An SQL extension supporting user viewpoints

Giuseppe Bellavia, Dario Maio, Stefano Rizzi
DEIS - Facolta' di Ingegneria, Universita' di Bologna

Abstract - In order to accomplish independence on the logical data organization, a relational DBMS must be capable of interpreting query language sentences which reference attributes belonging to different relations even if the necessary joins are not explicitly formulated (query inference problem). In this work we propose an SQL extension which supports concise formulation of queries through the dynamic definition of user viewpoints. User viewpoints are perspectives for accessing data; each user viewpoint defines a virtual derived relation (user viewpoint relation) which includes all those in the database scheme, each accessed by means of exactly one chain of system-activated equi-joins. Our approach to the definition of user viewpoint relations aims at reducing the global cost of query formulation, that is, the number of joins which must be explicitly written.

1 Introduction

A relational query language is independent of the logical data organization if its sentences can reference attributes belonging to different relations without explicitly formulating the necessary joins. In order to support this functionality, a Relational Database Management System (RDBMS) must be able to translate the query language sentences into unambiguous representations based, for instance, on the formalism of relational algebra. This is known in the literature as the *query inference problem* [7].

Let the logical scheme of the database be represented by a graph, which we call *database graph*, where each node corresponds to a relation scheme and each arc to a logical association between two relation schemes (typically determined by a foreign key definition). Solving the query inference problem requires the selection, for each given sentence of the query language, of an acyclic sub-graph of the database graph spanning all the nodes corresponding to the relations whose attributes are referenced in that sentence.

Different approaches for simplifying query formulation can be found in the literature. In the *universal relation* approach, query inference is addressed by building a derived relation which combines all the relations in the database through natural joins [3]. On the other hand, the universal relation calls for requirements not always satisfied in practical applications [4], and generates a fixed sight of the database, on whose structure the user cannot intervene.

In [7], derived relations are computed following a graph-theoretic approach. Each sentence of the query language is translated into a query by determining, on the database graph, the minimum directed cost Steiner tree. In [6], query disambiguation is carried out by considering the *relatedness* of the relations involved and the existence of directed paths between them. In [5], query disambiguation is carried out by choosing the interpretations which contains less connections between attributes defined on the same domain).

All these approaches differ from ours, since they do not consider the possibility of accessing data through multiple perspectives. In [2] we have proposed an original approach to query inference, in which each relation scheme may be used as a perspective to formulate queries (*user viewpoint*). Each user viewpoint virtually defines a derived relation (*user viewpoint relation*, UVR) which includes all the relations in the database, each accessed by means of exactly one chain of system-activated equi-joins. The guiding criterion in defining UVRs is the reduction of the global cost of query formulation,

being the *cost* of a query a function of the number of joins which must be explicitly included in a sentence of the query language in order to obtain that particular query.

In this work we propose an extension to SQL which allows for user viewpoints to be defined at query level; an extension to standard SQL is described first; also an SQL2 extension is proposed, in order to enable formulation of outer joins. In each sentence, one or more user viewpoints may be adopted. The adoption of a user viewpoint in a query entails the automatic derivation of a virtual relation (*local UVR*) including only the relations mentioned in the query with reference to that viewpoint; this view can then be subjected to selection, projection, and grouping operations through the classic SQL clauses.

It is remarkable that query formulation from a viewpoint is not equivalent to query formulation on a relational view defined by a **create view** statement. In fact, our approach brings several advantages:

- Defining a relational view requires several joins to be explicitly formulated, while defining a local UVR only requires to choose a viewpoint.
- The viewpoint can be changed dynamically; instead, (at least) one different view should be written for each viewpoint.
- The join type (inner or outer) is fixed within the view, whereas it can be changed within each local UVR.
- If the database scheme is changed (a relation scheme is added, dropped or modified), all involved views must be rewritten; instead, local UVRs will be automatically generated based on the new scheme.

2 From Database Schemes to Graphs

Let D be a database scheme including the relation schemes $R_1,...R_n$ and characterized by a set of logical associations between pairs of relation schemes. A logical association is typically determined by a foreign key constraint on two (comparable) attributes; the database designer can declare explicitly other logical associations when the comparison between two attributes is relevant. The logical associations determined from foreign keys may be many-to-one or one-to-one, while those declared explicitly may also be many-to-many.

We describe the logical association involving attributes A_i of R_i and B_j of R_j by means of a pair of opposite connections between R_i and R_j, which we call *potential links* (PLs) and denote with $(R_i.A_i,R_j.B_j)$ and $(R_j.B_j,R_i.A_i)$, respectively. Two relation schemes may be connected by several distinct pairs of opposite PLs corresponding to logical associations involving different pairs of attributes.

We are interested in describing each PL through two properties concerning the corresponding logical association: *strength* and *multiplicity*. By default, a PL $(R_i.A_i,R_j.B_j)$ is strong if attributes A_i and B_j have the same name; weak otherwise. If the scheme has name inconsistencies, the database administrator may explicitly declare which PLs are strong, that is, which logical associations are most relevant and should be privileged for query inference. We say a PL $(R_i.A_i,R_j.B_j)$ is *single* if the logical association involving A_i and B_j is one-to-one or many-to-one, *multiple* if it is one-to-many or many-to-many. In single PLs, one tuple of r_i references one tuple of r_j, therefore the association between R_i and R_j is assumed to be more significant than in multiple PLs, in which one tuple of r_i references several tuples of r_j.

We represent the database scheme D by means of a *database graph* $\mathcal{D} = (\mathcal{R}, \mathcal{L})$; each node in $\mathcal{R}$ corresponds to a relation scheme, each arc in $\mathcal{L}$ corresponds to a PL and is

labelled with the two attributes involved and with the properties of that PL.

Also queries can be described through a graph formalism. Let q be a query on database scheme D. We represent q by means of a *query graph* $\mathcal{Q}_q$; each node corresponds to a relation scheme whose attributes are referenced by q, each arc represents a join required by q and is labelled with the two attributes involved.

We will explain these concepts with two examples, which will be used also in the rest of the paper: the well-known suppliers-and-parts database (SPD) and the conference database (CD).

The scheme we adopt for the SPD is sketched below (primary keys are underlined):

D = { PART (#P, Description, Colour, Weight, City),
SUPPLIER (#S, Name, City, Status), SUPPLY (#P, #S, Quantity) }

The conceptual E/R scheme of the CD is shown in Figure 1. A possible database scheme is as follows:

D = { TOPIC (#Topic, Description), REFEREE (#Referee, Name),
IS_EXPERT (#Referee, Expertise, Confidence),
PAPER (#Paper, Title, AboutTopic), INVITED (#Paper, Proposer),
SUBMITTED (#Paper, SubDate, Accepted), AUTHOR (#Author, Name),
WRITES (#Author, #Paper), REFEREES (#Paper, #Referee, Rating) }

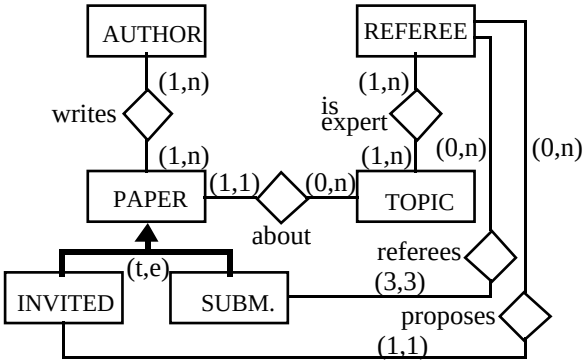

Fig. 1. E/R scheme for the CD. The minimum and maximum multiplicity of relationships is shown in parentheses. The IS-A hierarchy is total and exclusive.

Figure 2 shows the database graphs representing the two database schemes; the compact graphical notation used for arcs emphasizes the properties of the corresponding PLs.

An example of query graph is the one in Figure 3; a query described from this query graph is the one displaying, for each submitted paper, its topic and its authors. Note that, if the query requires only inner joins, the directions of arcs are irrelevant.

3 From Graphs to Derived Relations

In this section we discuss how a sub-graph of the database graph can be univocally associated with a derived relation.

Let $\mathcal{D} = (\mathcal{R}, \mathcal{L})$ be a database graph and $\mathcal{T} = (\mathcal{R}', \mathcal{L}')$, $\mathcal{R}' \subseteq \mathcal{R}$, $\mathcal{L}' \subseteq \mathcal{L}$, be a (directed) tree with root in $R_p \in \mathcal{R}'$. Let $(R_{k1},...R_{km})$ be any sequence of the nodes in $\mathcal{R}'$ such that:

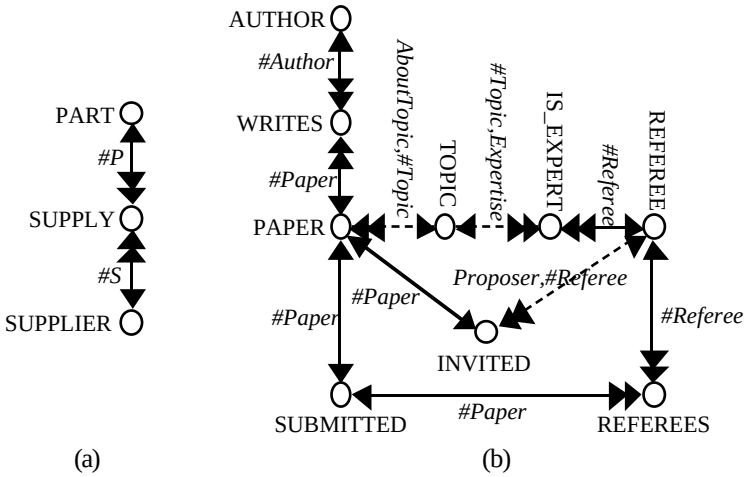

Fig. 2. Database graphs representing the SPD (a) and the CD (b) schemes. For simplicity, each pair of opposite arcs is represented by one connection; when the two attributes involved in a PL have the same name, the name is written only once. Normal and dashed lines represent strong and weak PLs, respectively. Double and single arrows represent multiple and single PLs, respectively.

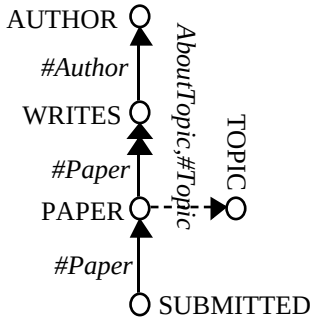

Fig. 3. A query graph on the CD.

1. the first node in the sequence is R_p;
2. for each other node R_{ki} in the sequence, there exists in $\mathcal{L}'$ an arc whose second endpoint is R_{ki} and whose first endpoint is a node appearing in the sequence before R_{ki}.

The derived relation associated to $\mathcal{T}$ is defined as follows:

$$r_p \rhd\lhd_{A_2=B_2} r_{k2} \ldots \rhd\lhd_{A_n=B_n} r_{km}$$

where "$r_i \rhd\lhd_{A=B} r_j$" denotes the inner equi-join between relations r_i and r_j on attributes A and B, and A_i and B_i are the attributes labelling the arc which enters R_{ki} in $\mathcal{T}$. It can be proved that the derived relations corresponding to the different sequences of the nodes in $\mathcal{R}'$ which may satisfy the condition above, are identical (except for the ordering of the attributes).

The derived relation associated to the graph in Figure 3, which is a tree with root in SUBMITTED, can be expressed as:

$$\text{submitted} \rhd\lhd_{(\#Paper=\#Paper)} \text{paper} \rhd\lhd_{(\#Paper=\#Paper)} \text{writes} \rhd\lhd_{(\#Author=\#Author)} \text{author} \rhd\lhd_{(AboutTopic=\#Topic)} \text{topic}$$

4 User Viewpoint Relations

A *user viewpoint* is a specific perspective for accessing data. Users define the viewpoint by selecting a *primary relation* (PR), that is, the relation which interests them most with reference to one or more queries. Choosing a PR allows for determining a *user viewpoint relation* (UVR), that is, a derived relation including all those in the database scheme, each accessed from the PR by means of exactly one chain of system-activated equi-joins. The UVR solves the query inference problem, since it allows for any sentence of the query language to be translated into an unambiguous representation.

Let $\mathcal{D} = (\mathcal{R}, \mathcal{L})$ be a database graph and $R_p \in \mathcal{R}$ be the PR. The requirements outlined for the UVR are satisfied from the derived relation associated to a spanning tree on $\mathcal{D}$ with root in R_p. In fact, a spanning tree on $\mathcal{D}$ includes all nodes in $\mathcal{R}$, and connects the root with each other node through exactly one directed path.

Let q be a query on $\mathcal{D}$ and $\mathcal{Q}_q$ be its query graph. Let $\mathcal{T}_p$ be the spanning tree determining the UVR from viewpoint R_p. We say q is *implicit* for viewpoint R_p if all the arcs in $\mathcal{Q}_q$ also belong to $\mathcal{T}_p$, that is, $\mathcal{Q}_q \subseteq \mathcal{T}_p$; *explicit* otherwise. Implicit queries can be formulated relying entirely on system-activated joins, that is, by referencing in a sentence of the query language the attributes to be displayed. Formulation of explicit queries require, in addition, a join to be explicitly written for each arc in the query graph which does not belong to the tree.

If $\mathcal{D}$ does not contain cycles (except those generated by each pair of opposite PLs), each choice of the PR determines exactly one spanning tree on $\mathcal{D}$, which can be obtained by dropping an element from each pair of PLs in $\mathcal{D}$. For instance, Figure 4 shows the spanning trees from viewpoints PART and SUPPLIER in the SPD, whose database graph is acyclic. Note that the UVR does not automatically connect identical attributes; for instance, the existence of the attribute "City" in both relation schemes PART and SUPPLIER of the SPD does not create inconsistencies.

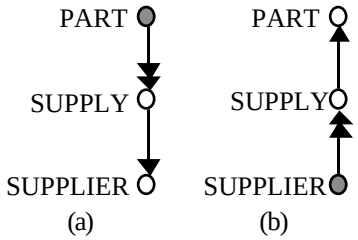

Fig. 4. Spanning trees from viewpoints PART (a) and SUPPLIER (b) on the database graph for the SPD.

If the database graph $\mathcal{D}$ contains one or more cycles, several spanning trees can be defined for each PR. In order to ensure unambiguous interpretation of sentences of the query language, exactly one of them must be chosen to generate the UVR. Since we are primarily interested in the query inference problem from the position of the software developer, we choose the optimal tree for a given PR by evaluating how concisely each given query can be formulated.

We define the *formulation cost* of a query q, c(q), as the number of joins which must be explicitly written in a sentence of the query language in order to produce q.

The global cost for formulating the queries of a given workload $\mathcal{W}$ is:

$$U = \sum_{q \in \mathcal{W}} c(q) \cdot f(q)$$

where f(q) is the expected formulation frequency of query q. In a query language where no inference technique is adopted, c(q) is equal to the number of arcs in the query graph of q, u(q). The adoption of a user viewpoint R_p reduces the global cost of query formulation. In fact, let $\mathcal{T}_p$ be a spanning tree with root R_p. The cost of implicit queries is c(q) = 0; as to explicit queries, some of the arcs of their query graphs may belong to $\mathcal{T}_p$, in which case the corresponding joins must not be written: in general, $c(q) \leq u(q)$. With respect to workload $\mathcal{W}$, the optimal tree for viewpoint R_p is the spanning tree which minimizes the global cost of query formulation U.

In general, during the software development phase, workload $\mathcal{W}$ is not precisely known; for this reason, for each PR the cost of query formulation is minimized according to a synthetic workload including only the queries whose query graphs are included in the database graph and are paths from the PR to any other relation scheme R_i in $\mathcal{R}$. The frequencies of the queries in the synthetic workload are estimated by taking into account the semantics of the database scheme, which we believe is decisive in making the inference predictable. Hence, each query is assigned a frequency which depends on the strength and multiplicity of the PLs involved: coarsely, we assume that the most frequent queries are those which require joins corresponding to strong and single PLs. A detailed definition of the synthetic workload can be found in [2].

It can be proved that, on the assumptions made, calculating the optimal spanning tree is equivalent to solving a shortest path problem; the complete demonstration can be found in [1]. The algorithm employed by the query inference manager to construct the optimal tree derives from Dijkstra's shortest path algorithm. The complexity in time of the Dijkstra algorithm does not exceed $O(n^2)$, where n is the number of relation schemes in $\mathcal{R}$.

The database graph for the CD is cyclic; the optimal trees for the SUBMITTED and the REFEREE user viewpoints are shown in Figure 5.

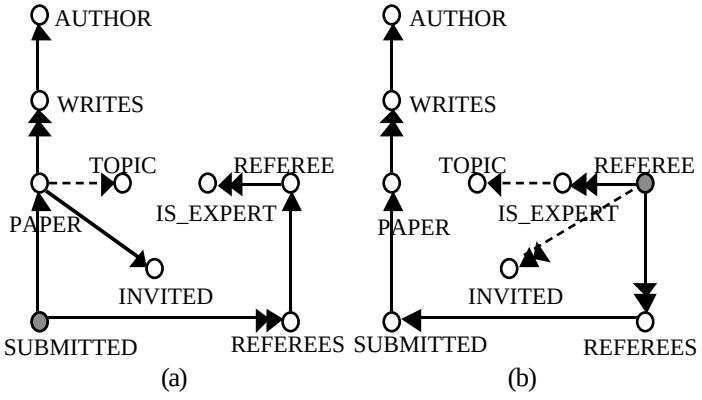

Fig. 5. The optimal spanning trees with primary relations SUBMITTED (a) and REFEREE (b) on the database graph for the CD.

5 An SQL Extension Supporting User Viewpoints

In this section we propose a standard-SQL extension which enables concise query formulation from user viewpoints. In this extension, the clause **from** is used to specify the user viewpoint(s) for the current query and, if necessary, to add one or more joins to the optimal tree(s).

The syntax we propose may be expressed as follows:

<fromClause> ::= **from** <itemDef> {, <itemDef>}
<itemDef> ::= <uvDef> | <aliasDef>
<uvDef> ::= **viewpoint** <relName> [<aliasName>]
<aliasDef> ::= <relName> <aliasName>

The simplest form of the **from** clause allows for all implicit queries with single viewpoint to be expressed:

select <list of expressions> **from viewpoint** R_p

Let $\mathcal{T}_p$ be the spanning tree corresponding to viewpoint R_p, and $R_{k1},...R_{km}$ be the relation schemes referenced in the query (within <list of expressions> and in other clauses such as <where>, <group by>, etc.). Let $\mathcal{T}_p'$ be the tree union of the m paths leading in $\mathcal{T}_p$ from R_p to the nodes $R_{i1},...R_{im}$ ($\mathcal{T}_p' \subseteq \mathcal{T}_p$); we call *local UVR* the derived relation associated to $\mathcal{T}_p'$. In order to solve the query, the <where>, <group by>, etc. clauses are applied to the local UVR.

The attributes of the local UVR are all the attributes of the relations schemes mentioned in the query. An attribute whose name is non-ambiguous within the UVR (for instance, Description) may be referenced directly; when addressing an attribute whose name is ambiguous (for instance, a Name attribute exists in both AUTHOR and REFEREE relation schemes), the name of the relation scheme must be specified.

It should be noted that, in general, the local UVR is *not* a projection of the UVR. In fact, while the UVR entails a join for each arc in $\mathcal{T}_p$, in the local UVR only the joins corresponding to arcs in $\mathcal{T}_p'$ are executed. Since joins are of inner type, it is possible that some tuples appearing in the local UVR are absent from the UVR because they do not match on a PL not included in $\mathcal{T}_p'$.

Consider for instance the implicit query

select AUTHOR.Name **from viewpoint** SUBMITTED
where Description = "Query Inference"

on the CD, which asks the names of the authors of submitted papers concerning the query inference problem (the query graph is shown in Figure 3). The optimal tree from viewpoint SUBMITTED is shown in Figure 5.a; the local UVR used in this query can be expressed in standard-SQL as follows:

select * **from** SUBMITTED, PAPER, TOPIC, WRITES, AUTHOR
where SUBMITTED.#Paper = PAPER.#Paper
and PAPER.AboutTopic = TOPIC.#Topic
and PAPER.#Paper = WRITES.#Paper
and WRITES.#Author = AUTHOR.#Author

Note that, due to the inner join between PAPER and INVITED, the UVR from viewpoint SUBMITTED is empty (the IS-A hierarchy is exclusive).

When formulating an explicit query, aliases are used in order to add joins to the local UVR. The query

select <list of expressions> **from viewpoint** R_p, R_{k1} AL1, ... R_{kj} ALJ

defines the Cartesian product between the local UVR and the relation schemes R_{k1},... R_{kj} with aliases AL1,...ALJ, respectively. Thus, each of the relation schemes R_{k1},... R_{kj} appears in the query (at least) twice: with its name (connected to the PR through the joins in the optimal tree) and with its alias (external to the local UVR, and possibly joined with it through a selection predicate).

For instance, consider the query which asks, for each submitted paper, its title, its topic and the topics in which its referees are expert with high confidence. This query is explicit for the SUBMITTED viewpoint, since the UVR does not provide the join between TOPIC and IS_EXPERT (see Figure 6). The missing join must be formulated explicitly:

select Title, Description, T.Description
from viewpoint SUBMITTED, TOPIC T
where Expertise = T.#Topic **and** Confidence = "High"

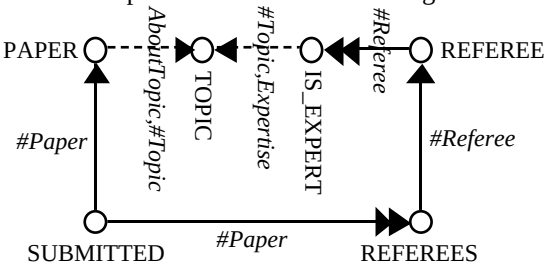

Fig. 6. The query graph of an explicit query for viewpoint SUBMITTED.

This formulation is equivalent to the following standard formulation:

select Title, T1.Description, T2.Description
from SUBMITTED, PAPER, TOPIC T1, REFEREES, IS_EXPERT, TOPIC T2
where SUBMITTED.#Paper = PAPER.#Paper
and PAPER.AboutTopic = T1.#Topic **and** PAPER.#Paper = REFEREES.#Paper
and REFEREES.#Referee = IS_EXPERT.#Referee
and IS_EXPERT.Expertise = T2.#Topic **and** Confidence = "High"

When formulating nested queries, both the external and the internal queries may have their own user viewpoints. Thus, in order to differentiate the local UVRs, an alias should be given to each PR. The following correlated query returns the supplier-and-part pairs such that the supplier's status is greater than the average status of the suppliers which live in the same city and supply that part; both the external and the internal queries are formulated from viewpoint SUPPLIER:

select SX.Name, SX.Description **from viewpoint** SUPPLIER SX
where SX.Status >
(**select** avg (SY.Status) **from viewpoint** SUPPLIER SY
where SY.SUPPLIER.City = SX.SUPPLIER.City
and SY.SUPPLY.#P = SX.SUPPLY.#P)

Note that, for ambiguous attributes such as City, referencing requires both the PR alias and the relation name; for non-ambiguous attributes such as Name, the PR alias is sufficient.

In some cases, it may be useful to create two or more local UVRs within a single

query. Consider for instance the query asking for the couples of suppliers who supply parts of the same colour:

```
select distinct SX.Name, SY.Name
from viewpoint SUPPLIER SX, viewpoint SUPPLIER SY
where SX.Colour = SY.Colour and SX.Name > SY.Name
```

Standard SQL allows for formulating only inner joins; instead, in several queries it is necessary to return also unmatched tuples. The user viewpoint approach supports outer joins as well; in particular, local UVRs can be generated by mixing inner and outer equi-joins. The language we propose to support user viewpoints with outer joins is an extension of SQL2:

```
<fromClause>   ::= from <itemDef> {, <itemDef>}
<itemDef>      ::= <uvDef> | <aliasDef> | <addJoinDef>
<uvDef>        ::= viewpoint <relName> [<aliasName>] {, <uvrJoinDef>}
<addJoinDef>   ::= <name> <joinType> <aliasDef> on <joinPredicate>
<uvrJoinDef>   ::= <joinType> <relName>
<aliasDef>     ::= <relName> <aliasName>
<name>         ::= <relName> | <aliasName> | <aliasName>.<relName>
<joinType>     ::= left outer join | right outer join | full outer join | inner
```

In this extension, the **from** clause allows for specifying which PLs must be accomplished in the local UVR through outer joins. Since in the optimal tree only one PL enters each relation scheme, an outer join on a PL can be declared by referencing only the relation scheme where the PL ends up (<uvrJoinDef> production). Besides, it is possible to add inner or outer joins to the local UVR; inner joins are still declared through an alias definition, whereas outer joins are declared in the classic SQL2 syntax (<addJoinDef> production).

An example of implicit query on a local UVR including an outer join is:

```
select REFEREE.Name, Title
from viewpoint REFEREE, left outer join REFEREES
```

which returns all the referee names together with the titles of the papers they have reviewed, if any (the optimal tree for the REFEREE viewpoint is shown in Figure 5.b). This query would be formulated in SQL2 as:

```
select REFEREE.Name, Title
from REFEREE, REFEREE left outer join REFEREES
        on REFEREE.#Referee = REFEREES.#Referee, SUBMITTED, PAPER
where REFEREES.#Paper = SUBMITTED.#Paper
and SUBMITTED.#Paper = PAPER.#Paper
```

An example of explicit query which adds an outer join to the UVR is:

```
select REFEREE.Name, Description, Title
from viewpoint REFEREE, TOPIC left outer join PAPER P
        on TOPIC.#Topic = P.AboutTopic
where TOPIC.#Topic = P.AboutTopic
```

which returns all the referee names together with the topics they are expert in and the titles of the papers on that topics, if any. This query would be formulated in SQL2 as:

```
select REFEREE.Name, Description, Title
from REFEREE, IS_EXPERT, TOPIC, TOPIC left outer join PAPER
        on TOPIC.#Topic = PAPER.AboutTopic
```

where REFEREE.#Referee = IS_EXPERT.#Referee
and IS_EXPERT.Expertise = TOPIC.#Topic
and TOPIC.#Topic = PAPER.AboutTopic

6 Conclusions

This paper has presented an SQL extension which supports query inference through user viewpoints. From any viewpoint the user can formulate queries in a very concise and straightforward way, leaving to the system the task of activating paths between relations and executing joins. The queries which are not implicitly supported can be formulated by means of explicit joins.

The user viewpoint approach could be integrated within a graphic database development tool. This tool should manage the different viewpoints, in particular by displaying and navigating the UVR corresponding to each of them. The designer would thus be allowed to immediately verify the system's interpretation of the queries. Should this interpretation be inadequate, the designer could choose to interactively modify the tree in order to take extra semantic information not supported by the system into account.

We conducted extensive testing of the user viewpoint approach by developing medium-size applications. The most significant example is the information system of the Faculty of Engineering of the University of Bologna, which was entirely developed by a team of non-skilled programmers. In all the applications the user viewpoint approach reveals its robustness and flexibility of use; the inference provided by the system fits the user expectations in most cases.

We believe that the development of object-oriented applications, where the database scheme evolves rapidly, will take advantage of the user viewpoint approach. Suppose we add to the CD a new relation COMPANY which stores the companies for which authors work. COMPANY is automatically included in all the UVRs, and its attributes can be directly addressed. Of course, modifications which impact on cycles in the database graph may alter the trees. The most simple solution to face this problem consists in alerting the user issuing the modifications with a message reporting whether the UVRs will be altered, and to what extent.

References

[1] G. Bellavia, D. Maio and S. Rizzi. Resolving the query inference problem by optimizing the query-formulation cost. Technical Report CIOC-C.N.R. n. 85 (1992).

[2] G. Bellavia, D. Maio and S. Rizzi. Minimizing the cost of query formulation through User Viewpoint Relations. *Atti del Secondo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati*, Rimini, Italy, 141-159 (1994).

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

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

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

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

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