I. 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 summarised information which may help managers in making decisions. Data warehousing systems address this issue by enabling managers to acquire and integrate information from different sources and to query very large databases efficiently. Monitoring and communication systems should be integrated with information of the new acquired data, which could be used to enhance the performance of the decision-making process of the organisation. The support of these tasks is provided by the development of data warehouses, the analysis of data, and the use of information technologies. The representation of data in a data warehouse is through the use of a conceptual model. This model is used to store, retrieve, and manipulate data. The most common conceptual model is the entity-relationship (ER) model, which is based on the use of entities, relationships, and attributes. The ER model is a simple and easy-to-use model that can be used to represent the data in a data warehouse. The ER model is based on the concept of a database, which is a collection of data that can be stored, retrieved, and manipulated. The ER model is based on the concept of a database, which is a collection of data that can be stored, retrieved, and manipulated. The ER model is based on the concept of a database, which is a collection of data that can be stored, retrieved, and manipulated.
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.

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 spent to use E/R schemes as the input for designing non-relational databases as well; 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 the real world 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 unifying scheme that manages dimensions and facts in terms of sets of real instances.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.

In this paper we present a graphical conceptual model for DWs, called Dimensional Fact Model (DFM). The representation of the real world 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.
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, optimization of complex queries, advanced indexing techniques, and friendly visual interface to be used for On-Line Analytical Processing (OLAP) and data mining.

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 and index selection, 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.

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. On the other hand, in relational DBMSs the multidimensional model of the DW is mapped in most cases through star schemes 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 performances of data warehousing applications, which is the main goal in data warehousing. The second phase, therefore, focuses on optimizing the system performances. As to the third phase, optimizing the system performances, involves further optimization of complex queries, advanced indexing techniques, and friendly visual interface to be used for OLAP and data mining.
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 \( \text{path}_{0j}(g) \subseteq g \) a directed path starting in \( v_0 \) and ending in \( v_j \); given \( v_i \in \text{path}_{0j}(g) \), we will denote with \( \text{path}_{ij}(g) \subseteq g \) a directed path starting in \( v_i \) and ending in \( v_j \). We will denote with \( \text{sub}(g, v_i) \subseteq g \) the quasi-tree rooted in \( v_i \neq v_0 \).

Within a quasi-tree, two or more directed paths 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 one-to-one relationships, and determine how facts may be grouped into higher categories. In relation hierarchies, dimension attributes are organized according to a partial order on the dimension attributes. Dimension attributes which describe the fact at the lowest level are called attributes, while attributes which describe the fact at a higher level are called hierarchies. A hierarchy on the product dimension will probably include the dimension attributes product type, category, department, manager, etc.

Some positive aspects of the approach of treating dimensions and measures symmetrically (e.g., the approach taken by some applications of the formal data model) are the uniformity and ease of the logical model of the database, and 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 fine-grained. Moreover, we consider the formal definition of the logical model of the database to be an important enhancement from the perspective of OLAP.

Dimension attributes cannot be used for aggregation.

Definition 2. A fact scheme is a sextuple \( f = (M, A, N, R, O, S) \) where:

\[
\begin{align*}
&f = (M, A, N, R, O, S) \\
&\text{where:} \\
&M \subseteq \mathbb{R} \cup \mathbb{Q} \cup \mathbb{Z} \\
&A \subseteq M \\
&N \subseteq \{ \text{dimension attributes} \} \\
&R \subseteq \{ \text{relationships} \} \\
&O \subseteq \{ \text{operators} \} \\
&S \subseteq \{ \text{storage strategies} \}
\end{align*}
\]
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, \( \text{Dom}(a_i) \).

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 a couple \((a_i, a_j)\) models a one-to-one relationship between attributes \( a_i \) and \( a_j \). We call this a dimension pattern the set \( \text{Dim}(f) = \{a_i \in A | \exists (a_0, a_i) \in R\} \); each element in \( \text{Dim}(f) \) is a dimension.

We denote with \( \alpha_i.a_j \) the value of \( a_j \) determined by value \( \alpha_i \) assumed by \( a_i \) (for instance, Venice.state denotes Italy); by convention, \( \alpha_i.a_i = \alpha_i \).

In the following we will discuss the graphic representation of the concepts introduced above with reference to the fact scheme \( \text{SALE} \), shown in Figure 1, which describes the sales in a chain store. This scheme is well structured and the \( \text{INVENTORY} \) and the \( \text{SHIPMENT} \) schemes proposed in Section 4 are based on the same scheme 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, \( \text{quantity sold} \), \( \text{revenue} \) and \( \text{no. of customers} \) 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, \( \text{address} \)).

Subtrees rooted in dimensions are hierarchies. The arc connecting two attributes represents a one-to-one relationship between them (for instance, there is a many-to-one relationship between \( \text{city} \) and \( \text{county} \)); thus, every directed path within one hierarchy necessarily represents a one-to-one relationship between city and county.

Subtrees rooted in dimensions are hierarchies. The arc connecting two attributes represents a one-to-one relationship between them (for instance, there is a many-to-one relationship between city and county).
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 **α** 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** and dimension **d**:

- **m**:**d**
- **d**:**m**
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 \(\Omega\) (if any) such that \((m_j, d_i, \Omega) \in S\) (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 \(\exists / \Omega \neq 'SUM' \mid (m_j, d_i, \Omega) \in S\) (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 \((m_j, d_i, \Omega) \in S\).

Additivity will be discussed in more detail in Subsection 3.3.
A dimension $d_i \in \text{Dim}(f)$ is said to be hidden within $P$ if no attribute of its hierarchy $\text{sub}(q, f, d_i)$ appears within $P$. An aggregation pattern $P$ is legal with reference to measure $m_j \in M$ if

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

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

Let $P = \{a_1, \ldots, a_v\}$ be an aggregation pattern, and $d_{h^*}$ denote the dimension whose hierarchy includes $a_h \in P$. The secondary fact instance $sf(\beta_1, \ldots, \beta_v)$ corresponding to the combination of values $(\beta_1, \ldots, \beta_v) \in \text{Dom}(a_1) \times \ldots \times \text{Dom}(a_v)$ aggregates the set of primary fact instances $\{\text{pf}(\alpha_1, \ldots, \alpha_n) \mid \forall k \in \{1, \ldots, n\} \alpha_k \in \text{Dom}(d_k) \land \forall h \in \{1, \ldots, v\} \alpha_{h^*}.a_h = \beta_h\}$ 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 instances of values $(\beta_1, \ldots, \beta_v) \in \text{Dom}(a_1) \times \ldots \times \text{Dom}(a_v)$.

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 \text{no. of customers} is not included since it cannot be aggregated along the product dimension. Figure 2.a shows a primary fact instance on the sale scheme. Figure 2.b shows the corresponding primary fact instances.
In the following, we will use sometimes the term *pattern* to denote either a 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:

\[
\text{fact instance expression} ::= \text{fact name} (\text{pattern clause} ; \text{selection clause})
\]

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:

\[
\text{measure values} ::= \text{fact instance expression}.\text{measure}
\]

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

\[
(f(d_1, ... , d_p, a_{p+1}, ... , a_{v}; e_1(b_{i1}), ... , e_h(b_{ih})))
\]

where we have assumed, without loss of generality, that:

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

\[
\text{measure values} ::= \text{fact instance expression}.\text{measure}
\]

A fact instance expression has the general form:

\[
\text{fact instance expression} ::= \text{fact name} (\text{pattern clause} ; \text{selection clause})
\]

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:

\[
\text{measure values} ::= \text{fact instance expression}.\text{measure}
\]

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

\[
(f(d_1, ... , d_p, a_{p+1}, ... , a_{v}; e_1(b_{i1}), ... , e_h(b_{ih})))
\]

where we have assumed, without loss of generality, that:

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

\[
\text{measure values} ::= \text{fact instance expression}.\text{measure}
\]

A fact instance expression has the general form:

\[
\text{fact instance expression} ::= \text{fact name} (\text{pattern clause} ; \text{selection clause})
\]

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:

\[
\text{measure values} ::= \text{fact instance expression}.\text{measure}
\]
If \( p = v = n \) (i.e., the pattern clause describes the dimension pattern), expression (1) denotes the set of primary fact instances
\[
\{ pf(\alpha_1, \ldots, \alpha_n) \mid \forall k \in \{1, \ldots, n\} \alpha_k \in \text{Dom}(d_k) \wedge \forall j \in \{1, \ldots, h\} e_j(\alpha_{ij}^*.b_{ij}) \}
\]

For instance, the expression
\[
\text{SALE}(\text{date}, \text{product}, \text{store}, \text{promotion}; \text{date}.\text{year} \geq '1995', \text{product} = 'P5').
\]
\( \text{qtySold} \) 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 \( b_{ij} \) be the attribute involved by \( e_j \); we say \( e_j \) is \emph{external} if \( \exists a_{ij}^* \in P \mid a_{ij}^* \in \text{path}_0(ij)(\text{qt}(f)) \), \emph{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 \( e_1, \ldots, e_r \) and \( e_{r+1}, \ldots, e_h \) be, respectively, the external and the internal predicates (\( 0 \leq r \leq h \)); in this case, expression (1) denotes the set of secondary fact instances
\[
\{ sf(\beta_1, \ldots, \beta_v) \mid \forall k \in \{1, \ldots, v\} \beta_k \in \text{Dom}(a_k) \wedge \forall j \in \{1, \ldots, r\} e_j(\beta_{ij}^*.b_{ij}) \}
\]

where each \( sf(\beta_1, \ldots, \beta_v) \) aggregates the set of primary fact instances
\[
\{ pf(\alpha_1, \ldots, \alpha_n) \mid \forall k \in \{1, \ldots, n\} \alpha_k \in \text{Dom}(d_k) \wedge \forall h \in \{1, \ldots, v\} \alpha_h^*.a_h = \beta_h \wedge \forall j \in \{r+1, \ldots, h\} e_j(\alpha_{ij}^*.b_{ij}) \}
\]

Consider, for instance, the two expressions
\[
\text{SALE}(\text{date}.\text{month}, \text{product}.\text{type}; \text{date}.\text{month} = 'JAN98', \text{product}.\text{category} = 'food').
\]
\( \text{qtySold} \) \( \text{SALE}(\text{date}.\text{month}, \text{product}.\text{type}; \text{date}.\text{month} = 'JAN98', \text{product}.\text{brand} = 'General'). \( \text{qtySold} \)

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 \text{month} and on \text{category} are external, whereas that on \text{brand} is internal.
A significant amount of DW queries require consolidating data on multiple levels of abstraction; these 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:

\[
\text{SALE}(\text{date}, \text{month}, \text{product}) \cup \text{SALE}(\text{date}, \text{year}, \text{product}) \cup \text{SALE}(\text{product}, \text{brand} = \text{General})
\]

Table I. A sample set of data for products.

<table>
<thead>
<tr>
<th>Type</th>
<th>Month</th>
<th>Qty Sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>shirt</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>drink</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>clothes</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>food</td>
<td>JAN98</td>
<td>200</td>
</tr>
<tr>
<td>General</td>
<td>JAN98</td>
<td>50</td>
</tr>
<tr>
<td>General</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>General</td>
<td>JAN98</td>
<td>20</td>
</tr>
</tbody>
</table>

Table II. Results of two expressions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Month</th>
<th>Qty Sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>drink</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>drink</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>clothes</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>food</td>
<td>JAN98</td>
<td>200</td>
</tr>
<tr>
<td>General</td>
<td>JAN98</td>
<td>50</td>
</tr>
<tr>
<td>General</td>
<td>JAN98</td>
<td>100</td>
</tr>
<tr>
<td>General</td>
<td>JAN98</td>
<td>20</td>
</tr>
</tbody>
</table>

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).
Additivity

Additivity 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 $d_k \in \text{Dim}(f)$ if there exists $\Omega \in \Sigma$, non-aggregable otherwise. Measure $m_j$ is said to be additive on $d_k$ if there exists $\Omega \in \Sigma$, non-additive otherwise.

As a guideline, most measures in a fact scheme should be additive. An example of an additive measure in the sale scheme is $\text{qty sold}$: 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-aggregable 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 operations such as average, maximum, minimum: Figure 5 shows an example where both operators AVG and MIN can be used for aggregation of measure $\text{qty}$, expressing, for each product, the number of copies present within each warehouse during each week.
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 \( \Omega \) and the aggregation pattern
\( P = \{d_1, \ldots, d_{k-1}, a_k, d_{k+1}, \ldots, d_n\} \), 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:

\[
\begin{align*}
&\mathcal{O} = \{ \beta \in \text{Dom}(d_k) : a_k = \alpha_k \} \\
&f(d_1, \ldots, d_k, \ldots, d_n; d_1 = \alpha_1, \ldots, d_k = \beta, \ldots, d_n = \alpha_n). m_j = \Omega \mathcal{O}
\end{align*}
\]

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 each product of that type for that week. The total
quantity for each week is the sum of the quantities of all products of that type for that week: the local
aggregation pattern for \( qty \) along the product dimension is the sum. The secondary fact
instances at patterns \{month, product\} and \{product\} are shown in Table V; they are
calculated using the average function to aggregate \( qty \) by month.

<table>
<thead>
<tr>
<th>Month</th>
<th>Product</th>
<th>Jan98</th>
<th>Feb98</th>
<th>Mar98</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Product</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>11</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>10</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>

Table III. Primary fact instances for a given warehouse (symbol '-' denotes a missing fact instance).
As a matter of fact, when using for instance pattern \{\text{week}\}, secondary fact instances could be more conveniently computed by aggregating secondary fact instances at pattern \{\text{week}, \text{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: \text{SUM}, \text{MIN}, \text{MAX}, \text{COUNT}, \text{AND}, \text{OR} fall within the first category, \text{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 \(\Omega'\) and \(\Omega''\) 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 \(P\), two different aggregation sequences can be adopted:

\[
\begin{align*}
\text{sequence 1:} & \quad \{d_1, d_2\} \rightarrow \Omega'\{a_1, d_2\} \rightarrow \Omega''\{a_1, a_2\} \\
\text{sequence 2:} & \quad \{d_1, d_2\} \rightarrow \Omega''\{d_1, a_2\} \rightarrow \Omega'\{a_1, a_2\}
\end{align*}
\]

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

\(\Omega' = \Omega'' \in \{\text{SUM}, \text{MIN}, \text{MAX}, \text{COUNT}, \text{AND}, \text{OR}\}\) •

As a matter of fact, when using for instance pattern \{\text{product}\}, secondary fact instances at pattern \{\text{product}\} can be more conveniently computed by aggregating secondary fact instances at pattern \{\text{product}\} instead of aggregating the primary fact instances.

As a matter of fact, when using for instance pattern \{\text{type}\}, secondary fact instances at pattern \{\text{type}\} can be more conveniently computed by aggregating secondary fact instances at pattern \{\text{type}\} instead of aggregating the primary fact instances.

### Table IV. Secondary fact instances at patterns \{\text{week}\}, \text{type}\}

<table>
<thead>
<tr>
<th>Week</th>
<th>Product</th>
<th>January</th>
<th>February</th>
<th>March</th>
<th>April</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1</td>
<td>9.60</td>
<td>9.25</td>
<td>9.44</td>
<td>9.68</td>
</tr>
<tr>
<td>2</td>
<td>P2</td>
<td>5.00</td>
<td>4.62</td>
<td>4.83</td>
<td>5.63</td>
</tr>
<tr>
<td>3</td>
<td>P3</td>
<td>2.80</td>
<td>2.37</td>
<td>2.61</td>
<td>3.26</td>
</tr>
<tr>
<td>4</td>
<td>P4</td>
<td>1.50</td>
<td>1.10</td>
<td>1.19</td>
<td>1.41</td>
</tr>
<tr>
<td>5</td>
<td>P5</td>
<td>0.67</td>
<td>0.72</td>
<td>0.69</td>
<td>0.81</td>
</tr>
</tbody>
</table>

### Table V. Secondary fact instances at patterns \{\text{month}\}, \text{product}\}

<table>
<thead>
<tr>
<th>Month</th>
<th>Product</th>
<th>January</th>
<th>February</th>
<th>March</th>
<th>April</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1</td>
<td>15.00</td>
<td>13.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>2</td>
<td>P2</td>
<td>14.00</td>
<td>11.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>3</td>
<td>P3</td>
<td>13.00</td>
<td>10.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>4</td>
<td>P4</td>
<td>12.00</td>
<td>9.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>5</td>
<td>P5</td>
<td>11.00</td>
<td>8.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
</tbody>
</table>

### Table VI. Secondary fact instances at patterns \{\text{product}\}

<table>
<thead>
<tr>
<th>Product</th>
<th>January</th>
<th>February</th>
<th>March</th>
<th>April</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>15.00</td>
<td>13.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>P2</td>
<td>14.00</td>
<td>11.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>P3</td>
<td>13.00</td>
<td>10.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>P4</td>
<td>12.00</td>
<td>9.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
<tr>
<td>P5</td>
<td>11.00</td>
<td>8.25</td>
<td>12.78</td>
<td>15.00</td>
</tr>
</tbody>
</table>
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 \(\text{qty}\) along \(\text{week}\), applying the two aggregation sequences \{\text{week}, \text{product}\} \rightarrow \text{SUM} \{\text{week}, \text{type}\} \rightarrow \text{MIN} \{\text{month}, \text{type}\}\) or \{\text{week}, \text{product}\} \rightarrow \text{MIN} \{\text{month}, \text{product}\} \rightarrow \text{SUM} \{\text{month}, \text{type}\}\) would lead to different results.

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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})). It should be noted that the COUNT operator behaves differently from the others.

\(3.4.\) Empty facts

A fact scheme is said to be empty if it has no measures (\(M = \emptyset\)). In this case, primary facts can be described in terms of other dimensions, as shown in Table VI. Secondary fact instances at patterns \{month, type\} (top left), \{month\} (top right), \{type\} (bottom left), {} (bottom right).

In this form each fact scheme, we believe that the most suitable ordering is (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})).

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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})).

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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})).

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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{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. Then, a preferred aggregation sequence is defined by specifying an ordering for each dimension. 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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})).

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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})).

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 (\text{product}, \text{warehouse}, \text{week}) (or, indifferently, (\text{warehouse}, \text{product}, \text{week})).
In the DFM, different facts are represented in different fact schemes. However, part of 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:

1. In the first approach, which requires using the COUNT operator, the information carried by each secondary fact instance is the number of corresponding primary fact instances. To find the number of students of each sex who attended the course.

2. In the second approach, which requires using the COUNT operator, the information carried by each secondary fact instance is related to the existence of the corresponding primary fact instances. To find the number of students of each sex who attended the course.

In an empty fact scheme, two approaches to the problem of aggregation can be used for event tracking or as coverage tables.

Moreover, the notion of an empty fact scheme can be useful for introducing the concept of a new scheme; since the same attribute a fact scheme may denote either the students who during 1998 attended all the database courses in the Computer Science Faculty (AND operator), or the students who during 1998 attended at least one database course in the Computer Science Faculty (OR operator).
possibly with different domains, we will denote with $\text{Dom}_f(a_i)$ 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 $\text{Dom}_{f'}(a_i) \cap \text{Dom}_{f''}(a_i) \neq \emptyset$.

**Definition 6.** Given a quasi-tree $t=(V \cup \{a_0\},E)$ with root $a_0$, a subset of vertices $I \subseteq V$, we define the **contraction** of $t$ on $I$ as the quasi-tree $\text{cnt}(t,I)=(I \cup \{a_0\},E^*)$ where:

$$E^* = \{(a_i,a_j) | a_i \in I \cup \{a_0\} \land a_j \in I \land \exists \text{path}_{ij}(t) \land \forall a_k \in I - \{a_i,a_j\} a_k \not\in \text{path}_{ij}(t)\}$$

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

**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 $\text{cnt}(\text{qt}(f'),I)$ and $\text{cnt}(\text{qt}(f''),I)$ are equal.

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

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 connected by an arc in the quasi-tree to be contracted. Figure 7 shows a quasi-tree and its contraction on a subset of the vertices.

**Figure 7.** A quasi-tree (a) and its contraction on the black vertices (b). The grey vertex is the root.
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' ⊗ f" = (M, A, N, R, O, S) where:

- M = M' ∪ M";
- A = A' ∩ A";
- ∀ a_i ∈ A (Dom f' ⊗ f"(a_i) = Dom f'(a_i) ∩ Dom f"(a_i));
- N = N' ∩ N";
- R = {(a_i, a_j) | (a_i, a_j) ∈ cnt(qt(f'), A) or (a_i, a_j) ∈ cnt(qt(f"), A)};
- O = {(a_i, a_j) ∈ R | ∃ (a_w, a_z) ∈ O' or (a_w, a_z) ∈ O" | (a_w, a_z) ∈ path_{ij}(qt(f')) or (a_w, a_z) ∈ path_{ij}(qt(f"))};
- S = {(m_j, d_i, Ω) | d_i ∈ Dim(f' ⊗ f") and (∃ (m_j, d_k, Ω) ∈ S' or d_i ∈ sub(qt(f"), d_k)) or (∃ (m_j, d_k, Ω) ∈ S" and d_i ∈ 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. However, the primary dimension week is included only in the INVENTORY scheme, since the SHIPMENT scheme does not provide it.

In general, 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. However, the primary dimension week is included only in the INVENTORY scheme, since the SHIPMENT scheme does not provide it.

Even two non-strictly compatible schemes can be overlapped; since in this case the two contracted quasi-trees are different, there must be at least one conflict in the inter-attribute dependencies. The resulting scheme is defined as in the case of strict compatibility, except that each conflict is resolved by representing an inter-attribute dependency that 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, 3root → 1, 2
2 → 42 → 5
4 → 54 → 5
1 → 3

The overlap of f and f' is the scheme f' ⊗ f" = (M, N, R, O, S) where:

 Definition 8. Given two strictly compatible schemes f' and f", we define the overlap of f' and f" as the scheme f' ⊗ f" = (M, N, R, O, S) where:

- M = M' ∪ M";
- A = A' ∩ A";
- ∀ a_i ∈ A (Dom f' ⊗ f"(a_i) = Dom f'(a_i) ∩ Dom f"(a_i));
- N = N' ∩ N";
- R = {(a_i, a_j) | (a_i, a_j) ∈ cnt(qt(f'), A) or (a_i, a_j) ∈ cnt(qt(f"), A)};
- O = {(a_i, a_j) ∈ R | ∃ (a_w, a_z) ∈ O' or (a_w, a_z) ∈ O" | (a_w, a_z) ∈ path_{ij}(qt(f')) or (a_w, a_z) ∈ path_{ij}(qt(f"))};
- S = {(m_j, d_i, Ω) | d_i ∈ Dim(f' ⊗ f") and (∃ (m_j, d_k, Ω) ∈ S' or d_i ∈ sub(qt(f"), d_k)) or (∃ (m_j, d_k, Ω) ∈ S" and d_i ∈ sub(qt(f'), d_k))}.

Formally:

Definition 8. Given two strictly compatible schemes f' and f", we define the overlap of f' and f" as the scheme f' ⊗ f" = (M, N, R, O, S) where:

- M = M' ∪ M";
- A = A' ∩ A";
- ∀ a_i ∈ A (Dom f' ⊗ f"(a_i) = Dom f'(a_i) ∩ Dom f"(a_i));
- N = N' ∩ N";
- R = {(a_i, a_j) | (a_i, a_j) ∈ cnt(qt(f'), A) or (a_i, a_j) ∈ cnt(qt(f"), A)};
- O = {(a_i, a_j) ∈ R | ∃ (a_w, a_z) ∈ O' or (a_w, a_z) ∈ O" | (a_w, a_z) ∈ path_{ij}(qt(f')) or (a_w, a_z) ∈ path_{ij}(qt(f"))};
- S = {(m_j, d_i, Ω) | d_i ∈ Dim(f' ⊗ f") and (∃ (m_j, d_k, Ω) ∈ S' or d_i ∈ sub(qt(f"), d_k)) or (∃ (m_j, d_k, Ω) ∈ S" and d_i ∈ sub(qt(f'), d_k))}.

Formally:

Definition 8. Given two strictly compatible schemes f' and f", we define the overlap of f' and f" as the scheme f' ⊗ f" = (M, N, R, O, S) where:

- M = M' ∪ M";
- A = A' ∩ A";
- ∀ a_i ∈ A (Dom f' ⊗ f"(a_i) = Dom f'(a_i) ∩ Dom f"(a_i));
- N = N' ∩ N";
- R = {(a_i, a_j) | (a_i, a_j) ∈ cnt(qt(f'), A) or (a_i, a_j) ∈ cnt(qt(f"), A)};
- O = {(a_i, a_j) ∈ R | ∃ (a_w, a_z) ∈ O' or (a_w, a_z) ∈ O" | (a_w, a_z) ∈ path_{ij}(qt(f')) or (a_w, a_z) ∈ path_{ij}(qt(f"))};
- S = {(m_j, d_i, Ω) | d_i ∈ Dim(f' ⊗ f") and (∃ (m_j, d_k, Ω) ∈ S' or d_i ∈ sub(qt(f"), d_k)) or (∃ (m_j, d_k, Ω) ∈ S" and d_i ∈ sub(qt(f'), d_k))}.

Formally:

Definition 8. Given two strictly compatible schemes f' and f", we define the overlap of f' and f" as the scheme f' ⊗ f" = (M, N, R, O, S) where:

- M = M' ∪ M";
- A = A' ∩ A";
- ∀ a_i ∈ A (Dom f' ⊗ f"(a_i) = Dom f'(a_i) ∩ Dom f"(a_i));
- N = N' ∩ N";
- R = {(a_i, a_j) | (a_i, a_j) ∈ cnt(qt(f'), A) or (a_i, a_j) ∈ cnt(qt(f"), A)};
- O = {(a_i, a_j) ∈ R | ∃ (a_w, a_z) ∈ O' or (a_w, a_z) ∈ O" | (a_w, a_z) ∈ path_{ij}(qt(f')) or (a_w, a_z) ∈ path_{ij}(qt(f"))};
- S = {(m_j, d_i, Ω) | d_i ∈ Dim(f' ⊗ f") and (∃ (m_j, d_k, Ω) ∈ S' or d_i ∈ sub(qt(f"), d_k)) or (∃ (m_j, d_k, Ω) ∈ S" and d_i ∈ sub(qt(f'), d_k))}.
The common elemental dependencies (namely, root \( 1 \rightarrow 2 \)) 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 vertex 2 since, in both (a) and (b), the dependency 2 \( \rightarrow \) 5 holds.

\[
\begin{align*}
\text{(a)} & \quad \text{(b)} & \quad \text{(c)}
\end{align*}
\]

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 \( f' \odot f'' = (M,A,N,R,O,S) \) where

- \( M = M' \cup M'' \)
- \( A = A' \cap A'' \)
- \( \forall a_i \in A, \text{Dom}(f' \odot f''(a_i)) = \text{Dom}(f'(a_i)) \cap \text{Dom}(f''(a_i)) \)
- \( N = N' \cap N'' \)
- \( R = \{(a_i,a_j) | \exists p_{ij}(\text{cnt}(qt(f'),A)) \land \exists p_{ij}(\text{cnt}(qt(f''),A)) \land \forall a_w \neq a_i | (\exists p_{wj}(\text{cnt}(qt(f'),A)) \land \exists p_{wj}(\text{cnt}(qt(f''),A)) \land p_{ij}(\text{cnt}(qt(f'),A)) \subseteq p_{wj}(\text{cnt}(qt(f'),A)) \land p_{ij}(\text{cnt}(qt(f''),A)) \subseteq p_{wj}(\text{cnt}(qt(f''),A)))} \)
- \( O = \{(a_i,a_j) \in R | (\exists (a_w,a_z) \in O' | (a_w,a_z) \in \text{path}_{ij}(qt(f'))) \lor (\exists (a_w,a_z) \in O'' | (a_w,a_z) \in \text{path}_{ij}(qt(f''))) \} \)
- \( S = \{(m_j,d_i,<op>) | d_i \in \text{Dim}(f' \odot f'') \land (\exists (m_j,d_k,<op>) \in S' \land d_i \in \text{sub}(qt(f'),d_k)) \lor (\exists (m_j,d_k,<op>) \in S'' \land d_i \in \text{sub}(qt(f''),d_k)) \} \)

Queries formulated on the overlap of two schemes are actually formulated on one or over of the schemes involved, depending on which measures are involved in the query. In general, let \( q = f(P,<\text{sel}>) \) be a query formulated on the overlapped fact scheme. From the conceptual point of view, \( q \) is equivalent to \( m \) queries \( q_1,...,q_m \), where \( q_i = f_i(P;<\text{sel},d_1 \in \text{Dom}(f_i(d_1)) \land ... \land d_n \in \text{Dom}(f_i(d_n)) \) and \( d_1,...,d_n \) are the dimensions of \( f_i \). An example of query formulated on an overlap is:

\[
\text{SHIPMENT} \odot \text{INVENTORY} (\text{month}, \text{product} ; \text{month} \cdot \text{year} = '1997')
\]

\[
\text{inventoryQty} = \text{qtyShipped}
\]

where

\[
\begin{align*}
\text{overof}(f',f'') & = f' \odot f'' \quad \text{and} \quad \text{LX} \odot \text{LX} = \text{LX} \odot \text{LX} = \text{LX} \odot \text{LX}
\end{align*}
\]
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 SALES 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.

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.

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

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

On the physical scheme: A fact may be represented either by an entity F or by an n-ary relationship R between entities E₁,...,Eₙ. In the latter case, for the sake of simplicity, we consider μ₁(ER) and max₁(ER) respectively the minimum and maximum cardinalities with which entity Eᵢ participates in relationship R, i.e., μ₁(ER) ∈ {0,1} and max₁(ER) ∈ {1,N}

Facts are concepts of primary interest for the decision-making process. Typically, they correspond to events occurring dynamically in the enterprise world. Properties of the domain correspond to nearly-static entities such as STORE and CITY, and nearly-static relationships such as SALES and TICKETS, for defining facts whose representation exhibits entities and relationships of relationships (relation schemes) representing frequently updated archives - such as SALES and TICKETS, for defining facts whose representation exhibits entities and relationships of relationships (relation schemes) representing nearly-static archives - such as STORE and CITY.
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.

```
5.2. Building the attribute tree

Given a portion of interest of a source scheme and an entity (relation scheme) $F$, the attribute tree will be used in the following subsections to build the fact scheme for the fact corresponding to $F$.

The attribute tree will also be used in the following recursive procedure:

- The root of the tree corresponds to the entity $F$.
- The attribute tree will be constructed on the following rules:
  - For each vertex $v$, the corresponding attribute functionally determines all the attributes corresponding to the descendants of $v$.
  - The root corresponds to the entity (relation scheme) $F$.
  - Each vertex corresponds to an attribute - simple or compound - of the scheme.

After a portion of interest of a source scheme and an entity (relation scheme) $F$ belonging to the E/R schema, the following recursive procedure can be applied to build the attribute tree:

```
for each attribute $a$ of $F$ do
  // $a$ is the current entity, $v$ is the current vertex
  translate($a$, $v$);
end for each

where

- $translate(a, v)$ is the set of names of the attributes in $a$.
- $translate(a, v)$ returns a new vertex labeled with the set of attributes which make up the identifier of $a$.

5.2. Building the attribute tree

The following recursive procedure can be applied to build the attribute tree for $F$:

```
root=newVertex(identifier($F$));
translate($F$, root);
```

where

- $newVertex(<attributeSet>)$ returns a new vertex labeled with the concatenation of the names of the attributes in $<attributeSet>$.

The following recursive procedure can be applied to build the attribute tree for $F$:

```
translate($E$, $v$):
for each attribute $a$ of $E$ do
  // $a$ is the current entity, $v$ is the current vertex
  translate($a$, $v$);
end for each
```

where

- $translate(E, v)$ is the set of attributes $E$.
- $translate(E, v)$ returns a new vertex labeled with the set of attributes which make up the identifier of $E$.

The attribute tree will be used in the following subsections to build the fact scheme.
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);

// Renamed sale
foreach newvertex(Producto+Producto')

inserted into the attribute tree. In this case, the less significant path should be dropped.

in both directions. Thus, it may happen that two paths including opposite arcs are

A one-to-one relationship belonging to a cycle within the ER scheme can be crossed

either by removing the cycle or by crossing the arc of the cycle. The former is the most

The existence of optional relationships between attributes in a hierarchy should be

It is worth adding some further notes:

• As procedure `translate` "explores" a cycle source schema, the same entity E may

commit. Once the `translate` procedure has been called, the cycle source schema is no

• As procedure `translate` "explores" a cycle source schema, the same entity E may

• In order to avoid confusion, we prefer to label the root of the attribute tree with

• The existence of optional relationships between attributes in a hierarchy should be

• The existence of optional relationships between attributes in a hierarchy should be

• The existence of optional relationships between attributes in a hierarchy should be

• The existence of optional relationships between attributes in a hierarchy should be

It is worth adding some further notes:

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact

• As the attribute tree undergoes the next step in the methodology, the granularity of fact
• 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 (\( \max(\mathcal{E}, \mathcal{R}) > 1 \)) 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 \( n \) one-to-one relationships.

On the logical scheme:

Let \( \mathcal{E}(R) \) and \( \mathcal{F}(R,S) \) 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:

1. Let \( \mathcal{E}(R) \) and \( \mathcal{F}(R,S) \) denote the sets of the attributes of \( R \) forming, respectively, the primary key of \( R \) and a foreign key referencing \( S \).

2. For each attribute set \( A \subseteq \mathcal{E}(R) \) |
   \[ \exists S | A = \mathcal{F}(R,S) \]
   add \( A \) to vertex \( v \) and translate \( S \), \( v \)

3. For each relational scheme \( T \) |
   \[ \mathcal{F}(T,R) = \mathcal{F}(R,S) \]
   for each attribute \( b \subseteq T \) |
   \[ b \notin \mathcal{E}(R) \wedge \exists S | b \in \mathcal{F}(T,S) \]
   add child \( b \) to vertex \( v \) and translate \( T \), \( v \)

4. For each attribute set \( A \subseteq \mathcal{E}(R) \) |
   \[ \exists S | \forall \mathcal{E}(R',S) \subseteq \mathcal{F}(R,S) \]
   add child \( A \) to vertex \( v \) and translate \( R', v \)

where

- the set \( \mathcal{F}(R', v) \) with the concatenation of the names of the attributes in the current vertex \( v \) returns a new vertex labelled with the names of the attributes in the current vertex \( v \) returns a new vertex labelled with the names of the attributes in the current vertex

A compound attribute of the E/R scheme, consisting of the simple attributes

\[ a_{1}, a_{2}, \ldots, a_{m} \] is inserted in the attribute tree as a vertex with child attributes \[ a_{1}, a_{2}, \ldots, a_{m} \] if they can be inserted

without ambiguity and yield a one-to-one many relationship which

- cannot be inserted into the attribute tree of the source scheme, and

- is already stated in Section 5.1, an \( n \)-ary relationship is equivalent to \( n \) one-to-one relationships.
for each attribute set \( B \subseteq T \mid (\exists S \neq R \mid B = f_k(T, S)) \)

```c
for each \( v \in T \) do
  if \( v \) is child of \( \alpha \) then
    for each \( v' \) in \( \alpha \) do
      addChild(v, v');
    end for
  end if
end for
```

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:

\[
R(\text{key}_R, ...) \leftarrow T(\text{key}_T : R, ... \text{key}_S : S) \leftarrow S(\text{key}_S, ...)
\]

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 attribute 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 an uninteresting piece of information is classified by category; without considering the information on their type, piece of information the descendants must be preserved: for instance, one may want to translate a vertex of the attribute tree corresponding to the attribute `country` into another vertex corresponding to a group of countries.

```c
for each vertex \( v \) in the attribute tree do
  if \( v \) is child of \( \alpha \) then
    for each \( v' \) in \( \alpha \) do
      addChild(v, v');
    end for
  end if
end for
```

Let \( v \) be the vertex to be eliminated:

```c
addChildren(v, \alpha) for each \( \alpha \) in child of \( v \) do
  for each \( v' \) in \( \alpha \) do
    addChild(v, v');
  end for
end for
```
The grafting process, however, may be used to eliminate attributes whose granularity is too fine. For example, in the airline ticket sale example, the detail of purchase tickets is uninteresting and the attribute "ticket number" can be grafted. In general, grafting a child of the root vertex. The corresponding aggregation level will be lost; on the other hand, all the attributes and the corresponding aggregation level will be lost. If we drop the attribute "ticket number" and with it the set of its vertices, the decision-making process will be impaired.

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, the procedure grafting is performed as follows:

\[
\text{graft}(v) \Rightarrow \text{cnt}(t, I - \{v\}).
\]

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 airline ticket sale example, the detail of purchase tickets is uninteresting and the attribute "ticket number" can be grafted. In general, grafting a child of the root vertex, the corresponding aggregation level will be lost; on the other hand, all the attributes and the corresponding aggregation level will be lost. If we drop the attribute "ticket number" and with it the set of its vertices, the decision-making process will be impaired.

Two considerations:

1. **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 environment, it can be treated in a different way. For instance, if an attribute \( E \) has a compound identifier including the internal attributes \( a_1, \ldots, a_m \) and the external attributes \( b_1, \ldots, b_t \) (\( m, t \geq 0 \)), the algorithm outlined in Subsection 5.2 can be applied to represent these as non-dimension attributes. When the source scheme is logical, the relation schemes with compound primary key.

2. **Let \( E \) have a compound identifier including the internal attributes \( a_1, \ldots, a_m \) and the external attributes \( b_1, \ldots, b_t \). The algorithm outlined in Subsection 5.2 translates \( E \) into a vertex with children \( a_1, \ldots, a_m \). If the granularity of \( E \) must be preserved in the fact scheme, vertex \( c = a_1 + \ldots + a_m + b_1 + \ldots + b_t \) is maintained while one or more of its children may be pruned; for instance, vertex \( \text{district no. + state} \) is maintained since aggregation must be carried out at the level of single districts, while vertex \( \text{district no.} \) may be pruned since it does not express any meaningful aggregation. Otherwise, if the granularity of \( E \) is too fine, \( c \) may be grafted and some or all of its child vertices may be grafted as well. If the granularity of \( E \) is too fine, \( c \) may be pruned since it does not express any meaningful aggregation. Otherwise, if the granularity of \( E \) is too fine, \( c \) may be pruned since it does not express any meaningful aggregation.
ranges of discrete or continuous attributes. Their choice is crucial for the DW design since it determines the granularity of fact instances. Measures are defined by applying a function to numerical attributes of the attribute tree, e.g., a measure could be the sum, average, maximum, or minimum of a numerical attribute.

5. Defining measures

Measures are defined by applying a function to numerical attributes of the attribute tree, e.g., a measure could be the sum, average, maximum, or minimum of a numerical attribute.
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:

\[
\begin{align*}
\text{qty sold} &= \text{SELECT SUM(S.qty)} \\
&\text{FROM SALES S,TICKETS T} \\
&\text{WHERE S.tickNo = T.tickNo} \\
&\text{GROUP BY S.product,T.date,T.store} \\
\text{revenue} &= \text{SELECT SUM(S.qty \times S.unitPrice)} \\
&\text{FROM SALES S,TICKETS T} \\
&\text{WHERE S.tickNo = T.tickNo} \\
&\text{GROUP BY S.product,T.date,T.store} \\
\text{no. of customers} &= \text{SELECT COUNT(*)} \\
&\text{FROM SALES S,TICKETS T} \\
&\text{WHERE S.tickNo = T.tickNo} \\
&\text{GROUP BY S.product,T.date,T.store} \\
\end{align*}
\]

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. Dimensions for which aggregation should be performed should be specified; the preferred ordering of dimensions for each combination of measures/dimensions should be specified.

At this point, the aggregation function is more used for each combination of measure/dimension should be specified. In addition, the preferred ordering of dimensions for each hierarchy should be specified. It is also possible to arrange new levels of aggregation by defining ranges for numerical attributes. It is still possible to prune and enrich the quasi-tree in order to eliminate irrelevant details. The attribute tree already shows a plausible organization for hierarchies at this stage.

The last step in building the fact scheme is the definition of hierarchies on dimensions.

At this point, the aggregation functions more used for each combination of measure/dimension should be specified. In addition, the preferred ordering of dimensions for each hierarchy should be specified. It is also possible to arrange new levels of aggregation by defining ranges for numerical attributes. It is still possible to prune and enrich the quasi-tree in order to eliminate irrelevant details. The attribute tree already shows a plausible organization for hierarchies at this stage.

The last step in building the fact scheme is the definition of hierarchies on dimensions.

At this point, the aggregation functions more used for each combination of measure/dimension should be specified. In addition, the preferred ordering of dimensions for each hierarchy should be specified. It is also possible to arrange new levels of aggregation by defining ranges for numerical attributes. It is still possible to prune and enrich the quasi-tree in order to eliminate irrelevant details. The attribute tree already shows a plausible organization for hierarchies at this stage.

The last step in building the fact scheme is the definition of hierarchies on dimensions.

At this point, the aggregation functions more used for each combination of measure/dimension should be specified. In addition, the preferred ordering of dimensions for each hierarchy should be specified. It is also possible to arrange new levels of aggregation by defining ranges for numerical attributes. It is still possible to prune and enrich the quasi-tree in order to eliminate irrelevant details. The attribute tree already shows a plausible organization for hierarchies at this stage.
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

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.

6. Conclusion
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).