# A Methodological Framework for Data Warehouse Design

Matteo Golfarelli
DEIS - University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
+39-547-642862
golfare@csr.unibo.it

Stefano Rizzi
DEIS - University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
+39-51-6443542
srizzi@deis.unibo.it

## ABSTRACT

Though designing a data warehouse requires techniques completely different from those adopted for operational systems, no significant effort has been made so far to develop a complete and consistent design methodology for data warehouses. In this paper we outline a general methodological framework for data warehouse design, based on our Dimensional Fact Model (DFM). After analyzing the existing information system and collecting the user requirements, conceptual design is carried out semi-automatically starting from the operational database scheme. A workload is then characterized in terms of data volumes and expected queries, to be used as the input of the logical and physical design phases whose output is the final scheme for the data warehouse.

## Keywords

Data warehouse, design methodology, conceptual model.

## 1. INTRODUCTION

The database community is devoting increasing attention to the research themes concerning data warehouses (DWs); nevertheless, the crucial issues related to DW design have not been deeply investigated yet [14].

Designing a DW requires techniques completely different from those adopted for operational information systems. While most scientific literature on the design of DWs focuses on specific issues such as multidimensional data models [1] [8], materialization of views [2] [9] and index selection [7] [10], no significant effort has been made so far to develop a complete and consistent design methodology [12]. In [3] the different phases in DW design are described informally, but no *ad hoc* conceptual model to support them is devised.

In this paper we outline a general methodological framework for DW design, based on the conceptual DW model we developed, called *Dimensional Fact Model* (DFM). The methodology we propose features 6 phases, briefly sketched in Table I and examined in Sections from 2 to 7. The issues we have already addressed are discussed in more detail; in particular, Subsection 4.1 formalizes the DFM, while Subsection 5.1 includes the definition of workload on the DFM. As to the issues we are currently investigating, an overview of the main research topics is provided.

*Proceedings ACM First International Workshop on Data Warehousing and OLAP (DOLAP 98), Nov. 7, 1998, Washington, D.C., USA.*

## 2. ANALYSIS OF THE INFORMATION SYSTEM

The aim of this phase is to collect the documentation concerning the pre-existing operational information system. It involves the designer, in tight collaboration with the people involved in managing the information system and, if possible, with its designers, and produces in output the (conceptual or logical) schemes of either the whole or part of the information system.

Typically, this phase entails integration of heterogeneous views. This topic has been largely dealt with in the database literature, see [15] for a comprehensive survey.

## 3. REQUIREMENT SPECIFICATION

This phase consists in collecting and filtering the user requirements. It involves the designer and the final users of the DW, and produces in output the specifications concerning the choice of facts on the one hand, preliminary indications concerning the workload on the other.

In particular, the choice of facts is based on the documentation on the information system produced at the previous step. Facts are concepts of primary interest for the decision-making process, and typically correspond to events occurring dynamically in the enterprise world. If the operational information system is documented by E/R schemes, a fact may be represented either by an entity or by an n-ary relationship; conversely, if it is represented by relational schemes, facts correspond to relation schemes. In general, entities or relation schemes representing frequently updated archives are good candidates for defining facts; those representing structural properties of the domain, corresponding to nearly-static archives, are not.

The preliminary workload is expressed in pseudo-natural language and is aimed at enabling the designer to identify dimensions and measures during conceptual design; for each fact, it should specify the most interesting measures and aggregations.

## 4. CONCEPTUAL DESIGN

This phase is carried out starting from the schemes of the operational information system and considering the facts and the preliminary workload defined at the previous step. It produces a *dimensional scheme*, structured according to the DFM, which consists of a set of *fact schemes*, one for each fact.

In [5] we proposed a semi-automated technique to carry out conceptual modelling starting, respectively, from the E/R schemes and from the logical relational schemes describing the operational information system. In both cases, the following steps must be executed to produce each fact scheme:

| *Step* | *Input* | *Output* | *Involves* |
|---|---|---|---|
| **Analysis of the information system** | existing documentation | database scheme | designer; managers of the information system |
| **Requirement specification** | (database scheme) | facts; preliminary workload | designer; final users |
| **Conceptual design** | database scheme; facts; preliminary workload | dimensional scheme | designer |
| **Workload refinement, Dim. scheme validation** | dimensional scheme; preliminary workload | workload | designer; final users |
| **Logical design** | dimensional scheme; target logical model; workload | logical DW scheme | designer |
| **Physical design** | logical DW scheme; target DBMS; workload | DW physical scheme | designer |

**Table I. The six phases in the DW design methodology.**

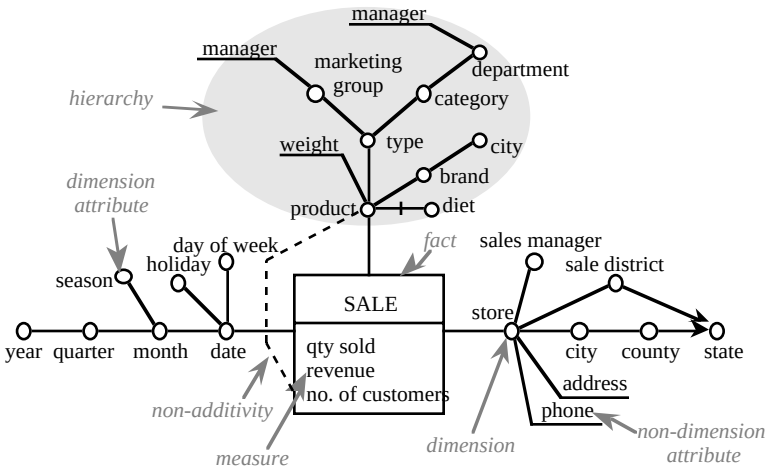


**Figure 1. The *SALE* fact scheme.**

a. *Building the attribute tree.*

b. *Pruning and grafting the attribute tree.*

c. *Defining dimensions.*

d. *Defining measures.*

e. *Defining hierarchies.*

Steps b, c and d are supported by the preliminary workload declared by the final users.

## 4.1 The Dimensional Fact Model

The representation of reality built using the DFM is called *dimensional scheme* and consists of a set of *fact schemes* whose basic elements are facts, dimensions and hierarchies.

**Definition 1**. Let g=(V,E) be a directed, acyclic and weakly connected graph. We say g is a *quasi-tree* with root in $v_0 \in V$ if each other vertex $v_j \in V$ can be reached from $v_0$ through at least one directed path. We will denote with $path_{ij}(g) \subseteq g$ a directed path (if it exists) starting in $v_i$ and ending in $v_j$; we will denote with $sub(v_i) \subset g$ the quasi-tree rooted in $v_i \neq v_0$.

**Definition 2**. A fact scheme is a six-tuple

$$f = (M, A, N, R, O, S)$$

where:

- M is a set of *measures*; each measure $m_i \in M$ is defined by a numerical or Boolean expression which involves values acquired from the information systems.
- A is a set of *dimension attributes*. Each dimension attribute $a_i \in A$ is characterized by a discrete domain of values, $Dom(a_i)$.
- N is a set of *non-dimension attributes*.
- R is a set of ordered couples, each having the form $(a_i,a_j)$ where $a_i \in A \cup \{a_0\}$ and $a_j \in A \cup N$ ($a_i \neq a_j$), such that the graph $qt(f) \doteq (A \cup N \cup \{a_0\}, R)$ is a quasi-tree with root $a_0$. $a_0$ is a dummy attribute playing the role of the *fact* on which the scheme is centred. The couple $(a_i,a_j)$ models a -to-one relationship between attributes $a_i$ and $a_j$. We call *dimension pattern* the set $Dim(f) \doteq \{a_i \in A \mid \exists (a_0,a_i) \in R\}$; each element in Dim(f) is called a *dimension*. When we need to emphasize that a dimension attribute $a_i$ is a dimension, we will denote it as $d_i$. We call *hierarchy* on dimension $d_i \in Dim(f)$ the quasi-tree $sub(d_i)$.
- $O \subset R$ is a set of *optional* relationships.
- S is a set of *aggregation statements*, each consisting of a triple $(m_j, d_i, \Omega)$ where $m_j \in M$, $d_i \in Dim(f)$ and $\Omega$ is an *aggregation operator*. Statement $(m_j, d_i, \Omega) \in S$ declares that measure $m_j$ can be aggregated along dimension $d_i$ by means of $\Omega$. If no aggregation statement exists for a given pair $(m_j, d_i)$, then $m_j$ cannot be aggregated at all along $d_i$.

In the following we will discuss the different components introduced above with reference to the fact scheme *SALE*, shown in Figure 1, which describes the sales in a chain store.

From a graphical point of view, a fact scheme is structured as a

quasi-tree whose root is a fact. A *fact* is represented by a box which reports the fact name and, typically, one or more numeric and continuously valued measures (in the sale scheme, *quantity sold*, *revenue* and *no. of customers*).

*Dimension attributes* are represented by circles and may assume a discrete set of values. Each dimension attribute directly attached to the fact is a *dimension*; dimensions determine the granularity adopted for representing facts. The dimension pattern of the sale scheme is {*date*, *product*, *store*}.

Subtrees rooted in dimensions are *hierarchies*, and determine how fact instances may be aggregated and selected significantly for the decision-making process. The dimension in which a hierarchy is rooted defines its finest aggregation granularity; the dimension attributes in the vertices along each path of the hierarchy starting from the dimension define progressively coarser granularity. The arc connecting two attributes represents a -to-one relationship between them (for instance, there is a many-to-one relationship between *city* and *county*); thus, every directed path within one hierarchy necessarily represents a -to-one relationship between the starting and the ending attributes. We denote with $\alpha_i.a_j$ the value of $a_j$ determined by value $\alpha_i \in Dom(a_i)$ assumed by $a_i$ (for instance, `Venice`.*state* denotes `Italy`).

The fact scheme may not be a tree: in fact, two or more distinct paths may connect two dimension attributes within a hierarchy, provided that still every directed path represents a -to-one relationship. Consider for instance the hierarchy on dimension *store*: states are partitioned into counties and sale districts, and no relationship exists between them; nevertheless, a store belongs to the same state whichever of the two paths is followed (i.e., *city* determines *state*). Thus, notation $\alpha_i.a_j$ explained above is still not ambiguous even if two or more paths connect $a_i$ to $a_j$. Whenever two or more arcs enter the same attribute, arrows are used to convey the direction of the -to-one relationships.

Some terminal vertices in the fact scheme may be represented by lines instead of circles (for instance, *address* in Figure 1); these vertices correspond to the *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.

The arcs marked by a dash express optional relationships between pairs of attributes. For instance, attribute *diet* takes a value only for food products.

A measure is *aggregable* on a dimension if its values can be aggregated along the corresponding hierarchy by at least one operator; an aggregable measure is *additive* if its values can be aggregated by the sum operator. Since most measures are additive, in order to simplify the graphic notation in the DFM, only the exceptions are represented explicitly. In particular, if $m_j$ is not additive along $d_i$, $m_j$ and $d_i$ are connected by a dashed line labelled with all operators $\Omega$ (if any) such that $(m_j, d_i, \Omega) \in S$ (for instance, in Figure 1, measure *no. of customers* is not aggregable along dimension *product*).

## 4.2 Fact instances

Given a fact scheme f, each n-ple of values taken from the domains of its n dimensions defines an elemental cell where one unit of information for the DW can be represented. We call *primary fact instances* the units of information present within the DW, each characterized by exactly one value for each measure. We denote with $pf(\alpha_1,...\alpha_n)$ the primary fact instance corresponding to the combination of values $(\alpha_1,...\alpha_n) \in Dom(d_1) \times ... \times Dom(d_n)$. In the sale scheme, each primary instance describes the sales of one product during one day in one store.

Since analysing data at the maximum level of detail is often overwhelming, it may be useful to aggregate primary fact instances at different levels of abstraction, each corresponding to an aggregation pattern.

**Definition 3**. Given a fact scheme f with n dimensions, a v-dimensional *aggregation pattern* is a set P of v dimensional attributes such that no directed path exists within qt(f) between each pair of attributes in P (that is, each attribute in P is functionally independent of the others). A dimension $d_i \in Dim(f)$ is said to be *hidden* within P if no attribute of its hierarchy appears within P. An aggregation pattern P is *legal* with reference to measure $m_j \in M$ if

$$\forall\, d_k \mid \exists\, (m_j, d_k, \Omega) \in S \;\; d_k \in P$$

Examples of aggregation patterns in the sale scheme are {*product*,*county*,*month*}, {*state*,*date*} where *product* is hidden, {} where all dimensions are hidden. Pattern {*brand*,*month*} is illegal with reference to *no. of customers* since the latter cannot be aggregated along the product hierarchy.

An aggregation pattern declares how primary fact instances should be aggregated. If a given dimension is not interesting for the current analysis, aggregation is carried out over all the possible values the corresponding dimension can assume. Let $P=\{a_1,...a_v\}$ be an aggregation pattern, and $d_{h*}$ denote the dimension whose hierarchy includes $a_h \in P$. The *secondary fact instance* $sf(\beta_1,...\beta_v)$ corresponding to the combination of values $(\beta_1,...\beta_v) \in Dom(a_1) \times ... \times Dom(a_v)$ aggregates the set of primary fact instances

$$\{\, pf(\alpha_1,...\alpha_n) \mid \forall k \in \{1,...n\}\ \alpha_k \in Dom(d_k) \wedge \forall h \in \{1,...v\}\ \alpha_{h*}.a_h = \beta_h \,\}$$

and is characterized by exactly one value for each measure for which P is legal. This value is calculated by applying an aggregation operator to the values that measure assumes within the primary fact instances aggregated. In the sale scheme, an example of secondary instance is the one describing the sales of products of a given category during one day in a city. Figure 2 shows the corresponding primary fact instances; measure *no. of customers* is not reported since it is non-aggregable along the product dimension.

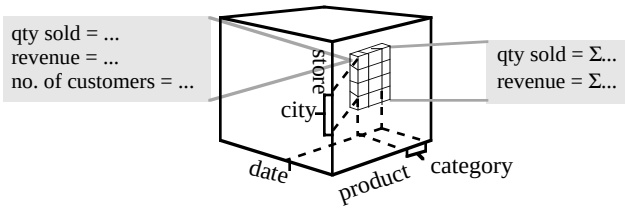


**Figure 2. The primary fact instances aggregated by a secondary fact instance.**

In the following, we will use sometimes the term *pattern* to denote either the dimension pattern or an aggregation pattern.

## 4.3 Overlapping fact schemes

In the DFM, different facts are represented in different fact schemes. However, part of the queries the user formulates on

the DW may require comparing measures taken from distinct, though related, schemes (*drill-across*). In this subsection we show how two related fact schemes may be combined into a new scheme; since the same attribute $a_i$ may appear within different fact schemes, possibly with different domains, we will denote with $Dom_f(a_i)$ the domain of $a_i$ within scheme f.

**Definition 4**. Two fact schemes f'=(M',A',N',R',O',S') and f"=(M",A",N",R",O",S") are said to be *compatible* if they share at least one dimension attribute: $A' \cap A" \neq \varnothing$. Attribute $a_i$ is considered to be common to f' and f" if, within the two schemes, it has the same semantics and if $Dom_{f'}(a_i) \cap Dom_{f"}(a_i) \neq \varnothing$.

**Definition 5**. Given a quasi-tree $t=(V \cup \{a_0\},E)$ with root $a_0$, and a subset of vertices $I \subseteq V$, we define the *contraction* of t on I as the quasi-tree $cnt(t,I) \doteq (I \cup \{a_0\},E^*)$ where

$$E^* = \{(a_i,a_j) \mid a_i \in I \cup \{a_0\} \wedge a_j \in I \wedge \exists path_{ij}(t) \wedge \forall a_k \in I - \{a_i,a_j\}\ a_k \notin path_{ij}(t)\}$$

The arcs of cnt(t,I) are the directed paths which, inside t, connect pairs of vertices of I without including other vertices of I.

Figure 3 shows a quasi-tree and its contraction on a subset of the vertices.

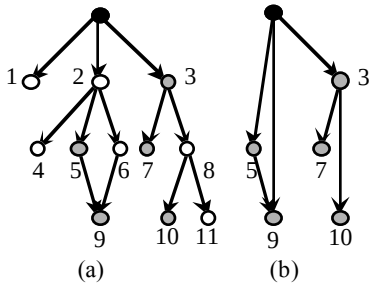


**Figure 3. A quasi-tree (a) and its contraction on the grey vertices (b); the root is in black.**

**Definition 6**. Let two compatible fact schemes f'=(M',A',N',R',O',S') and f"=(M",A",N",R",O",S") be given, and let $I=A' \cap A"$. Schemes f' and f" are said to be *strictly compatible* if cnt(qt(f'),I) and cnt(qt(f"),I) are equal.

Two compatible schemes f' and f" may be overlapped to create a resulting scheme f; if the compatibility is strict, the inter-attribute dependencies in the two schemes are not conflicting and f may be defined as follows.

**Definition 7**. Given two strictly compatible schemes f' and f", we define the *overlap* of f' and f" as the scheme $f' \otimes f"=(M,A,N,R,O,S)$ where:

$M = M' \cup M"$
$A = A' \cap A"$
$\forall a_i \in A\ (Dom_{f' \otimes f"}(a_i) = Dom_{f'}(a_i) \cap Dom_{f"}(a_i))$
$N = N' \cap N"$
$R = \{(a_i,a_j) \mid (a_i,a_j) \in cnt(qt(f'),A)\}$
$\quad = \{(a_i,a_j) \mid (a_i,a_j) \in cnt(qt(f"),A)\}$
$O = \{(a_i,a_j) \in R \mid \exists (a_w,a_z) \in O' \mid (a_w,a_z) \in path_{ij}(qt(f')) \vee \exists (a_w,a_z) \in O" \mid (a_w,a_z) \in path_{ij}(qt(f"))\}$
$S = \{(m_j,d_i,\Omega) \mid d_i \in Dim(f' \otimes f") \wedge (\exists (m_j,d_k,\Omega) \in S' \wedge d_i \in sub(qt(f'),d_k)) \vee (\exists (m_j,d_k,\Omega) \in S" \wedge d_i \in sub(qt(f"),d_k))\}$

Figure 4 shows the overlapping between the two strictly compatible schemes *INVENTORY* and *SHIPMENT*, which share the time and the product dimensions. The scheme resulting from overlapping can be used, for instance, to compare the quantities shipped and stored for each product.

# 5. WORKLOAD REFINEMENT AND SCHEME VALIDATION

This phase is primarily aimed at refining the preliminary workload by reformulating it in deeper detail on the dimensional scheme; Subsection 5.1 defines a simple language to denote queries according to the DFM. Another significant aspect, discussed in Subsection 5.2, concerns the computation of the expected data volumes. Both the query workload and the data volumes will have a crucial role in guiding logical and physical design.

This phase is also aimed at validating the conceptual scheme produced at the previous step; in fact, the query workload can be exhaustively and correctly expressed only if the dimensions and measures have been properly identified and hierarchies are well-structured.

## 5.1 Queries

Within our framework, a typical DW query can be represented by the set of fact instances, at any aggregation level, whose measure values are to be retrieved. In this subsection we discuss how sets of fact instances can be denoted by writing *fact instance expressions* having the general form:

<*fact instance expression*> ::=
    <*fact name*> ( <*pattern clause*> ; <*selection clause*> )
<*pattern clause*> ::= comma-list of <*pattern elements*>
<*pattern elements*> ::= <*dimension name*> |
    <*dimension name*>.<*attribute name*>
<*selection clause*> ::= comma-list of <*predicate*>

The pattern clause describes a pattern. The selection clause contains a set of Boolean predicates which may either select a subset of the aggregated fact instances or affect the way fact instances are aggregated. If an attribute involved either in a pattern clause or in a selection clause is not a dimension, it should be referenced by prefixing its dimension name.

The value(s) assumed by a measure within the fact instance(s) described by a fact instance expression is(are) denoted as follows:

<*measure values*> ::= <*fact instance expression*>.<*measure*>

Given a fact scheme f having n dimensions $d_1,...d_n$, consider the fact instance expression

$$f(d_1,...d_p,a_{p+1},...a_v\ ;\ e_1(b_{i1}),...e_h(b_{ih})) \qquad (1)$$

where we have assumed, without loss of generality, that the first p aggregation statements involve a dimension and the other v−p involve a dimension attribute. Each Boolean predicate $e_j$ involves one attribute $b_{ij}$ belonging to the hierarchy rooted in $d_{ij}*$, which may also be hidden.

If p=v=n (i.e., the pattern clause describes the dimension pattern), expression (1) denotes the set of primary fact instances

$$\{ pf(\alpha_1,...\alpha_n) \mid \forall k \in \{1,...n\}\ \alpha_k \in Dom(d_k) \wedge \forall j \in \{1,...h\}\ e_j(\alpha_{ij}*.b_{ij}) \}$$

For instance,

*SALE*(*date*, *product*, *store*; *date.year*>='1995', *product*='P5').*qtySold*

denotes the quantities of product P5 sold in each store during the days of the years since 1995.

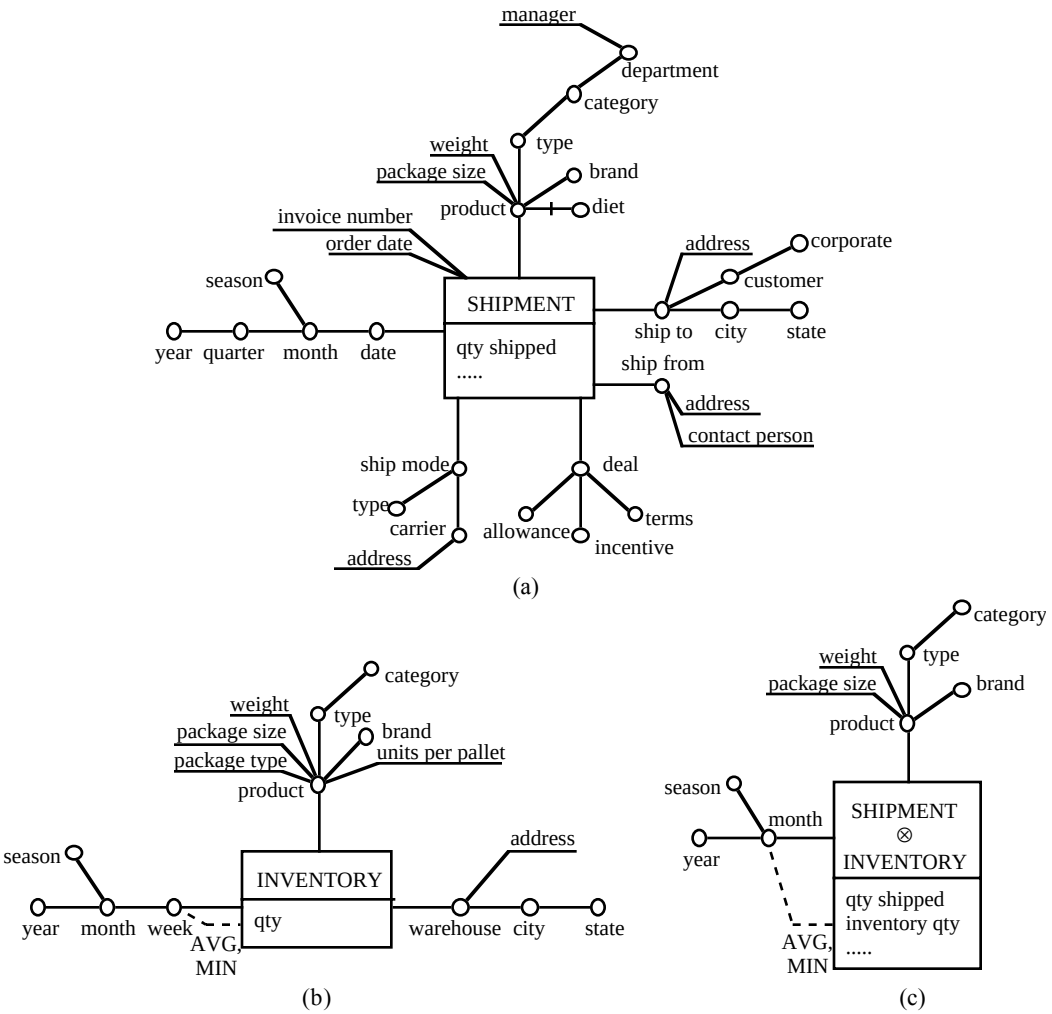


**Figure 4. The *SHIPMENT* scheme (a), the *INVENTORY* scheme (b) and their overlap (c).**

Otherwise ($p<v$ and/or at least one dimension is hidden), let P be the aggregation pattern described by the pattern clause. Let $b_{i_j}$ be the attribute involved by $e_j$; we say $e_j$ is *external* if $\exists a_{i_j*} \in P \mid a_{i_j*} \in path_{0i_j}(qt(f))$, *internal* otherwise. External predicates restrict the set of secondary fact instances to be returned, while internal predicates determine which primary fact instances will form each secondary fact instance. Let $e_1,...e_r$ and $e_{r+1},...e_h$ be, respectively, the external and the internal predicates ($0 \le r \le h$); in this case, expression (1) denotes the set of secondary fact instances

$$\{ sf(\beta_1,...\beta_v) \mid \forall k \in \{1,...v\}\ \beta_k \in Dom(a_k) \wedge \forall j \in \{1,...r\}\ e_j(\beta_{i_j}.b_{i_j}) \}$$

where each $sf(\beta_1,...\beta_v)$ aggregates the set of primary fact instances

$$\{ pf(\alpha_1,...\alpha_n) \mid \forall k \in \{1,...n\}\ \alpha_k \in Dom(d_k) \wedge \forall h \in \{1,...v\}\ \alpha_{h*}.a_h = \beta_h) \wedge \forall j \in \{r+1,...h\}\ e_j(\alpha_{i_j*}.b_{i_j}) \}$$

For instance, the expressions

*SALE*(*date.month*, *product.type* ; *date.month*='JAN98', *product.category*='food').*qtySold*
*SALE*(*date.month*, *product.type* ; *date.month*='JAN98', *product.brand*='General').*qtySold*

denote, respectively, the total sales of each type of products of category 'food' for January 1998 and the total sales of each type of products of brand 'General' for January 1998. The predicates on *month* and on *category* are external, whereas that on *brand* is internal.

The DW workload will typically include drill-across queries, that is, queries formulated on the overlap of two or more schemes. Let q be defined by the fact instance expression f(P;<sel>) where $f=f_1 \otimes ... \otimes f_m$. Each fact instance returned by q is the concatenation of m fact instances returned by the m queries $q_1,...q_m$, where $q_i=f_i(P;<sel>,d_1 \in Dom_f(d_1),... d_n \in Dom_f(d_n))$ and $d_1,...d_n$ are the dimensions of f. An example of drill-across query is:

*SHIPMENT*⊗*INVENTORY*(*month*,*product* ; *month.year*='1997').*inventoryQty*−*qtyShipped*

**Definition 8**. The workload W on a dimensional scheme is a set of pairs $(q_i,v_i)$, where $q_i$ denotes a query and $v_i$ its expected frequency. Each query is represented by a fact instance expression f(P;<sel>).M where f is a fact scheme (elemental or overlapped), M is a set of measures of f, <sel> is a selection clause and pattern P is legal with reference to every measure $m \in M$.

## 5.2 Data volumes

Primary data volumes are computed for each fact scheme f by considering the sparsity of facts and the cardinality of the dimension attributes. Let $nk_{a_i}$ denote the domain cardinality of attribute $a_i$ within f; the maximum number of primary fact instances is

$$cp = \prod_{d_i \in Dim(f)} nk_{d_i}$$

We denote with np the actual number of primary fact instances, which we assume to be known; typically, np<<cp.

In most DW cost models (see for instance [8]), the cost of a query q is assumed to be proportional to the number of n-ples in the view on which q is executed. Since views may be materialized at any level of abstraction, it is necessary to estimate the number of secondary fact instances corresponding to an aggregation pattern P. The maximum number of secondary fact instances corresponding to P is

$$cs(P) = \prod_{a_i \in P} nk_{a_i}$$

The actual number of secondary fact instances, ns(P), may be estimated using the Yao formula [16]:

$$ns(P) = cs(P) \times \left( 1 - \frac{\left( cp - \frac{cp}{cs(P)} \right)}{\binom{cp}{np}} \right)$$

When cp/cs(P) is sufficiently large, this formula is well approximated by the Cardenas formula [4]:

$$ns(P) \approx cs(P) \times \left( 1 - \left( 1 - \frac{1}{cs(P)} \right)^{np} \right)$$

In the following we consider a numerical example from the *SALE* scheme. Let $nk_{product}$=1000, $nk_{date}$=1000 (about 3 years) and $nk_{store}$=100, which implies $cp=10^8$; let $np=10^6$. Consider the aggregation pattern P={*product.type*,*date*, *store.city*}, where $nk_{typte}$=100 and $nk_{city}$=50, which implies $cs(P)=5\times10^6$. In this case, the Cardenas formula yields ns(P)=906347.

# 6. LOGICAL DESIGN

Several issues must be addressed in order to obtain a correct definition of the DW logical scheme. Logical design receives in input a dimensional scheme, a workload and a set of additional information (update frequencies, total disk space available, etc.) to produce a DW scheme which should minimize the query response times by respecting the disk space constraint. In this context, we believe that update queries should be considered separately from the workload. In fact, DWs are typically updated only periodically, in an off-line fashion, and during this process the warehouse is unavailable for querying. Thus, the update process does not affect directly the DW performance, and it is sufficient to ensure that it is properly bounded in time.

At this time, it is necessary to choose the target logical model, relational or multidimensional; in this paper, we consider only the relational case. A dimensional scheme can be mapped on the relational model by adopting the well-known star scheme [11]; it may be convenient to snowflake one or more dimensions, depending on the cardinality of the domains.

The definition of an accurate cost model has a primary role in correctly evaluating the system performance. During logical design, adopting a simplistic model may cause gross mistakes while adopting one too accurate could make the design very complex. In our approach, different cost models at increasing levels of detail will support the different design steps.

In the following subsections we outline the sequence of the logical design steps.

## 6.1 View materialization

A technique commonly used in order to reduce the overall response time is to pre-compute (consolidate) the information that can be useful to answer frequent queries. Fact tables reporting data consolidated from other fact tables are often called *views*; each view allows the costs for a set of queries to be reduced but leads to additional update costs and disk space occupancy.

A massive work has been done in the literature on view materialization; see for instance [6] and [2] where a *multidimensional lattice* is defined for each fact, based on a partial ordering relationship between the aggregation patterns. In general, the materialization problem is faced by considering each single lattice separately. On the other hand, we claim that the whole dimensional scheme should be involved; in fact, drill-across queries may weigh significantly upon the workload and cannot be optimized on one lattice at a time. Typically, a drill-across query executed by retrieving data from two or more views defined on the same pattern though on different fact schemes; on the other hand, if some of these views were unified into a single fact table, the access costs could be significantly reduced.

The cost model adopted at this step is based on the number of logical accesses made to both fact and dimension tables in order to solve a query; the access technique adopted for each query (e.g., indices vs. sequential scan) is not taken into account.

## 6.2 Translation into tables

During this phase, the fact and dimension tables are created starting from the dimensional scheme and according to the logical model adopted. In the simplest case, in which the classic star scheme is adopted, each fact scheme f = (M,A,N,R,O,S) having Dim(f)={$d_1$,...$d_n$} and M={$m_1$,...$m_z$} is translated into one fact table

FT_f($\underline{k_1}$,...$\underline{k_n}$,$m_1$,...$m_z$)

and n dimension tables

DT_$d_1$($\underline{k_1}$,$a_{11}$,...$a_{1v_1}$,$a'_{11}$,...$a'_{1u_1}$)
.....
DT_$d_n$($\underline{k_n}$,$a_{n1}$,...$a_{nv_n}$,$a'_{n1}$,...$a'_{nu_n}$)

(where the hierarchy on $d_i$ includes the dimension attributes $a_{i1}$,...$a_{iv_i}$ and the non dimension attributes $a'_{i1}$,...$a'_{iu_i}$).

The star scheme for the *SALE* example turns out to be:

FT_SALE(<u>prodKey</u>,<u>dateKey</u>,<u>storeKey</u>,qtySold,
revenue,noOfCustomers)
DT_PROD(<u>prodKey</u>,product,weight,diet,brand,city,
type,category,department,deptManager,...)
DT_DATE(<u>dateKey</u>,date,dayOfWeek,holiday,month,..)
DT_STORE(<u>storeKey</u>,store,phone,address,
salesManager,city,county,state,saleDistrict)

## 6.3 Vertical partitioning of fact tables

Fact schemes usually include several measures that describe the same fact but, in practice, are seldom requested together. Vertical partitioning aims at reducing the global query response time by optimizing the queries requiring a subset of measures. Given a fact table $FT\_f(k_1,...k_n,m_1,...m_z)$, vertical partitioning is carried out by defining a partitioning of the set of measures $M=\{m_1,...m_z\}$ into $\alpha$ subsets ($\alpha \geq 2$) and by splitting accordingly $FT\_f$ into $\alpha$ tables each containing the complete key $k_1,...k_n$ and one of the measure subsets.

The problem of determining the optimal partitioning given a workload has been widely investigated within the context of centralized as well as distributed database systems [13]. Unfortunately, the results reported in the literature cannot be applied to the DW case since the redundancy introduced by materializing views binds the partitioning problem to that of deciding on which view each query should be executed.

The cost function used here takes into account the reduced cost in accessing shorter tuples as well as the overhead in accessing multiple tables.

## 6.4 Horizontal partitioning of fact tables

Horizontal partitioning aims at reducing the query response time by considering the selectivity of each query. In fact, most queries will not access all the n-ples within the fact table, but only a subset determined by a selection predicate involving one or more dimension attributes. Given a fact table $FT\_f$, horizontal partitioning is carried out by determining an optimal set of dimension attributes and by reallocating the n-ples in $FT\_f$ to a set of tables $FT\_f_1,...FT\_f_\beta$ each having the same relation scheme as $FT\_f$ and associated to a given element of the Cartesian product between the domains of the dimension attributes involved.

The problem of horizontal partitioning for relational databases is addressed in [13]. Also in this case, the problem is made more complex by the presence of views; the cost function takes into account the reduced cost in accessing smaller tables.

# 7. PHYSICAL DESIGN

The main issue in physical design concerns the optimal selection of indices, which is based on both the logical scheme and the workload and requires the specific access structures provided by the DBMS to be taken into account. Index selection has a crucial role in determining the DW performance; due to its high complexity, it is usually solved heuristically [7] [10].

Since DWs are updated off-line, the indices may be periodically reorganized in an optimal clustered form. Maintaining indices in the presence of concurrent updates is not necessary, and adopting more complex access structures becomes possible. In particular, besides traditional value-list indices such as B-trees, data warehousing systems usually support *bitmap index*, *join index* and *projection index*.

The index selection phase is aimed at determining the best subset of indices for a given workload and considering, for each type of index, an appropriate cost function. The best subset is the one that minimizes the access cost for the queries in the workload under a space constraint varying form application to application. Since DW queries usually require one or more joins to be executed, index selection should consider the different join algorithms, the most used being nested loop, sort merge and simple hash join. As to update queries we believe that, similarly to logical design, an upper bound should be placed on the total update time.

# 8. CONCLUSION

In this paper we outlined a general methodological framework for DW design, based on the Dimensional Fact Model. While conceptual design and workload definition have already been achieved and tested on a set of sample applications, we are currently studying effective algorithms for logical and physical design.

# REFERENCES


[1] Agrawal, R., Gupta, A., and Sarawagi, S. Modeling multidimensional databases. IBM Research Report, 1995.

[2] Baralis, E., Paraboschi, S., and Teniente, E. Materialized view selection in multidimensional database, in Proc. 23rd VLDB (Athens, Greece, 1997), 156-165.

[3] Cabibbo, L., and Torlone, R. Un quadro metodologico per la costruzione e l'uso di un data warehouse, in Proc. Sesto Convegno Nazionale sui Sistemi Evoluti per Basi di Dati (Ancona, Italy, 1998), 1, 123-140.

[4] Cardenas, A.F. Analysis and performance of inverted database structures. Comm. ACM, 18, 5, 253-263, 1975.

[5] Golfarelli, M., Maio, D., and Rizzi, S. Conceptual design of data warehouses from E/R schemes, in Proc. HICSS-31, VII, (Kona, Hawaii, 1998), 334-343.

[6] Gupta, H. Selection of views to materialize in a data warehouse, in Proc. Int. Conf. on Database Theory (Athens, Greece, 1997).

[7] Gupta, H., Harinarayan, V., and Rajaraman, A. Index selection for OLAP, in Proc. Int. Conf. Data Engineering (Binghamton, UK, 1997).

[8] Gyssens, M., and Lakshmanan, L.V.S. A foundation for multi-dimensional databases, in Proc. 23rd VLDB (Athens, Greece, 1997), 106-115.

[9] Harinarayan, V., Rajaraman, A., and Ulman, J. Implementing Data Cubes Efficiently, in Proc. of ACM Sigmod Conf. (Montreal, Canada, 1996).

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

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

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

[13] Özsu, M.T., and Valduriez, P. Principles of distributed database systems. Prentice-Hall Int. Editors, 1991.

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

[15] Wiederhold, G. et al. Integrating artificial intelligence and database technologies. Journal of Intelligent Information Systems, Special issue: Intelligent Integration of Information, 6, 2/3, 1996.

[16] Yao, S.B. Approximating block accesses in database organizations. Comm. ACM, 20, 4, 260-261, 1977.