VISIONARY: a Viewpoint-Based Visual Language for Querying Relational Databases

FRANCESCA BENZI, DARIO MAIO, STEFANO RIZZI

DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
e-mail: {dmaio|srizzi}@deis.unibo.it

Abstract

The adoption of a visual interface can simplify the query formulation process in DBMSs by enabling naive users to interact with a friendly environment. In this work we propose a visual query language based on a diagrammatic paradigm, used for both data and query representation. The external data model is called vision and is made up of the visual primitives of concept and association. The external query model is based on the definition of a viewpoint, which is a perspective for accessing data defined dynamically by selecting a concept of primary interest. Internally, the data model is relational and the query language is SQL. An intermediate graph-based model ensures consistent mapping between the visual and the relational worlds. Our language has been implemented within a tool which can be mounted on top of any relational DBMS supporting ODBC. The system has been tested with naive users; the results of the experiment are reported and compared with those obtained with other visual languages.

Keywords and phrases: *VPL-II.B.1: Diagrammatic languages, VPL-V.B: Database languages*

1. Introduction

Visual styling for user interfaces is aimed at improving usability. In this direction the graphic presentation of the information is essential, since an effective drawing often conveys a concept more immediately and more clearly than a long written explanation [1]. Good usability results have been achieved by applications such as word processors, drawing tools and, in general, by those applications which do not require specific theoretical knowledge to be used.

Query languages for database management systems (DBMSs) often lack both intuitive understanding and visual aid; this prevents inexperienced users from taking full advantage of them. In fact, an inexperienced user willing to query a DBMS by means of a traditional query language must face a number of problems:

- *Learning*: mastering even the basic principles of the relational theory requires a lot of time and study.

- *Difficult use*: a user querying a DBMS is subjected to syntax that is rigid and hard-to-remember.
- *Poor feedback*: the user receives little feedback about the semantics of the query (s)he is formulating.
- *Little interaction*: the query-building process does not usually contemplate any form of dialogue with the user.

A visual interface can effectively support the query design process: although the difficulty of the subject cannot be completely overcome, users can interact with a friendly and helpful environment, and query formulation becomes easier.

In this paper we propose a visual query language called VISIONARY, based on a diagrammatic paradigm used for both data and query representation. In particular, the external data model is called *vision* and is made up of the visual primitives of *concept* (represented by the combination of a text and an icon) and *association* (represented, for each possible orientation, by a name and a multiplicity); the external query model is based on the definition of a viewpoint, that is, a dynamic and adaptive perspective for accessing data. The internal data model is relational, and the internal query model is SQL. The mapping between the visual and the relational worlds is established through an intermediate graph-based level [2].

VISIONARY is mainly directed to naive and occasional users, who have no experience with data models and only have a general idea about the contents of the database; for this reason, the expressive power of VISIONARY is limited to restricted join queries.

We have implemented VISIONARY within a tool which can be mounted on top of any relational DBMS supporting ODBC. The tool acquires an existing database scheme from the DBMS and assists an expert user in building a vision by allowing the proper definition of the logical and visual aspects of the different entities. Finally, a naive user can access and query the vision; the visual queries are translated into SQL queries and handed on to the DBMS, which executes them and returns the results to the visual tool. The architecture of the system and its use by expert and naive users are sketched in Figure 1.

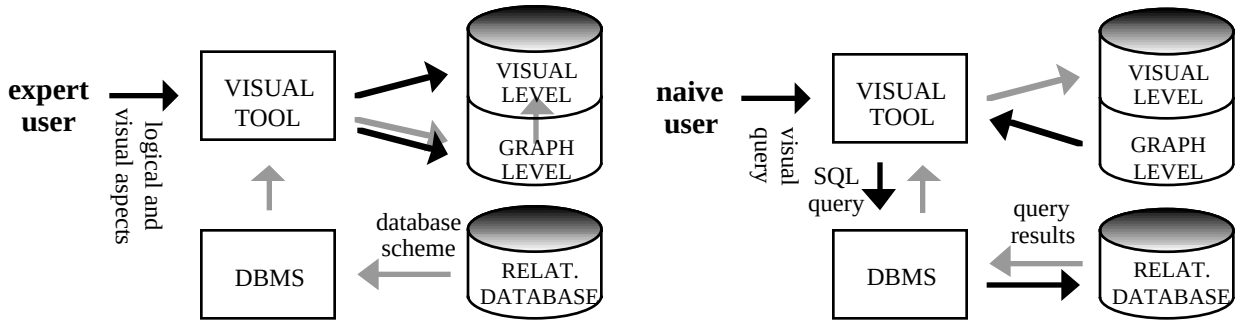

Figure 1. Expert and naive users interacting with the VISIONARY tool. Grey and black arrows denote different flows of data.

Section 2 briefly surveys some approaches to visual query languages in the literature. Section 3 introduces the working example which will be used throughout the paper. In Sections 4 and 5 we describe, respectively, the data and the query models. Section 6 gives some examples of query formulation, while Section 7 reports the results of an experiment made with naive users and compares them with those obtained with other visual languages.

2. Visual query systems in the literature

A classification of Visual Query Systems (VQSs) based only on their graphic appearance would be poor; three classification parameters have been proposed in [3]: the visual metaphor the VQS uses to represent and query the database, its expressive power and the classes of users it is intended for. A comprehensive survey of VQSs is proposed in [4].

2.1. Visual metaphor

Four visual paradigms can be followed in building a visual interface for a DBMS: *tabular*, *diagrammatic*, *iconic* and *hybrid*.

Among tabular approaches we recall QBE [5], the first successful attempt to build a graphical user interface for a relational DBMS. Data are presented as tables, and queries are formulated by compiling them; the main drawback is that a satisfactory global view of the database cannot easily be produced.

In a diagrammatic approach, simple geometric figures are connected to build a global vision of the database. Most VQSs in this category, for instance GUIDE [6], SUPER [7] and QBD* [8], follow the Entity/Relationship formalism and enable the user to formulate queries by navigating on the conceptual scheme. Some diagrammatic languages adopt a 3-D representation for data and results (see AMAZE [9]). In general, the diagrammatic metaphor generates a good global view of the data, but its dominant symbolic nature may be difficult to understand.

In the iconic approaches, queries are composed by selecting and combining icons, and by creating new ones (see for instance [10] and [11]). The iconic paradigm is very intuitive and has great expressive power. Nevertheless, the ability to distinguish one icon from the other decreases with the increase of the total number of icons; besides, icons which represent actions or abstract concepts are very difficult to design.

Hybrid paradigms attempt to overcome this problem by uniting the immediacy of icons with the expressive power of diagrams and text. A multiparadigmatic VQS is proposed in [12]; the user is provided with an adaptive interface supporting different visual representations of data and queries.

VISIONARY offers a diagrammatic global view, but uses icons to achieve better understanding of concepts; query results are represented in tabular form.

2.2. Expressive power

The expressive power of a VQS is its ability to extract consistent information from a database; in [3], its definition is based on a classification of the queries. The expressive power of most VQSs is limited to first order queries, that is, all queries which can be expressed in relational algebra. There have been some attempts to further extend the expressive power by including recursive queries [8] [13].

The expressive power of VISIONARY is limited to restricted join queries enriched with special purpose operators (aggregation functions).

2.3. Users

A VQS may be used by different kinds of users. Currently, few languages are versatile from this point of view, while most are specifically intended for a given class of users. The non-professional users of a VQS can be classified according to their experience, how frequently they use the VQS, their knowledge of the database and the predictability of their queries.

The users VISIONARY is mainly aimed at are *naive* (they generally do not need to formulate complex queries), *casual* (they do not access the database frequently) and *uninformed* (they only have a superficial knowledge of the database).

3. Working example: the "Order Database"

The Order Database models the employees, the clients, the suppliers, the products and the price-lists of a small firm. The database scheme is represented below (primary keys are underlined; foreign keys are followed by ":" and by the relation they refer to).

```
LIST(codList,areaName)
PRODUCT(codProd,description)
CURRENCY(currName,country)
QUOTES(codProd:PRODUCT,codList:LIST,currName:CURRENCY,price)
STORE(codStore,address,inArea:LIST)
PROD_IN_STORE(codProd:PRODUCT,codStore:STORE,quantity)
EMPLOYEE(codEmpl,name,salary)
VENDOR(codEmpl,supervises:LIST)
SOLD_BY(codProd:PRODUCT,codEmpl:VENDOR)
CLIENT(codCli,name,address,firmOrCustomer,belongsTo:LIST)
ORDER(codOrder,date,signedBy:VENDOR,madeBy:CLIENT,expressedIn:CURRENCY,
      amount,ifCarriedOut)
PROD_IN_ORD(codProd:PRODUCT,codOrder:ORDER,quantity)
SUPPLIER(codSupplier,name,address)
SUPPLIES(codSupplier:SUPPLIER,codProd:PRODUCT,quantity)
```

In order to better understand this simple logical scheme it may be useful to consider its conceptual representation, sketched in Figure 2 using the Entity/Relationship formalism.

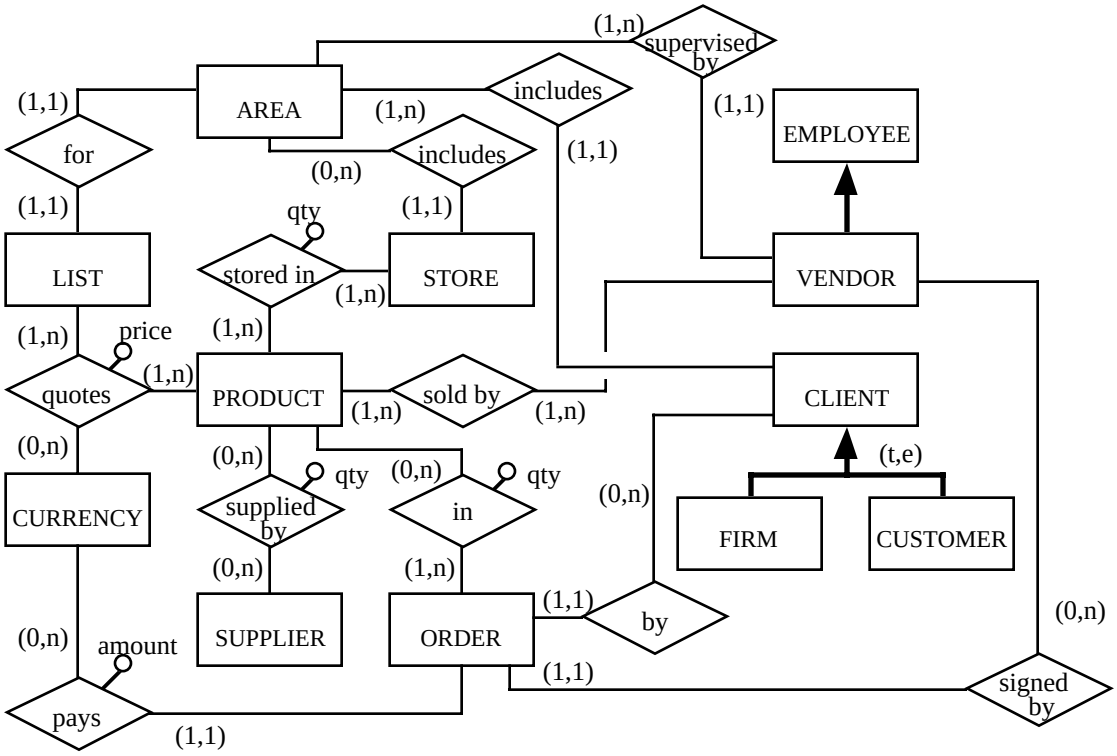

Figure 2. Entity/Relationship conceptual scheme of the Order Database.

4. The data model

VISIONARY enables users to access a relational database through an effective diagrammatic representation similar to an Entity/Relationship scheme. An intermediate graph-based level establishes the mapping between the internal data model, which is relational, and the visual metaphor adopted as the external data model.

4.1. Internal data model

A database scheme includes a set of relations and a set of integrity constraints. Each relation has a name and a set of attributes. Among integrity constraints, entity integrity and referential integrity constraints specify, respectively, the primary key and the foreign keys (if any) of each relation. The *primary key* of relation r_i is a subset of the attributes of r_i whose values exactly identify one tuple of r_i. A *foreign key* of r_i is a subset of the attributes of r_i whose values either occur as a value of the primary key of a relation r_j (not necessarily distinct from r_i) or are null.

A comprehensive explanation of the relational theory can be found in [14].

4.2. Intermediate data model

Let $\mathcal{D}$ be a relational database scheme including a set of relations R and a set of integrity constraints. Each referential integrity constraint in $\mathcal{D}$ determines a relationship between the relations involved; we call this relationship *potential link*. For instance, the foreign key `signedBy` in relation `ORDER` determines a potential link between attribute `signedBy` in `ORDER` and attribute `codEmpl` in `VENDOR`. Other potential links may be defined explicitly by the database designer when the comparison between two attributes is expected to be relevant. We call L the set of all the potential links defined on $\mathcal{D}$.

In the following we will assume, for notational simplicity but without loss of generality, that all the primary and foreign keys in $\mathcal{D}$ consist of a single attribute.

A relation $r \in R$ is defined by its name $name(r)$ and by the set of its attributes $Attr(r)$. Relation names are unique within a database scheme; attribute names are unique within a relation.

A potential link $l \in L$ is defined by its two directions, called *directed links* (see Figure 3), and by a strength. The *strength* of a potential link l, $strength(l) \in$ {'strong','weak'}, expresses its expected relevance in query formulation; it may be explicitly defined by the designer or implicitly determined by means of a default rule. For each directed link dl_i, $i = 1, 2$:

- $rel(dl_i) \in R$ and $attr(dl_i) \in Attr(rel(dl_i))$ denote, respectively, the starting relation and the starting attribute;
- $mult(dl_i) \in \{1,N\}$ denotes the multiplicity of the relationship represented by l when traversed starting from $rel(dl_i)$;
- $freq(dl_i)$ denotes the frequency with which the user has selected l so far starting from $rel(dl_i)$ when formulating his/her queries.

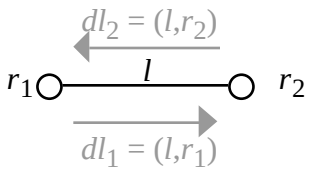

Figure 3. Directed links corresponding to a potential link.

Given a potential link l, the two corresponding opposite directed links dl_1 and dl_2 may be identified as (l,r_1) and (l,r_2), respectively, where $r_i = rel(dl_i)$. We call DL the set of all the directed links corresponding to the potential links in L; obviously, DL has double cardinality with respect to L.

An example of potential link in the Order Database is the one defined by the directed links dl_1 and dl_2 such that:

$$rel(dl_1) = \texttt{ORDER},\ attr(dl_1) = \texttt{expressedIn},\ multi(dl_1) = 1,$$
$$rel(dl_2) = \texttt{CURRENCY},\ attr(dl_2) = \texttt{codCurr},\ multi(dl_2) = N$$

We represent $\mathcal{D}$ by a non-directed graph $G = (R,L)$, called *database graph*, where each vertex corresponds to a relation and each edge to a potential link. Figure 4 shows the database graph representing the Order Database.

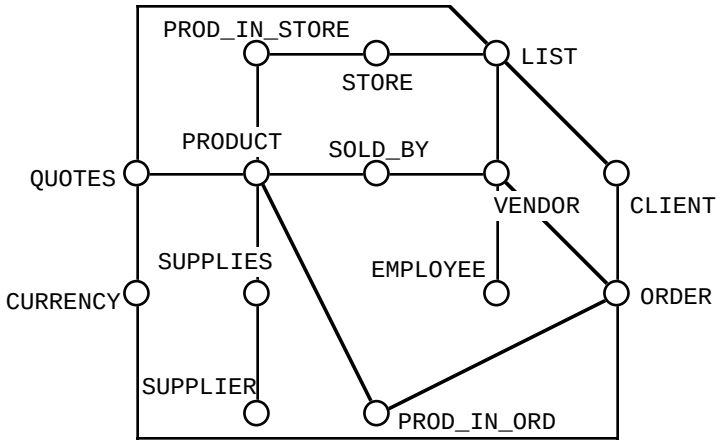

Figure 4. Database graph for the Order Database.

4.3. External data model

In this section we describe the visual metaphor adopted in VISIONARY, by defining different kinds of visual objects. Each visual object is defined at two levels: at the first one, by listing its visual components; at the second one, by establishing a mapping between the visual object and its representation at the graph level. The visual objects and their properties, as well as the graphic layout of the vision, are completely defined by the expert user.

4.3.1. Visions

A vision is a visual object aimed at conveying a high-level, clear and expressive representation of a relational database, largely independent of the structuring of data within the database schema. A vision is built by an expert user taking into account both the semantics of the database scheme and the needs of the inexperienced users who will access the database through the vision itself; if a conceptual scheme for the database is known, it may be helpful in defining the most effective representation for data. Several visions may be built on the same database, in order to address different classes of users and/or different application requirements.

Let $G = (R,L)$ be a database graph. A *vision* on G is defined as a graph $V = (C,A)$, where C is a set of concepts and A a set of associations. Concepts represent relations, while associations express relationships between two or more concepts. It should be noted that not every relation needs to be represented by a visual object, nor does every attribute of a relation represented by a

visual object need to be included in the object itself; thus, a vision may be used to hide a part of the database from the inexperienced user.

An example of a vision on the Order Database is shown in Figure 5.

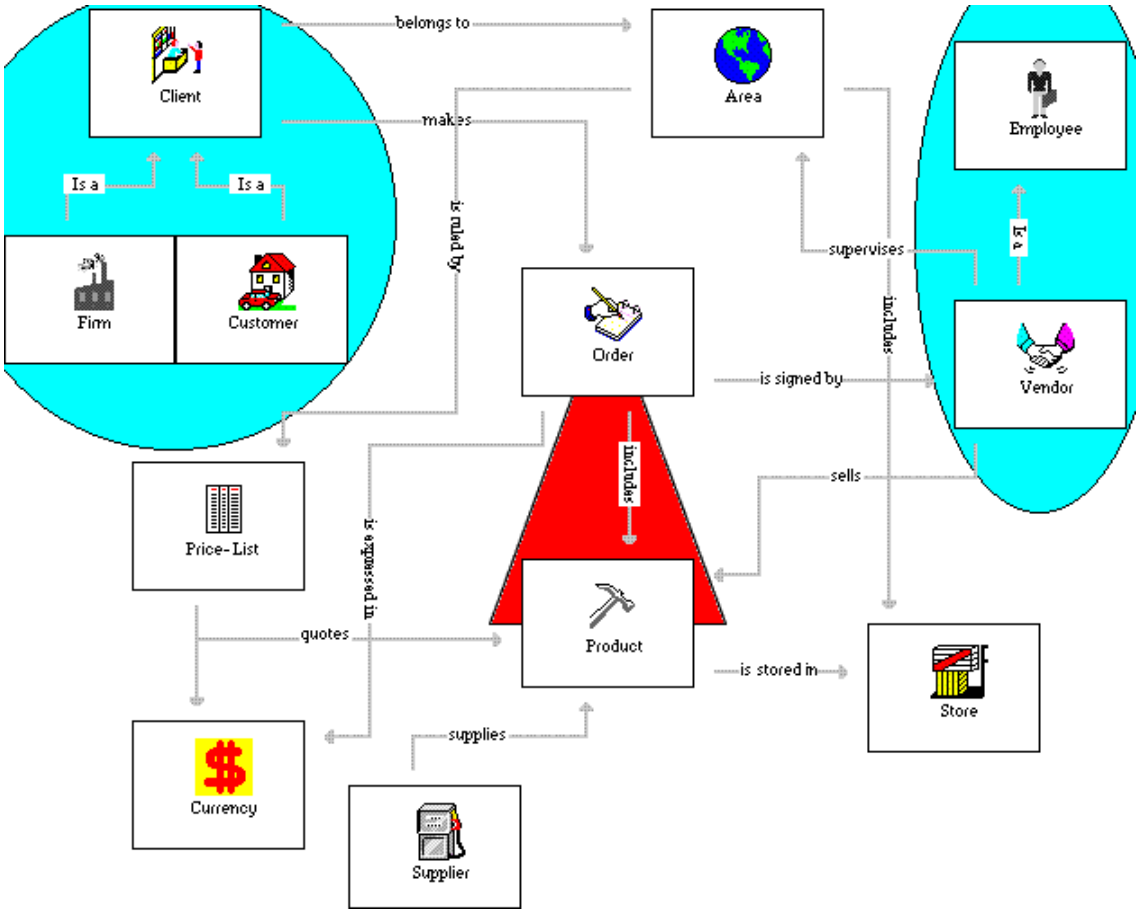

Figure 5. A vision on the Order Database.

4.3.2. Concepts

A *concept c* is defined by an icon *icon*(*c*), a name *name*(*c*), a set of attributes *Attr*(*c*) and by the mapping

$$cm : C \rightarrow R$$

where $Attr(c) \subseteq Attr(cm(c))$. Function *cm* associates each concept to a relation from which the concept's attributes are drawn.

A concept is represented visually by displaying its icon and its name (an example is *Employee* in Figure 5); the attributes of the concept may be viewed by double-clicking the icon.

In some cases a given relation *r* may be associated to different concepts, among which the attributes of *r* can be distributed in order to:

1. partition *r* vertically, with the aim of emphasizing conceptually distinct aspects. For instance, a relation `PERSON` may be associated with two concepts: one defining each person from the private point of view (attributes `name`, `address`, etc.), the other from the work point of view (attributes `salary`, `duties`, etc.).
2. partition *r* vertically and horizontally, with the aim of emphasizing two concepts related by an *is-a* relationship. This may happen when, during the logical project, in order to translate a generalization hierarchy from the Entity/Relationship scheme, the attributes of the sub-entities and those of the super-entity have been brought together into one relation. Thus, for instance, relation `CLIENT` may be associated with three concepts: one defining all clients (with their general attributes), the others defining firm and customers (each with its specific attributes), respectively.
3. partition *r* vertically, with the aim of emphasizing two concepts related by a relationship. This may happen when a one-to-one relationship in the Entity/Relationship scheme has been translated by one relation where the attributes of the two entities involved are brought together. Thus, for instance, relation `LIST` may be associated with two concepts: one defining the price-lists, the other the sale areas.

Table 1 contains the definitions of all the concepts belonging to the vision represented in Figure 5.

name(*c*)	*name*(*cm*(*c*))	*Attr*(*c*)
Price-List	`LIST`	{code}
Area	`LIST`	{areaName}
Product	`PRODUCT`	{code, description}
Currency	`CURRENCY`	{name, country}
Store	`STORE`	{address}
Employee	`EMPLOYEE`	{name, salary}
Vendor	`VENDOR`	{}
Client	`CLIENT`	{name, address}
Firm	`CLIENT`	{}
Customer	`CLIENT`	{}
Order	`ORDER`	{code, date, ifCarriedOut}
Supplier	`SUPPLIER`	{name, address}

Table 1. The concepts in the Order Database.

4.3.3. Associations

An *n*-ary *association a* is defined by a set of $n \geq 2$ *directed associations* and by a set of attributes *Attr*(*a*). For each directed association da_i, $i = 1,...n$:

- $conc(da_i) \in C$ denotes the *i*-th concept involved in *a*;
- $name(da_i)$ denotes the name given to *a* when traversed starting from $conc(da_i)$;

- $min(da_i) \in \{0,1\}$ and $max(da_i) \in \{1,N\}$ denote, respectively, the minimum and the maximum number of instances of the other $n-1$ concepts associated to one instance of $conc(da_i)$.

Given an association a, the n corresponding directed associations $da_1,...da_n$ may be identified as $(a,c_1),...(a,c_n)$, respectively, where $c_i = conc(da_i)$. We call DA the set of all the directed associations corresponding to the associations in A.

The correspondence between each directed association and one or more directed links is defined by the mapping

$$Am : DA \rightarrow 2^{DL}$$

explained in detail in the following.

Binary associations

A binary association a involves two distinct concepts c_1 and c_2. Let r_1 and r_2 be, respectively, the relations into which c_1 and c_2 map.

- Binary association a may correspond to one potential link l that has ends r_1 and r_2:

  $$Am((a,c_1)) = \{(l,r_1)\}$$
  $$Am((a,c_2)) = \{(l,r_2)\}$$

  In this case, the attributes of a may belong to one or both the relations involved:

  $$Attr(a) \subseteq Attr(r_1) \cup Attr(r_2)$$

  An example is the association between *Order* and *Currency*.
- Binary association a may correspond to two potential links l_1 and l_2 that have ends r_1 and r_z and r_2 and r_z, respectively, where r_z is a relation not associated to any concept (*ghost relation*):

  $$Am((a,c_1)) = \{(l_1,r_1),(l_2,r_z)\}$$
  $$Am((a,c_2)) = \{(l_2,r_2),(l_1,r_z)\}$$

  In this case, the attributes of a must belong to the ghost relation:

  $$Attr(a) \subseteq Attr(r_z)$$

  This form is generally used when r_z is the relation modelling the many-to-many binary relationship between r_1 and r_2 and is not considered worth being represented as a concept. An example is the association between *Product* and *Store* (the ghost relation is `PROD_IN_STORE`).
- If $r_1 \equiv r_2$, it must be

  $$Am((a,c_1)) = Am((a,c_2)) = \varnothing$$

(*empty associations*). In this case, the attributes of *a* must belong to the relation involved:

$$Attr(a) \subseteq Attr(r_1)$$

This case occurs when two or more concepts have been defined on the same relation aimed, as explained in Section 5.1.2, at partitioning the relation itself. Normally, the partitioning is accompanied by the definition of an association which does not correspond to any potential link. An example is the association between *Area* and *Price-List*.

Tables 2 to 4 contain the definitions of all the binary associations belonging to the vision represented in Figure 5; each association is described by 2 rows, one for each corresponding directed association.

name(*conc*(*da*))	*name*(*da*)	*min*(*da*)	*max*(*da*)	*Am*((*a*,*conc*(*da*)))	*Attr*(*a*)
Store	is in	1	1	{(STORE.inArea,LIST.codList)}	{}
Area	includes	0	*N*	{(LIST.codList,STORE.inArea)}	
Employee	is a	0	1	{(EMPLOYEE.codEmpl,VENDOR.codEmpl)}	{}
Vendor	is a	1	1	{(VENDOR.codEmpl,EMPLOYEE.codEmpl)}	
Vendor	supervises	1	1	{(VENDOR.supervises,LIST.codList)}	{}
Area	is supervised by	1	*N*	{(LIST.codList,VENDOR.supervises)}	
Client	belongs to	1	1	{(CLIENT.belongsTo,LIST.codList)}	{}
Area	includes	1	*N*	{(LIST.codList,CLIENT.belongsTo)}	
Order	is signed by	1	1	{(ORDER.signedBy,VENDOR.codEmpl)}	{}
Vendor	signs	0	*N*	{(VENDOR.codEmpl,ORDER.signedBy)}	
Order	is made by	1	1	{(ORDER.madeBy,CLIENT.codCli)}	{}
Client	makes	0	*N*	{(CLIENT.codCli,ORDER.madeBy)}	
Order	is expressed in	1	1	{(ORDER.expressedIn,CURRENCY.codCurr)}	{amount}
Currency	expresses	0	*N*	{(CURRENCY.codCurr,ORDER.expressedIn)}	

Table 2. The non-empty binary associations without ghost relation in the Order Database.

name(*conc*(*da*))	*name*(*da*)	*min*(*da*)	*max*(*da*)	*Am*((*a*,*conc*(*da*)))	*Attr*(*a*)
Product	is stored in	1	*N*	{(PRODUCT.codProd,PROD_IN_STORE.codProd), (PROD_IN_STORE.codStore,STORE.codStore)}	{quantity}
Store	stores	1	*N*	{(STORE.codStore,PROD_IN_STORE.codStore), (PROD_IN_STORE.codProd,PRODUCT.codProd)}	
Product	is sold by	1	*N*	{(PRODUCT.codProd,SOLD_BY.codProd), (SOLD_BY.codEmpl,VENDOR.codEmpl)}	{}
Vendor	sells	1	*N*	{(VENDOR.codEmpl,SOLD_BY.codEmpl), (SOLD_BY.codProd,PRODUCT.codProd)}	
Order	includes	1	*N*	{(ORDER.codOrder,PROD_IN_ORD.codOrder), (PROD_IN_ORD.codProd,PRODUCT.codProd)}	{quantity}
Product	belongs to	1	*N*	{(PRODUCT.codProd,PROD_IN_ORD.codProd), (PROD_IN_ORD.codOrder,ORDER.codOrder)}	
Supplier	supplies	0	*N*	{(SUPPLIER.codSupplier,SUPPLIES.codSupplier),(SUPPLIES.codProd,PRODUCT.codProd)}	{quantity}
Product	is supplied by	0	*N*	{(PRODUCT.codProd,SUPPLIES.codProd), (SUPPLIES.codSupplier,SUPPLIER.codSupplier)}	

Table 3. The binary associations with ghost relation in the Order Database.

name(*conc*(*da*))	*name*(*da*)	*min*(*da*)	*max*(*da*)	*Am*((*a*,*conc*(*da*)))	*Attr*(*a*)
Price-List	is for	1	1	{}	{}
Area	ruled by	1	1	{}	
Client	is a	0	1	{}	{}
Firm	is a	1	1	{}	
Client	is a	0	1	{}	{}
Customer	is a	1	1	{}	

Table 4. The empty binary associations in the Order Database.

Self-associations

A self-association *a* is a binary association involving the same concept *c* twice. The same holds as above, with $c_1 = c_2 = c$. A self-association may or may not have a ghost relation, but it cannot be empty.

N-ary associations

An *n*-ary association *a* involves *n* distinct concepts $c_1,...c_n$ (n≥3). Let $r_1,...r_n$ be the relations into which $c_1,...c_n$ map, respectively.

- *n*-ary association *a* always corresponds to *n* potential links $l_1,...l_n$ that have ends r_1 and r_z,... r_n and r_z, respectively, where r_z is a relation not associated to any concept (*ghost relation*):

$$Am((a,c_1)) = \{(l_1,r_1),(l_2,r_z),(l_3,r_z),...(l_n,r_z)\}$$

$$\cdots\cdots$$

$$Am((a,c_n)) = \{(l_n,r_n),(l_1,r_z),(l_2,r_z),...(l_{n-1},r_z)\}$$

(see Figure 6). The attributes of *a* must belong to the ghost relation:

$$Attr(a) \subseteq Attr(r_z)$$

This form is used when r_z is the relation modelling the many-to-many *n*-ary relationship between $r_1,...r_n$ and is not considered worth being represented as a concept. On the other hand, should the designer choose to represent the ghost relation as a concept c^*, the *n*-ary relationship would be represented by defining *n* binary associations (without the ghost relation) between c^* and $c_1,...c_n$.

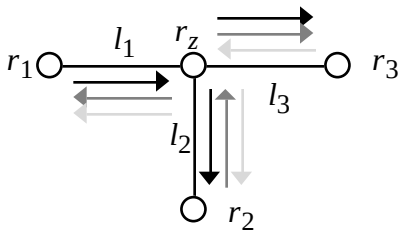

Figure 6. Three sets of directed links (in black, dark grey and light grey) corresponding to the three directed associations defining a ternary association.

Table 5 contains the definitions of the ternary association belonging to the vision in Figure 5; it is described by 3 rows, one for each direction.

name(*conc*(*da*))	*name*(*da*)	*min*(*da*)	*max*(*da*)	*Am*((*a*,*conc*(*da*)))	*Attr*(*a*)
Price-List	quotes	1	*N*	{(`LIST.codList`,`QUOTES.codList`), (`QUOTES.codProd`,`PRODUCT.codProd`), (`QUOTES.currName`,`CURRENCY.currName`)}	{price}
Product	is quoted in	1	*N*	{(`PRODUCT.codProd`,`QUOTES.codProd`), (`QUOTES.codList`,`LIST.codList`), (`QUOTES.currName`,`CURRENCY.currName`)}	
Currency	expresses	0	*N*	{(`CURRENCY.currName`,`QUOTES.currName`) (`QUOTES.codProd`,`PRODUCT.codProd`), (`QUOTES.codList`,`LIST.codList`)}	

Table 5. The ternary associations in the Order Database.

The maximum multiplicity of a directed association is determined by the multiplicity of the corresponding directed links as follows:

$$max(da) = \begin{cases} \text{maximum}\{mult(dl) \mid dl \in Am(da)\}, & \text{if } Am(da) \neq \varnothing \\ 1, & \text{if } Am(da) = \varnothing \end{cases}$$

Associations are used to convey an expressive and accurate picture of the relationships between concepts. Each association is always represented visually in one of its directions, by displaying a grey arrow connecting one end to the others, tagged with the name of the association when read in the direction of the arrow; the direction in which an association is displayed within a vision can be changed by right-clicking on the association. The attributes of the association may be viewed by double-clicking the association name; the user can further "explore" the association by left-clicking on it, which leads to visualizing the association multiplicity (Figure 7 shows some examples).

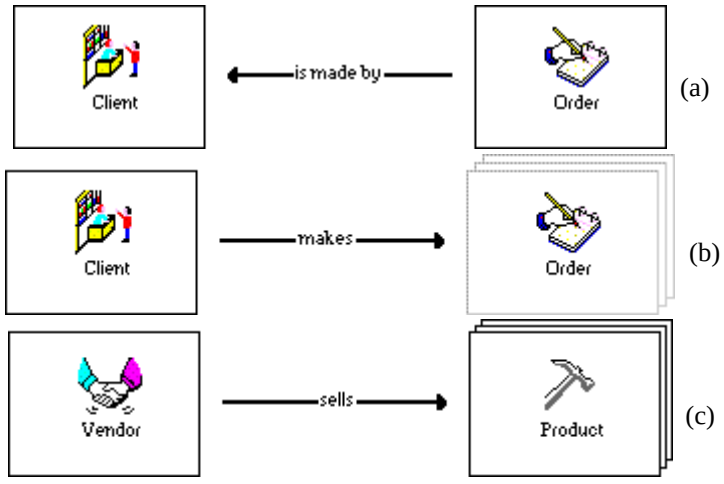

Figure 7. Visual representation of associations with multiplicity 1-1 (a), 0-n (b), 1-n (c).

Two types of binary associations are treated differently from the others within the vision: those modelling relationships with semantics PART-OF and IS-A.

PART-OF associations express the aggregation between a concept and one or more component concepts. A PART-OF association may or may not have a ghost relation, or be empty; an example is the association between `ORDER` and `PRODUCT`. Visually, PART-OF associations are emphasized by framing them within a triangle. Names are predefined in the two directions (*includes, belongs to*) and cardinalities may have any values.

IS-A associations express the specialization of a concept into one or more derived concepts. An IS-A association may either correspond to one potential link (when a hierarchy in the Entity/Relationship scheme has been translated by creating a different relation for each sub-entity and one relation for the super-entity) or be empty (when the hierarchy has been translated by creating only one schema for the super-entity, on which two or more distinct concepts have been defined); when defining an empty IS-A association, a Boolean predicate (*flag*) capable of selecting the instances of the derived concept must be specified. Examples of the two kinds are the associations between, respectively, `EMPLOYEE` and `VENDOR` and `CLIENT` and `FIRM` (with associated flag `firmOrCustomer='F'`). Visually, IS-A associations are emphasized by framing them within an ellipse. Names are predefined (*is a* in both directions); minimum and maximum cardinalities must be, respectively: 1 and 1 towards the base concept, 0 and 1 towards the derived concept.

5. The query model

Naive users access the database through the visions previously defined by expert users. The intuitiveness and expressivity of visions help them understand the database semantics, and assist them in formulating queries without knowing the relational theory.

The external query model is based on the definition of perspectives for accessing data; the internal query model is SQL.

5.1. Intermediate query model

When conceiving a query on a database, the user of a DBMS has in mind a set of attributes, belonging to one or more relations, whose values (s)he is interested in obtaining. Among these relations, one has a primary role since it embodies the point of view from which the user wants to access data in the database. For instance, suppose the user querying the Order Database is interested in knowing, for each order, the stores where the products included in the order are kept. Since the products *of* an order and the stores *of* the products *of* that order are required, the primary role is played by the entity *order*; in other words, the attributes of the products can be considered as "extended" attributes of the order, and those of the stores as "extended" attributes of the products of the orders.

In our approach to query inference the user, when formulating a query, can specify a *primary relation* (PR) which defines an *inference tree*, i.e., a perspective for accessing data. Within an

inference tree, the PR is connected to every other relation through exactly one path of relationships; thus, in a query interpreted on a tree, it is possible to reference attributes belonging to any relation without explicitly formulating the necessary joins. The inference tree for each given PR is adaptively built by privileging the query interpretations more frequently adopted by the user so far.

Choosing a PR bears some similarities with defining a root for the query tree in SUPER [7]. On the other hand, while in VISIONARY the choice of the PR leads to the automatic building of a tree on the database graph, in SUPER the choice of the root is made after the tree has been manually built by the user. In QBI [10], the user selects a concept which acts as the viewpoint for the current query and browses the set of the generalized attributes. Generalized attributes correspond to our extended attributes; while QBI provides all the different interpretations, which may be confusing to the user, VISIONARY gives one default interpretation and allows the user to modify it progressively.

5.1.1. Inference trees

Let $G = (R,L)$ be a database graph; an *inference tree* associated to the PR $r_p \in R$ is a directed subtree of G with root in r_p: $T = (R',DL')$ where R' is a multiset[a] whose elements belong to R, $DL' \subseteq DL$. Note that, since T is a tree, each of its vertices (except the root) is entered by exactly one arc. Besides, since R' is a multiset, the same relation may appear twice or more within T; in this case, the copies are discriminated by using aliases.

In our approach, a *default inference tree* is built automatically for each choice of the PR. The user may then modify this tree by means of the visual language; in particular, (s)he may add some arcs, drop some others, and duplicate some vertices. In this section we briefly explain how the default tree is built; further details can be found in [15]. Modifications of the inference tree are discussed in Section 5.3.2.

The default tree associated to the database graph G and to the PR r_p is denoted with $Dt(G,r_p) = (R',DL')$ and is represented by a spanning tree on G:

$$R' \equiv R$$

If the database graph is acyclic, exactly one spanning tree exists for each PR; it can be obtained by giving each potential link in the database graph a direction in such a way that no arc enters the PR and all other vertices are entered by exactly one arc. In this case the default tree is univocally determined, and only one interpretation is possible for each query sentence.

If, on the other hand, the database graph is cyclic, a number of different spanning trees may correspond to each PR; in order to give exactly one interpretation of each query sentence, a

[a] A *multiset* is an unordered collection of elements; it differs from a set since the same element may appear several times. We will denote multisets with double braces: {{...}}.

criterion must be used to select, for each PR, one spanning tree so that the resulting interpretations are the most reasonable, that is, those which the user most probably expects. For this purpose, we estimate the *soundness* of the possible query interpretations by considering the properties of the potential links involved; the default tree associated to a PR is then defined as the spanning tree which maximizes the soundness for the queries formulated from that PR. The three potential link properties taken into account to estimate the soundness are: the strength of the potential link (strong links are sounder than weak links), its multiplicity (*to-one* links are sounder than *to-many* links) and its frequency (frequent links are sounder than infrequent links). Details on the mathematical definition of soundness, together with the algorithm which determines the spanning tree with maximum soundness, can be found in [15].

Figure 8 shows the default inference trees associated to PRs `PRODUCT` and `CLIENT`.

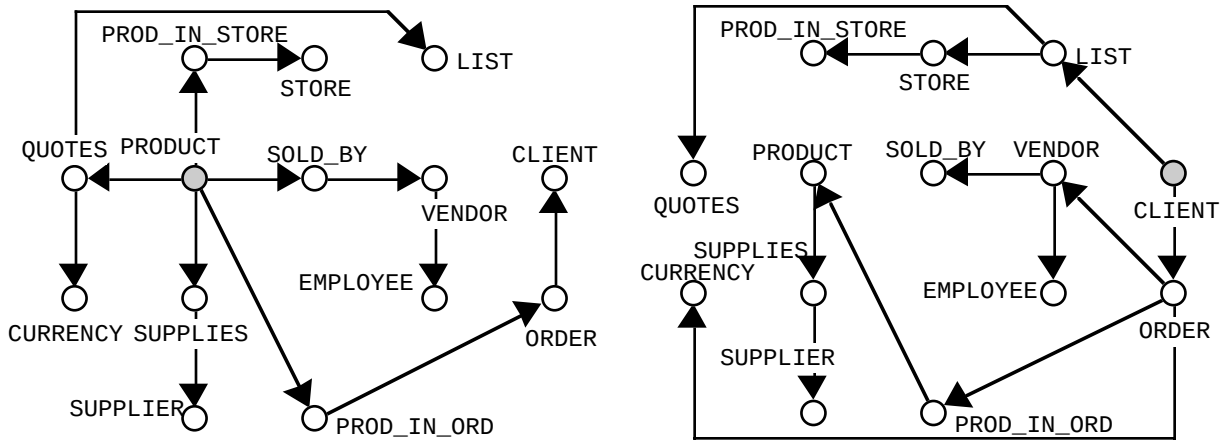

Figure 8. Default inference trees associated to PRs `PRODUCT` (left) and `CLIENT` (right).

5.1.2. Query interpretation

On the intermediate model, a query is defined as

$$q = (T, SC, JP, RA, GA, SA)$$

where $T = (R', DL')$ is an inference tree, *SC* a set of selection conditions (local Boolean predicates on the attributes of the relation), *JP* a set of additional join predicates (those not corresponding to any potential link), *RA* a set of attributes (or aggregate functions applied to attributes) to be retrieved, *GA* a set of grouping attributes and *SA* a list of sorting attributes[b]. The multiset *M* of the relations *mentioned* in *q* includes all the relations $r \in R'$ such that at least one of the attributes of *r* appears in at least one of the sets *SC*, *JP*, *RA*, *GA*, *SA*. We call *active links* for *q* the directed links belonging to the directed paths which, within *T*, connect its root r_p to every other relation

[b] All attributes are assumed to be referenced with the name of the relation scheme they belong to. If the same relation scheme appears twice or more within the inference tree, its aliases are used.

belonging to *M* (within a tree, the root is connected with each other vertex through exactly one directed path).

Query *q* is translated into an SQL formulation in which:

1. the select-list includes all the attributes in *RA*;
2. the `WHERE` clause includes:
   2.1 for each active link *dl* (if any), an equi-join predicate between the two attributes involved in *dl*;
   2.2 all the join predicates in *JP* (if any);
   2.3 all the Boolean predicates in *SC* (if any), connected by and/or operators;
3. the `GROUP BY` clause includes all the attributes in *GA* (if any);
4. the `ORDER BY` clause includes all the attributes in *SA* (if any).

For instance, consider a query formulated on the default inference tree with PR `PRODUCT` (see Figure 8) and mentioning the following attributes:

SC = {`STORE.address`='1 Wall St.',`PROD_IN_ORD.quantity`>10}, $JP = \emptyset$, $GA = \emptyset$

RA = {`PRODUCT.description`,`CLIENT.name`}, $SA = \emptyset$

The active links (briefly denoted with their two *relation.attribute* ends) are:

(`PRODUCT.codProd`,`PROD_IN_STORE.codProd`),
(`PROD_IN_STORE.codStore`,`STORE.codStore`),
(`PRODUCT.codProd`,`PROD_IN_ORD.codProd`),
(`PROD_IN_ORD.codOrder`,`ORDER.codOrder`),
(`ORDER.madeBy`,`CLIENT.codCli`)

The interpretation adopted for this query can be expressed in SQL as follows:

```
select PRODUCT.description, CLIENT.name
from   PRODUCT, PROD_IN_ORD, ORDER, CLIENT, PROD_IN_STORE, STORE
where  PRODUCT.codProd = PROD_IN_ORD.codProd
and    PROD_IN_ORD.codOrder = ORDER.codOrder
and    ORDER.madeBy = CLIENT.codCli
and    PRODUCT.codProd = PROD_IN_STORE.codProd
and    PROD_IN_STORE.codStore = STORE.codStore
and    STORE.address='1 Wall St.'
and    PROD_IN_ORD.quantity>10
```

For each product, `CLIENT` is interpreted as the clients who have ordered the product, and `STORE` as the stores where the product is kept. Thus, the query returns the products stored at the address '1 Wall St.' and the clients who ordered more than 10 pieces of them.

It should be noted that, owing to the query model adopted, formulating joins in VISIONARY typically requires defining the corresponding potential links. This forces the expert user to predict all the reasonable possibilities for joining relations while designing the database graph underlying the vision; on the other hand we assume that, even if supported by visual facilities, a naive user would hardly be capable of understanding the meaning of a join and of formulating it explicitly.

5.1.3. Query inference in the literature

Several approaches for simplifying query formulation can be found in the literature. Among those based on the building of derived relations, we mention relational views and the universal relation.

Relational views are derived relations defined by the user in terms of one or more physical relations or other views, and show advantages both in terms of logical independence and data security. Query formulation on an inference tree is not equivalent to query formulation on a relational view; in fact:

- Defining a relational view requires several joins to be explicitly formulated, while defining a default inference tree only requires choosing a PR.
- One default inference tree for each relation in the database scheme is made available; a different relational view should be written for each viewpoint.
- Adopting an inference tree may not be equivalent to defining one relational view. Consider for instance the association between clients and orders, which is optional. When a query is formulated with PR `CLIENT`, the clients who did not make any orders will appear in the resulting relation if `ORDER` is not mentioned, but will not if `ORDER` is mentioned. This behaviour can be reproduced only by creating two distinct relational views.
- If the database scheme is changed (a relation is added, dropped or modified), all the relational views involved must be rewritten, while all the default inference trees will be automatically generated based on the new scheme.

In the universal relation, query inference is approached by building a view which combines all the relations in the database through natural joins [16]. On the other hand, the universal relation calls for requirements that are not always satisfied in practical applications [17]. A basic assumption is that each attribute plays only one role, so that an attribute like "address" can only stand for the address of either the supplier, the department or the employee; the database designer is thus forced to differentiate names of attributes which are defined on the same domain but play different roles. Besides, the universal relation generates a fixed sight of the database, on whose structure the user cannot intervene. Our approach does not require the attributes to be unique; besides, distinct inference trees are created for the different PRs.

In [18], ambiguous sentences of the query language are interpreted by determining, on the database graph, the minimum directed cost Steiner tree; the cost of a query depends only on the

cardinalities of the relationships involved. In [19], query disambiguation is carried out by considering the *relatedness* of the relations involved and the existence of directed paths between them. In [20], disambiguation is carried out by choosing the interpretation which contains fewer *similarity arcs* (arcs connecting attributes defined on the same domain) and, possibly, by starting a clarification dialogue with the user. All these approaches differ from ours, since they do not consider the possibility of accessing data through multiple perspectives.

5.2. External query model

In the same way as a vision is the visual representation of a database graph, a viewpoint is the visual representation of an inference tree. A default viewpoint is automatically built by selecting a primary concept, and may then be edited visually by the user.

5.2.1. Viewpoints

Given a vision V and a *primary concept* c_p, a *viewpoint* is a directed subtree of V rooted in c_p: $P = (C',DA')$, where C' is a multiset whose elements belong to C, and $DA' \subseteq DA$.

The definition of a viewpoint partitions the associations of a vision into enabled and disabled. We call *enabled* all the associations $a \in A$ such that $\exists\ c \in C' \mid (a,c) \in DA'$ (one of the directed associations corresponding to a is included in the viewpoint); the others are *disabled.*

A viewpoint on a vision is represented visually by

1. highlighting the primary concept;
2. orientating each enabled association according to the direction in which it appears in DA';
3. drawing in black each enabled association;
4. drawing in grey each disabled association.

A viewpoint $P = (C',DA')$ determines univocally on the database graph an inference tree $T(P) = (R',DL')$ whose vertices are all the relations corresponding to the concepts in the viewpoint and all the ghost relations of the associations enabled, and whose arcs are all the directed links corresponding to the directed associations enabled:

$$R' = \{\{r \in R \mid r = cm(c),\ c \in C'\}\} \cup \{r \in R \mid (\exists\ da \in DA' \mid r \text{ is the ghost relation for } da)\}$$

$$DL' = \{dl \in DL \mid (\exists\ da \in DA' \mid dl \in Am(da))\}$$

Duplicate concepts in P determine duplicate vertices in $T(P)$. If P includes two distinct concepts c_1 and c_2 corresponding to the same relation r, two cases may occur:

1. If the empty association between c_1 and c_2 is enabled, r appears in $T(P)$ only once; in fact, duplicating r would require the formulation of a useless self-join.

2. If the empty association between c_1 and c_2 is disabled, r is duplicated within $T(P)$. In this case, the user is willing to access c_1 and c_2 separately, hence, r must appear twice within the inference tree.

Similarly, an inference tree $T = (R', DL')$ determines univocally a viewpoint $P(T) = (C', DA')$ such that:

$$C' = \{\{c \in C \mid (\exists\, r \in R' \mid r = cm(c))\}\}$$
$$DA' = \{da \in DA \mid \forall dl \in Am(da)\ (dl \in DL')\}$$

Note that an association is enabled only if all the corresponding directed links appear within the inference tree.

Figure 9 shows the viewpoint corresponding to the default inference tree with PR `PRODUCT` in Figure 8.

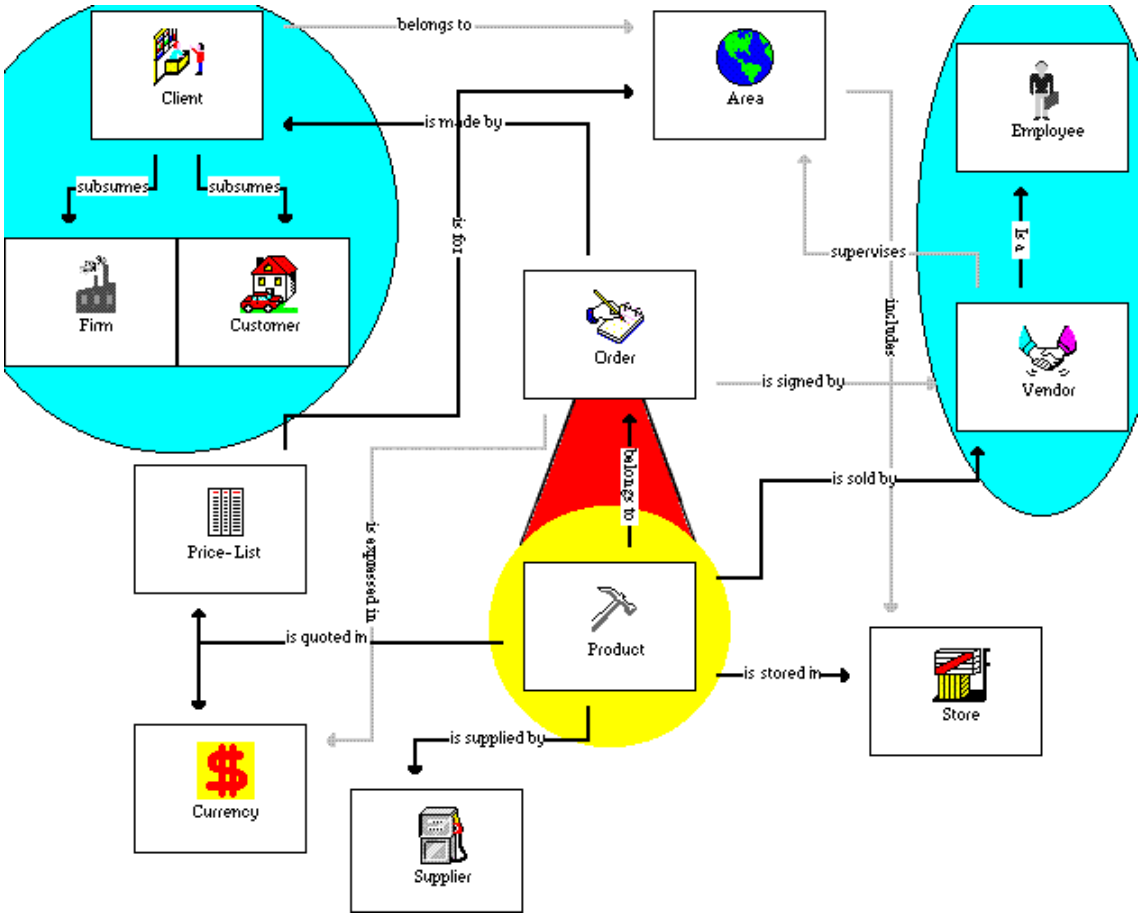

Figure 9. Viewpoint corresponding to the default inference tree with PR `PRODUCT`.

5.2.2. Visual formulation of queries

A visual query is defined as

$$q' = (P, SC', JP', RA', GA', SA')$$

where P is a viewpoint, SC' is a set of selection conditions, JP' is a set of additional join predicates, RA' is a set of attributes to be retrieved, GA' is a set of grouping attributes and SA' is an ordered list of sorting attributes.

The formulation of a visual query consists of five steps:

1. Choose a primary concept (implicit formulation of default joins).
2. Edit the viewpoint (implicit formulation of overridden joins - optional).
3. Choose attributes to be retrieved (projection).
4. Formulate selections on attributes (selection - optional).
5. Order and/or group the results (optional).

The order in which the five steps are presented is essentially conceptual; in fact, steps 2. to 5. may be interleaved (but step #1 must be carried out first).

The first two steps are aimed at building P, that is, at determining an inference tree in order to define which joins will be *potentially* part of the query; these steps are sketched in Figure 10. The remaining steps, besides their specific functions, all go towards determining a multiset of concepts "mentioned" within the query, which includes all the concepts $c \in C$ such that at least one of the attributes of c appears in at least one of the sets SC', JP', RA', GA', SA'. The associations belonging to the paths connecting the primary concept to each concept mentioned are called *active associations* and displayed in red within the viewpoint, to emphasize the fact that they determine which joins will be actually formulated for the current query.

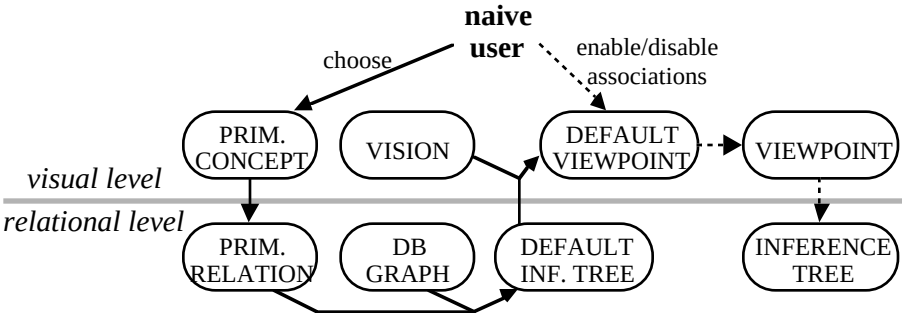

Figure 10. Query formulation in VISIONARY.

Query formulation may be carried out in "preview mode": in this case the inexperienced user can see, at each step, the results of the query (s)he is formulating and verify its correctness. Since execution of a complex query on a large database may take a long time, preview mode can be disabled in order to avoid slowing down the formulation phase too much.

In the following we describe in detail the five formulation steps.

Choosing the primary concept

This step, which starts the query formulation session, is executed visually by right-clicking on a concept c_p, whose corresponding relation $r_p = cm(c_p)$ becomes the PR. The default inference tree associated to r_p, $Dt(G,r_p)$, is calculated and the corresponding viewpoint, $P(Dt(G,r_p))$, is displayed (see Figure 10, solid arrows).

The associations enabled within the viewpoint determine the interpretation given to each concept in the vision. Consider for instance the viewpoint in Figure 9, obtained by choosing *Product* as the primary concept. The default interpretations given to concepts are, for each product P:

the vendors who sell P;	the orders made for P;
the stores stocking P;	the price-lists where P is quoted;
the clients who made orders for P;	the currencies in which P is quoted;
the suppliers of P;	the areas of the price-lists where P is quoted.

Editing the viewpoint

This step must be executed if the enabled associations contained in the default viewpoint are not those the user is interested in. Suppose that the user wants to know, for each product, the orders signed by the vendors who sell the product. In the default interpretation proposed by the system with primary concept *Product*, *Order* denotes the orders made for a product; thus, the user must disable the association *belongs to* from *Product* to *Order* and enable the association *signs* from *Vendor* to *Order*. This action changes the interpretation given to queries by forcing the formulation of paths of joins different from those provided by the default viewpoint.

Within viewpoint *P*, clicking on an enabled association leads to disabling it, that is, to removing the corresponding directed association from *P*. Clicking on a disabled association leads to enabling it, that is, to adding a directed association to *P* (see Figure 10, dashed arrows); the direction is the one currently displayed, which can be changed with a right-click.

Enabling a disabled association may lead to an attempt to form a cycle, that is, to having two enabled associations enter the same concept *c*. This happens in general when the user is interested in considering contemporarily different interpretations of c. In this case, in order to make the viewpoint remain a tree, concept *c* is duplicated within the viewpoint. Suppose the user wants to retrieve, for each product, the vendors who sell the product and those who signed the orders for the product. If, with primary concept *Product*, association *is signed by* is enabled without disabling association *is sold by*, concept *Vendor* is duplicated. Thus, the user may access contemporarily both interpretations of *Vendor* (see Figure 11). If the user is interested in the areas of the vendors who have signed the orders, (s)he can drag association *supervises* on the instance of *Vendor* corresponding to this interpretation. The association is graphically duplicated, and the

user can enable one or both interpretations; in the latter case, also concept *Area* is duplicated. Another example of concept duplication can be found in Section 6 (example #3).

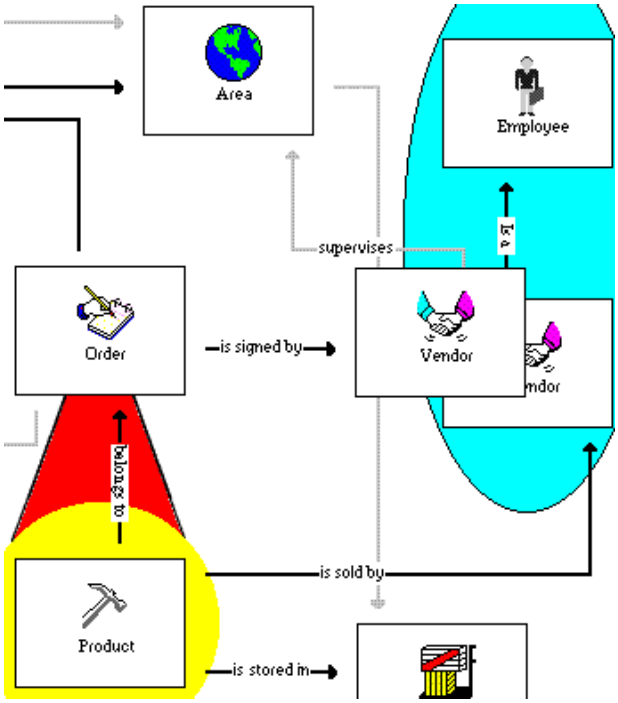

Figure 11. Duplicating concept *Vendor*.

A viewpoint is inherently acyclic; on the other hand, most useful cyclic queries can be formulated simply by writing a natural join between two instances of a duplicate concept. This can be done graphically by dragging and dropping one instance of the duplicate concept on the other, and leads to adding the natural join between the two instances to *JP'*.

Choosing the attributes to be retrieved

This step is carried out by double-clicking on concepts and associations and by dragging one or more attributes inside the *projection window*. Every attribute selected is added to *RA'*.

Aggregate functions (*sum*, *max*, *min*, *average*) may be picked up from a list to be applied to numerical attributes; function *count* may be applied to numerical and non-numerical attributes.

The user may want to mention a concept even without retrieving any of its attributes. For instance (s)he may want to retrieve, for each product, the names of the customers who made orders for it. Attribute `name` belongs to concept *Client*; thus, the association between *Client* and *Customer* must be explicitly activated. This can be done by dragging the whole concept inside the projection window, and leads to adding a dummy attribute to *RA'*.

Formulating selections on the attributes

A selection condition can be formulated by double-clicking on a concept or an association, dragging an attribute within the *selection window* and writing a Boolean predicate involving an

operator (picked from a list including all the standard SQL operators) and one or more values (typed by the user). Every condition formulated is added to *SC'*.

Grouping and sorting

The last step may be performed on a *group-and-sort window* showing a preview of the results in tabular form. When choosing to order or group data according to a specific attribute, the user will see the results of his/her action on a sample set of data. Grouping and sorting attributes are added, respectively, to *GA'* and *SA'*.

5.2.3. Query interpretation

Query interpretation is based on the associations which are active within the current viewpoint *P* at the time the query is executed; it is carried out by mapping from the visual level to the graph level.

Given a visual query $q' = (P,SC',JP',RA',GA',SA')$, the corresponding graph-level query is $q = (T(P),SC,JP,RA,GA,SA)$; the sets in q are obtained from the corresponding ones in q' by mapping each concept/association attribute into the corresponding relation attribute. The SQL interpretation of q is then built on the inference tree $T(P)$ as shown in Section 4.3, and handed on to the DBMS. Once the query has been executed, its results are shown to the user in tabular form.

Two particular situations which may occur deserve further attention:

- An enabled empty association does not correspond to any link in the inference tree. However, if an empty association has IS-A semantics, activating it within the viewpoint leads to adding the corresponding flag to the set of selection conditions *SC'*. This corresponds to selecting only the instances of the super-concept which are also instances of the sub-concept.
- Editing the viewpoint may lead to splitting it into two or more trees. Consider for instance the default viewpoint *Product*, and suppose that association *belongs to* from *Product* to *Order* is disabled. In this case, the resulting viewpoint is non-connected. The inference tree is still generated as shown in Section 5.2, and turns out to be non-connected, too. Within a non-connected inference tree, queries are interpreted by formulating an equi-join for each directed link belonging to the paths which, within each connected sub-tree, connect the root to each other relation mentioned; from a conceptual point of view, since no joins connecting the different sub-trees are provided, the query returns the Cartesian product between the interpretations given to the connected portions of the viewpoint.

6. Examples

1. *For each product supplied by John, retrieve its description and the address of the stores where it is kept.*

This query is formulated visually by choosing *Product* as the primary concept; the default viewpoint need not be edited. The projection is carried out by double-clicking on *Product* and *Store*, and selecting attributes *description* and *address*, respectively. The selection is carried out by formulating predicate *name equal John* on *Supplier*.

2. *Retrieve the names and areas of the vendors who signed the orders dated 1996 for client Marylin.*

This query is formulated visually by choosing *Client* as the primary concept. According to the default viewpoint shown in Figure 12, the interpretation given to concept *Area* is the area to which the client belongs. Since the interpretation required is "the areas supervised by the vendors who signed the orders", the user must edit the viewpoint by disabling the association *belongs to* and enabling the association *supervises*.

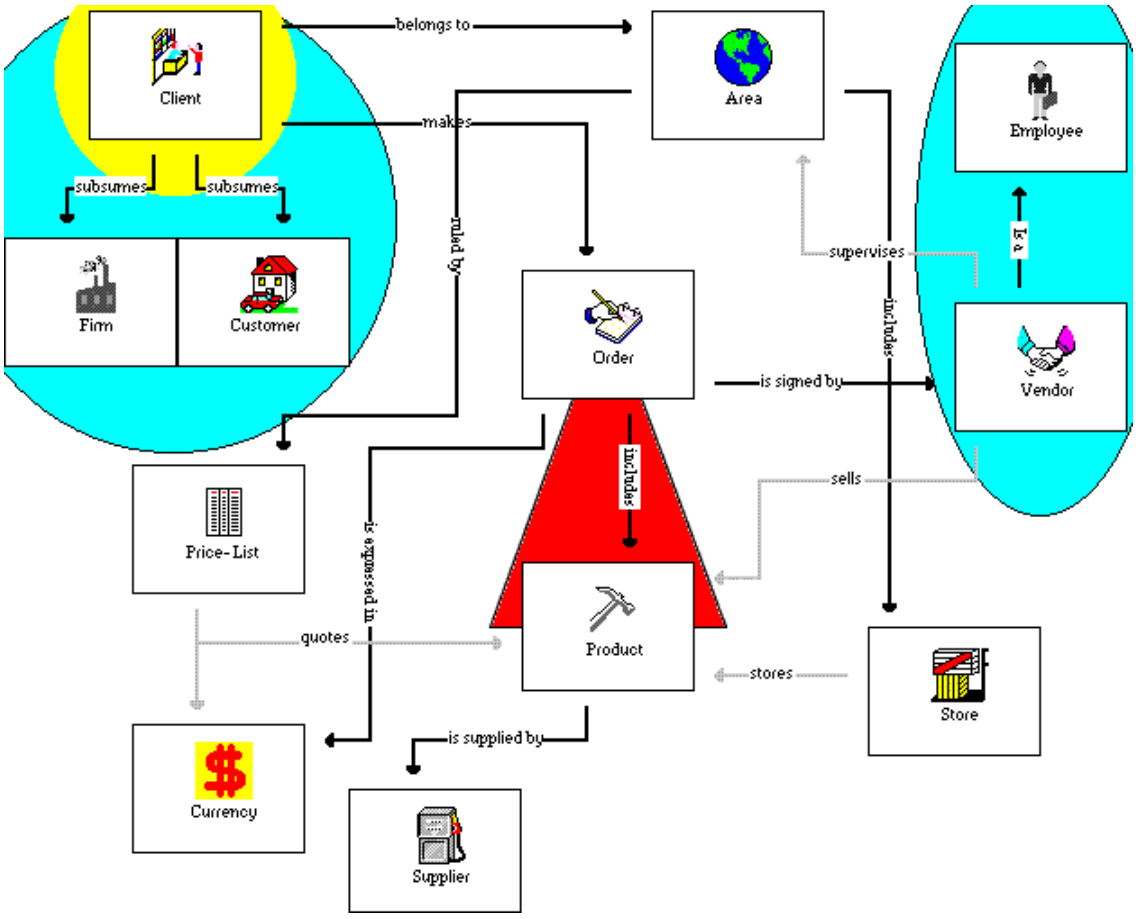

Figure 12. Default viewpoint with primary concept *Client*.

3. *For each client belonging to the area supervised by vendor Lewis, retrieve the name of the client and that of the vendors who signed the orders made by the client for at least 100 hammers.*

This query may be formulated by choosing *Client* as the primary concept. The concept *Vendor* is used twice, with different interpretations: the vendor signing the orders made by the client (this interpretation is the one implicitly given within the default viewpoint), and the vendor supervising the area of the client. If the association *is supervised by* is enabled (without disabling the association *is signed by*), the concept *Vendor* is duplicated. Thus, the projection is carried out by choosing attribute *name* from *Client* and from *Vendor* (the one associated to *Order*); the selection is carried out by formulating the predicates *name equal Lewis* on *Vendor* (the one associated to *Area*), *description equal hammer* on *Product* and *quantity greater or equal 100* on *includes*.

4. *Retrieve the names of the clients who made orders signed by the same vendor supervising their area.*

This is a cyclic query. Like query #3, it can be expressed by duplicating concept *Vendor*; the join necessary to create the cycle is formulated by dragging one instance of *Vendor* on the other.

7. Experimental tests

Query formulation in VISIONARY shares some ideas with QBD* [8]: in fact, it is carried out by selecting nodes and arcs from a graph-like representation of the database. QBD* enables formulation of recursive queries, which at present cannot be formulated with VISIONARY. On the other hand, the selection of a subscheme and the choice of the paths of joins in VISIONARY can in most cases be made implicitly by selecting a viewpoint. VISIONARY also presents some similarities with QBI [10], an iconic language which builds completely encapsulated objects by allowing all the attributes in the scheme to be viewed as generalized attributes of a single concept.

In [21], an experiment aimed at comparing QBI and QBD* in terms of their usability is described. The experiment consists in proposing six queries with different degrees of complexity to a set of users after briefly training them on the use of the visual language. Users are distinguished into skilled and unskilled. Queries are formulated on a small database describing the university domain and are classified according to their *semantic distance* (essentially a function of the length of the path of joins and of the number of printable attributes crossed) and on whether they are cyclic or not. Usability is measured by both the time needed to formulate each query and the accuracy in query completion.

We have reproduced the same experiment with VISIONARY; in this section we report and comment on the results. In our experiment, unskilled users were first-year Computer Science students who had no knowledge of information sytems and query languages but were familiar with Windows environments; skilled users were fourth-year Computer Science students. The training session lasted about 20 minutes for each user and was carried out on the Order database. Accuracy was evaluated qualitatively by recording if the primary concept was chosen correctly and if useless

associations were enabled. Tables 6 and 7 summarize the results obtained with VISIONARY and those obtained with QBD* and QBI, reported in [21].

The analysis of variance (ANOVA) technique yielded significant difference on the average formulation time between the three systems ($F(2,45)=6.28>f_{0.95,2,45}=3.20$); the results in Table 7 (last row) suggest that the users spent on the average less time formulating queries using VISIONARY. As to skill level, ANOVA yielded significant difference only for skilled users ($F(2,21)=11.19>f_{0.95,2,21}=3.47$); the results in Table 7 (first row) suggest that, for skilled users, the VISIONARY approach works better than the others.

		QBD*		QBI		VISIONARY	
		time	accur.	time	accur.	time	accur.
Query	Q1	80	*exc.*	134	*exc.*	58	*exc.*
	Q2	84	*fair*	27	*exc.*	57	*exc.*
	Q3	111	*exc.*	156	*exc.*	83	*good*
	Q4	179	*exc.*	599	*fair*	229	*fair*
	Q5	187	*fair*	55	*exc.*	76	*good*
	Q6	134	*good*	298	*fair*	193	*fair*

Table 6. Formulation time (in seconds) and accuracy for 6 queries with QBD*, QBI and VISIONARY. Each language was tested with 8 skilled and 8 unskilled users.

		QBD*	QBI	VISIONARY
Users	skilled	104	218	91
	unskilled	154	205	141
	all	129	211	116

Table 7. Formulation time as a function of skill level.

The tests pointed out that the hardest operation for unskilled users is viewpoint editing, especially when concept duplication is needed; in particular, some users forgot to disable useless associations and were confused by the resultant proliferation of concepts on the screen. On the other hand, skilled users are more familiar with graph-like representations and they better understood the semantics of enabling/disabling associations on the screen. We believe that the main reason why the formulation time with VISIONARY is slightly better than with QBD* is that, while with the latter users must select the useful paths of associations by themselves, with the former they can, in most cases, formulate their query on the tree automatically built by the system.

8. Conclusions

In this work we have described a visual query language based on an iconic-diagrammatic paradigm. The user perceives the database through a metaphor called vision, which is made up of concepts and associations. Queries are formulated by choosing a primary concept, deciding the attributes to be retrieved and expressing selection predicates. If the interpretation given to a query is not the one the user had in mind, the user can force a different interpretation by disabling some associations and enabling others.

Currently we are working on improving the visual metaphor which models the database and on extending the expressive power of the visual query language. As to the first issue, we will study how to give an effective visual representation to time as involved in concepts or associations. As to expressive power, in [22] we have shown an SQL extension where multiple viewpoints could be adopted for query formulation; in VISIONARY, the user selecting two or more viewpoints within a single query will be abled to choose between as many different interpretations for the concepts included in the vision. Finally, we are investigating the possibility of using VISIONARY as a design tool for databases.

References

[1] E. Tufte (1983) *The visual display of quantitative information*. Graphic Press.

[2] F. Benzi, D. Maio & S. Rizzi (1996) VISIONARY: a visual query language based on the user viewpoint approach. In: *Electronic Series Workshop in Computing*, Springer, London, `http://www.springer.co.uk/eWiC/Workshops/IDS3.html`.

[3] C. Batini, T. Catarci, M.F. Costabile & S. Levialdi (1992) Visual query systems: a taxonomy. In: *Visual Database System II*, Elsevier Science Publishers B.V, North-Holland, pp. 153-168.

[4] T. Catarci, M.F. Costabile, S. Levialdi & C. Batini (1997) Visual query systems for databases: a survey. *Journal of Visual Languages and Computing* **8**, 215-260.

[5] M.M. Zloof (1975) Query-by-example. In: *Proceedings AFIPS Conference*, National Computer Conference, 44, pp. 431-438.

[6] H.K.T. Wong & I. Kuo (1982) GUIDE: Graphical User Interface for Database Exploration. In: *Proceedings 8th VLDB Conference*, Mexico City, pp. 22-31.

[7] Y. Dennebouy, M. Andersson, A. Auddino, Y. Dudont, E. Fontana, M. Gentile & S. Spaccapietra (1995) SUPER: visual interfaces for object + relationship data models. *Journal of Visual Languages and Computing* **5**, 73-99.

[8] M. Angelaccio, T. Catarci & G. Santucci (1990) QBD*: a graphical query language with recursion. *IEEE Transactions on Software Engineering* **16**(10), 1150-1163.

[9] J. Boyle, S. Leishman, J. Fothergill & P. Gray (1994) Design of a visual language for a database. Technical Report University of Aberdeen.

[10] A. Massari & P.K. Chrysanthis (1995) Visual query of completely incapsulated objects. In: *Proceedings 5th International Workshop on Research Issues on Data Engineering*, Taipei, Taiwan, pp. 18-25.

[11] A. Del Bimbo, M. Campanei & P. Nesi (1992) A 3D visual environment for querying image databases. In: *Proceedings International Workshop of Advanced Visual Interfaces*, World Scientific Publishing Co. Ltd., Singapore, pp. 12-25.

[12] T. Catarci, S. Chang, M.F. Costabile, S. Levialdi & G. Santucci (1996) A graph-based framework for multiparadigmatic visual access to databases. *IEEE Transactions on Knowledge and Data Engineering* **8**(3), 455-475.

[13] M. Consens & A.O. Mendelzon (1990) Graphlog: a visual formalism for real life recursion. In: *Proceedings ACM Symposium on Principles of Database Systems*, pp. 404-416.

[14] E.F. Codd (1970) A relational model of data for large shared data banks. *Communications of the ACM* **13**(6), 377-387.

[15] G. Bellavia, D. Maio & S. Rizzi (1994) Minimizing the cost of query formulation through user viewpoint Relations. In: *Proceedings Secondo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati*, Rimini, Italy, pp. 141-159.

[16] R. Fagin, A.O. Mendelzon and J.D. Ullman (1982) A simplified universal relation assumption and its properties, *ACM Trans. Database Syst.* **7**(3), 343-360.

[17] W. Kent (1981) Consequences of assuming a universal relation, *ACM Trans. Database Syst.* **6**(4), 539-556.

[18] J.A. Wald and P.G. Sorenson (1984) Resolving the query inference problem using Steiner trees, *ACM Trans. Database Syst.* **9**(3), 348-368.

[19] E. Sciore (1994) Query abbreviation in the entity-relationship data model, *Information Syst.* **19**(6), 491-511.

[20] A. Motro (1986) Constructing queries from tokens. In: *Proc. ACM SIGMOD Int. Conf. Management of Data*, Washington D.C., pp. 120-131.

[21] A.N. Badre, T. Catarci, A. Massari & G. Santucci (1996) Comparative ease of use of a diagrammatic vs. an iconic query language. In: *Electronic Series Workshop in Computing*, Springer, London, `http://www.springer.co.uk/eWiC/Workshops/IDS3.html`.

[22] G. Bellavia, D. Maio & S. Rizzi (1995) An SQL extension supporting user viewpoints. In: *Proceedings 6th International Conference on Database and Expert Systems Applications*, London, pp. 334-343.