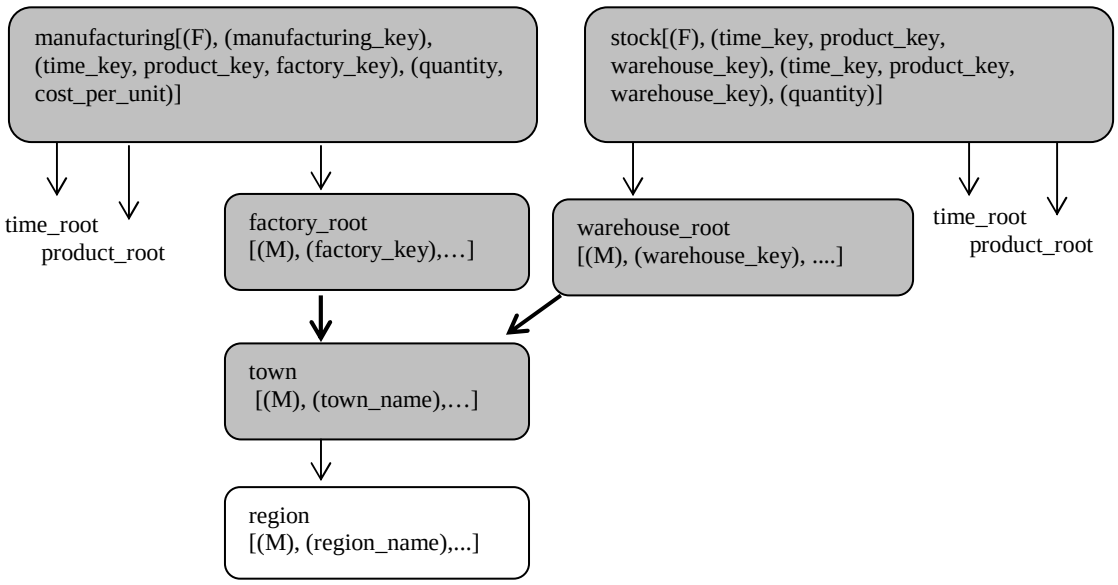

Fig. 11. The marking of the DWG for illustrating Query 2

In a first step we elaborate the stock for the product 40 on the 12 December 2004 for each town and we incorporate the result in the corresponding town instance using a property attribute.

```
Create view v(sum_stock, town_name) as
   select sum(quantity), town_name from stock
   where time_key= “12 December 2004”and product_no=40
   group by town_name;

Alter table town add (sum_stock number(6) default 0).
Update town t set sum_stock=(select sum_stock from v where v.town_name=t.town_name);
```

International Journal of Production Economics, Volume 112, Issue 1, March 2008, Pages 309-325
Elsevier, doi:10.1016/j.ijpe.2006.11.027

In a second step, we can specify the final query by using the new attribute sum_stock in the town table.

```
Select m.town_name, sum(quantity) from manufacturing m, town t
where m.town_name=t.town_name and m.time_key= “12 December 2004”and
m.product_no=40 and t.sum_stock>200
group by m.tow_name;
```

8.3 Towards an automatic generation of queries

The DWG can be used to support a graphic interface so as to allow a user to specify its aggregation queries. For a simple query, without filtering conditions, it is necessary to select the measure attribute and to indicate the cut (one saw in sections 6 and 8 how a cut of the DWG allows to specify the legal groupings). For a more complex query, with filtering conditions, it is necessary also to be able to specify graphically these conditions. From these graphic specifications one can rather easily design an algorithm which allows generating the SQL code of the query by taking into account the mapping which was used to generate the data base.

9 Conclusion

In this paper we propose a model which unifies the notion of fact and dimension member. This model can describe various data warehouse structures. It extends existing models for sharing dimensions and for representing relationships between facts. It allows for different entries in a dimension corresponding to different granularities. A dimension can also have several roots corresponding to different views and uses. It is possible to apprehend the concept of facts of fact which is very frequently encountered in the real world. Based on this model, the schema of the data warehouse is graphically represented through a graph called Data Warehouse Graph (DWG). Thanks to the DWG, we define the notion of well-formed data warehouse structures which guarantees desirable properties.

We have shown how well-formed structures can be mapped to the relational model in different ways. To represent the references we have used values of reference attributes. Instead, we can adopt an object-oriented model. Identifiers and references would then be represented through OIDs. This would make it possible to define mappings towards the object relational model.

We showed also how the DWG could be used to help in the specification of aggregation queries, by using notably the notion of cut in the graph. Moreover, the model which we propose allows queries which use aggregates resulting from various facts. For example it is possible to use an aggregate which results from a fact table F1 to filter aggregates resulting from another fact table F2.

The DWG can also be very helpful for the design of a data warehouse. An end user can easily express his decision-making needs by specifying with our model the multi-dimensional schema which seems the most appropriate. It is then possible to envisage automatic techniques to look for the compatibility of this schema with those from the available sources. The semantics problems can be solved by leaning on the techniques of the semantic Web. Incompatibilities will be solved through a dialogue between the user and the system. The schema being strengthened then definitively, it is possible to use approaches by materialized views (as in [26]) to make the connection between the sources and the repository and to solve the problems of data refreshment.

We think that such facilities would permit an end user to handle himself the design and the manipulation of his data warehouse. In many domains there exist multiple needs for constructing a data warehouse and for extracting knowledge from the data. With our approach end users have the opportunity to be freed from the technical constraints.

We think that our model can be enriched in different ways. We are investigating extensions to allow specializations in dimensions ([13], [19]) and to handle so the cases of null values. We envisage also the possibility for defining a reflexive relation on a member type in order to take into account the hierarchies of data of various levels. Handling mixed-granularity hierarchies ([19]) is another important extension we have in project. In all cases it would be necessary to fit out the

relational mappings in order to propose adequate representations which can be exploited with the relational engine.

Other extensions are less immediate because they suppose to introduce specific operations through an algebra. For example for the extension in the treatment of qualitative data as in [11], it would be necessary to introduce operations to scan efficiently the selected data. The situation is the same for the dynamic treatment of data for which it is a question of tracking down the changes which occurred during a time interval.

As far as our model permits to exploit simultaneously data stemming from several cubes, it would be also interesting to think about operations which allow to get back an aggregate of a cube to use it in another cube (in a condition of filtering for example as in section 8.2). For this purpose, the relational mapping could be adapted to allow the materialization of aggregates, either in the tables of the dimension members, or by using materialized views. This last solution would be doubtless more interesting because it could exploit the well-known rewritings techniques of queries from views for generating efficient execution plans.

It would be also interesting to establish mappings for implementation on other OLAP engines. We think that our model can be used in various OLAP environments to help the design of data warehouses.

Acknowledgements
The author is grateful to the reviewers for their very useful and helpful comments.

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. Abello, A., Samos, J., Saltor, F. : A framework for the classification and description of multidimensional data models. Proc. of the 12th International Conference on Database and Expert Systems Applications, DEXA'01, Munich, Germany, pp. 668-677, 2001.
3. Abello, A., Samos, J., Saltor, F. : YAM2 (Yet Another Multidimensional Model): an extension of UML. Proc of the International Database Engineering & Applications Symposium, IDEAS 2002, Edmonton, Canada, pp. 172-181, 2002.
4. Agrawal, R., Gupta, A., Sarawagi, S. : Modelling Multidimensional Databases. ICDE'97, 13th International Conference on Data Engineering, Birmingham, UK, pp. 232-243, April 7-11, 1997.
5. Blaschka, M., Sapia, C., Hofling, G., Dinter, B. : Finding your way through multidimensional data models. Proc of the 9th International Conference on Database and Expert Systems Applications, DEXA '98, Vienna, Austria, August 1998, LNCS, vol. 1460, SpringerVerlag, pp. 198-203, 1998.
6. Czarnecki, K., Helsen, S. : Classification of model transformation approaches. Proc of the OOPSLA '03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, USA, 2003.
7. Datta, A., Thomas, H. : The Cube Data Model : A Conceptual Model and Algebra for on-line Analytical Processing in Data Warehouses. Decision Support Systems, pp. 289-301, 27(3), December 1999.
8. Golfarelli, M., Maio, D., Rizzi, S. : Conceptual Design of Data Warehouses from E/R Schemes. Proc. of the 32th HICSS, 1998.
9. Gyssens, M., Lakshmanan, V.S. : A Foundation for Multi-dimensional Databases. Proc. of the 23rd Conference on Very Large Databases, pp. 106-115, 1997.
10. Hahn, K., Sapia, C., Blaschka, M. : Automatically Generating OLAP Schemata from Conceptual Graphical Models. Proc. of the 3rd ACM international workshop on Data warehousing and OLAP, DOLAP'00, McLean, Virginia, USA, 2000. (http://citeseer.ist.psu.edu/341251.html)
11. Huang, C.C, Tseng, T.L., Li, M.Z., Gung R.R. : Models of multi-dimensional analysis for qualitative data and its application. European Journal of Operational Research. In Press, available

on Elsevier web site.
12. 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.
13. Lechtenbôrger, J., Vossen, G. : Multidimensional Normal Forms for Data Warehouse Design. Information Systems, Volume 28, Issue 5, July 2003.
14. Lehner, W., Albrecht, J., Wedekind, H. : Normal Forms for Multidimensional Data Bases. l0th Intl Conference on Scientific and Statistical Data Management (SSDBM'98), Capri, Italy, pp. 63-72, 1998.
15. Lenz, H.J., Shoshani, A. : Summarizability in OLAP and Statistical Data Bases. Proc. of the 9th SSDBM, pp. 132-143, 1997.
16. Levene, M., Loizou, G. : Why is the Star Schema a Good Data Warehouse Design. http://citeseer.ni.nec.com/457156.html, April 1999.
17. Levene, M., Loizou, G. : Why is the Snowflake Schema a Good Data Warehouse Design. Information Systems, Volume 28, Issue 3, pp. 225-240, 2003.
18. Li, C., Wang, X.S. : A Data Model for Supporting on-line Analytical Processing. Proc. of the Fifth International Conference on Information and Knowledge Management, pp. 81-88, 1996.
19. Mansmann S., Scholl M.H. : Extending Visual OLAP for Handling Irregular Dimensional Hierarchies. DaWak 2006, LNCS 4081, pp. 95-105, 2006.
20. Moody, L.D., Kortink, M.A.R. : From Enterprise Models to Dimensional Models: A methodology for Data Warehouse and Data Mart Design. Proc of the International Workshop on Design and Management of Data Warehouses, DMDW'2000, Stockholm, Sweden, 2000. (http://citeseer. nj.nec.com/moody00from.html)
21. Niemi, T., Nummenmaa, J. : Logical Multidimensional Database Design for Ragged and Unbalanced Aggregation Hierarchies. Proc of the International Workshop on Design and Management of Data Warehouses, DMDW'01, Interlaken, Switzerland, 2001. (http://citeseer.ist.psu.edu/472790.html)
22. Pedersen, T.B., Jensen, C.S. : Multidimensional Data Modelling for Complex Data. Proc of the Intl Conference on Data Engineering, ICDE' 99, pp. 336-345, 1999.
23. Pourabbas, E., Rafanelli, M. : Characterization of Hierarchies and some Operators in OLAP Environment. DOLAP'99, Kansas City, USA, pp. 54-59, 1999.
24. Vassiliadis, P., Skiadopoulos, S. : Modelling and Optimisation Issues for Multidimensional Databases. Proc. of the 12th Intl Conference CAISE, Stockholm, Sweden, pp. 482-497, 2000.
25. Thalhammer, T., Schrefl, M., Mohania, M. : Active Data Warehouses : Complementing OLAP with analysis rules. Data & Knowledge Engineering, 39, pp 241-269, 2001.
26. Theodoratos, D., Ligoudistianos, S., Sellis, T.K. : View selection for designing the global Data Warehouse. Data & Knowledge Engineering, 39, pp. 219-240, 2001.
27. Tsois, A., Karayannidis, N., Sellis, T. : MAC: Conceptual Data Modeling for OLAP. Proc. of the Intl Workshop on Design and Management of Data Warehouses (DMDW'2001), Interlaken, Switzerland, June 4, 2001.