THE DIMENSIONAL FACT MODEL: A CONCEPTUAL MODEL FOR DATA WAREHOUSES[1]

MATTEO GOLFARELLI, DARIO MAIO and STEFANO RIZZI
DEIS - Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
{*mgolfarelli,dmaio,srizzi*}*@deis.unibo.it*

Data warehousing systems enable enterprise managers to acquire and integrate information from heterogeneous sources and to query very large databases efficiently. Building a data warehouse requires adopting design and implementation techniques completely different from those underlying operational information systems. Though most scientific literature on the design of data warehouses concerns their logical and physical models, an accurate conceptual design is the necessary foundation for building a DW which is well-documented and fully satisfies requirements. In this paper we formalize a graphical conceptual model for data warehouses, called Dimensional Fact model, and propose a semi-automated methodology to build it from the pre-existing (conceptual or logical) schemes describing the enterprise relational database. The representation of reality built using our conceptual model consists of a set of fact schemes whose basic elements are facts, measures, attributes, dimensions and hierarchies; other features which may be represented on fact schemes are the additivity of fact attributes along dimensions, the optionality of dimension attributes and the existence of non-dimension attributes. Compatible fact schemes may be overlapped in order to relate and compare data for drill-across queries. Fact schemes should be integrated with information of the conjectured workload, to be used as the input of logical and physical design phases; to this end, we propose a simple language to denote data warehouse queries in terms of sets of fact instances.

Keywords: Data warehouse, Conceptual models, Multidimensional data model, Entity-Relationship model

1. Introduction

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

The topic of data warehousing encompasses application tools, architectures, information service and communication infrastructures to synthesize information useful for decision-making from distributed heterogeneous operational data sources. This

[1] This work was partially supported by the INTERDATA project from the Italian Ministry of University and Scientific Research and by Olivetti Sanità.

information is brought together into a single repository, called a *data warehouse* (DW), suitable for direct querying and analysis and as a source for building logical *data marts* oriented to specific areas of the enterprise.[17]

While it is universally recognized that a DW leans on a multidimensional model, little is said about how to carry out its conceptual design starting from the user requirements. On the other hand, we argue that an accurate conceptual design is the necessary foundation for building an information system which is both well-documented and fully satisfies requirements. The Entity/Relationship (E/R) model is widespread in the enterprises as a conceptual formalism to provide standard documentation for relational information systems, and a great deal of effort has been made to use E/R schemes as the input for designing non-relational databases as well[8]; unfortunately, as argued in Ref. 17:

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

In this paper we present a graphical conceptual model for DWs, called *Dimensional Fact Model* (DFM). 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. Compatible fact schemes may be overlapped in order to relate and compare data. Fact schemes may be integrated with information of the conjectured workload, expressed in terms of fact instance expressions denoting queries, to be used as the input of a design phase whose output are the logical and physical schemes of the DW. To this end, we propose a simple language to denote data warehouse queries in terms of sets of fact instances.

Most information systems implemented in enterprises during the last decade are relational, and in most cases their analysis documentation consists of E/R schemes. In this paper we propose a semi-automated methodology to carry out conceptual modelling starting from the pre-existing E/R schemes describing the operational information system. In some cases, the E/R documentation held by the enterprise is incomplete or incorrect; often, the only documentation available consists of logical relational schemes. Thus, we show how our methodology can be applied starting from the database logical scheme.

After surveying the literature on DWs in Section 2, in Section 3 we describe the DFM and introduce fact instance expressions as a formalism to denote DW queries. In Section 4, the overlapping of related fact schemes is discussed. Section 5 describes a methodology for deriving fact schemes from the schemes describing the operational database.

2. Background and literature on data warehousing

From a functional point of view, the data warehouse process consists of three phases: extracting data from distributed operational sources; organizing and integrating data consistently into the DW; accessing the integrated data in an efficient and flexible fashion. The first phase encompasses typical issues concerning distributed heterogeneous

information services, such as inconsistent data, incompatible data structures, data granularity, etc. (for instance, see Ref. 23). The third phase requires capabilities of aggregate navigation[12], optimization of complex queries[6], advanced indexing techniques[18] and friendly visual interface to be used for On-Line Analytical Processing (OLAP) [7,5] and data mining.[9]

As to the second phase, designing the 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 materialization of views[2,15] and index selection[13,16], no significant effort has been made so far to develop a complete and consistent design methodology. The apparent lack of interest in the issues related to conceptual design can be explained as follows: (a) data warehousing was initially devised within the industrial world, as a result of practical demands of users who typically do not give predominant importance to conceptual issues; (b) logical and physical design have a primary role in optimizing the system performances, which is the main goal in data warehousing applications.

In Ref. 19, the author proposes an approach to the design of DWs based on a business model of the enterprise which is actually a relational database scheme. Regretfully, conceptual and logical design are mixed up; since logical design is necessarily targeted towards a logical model (relational in this case), no unifying conceptual model of data is devised. Ref. 1 and Ref. 14 propose two data models for multidimensional databases and the related algebras. Both models are at the logical level, thus, they do not address conceptual modelling issues such as the structure of attribute hierarchies and non-additivity constraints. The approach to conceptual DW modeling presented in Ref. 4 shares several ideas with our early work on the topic[10], though it is mainly addressed towards representing attribute hierarchies and neglects other conceptual issues such as additivity and scheme overlapping.

The multidimensional model may be mapped on the logical level differently depending on the underlying DBMS. If a DBMS directly supporting the multidimensional model is used, fact attributes are typically represented as the cells of multidimensional arrays whose indices are determined by key attributes.[15] On the other hand, in relational DBMSs the multidimensional model of the DW is mapped in most cases through star schemes[17] consisting of a set of *dimension tables* and a central *fact table*. Dimension tables are strongly denormalized and are used to select the facts of interest based on the user queries. The fact table stores fact attributes; its key is defined by importing the keys of the dimension tables.

Different versions of these base schemes have been proposed in order to improve the overall performances[3], handle the sparsity of data[20] and optimize the access to aggregated data.[16] In particular, the efficiency issues raised by data warehousing have been dealt with by means of new indexing techniques (see Ref. 22 for a survey), among which we mention bitmap indices.[20]

3. The Dimensional Fact Model

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 $path_{0j}(g) \subseteq g$ a directed path starting in v_0 and ending in v_j; given $v_i \in path_{0j}(g)$, we will denote with $path_{ij}(g) \subseteq g$ a directed path starting in v_i and ending in v_j. We will denote with $sub(g,v_i) \subset g$ the quasi-tree rooted in $v_i \neq v_0$.

Within a quasi-tree, two or more directed path may converge on the same vertex. A quasi-tree in which the root is connected to each other vertex through exactly one path degenerates into a directed tree.

A *dimensional scheme* consists of a set of *fact schemes*. The components of fact schemes are facts, measures, dimensions and hierarchies. In the following an intuitive description of these concepts is given; a formal definition of fact schemes can be found in Definition 2.

A *fact* is a focus of interest for the decision-making process; typically, it models an event occurring in the enterprise world (e.g., sales and shipments). *Measures* are continuously valued (typically numerical) attributes which describe the fact from different points of view; for instance, each sale is measured by its revenue. *Dimensions* are discrete attributes which determine the minimum granularity adopted to represent facts; typical dimensions for the sale fact are product, store and date. *Hierarchies* are made up of discrete *dimension attributes* linked by -to-one relationships, and determine how facts 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 other dimension attributes define progressively coarser granularities. A hierarchy on the product dimension will probably include the dimension attributes product type, category, department, department manager. Hierarchies may also include *non-dimension attributes*. A non-dimension attribute contains additional information about a dimension attribute of the hierarchy, and is connected by a -to-one relationship (e.g., the department address); unlike dimension attributes, it cannot be used for aggregation.

Some multidimensional models in the literature focus on treating dimensions and measures symmetrically.[1,14] This promises to be an important achievement from both the point of view of the uniformity of the logical model and that of the flexibility of OLAP operators. Nevertheless we claim that, at a conceptual level, distinguishing between measures and dimensions is important since it allows the logical design to be more specifically aimed at the efficiency required by data warehousing applications.

Definition 2. A fact scheme is a sextuple

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

where:

- M is a set of *measures*. Each measure $m_i \in M$ is defined by a numeric or Boolean expression which involves values acquired from the operational 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 a *dimension*. When we need to emphasize that an attribute a_i is a dimension, we will denote it as d_i. The *hierarchy* on dimension $d_i \in Dim(f)$ is the quasi-tree rooted in d_i, $sub(qt(f),d_i)$.

- $O \subset R$ is a set of *optional* relationships. The domain of each dimension attribute a_j such that $\exists(a_i,a_j) \in O$ includes a 'null' value.
- 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 $\Omega \in$ {'SUM','AVG','COUNT','MIN','MAX','AND','OR',...} (*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 the grouping operator 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 graphic representation of the concepts introduced above with reference to the fact scheme *SALE*, shown in Figure 1, which describes the sales in a chain store. This scheme, as well as the *INVENTORY* and the *SHIPMENT* schemes proposed in Section 4, are based on the star schemes reported in Ref. 17.

In the DFM, 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 measures. In the sale scheme, *quantity sold*, *revenue* and *no. of customers* are measures.

Dimension attributes are represented by circles. Each dimension attribute directly attached to the fact is a dimension. The dimension pattern of the sale scheme is {*date*, *product*, *store*, *promotion*}. Non-dimension attributes are always terminal within the quasi-tree, and are represented by lines (for instance, *address*).

Subtrees rooted in dimensions are *hierarchies*. 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`); by convention, $\alpha_i.a_i = \alpha_i$.

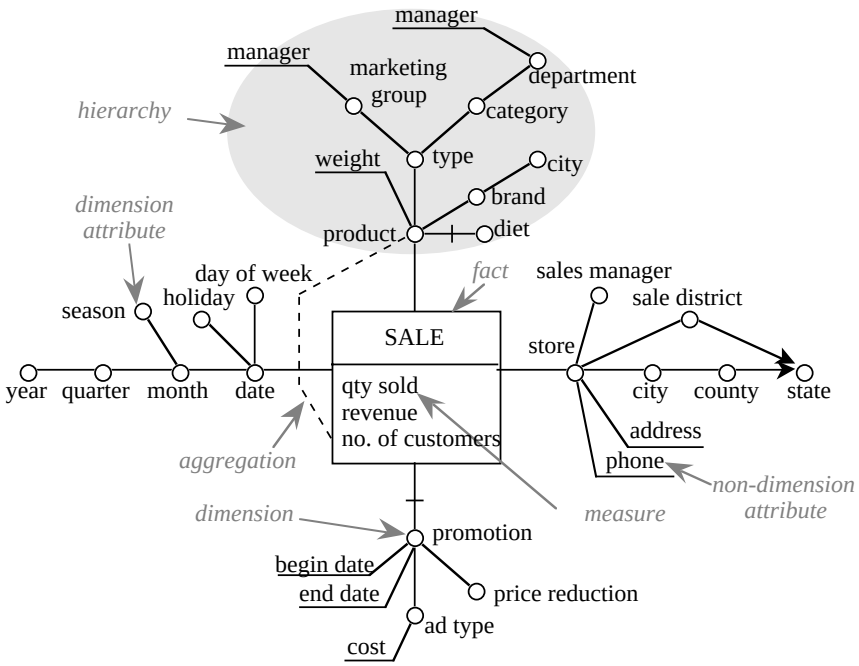


Fig. 1. The *SALE* fact scheme. Arrows are placed by convention only on the attributes where two or more paths converge.

The fact scheme may not be a tree: in fact, two or more distinct paths may connect two given dimension attributes within a hierarchy, provided that every directed path still 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., *store* 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. On the other hand, consider attribute *city* on the *product* dimension, which represents the city where a brand is manufactured. In this case the two *city* attributes have different semantics and must be represented separately; in fact, a product manufactured in a city can be sold in stores of other cities.

Optional relationships between pairs of attributes are represented by marking with a dash the corresponding arc. For instance, attribute *diet* takes a value only for food products; for the other products, it will take a conventional null value.

A measure is additive on a dimension if its values can be aggregated along the corresponding hierarchy by the sum operator. Since this is the most frequent case, in order to simplify the graphic notation in the DFM, only the exceptions are represented explicitly. In particular, given measure m_j and dimension d_i:

1. If (m_j, d_i, 'SUM')$\notin$ S (m_j is not additive along d_i), m_j and d_i are connected by a dashed line labelled with all aggregation operators Ω (if any) such that (m_j, d_i, Ω)$\in$ S (for instance, see Figures 1 and 5).
2. If (m_j, d_i, 'SUM')$\in$ S (m_j is additive along d_i):
   2.1 If $\nexists\ \Omega\neq$'SUM' | (m_j, d_i, Ω)$\in$ S (only sum can be used for aggregation), m_j and d_i are not graphically connected.
   2.2 Otherwise (other operators can be used besides the sum), m_j and d_i are connected by a dashed line labelled with the symbol '+' followed by all the other operators $\Omega\neq$'SUM' such that (m_j, d_i, Ω)$\in$ S.

Additivity will be discussed in more detail in Subsection 3.3.

3.1. *Fact instances*

Given a fact scheme f, each n-tuple of values taken from the domains of the n dimensions of f 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 will 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 adopting one promotion ('no promotion' should be considered as a particular case of promotion).

Not every possible combination of values necessarily originates a primary fact instance. For instance, in the sale scheme, a missing primary fact instance denotes that a product was not on sale on a given day in a given store (*null assumption*); this is different from having a primary fact instance with *qty*=0, which denotes that the product remained unsold. Alternatively, it might be reasonable to assume that all products are always on sale, hence, that a missing primary fact instance denotes that the product remained unsold (*zero assumption*). Some issues related to these two different interpretations will be discussed in Subsection 3.3.

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; if a given dimension is not interesting for the current analysis, aggregation is carried out over all the possible values that dimension can assume. In the OLAP terminology, this operation is called *roll-up*.

Definition 3. Given a fact scheme f with n dimensions, a v-dimensional *aggregation pattern* ($0\leq v$) is a set $P=\{a_1,...a_v\}$ where:

1. $\forall i=1,...v\ (a_i\in A)$;
2. $P\neq Dim(f)$;
3. $\forall a_i\in P\ (\nexists\ a_j\in P,\ a_i\neq a_j \mid a_j\in sub(qt(f),a_i))$ (i.e., no directed path exists between each pair of attributes in P).

A dimension $d_i \in Dim(f)$ is said to be *hidden* within P if no attribute of its hierarchy $sub(qt(f),d_i)$ appears within P. An aggregation pattern P is *legal* with reference to measure $m_j \in M$ if

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

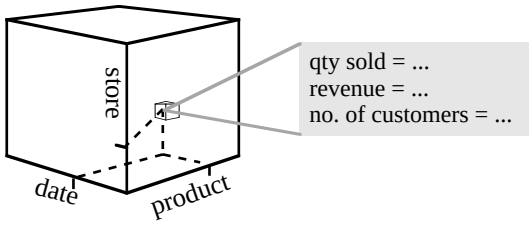
Examples of aggregation patterns in the sale scheme are {*product*,*county*,*month*, *promotion*}, {*state*,*date*} (*product* and *promotion* are hidden), {*year*,*season*} (two attributes are taken from dimension *date*), {} (all dimensions are hidden). Pattern {*brand*,*month*} is illegal with reference to *no. of customers* since the latter cannot be aggregated along the product hierarchy.

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

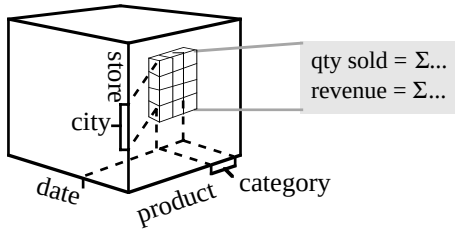
$$\{pf(\alpha_1,...\alpha_n) \mid \forall k \in \{1,...n\}\ \alpha_k \in Dom(d_k) \wedge \forall h \in \{1,...v\}\ \alpha_{h*}.a_h = \beta_h\}$$

and is characterized by exactly one value for each measure for which P is legal, calculated by applying an aggregation operator to the values that measure assumes within the primary fact instances aggregated (see Subsection 3.3).

Figure 2.a shows a primary fact instance on the sale scheme. Figure 2.b shows the primary fact instances corresponding to the secondary fact instance describing the sales of products of a given category during one day in a city; measure *no. of customers* is not reported since it cannot be aggregated along the product dimension.



(a)



(b)

Fig. 2. A primary (a) and a secondary (b) fact istance for the *SALE* scheme (dimension *promotion* is omitted for clarity).

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

3.2. *Representing queries on the dimensional scheme*

In general, querying an information system means linking different concepts through user-defined paths in order to retrieve some data of interest; in particular, for relational databases this is done by formulating a set of joins to connect relation schemes. On the other hand, a substantial amount of queries on DWs are aimed at extracting summary data to fill structured reports to be analysed for decisional or statistical purposes. Thus, 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*. The simple language we propose is aimed at defining, with reference to a dimensional scheme, the queries forming the expected workload for the DW, to be used for logical design; thus, it focuses on which data must be retrieved and at which level they must be consolidated.

A fact instance expression has 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

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

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 ($0 \le p \le v$).
- Each Boolean predicate e_j (j=1,...h, $h \ge 0$) 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

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

For instance, the expression

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

denotes the quantities of product P5 sold in each store, with each promotion, during each day 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 $\exists a_{i_j *} \in P \mid a_{i_j *} \in path_{0 i_j}(qt(f))$, *internal* otherwise (see Figure 3). 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

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

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

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

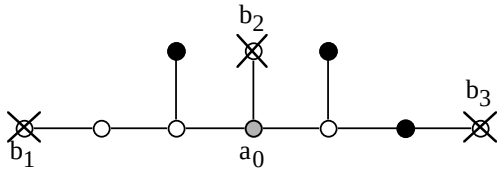


Fig. 3. Representation of a fact instance expression on qt(f): black circles represent the attributes in the aggregation pattern, crosses mark the attributes on which selection predicates are defined. The predicates on b_1 and b_2 are internal; that on b_3 is external.

Consider, for instance, the two 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*

which denote, respectively, the total sales of each type of products of category 'food' for January 1998 (Figure 4.a) and the total sales of each type of products of brand 'General' for January 1998 (Figure 4.b). The predicates on *month* and on *category* are external, whereas that on *brand* is internal. With reference to the sample set of data in Table I, and considering that *qtySold* is additive on all the dimensions, the results of the two expressions are shown in Table II.

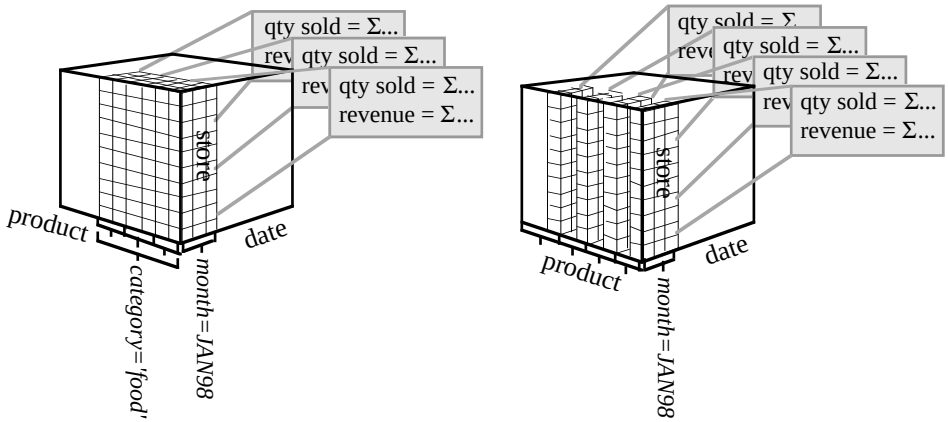


Fig. 4. Sales of the three types of products of category 'food' (a) and sales of all four types of products but including only the products of brand 'General' (b).

product	*brand*	*type*	*category*	*month*	*qtySold*
GD	General	soft drink	food	JAN98	100
BB	Best	biscuits	food	JAN98	200
GB	General	biscuits	food	JAN98	50
BS	Best	shirt	clothing	JAN98	100
GS	General	shirt	clothing	JAN98	50
BT	Best	tie	clothing	JAN98	20

Table I. A sample set of data for products.

type	*month*	*qtySold*
soft drink	JAN98	100
biscuits	JAN98	250

type	*month*	*qtySold*
soft drink	JAN98	100
biscuits	JAN98	50
shirt	JAN98	50

Table II. Results of two expressions.

A significant amount of DW queries require consolidating data on multiple levels of abstraction; this queries can be expressed in our language as the union of two or more sets of fact instances. For instance, the query requiring the sales of products of brand 'General' for each month, showing also the subtotals for each year and the total, can be expressed as follows:

$$SALE(date.month, product\ ;\ product.brand=\text{'General'}).qtySold$$
$$\cup\ SALE(date.year, product\ ;\ product.brand=\text{'General'}).qtySold$$
$$\cup\ SALE(product\ ;\ product.brand=\text{'General'}).qtySold$$

3.3. *Additivity*

Aggregation requires defining a proper operator to compose the measure values characterizing primary fact instances into measure values characterizing each secondary fact instance.

Definition 4. Given a fact scheme f, measure $m_j \in M$ is said to be *aggregable* on dimension $d_k \in Dim(f)$ if $\exists(m_j, d_k, \Omega) \in S$, *non-aggregable* otherwise. Measure m_j is said to be *additive* on d_k if $\exists(m_j, d_k, \text{'SUM'}) \in S$, *non-additive* otherwise.

As a guideline, most measures in a fact scheme should be additive. An example of additive measure in the sale scheme is *qty sold*: the quantity sold for a given sales manager is the sum of the quantities sold for all the stores managed by that sales manager.

A measure may be *non-additive* on one or more dimensions. Examples of this are all the measures expressing a level, such as an inventory level, a temperature, etc. An inventory level is non-additive on time, but it is additive on the other dimensions. A temperature measure is non-additive on all the dimensions, since adding up two temperatures hardly makes sense. However, this kind of non-additive measures can still be aggregated by using operators such as average, maximum, minimum; Figure 5 shows an example where both operators AVG and MIN can be used for aggregation; measure *qty* expresses, for each product, the number of copies present within each warehouse during each week.

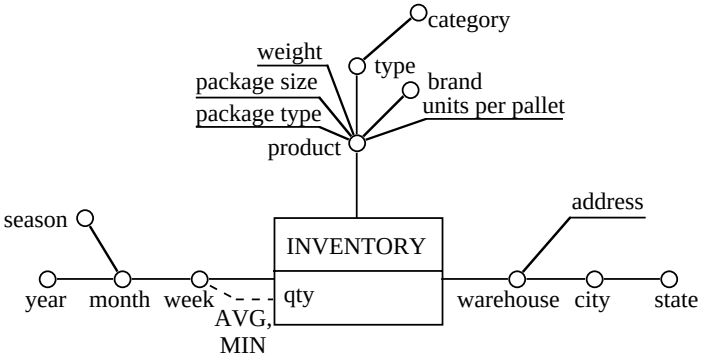


Fig. 5. The INVENTORY fact scheme.

For other measures, aggregation is inherently impossible for conceptual reasons. Consider the measure *number of customers* in the sale example, estimated for a given product, day and store by counting the number of purchase tickets for that product printed on that day in that store. Since the same ticket may include other products, adding or averaging the number of customers for two or more products would lead to an inconsistent result. Thus, *number of customers* is non-aggregable on the product dimension (while it is additive on the time and the stores dimensions). In this case, the reason for non-aggregability is that the relationship between purchase tickets and products is many-to-many instead of many-to-one: measure *number of customers* cannot be consistently

aggregated on the product dimension, whatever operator is used, unless the grain of fact instances is made finer. If m_j is non-aggregable on d_k, any aggregation pattern not including d_k is illegal with reference to m_j.

Given a measure m_j aggregable on d_k by operator Ω and the aggregation pattern $P=\{d_1,...,d_{k-1},a_k,d_{k+1},...d_n\}$, which includes all the dimensions except d_k which is represented by any other dimension attribute a_k belonging to its hierarchy, the value of m_j may be computed for each secondary fact instance at pattern P as:

$$f(d_1,...a_k,...d_n\ ;\ d_1=\alpha_1,...a_k=\alpha_k,...d_n=\alpha_n).m_j =$$

$$= \Omega_{\beta\in Dom(d_k)|\beta.a_k=\alpha_k} f(d_1,...d_k,...d_n\ ;\ d_1=\alpha_1,...d_k=\beta,...d_n=\alpha_n).m_j$$

for each $\alpha_k\in Dom(a_k)$, $\alpha_i\in Dom(d_i)$ $(i=1,...n;\ i\neq k)$. Similarly, if d_k is hidden within P, it is:

$$f(d_1,...d_{k-1},d_{k+1},...d_n\ ;\ d_1=\alpha_1,...d_n=\alpha_n).m_j =$$

$$= \Omega_{\beta\in Dom(d_k)} f(d_1,...d_k,...d_n\ ;\ d_1=\alpha_1,...d_k=\beta,...d_n=\alpha_n).m_j$$

In the following these formulae are explained with an example. Let the primary fact instances for the *INVENTORY* fact scheme be those represented in Table III. The matrix reports the values of measure *qty*; dimension *warehouse* is not considered for simplicity. A missing primary fact instance denotes that a product was not in the catalogue on a given week. The secondary fact instances at patterns {*week*, *type*} and {*week*} are shown in Table IV. Since *qty* is additive along *product*, the quantity for each product type for each week is the sum of the quantities for the products of that type for that week; the total quantity for each week is the sum of all quantities for that week. The secondary fact instances at patterns {*month*, *product*} and {*product*} are shown in Table V; they are calculated using the average function to aggregate *qty* along *week*.

	type	T1			T2	
	product	P1	P2	P3	P4	P5
month	*week*					
jan98	1-98	10	50	35	15	-
	2-98	10	60	30	15	20
	3-98	8	60	30	15	20
	4-98	8	40	25	15	30
	5-98	12	40	20	15	20
feb98	6-98	12	40	20	15	20
	7-98	9	35	20	15	10
	8-98	9	55	20	5	10
	9-98	7	55	35	5	5

Table III. Primary fact instances for a given warehouse (symbol '-' denotes a missing fact instance).

		type	T1	T2	
month	*week*				
jan98	1-98		95	15	110
	2-98		100	35	135
	3-98		98	35	133
	4-98		73	45	118
	5-98		72	35	107
feb98	6-98		72	35	107
	7-98		64	25	89
	8-98		84	15	99
	9-98		97	10	107

Table IV. Secondary fact instances at patterns {*week*,*type*} (left) and {*week*} (right).

	type	T1			T2	
	product	P1	P2	P3	P4	P5
month						
jan98		9.60	50.00	28.00	15.00	18.00
feb98		9.25	46.25	23.75	10.00	11.25
		9.44	48.33	26.11	12.78	15.00

Table V. Secondary fact instances at patterns {*month*,*product*} (top) and {*product*} (bottom).

As a matter of fact, when using for instance pattern {*week*}, secondary fact instances could be more conveniently computed by aggregating the secondary fact instances at pattern {*week*, *type*} instead of aggregating the primary fact instances. As pointed out in Ref. 11, this can be done efficiently only for distributive and algebraic functions: SUM, MIN, MAX, COUNT, AND, OR fall within the first category, AVG within the second. These optimization issues, which in Ref. 21 are discussed also for complex aggregation queries, fall outside the scope of this paper.

When aggregating instances along two or more dimensions at the same time, it is necessary to declare in which order dimensions are to be considered. Let Ω' and Ω" be the operators used to aggregate m_j along d_1 and d_2 respectively, and $P=\{a_1,a_2\}$ be the aggregation pattern to be computed, where a_1 and a_2 belong to the hierarchies defined on d_1 and d_2, respectively. In order to compute the values of m_j at P, two different aggregation sequences can be adopted:

$$\{d_1,d_2\} \overset{\Omega'}{\rightarrow} \{a_1,d_2\} \overset{\Omega''}{\rightarrow} \{a_1,a_2\}$$
$$\{d_1,d_2\} \overset{\Omega''}{\rightarrow} \{d_1,a_2\} \overset{\Omega'}{\rightarrow} \{a_1,a_2\}$$

In general, the outcome depends on which sequence is adopted unless one of the following situations occurs:

- Ω' = Ω" ∈ {'SUM','MIN','MAX','AND','OR'};
- Ω'∈ {'SUM','AVG'} and Ω" = 'AVG' (or vice versa) and the zero assumption is made (missing fact instances denote products out of stock).

The restrictions applied when the average operator is involved arise since, when the null assumption is made, the subsets on which average operates may not have the same cardinality.

Table VI shows, with reference to the inventory example, the secondary fact instances at patterns {*month, type*}, {*month*}, {*type*} and {} when the zero assumption is made. It is easy to verify that, if the null assumption were made instead, or if function MIN were used to aggregate *qty* along *week*, applying the two aggregation sequences

$$\{week, product\} \overset{SUM}{\rightarrow} \{week, type\} \overset{MIN}{\rightarrow} \{month, type\} \text{ or}$$
$$\{week, product\} \overset{MIN}{\rightarrow} \{month, product\} \overset{SUM}{\rightarrow} \{month, type\}$$

would lead to different results.

	type T1	T2	
month			
jan98	87.60	33.00	120.60
feb98	79.25	21.25	100.50
	83.89	27.78	111.67

Table VI. Secondary fact instances at patterns {*month*,*type*} (top left), {*month*} (top right), {*type*} (bottom left), {} (bottom right).

In order to give non ambiguous semantics to aggregation we suggest that, for each fact scheme, a preferred aggregation sequence is declared by specifying an ordering for dimensions. In the inventory scheme, we believe that the most suitable ordering is (*product*, *warehouse*, *week*) (or, indifferently, (*warehouse*, *product*, *week*)).

It should be noted that the COUNT operator behaves differently from the others. Firstly, it counts the number of primary fact instances within each secondary fact instance, hence, it does not operate on any measure. Furthermore, it is not obvious how counting on a given dimension can be combined with other operators working on the other dimensions. For this reason, we recommend using COUNT on all the dimensions contemporarily.

3.4. *Empty facts*

A fact scheme is said to be *empty* if it has no measures (M=∅). In this case, primary fact instances only record the occurrence of events. Consider for instance, within the university domain, the fact scheme shown in Figure 6. In this case, each fact instance states that a given student attended a given course during a given year; no measure is used to further describe this fact.

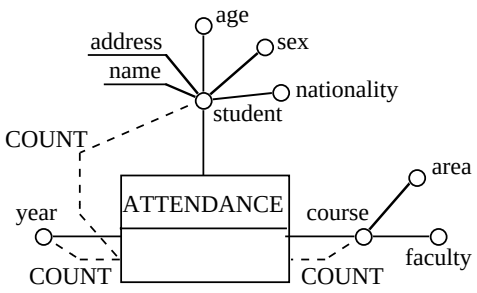


Fig. 6. The ATTENDANCE fact scheme.

In an empty fact scheme, two approaches to the problem of aggregation can be pursued. In the first approach, which requires using either the AND or the OR operators, the information carried by each secondary fact instance is related to the existence of the corresponding primary fact instances. In order to explain this concept, we may suppose that the fact is described by an implicit Boolean measure, which is true if the event occurred and false otherwise: in this case, both operators AND and OR can be used for aggregation, with universal and existential semantics, respectively. For instance:

ATTENDANCE(*course.area*, *student* ;
year='1998', *course.area*='Databases', *course.faculty*='Computer Science')

may denote either the students who during 98 attended all the database courses in the Computer Science Faculty (AND operator), or the students who during 98 attended at least one database course in the Computer Science Faculty (OR operator).

In the second approach, which requires using the COUNT operator, the information carried by each secondary fact instance is the number of corresponding primary fact instances. Equivalently, one may suppose that the fact is described by an implicit integer measure, which has value 1 if the event occurred and 0 otherwise, and aggregate fact instances by the SUM operator. For instance:

ATTENDANCE(*course*, *student.sex* ; *year*='1998', *course.faculty*='Computer Science')

denotes, for each course in the Computer Science Faculty, the number of students of each sex who attended the course.

Empty fact schemes correspond, on the logical level, to *factless fact tables*, typically used for event tracking or as coverage tables.[17]

4. 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; in the OLAP terminology, these are called *drill-across* queries. In this section we define the rules for combining two related fact schemes 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 5. 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: $A' \cap A" \neq \emptyset$. Attribute a_i is considered to be common to f' and f" if, within f' and f", it has the same semantics and if $Dom_{f'}(a_i) \cap Dom_{f"}(a_i) \neq \emptyset$.

Definition 6. 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 $cnt(t,I)=(I \cup \{a_0\},E^*)$ 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.

A quasi-tree can be contracted on a given set of vertices by applying an appropriate sequence of arc contractions, i.e., a sequence in which each step replaces two consecutive vertices a_i and a_j by a single vertex a_i adjacent to those vertices that were previously adjacent to a_i or a_j. Figure 7 shows a quasi-tree and its contraction on a subset of the vertices.

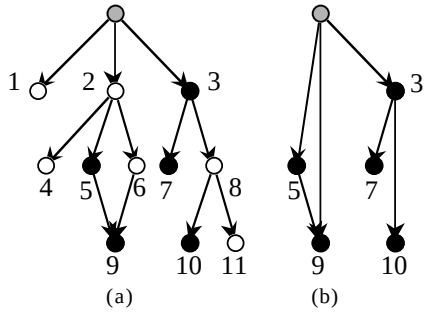


Fig. 7. A quasi-tree (a) and its contraction on the black vertices (b). The grey vertex is the root.

Definition 7. 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 [2].

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 intuitively described as follows:

---

[2] Actually, the semantics of the root and of the arcs exiting the root may be different in the two quasi-trees, since the corresponding facts may express different concepts. Nevertheless, since in this definition and in the following ones we are interested in facts only from a topological point of view (their connections with the attributes), for notational simplicity we will denote with the same dummy symbol, a_0, the roots of both quasi-trees.

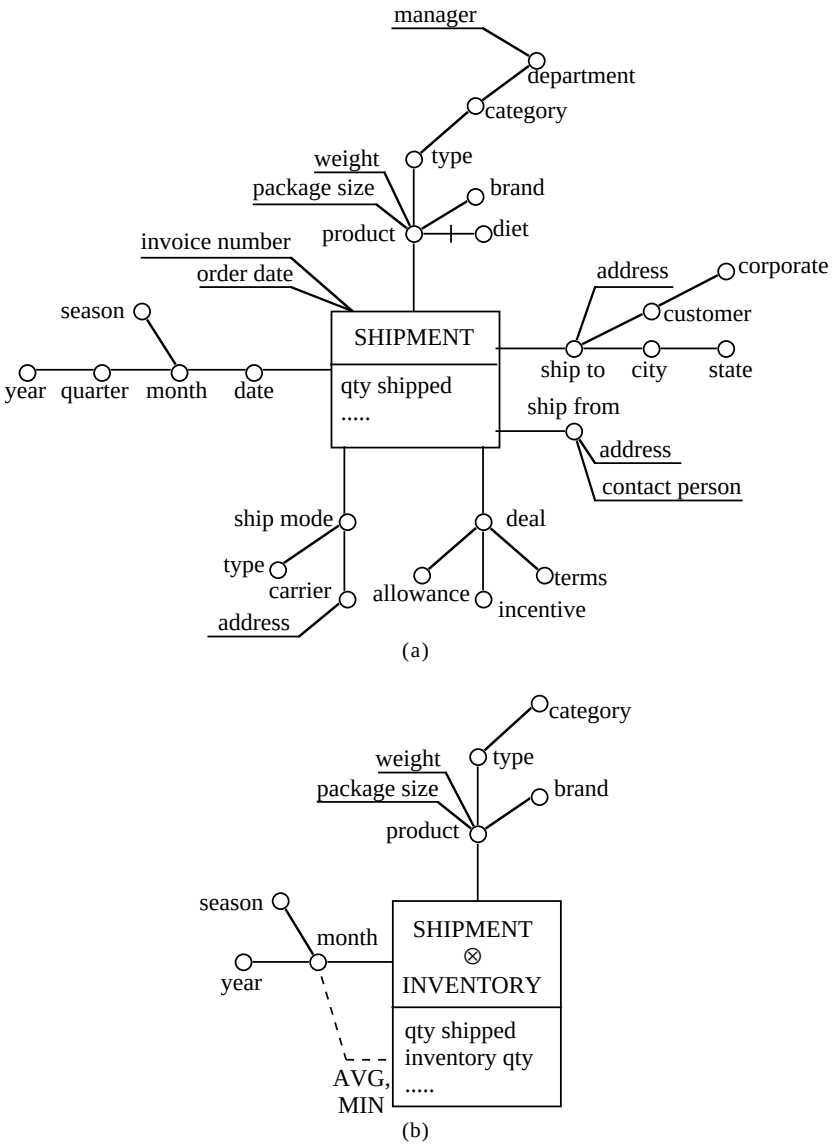


Fig. 8. The *SHIPMENT* scheme (a) and its overlap with *INVENTORY* (b).

- The measures in f are the union of those in f' and f''. Thus, the fact on which f is centred may be considered as a sort of "macro-fact" embracing both f' and f''.
- Each hierarchy in f includes all and only the attributes included in the corresponding hierarchies of both f' and f''. The functional dependencies expressed by the inter-attribute links in f' and f'' are preserved.
- The domain of each dimension attribute in f is the intersection of the domains of the corresponding attributes in f' and f''.

- An inter-attribute link in f is optional if at least one of the links in the corresponding paths in f' or f'' is optional.
- Aggregation statements of f' and f'' are preserved in f.

Formally:

Definition 8. 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:

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

Figure 8 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.

As a matter of fact, overlapping may be extended by considering more accurately the information expressed by the hierarchies in the two source schemes. Consider for instance the *INVENTORY* and *SHIPMENT* schemes, which include two compatible hierarchies on dimensions *week* and *date*, respectively. Based on Definition 8, their overlap should include only attributes *month*, *year* and *season*. Attribute *date* cannot definitely be included, since in the *INVENTORY* scheme it is impossible to disaggregate the primary fact instances at the date level. On the other hand, *quarter* could be included: in fact, the months represented in the overlap are those represented in both the source schemes, and for each month the quarter is known from *SHIPMENT*.

Even two non-strictly compatible schemes can be overlapped; since in this case the two contracted quasi-trees are different, there must be one or more conflicts in the inter-attribute dependencies in the two schemes. The resulting scheme is defined as in the case of strict compatibility, except that each conflict is solved by representing an inter-attribute dependency which subsumes both conflicting dependencies. Consider the example in Figure 9, where two non-strictly compatible fact schemes (a) and (b) are shown. The dependencies expressed by the two quasi-trees are as follows:

(a)	(b)
root → 1,2,3	root → 1,2
2 → 4	2 → 5
4 → 5	5 → 4
	1 → 3

The common elemental dependencies (namely, root → 1,2) are directly represented within the resulting scheme (c). The conflicts are solved by considering the transitive closure of the two sets of dependencies; thus, for instance, vertex 5 is positioned in (c) as a child of 2 since, in both (a) and (b), the dependency 2 → 5 holds.

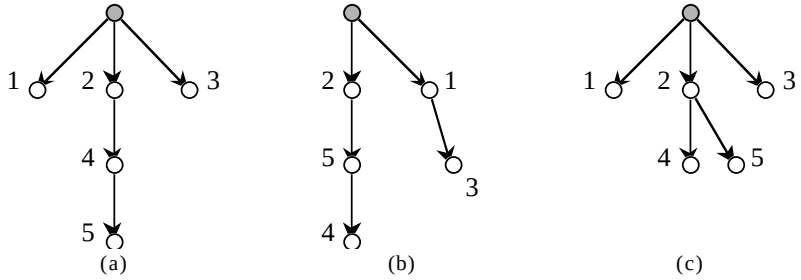


Fig. 9. Overlapping (c) of two non-strictly compatible fact schemes (a) (b).

Definition 9. Given two non-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

$M = M' \cup M''$

$A = A' \cap A''$

$\forall a_i \in A\ (Dom_{f' \otimes f''}(a_i) = Dom_{f'}(a_i) \cap Dom_{f''}(a_i))$

$N = N' \cap N''$

$R = \{(a_i,a_j) \mid \exists p_{ij}(cnt(qt(f'),A)) \wedge \exists p_{ij}(cnt(qt(f''),A)) \wedge \forall a_w \neq a_i \mid (\exists p_{wj}(cnt(qt(f'),A)) \wedge \exists p_{wj}(cnt(qt(f''),A))) (p_{ij}(cnt(qt(f'),A)) \subset p_{wj}(cnt(qt(f'),A)) \wedge p_{ij}(cnt(qt(f''),A)) \subset p_{wj}(cnt(qt(f''),A)))\}$

$O = \{(a_i,a_j) \in R \mid (\exists (a_w,a_z) \in O' \mid (a_w,a_z) \in path_{ij}(qt(f'))) \vee (\exists (a_w,a_z) \in O'' \mid (a_w,a_z) \in path_{ij}(qt(f'')))\}$

$S = \{(m_j,d_i,<op>) \mid d_i \in Dim(f' \otimes f'') \wedge (\exists (m_j,d_k,<op>) \in S' \wedge d_i \in sub(qt(f'),d_k)) \vee (\exists (m_j,d_k,<op>) \in S'' \wedge d_i \in sub(qt(f''),d_k))\}$

Queries formulated on the overlap of two schemes are actually formulated on one or both the source schemes, depending on which measures are involved in the query. In general, let q=f(P,<sel>) be a query formulated on the overlapped fact scheme $f=f_1 \otimes ... \otimes f_m$. From the conceptual point of view, q is equivalent to 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 query formulated on an overlap is:

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

5. Conceptual design from relational schemes

The methodology we outline in this section to build a DF model starting from the documentation describing the operational relational database consists of the following steps:

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

This methodology can be applied, with minor differences, starting from both E/R and logical schemes. In the following subsections we will describe the steps referring to the sale example, considering as two alternative sources its conceptual and its logical documentation. A simplified E/R scheme for sales (the part involving promotions is omitted) is shown in Figure 10. Each instance of relationship `SALE` represents an item referring to a single product within a purchase ticket. Attribute `unitPrice` is placed on `SALE` instead of `PRODUCT` since the price of the products may vary over time. The corresponding logical scheme is shown below (primary keys are underlined; for each foreign key, the referenced scheme is reported). For simplicity, no artificial codes are introduced to identify relation schemes.

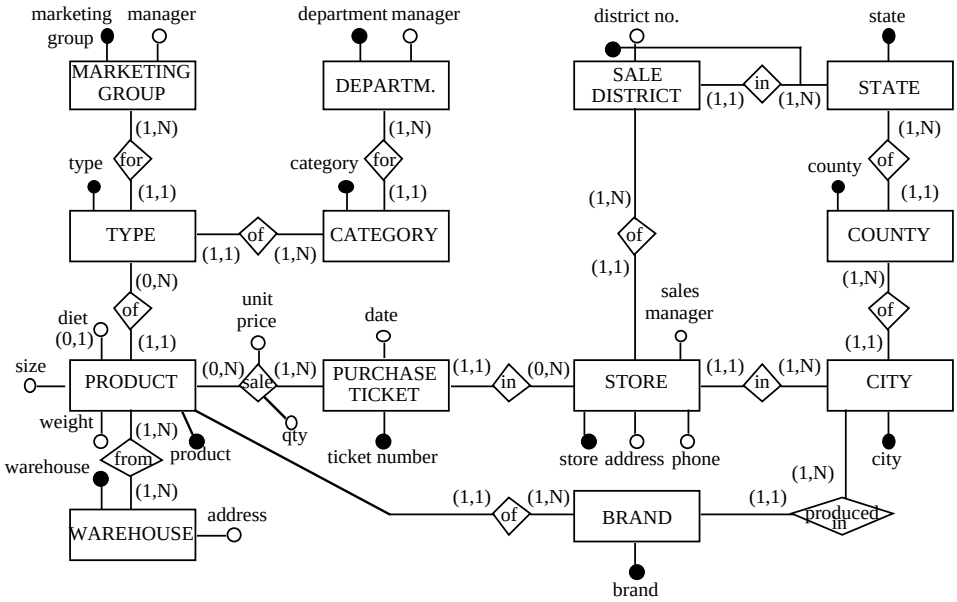


Fig. 10. The (simplified) E/R scheme for the sale fact scheme.

```
STORES(store,address,phone,salesManager,city:CITIES,
       saleDistr:DISTRICTS)
CITIES(city,county:COUNTIES)
COUNTIES(county,state:STATES)
STATES(state)
DISTRICTS(distrNo,state:STATES)
PRODUCTS(product,weight,size,diet,brand:BRANDS,type:TYPES)
BRANDS(brand,city:CITIES)
TYPES(type,markGroup:GROUPS,category:CATEGORIES)
GROUPS(markGroup,manager)
CATEGORIES(category,dept:DEPTS)
DEPTS(dept,manager)
TICKETS(tickNo,date,store:STORES)
SALES(product:PRODUCTS,tickNo:TICKETS,qty,unitPrice)
WAREHOUSES(warehouse,address)
PROD_IN_WH(product:PRODUCTS,warehouse:WAREHOUSES)
```

5.1. *Defining facts*

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

On the E/R scheme: A fact may be represented either by an entity F or by an n-ary relationship R between entities $E_1,...E_n$. In the latter case, for the sake of simplicity, it is worth transforming R into an entity F by replacing each branch E_i with a binary relationship R_i between F and E_i; if we denote with min(E,R) and max(E,R) [3], respectively, the minimum and maximum cardinalities with which entity E participates in relationship R, it is:

$$\min(F,R_i) = 1,\ \max(F,R_i) = 1,$$
$$\min(E_i,R_i) = \min(E_i,R),\ \max(E_i,R_i) = \max(E_i,R),\ i=1,...n$$

The attributes of the relationship become attributes of F; the identifier of F is the combination of the identifiers of E_i, i=1,...n.

On the logical scheme: A fact corresponds to a relation scheme F.

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

[3] Typically, $\min(E,R) \in \{0,1\}$ and $\max(E,R) \in \{1,N\}$.

Each fact identified on the source scheme becomes the root of a different fact scheme. In the following subsections, we will focus the discussion on a single fact, the one corresponding to entity (relation scheme) F. In the sale example, the fact of primary interest for business analysis is the sale of a product, represented in the E/R and in the logical schemes, respectively, by relationship sale and by relation scheme SALES. Figure 11 shows how relationship sale is transformed into an entity.

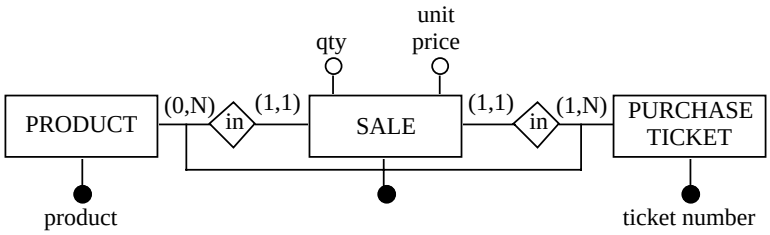


Fig. 11. Transformation of relationship sale into an entity.

5.2. *Building the attribute tree*

Given a portion of interest of a source scheme and an entity (relation scheme) F belonging to it, we call *attribute tree* the quasi-tree such that:

- each vertex corresponds to an attribute - simple or compound - of the scheme;
- the root corresponds to the identifier (primary key) of F;
- for each vertex v, the corresponding attribute functionally determines all the attributes corresponding to the descendants of v.

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

On the E/R scheme:

Let identifier(E) denote the set of attributes which make up the identifier of entity E. The attribute tree for F may be constructed automatically by applying the following recursive procedure:

```
root=newVertex(identifier(F));
// newVertex(<attributeSet>) returns a new vertex labelled
// with the concatenation of the names of the attributes in
// the set translate(F,root);
```

where

```
translate(E,v):
// E is the current entity, v is the current vertex
{ for each attribute a∈E | a≠identifier(E) do
```

```
    addChild(v,newVertex({a})); // adds child a to vertex v
  for each entity G connected to E
      by a relationship R | max(E,R)=1 do
  {  for each attribute b∈R do
      addChild(v,newVertex({b}));
    next=newVertex(identifier(G));
    addChild(v,next);
    translate(G,next);
  }
}
```

In the following we illustrate how procedure `translate` works by showing in a step-by-step fashion how a branch of the attribute tree for the sale example is generated; the resulting attribute tree is shown in Figure 12.

```
root=newVertex(ticketNumber+product)     // renamed sale

translate(E=SALE,v=sale):
  addchild(v,qty); addchild(v,unitPrice);
  for G=PURCHASE TICKET:
    addchild(v,ticketNumber);
    translate(PURCHASE TICKET,ticketNumber);
  for G=PRODUCT:
    addchild(v,product); translate(PRODUCT,product);

translate(E=PURCHASE TICKET,v=ticketNumber):
  addchild(v,date);
  for G=STORE:
    addchild(v,store); translate(STORE,store);

translate(E=STORE,v=store):
  addchild(v,address); addchild(v,phone);
  addchild(v,salesManager);
  for G=SALE DISTRICT:
    addchild(v,districtNo+state);
    translate(SALE DISTRICT,districtNo+state);
  for G=CITY:
    addchild(v,city); translate(CITY,city);

translate(E=SALE DISTRICT,v=districtNo+state):
  addchild(v,districtNo);
```

```
  for G=STATE:
    addchild(v,state); translate(STATE,state);

translate(E=STATE,v=state):
```

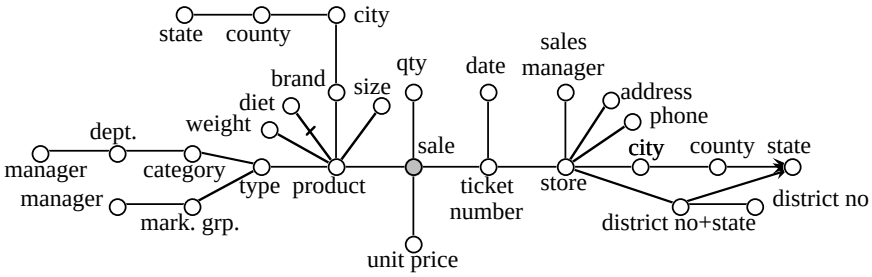


Fig. 12. Attribute tree for the sale example (the root is in grey).

It is worth adding some further notes:

- As the attribute tree undergoes the next step in the methodology, the granularity of fact instances may change and become coarser than that expressed by the identifier of F. Thus, in order to avoid confusion, we prefer to label the root of the attribute tree with the name of entity F rather than with its identifier.
- The source scheme may contain a cycle of -to-one relationships; the simplest example of this is given by a scheme representing the fact that a part is a component of another part. In this case, procedure `translate` would loop on this cycle generating an infinite branch. Since representing a recursive association at the logical level is impossible, the loop should be detected and the branch should be cut after a number of cycles depending on the relevance of the association within the application domain.
- As procedure `translate` "explores" a cyclic source scheme, the same entity E may be reached twice through different paths, thus generating two homologous vertices v' and v" in the quasi-tree. If each instance of F determines exactly one instance of E whichever of the two paths is followed (i.e., if the cycle is redundant), v' and v" may be merged into a vertex v entered by two arcs; the same applies to each couple of homologous vertices descending from v' and v". Otherwise, v' and v" must be left distinct.
- The existence of optional relationships between attributes in a hierarchy should be emphasized on the fact scheme by marking the arcs corresponding to optional relationships (min(E,R)=0) or optional attributes of the E/R scheme with a dash.
- A one-to-one relationship belonging to a cycle within the E/R scheme can be crossed in both directions. Thus, it may happen that two paths including opposite arcs are inserted into the attribute tree. In this case, the less significant path should be dropped.

- Generalization hierarchies in the E/R scheme are equivalent to one-to-one relationships between the super-entity and each sub-entity, and should be treated as such by the algorithm.
- -to-many relationships (max(E,R)>1) and multiple attributes of the source scheme cannot be inserted into the attribute tree since representing them at the logical level, for instance by a star scheme, would be impossible without violating the first normal form.
- As already stated in Section 5.1, an n-ary relationship is equivalent to n binary relationships. Most n-ary relationships have maximum multiplicity greater than 1 on all their branches; in this case, they determine n one-to-many binary relationships which cannot be inserted into the attribute tree. On the other hand, a branch with maximum multiplicity equal to 1 determines a one-to-one binary relationship which can be inserted.
- A compound attribute c of the E/R scheme, consisting of the simple attributes $a_1,...a_m$, is inserted in the attribute tree as a vertex c with children $a_1,...a_m$. It is then possible either to graft c or to prune its children (see Section 5.3).

On the logical scheme:

Let pk(R) and fk(R,S) denote the sets of the attributes of R forming, respectively, the primary key of R and a foreign key referencing S. The attribute tree for F may be constructed automatically by applying the following recursive procedure:

```
root=newVertex(pk(F));
// newVertex(<attributeSet>) returns a new vertex labelled
// with the concatenation of the names of the attributes in
// the set translate(F,root);
```

where

```
translate(R,v):
// R is the current relational scheme,
// v is the current vertex
{ for each attribute a∈R | (a≠pk(R) ∧ (∄S | a∈fk(R,S)))
    addChild(v,newVertex({a})); // adds child a to vertex v
  for each attribute set A⊂R | (∃S | A=fk(R,S))
  { next=newVertex(A);
    addChild(v,next);
    translate(S,next);
  }
  for each relational scheme T | pk(T)=fk(T,R)
  { for each attribute b∈T
        | (b∉pk(R) ∧ (∄S | b∈fk(T,S)))
      addChild(v,newVertex({b}));
```

```
    for each attribute set B⊂T | (∃S≠R | B=fk(T,S))
    { next=newVertex(B);
      addChild(v,next);
      translate(S,next);
    }
  }
}
```

Procedure `translate` builds the tree by following the functional dependencies represented within the database scheme. The first cycle considers the dependencies between the primary key of R and each other attribute of R (including, if the key is compound, the single attributes which make it up but excluding those belonging to foreign keys, which are considered at the next step). The second cycle deals with the dependencies between the primary key and each foreign key referencing a relational scheme S, by triggering the recursion on S. The third cycle considers the situation:

$R(\underline{k_R},...)$
$T(\underline{k_T}:R,...k_S:S)$
$S(\underline{k_S},...)$

in which the relationship one-to-many between R and S has been represented through a third relation scheme T.

The same considerations made for the E/R case hold when the attribute tree is built from the logical scheme. The attribute tree obtained for the sale example is the same shown in Figure 12.

5.3. *Pruning and grafting the attribute tree*

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

Pruning is carried out by dropping any subtree from the quasi-tree. The attributes dropped will not be included in the fact scheme, hence it will be impossible to use them to aggregate data. For instance, on the sale example, the subtree rooted in *county* may be dropped from the *brand* branch.

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

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

```
   drop v;
}
```

Thus, grafting is carried out by moving the entire subtree with root in v to its father(s) v'; if we denote with t the attribute tree and with I the set of its vertices, procedure `graft(v)` returns cnt(t,I–{v}). As a result, attribute v will not be included in the fact scheme and the corresponding aggregation level will be lost; on the other hand, all the descendant levels will be maintained. In the sale example, the detail of purchase tickets is uninteresting and vertex *ticket number* can be grafted. In general, grafting a child of the root corresponds to making the granularity of fact instances coarser and, if the node grafted has two or more children, leads to increasing the number of dimensions in the fact scheme.

Two considerations:

- A one-to-one relationship can be thought of as a particular kind of many-to-one relationship, hence, it can be inserted into the attribute tree. Nevertheless, in a DW query, drilling down along a one-to-one relationship means adding a row header to the result without introducing further detail; thus, it is often worth grafting from the attribute tree the attributes following one-to-one relationships, or representing them as non-dimension attributes.
- Let entity E have a compound identifier including the internal attributes $a_1,...a_m$ and the external attributes $b_1,...b_t$ ($m,t \geq 0$). The algorithm outlined in Subsection 5.2 translates E into a vertex $c=a_1+...a_m+b_1+...b_t$ with children $a_1,...a_m$ (children $b_1,...b_t$ will be added when translating the entities which they identify). Essentially, two situations may occur. If the granularity of E must be preserved in the fact scheme, vertex c is maintained while one or more of its children may be pruned; for instance, vertex *district no.+state* is maintained since aggregation must be carried out at the level of single districts, while *district no.* may be pruned since it does not express any interesting aggregation. Otherwise, if the granularity expressed by E is too fine, c may be grafted and some or all of its children maintained. Similar considerations can be made, when the source scheme is logical, for the relation schemes with compound primary key.

After grafting *ticket number* and pruning *county*, *district no.* and *size*, the attribute tree is transformed as shown in Figure 13.

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

5.4. *Defining dimensions*

Dimensions determine how fact instances may be aggregated significantly for the decision-making process. The dimensions must be chosen in the attribute tree among the children vertices of the root (including the attributes which have become children of the root after the quasi-tree has been grafted); they may correspond either to discrete attributes, or to

ranges of discrete or continuous attributes. Their choice is crucial for the DW design since it determines the granularity of fact instances.

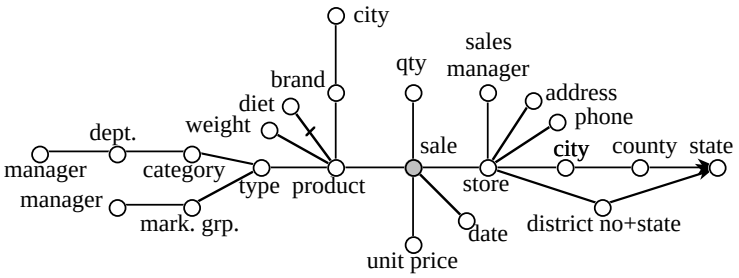


Fig. 13. Attribute tree for the sale example after pruning and grafting.

Each primary fact instance "summarizes" all the instances of entity (relation scheme) F corresponding to a combination of dimension values. If the dimension pattern includes all the attributes which constitute an identifier (the primary key) of F, every primary instance corresponds to one instance (tuple) of F; often, one or more of the attributes which identify F are either pruned or grafted, hence, each primary instance may correspond to several instances (tuples) of F.

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

In the sale example, the attributes chosen as dimensions are *product*, *store* and *date*.

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

5.5. *Defining measures*

Measures are defined by applying, to numerical attributes of the attribute tree, aggregation functions which operate on all the instances (tuples) of F corresponding to each primary fact instance. The aggregation function typically consists either in the sum/average/maximum/ minimum of expressions or in the count of the number of entity

instances (tuples). A fact may have no attributes, if the only information to be recorded is the occurrence of the fact.

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

qty sold =
```
SELECT SUM(S.qty)
FROM SALES S,TICKETS T
WHERE S.tickNo = T.tickNo
GROUP BY S.product,T.date,T.store
```
revenue =
```
SELECT SUM(S.qty*S.unitPrice)
FROM SALES S,TICKETS T
WHERE S.tickNo = T.tickNo
GROUP BY S.product,T.date,T.store
```
no. of customers =
```
SELECT COUNT(*)
FROM SALES S,TICKETS T
WHERE S.tickNo = T.tickNo
GROUP BY S.product,T.date,T.store
```

At this point, the aggregation functions more used for each combination measure/dimension should be represented; if necessary, the preferred ordering of dimensions for aggregation should be specified.

5.6. *Defining hierarchies*

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

The attribute tree already shows a plausible organization for hierarchies; at this stage, it is still possible to prune and graft the quasi-tree in order to eliminate irrelevant details. It is also possible to add new levels of aggregation by defining ranges for numerical attributes; typically, this is done on the time dimension. In the sale example, the time dimension is enriched by introducing attributes *month*, *quarter*, etc.

During this phase, the attributes which should not be used for aggregation but only for informative purposes may be identified as non-dimension attributes (for instance, *address*, *weight*, etc.). It should be noted that non-numerical attributes which are children of the root but have not been chosen as dimensions must necessarily either be grafted (if the granularity of the primary fact instances is coarser than that of the fact) or be represented as non-dimension (if the two granularities are equal).

6. Conclusion

In this paper we have proposed a conceptual model for data warehouse design and a semi-automated methodology for deriving it from the documentation describing the information system of the enterprise. The DFM is independent of the target logical model (multidimensional or relational); in order to bridge the gap between the fact schemes and the DW logical scheme, a methodology for logical design is needed. As in operational information systems, DW logical design should be based on an estimate of the expected workload and data volumes. The workload will be expressed in terms of query patterns and their frequencies; data volumes will be computed by considering the sparsity of facts and the cardinality of the dimension attributes.

Our current work is devoted to developing the methodology for logical design and implementing it within an automated tool. Among the specific issues we are investigating, we mention the following:

- *Partitioning of the DW into integrated data marts.*
- *View materialization.* This problem involves the whole dimensional scheme; in fact, due to the presence of drill-across queries, cross-optimization must be carried out.
- *Selection of the logical model*. Each materialized view can be mapped on the logical level by adopting different models (star scheme, constellation scheme, snowflake scheme).
- *Translation into fact and dimension tables*. The fact and dimension tables are created according to the logical models adopted.
- *Vertical partitioning of fact tables*. The query response time can be reduced by considering the set of measures required by each query.
- *Horizontal partitioning of fact tables*. The query response time can be reduced by considering the selectivity of each query.

References

1. R. Agrawal, A. Gupta and S. Sarawagi, Modeling multidimensional databases, *IBM Research Report*, IBM Almaden Research Center, 1995.
2. E. Baralis, S. Paraboschi and E. Teniente, Materialized view selection in multidimensional database, *Proc. 23rd Int. Conf. on Very Large Data Bases*, Athens, Greece, 1997, 156-165.
3. R. Barquin and S. Edelstein, *Planning and Designing the Data Warehouse*. (Prentice Hall, 1996).
4. L. Cabibbo and R. Torlone, A logical approach to multidimensional databases, eds. H.J. Schek, F. Saltor, I. Ramos, G. Alonso, *Advances in DB technology - EDBT 98*, (LNCS 1377, Springer, 1998) 183-197.
5. S. Chaudhuri and U. Dayal, An overview of data warehousing and OLAP technology, *SIGMOD Record* **26**, 1 (1997) 65-74.
6. S. Chaudhuri and K. Shim, Including group-by in query optimization, *Proc. 20th Int. Conf. on Very Large Data Bases* (1994) 354-366.
7. G. Colliat, OLAP, relational and multidimensional database systems, *SIGMOD Record* **25**, 3 (1996) 64-69.

8. C. Fahrner and G. Vossen, A survey of database transformations based on the Entity-Relationship model, *Data & Knowledge Engineering* **15**, 3 (1995) 213-250.
9. U.M. Fayyad, G. Piatetsky-Shapiro and P. Smyth, Data mining and knowledge discovery in databases: an overview, *Comm. of the ACM* **39**, 11 (1996).
10. M. Golfarelli, D. Maio and S. Rizzi, Conceptual design of data warehouses from E/R schemes, *Proc. Hawaii International Conference on System Sciences*, Kona, Hawaii (1998) 334-343.
11. J. Gray, A. Bosworth, A. Lyman and H. Pirahesh, Data-Cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals, *Technical Report* MSR-TR-95-22, Microsoft Research, 1995.
12. A. Gupta, V. Harinarayan and D. Quass, Aggregate-query processing in data-warehousing environments, *Proc. 21th Int. Conf. on Very Large Data Bases*, Zurich, Switzerland (1995).
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 Int. Conf. on Very Large Data Bases*, Athens, Greece (1997) 106-115.
15. V. Harinarayan, A. Rajaraman and J. Ullman, 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, *Bullettin of Technical Committee on Data Engineering* **20**, 1 (1997).
17. R. Kimball, *The data warehouse toolkit* (John Wiley & Sons, 1996).
18. D. Lomet and B. Salzberg, The Hb-Tree: a multidimensional indexing method with good guaranteed performance, *ACM Trans. On Database Systems* **15**, 44 (1990) 625-658.
19. F. McGuff, Data modeling for data warehouses, `http://members.aol.com/fmcguff/dwmodel/dwmodel.htm` (1996).
20. P. O'Neil and G. Graefe, Multi-table joins through bitmapped join indices, *SIGMOD Record* **24**, 3 (1995) 8-11.
21. K. Ross, D. Srivastava and D. Chatziantoniou, Complex aggregation at multiple granularities, *Proc. Int. Conf. on Extending Database Technology* (1998) 263-277.
22. S. Sarawagi, Indexing OLAP data, *Bullettin of Technical Committee on Data Engineering* **20**, 1 (1997).
23. Y. Zhuge, H. Garcia-Molina and J. L. Wiener, The Strobe Algorithms for Multi-Source Warehouse Consistency, *Proc. Conference on Parallel and Distributed Information Systems*, Miami Beach, FL (1996).