A Hypertabular Visualizer of Query Results

Giuseppe Santucci

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria, 113, I-00185 Roma, Italy
santucci@dis.uniroma1.it

Laura Tarantino

Dipartimento di Ingegneria Elettrica
Università degli Studi dell'Aquila
Poggio di Roio, I-67040 L'Aquila, Italy
tarantino@vaxaq.cc.univaq.it

Abstract

A visual database front-end for handling query result presentation and exploration is presented. The system is based on a formal interaction structure called hypertable, and can be seamlessly integrated with any visual semantic query language adopting a data model that supports the notion of relationship cardinality. We aim at overcoming two kinds of problems in the interaction with tabular displays: (1) application independent problems, pertaining to the mapping from logical tables to graphical windows, and (2) application dependent problems, related to the content of tables as query results, such as repetitions of values and contiguity of unrelated data. The first kind of problem is addressed by the interaction structure, while the latter is faced by the navigation paradigm designed for the analysis of the query result.

1 Introduction

Pure relational databases are widely diffused and accessed by diverse classes of users. Much work has been done for devising user interfaces able to significantly reduce the user's effort for selecting, digesting and assimilating information. Notwithstanding that the ultimate goal of the information consumer is the result of the seeking process and not the query formulation per se, most DBMS front-ends provide effective (visual) support for the information *selection* while lacking adequate support for information *digestion* and *assimilation* (with notable exceptions, as [1]).

Suitable data set organizers are necessary to help users make sense of retrieved information. Organizers have to make patterns visible, capture relevant regularities, and allow the construction of new information patterns from old [4]. A static and a dynamic aspect can be singled out in such tools: the *static* aspect refers to the visualization techniques, while the *dynamic* aspect refers to the modalities offered to interact with the visual structures.

Tables are widely used for the visualization of relational query results for their effectiveness: they are a very familiar data organization, and require low cost graphical representations. Tables may suffer from a number of problems that sensibly decrease their efficacy. A first kind of problems is application dependent and relate to the content of tables as query results, while a second type is application independent and pertains to the mapping from logical tables to graphical windows. In the remainder of this section, we discuss these two classes of problems along with possible solutions for overcoming them by acting mainly on the dynamic aspects, in contrast with other approaches that are mainly focussed on the static ones (e.g., [6, 11]).

With regards to the first class of problems, we observe that user requests often require the join of two or more relational tables, which, in general, may result in a table that is not in third normal form, thus exhibiting a tedious repetitions of values. The readability of the table is also decreased by the contiguity of unrelated data, such as attribute values gathered from different tables. We observe that when the DBMS is equipped with a query interface based on a semantic model, additional information stored into the semantic representation of data can be used to define output presentations richer than flat sets of tuples.

As to the mapping from logical tables to graphical windows, the approach used when the table dimension exceeds the dimension of the window is to regard the window as a view panning over the table. This is equivalent to having a continuum of adjacent sub-tables, successively disclosed by means of horizontal and vertical scrolling steps (by acting on scroll bars). The main drawback is the fact that a view may clip out relevant portion of the structure (e.g., attributes that identify the rest of the values), giving raise to not meaningful sub-tables. The cause for this undesired *loss-of-context* effect can be found in the nature of the interaction provided by the scroll bars, purely syntactic because originated in the windowing system and not in the specific application[1]. The application must hence have as much control as possible over the interaction, without relying on system-specific tools.

We also recall that cognitive studies on *display density* show that it is not effective to force too much information into one (overloaded) display. It is preferable to map the

[1] Some tools (e.g., spreadsheets) provide a partial solution to this problem by allowing to split the sets of columns and rows into independent scrollable subgroups, leaving on the user the burden of identifying the portions of the table that have to remain displayed for obtaining meaningful sub-tables.

total volume of information onto a set of smaller displays, each containing a closely related subset of information, between which the user should easily move using navigation techniques. In our framework this leads to the association of one (large) relation with a sets of linked display.

In this paper we present our approach for solving the above discussed problems that, being different in nature have to be faced separately (preliminary results about the matter appear in [15]). In Section 2, to suitably handle the mapping from logical tables to graphical windows, we introduce a general interaction structure, called *hypertable*, based on the assumption that the information should be presented on demand as a set of interconnected displays. The displays are dynamically generated on the basis of window dimensions, metadata information, and suitable exploration paradigms somehow captured by the interdisplay links. The hypertable is general in the sense that it does not assume any particular exploration strategy. Then, in Section 3, we specialize the approach to a semantic visual query language, to manage the query result through the *Table Expander*, a tool able to arrange the output of the query in a hypertabular manner, on the basis of the cardinalities of the involved relationships .

2 Embedding interaction into hypertables

Interaction models are introduced to bridge the conceptual distance between the visualization model and the data model, and to provide the structures for the user interaction. An interaction model must be formal to capture the concepts of the data model, and suitable to act as a formal counterpart of visual elements and interaction primitives. In this section we outline a methodology for defining interaction models based on *hypertables*, which are multi-display visualizations of (large) relations embedding links among relation fragments. As the output device is limited in terms of the containment area, the visual and the interaction models must be designed in terms of representations consequently constrained. We hence introduce the concept of *bounded relation* (or *fragment*), with upper-bounds on the number of tuples and on the number of schema attributes. Roughly speaking, any relation r is represented by a set of fragments from which it is possible to retrieve all the information contained in r. More formally:

Definition 1 Let U be a universe of *attributes*, and let $dom(A)$ be the *domain* of *values* associated to each attribute A in U. A *relation schema* R is a non empty subset of U. A *tuple* t on R is a function that associates a value $t(A)$ in $dom(A)$ to each attribute A in R. A *relation* $r(R)$ is a finite set of tuples on R. We say that $r(R)$ is an *(h,w)-relation* on R if w is the cardinality of R and h is the number of tuples of r. We refer to the ordered pair (h, w) as to the *dimensions* (*height* and *width*) of r.

When either dimension of a relation r leads to a table exceeding the display dimensions, it is necessary to associate to r a set of $(h, w)-$relations (*fragments* of r), where both h and w satisfies the dimensional constraints of the display.

Definition 2 A set F of fragments is said an *F-representation* of a relation r if and only if there exist two computable procedures $split$ and $coalesce$ such that $F = split(r, h, w)$ and $r = coalesce(F)$.

This definition of *F-representation*, though similar to the definition of lossless decomposition typical of the normalization process of the relational theory, differs from it in two ways: (1) dimensional constraints are used to define the fragments, (2) we do not limit the set of attributes and values in F to be subsets of those belonging to r (e.g., fragments containing aggregated values may belong to an F-representation).[2]

While fragments provide the navigation space, an adjacency structure linking them defines the admissible interaction. Depending upon the paradigm underlying the $split$ and $coalesce$ procedures, the links might be attached to the fragment as a whole, or to some element of it (e.g., attributes, values or tuples).

To be *good* with respect to the interaction, the adjacency structure must satisfy a number of criteria (we refer to [16] for an extensive discussion).

Completeness The set of links must guarantee that each fragment is reachable.

Regularity Some sort of consistency must be given to the adjacency structure, to enhance the navigation: the predictability of regular patterns allows the user to scan more easily the area of interest.

Meaningfulness Intuitive semantics must be defined for the links, according to navigation paradigm underlying the $split$ procedure.

Orderings on fragment schemata and on fragment extensions may also be required to help the navigation (e.g., it may be useful to visualize related attributes in adjacent columns of a table). As a matter of fact, ordering is somehow implied also by the existence of oriented links among fragments. To reflect this requirement in the interaction model, additional concepts are introduced.

Definition 3 An *(h-w)-table* is a triple $\langle r, S1, S2\rangle$ where r is an (h, w)-relation on a schema R, $S1$ is a sorting of R, and $S2$ is a sorting of the tuples of r. $S1$ and $S2$ may remain unspecified when no predefined ordering is required. We will often refer to (h-w)-tables simply as tables.

Definition 4 Let T be a set of (h, w)-tables. The set $F = \{f \mid \langle f, S1, S2\rangle \in T\}$ is said *induced* by T.

Definition 5 An *hypertable* HT is an ordered pair $\langle T, L\rangle$, where T is a set of (h, w)-tables, and L is an adjacency structure defined over the set of fragments induced by T.

Given its hypertextual nature, the proposed interaction structure is also suitable as a methodological framework for the definition of user interfaces in innovative environments as

[2] A consequence of the second aspect is that, in principle, there are infinite F-representations associated to r. It is therefore necessary to enforce some properties that F has to satisfy in order for the representation to be *good* with respect to the interaction (we refer to [16] for a discussion non redundancy, minimality, and meaningfulness criteria).

the World Wide Web. Reducing the dimensions of the table displayed on a single web page is profitable also in terms of performances, since the duration of individual transactions is shorter and more predictable. In more traditional database front-ends, however, the violation of height constraints is not as detrimental as the violation of width constraints, since it does not give rise to loss of context. Hence, for the sake of simplicity, we will not address this point in the proposed application framework.

3 A Hypertabular Visualizer

In this section we show how the approach can be specialized to a given query interface and query result exploration paradigm. In particular, we focus on a visual semantic query language, and, after a brief description of the query formulation strategy, we show how the information stored into the semantic schema can be usefully exploited for (1) fragmenting the resulting table, and (2) interconnecting such fragments in hypertabular manner.

3.1 The Query Interface

The availability of a high level description of the database through a semantic model results naturally in query interfaces in which the user visually interacts with a diagram representing the underlying semantic schema.

A noticeable effort has been spent in the definition and the experimentation of semantic query languages, ranging on a wide variety of data models (see, e.g., [8, 3, 7, 2, 9]). All of the available proposals adopt one between two opposite strategies: the *path-based* approach is seated on the idea of constructing the query by specifying a path on the schema, while the *view-based* strategy allows for defining a view on the schema. It is out the scope of this paper to discuss such a matter and here we recall the main ideas underlying the two strategies (a deeper analysis can be found in [12]).

In the path-based approach, the user specifies a path among the classes and the relationships of the schema. Roughly speaking, it corresponds to an ordered sequence of joins between the pairs ⟨class, relationship⟩ constituting the path, followed by a final selection and projection. The explicit presence of the relationships prevents the user from looking for concepts like foreign keys. More formally, each time the user path involves a class, say *c*, a variable is existentially quantified for that class and such a variable is required to appear in the incoming and in the outcoming relationships of the class *c* in the path. If the path involves *n* times a class, *n* different variables are quantified.

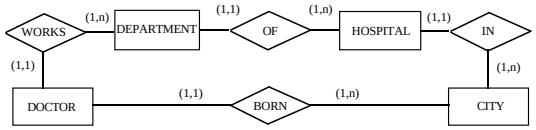

Figure 1. *A sample schema*

In the view-based approach the user specifies a view and the query corresponds to an unordered sequence of "natural joins". As an example, the query corresponding to the whole view of Fig. 1 coincides with the natural join of the four relationships $Works, Of, In$, and $Born$ on all the shared entities (in this case the relationship $Born$ shares the entity $Doctor$ with $Works$ and the entity $City$ with In), thus furnishing as answer all the doctors working in a hospital located in the same city the doctors were born.

3.2 The initial fragmentation

To introduce our query result presentation and exploration paradigm we use the example of Fig. 1. The resulting table contains attributes coming from different entities, and may present many repetitions of values.

Let $sch(E_i)$ and $key(E_i)$ denote the set of attributes and a selected key attribute of the entity E_i, respectively. Given a query Q, if $E_1, E_2, \ldots, E_n$ are the entities belonging to a query path/view, the resulting relation τ_Q has schema $T_Q = \cup_{i=1}^{n} sch(E_i)$.

The first natural choice in the fragmentation process is the isolation of clusters of attributes belonging to the same entity. Since an entity represents a class of real world objects sharing common properties, its common attributes possess a strong cohesion. Furthermore, at least under the hypothesis that the ER schema satisfies the first normal form requirements, these attributes are all in a one-to-one relationship with one another.

The initial set of fragments $\mathcal{F}$ is hence defined as:

$$\mathcal{F} = \{\cup_{i=1}^{n}\tau_{E_i} \cup \tau \mid \quad \tau_{E_i} = \pi_{sch(E_i)}(\tau_Q), \tau = \pi_T(\tau_Q), \\ T = \cup_{i=1}^{n} key(E_i), 1 \leq i \leq n\}$$

Roughly speaking, in each τ_{E_i} we isolate all the attributes of each entity E_i, while in τ we retain one key for each entity. The set $\mathcal{F}$ is an F-representation of τ_Q (it is easy to prove that the natural join of the fragments in $\mathcal{F}$ yields back the original relation τ_Q).

From each fragment τ_{E_i} a table $\mathcal{T}_{E_i}$ is defined by imposing an ordering on the attributes in $sch(E_i)$. The optimum sequence for presenting the attributes is determined by *semantic* factors (e.g., sequence of use, or frequency of conjoint use are taken into account to avoid contiguity of unrelated values), or *syntactic* factors (e.g., alphabetic order) in absence of the necessary statistical data.

Having isolated the entity attributes, one attribute per entity belongs to the fragment $\tau(T)$. The definition of a table associated to it requires the definition of a sorting of its schema attributes, described in the next subsection.

As to the the adjacency structure, each attribute $key(E_i) \in T$ will be the source of a link pointing to the fragment τ_{E_i} (see Fig. 2).

By applying the above procedure to the example under consideration, the fragment τ can be associated to a table $\mathcal{T}$ like the one in Fig. 3, containing many repetitions of values.

Figure 2. *A sample initial fragmentation*

CITY	HOSPITAL	DEPARTMENT	DOCTOR
ROME	GEMELLI	SURGERY	SPOCK
ROME	GEMELLI	SURGERY	ABBOT
ROME	GEMELLI	RADIOLOGY	ACTON
ROME	UMBERTO I	ONCOLOGY	TAFT
ROME	UMBERTO I	MATERNITY	UDALL
ROME	SAN GIOVANNI	SURGERY	LIDDEL
ROME	SAN GIOVANNI	SURGERY	LEE
VENICE	SAN MARCO	SURGERY	GALLUP
VENICE	SAN MARCO	SURGERY	GRAVES
VENICE	SAN MARCO	RADIOLOGY	NASH
VENICE	SAN MARCO	RADIOLOGY	BARTON
VENICE	POLICLINICO	MATERNITY	BELL
VENICE	POLICLINICO	ONCOLOGY	BENSON

Figure 3. *A table displaying* $\tau(T)$

One may notice that $\mathcal{T}$ can be iteratively partitioned from left to right, on the basis of repeated values. This is due to the particular orderings chosen for placing the attributes: the attributes in a one-to-many relationships with all the others (i.e., showing the greatest number of repetitions) appear first on the left, and the more we go to the right, the greater is the number of distinct values appearing in a column. This is exactly the way a table is expected to be: more general to the left, more specific to the right.

The following steps are hence: (1) to devise an algorithm for automatic attribute sorting of the above type, and (2) to take advantage of such sortings for the definition of a proper hypertabular structure.

3.3 Computing a partial order on attributes

The problem now is to compute the arity of the relationship between two clusters of attributes coming from two different ER concepts. The way in which to compute such an arity strongly differs, depending on which strategy (path-based or view based) is involved in the query formulation. The case of path-based has been discussed in [13], and here we report only the main results; the case of view-based strategy is, instead, deeply analyzed.

3.3.1 The Path-based case

Two cases are given: (1) the attributes under consideration are part of two concepts of the query path directly linked or (2) they are still on the same path, but "far" from each other. In the first case the numerical proportion can be derived directly from the cardinality of the ER schema, the maximum one being an indication of the greatest number of instances of one entity for each instance of the other. The second case is more complex: cardinalities found through a path must be combined to determine a *composed cardinality*.

Let us briefly describe the case of two near entities in the path $\langle Doctor, Works, Department, Of, Hospital\rangle$ of Fig. 1. Note that cardinalities are considered from the point of view of the relationship itself: the pair $(1,1)$ between $Doctor$ and $Works$ means that each instance of $Doctor$ is involved in the relationship $Works$ at least and at most once. Moreover, when comparing two entities we often consider only the maximum cardinalities and we say, e.g., that $Doctor$ and $Department$ are in many-to-one relationship through $Works$. More formally, we denote with $Gain(e_i, e_j, p)$ the maximum number of e_j instances associated with each e_i instance by the path p. For instance, $Gain(Doctor, Department, \langle Doctor, Works, Department\rangle) = 1$, while $Gain(Department, Doctor, \langle Department, Works, Doctor\rangle) = n$.

Composing the cardinalities, $Doctor$ is found to be in a (many-to-one)^2 relationship with $Hospital$, since $Doctor$ is in many-to-one relationship with $Department$, and $Department$ is in many-to-one relationship with $Hospital$. In this case, the natural presentation of attributes is in contrast to the specified path flow. It is more effective, in fact, to present first the attributes coming from $Hospital$, showing for each hospital the set of its departments and for each department the set of its doctors, avoiding the tedious repetition of the hospital for all its departments and the repetition of the department for each doctor in that department. Summarizing, on the basis of the arity of the relationships of the ER schema, a partial order relation on the table attributes is given. Such a partial order is matched against the total order defined by the user path that, when necessary, is altered according to the above considerations.

3.3.2 The View-based case

To find mutual numerical relationships between entities belonging to a view is a more complex task. Difficulty arises from the (possible) presence of multiple paths between concepts that must be composed together to devise a unique cardinality. In order to do that, we borrow from the above subsection the notion of composed cardinality, still valid when considering a single path on the view. Moreover, in this case we consider also minimum cardinalities. In order to devise a partial order among the entities belonging to a view S, we apply the following algorithm.

1. List all the entity pairs $\langle e_i, e_j\rangle$ with $i \neq j$ and $e_i \in S$;
2. For each entity pair devise all the non-cyclic paths connecting the two entities and, for each path, compute the composed cardinality;
3. For each entity pair $\langle e_i, e_j\rangle$, compute a *global path* and associate with it a *global cardinality*. Discard for each couple $\langle e_i, e_j\rangle\langle e_j, e_i\rangle$ the one characterized by the minimum $Gain(e_k, e_l, global - path)$ (if any);
4. Using the pairs coming from the previous step find out the partial order $\langle e_1, e_2, \ldots, e_{|E|}\rangle$ that maximizes

the $Gain(e_1, e_{|E|}, \langle e_1, e_2, \ldots, e_{|E|}\rangle)$ under the constraint that $\forall\langle e_i, e_j, e_k\rangle$ in the partial order, with $i < j < k$ it holds that $Gain(e_i, e_k, \langle e_i, e_k\rangle) > Gain(e_j, e_k, \langle e_j, e_k\rangle)$.

Let us analyze the above algorithm, applying it to the view in figure Fig. 1.

Step 1 We find out twelve pairs, combining the four entities belonging to the view. It is worth noting that, even if the complexity of this step is quadratic in terms of $|E|$, the number of entities belonging to S, the values of $|E|$ coming from real views make it tractable.

Step 2 For each of the above 12 pairs we look for all the non-cyclic paths (they are bounded by $|R|$, the number of relationships belonging to the view). In devising the composed cardinality we use the technique shown in the previous section, collecting also the minimum cardinality. For example, referring to the pair $\langle Doctor, Department\rangle$ we have two paths: $\langle Doctor, Works, Department\rangle$ and $\langle Doctor, Born, City, In, Hospital, Of, Department\rangle$.

Each instance of $Doctor$ is involved by the first path at least and at most 1 time; each instance of $Department$ is involved at least 1 time and at most n times.

The second path, instead, involves each instance of $Doctor$ at least 1 time and at most n^2 times; moreover it involves each instance of $Department$ is involved at least 1 time and at most n times.

We compute the other cardinalities in a similar manner.

Step 3 Now we have to combine, for each pair, the composed cardinalities coming from all the paths. Again we refer to the pair $\langle Doctor, Department\rangle$ to discuss the matter. As shown in Fig. 4, we can consider the two paths connecting $Doctor$ and $Department$ as two relationships[3].

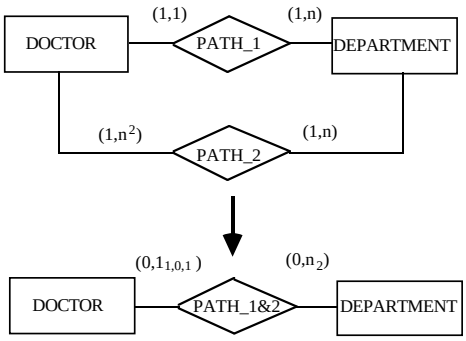

Figure 4. *Computing a global cardinality*

The semantics associated with the first path is the one of the relationships $Works$; through the second path each doctor is linked to all the departments of all the hospitals located in the city the doctor was born (this give also an intuitive meaning to the n^2 maximum cardinality).

The relationship resulting from the combination of the two paths corresponds to the logical AND: a pair $\langle Doctor, Department\rangle$ belongs to the new relationship only if it appeared in the two old ones. In other words, the new relationship links a doctor to the department (if any) that is at the same time the department in which the doctor works and it is located in the same city in which the doctor was born. This clarifies the value 0 assigned to the minimum cardinalities labeling the new relationship in Fig. 4. Concerning the maximum ones, they are bounded by the least maximum cardinality of all the paths: if a doctor is linked to at most 1 working department and at most n in-the-same-born-city departments, the intersection between the two sets will be either empty or composed by a unique instance.

Moreover we are interested in weighting the maximum cardinalities of the global cardinality, in order to have a thinner scale for the $Gain$ function. We introduce the weighting approach through an example. We have said that each doctor is involved $(0, 1)$ times in the new relationship and that the maximum cardinality 1 comes from the intersection of the n in-the-same-born-city departments and the single department the doctor works in. The new 1 maximum cardinality is less likely then the old 1 cardinality it comes from: the number of doctors involved in the new relationship are a subset of the ones involved in the old one. In order to capture this pieces of information, we keep track of all the maximum cardinalities contributing to the maximum global cardinality with the following notation: $1_{k,i_1,\ldots i_m}$, where k denotes the number of 1 cardinalities and i_j the number of n^j cardinalities involved in the construction of the global cardinality. If the maximum cardinality is a n, the k term is missing, being in this case the number of 1 cardinalities always 0. In this way, we can distinguish, for example, the $(1, n_{0,2})$ cardinality from the $(0, n_{1,1})$ one: they are both $(0, n)$ cardinality, but the first one comes from the intersection of $(1, n)$ and $(1, n^2)$ while the second one comes from the intersection of $(1, n^2)$ and $(1, n^2)$, so the latter maximum cardinality is greater than the former.

Comparing the global cardinalities of the pair $\langle Doctor, Department\rangle$, i.e., $(0, 1_{1,01}), (0, n_2)$, with the ones of $\langle Department, Doctor\rangle$, i.e., $(0, n_2), (0, 1_{1,01})$ we decide to discard the former an to keep the latter to be used in the next step.

Following the same strategy, we find out the pairs shown in Fig. 5.

<Department,Doctor>	(0,n_2)	(0,$1_{1,0,1}$)
<City,Doctor>	(0,$n_{1,0,1}$)	(0,1_2)
<Hospital,Doctor>	(0,$n_{1,1}$)	(0,$1_{1,1}$)
<City,Department>	(0,$n_{1,1}$)	(0,$1_{1,1}$)
<City,Hospital>	(0,$n_{1,1}$)	(0,$1_{1,0,1}$)
<Hospital,Department>	(0,n_2)	(0,$1_{1,0,1}$)

Figure 5. *The selected pairs for the view*

[3]To extend the following considerations to more than two paths is straightforward

Step 4 Using the pairs outcoming from Step 3, we have to find the partial order among the involved entities. Looking at Fig. 5 we notice that $Doctor$ is always on the right, so it has to be the last one in the order. Once $Doctor$ has been removed, the same happens for $Department$, that is the last but one. The same happens for $Hospital$, so the final partial order is the following: $\langle City, Hospital, Department, Doctor\rangle$.

3.4 Refining the fragmentation

Once the numerical relationships are defined, they are used to generate new meaningful fragments. As discussed before, the distinct values in any column determine a partition of the subtable on the right of the column. It makes sense to provide the distinct values of the first column as starting hints for the exploration of the query result. The user can then select one of them to see the associated information. To enforce regularity, and let the user feel completely free to browse through the entire table, the remaining columns are maintained.

Let us consider the schema $T_1 = \cup_{i=1}^{n} A_i$. Subscripts are representative of the relative order of attributes determined by the algorithm described in the previous subsection. Hence A_i is said *on the left* (resp. right) of A_j if $i < j$ (resp. $i > j$), denoted by $A_i < A_j$ (resp. $A_i > A_j$). From now on the notation T_i will be used to denote the schema $\cup_{j=i}^{n} A_j$, the Greek letter τ to denote fragments, and $\mathcal{T}$ to denote tables (i.e., used to put the emphasis on the sorting of the schema attributes).

Let us denote with $\mathcal{T}^*(T_1)$ the table associated to the fragment τ produced by the initial fragmentation in Section 3.2. The *starting fragment* $\tau_{start}(T_1)$ is defined in the following way:

1. determine the set $I = \pi_{A_1}(\mathcal{T}^*)$
2. for each value $v \in I$ insert in τ_{start} one and only one tuple $t(T_1)$ such that $t(A_1) = v$ and $t(T_1) \in \mathcal{T}^*$.

Step (1) determines the active domain of A_1 in $\mathcal{T}^*$, while Step (2) populates the starting fragment with one sample tuple for each such value. Notice that the links to the entity fragments are actually anchored to the attributes of the schema of $\tau_{start}(T_1)$.

Coming back to our example, in the first column two distinct values appear, namely $Roma$ and $Venice$. The starting point for the interaction is hence the table $\mathcal{T}_{start}$ shown in Fig. 6, in which only the distinct instances of the first column appear, each completed with one of the tuples associated to it.

At this point, a method must be introduced to allow the selective presentation of new data.

Given the schema T_1, we define $n-1$ fragment schemata T_i, $2 \leq i \leq n$. Given the table $\mathcal{T}^*$ and the fragment τ_{start}, each distinct value v_k, $1 \leq k \leq m$, in the first column of τ_{start} will be the source of $n-1$ links $l_i(v_k)$, $2 \leq i \leq n$,

CITY	HOSPITAL	DEPARTMENT	DOCTOR
ROME	GEMELLI	SURGERY	SPOCK
VENICE	SAN MARCO	SURGERY	GALLUP

Figure 6. *The starting fragment*

where $l_i(v_k)$ points to a fragment $\tau_i(v_k)$ with schema T_i and extension defined by the following procedure:

1. determine the set $I_i(v_k) = \pi_{A_i}(\sigma_{A_1=v_k}\mathcal{T}^*)$
2. for each value $w \in I_i(v_k)$ insert in $\tau_i(v_k)$ one and only one tuple $t(T_i)$ such that $t(A_i) = w$ and $t(T_i) \in \pi_{T_i}(\sigma_{A_1=v_k}(\mathcal{T})^*)$

Step (1) determines the distinct instances of the active domain of A_i belonging to tuples of $\mathcal{T}^*$ showing the value v_k in the first column. Step (2), consistently with the procedure followed to build $\mathcal{T}_{start}$, fills the remaining tuple fields.

By applying the above procedure to our example, for each distinct instance of the first column three additional fragments are defined and linked to it, with schemata $\langle Hospital, Department, Doctor\rangle$, $\langle Department, Doctor\rangle$, and $\langle Doctor\rangle$, respectively. Fig. 7 shows the fragments associated with the value $Rome$ from the starting fragment.

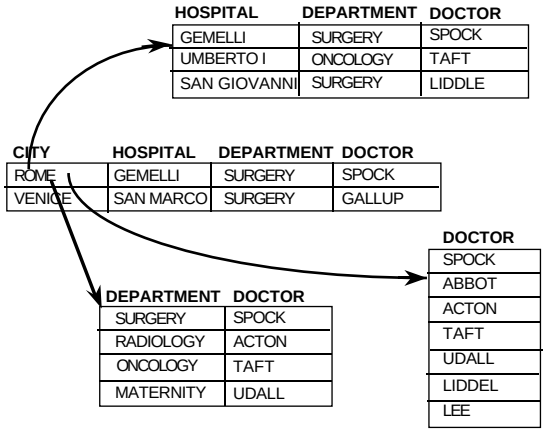

Figure 7. *Subtables associated with Rome*

Let us now introduce some aspects of the *visual model* adopted by our system to represent the abstract concepts of the interaction model.

From the user point of view, these new tables can be viewed as *expansions* of $\mathcal{T}_{start}$. In terms of the visual model, to expand $\mathcal{T}_{start}$, the user must select one instance of $City$ and one attribute on its right (thus uniquely identifying one of the three links attached to the instance value). Fig. 8 shows the graphical window containing the visual representation of the starting fragment. Each column is equipped withseveral buttons allowing for the necessary interaction:

Figure 8. *The starting fragment:the user interface*

- $Expand$ is used to select the starting column of the expansion;
- $Collapse$ closes the performed expansion;
- the right (left) arrow opens (closes) the entity fragment associated with the corresponding key attribute.

Fig. 9 shows the effect of the selection of $Rome$ and its further expansion starting from $Hospital$, while Fig. 10 shows the expansion of $Rome$ starting from $Department$.

Figure 9. *Expansion of the value Rome*

Figure 10. *A different expansion of Rome*

It should be intuitive that such an approach can be applied to any attribute and value of $\mathcal{T}_{start}$ as well as to any attribute and value of any new fragment. Selection and expansion can involve almost every pair of columns $\langle A_i, A_j \rangle$, with the only constraint that the $A_i < A_i$, since *everything* that is on the left (including the path of links necessary to reach the fragment) acts as a "key" to the sub-table.

The generalization of the above procedure is omitted due to space restrictions (it can be found in [14]). The basic idea is that given a fragment $\tau(T_i)$, the path of links from τ_{start} to τ, a value v in a column A_j (the selection column), and a column $A_k > A_j$ (the expansion column), it is possible to reconstruct a tuple $t(T_{-j})$, with $T_{-j} = \cup_{l=1}^{j} A_l$, used to determine the distinct instances of the active domain of A_k belonging to tuples of $\mathcal{T}^*$ coinciding with $t(T_{-j})$ on the subschema T_{-j}.

It is easy to prove it is possible to reconstruct $\mathcal{T}^*$ from the hypertable $\mathcal{HT}^*$ containing all and only the fragments obtained by iteratively expanding the first columns [14].

It must be observed that, in practice, new fragments are dynamically generated by the interface on-demand, only if the user explicitly asks to access such information.

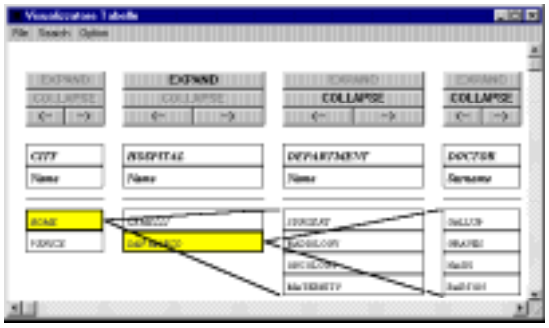

Figure 11. *Multiple expansions*

3.5 The Table Expander Prototype

A prototype of the system has been implemented, using the C++ language and the XVT graphical toolkit, available for different platforms (Dos, Mac-OS, and Unix). Some additional features have been included in the prototype to enrich the interaction. A *change-selection* method is supplied. It is likely that the user is interested in moving through the instances of one attribute to browse different sub-table. For example, after the analysis of the information related to $Rome$ in Fig. 9, the selection of $Venice$ yields the table shown in Fig. 12.

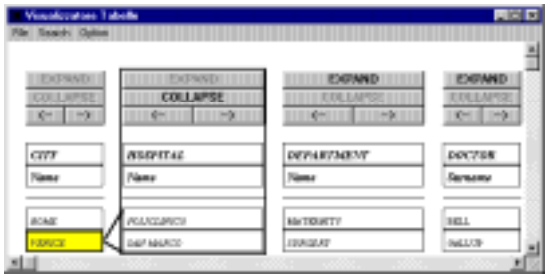

Figure 12. *Expansion of the value Venice*

To the left of the selection, however, there might be many expansions depending on it, directly or indirectly. The default approach is to contract all subsequent expansions, related to the changed selections, forcing them to fit the expansion level of the column containing the selection. A *keep-expansions* option, anyway, allows the user to maintain the expansion pattern. Finally, the user is always able to bring things back to a situation equal – or at least compatible if something else has changed – to the one present before an expansion (proper strategies are supplied for propagating such *contractions*).

4 Conclusion

In this paper we presented (1) a general hypertextual framework for the interaction with tables, and (2) a possible specialization for visual semantic query languages, able to cluster attributes in homogeneous way, and to suitably handle tedious value repetitions. In contrast with [10] where tables are viewed as a data model, here (hyper)tables are regarded as a *data set organizer* for the analysis of query results.

The main idea is based on the assumption that information should be visually presented on demand as a set of displays, dynamically generated on the basis of: (1) window dimensions, (2) metadata information, and (3) suitable exploration paradigms captured by interdisplay links.

The selected visualization and exploration strategies are based on information stored in the semantic representation of data. The table content is rearranged as a set of smaller interconnected displays that avoids repetitions of values and contiguity of unrelated data (typically produced by the join of two or more relational tables). Result exploration is performed by following interdisplay links, which activate table expansions or contractions.

Given its hypertextual nature, the proposed interaction structure is also suitable as a methodological framework for the definition of user interfaces in innovative environments (as the World Wide Web).

References

[1] C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *CHI '94*, pages 151–164, ACM, New York, 1994.

[2] M. Angelaccio, T. Catarci, and G. Santucci. Qbd*: a graphical query language with recursion. *IEEE Transactions on Software Engineering*, 16(10):1150+, 1990.

[3] D. Bryce and R. Hul. Snap: A graphics-based schema manager. In *Proc. IEEE Intl. Conference on Data Engineering*, pages 151–164, Los Angeles, USA, 1986.

[4] S.K. Card. Visualizing retrieved information: a survey. *IEEE Computer Graphics and Applications*, 16(2):63–67, 1996.

[5] P.P. Chen. The entity relationship model toward a unified view of data. *ACM Transactions on Data Base Systems*, 1(1), 1976.

[6] R. Chimera. Value bars : an information visualization and navigation tool for multi-attribute listings. In *CHI '92*, pages 293–294, 1992.

[7] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proc. ACM-SIGMOD Conference on the Management of Data*, pages 151–164, 1987.

[8] R. Elmasri and G. Wiederhold. Gordas : a formal high-level query language for the entity-relationship model. In *Proc. 2nd Intl. Conference on the Entity Relationship Approach*, pages 49–72, 1981.

[9] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object model for database end-user interfaces. In *Proc. ACM-SIGMOD Conference on the Management of Data*, pages 24–33, Atlantic City, USA, 1990.

[10] L.V.S. Lakshmanan M. Gyssens and I.N. Subramanian. Tables as a paradigm for querying and restructuring. In *Proc. of IEEE 1996 Symposium on Visual Languages*, pages 93–103, 1996.

[11] R. Rao and S.K. Card. The table lens: Merging graphical and symbolic representation in an interactive focus + context visualization for tabular information. In *CHI 94*, pages 318–322, 1994.

[12] G. Santucci. On graph-based interaction for semantic query languages. In *Proc. of IEEE 1996 Symposium on Visual Languages*, pages 76–83, 1996.

[13] G. Santucci and F. Palmisano. A dynamic form-based data visualizer for semantic query languages. In *Sawyer P (ed) Interfaces to Database Systems*, pages 249–265, 1994.

[14] G. Santucci and L. Tarantino. A hypertabular front-end for the visualization of query result. *unpublished manuscript*, 1996.

[15] G. Santucci and L. Tarantino. To table or not to table: a hypertabular answer. *Special Issue on Information Visualization, Sigmod Record*, 25(4), 1996.

[16] L. Tarantino. Hypertabular representations of database relations in world wide web front-ends. In *(Barclay, P. and Kennedy, J. eds) Interfaces to Databases (IDS-3)*, Springer Verlag, 1996.