Journal of Computer Science and Information Management, Vol. 2, N. 3, 1999

Designing the Data Warehouse: Key Steps and Crucial Issues

Matteo Golfarelli

DEIS - University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
golfare@csr.unibo.it

Stefano Rizzi

DEIS - University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
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 DW design discussing the relationships between the different steps and the difficulties in carrying them out. Within this framework, conceptual design is based on the Dimensional Fact Model, while logical design exploits multiple cost functions at increasing levels of detail in order to improve both the efficiency and efficacy of the algorithms. A workload is 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. In particular, drill-across queries are explicitly taken into account throughout the design steps.

Keywords

Data warehouse, design methodology, conceptual model, cost function, workload.

• INTRODUCTION

The database community is devoting increasing attention to the research themes concerning data warehouses (DWs); most scientific literature concerning DWs focuses on specific issues such as multidimensional data models [1] [14], materialization of views [2] [15] and index selection [13] [16]. It is easy to note that most of the interest has been gathered by those practical subjects that mainly determine the DW performances, while more conceptual issues like requirement specification and conceptual models have not been investigated enough [31]. The lack of interest in such abstract issues is probably a consequence of the practical requirements of enterprises where early DW projects were developed. However, we believe that a methodological framework for design is an essential requirement to ensure the success of complex projects. This can be easily confirmed by analyzing the statistic reports related to DWs project failures [17] [29], which state that a major cause lies in the absence of a global view of the design process and, in other terms, in the absence of an organic methodology. In this direction, our research is aimed at defining the basic steps required for a correct DW design.

To the best of our knowledge, no significant effort has been made so far to develop a complete and consistent design methodology [23]. In [5] the different phases in DW design are described informally, but no *ad hoc* conceptual model to support them is devised. Most of the techniques adopted in designing operational systems do not fit DW requirements and lead to miss some aspects which are not relevant in the conventional systems but, due to the different targets, become fundamental in data warehousing. For example, the E/R conceptual model, that is widely used for designing operational systems, does not explicitly show which are the hierarchies and the measures related to a fact.

The aim of this paper is to outline the basic phases in DW design motivating the sequence of actions, discussing the relationships between the different steps and the difficulties in carrying them out. Our methodological framework is 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 the following.

• ANALYSIS OF THE INFORMATION SYSTEM

The aim of this phase is to collect the documentation concerning the pre-existing operational information system. It represents a new step with respect to the design of operational systems, which do not require a pre-existing database as a data source. This phase involves the designer, in tight collaboration with the people managing the information system and, if possible, with its designers. It produces in output the (conceptual or logical) schemes of either the whole or part of the information system. In particular, the conceptual schemes are usually unreliable and sometimes unavailable at all due to a poor design of the operational system or as the result of undocumented changes to the database structure.

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 our DW design methodology.

While analyzing the operational system the designer should:

- Exploit the experience of the database manager in order to find out possible missing or abnormal data. In particular, correct handling of null values is an essential precondition for the quality of the query answers. A null value may mean that, at the time the measurement was taken, either the value did not exist or, conversely, that the measurement process failed to deliver the data [19]. This difference must be well understood by the designer when (s)he plans the DW loading process.

- Select operational data sources by considering the data quality and the stability of their schemes. Other equivalent data sources should be determined as well, in order to allow the so-called *view synchronization* to take place. View synchronization algorithms [21][24][20] exploit meta-data to gather consistent information even after a change in the operational system scheme caused the basic data sources to be no more available.
- Determine which data can be usefully integrated in order to obtain a complete view of the database domain. Integration of heterogeneous views has been largely dealt with in the database literature, see [32] for a comprehensive survey.
- Deeply understand the data semantic in order to make cross-footing possible during the data staging process.

While the logical and physical phases are characterized by a high computational complexity and conceptual design involves most of the crucial choices, the analysis of the information system will require most of the time due to its complexity and to the huge volume of data that should be gathered.

• REQUIREMENT SPECIFICATION

While the first phase of our methodology deals with the input for the DW process, the second one concerns its output. In fact, it 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.

It is well-known that most queries on DWs are extemporary; nevertheless, we believe that logical and physical design should be based on an expected workload including at least all the queries used in the enterprise to generate periodical reports. 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. By comparing the availability of data collected in the previous phase and the preliminary workload, it is also possible to understand which of the user requirements can be satisfied. This knowledge triggers off an iterative dialogue between the designer and the users, aimed at finding out alternative solutions when unavailability of data makes user requirements unachievable.

• CONCEPTUAL DESIGN

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.

Despite its importance, conceptual design is one of the less discussed subjects within the DW literature. In fact, besides our early works [9][10], only few papers focus on this issue. In [6] the authors propose as a conceptual model an extension of the multidimensional model described in [4]; a declaratory query language is introduced and its expressiveness is evaluated. However, fact tables are used to support conceptual design, thus blurring the borders between conceptual and logical design. In [22] an object-oriented model called Nested Multidimensional Data Model has been proposed, which extends the classical multidimensional model to allow complex OLAP scenarios to be modeled. In [30] the authors present an object-oriented conceptual model based on the *fact class* and *dimension class* concepts, including both static and dynamic properties.

Within our methodology, if the two previous phases have been successfully carried out, the operational system scheme and the preliminary workload should be available to the designer who will be able to carry out conceptual design almost autonomously. Assuming that the operational system is relational, in [9] we proposed a semi-automated technique to carry out conceptual modeling starting, respectively, from the E/R schemes and from the logical schemes describing it. In both cases, the following steps must be executed for each fact:

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 defined together with the final users or derived by analyzing the reports the DW users most frequently ask for.

The Dimensional Fact Model

The representation of reality built using the DFM is called *dimensional scheme* and consists of a set of *fact schemes* (one for each fact) whose basic elements are facts, dimensions and hierarchies. The dimensional fact model is aimed at:

- Efficiently supporting conceptual design
- Providing an expressive environment where the user can intuitively formulate queries
- Allowing the designer and the users to discuss constructively in order to refine the requirement specification
- Representing a solid platform for the logical design phase
- Providing a non ambiguous and expressive *a posteriori* documentation

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 operational information system.
- 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, Ω) where $m_j \in M$, $d_i \in Dim(f)$ and Ω 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 Ω. 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.

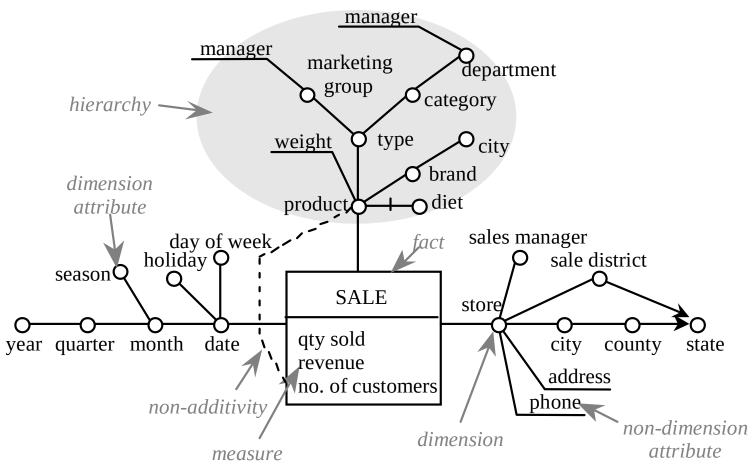


Figure 1. The SALE fact scheme.

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*}.

Sub-trees 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 labeled with all operators Ω (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*).

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 analyzing 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

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

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 the 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.

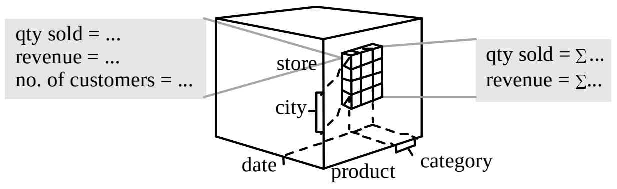


Figure 2. The primary fact instances aggregated by a secondary fact instance.

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

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 can 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.

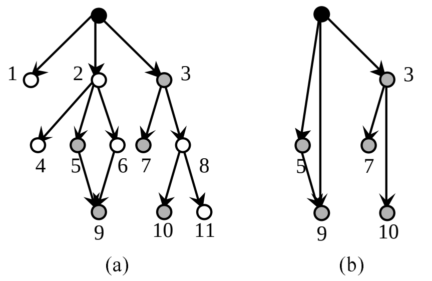


Figure 3. A quasi-tree (a) and its contraction on the grey vertices (b); the root is in black.

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'⊗f''; if the compatibility is strict, the inter-attribute dependencies in the two schemes are not conflicting and f'⊗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'⊗f''=(M,A,N,R,O,S) where:

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

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.

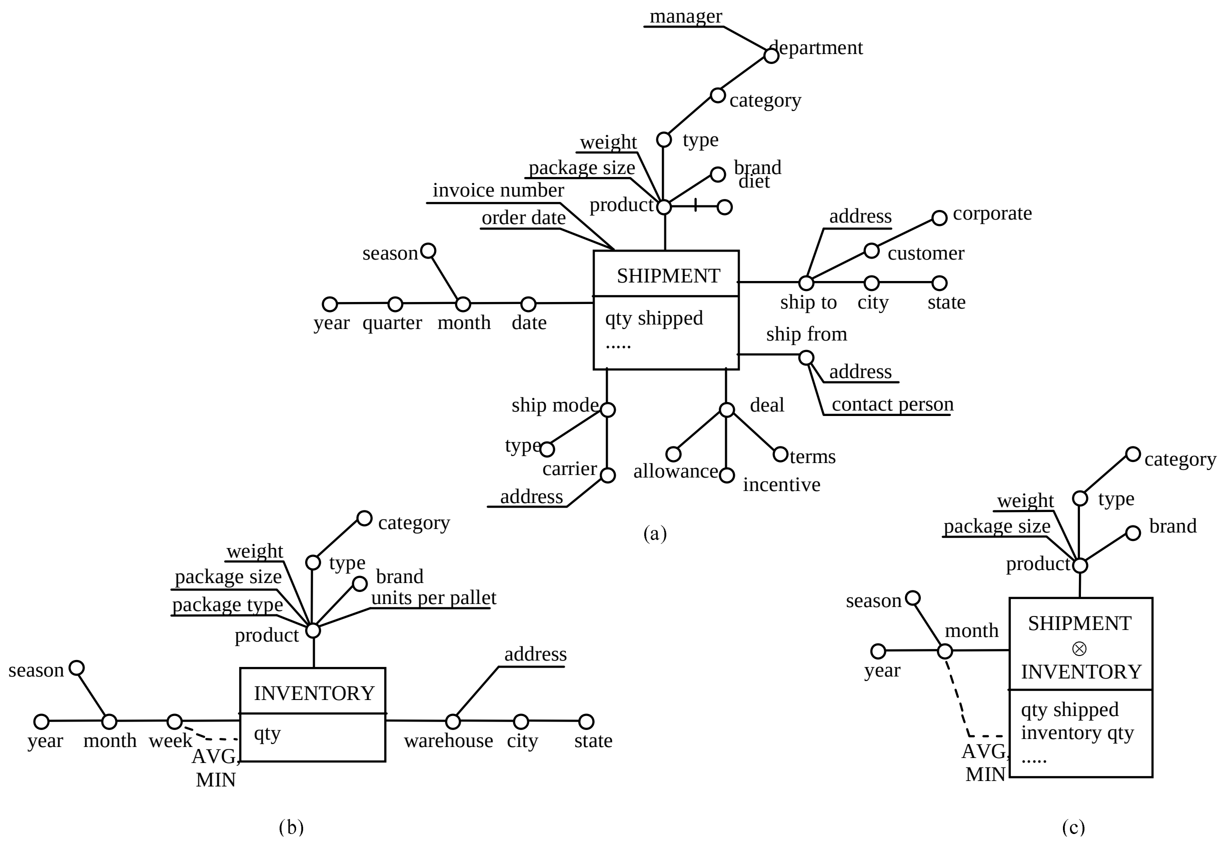


Figure 4. The *SHIPMENT* scheme (a), the *INVENTORY* scheme (b) and their overlap (c).

• 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; the next subsection defines a simple language to denote queries according to the DFM. Another significant aspect 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.

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. These information, together with the corresponding meta-data (i.g. the DBMS data type lengths), appear in the cost functions adopted within the logical and physical design steps, thus impacting on the optimization process. 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

$$f(d_1,...d_p,a_{p+1},...a_v \,;\, e_1(b_{i_1}),...e_h(b_{i_h})) \qquad (1)$$

where we have assumed, without loss of generality, that the first p pattern elements involve a dimension and the other v−p involve a dimension attribute. Each Boolean predicate e_j involves one attribute b_{i_j} belonging to the hierarchy rooted in $d_{i_{j*}}$, 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

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

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.

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. In particular, internal attributes have great importance in defining the granularity of data involved in the query solution. In fact, the aggregation pattern is not sufficient to determine the required level of detail that can be made finer by the internal attributes contained in the selection clauses.

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>).G where f is a fact scheme (elemental or overlapped), G is a set of measures of f, <sel> is a selection clause and pattern P is legal with reference to every measure $m\in G$.

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 [14]), 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 [33]:

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

When cp/cs(P) is sufficiently large, this formula is well approximated by the Cardenas formula [7]:

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

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.

• 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 availability, 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. Although in this paper we consider only the relational case (which represents, at the moment, the most popular choice), several intermediate approaches have been developed (e.g. Deductive OLAP, Hybrid OLAP, etc.). In particular, a new trend in logical/physical models consists in adopting the object-oriented paradigm in order to combine the advantages of MOLAP and ROLAP and, at the same time, eliminate their disadvantages [3].

A dimensional scheme can be mapped on the relational model by adopting the well-known star scheme [18]; 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 a too accurate one could make the design very complex. In our approach different cost models at increasing levels of detail will support the different design steps. Obviously, the different cost functions are not conflicting with each other but they evaluate the access cost with a different granularity.

The following subsections outline the sequence of the logical design steps; each step optimizes a different aspect in the DW structure. Dealing with different but related problems separately induces a sub-optimal solution, nevertheless a *divide-and-conquer* approach must be adopted due to the huge computational cost of an integrated solution. As a matter of fact, we do not propose any specific optimization algorithm since this is out of the scope of the paper; our aim is instead to emphasize and motivate the basic principles that must be considered regardless of the technique that will be adopted.

View materialization

A technique commonly used 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 [11] 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. Since the computational complexity arising when multiple lattices are optimized at the same time cannot be successfully afforded, drill-across query optimization will be carried out in two phases. During the view materialization step drill-across queries are split into separate queries on the different fact schemes involved. Let q=f (P;<sel>).G be a drill-across query operating on the overlapped fact scheme $f= f_1\otimes\ldots\otimes f_n$. By splitting q we will obtain n new queries $q_1\ldots q_n$ defined as follow

$$q_j=f_j(P;<sel>).G_j$$

where f_j is an elemental fact scheme, $G_j = G \cap M_j$, (M_j is the set of measures for f_j). Splitting drill-across queries allows the real workload to be better estimated without considering all the related fact schemes at once. If, at the end of this step, two views with the same aggregation pattern are materialized on both f_i and f_j, it will be possible to unify them. This possibility will be considered during the vertical partitioning step.

Since the final table structure is not defined yet, the cost model adopted at this step should be based on the number of logical accesses made to both fact and dimension tables in order to solve a query. The tables that will be actually accessed are not strictly defined by the aggregation pattern but are obviously selected from the set of materialized views.

Definition 9. Let P_i and P_j be two distinct patterns on fact scheme f; we say that P_i is *projectable* on P_j ($P_i < P_j$) iff $\forall a_h\in P_i$ ($\exists a_k\in P_j \mid a_h\in sub(qt(f),a_k)$). Given P_i and P_j such that $P_i < P_j$, we define the *projection* of P_i on P_j as

$$P_i\rightarrow P_j=\{(a_k\in P_j \mid P_i \cap sub(qt(f),a_k) \neq\varnothing \}$$

For example, with reference to the sale fact scheme, the pattern P_i={*category*, *state*} is projectable on P_j={*category*, *city*, *month*} (and it is $P_i\rightarrow P_j$={*category*, *city*}) but not on P'_j={*marketing group*, *month*}. The pattern P_i={*category*} is projectable on P_j={*type*, *brand*} (and it is $P_i\rightarrow P_j$={*type*}).

The cost of the query q, defined on pattern P and executed on a materialized view with pattern P_k ($P < P_k$) is defined by:

$$C(q,P_k) = ns(f.P_k)sel + \sum_{a_j\in P\rightarrow P_k} nk_{a_j} \times sel_j \qquad (2.a)$$

$$sel = \prod_{a_j\in P\rightarrow P_k} sel_j \qquad (2.b)$$

where sel_j is the selectivity of the predicates (if any) within the selection clause of q.

Differently from the cost function adopted in [2] and [15], ours is not necessarily monotonic with respect to the cardinality of the fact table used (i.e. $card(v_i)<card(v_k) \not\Rightarrow C(q,P_i)<C(q,P_k)$); as a consequence, the best pattern between two can be identified without computing the cost function only if one of the

patterns is projectable on the other. Consider in fact query

q = *SALE*(*product*,*date*; *product.category*="food")

and suppose that two views corresponding, respectively, to P'={*product*,*date*,*state*} and P"={*product*,*date*,*sales manager*} have been materialized (P' is not projectable on P", and vice versa); let $nk_{state} < nk_{sales\ manager}$ and ns(P')>ns(P') (several sale managers do not sell food). Both P' and P" can be used to answer q, but in this case it is not necessarily C(q,P')>C(q, P").

As to update cost, please note that with the term *update query* we denote the queries used to create a materialized view starting from a fact table or another materialized view. We do not never consider within the workload the queries necessary to update the fact tables starting from the operational data; in fact, since we assume that the fact tables containing the primary fact instances are always materialized, their update cost cannot be avoided and thus it cannot be optimized.

Let t be the time bound for the view update process and V the number of pages per time unit that can be read or written by the system. Thus, the constraint on the update cost can be formulated as:

$$\frac{\sum_{v_j \in F} Cu(v_j, F)}{V} \leq t$$

where F is the set of materialized views and $Cu(v_j,F)$ is defined as follows:

$$Cu(v_j, F) = C(q_{v_j}, P_{v_j}) + ns(f_j.P_j) \cdot (|f_j.P_j| + 1)$$

The first term within the sum defines the cost in reading the data necessary to update v_j from its closest materialized view[1], having pattern P_{v_j}. The second term expresses the cost for writing on v_j: for each fact instance, one read access for each dimension table and one write access on the fact table are made.

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

$$\mathtt{FT\_f}(\underline{k_1}, \ldots \underline{k_n}, m_1, \ldots m_z)$$

and n dimension tables

$$\mathtt{DT\_d_1}(\underline{k_1}, a_{11}, \ldots a_{1v_1}, a'_{11}, \ldots a''_{1u_1})$$
$$\ldots\ldots\ldots\ldots$$
$$\mathtt{DT\_d_n}(\underline{k_n}, a_{n1}, \ldots a_{nv_n}, a'_{n1}, \ldots a'_{nu_n})$$

(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(prodKey,dateKey,storeKey,qtySold,revenue,noOfCustomers)

DT_PROD(prodKey,product,weight,diet,brand,city,type,category,department,
        deptManager,...)

DT_DATE(dateKey,date,dayOfWeek,holiday,month,..)

DT_STORE(storeKey,store,phone,address,salesManager,city,county,state,
         saleDistrict)
```

By snowflaking one ore more dimensions better performances can be achieved [19]. In fact, snowflaking leads to (partial) normalization of the dimension tables since it cuts the functional dependencies induced by hierarchies; fact table data are split accordingly. Performance improvements derive from the reduced dimension of the accessed tables; on the other hand in a completely snowflaked scheme solving a query will require several expensive joins between dimension tables in order to filter the requested tuple-ids. Snowflaking must be transparent to the users, thus, it is possible only if the DBMS adopted is capable of determining which tables can be more conveniently accessed[2]. As a general rule, snowflaking aggregation levels correspond to the materialized view aggregation levels. It should be noted that, even if these two design choices are strictly related, they do not always coincide, thus a snowflaked fact table can still contain data at different aggregation levels.

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 $\mathtt{FT\_f}(\underline{k_1}, \ldots \underline{k_n}, m_1, \ldots m_z)$, vertical partitioning is carried out by defining a partitioning of the set of measures M={m_1,...m_z} into γ subsets ($\gamma \geq 2$) and by splitting accordingly `FT_f` into γ tables each containing the complete key $k_1, \ldots k_n$ 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 [27]. 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.

Let us consider for example a couple of queries q_1=f(P,<sel>).G_1, q_2=f(P,<sel>).G_2 requiring different sets of measures, and two materialized views v_1 and v_2 with patterns P_1 and P_2 respectively ($P \rightarrow P_2 \rightarrow P_1$). Though both views can be used to solve the queries, the optimal solution does not necessarily require the queries to be executed on v_2. In fact, if v_1 and v_2 have been differently partitioned, it may be less expensive to execute one of the queries on v_1: a cost reduction will arise if the extra-cost due to the finer granularity of v_1 is smaller than the saving obtained by reading better fitting tuples (see Figure 5). The final solution for vertical

[1] It is easy to show [27] that the global update cost is minimized if each view is updated after those with a finer pattern.

[2] The DBMS module that achieves this task is commonly called *aggregate navigator* [11].

partitioning includes determining on which view(s) each query will be executed.

A second element that makes vertical partitioning in DWs more difficult is the presence of drill-across queries. Their optimization requires all the tables obtained from related fact schemes to be considered at the same time. The optimal sets of measures are not necessarily subsets of the set of measures of the single fact tables but may be obtained by merging measures from different facts.

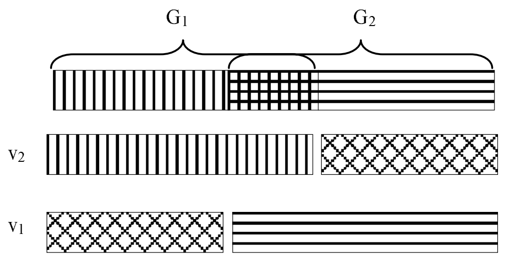


Figure 5. Vertical partitioning for two views v_1 and v_2. While q_1 can be surely executed more efficiently by accessing v_2, it is not obvious whether it is convenient to solve q_2 on v_2 or on v_1.

Merging measures coming from two different facts f_1 and f_2 is subordinated to the following constraints on the corresponding fact tables:

- They must have the same aggregation pattern
- They must have been selected for materialization during the materialization view optimization

Since optimization is based on the dimension of the accessed tuples, a cost function based on logical accesses is no more useful since by applying vertical partitioning the number of logical accesses to the tables will be always increased. Thus, the cost function used here takes into account the reduced cost in accessing shorter tuples as well as the overhead in accessing multiple tables. Let α be the cost in accessing a table descriptor, β the number of tuples within a disk page for the unpartitioned table, vf_k the number of fragments defined on the view materialized on pattern P_k and φ_i the fraction of these fragments used to answer the query q_i; thus the cost function can be defined as follows:

$$C(q_i, P_k) = \alpha \cdot \lceil vf_k \cdot \varphi_i \rceil + \frac{ns(P_k)}{\beta \cdot vf_k} \cdot \lceil vf_k \cdot \varphi_i \rceil$$

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 $FT\_f_1, \ldots FT\_f_m$ each having the same relation scheme as FT_f and associated to one or more elements of the Cartesian product between the domains of the dimension attributes involved.

The problem of horizontal partitioning for relational databases is addressed in [27]. Also in this case, the problem is made more complex by the presence of views. Both *primary* and *secondary* horizontal partitioning can be useful:

- Primary horizontal partitioning is based on a selection predicate involving an attribute belonging to both the fact table and the dimension tables (i.e. a foreign key k_i on the fact table) and can produce a cost reduction for the queries selecting a subset of the tuples from the fact table according to the values assumed by the i-th dimension.
- Secondary horizontal partitioning is based on a selection predicate involving an attribute a_j belonging only to a dimension table. It can be useful for the queries containing selection clauses which involve a_j or other attributes functionally determined by a_j.

The cost function adopted in this phase should take the reduced cost in accessing smaller tables into account. Let α be the cost in accessing a table descriptor, β the number of tuples within a disk page for the unpartitioned table, hf_k the number of fragments defined on the view materialized on pattern P_k and φ_i the fraction of these fragments used to answer the query q_i; thus the cost function can be defined as follows:

$$C(q_i, P_k) = \alpha \cdot \lceil hf_k \cdot \varphi_i \rceil + \frac{ns(P_k)}{hf_k \cdot \beta} \cdot \lceil hf_k \cdot \varphi_i \rceil$$

• 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.

With respect to conventional databases, DW update takes place in an off-line fashion; as a consequence, more complex access structures can be adopted. In particular, besides traditional value-list indices such as B-trees, data warehousing systems usually support *bitmap index*[25], *join index*[8] and *projection index*[26]. Furthermore, at update time the indices may be reorganized in an optimal clustered form producing an extra gain. Index selection should also consider the different join algorithms, the most used being nested loop, sort merge and simple hash join.

Due to its high complexity, physical design must be carried out using heuristic algorithms [13] [16]. Such algorithms are aimed at determining the best subset of indices that minimizes the access cost for the queries in the workload under the space and time constraints. The space of the possible solutions can be reduced by considering the classical form of an OLAP query. Though OLAP query can require every possible subset of measures and can involve several dimensional attributes, its structure is almost fixed. Let us consider for example the following query based on the fact scheme presented in Figure 1:

SALE(*Date.Month*, *Product.Category*, *Store.City*; *Product.Type*="Sport", *Store.State*="I").*Qty Sold* (3)

A possible execution tree for this query is presented in Figure 5. It should be noted that:

- When possible, selection and projection operations should be executed before join operations
- It is never convenient to execute a join between two dimension tables
- The best sequence for the join operations depends on the cardinality of the filtered dimension tables

Based on these considerations, the set of attributes on which it could be useful to build an index can be restricted to:

- *Non-key attributes of the dimension tables*: the index, for example a bitmap or a B^+-Tree, will be useful to speed up the selection operation
- *Foreign keys attributes of the fact table belonging to the primary key*: these indices will speed up the join operations

As to the update queries we believe that, similarly to logical design, an upper bound should be placed on the total update time.

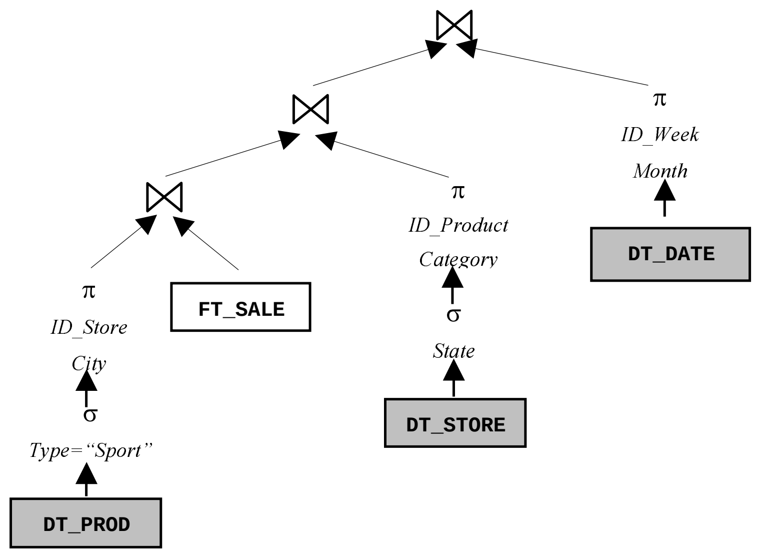


Figure 6. A possible execution tree for the OLAP query (3).

• CONCLUSION

In this paper we outlined a general methodological framework for DW design, discussing the relationships between the different steps and the difficulties in carrying them out. The framework covers the conceptual, logical and physical steps. The conceptual step is based on the Dimensional Fact Model, while the logical one exploits multiple cost functions at increasing levels of detail in order to improve both the efficiency and efficacy of the algorithms.

While conceptual design and workload definition have already been achieved and tested on a set of sample applications producing satisfying results, we are currently studying effective algorithms for logical and physical design.

REFERENCES

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

[2] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized view selection in multidimensional database", Proc. 23rd Very Large Database Conf. (VLDB97), Athens, Greece, 1997, pp. 156-165.

[3] J.W. Buzydlowski, I. Song and L. Hassel. "A Framework for Object-Oriented On-Line Analytic Processing", Proc. of the 1st Int. Workshop on Data Warehousing and OLAP (DOLAP'98), Washington D.C., November 1998.

[4] L. Cabibbo, and R. Torlone, "Querying Multidimensional databases", 6th Workshop on Database Programming Languages (DBPL'97), 1997.

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

[6] Cabibbo, L., and Torlone, R., "A logical Approach to Multidimensional Databases". Lecture Notes in Computer Science, n. 1377, Proc. of the 6th Int. Conf. on Extending Database Technology, (EDBT'98), Valencia, March 1998, pp. 183-197.

[7] A.F. Cardenas, "Analysis and performance of inverted database structures", Comm. ACM, vol. 18, n. 5, pp. 253-263, 1975.

[8] C.D. French, " 'One Size Fits All' Database Architectures Do Not Work for DSS", Proc. ACM SIGMOD Conf., 1995, pp. 449-450.

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

[10] M. Golfarelli, D. Maio, and S. Rizzi, "The Dimensional Fact Model: a Conceptual Model for Data Warehouses" to appear on Int. Jour. Computer and Information Systems.

[11] A. Gupta, V. Harinarayan, and D. Quass, "Aggregate query processing in data warehousing environments", Proc. 21st Very Large Database Conf. (VLDB95), 1995, pp. 358-369.

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

[13] H. Gupta, V. Harinarayan, and A. Rajaraman, "Index selection for OLAP", Proc. Int. Conf. Data Engineering, Binghamton, UK, 1997.

[14] M. Gyssens, and L.V.S. Lakshmanan, "A foundation for multi-dimensional databases", Proc. 23rd Very Large Database Conf. (VLDB97), Athens, Greece, 1997, pp. 106-115.

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

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

[17] S. Kelly, "Data Warehousing in Action", New York: John Wiley & Sons, 1997.

[18] R. Kimball, "The data warehouse toolkit", New York: John Wiley & Sons, 1996.

[19] R. Kimball, L. Reeves, M. Ross and W. Thornthwaite, "The data Warehouse Lifecycle Toolkit", NewYork: John Wiley & Sons, 1998.

[20] A. Koeller, E.A. Rundensteiner and N. Hachem, "Integrating the Rewriting and Ranking Phases of View Synchronization", Proc. of the 1st Int. Workshop on Data Warehousing and OLAP (DOLAP'98), Washington, D.C. November 1998.

[21] A.J. Lee, A. Nica and E.A. Rundensteiner, "Keeping Virtual Information Resources Up and Running", Proc. of IBM CASCO97, November 1997, pp. 1-14.

[22] W. Lehner, "Modeling Large Scale OLAP Scenarios", Lecture Notes in Computer Science, n. 1377, Proc. of the 6th Int. Conf. on Extending Database Technology, (EDBT'98), Valencia, March 1998, pp. 183-197.

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

[24] A. Nica, A.J. Lee and E.A. Rundensteiner, "The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems", Proc. of the 6th Int. Conf. on Extending Database Technology, (EDBT'98), Valencia, March 1998, pp. 359-373.

[25] P. O'Neil, "Model 204 Architecture and Performance", Lectures Notes in Computer Science n. 359, Springer-Verlag, 1987.

[26] P. O'Neil and G. Graefe, "Multi-Table Joins Through Bitmapped Join Indices", SIGMOD Record, pp. 8-11, September 1995.

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

[28] D. Quass, "Maintenance Expressions for Views with Aggregation", Proc. ACM Workshop on Materialized Views: Techniques and Applications, June 1996.

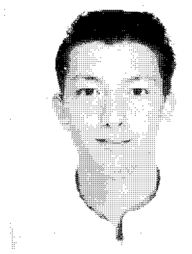
[29] L. Silverston, W.H. Inmon and K. Graziano. "The Data Model Resource Book", New York: John Wiley & Sons, 1997.

[30] J. Trujillo, M. Palomar. "An Object Oriented Approach to Multidimensional Database Conceptual Modeling (OOMD)", Proc. of the 1st Int. Workshop on Data Warehousing and OLAP (DOLAP'98), Washington, D.C. November 1998.

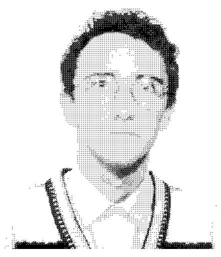
[31] J. Widom, "Research Problems in Data Warehousing", Proc. 4th Int. Conf. on Information and Knowledge Management, 1995.

[32] G. Wiederhold, "Integrating artificial intelligence and database technologies", Journal of Intelligent Information Systems, Special issue: Intelligent Integration of Information, vol. 6, 1996.

[33] S.B. Yao, "Approximating block accesses in database organizations", Comm. ACM, vol. 20, n. 4, pp. 260-61, 1977.



Matteo Golfarelli received his degree in Computer Science from the University of Bologna, Italy, in 1995. Since November 1995 he has been a PhD student at the Faculty of Computer Science of the University of Bologna, with research theme "Autonomous Agent". His current research interests also include data warehousing and biometric systems.



Stefano Rizzi received his degree in Electronic Engineering from the University of Bologna, Italy, in 1988; he received the Ph.D. degree for his work on autonomous agents in 1996. Since 1998 he is Associate Professor at the Computer Science Department of the University of Bologna, Cesena, Italy. His current research interests include data warehousing, mobile robotics, pattern recognition and visual query systems.