Conceptual Design of Data Warehouses from E/R Schemes

Matteo Golfarelli
DEIS - Univ. of Bologna
mgolfarelli@deis.unibo.it

Dario Maio
DEIS, CSITE - Univ. of Bologna
dmaio@deis.unibo.it

Stefano Rizzi
DEIS - Univ. of Bologna
srizzi@deis.unibo.it

Abstract

Data warehousing systems enable enterprise managers to acquire and integrate information from heterogeneous sources and to query very large databases efficiently. Building a data warehouse requires adopting design and implementation techniques completely different from those underlying information systems. In this paper we present a graphical conceptual model for data warehouses, called Dimensional Fact model, and propose a semi-automated methodology to build it from the pre-existing Entity/Relationship schemes describing a database. Our conceptual model consists of tree-structured fact schemes whose basic elements are facts, attributes, dimensions and hierarchies; other features which may be represented on fact schemes are the additivity of fact attributes along dimensions, the optionality of dimension attributes and the existence of non-dimension attributes. Compatible fact schemes may be overlapped in order to relate and compare data. Fact schemes may be integrated with information of the conjectured workload, expressed in terms of query patterns, to be used as the input of a design phase whose output are the logical and physical schemes of the data warehouse.

1. Introduction

The database community is devoting increasing attention to the research themes concerning data warehouses; in fact, the development of decision-support systems will probably be one of the leading issues for the next years. The enterprises, after having invested a lot of time and resources to build huge and complex information systems, ask for support in obtaining quickly summary information which may help managers in planning and decision making. Data warehousing systems address this issue by enabling managers to acquire and integrate information from different sources and to query efficiently very large databases.

The topic of data warehousing encompasses application tools, architectures, information service and communication infrastructures to synthesize information useful for decision making from distributed heterogeneous operational data sources. These information are brought together into a single repository, called a *data warehouse* (DW), suitable for direct querying and analysis and as a source for building logical *data marts* oriented to specific areas of the enterprise [15].

While it is universally recognized that a DW leans on a multidimensional model, little is said about how to carry out its conceptual design starting from the user requirements. On the other hand, we argue that an accurate conceptual design is the necessary foundation for building an information system which is both well-documented and fully responding to requirements. The Entity/Relationship (E/R) model is widespread in the enterprises as a conceptual formalism to provide standard documentation for relational information systems, and many efforts have been done to use E/R schemes as the input for designing non-relational databases as well [5]; unfortunately, as argued in [10]:

"Entity relation data models [...] cannot be understood by users and they cannot be navigated usefully by DBMS software. Entity relation models cannot be used as the basis for enterprise data warehouses."

In this paper we present a graphical conceptual model for DWs, called *Dimensional Fact* (DF) model, and propose a semi-automated methodology to build it from the pre-existing E/R schemes describing an operational information system. The DF model is a collection of tree-structured fact schemes whose basic elements are facts, attributes, dimensions and hierarchies; other features which may be represented on fact schemes are the additivity of fact attributes along dimensions, the optionality of dimension attributes and the existence of non-dimension attributes. Compatible fact schemes may be overlapped in order to relate and compare data. Fact schemes may be integrated with information of the conjectured workload, expressed in terms of query patterns, to be used as the input of a design phase whose output are the logical and physical schemes of the DW.

In some cases, the E/R documentation is not available. The methodology we propose can be applied starting from the relational database scheme as well, provided that the multiplicities (-to-one or -to-many) of the logical associations established by foreign key constraints are known.

After surveying the literature on DWs in Section 2, in Section 3 we describe the DF model. Section 4 outlines a methodology for deriving fact schemes from E/R documentation, and Section 5 proposes an example in the health-care domain.

2. Background and literature on data warehousing

From a functional point of view, the data warehouse process consists in three phases: extracting data from distributed operational sources; organizing and integrating data consistently into the DW; accessing the integrated data in an efficient and flexible fashion. The first phase encompasses typical issues concerning distributed heterogeneous information services, such as inconsistent data, incompatible data structures, data granularity, etc. (for instance, see [16]). The third phase requires capabilities of aggregate navigation [7], optimization of complex queries [3], advanced indexing techniques [11] and friendly visual interface to be used for On-Line Analytical Processing (OLAP) [4] and data mining [6].

As to the second phase, designing the DW requires techniques completely different from those adopted for operational information systems. Most scientific literature on the design of DWs concerns their logical and physical models; the apparent lack of interest in the issues related to conceptual design can be explained as follows: (a) data warehousing was initially devised within the industrial world, as a result of practical demands of users who typically do not give predominant importance to conceptual issues; (b) logical and physical design have a primary role in optimizing the system performances, which is the main goal in data warehousing applications.

Data within a DW are organized according to the multidimensional model. In [12], the author proposes an approach to the design of DWs based on a business model of the enterprise which is actually a relational database scheme. Regretfully, conceptual and logical design are mixed up; since logical design is necessarily targeted towards a logical model (relational in this case), no unifying conceptual model of data is devised.

The multidimensional model may be mapped on the logical level differently depending on the underlying DBMS. If a DBMS supporting directly the multidimensional model is used, fact attributes are typically represented as the cells of multidimensional arrays whose indices are determined by key attributes [8]. On the other hand, in relational DBMSs the multidimensional model of the DW is mapped in most cases through star schemes [10] consisting of a set of *dimension tables* and a central *fact table*. Dimension tables are strongly denormalized and are used to select the facts of interest based on the user queries. The fact table stores fact attributes; its key is defined by importing the keys of the dimension tables.

Different versions of these base schemes have been proposed in order to improve the overall performances [1], handle the sparsity of data [13] and optimize the access to aggregated data [9]. In particular, the efficiency issues raised by data warehousing have been addressed by means of new indexing techniques (see [14] for a survey), among which we mention bitmap indices [13]. A bitmap index for attribute *a* is a matrix of bits with one column for each possible value *a* can assume and one row for each non-empty cell of a data-cube (or for each row of a fact table).

3. A conceptual model for data warehouses

In our approach, the conceptual model of a DW consists of a set of *fact schemes*. The basic components of fact schemes are *facts*, *dimensions* and *hierarchies*. A *fact* is a focus of interest for the enterprise; a *dimension* determines the granularity adopted for representing facts; a *hierarchy* determines how fact instances may be aggregated and selected significantly for the decision-making process.

A fact scheme is structured as a tree whose root is a fact. The fact is represented by a box which reports the fact name and, typically, one or more numeric and continuously valued attributes which "measure" the fact from different points of view. Figure 1 reports a small fact scheme for fact *SALE* in a store chain; *quantity sold* and *returns* are fact attributes.

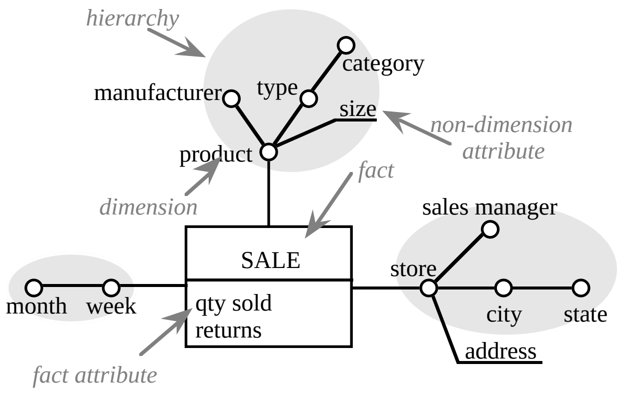

Figure 1. A simple three-dimensional fact scheme for a chain of stores.

Each vertex directly attached to the fact is a dimension. Subtrees rooted in dimensions are hierarchies. Their vertices, represented by circles, are attributes which may assume a discrete set of values; their arcs represent -to-one relationships between pairs of attributes. The dimension in which a hierarchy is rooted defines its finest aggregation granularity; the attributes in the vertices along each sub-path of the hierarchy starting from the dimension define progressively coarser granularities. The fact scheme in Figure 1 has three dimensions: *week*, *product* and *store*.

Some terminal vertices in the fact scheme may be represented by lines instead of circles (*size* and *address* in Figure 1); these vertices correspond to *non-dimension attributes*. A non-dimension attribute contains additional information about an attribute of the hierarchy, and is connected to it by a -to-one relationship; differently from the attributes of the hierarchy, it cannot be used for aggregation. For instance, aggregating sales according to the address of the store would not make sense; thus, *address* is represented as a non-dimension attribute.

Some arcs within the hierarchies may be marked by a dash: these arcs express optional relationships between pairs of attributes (see Section 5 for an example). Indicating explicitly optionality is useful for logical design: in fact, for each optional attribute, an extra record reporting a dummy value will be added to the corresponding dimension table.

A fact expresses a many-to-many relationship among the dimensions. Each combination of values of the dimensions defines a *fact instance*, characterized by exactly one value for each fact attribute; fact instances are the elemental information represented within the DW. In our example, a fact instance describes the quantity of one product sold during one week in one store, and the corresponding total returns.

A fact scheme may also have no fact attributes: in this case, each fact instance records the occurrence of an event. For example, consider a fact *ATTENDANCE* with dimensions *date*, *student*, *course* and *teacher*: a fact instance represents the fact that, on a given date, a student attends a course given by a teacher. Fact schemes with no fact attributes correspond, on the logical level, to *factless fact tables*, typically used for event tracking or as coverage tables [10].

3.1. Additivity

In general, querying an information system means linking different concepts through user-defined paths in order to retrieve some data of interest; in particular, for relational databases this is done by formulating a set of joins to connect relation schemes.

On the other hand, querying a DW is typically aimed at extracting summary data to fill a structured report to be analysed for decisional or statistical purposes. Since analysing data at the maximum level of detail is often overwhelming, it may be useful to aggregate fact instances into clusters at different levels of abstraction (*roll-up* in OLAP terminology). Aggregation requires to define a proper operator for composing the attribute values characterizing each fact instance into attribute values characterizing each cluster as a whole.

As a guideline, most fact attributes should be additive [10]. This means that the sum operator can be used to aggregate attribute values along all hierarchies. An example of additive attribute in the sale example is the number of sales: the number of sales for a given sales manager is the sum of the number of sales for all the stores managed by that sales manager.

A fact attribute is called *semi-additive* if it is not additive along one or more dimensions, *non-additive* if it is additive along no dimension. Examples of semi-additive attributes are all those measuring a level, such as an inventory level, a temperature, etc. An inventory level is non-additive along time, but it is additive along both the product and the store dimensions. A temperature attribute is non-additive, since adding up two temperatures hardly makes sense. However, this kind of semi- or non-additive attributes can still be aggregated by using operators such as average, maximum, minimum.

For other attributes, aggregation is inherently impossible for conceptual reasons. Consider an attribute *number of customers* in the sale example, and suppose it is estimated for a given product, day and store by counting the number of purchase tickets for that product printed on that day in that store. Since the same ticket may include other products, adding the number of customers for two or more products would lead to an inconsistent result. Thus, *number of customers* is non-additive along the product dimension (while it is additive along the time and the stores dimensions). In this case, the reason for non-additivity is that the relationship between purchase tickets and products is many-to-many instead of many-to-one. Aggregation cannot be consistently performed on the product dimension, whatever operator is used, unless the grain of fact instances is made finer.

In the DF model, attributes are additive along all the dimensions by default. Semi-additivity is represented explicitly by relating each semi- or non-additive attribute with the dimension(s) along which values cannot be added. If an aggregation operator (other than sum) can be used, it is indicated explicitly. Figure 2 shows an example.

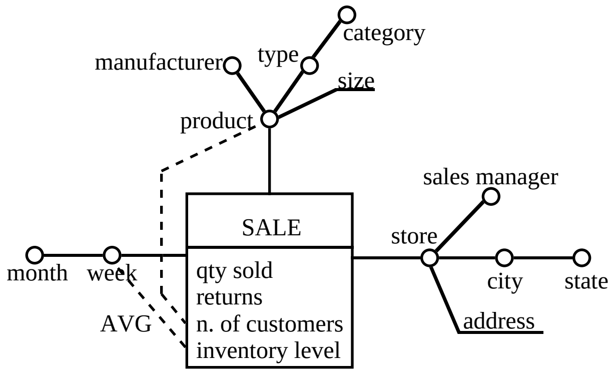

Figure 2. Semi-additivity on the fact scheme.

3.2. Overlapping compatible fact schemes

In our approach, different facts are represented in different fact schemes. However, part of the queries the user formulates on the DW may require comparing fact attributes taken from distinct, though related, schemes (*drill across* in OLAP terminology).

Two fact schemes are said to be *compatible* if they share at least one dimension attribute. Two compatible schemes *F* and *G* may be overlapped to create a resulting scheme *H*. In the simplest case, in which the interattribute dependencies in the two schemes are not conflicting:

- the set of the fact attributes in *H* is the union of the sets in *F* and *G*;
- the dimensions in *H* are the intersection of those in *F* and *G*, assuming that a given dimension is common to *F* and *G* if at least one dimension attribute is shared.
- each hierarchy in *H* includes all and only the dimension attributes included in the corresponding hierarchies of both *F* and *G*.

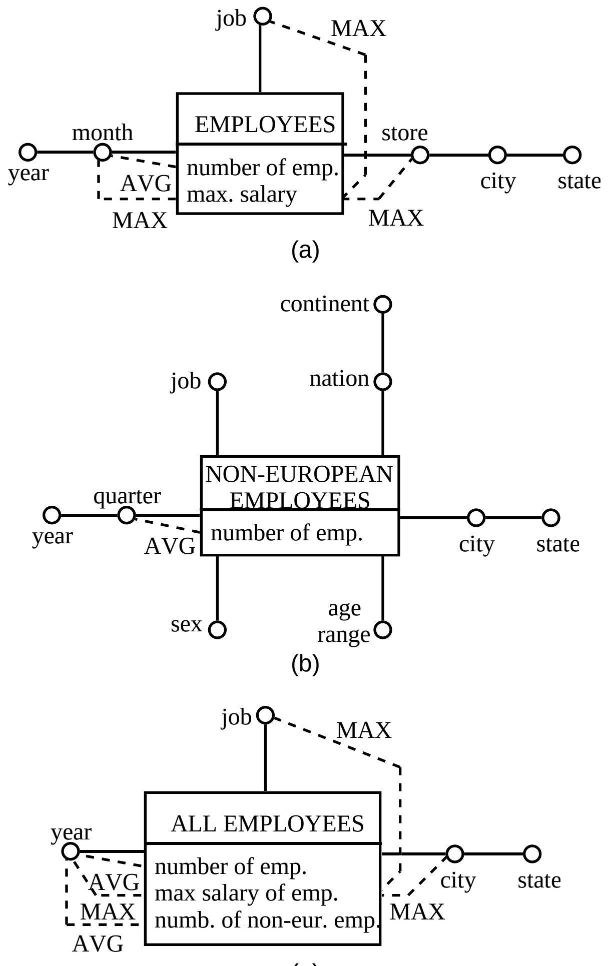

Figure 3. Scheme overlapping.

Consider the two fact schemes in Figures 3.a and 3.b: the first represents all employees of an enterprise, the second only the non-European employees. Although these schemes are aimed at extracting different information, they are compatible; in fact they share the time, job and store dimensions. The scheme resulting from overlapping is shown in Figure 3.c; it can be used, for instance, to calculate the percentage of non-European employees for each city, job and year.

In some cases, aggregation along a dimension can be carried out at different abstraction levels even if the corresponding dimension attributes were not explicitly shown. For instance, given a *month* attribute within a time hierarchy, fact instances can be aggregated by quarter, semester and year by performing a simple calculation. Thus, given the two compatible fact schemes in Figure 3, attribute *quarter* could in principle be added to the time dimension in the resulting scheme. On the other hand, the designer must keep in mind that, by adopting this solution, the time for extracting data by quarter will increase significantly; thus, the best solution would probably be to add explicitly the *quarter* attribute to the time hierarchy in the employee fact scheme.

3.3. Representing query patterns on a fact scheme

The basic OLAP operators for formulating typical queries on DWs are roll-up, drill down, drill across and slice-and-dice; they are used, respectively, to aggregate fact attributes in order to view data at a higher level of abstraction, disaggregate fact attributes in order to introduce further detail, relate and compare distinct facts, select and project facts so as to reduce their dimensionality [2].

On a fact scheme, a query may be represented by a *query pattern*, which consists in a set of markers placed on the dimension attributes. One or more markers can be placed within each hierarchy, to indicate at what level(s) fact instances must be aggregated. A dimension may also contain no markers, to indicate that none of its attributes is involved in the query. Non-dimension attributes need not be shown on the query pattern.

The data shown as a result of a query may be any combination of fact attributes, and/or the result of any computation made on them. Figure 4 shows the query pattern representing the following query: "*total quantity sold and average returns per unit sold for each week and for each type of product*". The average returns per unit sold is the ratio between the total returns and the quantity sold.

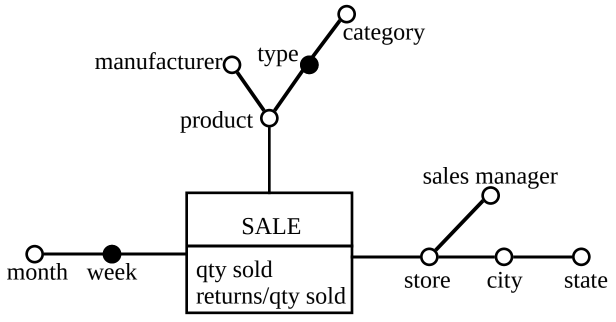

Figure 4. Query pattern.

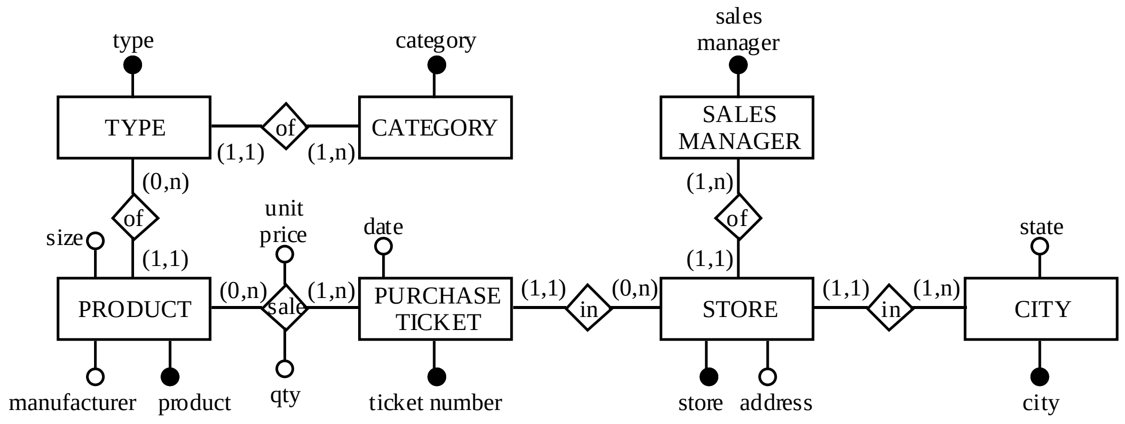

Figure 5. The (simplified) E/R scheme for the sale fact scheme.

A query pattern is well-formed if the fact attributes reported are compatible with the disposition of markers along the hierarchies. In particular, if an attribute which cannot be aggregated along a hierarchy is present, a marker must necessarily be placed on the corresponding dimension.

4. From E/R schemes to fact schemes

Most information systems implemented in enterprises during the last decade are relational, and in most cases their analysis documentation consists of E/R schemes. Thus, it seems natural to derive the conceptual model of a DW from the existing E/R schemes. The methodology we outline in this section to build a DF model consists of 5 steps:

1. Defining facts.
2. For each fact:
   a. Building the attribute tree.
   b. Pruning and grafting the attribute tree.
   c. Defining dimensions.
   d. Defining fact attributes.
   e. Defining hierarchies.

In the following subsections we will describe these steps referring to the sale example, for which a simplified E/R scheme is shown in Figure 5. Each instance of relationship `SALE` represents an item referring to a single product within a purchase ticket. Attribute `unitPrice` is placed on `SALE` instead of `PRODUCT` since the price of products may vary over time.

4.1. Defining facts

Facts are concepts of primary interest for the decision making process. Typically, they correspond to events occurring dynamically in the enterprise world.

A fact may be represented on the E/R scheme either by an entity `F` or by an *n*-ary relationship between entities E_1,...E_n. In the latter case, for the sake of simplicity, it is worth to transform the relationship into an entity `F` by replacing each branch E_i with a binary relationship between `F` and E_i; each binary relationship has multiplicity (1,1) on the side of `F` and (m_i,M_i) on the side of E_i, where $m_i \in \{0,1\}$ and $M_i \in \{1,n\}$ are, respectively, the minimum and the maximum multiplicity of branch E_i. The attributes of the relationship become attributes of `F`; the identifier of `F` is the combination of the identifiers of E_i, i=1,...n.

Entities or relationships representing frequently updated archives (such as `SALE`) are good candidates for defining facts; those representing structural properties of the domain, corresponding to nearly-static archives (such as `STORE` and `CITY`), are not.

Each fact identified on the E/R scheme becomes the root of a different fact scheme. In the following subsections, we will focus the discussion on a single fact, the one corresponding to entity `F`. In the sale example, the fact of primary interest for business analysis is the sale of a product, represented in the E/R scheme by relationship `sale`. Figure 6 shows how relationship `sale` is transformed into an entity.

4.2. Building the attribute tree

Given a portion of interest of an E/R scheme and an entity `F` belonging to it, we call *attribute tree* the tree such that:

- each vertex corresponds to an attribute of the scheme;
- the root corresponds to the identifier of `F`;
- for each vertex *v*, the corresponding attribute functionally determines all the attributes corresponding to the descendants of *v*.

The attribute tree will be used in the following subsections to build the fact scheme for the fact corresponding to `F`.

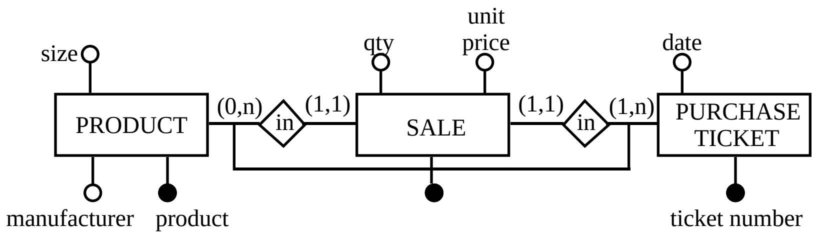

Figure 6. Transformation of relationship `sale` into an entity.

Let F be the entity chosen to represent a fact; the attribute tree may be constructed automatically by applying the following recursive procedure:

```
translate(F,identifier(F))
```

where

```
translate(E,v):
// E is the current entity, v is the current vertex
{  for each attribute a∈E | a≠identifier(F) do
      addChild(v,a);      // adds child a to vertex v
   for each entity G connected to E by a x-to-one relationship R do
   {  for each attribute b∈R do
         addChild(v,b);
      addChild(v,identifier(G));
      translate(G,identifier(G));
   }
}
```

If F is identified by the combination of two or more attributes, `identifier(F)` denotes their concatenation.

It is worth adding some further notes:

- It is useful to emphasize on the fact scheme the existence of optional relationships between attributes in a hierarchy. To this end, the arcs corresponding to optional relationships or optional attributes of the E/R scheme should be marked by a dash (see Section 5 for an example).
- A one-to-one relationship can be thought of as a particular kind of many-to-one relationship, hence, it can be inserted into the attribute tree. Nevertheless, in a DW query, drilling down along a one-to-one relationship means adding a row header to the result without introducing further detail; thus, it is often worth grafting from the attribute tree the attributes following one-to-one relationships (see Section 4.3), or representing them as non-dimension attributes.
- Generalization hierarchies in the E/R scheme are equivalent to one-to-one relationships between the super-entity and each sub-entity, and should be treated as such by the algorithm.
- x-to-many relationships cannot be inserted into the attribute tree. In fact, representing these relationships at the logical level, for instance by a star scheme, would be impossible without violating the first normal form.
- As already stated in Section 4.1, an *n*-ary relationship is equivalent to *n* binary relationships. Most *n*-ary relationships have maximum multiplicity greater than 1 on all their branches; in this case, they determine *n* one-to-many binary relationships which cannot be inserted into the attribute tree. On the other hand, a branch with maximum multiplicity equal to 1 determines a one-to-one binary relationship which can be inserted.

The attribute tree corresponding to the E/R scheme in Figure 5 is shown in Figure 7.

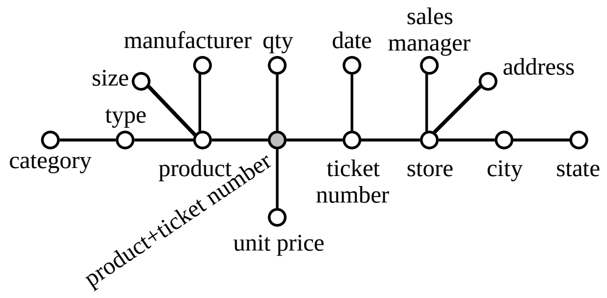

Figure 7. Attribute tree for the sale example (the root is in grey).

4.3. Pruning and grafting the attribute tree

Probably, not all of the attributes represented in the attribute tree are interesting for the DW. Thus, the attribute tree may be pruned and grafted in order to eliminate the unnecessary levels of detail.

Pruning is carried out by dropping any subtree from the tree. The attributes dropped will not be included in the fact scheme, hence, it will be impossible to use them to aggregate data. For instance, on the sale example, the subtree including *city* and *state* may be dropped.

Grafting is used when, though a vertex of the tree expresses an uninteresting information, its descendants must be preserved. For instance, one may want to classify products directly by category, without considering the information on their type. Let *v* be the vertex to be eliminated, and *v'* its father:

```
graft(v):
{  for each v" | v" is child of v do
      addChild(v',v");
   drop v;
}
```

Thus, grafting is carried out by moving the entire subtree with root in *v* to *v'*. As a result, attribute *v* will not be included in the fact scheme and the corresponding aggregation level will be lost; on the other hand, all the descendant levels will be maintained. In the sale example, the detail of purchase tickets is uninteresting; by grafting vertex *ticket number*, the attribute tree is transformed as shown in Figure 8. In general, grafting a child of the root corresponds to making the granularity of fact instances coarser and, if the node grafted has two or more children, leads to increasing the number of dimensions in the fact scheme.

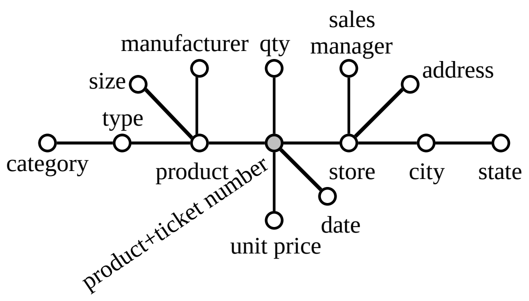

Figure 8. Attribute tree for the sale example after grafting vertex *ticket number*.

It should be noted that, when an optional vertex is grafted, all its children inherit the optionality dash.

4.4. Defining dimensions

Dimensions determine how fact instances may be aggregated significantly for the decision-making process. The dimensions must be chosen in the attribute tree among the children vertices of the root (including the attributes which have become children of the root after the tree has been grafted); they may correspond either to discrete attributes, or to ranges of discrete or continuous attributes. Their choice is crucial for the DW design since it determines the granularity of fact instances.

It is widely recognized that time is a key dimension for DWs. E/R schemes can be classified, according to the way they deal with time, into *snapshot* and *temporal*. A snapshot scheme describes the current state of the application domain; old versions of data varying over time are continuously replaced by new versions. On the other hand, a temporal scheme describes the evolution of the application domain over a range of time; old versions of data are explicitly represented and stored. When designing a DW from a temporal scheme, time is explicitly represented as an E/R attribute and thus it is an obvious candidate to define a dimension. Should time appear in the attribute tree as a child of some vertex different from the root, it is worth considering the possibility of grafting the tree in order to have time become a dimension (i.e., become a child of the root). In snapshot schemes, time is not explicitly represented (it is implicitly assumed that the scheme represents data at the current time); however, also for snapshot schemes time should be added as a dimension to the fact scheme.

In the sale example, the attributes chosen as dimensions are *product*, *store* and ranges of the *date* attribute corresponding to weeks.

At this stage, the fact scheme may be sketched by adding the chosen dimensions to the root fact.

4.5. Defining fact attributes

Fact attributes are typically either counts of the number of instances of `F`, or the sum/average/maximum/minimum of expressions involving numerical attributes of the attribute tree (the attributes chosen as dimensions for the fact scheme are excluded). A fact may have no attributes, if the only information to be recorded is the occurrence of the fact.

The fact attributes determined, if any, are reported on the fact scheme. At this step, it is useful for the phase of logical design to build a *glossary* which associates each fact attribute to an expression describing how it can be calculated from the attributes of the E/R scheme. Referring to the sale example, the glossary may be compiled as follows:

quantity sold = `SUM(SALE.qty)`
total returns = `SUM(SALE.qty * SALE.unitPrice)`
number of customers = `COUNT(SALE)`

If attribute `unitPrice` had been placed on entity `PRODUCT` in the E/R scheme:

total returns = `SUM(SALE.qty * PRODUCT.unitPrice)`

The aggregation operators are meant to work on all the instances of `SALE` which relate to the same week, store and product.

In some cases, aggregation is not necessary to define fact attributes, since it has already been executed at the relational level. For instance, each instance of entity `SALE` in the E/R scheme might describe the total sales for a given product, store and week; in this case, instances of the entity correspond one-to-one to fact instances, and entity attributes may be directly translated into fact attributes.

4.6. Defining hierarchies

The last step in building the fact scheme is the definition of hierarchies on dimensions. Along each hierarchy, attributes must be arranged into a tree such that a x-to-one relationship holds between each node and its descendants.

The attribute tree already shows a plausible organization for hierarchies; at this stage, it is still possible to prune and graft the tree in order to eliminate irrelevant details (for instance, in most cases, a vertex connected to its father by a one-to-one relationship is grafted). It is also possible to add new levels of aggregation by defining ranges for numerical attributes; typically, this is done on the time dimension. In the sale example, the time dimension is enriched by introducing attribute *month*, defined as a range of *week*.

During this phase, the attributes which should not be used for aggregation but only for informative purposes may be identified as non-dimension attributes.

5. The Hospital Example

In this section we apply our methodology for conceptual design of DWs to a real-world example. The E/R scheme shown in Figure 9 is part of the documentation for a typical health-care information system; in particular, it describes the admissions and their outcomes.

The attribute tree for fact *ADMISSION*, represented by entity `ADMISSION`, is shown in Figure 10.a. It should be noted that a *surgery* subtree has been created though the `causes` relationship is many-to-many. This can be done since the Boolean attribute `if main` on the relationship identifies one of the operations executed during the hospitalization as the principal one; thus, if only the main operations are considered, the `causes` relationship becomes many-to-one.

Transforming this attribute tree in the one shown in Figure 10.b requires four steps:

- grafting *date+time+op.th.* (one-to-one relationship);
- pruning *date of surgery, time of surgery* and *surgeon*;
- pruning the subtree rooted in *op.th.*;
- pruning *name* and *physician*;
- grafting *patCode.*

The fact scheme we have derived is shown in Figure 11. Dimension *month* is defined as a range on attribute *date*; dimension *age5* as a range of attribute *age* (5 years intervals). The hierarchies on dimensions *month* and *age5* are defined by adopting progressively wider ranges. Dimension *type of surgery* is optional. The glossary for the fact attributes is reported below:

number of admissions = `COUNT(ADMISSION)`
value = `SUM(has.value)`
number of days = `SUM(ADMISSION.nDays)`
score = `SUM(DRG.weight)`

All fact attributes are additive along all dimensions.

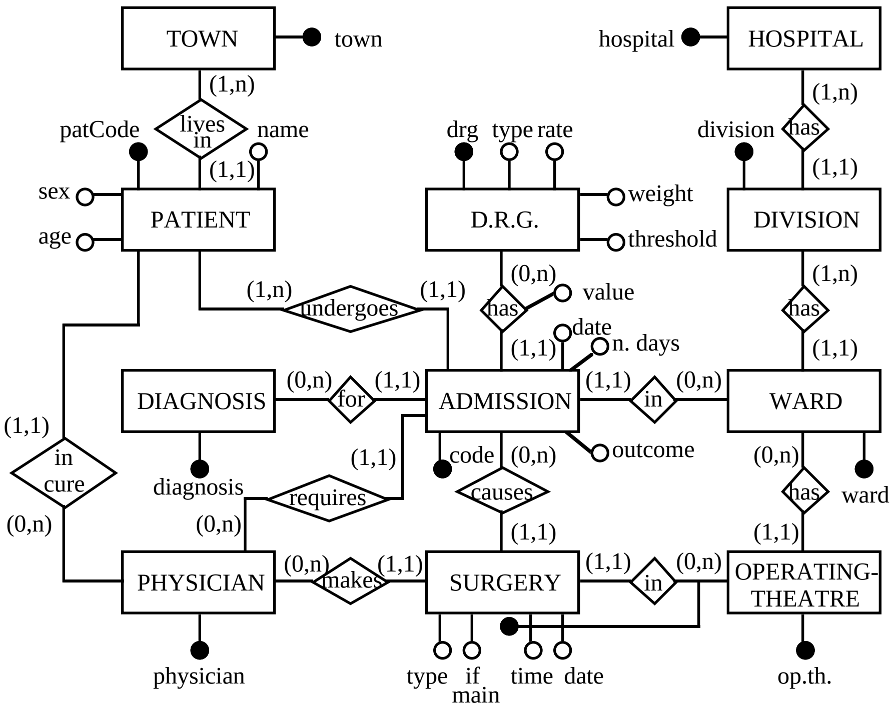

Figure 9. The hospital E/R scheme.

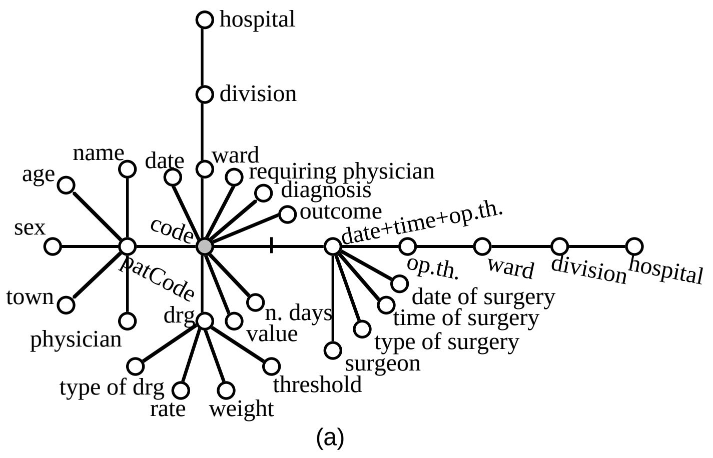

Figure 10. Attribute tree for the hospital E/R scheme.

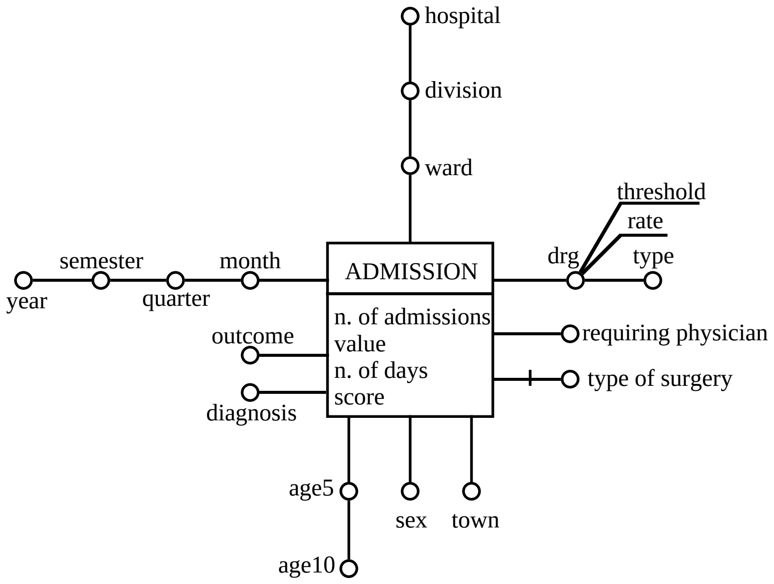

Figure 11. Fact scheme for hospitalizations.

6. Conclusion

In this paper we have proposed a conceptual model for data warehouse design and a semi-automated methodology for deriving it from the E/R documentation describing the information system of the enterprise.

The DF model is independent of the target logical model (multidimensional or relational); in order to bridge the gap between the fact schemes and the DW logical scheme, a methodology for logical design is needed. Like in operational information systems, DW logical design should be based on an estimate of the expected workload and data volumes. The workload will be expressed in terms of query patterns and their frequencies; data volumes will be computed by considering the sparsity of facts and the cardinality of the dimension attributes. Among the design topics specific of DWs, we mention the possibility of combining separate dimensions into one by defining composite keys, the recognition of degenerate dimensions for which no corresponding dimension table is needed, the choice of the aggregated fact tables to be introduced in a constellation scheme, the partitioning of the DW into integrated data marts, etc. Our future work will be devoted to developing the methodology for logical design and implementing it within an automated tool.

References

[1] R. Barquin, and S. Edelstein. *Planning and Designing the Data Warehouse*. Prentice Hall, 1996.

[2] S. Chaudhuri, and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, vol. 26, n. 1, pp. 65-74, 1997.

[3] S. Chaudhuri, and K. Shim. Including group-by in query optimization. In *Proc. 20th Int. Conf. on Very Large Data Bases*, pp. 354-366, 1994.

[4] G. Colliat. OLAP, relational and multidimensional database systems. *SIGMOD Record*, vol. 25, n. 3, pp. 64-69, 1996.

[5] C. Fahrner, and G. Vossen. A survey of database transformations based on the Entity-Relationship model. *Data & Knowledge Engineering*, vol. 15, n. 3, pp. 213-250. 1995.

[6] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Data mining and knowledge discovery in databases: an overview. *Comm. of the ACM*, vol. 39, n.11, 1996.

[7] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data-warehousing environments. In *Proc. 21th Int. Conf. on Very Large Data Bases*, Zurich, Switzerland, 1995.

[8] V. Harinarayan, A. Rajaraman, and J. Ulman. Implementing Data Cubes Efficiently. In *Proc. of ACM Sigmod Conf.*, Montreal, Canada, 1996.

[9] T. Johnson, and D. Shasha. Hierarchically split cube forests for decision support: description and tuned design. *Bullettin of Technical Committee on Data Engineering*.vol. 20, n. 1, 1997.

[10] R. Kimball. *The data warehouse toolkit*. John Wiley & Sons, 1996.

[11] D. Lomet, and B. Salzberg. The Hb-Tree: a multidimensional indexing method with good guaranteed performance. *ACM Trans. On Database Systems*, vol. 15, n. 44, pp.625-658, 1990.

[12] F. McGuff. Data modeling for data warehouses. October 1996. http://members.aol.com/fmcguff/dwmodel/dwmodel.htm.

[13] P. O'Neil, and G. Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, vol. 24, n. 3, pp. 8-11, 1995.

[14] S. Sarawagi. Indexing OLAP data. *Bullettin of Technical Committee on Data Engineering*.vol. 20, n. 1, 1997.

[15] J. Widom. Research Problems in Data Warehousing. In *Proc. 4th Int. Conf. on Information and Knowledge Management*, Nov. 1995.

[16] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *Proc. Conference on Parallel and Distributed Information Systems*, Miami Beach, FL, 1996.