TOWARDS A FLEXIBLE DATABASE INTERROGATION

Ines Fayech[1] and Habib Ounalli[2]

[1]Department of Computer Sciences, Faculty of Sciences Tunis, Tunisia
inesfaiech@yahoo.fr
[2]Department of Computer Sciences, Faculty of Sciences Tunis, Tunisia
habib.ounalli@fst.rnu.tn

ABSTRACT

In this paper, we are interested in the use of domain ontologies as a semantic enrichment for traditional databases. Our first aim is to help the user in his search when his initial query doesn't return any result. So, we propose a solution based on two different approaches allowing the user to express his interrogation in a relatively free way. The first approach detects and resolves naming and schematic conflicts. It is an ontological approach for SQL query expansion generating a set of queries. The second one is a join detection approach to eventually add all missed constraints in each generated query.

KEYWORDS

Domain ontology, Relational database, Query expansion

1. INTRODUCTION

Relational databases become increasingly complex on schema and data. Database with hundreds of relations are common in many companies. Although traditional database management systems are efficient to organize data, their semantic poverties prevent them from satisfying the increasing needs of users. This gives rise to crucial needs for methods and tools for helping users in their search.

The concept of ontology, defined by Gruber [1] as "an explicit and formal specification of a conceptualization being the subject of a consensus", appears in a good place for the realization of these solutions.

In this paper we are interested in the use of ontology as a semantic enrichment for traditional databases. Particularly, we address the problem of semantically querying a single database using domain ontology. Our first aim is to help the user in his search when his initial query doesn't return any result.

A wide variety of works has exploited ontologies to resolve query processing problems in various database environments including semantic query optimization [2] and data integration.

Semantic query optimization approaches reformulate a given query into another equivalent one that uses less time and/or resources during the execution. So, the initial query and the reformulated one should return the same answers.

This condition should be relaxed when the initial query doesn't return any answer. In this case, returning a flexible answer to the user is better than not receiving any answer at all.

The initial query is a SQL query which can be defined by a set of selections and projections over database objects satisfying a set of conditions. Moreover, we suppose that all the attributes in this query are prefixed with their table names.

DOI: 10.5121/ijdms.2012.4302

When this query fails, there might be terms in the database that are syntactically different from one another but semantically related to the user terms and that express the same intention of the user. We address this issue as a conflicts problem.

In [3], we have proposed an ontology based approach in order to detect and resolve naming, aggregation and generalization conflicts. It is an ontological approach for SQL query expansion allowing the user to express his interrogation in a relatively free way. If the initial query doesn't have any result, it will be reformulated by the proposed approach in order to better satisfy the user. The result is a set of queries, which might not necessary be equivalents to the original one. But, their answers should be included in the set of relevant solutions to the user.

This proposed approach may not satisfy the intention of the user. Indeed, for each generated query having two or more tables in the FROM clause, there might be some missing join conditions between the considered tables.

In this paper, we extend the previous work by adding a second approach which is a join detection approach applied for each generated query to eventually add all missed join conditions.

The rest of this paper is structured as follows. First, section two describes different classification of query expansion mechanisms. Second, section three is devoted to the presentation of the domain ontology used in our approach. Next, section four gives in details the query expansion approach. The join detection approach is described in section five. An example illustrating our proposal is presented in section six. Finally, a discussion of future work and concluding remarks will be presented.

2. CLASSIFICATION OF QUERY EXPANSION MECHANISMS

Different works was proposed in the literature in order to improve the relevance of search by using ontologies in the query expansion mechanism [4], [5], [6], [7]. These works can be distinguished by the semantic relationships used.

Synonymy, specialization, generalization and meronymy relationships are the most used in the literature.

-The expansion by synonymy consists in replacing some terms present in the initial query by their synonyms. Consequently, it generates a new query which is semantically equivalent to the initial one.

-The expansion by specialization represents a specialization of the initial query. As a consequence, the set of results generated by the new query is included in the set of relevant results for the user. This refinement was exploited in the case of an empty response to the initial query [5], [6] or when the user obtained too many answers for his query.

- The expansion by meronymy consists in replacing some concepts of the initial query by their meronyms present in the ontology.

-The refinement by generalization consists in replacing some concepts of the initial query by more general concepts present in the ontology. This refinement was exploited in [5] where a process of generalization of the initial query is started if it doesn't have any answer.

A query reformulated by generalization gives more general and less precise solutions compared to the user's needs. So, in the refined query, some generalized concepts can contain information which contradicts the specialized concepts present in the initial query. Since the user could be provided only with the answers which are relevant to his query, we choose to not apply this refinement in our work. So we use only expansion by synonymy, meronymy and specialization.

Another classification of the query expansion mechanisms was proposed in the literature according to the user's implication in the reformulation. The interactive expansion and the automatic expansion can be distinguished.

In information retrieval, the majority of query expansion approaches use the interactive expansion [6], [7]. For example, in [6] domain ontology was used in an interactive query expansion approach to solve a problem of information retrieval in bioinformatics domain. The authors don't specify any decision criterion of possible enrichment and they leave the choice to the user. This later can't choose the adequate refinement type without having an idea about the result of each reformulation.

In order to improve the results without implicating the user, we choose to apply an automatic expansion.

3. DOMAIN ONTOLOGY

The key technology involved here are ontologies that provide explicit representation of data semantics with data. We choose to explore domain ontology of the database describing the semantics of the terms of the query in order to identify their equivalents in the considered database.

Although the choice of domain ontology presents many advantages compared to a generic ontology, it causes the problem of its construction. This problem was treated in the literature and various solutions were proposed according to the need [8], [9], [10].
In [3] and [11] we have proposed two different approaches for building a domain ontology describing all the database content.

In this paper, we give only the specification of the domain ontology. This later is developed in OWL DL which present some limitations related to the representation of the lexicalization of a concept. Indeed, in OWL we use the rdfs: label annotation property to represent the terms denoting a concept. So, we can't add a property to a term nor a relation between terms.

This problem was treated in the literature and different models were proposed in order to represent the terminological part of an ontology [12], [13].

These models don't give the same importance to the notions of concept and term. For more details, readers are recommended to refer to [14] where the authors have presented a meta-model to model terms in OWL DL. They propose to distinguish terms from concepts: concepts are subclasses of the DomainThing class and terms are subclasses of the Term class. Their meta-model can't distinguish between properties concerning a term and properties concerning its instances and doesn't consider how to associate a term to a semantic relation.

In order to manipulate a term as easily as a concept, we represent both in the form of an owl: Class as it is considered in [14] and [15]. Consequently, the meta-model related to the domain ontology is based on two generic concepts: "c_domain_concept" and "concept_term" which represent respectively, the concepts of the considered domain and their lexicalizations.

Each "concept_term" denotes one domain concept ("c_domain_concept"). The later is divided into two sub-concepts in order to distinguish the concepts representing tables ("concept_table") and those describing attributes ("concept_attribute").

Since in OWL DL, a relation can be defined only between two individuals or between an individual and a literal, we choose to model terms as instances of the class "concept_term".

Furthermore, concepts describing tables and those describing attributes will occur as instances, respectively, of the "concept_table" and "concept_attribute" classes.

We consider two kinds of hierarchic relations between the domain concepts: the hyponymy (is-a relation) used to structure the concepts representing tables ("concept_table") and the meronymy (part-of relation) applied between the concepts describing attributes ("concept_attribute"). Figure 1 shows the specification of the Meta-ontology by using OntoViz, the visualization plugin of Protégé.

We describe the domain ontology in Horn clauses. So, we consider the following predicates representing the concepts and their relationships:

- Concept_attribute(C) means that C is a concept representing an attribute.

- Concept_table(C) means that C is a concept representing a table.

- Concept_term (T) means that T is a term.

- denote(C, T) means that the term T denotes the concept C.

- attribute_of(CA, CT) means that the concept attribute CA is an attribute of the concept table CT.

- PK(CT,CA) means that the concept attribute CA is a primary key of the concept table CT.

- FK(CT,CA) means that the concept attribute CA is a Foreign key of the concept table CT.

- CONSTRAINT_FK(CT1, CT2) means that there is a foreign key in the concept table CT1 that references the concept table CT2.

Moreover, we consider the predicate DB_ELT() applied to each concept in the DO ontology. DB_ELT(C) means that the concept C describes a database element. This predicate is useful to distinguish the concepts describing the content of the database (tables and attributes) from the others.

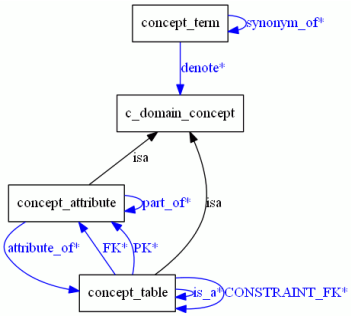

Figure1. The meta-model of the Domain ontology

We consider three semantic relations: the synonymy, the specialization and the meronymy. These semantic relations can obtain different properties which allow drawing additional conclusions from the presence of certain relations. Symmetry, reflexivity and transitivity are the important properties.

3.1. The synonymy relation

The synonymy between two terms is expressed by the predicate synonym(). Two terms T1 and T2 are synonyms if they denote the same concept.

synonym(T1, T1) ←

synonym(T1, T2) ← concept_term(T1), denote(C, T1), concept_term(T2), denote (C, T2)

This semantic relationship verifies the following properties:

Symmetry: synonym(T1, T2) ← synonym (T2, T1)

Transitivity: synonym(T1,T2)←synonym(T1,T3), synonym(T3, T2)

3.2. The semantic relationship of specialization

The semantic relationship of specialization between two concepts C1 and C2 expresses that the concept C1 specializes the concept C2. We use the predicate C-Spec() to check this relationship. This predicate verifies the following property of Transitivity:

C-Spec(C1, C2) ← C-Spec(C1, C3), C-Spec(C3, C2)

3.3. The semantic relationship of meronymy

A meronymic relation between a concept C and its meronym C1 is expressed by the predicate C-Mer(C, C1).

Since in several works in the literature meronymic relation is considered as being a transitive relation, the authors of [16] consider transitive and intransitive meronymy relations.

In our work, we use only transitive meronymy expressing the relation of "component of" (such as" a person's finger is a component of a person"). So, we avoid intransitive meronymy such as relation expressing "organizational part of" (such as "a person is a member of Information science department").

C-Mer(C1, C2) ← C-Mer (C1, C3), C-Mer (C3, C2)

The choice of transitive meronymy allows us to consider more than one hierarchical level in the expanded query.

4. THE QUERY EXPANSION APPROACH

Our objective is to help the user of a single database when his query doesn't have any answer. So, we propose an ontological approach for SQL query expansion.

We consider an automatic enrichment based on synonymy, specialization and transitive meronymy relationships. This semantic links are described in a domain ontology DO covering the domain of the considered database. The expansion can be applied to the tables and/or the columns (attributes). It is not uniform for all the terms of the query.

Necib and Freytag [17] have proposed a system that is most closely related to our work. It is an ontological approach for query processing within a single relational database management system. These authors suppose the pre-existence of an ontology describing the semantic instances of the considered database. Their approach is limited to some particular databases (such as product databases) where some attributes are enumerated from a list of terms.

The distinguishing aspects of our work from the above are:

- We consider an ontology describing the database schema and not the database instances.

- We apply our approach when the initial query doesn't return any answer.

- We propose an automatic expansion described by a generic algorithm which can be applied with any relational database.

Before presenting the proposed algorithm, we will describe the expansion of the columns and that of the tables.

4.1. The expansion of the columns related to a table

The expansion of a set Col of columns related to a table T consists in determining a set LC of terms in relationship with them and representing attributes of the table T in the database.

Considering the concept table CT denoted by the concept term T in the DO ontology.

We apply the synonymy or the transitive meronymy relationships for each term C in Col which is in the DO ontology and doesn't denote a concept attribute of CT.

If there exists a term C1, synonym of C, and denoting a concept which is an attribute of CT, we replace C by C1 in the query.

Else, if C is a term in the SELECT clause of the initial query, we consider the set LC of concept terms denoting concepts which are attributes of CT. We eventually replace C by its meronyms in LC.

We describe this type of expansion by the "columns_expansion" procedure.

```
columns_expansion(T,CT, Col, ncol, DO, LC)
Begin
LC= {}
Ok=true
For each C in Col do
LC1={}
If C is not in DO then ok=false
else
If (C denotes a concept attribute CA in DO Then
If CA is a concept attribute of CT) then
If DB_Elt(C) = True then LC1 = {C}
Else
LC1 = {C1/ Concept_term(C1) and synonym( C, C1) and DB_Elt(C1)}
End_if
Else
If C is a term in the SELECT clause then
*/ LC1 is the set of terms, meronyms of C, which are database element and attribute of T*/
LC1 = {AT/ Concept_term(AT) and denote(AT, CA1) and C_Mer(CA, CA1) and DB_Elt(AT
and attribute_of(CA1, CT)}
end_if
If LC1= {} then ok=false End_if
End_if
Else ok=false
End_if
End_if
LC= LC ∪ {(C, LC1)}
End_for
Return (ok)
End
```

4.2. The expansion of a table

The enrichment of each table Tab, in the initial query, is the first expansion step in our approach. It is applied when the term Tab introduced in the FROM clause is a concept term denoting a concept table CT in the DO ontology.

Firstly, we have to check if the term Tab is a database element in the DO ontology. If it isn't the case, it is necessary to replace the term Tab by other terms referring to tables in the database and which are semantically related to it. We distinguish two expansion types:

The first one is the expansion of the table by synonymy applied when the concept CT is a database element and the term Tab is not a database element. This refinement is used by the "get_correspd_table" procedure which can generate a new term Tab1, synonym of Tab and representing a database element.

If the concept table CT is not a database element, the second type of expansion based on the specialization relationship is applied. It is described by "get_specialized_tables" procedure which can generate a set LTab of terms which are database elements. Each concept term in LTab denotes a concept table which is a specialization of CT.

The expansion of the columns, applied to each element of LTab, generates new queries whose execution satisfies the user.

The expansion of a table is described in by the "Table_expansion" procedure.

```
Table_expansion (Tab, DO, CT, LTab)
Begin
LTab = {}
If Tab is in DO then
If Tab denotes a concept table CT then
If DB_Elt(Tab) = True then
LTab = {Tab}
Else
If DB_Elt(CT) = True then
get_correspd_table (Tab, CT, DO, Tab1)
LTab = {Tab1}
Else get_ specialized_table (Tab, CT, DO, LTab)
end_if
End_if
End_if
End
```

The expansion of a table and its attributes is described in by the "Reformul_table" procedure.

```
Reformul_table(ncol, Col, Tab, DO, LReq )
Begin     LReq = Null
Table_expansion (Tab, DO, CT, LTab)
If LTab <> {} then
For each T in LTab do
Ok= columns_expansion(T,CT,Col, ncol, DO, LC)
If ok then Insert(LReq, Tab, T, LC) End_if
End_for  End_if
End
```

4.3. The query expansion algorithm

Our query expansion approach is described by an algorithm named Reformulation. This later is elaborate in a generic way in order to be reusable with any relational database.

The inputs of the algorithm are a domain ontology (DO) and an SQL query (ReqInit) with an empty answer. The result is a list LReq_Ref of queries resulting from the expansion of the initial query. The union operator applied to all the queries in the LReq_Ref list, combines their output into a single result set satisfying the user.

We start by applying the "extract_from_query" procedure in order to extract, from the initial query, the columns related to each table. For each table, we apply the "Reformul_table" procedure to generate a set of queries.

```
ALGORITHM Reformulation
INPUTS: ReqIni, DO
OUTPUT: LReq_Ref
BEGIN
```

```
LReq_Ref= Null
extract_from_query(ReqInit, ncol, Col, nTab, Tab)
For (i=1; i<=nTab; i=i+1)
{
 Reformul_table(ncol[i],Col[i],Tab[i],DO, LReq[i])
}
Rebuild_ReqInit(LReq,ReqInit, nTab, LReq_Ref)
END
```

Although each generated query doesn't contain naming, aggregation or generalization conflicts [3], it may not either satisfy the intention of the user. This is due to the missing join conditions between the considered tables.

Indeed, when we consider two or more tables in the query, we have to add all join conditions which relate them. This problem is resolved by the join detection approach described in the following section.

5. JOIN DETECTION APPROACH

The previous approach for query expansion generates a set of queries. For each generated query having two or more tables in the FROM clause, we apply the join detection approach to eventually add all the missing join constraints.

Since we have specified database key constraints in the domain ontology, we can explore then in order to detect all missed constraints in a given query. So, we propose a graph-based formalism to represent join constraints between the concepts describing the tables ("concept_table") in the domain ontology.

In this section, we start by describing the construction of the graph from the domain ontology. Then, we define key constraints rules. Before presenting the join detection algorithm, we will present the shortest path algorithm used in our approach.

5.1. The dependency graph

The join detection approach is based on a directed graph constructed from the domain ontology. It is a directed graph called a dependency graph G(V,E), where V is a finite set of nodes and E is a finite set of edges.

Each node of V is labelled by a concept C describing a database table in the ontology. So, the concept C verifies the following predicates: Concept_table(C) and DB_Elt(C).

Nodes are interconnected by edges of type CONSTRAINT_FK. If there is a foreign key constraint between two concepts C1 and C2 (CONSTRAINT_FK (C1, C2)), then an edge e = (C1, C2) is created between their corresponding nodes.

5.2. The key constraints rules

Our aim is to detect join constraints in a SQL query. A join is a relationship established between keys (primary key or/and foreign key) in two different tables. It corresponds to a WHERE clause condition in a SQL statement.

Two common types of join can be distinguished: the first type links the primary key in one table to the foreign key in another table. The second one relates the primary key in one table to the primary key in another table.

Three rules can be applied in order to detect join constraints in a query. Rule1 and Rule2 express constraints of type (primary key, foreign key) and Rule3 expresses constraints of type ((primary key, primary key).

5.2.1. Rule1

This rule is applied when we consider two nodes, labelled by two concepts C1 and C2, and directly related by an edge (C1 has a foreign key corresponding to the primary key of C2). We have to add the constraint C1.CAi=C2.CAj Where:
CAi is a foreign key of C1 (FK(C1, CAi)) and CAj is a primary key of C2 (PK(C2, CAj)).

5.2.2. Rule2

This rule is applied when we consider two nodes, labelled by two concepts C1 and C2, and directly related by two edges (C1 has a foreign key corresponding to the primary key of C2 and C2 has a foreign key corresponding to the primary key of C1).

We have to add two constraints: C1.CAi=C2.CAj and C2.CAl=C1.CAk Where:
CAi is a foreign key of C1 ((FK(C1,CAi)), CAj is a primary key of C2 (PK(C2, CAj)), CAl is a foreign key of C2 (FK(C2,CAl)) and CAk is a primary key of C1 (PK(C1, CAk)).

5.2.3. Rule3

This rule is applied when we consider two nodes, labelled by two concepts C1 and C2, and the shortest path between them passes through a node labelled by a concept C3. Moreover, C3 has a primary key composed of two attributes referring to the primary keys of C1 and C2

We have to add two constraints: C1.CA1=C3.CA3 and C2.CA2=C3.CA4 Where
CA1 is the primary key of C1 (PK(C1, CA1)),
CA2 is a primary key of C2 (PK(C2, CA2)),
CA3 and CA4 are primary keys of C3 (PK(C3, CA3)) and (PK(C3, CA4)),
And CA3 and CA4 are foreign keys of C3 referencing respectively, C1 and C2 (FK(C3,CA3)) and (FK(C3, CA4)).

5.3. The shortest path algorithm

We need to find a path through the graph G' that passes through all the nodes labelled by each concept of C. A path is a sequence of consecutive edges in a graph.

Many algorithms are proposed in the literature to find a path in a graph such as the shortest path between two nodes [18].

In our approach, we have to reduce the number of nodes in the generated path in order to minimize the number of join constraints in the query. So we propose an algorithm (Shortest-paths) based on the DIJSKTRA approach [18] to find the shortest path connecting all the concepts of C.

Shortest-paths (G: graph, C: set of nodes)
Const: Max=1000
Begin
Choose s in C
S ={s}, ARC = { }, Cost(s) = 0
FOR each x in V DO Cost (x) = Max END_FOR
WHILE ((S ∩C <> C) and (Cost (xsolved) < Max)) DO
FOR each node x in (V- S), adjacent to xsolved DO
/ two nodes are adjacent if there is a common edge between them/
IF Cost(x)> Cost (xsolved) THEN
Cost(x) = Cost (xsolved) +1
ARC =ARC ∪ {(xsolved, x)}
END_IF
END_FOR

Xmin ={x in V-S/Cost(x) = {minimum of (Cost(y), y in V-S}}
IF (Xmin ∩ C = {}) THEN Choose new_xsolved from Xmin
ELSE Choose new_xsolved from Xmin ∩ C
END_IF
Xsolved = new_xsolved
S =S ∪ {xsolved}
END_WHILE
END

5.4. Join detection algorithm

We present here the join detection algorithm. The inputs are the directed graph G and an SQL query.

We consider the set C of concepts, in the ontology, describing all the tables in the FROM clause of the considered query.

G' is the undirected graph corresponding to the directed graph G.

1- Find the shortest path in G', that passes through all the nodes labelled by each concept of C.
2- Apply Rule1, Rule2 or Rule3 for each pair of concepts in the path to generate a set CONSTR of constraints.
3- Replace each concept in CONSTR by its corresponding table in the database in order to find join constraints.

6. ILLUSTRATION OF THE PROPOSED APPROACH

In this section, we give an example to illustrate the effectiveness of the proposed approach.

Considering a fragment of a relational database schema describing the faculty of sciences:

contract (identifier, name, speciality, city, Numdep)
PKey (identifier), FKey(Numdep) REFERENCES dept
permnt(roll, name, address, speciality, rank, Ndep)
PKey (roll), FKey(Ndep) REFERENCES dept
dept (Ndep, named) , PKey(Ndep)
section(NumS, sectionName, Ndep)
PKey(NumS), FKey(Ndep) REFERENCES dept
module(referenceM, NameM)
PKey(referenceM)
composed(NumS, referenceM)
PKey(NumS), FKey(NumS) REFERENCES section
FKey(referenceM) REFERENCES module

We give here a fragment of the domain ontology DO covering the domain of the considered database (CTi, (i=1..7) is a Concept_table and CAj (j=1..11) is a Concept_attribute).

denote(CT1, 'teacher') ←
denote(CT1, 'professor') ←
denote(CT2, 'permanent') ←
denote(CT2, 'permnt') ←
denote(CT3, 'contractual') ←
denote (CT3, 'contract') ←
denote(CT4, 'department') ←
denote(CT4, 'dept') ←
denote(CT5, 'composed') ←
denote(CT6, 'module') ←
denote(CT7, 'section') ←
denote(CA1, 'speciality') ←

denote(CA2, ‘identifier’) ←
denote(CA3, ‘address’) ←
denote(CA4, ‘city’) ←
denote(CA5, ‘street’) ←
denote(CA6, ‘code’) ←
denote(CA7, ‘roll’) ←
denote(CA8, ‘Ndep’) ←
denote(CA9, ‘referenceM’) ←
denote(CA10, ‘NumS’) ←
denote(CA11, ‘Numdep’) ←
C-Spec(CT1, CT2) ←
C-Spec(CT1, CT3) ←
synonym (‘speciality’, ‘disciplines’) ←
synonym (‘identifier’, ‘roll’) ←
synonym (‘Ndep’, ‘number’) ←
C-Mer(CA3, CA4) ←
C-Mer(CA3, CA5) ←
PK(C T3, C A2) ←
PK(CT2, C A7) ←
PK(CT4, CA8) ←
PK(C T5, C A9) ←
PK(CT5, C A10) ←
PK(CT6, CA9) ←
PK(CT7, C A10) ←
CONSTRAINT_FK(CT2, CT4) ←
CONSTRAINT_FK(CT3, CT4) ←
CONSTRAINT_FK(CT7, CT4) ←
CONSTRAINT_FK(CT5, CT7)←
CONSTRAINT_FK(CT5, CT6)←
FK(CT2, CA8)←
FK(CT3, CA11) ←
FK(CT7, CA8) ←
FK(CT5, CA10) ←
FK(CT5, CA9) ←

Figure 2 shows the dependency graph G corresponding to the previous fragment of the domain ontology DO.

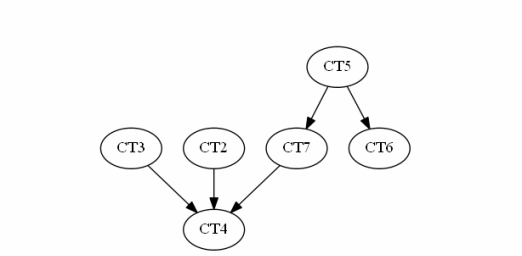

Figure 2.The dependency graph G

Considering the following queries Q1 and Q2:

Q1: SELECT professor.name, professor.address FROM professor WHERE professor.disciplines= ‘info’

Q2: SELECT module.name FROM module, department WHERE department.name=’computer science’

As they are formulated, these queries don't return any answers and their executions generate errors. We will correct them in order to satisfy the user.

Q1: The term "professor" doesn't have an equivalent term in the database. A specialization of this tem generates the terms "contract" and "permnt" referring to tables in the database. They have an attribute named "name" and another named "specialty" synonymous of "disciplines". The table "permnt" has an attribute "address" and the table "contract" has an attribute "city", meronyms of "address. The query expansion approach generates the following queries (NQ1 and NQ2): NQ1: SELECT contract.name, contract.city FROM contract WHERE contract.speciality= 'info' and NQ2: SELECT permnt.name, permnt.adress FROM permnt WHERE permnt.speciality= 'info'. Each generated query contains one table in the FROM clause. So, we haven't to apply the second approach. As a consequence, the union of NQ1 and NQ2 is a query satisfying the user.

Q2: This query includes two terms in the FROM clause which are "module" and "department". The former is a database table and the later refers to the table "dept" in the database. The query expansion approach generates the following query NQ: SELECT module.NameM FROM module, dept WHERE dept.named='computer science'

This query doesn't satisfy the user because of the missing of join constraints.

In the FROM clause, we have two tables named 'dept' and 'module' and represented in the ontology, respectively, by the concepts CT4 and CT6. We apply the join detection approach with the set C= {CT4, CT6}.

The shortest path between CT4 and CT6 is the sequence of the following consecutive edges: (CT4, CT7), (CT7, CT5), (CT5, CT6).

The application of the different rules generates the following constraints: CT7.CA8=CT4.CA8 (Rule1) and CT5.CA10=CT7.CA10 and CT5.CA9=CT6.CA9 (Rule3).

The new query is:

SELECT module.NameM FROM module, dept, section, composed WHERE dept.named='computer science' and section.Ndep=dept.Ndep and composed.Nums = section.Nums and composed.referenceM= module.referenceM

7. CONCLUSION

In traditional database management system, the user's query is treated at only syntactical level. So, if the user has limited knowledge or no knowledge at all about the database content, he can't obtain an answer to his query.

In this paper, we have defined an approach to overcome this limitation and improve the answers of database queries by detecting and resolving naming and schematic conflicts. The proposed solution includes two approaches based on a domain ontology constructed from the database schema. The first one is a query expansion approach used in the case of an empty response to the initial query in order to generate a set of queries. The second approach is a join detection approach to detect all missed constraints in each generated query.

The domain ontology is constructed using OWL language and the proposed approach is currently being implemented in JAVA.

In the future work, we intend to extend our proposal by detecting the reasons of failure. This later can be related to some terms in the query (attributes and/or tables) which are not found in the domain ontology.

Structural conflicts can also cause the failure of the proposed approach. We have a structural conflict when the same information is modelled in the database schema and the query in two different ways.

REFERENCES

[1] Gruber T.R., (1993) "A translation approach to portable ontology specification", *knowl. Acquis, vol. 5 num2,* p199-220, Academic Press Ltd.

[2] King, J.J, (1981) "QUIST: A system for semantic query optimization in relational Databases", *In: VLDB*, pp. 510–517.

[3] Fayech I. & Ounalli H., (2012) "An Ontological Approach for SQL Query Expansion", *IEEE 2nd International Conference on Information Technology and e-Services,* Sousse, Tunisia, p.154-159.

[4] Baziz M., Aussenac-Gilles, N., & Boughanem, M., (2003) "Exploitation des liens sémantiques pour l'expansion de requêtes dans un système de recherche d'information", *XXIème Congrès INFORSID*, Nancy, France, p.121-134.

[5] Bidaut A., Froidevaux C. & Safar B., (2002) "Similarity between queries in a mediator", *In proc. of ECAI'02*, pages 235-239,

[6] Messai N., Devignes M., Napoli A. & Smaïl-Tabbone M., (2006) "Treillis de concepts et ontologies pour interroger l'annuaire de sources de données biologiques BioRegistry", *Revue ISI, Ingénierie des Systèmes d'Information: Systèmes d'information spécialisés* 11(1), pages 39–60.

[7] Safar B., Kefi H. & Reynaud C., (2004)) "OntoRefiner, a user query refinement interface usable for Semantic Web Portals", *Application of Semantic Web Technologies to Web Communities Workshop,* August 23rd, 16th European Conference on Artificial Intelligence, Valencia, Spain.

[8] Barrasa J., Corcho Ó. & Gómez-Pérez A., (2004) "R2O, an Extensible and Semantically Based Database-to-ontology Mapping Language", *Ontology Engineering Group*, SWDB.

[9] Bizer C, (2003) "D2R MAP– A Database to RDF Mapping Language", The twelfth international World Wide Web Conference, *WWW2003*, Budapest, Hungary.

[10] Cullot N., Ghawi R. & Yétongnon K., (2007) "DB2OWL: A Tool for Automatic Database-to-Ontology Mapping", *In Proceedings of the 15th Italian Symposium on Advanced Database Systems*, Italy, pp. 491-494.

[11] Fayech I. & Ounalli H., (2012) "An approach for semantically enriching traditional databases", *The International Journal of Information Studies : Vol. 4 issue 1 pages: 26-36.*

[12] SZULMAN S. & BIÉBOW B., (2004) " OWL et Terminae ", *In Actes des 15es journées francophones d'ingénierie des connaissances*, p. 41–52 : Presses Universitaires de Grenoble.

[13] BONTCHEVA K., TABLAN V., MAYNARD D. & CUNNINGHAM H. (2004) "Evolving GATE to Meet New Challenges in Language Engineering", *Natural Language Engineering*, 10(3/4), 349–373.

[14] Axel Reymonet, Jérôme Thomas & Nathalie Aussenac-Gilles (2007) " Modélisation de Ressources TerminoOntologiques en OWL ", *Actes d'IC 2007*: 169-181.

[15] Anita C. Liang, Boris Lauser & Margherita Sini (2006) "From AGROVOC to the Agricultural Ontology Service / Concept Server - An OWL Model for Creating Ontologies in the Agricultural Domain", OWLED.

[16] Katrin Weller & Wolfgang G. Stock, (2008) "Transitive Meronymy. Automatic Concept-Based Query Expansion Using Weighted Transitive Part-Whole Relations.Information", *Wissenschaft und Praxis 59*, 165-170,

[17] Necib,C.B. & Freytag, J (2005) "Query processing using ontologies", *CAiSE 2005*, p.167-186.

[18] E. W. Dijkstra, (1959) "A note on two problems in connexion with graphs", *Numerische Mathematik journal*, volume 1, pages 269-271.