From: Proceedings of the Twelfth International FLAIRS Conference. Copyright © 1999, AAAI (www.aaai.org). All rights reserved.

Cooperative Relational Database Querying Using Multiple Knowledge Bases

José Luís Braga+
Departamento de Informática
Universidade Federal de Viçosa
36571-000-Viçosa-MG-Brazil
(zeluis@mail.ufv.br)

Alberto H. F. Laender
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-901-Belo Horizonte-MG-Brazil
(laender@dcc.ufmg.br)

Claudiney Vander Ramos
Engetron Engenharia
Belo Horizonte-MG-Brazil
(cvramos@engetron.com.br)

Abstract

We present in this paper an approach for providing cooperativeness in database querying using artificial intelligence techniques. The main focus is a cooperative interface that assists non-experienced and casual users in extracting useful answers from a relational database. We propose an architecture for our approach that comprises two knowledge bases which store rules that describe the application domain and that guide the process of query formulation and answering. A subset of SQL is used for expressing queries, and the cooperative interface relieves the user from knowing its full syntax and also the database schema.

1. Introduction and context

A common problem for many database users is how to formulate and submit correct queries in order to get useful responses from the system, with little or no knowledge of the database structure. This problem becomes even worse when the target database is generated and maintained by some legacy system, usually in a distributed environment around remote locations.

In traditional systems, users may pose queries to a database by using a query language (such as SQL, QBE or QUEL) or a form-based interface or by embedding commands in an application program [7]. However, such facilities are not, in general, user-oriented and therefore do not provide means for helping users to formulate and submit queries in a cooperative fashion in order to extract the required information from the database [9]. When there is no answer for a query, a DBMS does not usually look for an alternative one which may satisfy the users' information needs. This lack of cooperativeness [11] imposes limitations to the widespread use of DBMS's [1], making them hard to be used by non-experienced and casual users, since they have to learn many details about the database schema and about the DBMS itself.

In this paper, we propose an architecture for cooperative access [11,10] to databases and describe a cooperative interface for querying relational databases which has been implemented based on this architecture. The term *cooperative interface* is used to refer to interfaces that assist non-experienced and casual users to get useful answers from a database [14]. In this way, users and the interface become partners in the querying process with the aim to extract more relevant information from the database. This class of interfaces is associated to a larger research area, called Cooperative Information Systems [11]. This area became one of great interest today, because it provides solutions to the problem of integration and use of legacy databases. Other related technologies are those of the mediators and software agents [17].

In our approach, we provide cooperativeness by means of two knowledge bases. These knowledge bases contain knowledge at different levels of abstraction and are built using some of the ideas proposed in [2,4]. We also adopted a subset of SQL [8] for query formulation that eliminates most of the language features that make it difficult to be used by non-experienced and casual users. Particularly, join conditions, when required, are automatically generated based on rules stored in the knowledge bases [12,13].

2. A knowledge-based architecture for cooperative database querying

Our proposal is based on two different and complementary goals in achieving cooperativeness: cooperativeness in the use of a query language, which allows users to reach a higher level of competence in query formulation without necessarily having to understand in detail language features, and cooperativeness in the use of a database, which makes it easier for users to query a database without knowing

Copyright © 1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

+On sabbatical leave at University of Florida, IFAS-IT

From: Proceedings of the Twelfth International FLAIRS Conference. Copyright © 1999, AAAI (www.aaai.org). All rights reserved.
details of its structure, such as relation and attribute names, domains, integrity constraints, etc.

The proposal is knowledge-based in the sense used in artificial intelligence [16]: the knowledge necessary to solve the problem is encoded in knowledge bases in a modular way. The knowledge bases were built through a process of knowledge extraction, structuring and encoding, and must be consistent. In this context, a cooperative interface works as a partner in the querying formulation process, assisting the users in the correct specification of their queries and in their reformulation whenever an empty or unsatisfactory answer is obtained. The proposed architecture is shown in Figure 1, and it consists of:

- A knowledge base (KB1) on the application domain that describes the database schema. This knowledge base stores rules that encode information on relations, relation attributes, attribute domains, relation keys, and domain integrity constraints.

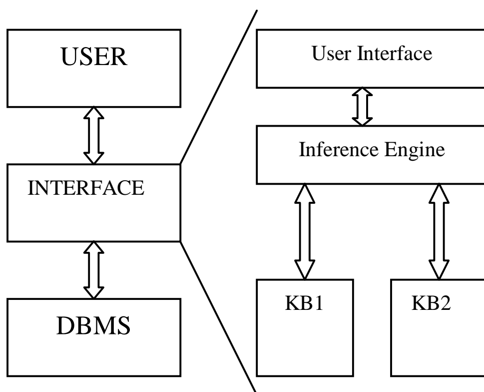

Figure 1. Basic architecture

- A knowledge base (KB2) containing rules that describe the transformations that may be applied to queries to derive alternative answers. KB2 may also contain general rules required, for example, to deal with different types of user, to classify and treat errors, to select appropriate dialogues according to external factors and situations, and so on. In addition to that, specific concepts required to query relaxation [3], such as topics of interest and attribute value ranges, are also described in this knowledge base. The knowledge in KB2 can be considered *meta-level* when compared to the knowledge in KB1, and its correct construction and use makes the system able to have an *active* behavior in the whole process.
- An inference engine or control algorithm that uses both KB1 and KB2 to obtain the rules that must be applied to transform queries whenever an alternative answer is required.
- A user interface that provides cooperativeness at the external level. This interface defines the communication process between users and the database.

The main goal of the proposed architecture is to improve the quality of the interaction between users and the database, aiming at increasing their level of satisfaction with the answers obtained for their queries. Moreover, it provides a great level of independence with regard to external factors, since the users are fully insulated from modifications in cooperative behavior patterns which are restricted to only one of the knowledge bases. In addition, another important goal of this architecture, and therefore of the interface described in this paper, is to make the use of a query language like SQL [8] easier, relieving users from the burden of having to know details of its syntax and of the database structure, as well as some cumbersome details usually not understandable by non-experienced and casual users. The immediate benefits of using a cooperative approach like this are a safer database querying process and an increased user satisfaction, demystifying the use of databases by non-experienced and casual users.

2.1 SQL subset

Our interface adopts a subset of SQL [8] for query formulation. After getting an SQL statement from the user, the interface activates the inference engine that processes the query using the rules stored in KB1 and KB2. In this process, the interface interacts with the user guided by rules stored in KB2, until it gets an answer that finally satisfies the user's needs. Considering the basic format **SELECT**<attribute list> **FROM**<relation list> **WHERE**<selection condition> of the SQL **SELECT** command [8], the following simplifications were made:

- The union, intersection, and difference operators are not allowed.
- Embedded queries [7], i.e. queries using the structure **IS IN (SELECT …)**, are not allowed. They must be replaced by appropriate **AND** expressions [15].
- The selection condition can only be a conjunction of terms. Thus only the connective **AND** is allowed.
- All terms in a selection condition must be in the format <*attribute operator expression*>, where *expression* is an attribute or a constant, and *operator* is one of the relational operators geq, leq, neq, =, > or <. When a term is a join condition, the only operator allowed is the equality, which means that only equijoins [7] can be specified in our interface. However, it is important to note that the join conditions (equijoin conditions) are automatically generated by the interface and need not be specified by the users.
- All attribute and relation names in the database schema must be unique. So any expression R1.A1 = R2.A2 can be unambiguously written as A1 = A2. Although this may be a severe restriction for most real

From: Proceedings of the Twelfth International FLAIRS Conference. Copyright © 1999, AAAI (www.aaai.org). All rights reserved.

world applications, the interface can be modified to accept fully qualified attribute names, overcoming the restriction.

• There is also no need to specify the **FROM** clause. The interface is able to identify from the schema definition the relation names that correspond to the attributes specified in the **SELECT** command. This partly justifies our assumption of unique names, since the interface requires from the user only the specification of the *attribute list* and the conditions that the corresponding attribute values must satisfy in order to appear in the final answer.

2.2 Knowledge-base KB1

The knowledge base KB1 describes the database schema by means of a set of Prolog predicates [6] that define the relations, attributes of relations, relation keys, attribute domains, and domain integrity constraints. These predicates are used to check the semantic consistency of the user's query, i.e. if all relation and attribute names specified in the query are defined in the database schema and if the query satisfies the integrity constraints. Thus given the notation R(A1, A2,...,An), where R is the name of a relation of the database schema and Ai (1=< i =< n), is an attribute name in R, the following predicates must be included in KB1:

R1: For each relation in the database, rel(R), where R is the relation name.
R2: For each attribute of each relation, attrib(R,A), where A is an attribute name of relation R.
R3: For the primary key of each relation, key(R,K), where R is a relation name and K is its primary key.
R4: For each attribute of each relation, domain(A,D) where A is an attribute name and D is a domain name.
R5: For each attribute subjected to a domain constraint, *constraint(condition)*, where *condition* constrains the attribute values. The *condition* must have the format *attribute op value*, where *op* is one of >, <, neq, geq, leq or =.

Figure 2 presents the relational schema of the traveling information database used as example in this paper and some of the predicates included in KB1 [12]. The meaning of the relations and predicates is hopefully infered from the context.

```
flight(flight#,fcompany#,fdeparture,farrival,ffrom,fto,fprice)
train(train#,tcompany#,tdeparture,tarrival,tfrom,tto,tprice)
bus(bus#,bcompany#,bdeparture,barrival,bfrom,bto,bprice)
company(company#,compname,compaddress)
flight-reservation(fticket#,frflight#,frdate)
train-reservation(tticket#,trtrain#,trdate)
attrib(flight, fdeparture)  attrib(flight, farrival)
key(flight, flight#)        domain(flight#, integer)
constraint(fdeparture >=0)
constraint(fdeparture =<24)
```

Figure 2. Example database schema

2.3 Knowledge base KB2

The knowledge base KB2 is a higher level knowledge base as compared with KB1, and contains predicates that define abstract relationships between database objects described in KB1. These predicates define query transformations that may be applied to input queries in order to generate answers containing more relevant and useful information to the users. In fact, these predicates are meta-predicates [15] in the sense that they are in a higher level when compared to those in KB1. These higher level predicates include predicates that define *topics of interest* [4,5], relating attributes of a same relation, predicates that define the concept of *neighborhood* inference among relations [2], predicates that define the concept of proximity between attribute values [12], and predicates that define the possible query transformations.

In what follows, we list the predicates that must be defined in KB2:

R6: topic(T)
R7: top-attrib(T,A)
R8: interval(A,V)
R9: top-int(Q,T):-appears(Q,A),top-attrib(A,T)
R10: relev-attrib(Q,R,A):-rel(R),top-int(Q,T), attrib(A,R),top-attrib(A,T)
R11: add-attrib(Q,A):-rel(R),relev-attrib(Q,R',A), isa(R,R').
R12: relev-rel(Q,R1,R2):-rel(R),isa(R1,R),isa(R2,R).
R13: relev-value(Q,V):-appears(Q,A),interval(A,V).
R14: imp-attrib(A,B):-topic(T),top-attrib(A,T), top-attrib(B,T).

Predicates R6 and R7 are used to relate each attribute A to a specific topic of interest T. Predicate R8 is used to define the acceptable interval V for relaxing the range of a condition on attribute A, when there is no answer to a query. Predicates R9 to R14 are used to define the possible transformations that may applied to a query in order to expand its answer. For example, predicate R9 states that if the attribute A appears in a query Q and this attribute is related to topic T, then topic T is a topic of interest (related topic) to query Q. Thus all attributes related to T must also be retrieved and exhibited in the answer for Q. A more detailed description of these predicates is out of the scope of this paper and can be found in [13].

2.4 The inference engine

The inference engine is the component of the architecture responsible for controlling the querying process. It uses knowledge from both KB1 and KB2, making it possible to rewrite the original query in order to get more relevant answers to the user. The high level algorithm presented below shows how the inference engine works. A more detailed description of this algorithm can be found in [12].

The first step in the querying process is the checking of the attribute list and the selection condition specified by

the user. In the second step, the interface tries to expand the list of attributes by using rules in KB2 that define semantic associations between attributes based on the concept of topic of interest [4,5]. At the end of this step, a correct SQL query is generated and it is passed to the database management system to be executed.

```
BEGIN {engine}
GET the SQL query from the user
CHECK the query syntax and semantic consistency using schema definition rules
WHILE the query is not correct DO
BEGIN {query correction}
   PRINT errors and possible correction actions
   ACCEPT corrections
   CHECK the query syntax and semantic consistency using schema definition rules
END {query correction}
BEGIN {expansion}
   EXPAND the list of attributes by using the concept of topic of interest
   EVALUATE the query
   IF the query has an answer
   THEN show the answer to the user
   ELSE BEGIN {relaxation}
        RELAX the selection condition by using the concept of attribute proximity
        EVALUATE the query
        IF the query has an answer
        THEN show the answer to the user
        ELSE BEGIN {abstract hierarchy searching}
             SEARCH for alternative concepts using type abstraction hierarchies
             EVALUATE the query
             IF the query has an answer
             THEN show the answer to the user
             ELSE PRINT ``no answer''
            END {abstract hierarchy searching}
        END {relaxation}
END {expansion}
END {engine}
```

Figure 3. Inference algorithm

A third step is performed when either there is no answer to the generated query or the user is not satisfied with the resulting answer. In this step, the *is-a* hierarchy is used to try to find a related concept that could also be a useful answer to the user. The decision of postponing the use of *is-a* relationships until this point was based on the assumption that it is more cooperative to keep the context established by the user in the original query as far as possible. The fourth and last step is to try alternative answers by using the concept of attribute relaxation. The query modifications are based on rules stored in KB2 that define the concepts of attribute proximity [2]. Such rules allow the relaxation of selection conditions (for example, by enlarging the range of values that some attributes may have in the answer).

3. Query examples using the interface

In this section, we present two query examples that illustrate the major features of our interface. Additional examples can be found in [12].

Query 1:

```
| ?- select([frdate,ffrom],where([fdeparture>10:00])).
 Original query:
  SELECT frdate, ffrom
  FROM flight-reservation, flight
  WHERE fdeparture > 10:00
 Modified query with additional attributes:
  SELECT frdate, fdeparture, farrival, ffrom, fto
  FROM flight-reservation, flight
  WHERE fdeparture > 10:00
 Modified query with relaxed conditions:
  SELECT frdate, fdeparture, farrival, ffrom, fto
  FROM flight-reservation, flight
  WHERE fdeparture > 08:00
*********** Query Answer *******************
```

frdate	fdeparture	farrival	ffrom	fto
21/09/95	09:00	10:30	B. Horizonte	S. Paulo
18/11/95	09:00	10:30	B. Horizonte	S. Paulo
20/12/95	08:30	09:30	B. Horizonte	Rio

This query was correctly specified by the user and the interface proceeds by executing it. In the first step, it modifies the query by expanding the list of attributes based on the concept of topic of interest. Attributes fto, fdeparture, and farrival were added to the query in this step. As the modified query does not have an answer in the database, a second transformation step is carried out. In this step, the selection condition of the **WHERE** clause is relaxed by replacing the condition (fdeparture > 10:00) by (fdeparture > 08:00). The final query then results in an approximate answer to the user.

Query 2:

```
| ?- select([flight#,ffrom,fprice],where([fdeparture>16:00])).
 Original query:
  SELECT flight#, ffrom, fprice
  FROM flight
  WHERE fdeparture > 16:00
 Modified query with additional attributes:
  SELECT flight#, fdeparture, farrival, ffrom, fto, fprice
  FROM flight
  WHERE fdeparture > 16:00
 Modified query with relaxed conditions:
  SELECT flight#, fdeparture, farrival, ffrom, fto, fprice
  FROM flight
  WHERE fdeparture > 14:00
 Modified query with alternative relations:
  SELECT train#, tdeparture, tarrival, tfrom, tto, tprice
  FROM train
  WHERE fdeparture > 16:00
************** Query Answer ********************
```

train#	tdeparture	tarrival	tfrom	tto	tprice
4	17:00	17:00	B. Horizonte	Salvador	40.00
5	18:00	16:00	B. Horizonte	P. Alegre	60.00

In this example, the query was also correctly specified by the user. The interface then modifies the query by expanding the list of attributes with fto, fdeparture, and farrival. However, this modified query does not have an answer in the database. Thus a second transformation step is performed by the interface by relaxing the selection

condition of the **WHERE** clause. The new modified query also does not have an answer in the database, and a new and final transformation step is performed to replace the relation specified in the query. For this, the interface uses the concept of neighborhood inference and the relation flight is then replaced by the relation train. This transformation requires the attribute names in the **SELECT** and **WHERE** clauses to be modified accordingly, using KB2 rules. The modified query is then executed, finally generating an approximate but relevant answer to the user.

4. Concluding remarks

In this paper, we presented partial results of a project aimed at providing cooperativenes in relational database querying. By cooperativeness we mean relieving the user from both understanding details of the query language and knowing the database structure.

Our approach is based on artificial intelligence techniques taken, particularly, from the knowledge-based systems area [16]. To evaluate the proposed architecture, we implemented a cooperative interface based on it, that used some of the ideas proposed in [2,4]. The implementation was carried out using Prolog in a UNIX environment.

Although the system was not extensively tested in a real environment, preliminary results have shown that the two-level knowledge base architecture [15] provides a general framework, with a richer set of query transformation rules, considerably improving the database querying process by non-experienced and casual users. Moreover, the modularity of our approach makes it general and flexible enough to be applied to other problems that might need solutions based on cooperativeness.

References

[1] Cha, S. K. Kaleidoscope: A Cooperative Menu-Guided Query Interface (SQL Version). *IEEE Trans. on Knowledge and Data Engineering* 3 (1), March 1991.

[2] Chu, W. W., Chen, Q. and Lee, R. Cooperative Query Answering via Type Abstraction Hierarchy. In Deen, S.M. (ed.). Cooperating Knowledge Based Systems. Springer-Verlag, London, 1991.

[3] Chu, W. W., Chen, Q. and Page Jr., T. W. CoBase: Cooperative Distributed Databases. *Proc. of the 6th Brazilian Symposium on Databases*, 1991.

[4] Cuppens, F. and Demolombe, R. Cooperative Answering: a Methodology to Provide Intelligent Access to Databases. *Proc. of the 2nd International Conference on Expert Database Systems*, 1988.

[5] Cuppens, F. and Demolombe, R. Extending Answers to Neighbor Entities in a Cooperative Answering Context. *Decision Support Systems* 11(1), January 1991.

[6] Clocksin, W.F. and Mellish, C.S. Programming in Prolog. Springer-Verlag, Berlin, 1995

[7] Elmasri, R. and Navathe, S. B. Fundamentals of Database Systems, 2nd ed. Benjamin/Cummings, Redwood City, California, 1994.

[8] Melton, J. and Simon, A. R. Understanding the New SQL: a Complete Guide. Morgan Kaufmann, San Francisco, California, 1993.

[9] Motro, A. FLEX: A Tolerant and Cooperative User Interface to Databases. *IEEE Trans. on Knowledge and Data Engineering* 2(2), June 1990.

[10] Mylopoulos, J., Papazoglou, M. Cooperative Information Systems. *IEEE Intelligent Systems* 12(5), Sept-Oct 1997.

[11] Papazoglou, M.P., Schlageter, G., eds.: Cooperative Information Systems: Trends and Directions. Academic Press, San Diego, California, 1998.

[12] Ramos, C. V. A Cooperative Interface for Relational Database Querying. MSc Thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 1996. (in Portuguese)

[13]Ramos, C.V., Braga, J.L., Laender, A.H.F. Cooperative Querying in Relational Databases. *Proc. of the XVII International Conference of the Chilean Computer Science Society*, 1997.

[14] Rettig, M. Cooperative Software. *Communications of the ACM* 36(4), April 1993.

[15] Silva, E. C., Laender, A. H. F. and Braga, J. L. Relational Database Query Optimization Driven by a Multi-level Knowledge Base. *Revista Brasileira de Computação* 4, 1990. (in Portuguese)

[16] Stefik, M. Introduction to Knowledge Systems. Morgan-Kaufman, San Francisco, CA, 1995.

[17] Wiederhold, G., Genesereth, M. The Conceptual Basis for Mediation Services. *IEEE Intelligent Systems* 12(5), Sept-Oct. 1997.