Journal of Enterprise Information Management

A novel method for providing relational databases with rich semantics and natural language processing

Kamal Hamaz, Fouzia Benchikha,

Article information:

To cite this document:
Kamal Hamaz, Fouzia Benchikha, (2017) "A novel method for providing relational databases with rich semantics and natural language processing", Journal of Enterprise Information Management, Vol. 30 Issue: 3, pp.503-525, https://doi.org/10.1108/JEIM-01-2015-0005
Permanent link to this document:
https://doi.org/10.1108/JEIM-01-2015-0005

Downloaded on: 04 June 2019, At: 06:26 (PT)
References: this document contains references to 35 other documents.
To copy this document: permissions@emeraldinsight.com
The fulltext of this document has been downloaded 274 times since 2017*

A novel method for providing relational databases with rich semantics and natural language processing

Kamal Hamaz and Fouzia Benchikha
LIRE Laboratory, University Abdelhamid Mehri Constantine 2, New City Ali Mendjeli, Algeria

Received 5 January 2015
Revised 27 April 2015
Accepted 31 July 2015

Abstract

Purpose – With the development of systems and applications, the number of users interacting with databases has increased considerably. The relational database model is still considered as the most used model for data storage and manipulation. However, it does not offer any semantic support for the stored data which can facilitate data access for the users. Indeed, a large number of users are intimidated when retrieving data because they are non-technical or have little technical knowledge. To overcome this problem, researchers are continuously developing new techniques for Natural Language Interfaces to Databases (NLIDB). Nowadays, the usage of existing NLIDBs is not widespread due to their deficiencies in understanding natural language (NL) queries. In this sense, the purpose of this paper is to propose a novel method for an intelligent understanding of NL queries using semantically enriched database sources.
Design/methodology/approach – First a reverse engineering process is applied to extract relational database hidden semantics. In the second step, the extracted semantics are enriched further using a domain ontology. After this, all semantics are stored in the same relational database. The phase of processing NL queries uses the stored semantics to generate a semantic tree.
Findings – The evaluation part of the work shows the advantages of using a semantically enriched database source to understand NL queries. Additionally, enriching a relational database has given more flexibility to understand contextual and synonymous words that may be used in a NL query.
Originality/value – Existing NLIDBs are not yet a standard option for interfacing a relational database due to their lack for understanding NL queries. Indeed, the techniques used in the literature have their limits. This paper handles those limits by identifying the NL elements by their semantic nature in order to generate a semantic tree. This last is a key solution towards an intelligent understanding of NL queries to relational databases.

Keywords Information retrieval, Ontologies, Natural language processing, Relational databases, Reverse engineering, Enrichment
Paper type Research paper

1. Introduction

A relational database (or RDBMS) is seen as a very effective way to store and structure data of a given domain. Consequently, the relational model has continued being one of the best used models for storing and manipulating data. With the development of technologies the number of users interfacing with databases has increased considerably. However, accessing a database using a specific query language is proved to be inadequate for a large number of users. These last are forced to learn a specific query language such as SQL to retrieve the desired data. The problem arises more specifically with users who have difficulties in using technical database aspects. On the other hand, there exist users who use forms to access some content of a database; unfortunately, their expectations depend on the capabilities of the forms being used. This is seen as a limitation for many database users. As a solution, the researchers have been trying to propose Natural Language Interfaces to Databases (NLIDB) to retrieve data using simple Natural Language (NL) queries. An NLIDB attempts to receive an NL query and converts it back into the corresponding SQL queries. As a consequence, a well-designed and implemented NLIDB system can do the whole work for users, maximizing thus the number of users

Journal of Enterprise Information Management
Vol. 30 No. 3, 2017
pp. 503-525
© Emerald Publishing Limited
1741-0398
DOI 10.1108/JEIM-01-2015-0005

interacting with the databases. However, the usage of existing and commercial NLIDBs is not widespread due to several deficiencies in understanding NL queries. Indeed, NL queries can be very ambiguous because it occurs when the users formulate queries using ambiguous sentences with no grammar rules. The other problem is that the NLIDBs are domain-dependent or language-dependent systems. This is a disadvantage because they cannot be generalized to other databases and are very hard to port. In other words, such systems are highly dependent and difficult to customize with other databases. Hence, it has become a crucial task to develop new methods to increase system robustness in understanding NL queries and their generalization.

Lately, researchers are attempting to use the advantages of semantics to improve NLIDB systems (Damljanovic, 2011; Khamis, 2010; Florencia-Juárez *et al.*, 2014). The increasing large usage of ontologies is motivated by the fact that they can rigorously and semantically represent a given domain. Therefore, works to integrate ontologies within databases have appeared (Pierra *et al.*, 2005; Broekstra *et al.*, 2002; Dehainsala *et al.*, 2007). They aim to facilitate the understanding of the stored data and increase interoperability in distributed and heterogeneous environments.

In this paper, we propose a novel method for processing NL queries by semantically enriching a relational database source. Our contribution is twofold; first we propose to enrich an existing relational database with semantics. The first step in enrichment extracts an ontology as triples (subject, predicate, object) using a reverse engineering process. In the second step, the extracted ontology is enriched with more semantics using a domain ontology and WordNet. The second enrichment step increases system accuracy in understanding NL queries. The last enrichment step stores all discovered semantics in the same relational database. The second contribution of our work consists in processing NL queries using the stored semantics.

The rest of the paper is organized as follows: Section 2 gives a background of our research context. Section 3 details our processes of enriching a relational database with semantics. Our NLIDB system is detailed in Section 4 followed by an evaluation in Section 5 and a conclusion.

2. Background

This section discusses the background of our research context including details of the relational database reverse engineering process and NLIDBs.

2.1 Relational database reverse engineering

The relational model has advantages as well as disadvantages, especially when it comes to represent the semantics of the stored data. A relational schema is the fruit of an entity-association model. This has the ability to represent some real-world semantics. However, these semantics become hidden once the entity-association model is translated to the relational model. In order to retrieve relational database hidden semantics a reverse engineering process is used.

A reverse engineering process is generally defined as the process of reconstruction or restructuration of a model into another model or a higher level model. For relational databases, the process of reverse engineering is being used to extract different models including ontologies. It is also mainly considered as the process of enriching a database source with semantics (Hainaut, 2002). Many models were defined as the source-to-target of this process, passing from the conceptual model (Johannesson, 1994) to the object (Ramanathan and Hodges, 1997), and then to the ontological model. Indeed, lately ontologies are being considered as the most used model to be the source-to-target of relational database reverse engineering. The related work can be classified as follows.

Approaches based on the analysis of relational schemes. Most works of the literature only exploit the relational schemes to extract an ontology. The proposed approach in Stojanovic *et al.* (2002) supposes that the ontology exists and applies rules to create mappings between the relational schemes and the ontology. This approach aims to help migrating static websites to the semantic web. The work in Astrova (2004) proposes an approach that analyses the SQL-DDL code of the relational database. Correlations between keys, attributes, and data are analysed to discover semantics. The process divides the relational schemes into base relations, composite relations, and dependent relations. An extension of this work has succeeded in Astrova and Kalja (2006) and represents the resulted ontology in OWL. Another recent work (Zdenka, 2010) focusses on converting relational database constructs to OWL constructs.

Approaches based on exploiting an external source. In this category of approaches an ontology is extracted from a relational database and is enriched by means of an external source. Some approaches are exploiting the semantics that could be found in related HTML pages which interact with the database (Astrova and Stantic, 2005), (Benslimane *et al.*, 2008). Such systems analyse HTML forms in order to discover additional semantics. The discovered new entities are organized in a way that permits discovering relevant constraints and dependencies between data. Another work (Kashyap, 1999) exploits user queries in order to add additional semantics. User queries can indeed show new additional semantics to be represented as new concepts or properties.

On the other hand, there exist approaches in the literature that use different techniques for dealing with data access in a relational database. More specifically, they simplify data access by providing users with an NLIDB. The concept of this is detailed in the following.

2.2 NLIDB

In the late 1960s and early 1970s, works on developing NLIDBs have appeared (Androutsopoulos *et al.*, 1995). An NLIDB system is a system that processes NL queries formulated by users. Once NL queries are received, they are processed and converted back into SQL queries in order to retrieve the desired data. A user can formulate NL queries in different ways to obtain any response, which can lead to ambiguities/misunderstandings. Hence, the main objective of NLIDB systems is to understand what is exactly being requested in the sentence and how. From this point, it appears that it is important to use database semantics which can reduce considerably such misunderstandings.

In the literature, three types of NLIDBs are distinguished, according to their type of architecture and the technique used.

Pattern matching systems. In the first category, the proposed systems deploy techniques based on the use of patterns. Patterns are seen as predefined rules such that when an NL query matches to a pattern, SQL queries are generated (Ahmad *et al.*, 2009; Androutsopoulos *et al.*, 1995). However, there must be a very large number of defined patterns in order to increase the chances to understand an NL query. Therefore, such systems are limited to the database to be queried. The other disadvantage is that such systems cannot understand sentences in which the pattern is not well defined, especially if the database has updates, changed names of attributes, or added others.

Tree-structure-based systems. In this category, the sentence is parsed and a tree is generated. Systems proposed in Hallett (2006) and Shah *et al.* (2013) are based on using a syntax parser. This is a programme that has as input an NL query and as output a syntactic representation which facilitates the mapping to a database query language (Li and Jagadish, 2014). Other works in this category are semantic grammar-based systems (Giordani and Moschitti, 2010; Gupta *et al.*, 2012). The result of a semantic grammar parser is similar to a syntax parser by giving a tree representation. However, it is more sophisticated and provides a better semantically enriched representation. Unfortunately, the main disadvantages of this category of systems are that parsers are highly dependent on the database domain. Also, they are designed for a specific NL. They can possibly generate a tree syntactic representation with global or local ambiguities. In other words, parsers work properly when NL queries are well formulated following grammatical rules.

Intermediate representation language systems. In this category of systems, NL queries are received and transformed to a logic representation before their translation to a database query language (Martin *et al.*, 1986; Popescu *et al.*, 2004; Minock, 2010). The advantage of these systems is the module of the NLIDB system that generates the intermediate representation. Indeed, this last is independent of the underlying DBMS. However, the disadvantage lies in the step of interpreting the NL query itself. Formulating the same NL query in different ways can lead to generation of different intermediate representations. This is why most of the works in this category have been proposed for deductive databases.

All these classifications are still not a standard option for interfacing databases due to many limitations, especially the following:

- lack in understanding user sentences;
- difficulty in mapping a sentence to a general database query language;
- limited linguistic coverage (supporting only a subset of a NL);
- not providing a solution to users to help them rephrase their wrong sentences (no explanations are returned to users in the case of a system failure); and
- complex configurations.

To overcome this problem we focus on semantically enriching a database source. This increases the levels of understanding user sentences. Also, enriched semantics provide promising solutions to the rest of the limitations. The processes we propose to semantically enrich a relational database are detailed next.

3. Enrichment of a relational database

Generally, the process of enrichment refers to an automatic or semi-automatic extension of a given representation or model. The purpose of the enrichment process is to provide more semantic richness and expressiveness. In the case of databases, the enrichment process gives a semantic support to the stored data. Usually, this is done by remodelling database schemas in a higher data model in order to explicitly express semantics that are implicit in the data. Several approaches in the literature deal with such schema conversions or reverse engineering, transforming relational schemas to entity-relationship schemas (Johannesson, 1994), object-oriented schemas (Ramanathan and Hodges, 1997), or ontologies (Benslimane *et al.*, 2008; Hamaz and Benchikha, 2013). Most approaches expect a relational schema and a list of integrity constraints, and then construct automatically a corresponding target representation. The typical forms of constraints that are taken into account are primary keys and candidate keys (Hamaz and Benchikha, 2012), functional dependencies, and inclusion dependencies.

In our work, we propose three phases for enriching a relational database with semantics (see Figure 2). In the first phase, a reverse engineering process with a set of rules is applied to extract an ontology. The second phase consists in enriching the extracted ontology with more concepts and synonyms. The last phase stores all discovered semantics locally in the same relational database.

Before detailing our enrichment processes, we first introduce the concepts of the relational model and ontologies.

3.1 *Basic relational concepts*

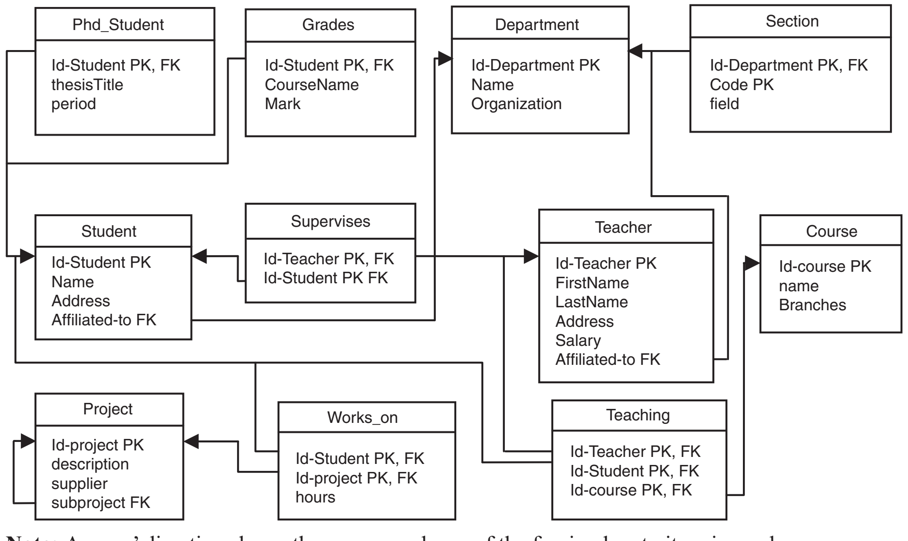
A relational schema R can be seen as a finite set of attributes that we denote as ∂. Attributes are denoted using the capital letters of the alphabet from the beginning A_i, B_i, C_i, … Sets of attributes are denoted by the last capital letters of the alphabet Y_i, X_i, Z_i. We use r to represent a relation schema (table), where r is a subset of ∂ and $(r_1 \cup r_2 \cup \cdots r_i) = \partial$. If we assume that *r(X)* represents a relation schema with n attributes, we write $X = A_1, A_2, \ldots, A_n$. Each attribute has a set of values also called domain. Each relation has a primary key that uniquely identifies each tuple and can also be used as a foreign key. Primary keys and foreign keys are defined as follows.

A primary key *PK* is chosen from among the candidate keys *CK* to identify entities of a relation r. We write *PK(r)* to denote a primary key of a relation, and *PK(X)* to denote the attributes used for a *PK*. A foreign key *FK* in a relation r_i is a *PK* in another relation r_j (sometimes in the same relation), where values of the *FK* are reference values of the *PK*. Such references are called inclusion dependencies denoted by *IncD*, and they exist between two relation schemas if the following form holds: $r_i(Y) \subseteq r_j\ (Z) | Y = Z$ (same sequence of attributes) where Z represents attribute(s) of the *PK* of r_j and Y represents attribute(s) of the *FK* of r_i. It also means that all values of attribute Y are included in the bag of values of attribute Z. *IncD* takes the following form: $r_i(Y) \rightarrow r_j(Z)$. We use *SetfKeys* to denote the set of all foreign keys in R. We denote by *Term(r), Term(a),* and *Term(fk)* the relations, attributes, and foreign keys, respectively.

An example of a relational schema is illustrated in Figure 1. We use it to illustrate the application of our different processes.

3.2 *Ontologies*

Lately, the use of ontologies grows considerably in the field of data modelling and processing. Ontologies are important because they provide continually new advantages. They give a rigorous and a formal manner to formulate conceptual schemes (Thomas, 1993) and they use online dictionaries and vocabularies as WordNet (Figure 2).



Note: Arrows' direction shows the correspondence of the foreign key to its primary key

Figure 1. An example of a relational database schema

3.3 Basic ontology concepts

An ontology can be seen as a four tuple $O = (C, A, R, H)$, where:

(1) C is a finite set of classes/concepts (C_i, C_j, C_k, …). We use the term "concept" in the rest of the paper.

(2) A is a collection that represents the set of attributes belonging to the concepts.

(3) R is a finite set of relationships/roles that exist between concepts. $< C_i\ R\ C_j >$ denotes a relationships/roles between two concepts | C_i (Domain), C_j (Range).

(4) H represents the hierarchy or taxonomy between concepts. We write $< C_i\ \textit{is-a}\ C_j >$ to denote that C_i is the subconcept of C_j.

3.4 Reverse engineering rules

Our reverse engineering process (presented in Table I) analyses the relations to discover what semantics to create. We assume that the database is in the third normal form. First, it creates concepts for relations in which the primary key is not composed exclusively of foreign keys (rule 1). Any relation that satisfies rule1 is called an atomic relation. The rest of the relations are called non-atomic relations. The generalization relationship between two concepts is created when the values of one of the primary keys are a subset of the values in another relation's primary key/candidate key (rule 2). Additionally, one key should be referencing the other one. Rule 3 discovers reflexive roles while rule 4 discovers one-to-many roles between two concepts. Rule 5 treats the case of weak entities which also have a semantic to be considered. The last case of role in rule 6 deals with many-to-many roles and we use variables i, j to treat all possible cases ($n \geqslant 2$). Finally in rule 7 attributes of the concepts are created.

After the ontology is extracted as triples, it will be enriched with more semantics using a domain ontology.

3.5 Enrichment of the extracted ontology

This step of enrichment using a domain ontology is not mandatory, but it enables better processing for user sentences. If no domain ontology is found, the system will process user sentences using the extracted ontology of the reverse engineering process. Our choice for enrichment using a domain ontology is motivated by the fact that every relational database is created to satisfy a specific purpose. Therefore, most designed databases lack some information/semantic about their domain (Hainaut, 2002). When an ontology is extracted from a relational database, it does not cover very well the semantics of its application domain. This is seen as a poor modelling. An enriched ontology offers more possibilities to understand and answer user sentences. Similarly, user sentences can contain abstracted, contextual, or synonymous concepts of those stored in the database. For this reason, our second enrichment process (see Algorithm 1)

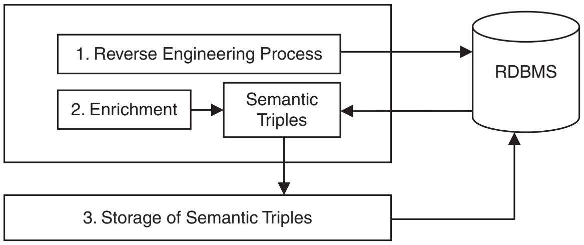

1. Reverse Engineering Process
2. Enrichment
Semantic Triples
RDBMS
3. Storage of Semantic Triples

Figure 2.
Semantic enrichment of a relational database

Rule	Preconditions	Equivalent semantics
R_1	Relation r where PK$(r) \bigcap$ Setfkeys $= \varnothing$	Concept C_r
R_2	r_i, r_j (atomic relations) with $r_i\,(Y) \rightarrow r_j(Z)$ Such that: PK$(Y) \subseteq$ PK(Z) or CK$(Y) \subseteq$ PK(Z)	Triple: $< C_{ri},$ is-a, $C_{rj} >$
R_3	r_i (atomic relation) with $r_i\,(Y) \rightarrow r_i(Z)$ Such that: FK$(Y) \subseteq$ PK(Z)	Triple: $< C_{ri}$, Term(fk), $C_{ri} >$ Domain (Role) C_{ri} Range (Role) C_{ri}
R_4	r_i, r_j (atomic relations)$\vert$*IncD*: $r_i\,(Y) \rightarrow r_j\,(Z)$ Such that: FK$(Y) \subseteq$ PK(Z)	Triple: $< C_{ri}$, Term(fk), $C_{rj} >$ Domain (Role) C_{ri} Range (Role) C_{rj}
R_5	r_i, r_j (non atomic)$\vert$PK(r_i) has PK(X, Y) & PK(r_j) has PK(Y) *IncD*: $r_i(Y) \rightarrow r_j(Y)$ Such that : PK$(r_i) \bigcap$ PK$(r_j) = Y$	Triple: $< C_{rj}$, has, $C_{ri} >$ Domain (Role) C_{rj} Range (Role) C_{ri}
R_6	r_i (non atomic)$\vert r_i(Z)$ has: $Z \in$ Setfkeys and $Z = n$ Such that: $n \geqslant 2$	$\forall i, j \leqslant n \wedge i \neq j \wedge i < j\ Do$: $\begin{cases} \text{Triple (role) : } < C_{ri}\text{ , Term}(r),\ C_{rj} > \\ \text{Triple (InvRole) : } < C_{rj}\text{ Term}(r)\ \ C_{ri} > \\ \text{Domain (Role) } C_{ri} \\ \text{Range (Role) } C_{rj} \end{cases}$
R_7	$\forall\ r$, $r(X)$ with $X = (A1, A2, \ldots, An\,)$ and $X \neq$ Setfkeys	Triples: $<$ Term(r), has, Term$(a_1) >$ $<$ Term(r), has, Term$(a_2) >$ $\ldots$ $<$ Term (r), has, Term$(a_n) >$

Table I.
Reverse engineering rules for discovering database hidden semantics

attempts to add super/subconcepts using a domain ontology, and synonymous concepts using WordNet. Some points of this enrichment are inspired from the work of Robin and Uma (2010).

Algorithm 1. Algorithm of enrichment

```
Input: Resulted Ontology (RO), Domain Ontology (DO);
Variables: List r1, r2, r3, co1, co2, Enrich;
String k, q, s, f, supc1, supc2; Boolean b;
Ensure: Concepts of RO are organized from top to down in r1;
Concepts of DO are organized from bottom to top in r2;
Output: A List called Enrich with new triples

Algorithm:
// c1, c2 represent respectively each concept from r1 and r2
for each c1 in r1
  k= Stemming (c1);    // Getting the root word in variable k
  r3= (Wordnet(k));       // Receiving k with its synonyms in r3
  Synonyms (c1, r3);    // Function to get synonyms from WordNet
    for each c2 in r2
      for each q in r3
        b=StringCompare(q,c2);   // b is set false by default
        if (b);     // if b=true (similarities are detected)
          Similitudes(c1, c2); //  Function to calculate the degree of similarities
          co1= SubConcepts of c1; co2= SubConcepts of c2;
        end if

  if (c1 or c2 is empty)
    Break;
// Attempting to add sub-concepts
    For each  s in co1
      For each f in co2
        if ((s && WordNet(s)) are all different of f)
          Enrich = (<f, is-a, s>);
          supc1= Super-concept of c1; supc2= Super-concept of c2;
        end if
        Break;
// Attempting to add super-concepts
if  (supc1 is empty && supc2 not empty)
        Enrich = (<c1 is-a supc2> );
// Attempting to reorganize the ontology
if (f.sub-concept == (any other rest of s in co1 and their synonyms))
      // we call d any concept s that satisfies this condition
Once a concept is added to c1 (Create Triple <d, is-a, f> and delete hierarchy between c1 and d);
return Enrich;
```

The enrichment process includes three main cases as illustrated in Figure 3.

RO represents the resulted ontology of the reverse engineering process. DO represents the domain ontology. In the first case, when two concepts C1, C2, respectively, from RO and DO show similarities, subconcept(s) of C2 are added to C1 (e.g. Cn added to C1). The second case adds a superconcept to a concept (e.g. Cs of C2 is added to C1). The last case makes sure to keep RO as coherent as DO while adding subconcepts. For example, Cn is added to C1;

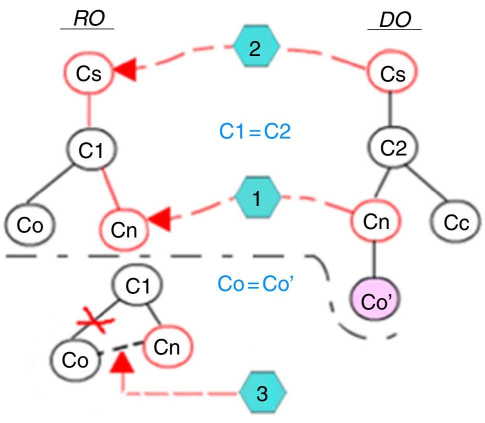


Figure 3. The three cases of the enrichment process

however, in DO, Cn has a subconcept Co' which is the same as Co from C1. Therefore a new subsumption is created for C1 and Cn in RO.

On the other hand, Figure 4 illustrates part of the results when applying our reverse engineering process, and enrichment using a domain ontology. For our example we use the domain ontology found in (D-O).

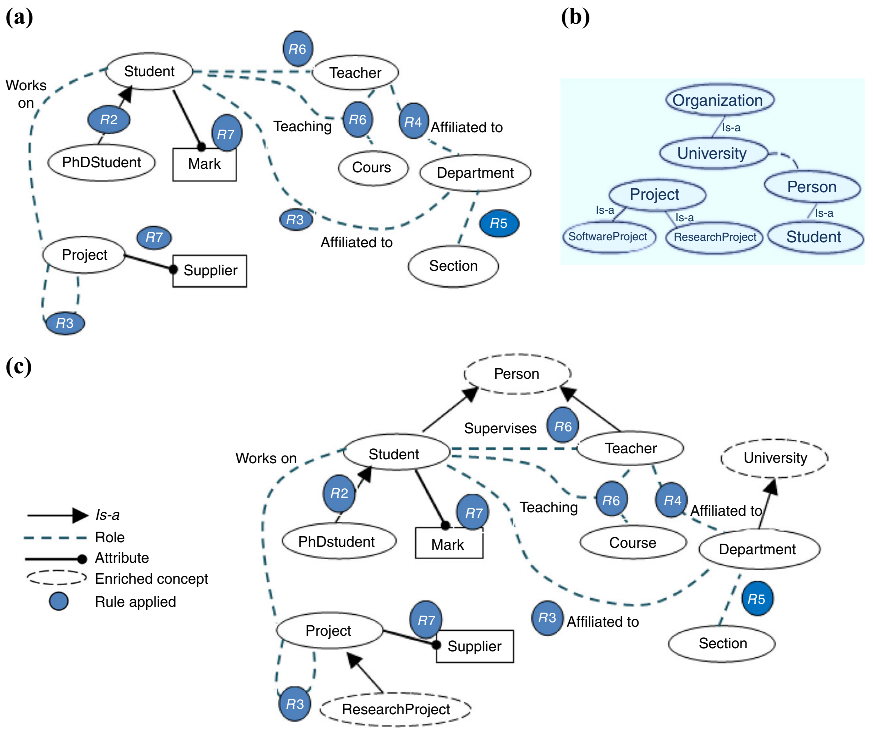


Figure 4. Generated and enriched ontology

Notes: (a) Part of the results when applying our reverse engineering process; (b) part of the domain ontology; (c) part of the results after the second enrichment phase

3.6 Storage of semantics

The storage of the discovered semantics is an important step in our approach. It enables semantics of data to be computed by the NLIDB system. It is preferred to store semantics in the same database rather than storing them in a file or another database to prevent system complexity. In the literature there exist two main specific approaches for storing data and semantics in a single database.

(1) Vertical representation approach: in this approach, the storage is simple where ontological concepts and instances of concepts (data) are stored as triples: "Subject, Predicate, Object". Jena (Wilkinson *et al.*, 2003) is an example for this kind of approaches.

(2) Specific representation approach: in this approach, the storage is different from an implementation to another. However, the most general strategy stores separately the ontological concepts and data, where each data is linked to its ontological concepts. IBM SOR (Jing *et al.*, 2007) follows this approach for storing ontology and data.

In our work we use the specific representation approach. We store the ontological concepts as triples in a new table called "Semantic-Table", and we leave the already existing data without any changes. To effectively use the stored semantics, our system makes a link between every data and its corresponding ontological concepts. By doing this, we avoid using a big amount of storage space. Consequently, even if we deal with large-sized databases the size of a semantic database will always need a fewer additional space for storing triples. The Semantic-Table has four columns (see Table II):

- IdTriple: triple are stored with identifiers to facilitate their exploitation by the system.
- Subject: stores concepts used as the subject of a given triple.
- Predicate: stores the relationship/roles used between the subject and an object.
- Object: stores the object used in a given triple (it can be a concept, an attribute, or an enriched concept).
- Triple_type: this column stores the type of each created triple. The triple_type helps the system generate correct SQL queries when answering user sentences. In addition, it helps understanding synonymous concepts and new enriched concepts which have not been covered by the database.

We give a brief explanation of each "Triple_type" using examples from Figure 1 as follows:

- Rule1: are triples created in Rule1 of our reverse engineering process and define a created concept (e.g. Student, defined-as, concept).
- Rule2: are triples created in Rule2 and are defining a hierarchy "is-a" between two created concepts. They help extracting data from the superconcep. (e.g. what is the

Table II.
Storage of semantic triples in the semantic table

IdTriple	Subject	Predicate	Object	Triple_type
Id-triple	Student	Defined-as	Concept	Rule1
Id-triple	PhD_Student	Is-a	Student	Rule2
Id-triple	Project	Has_SubProject	Project	Rule3
Id-triple	Teacher	Affiliated-to	Department	Rule4
Id-triple	Department	Has	Section	Rule5
Id-triple	Student	Supervised by	Teacher	Rule6
Id-triple	Teacher	Has	Name	Rule7
Id-triple	Student	Same-as	Scholar	Synonym
Id-triple	Student	Is-a	Person	Enriched

name of the PhD student working on a semantic project? Here, "PhD student" does not have a column "name", thus, the table superconcept "student" is used to extract the name).

- Rule3: are triples created in Rule3 and represent recursive cases occurring in one table (e.g. The attribute "SubProject" referring to the same table "Project").
- Rule4: are triples created in Rule4 and represent a One-to-Many relationship between two tables (e.g. Teacher, affiliated-to, department).
- Rule5: are triples created in Rule5 and represent relationship between a table and a weak entity table (e.g. The table "section" uses the *PK* of department as a part of its *PK*).
- Rule6: are triples created in Rule6 and represent (many-to-many) relationships between tables (e.g. The "Supervises" table shows that every teacher can supervise more than one student, and that a student can be supervised by more than one teacher).
- Rule7: are triples created in Rule7 and they represent attributes of concepts (e.g. Teacher, has, name).
- Synonym: are triples for synonyms database covered concepts (e.g. Student, same-as, scholar).
- Enriched: represent triples created after the second enrichment process using a domain ontology (e.g. Which person works on a project entitled reverse engineering of relational databases? Here, the word person is understood to be a student using the triple < student is-a person > . The other triple < teacher is-a person > is not used because it is out of the sentence's scope. Such triple filtering and sentence processing are detailed in the next section).

4. Intelligent understanding of NL queries

Once semantics of the database are discovered, enriched and stored, our NLIDB system can process NL queries. There are two phases for processing a NL query (see Figure 5):

1. Phase 1: identifying the semantic nature of NL query elements; and
2. Phase 2: understanding the NL query.

4.1 Identifying the semantic nature of NL query elements

Once an NL query is received the first phase consists in identifying the semantics of each of its elements. Users have a tendency to use different combinations of words (nouns, verbs, adjectives, etc.) when asking for something. If an NLIDB system misunderstands those words, the generated results are more probably to be wrong or incomplete. To overcome this problem we use enriched semantics to understand the semantic of each NL query element. The phase of identifying the semantic nature of NL query elements has four steps as follows.

4.1.1 Tokenization. The first step towards processing a received NL query is to divide it into atomic elements. For this, we use a process of tokenization which transforms a stream of text into tokens. For instance, consider the following example:

In which department scholar Adam Green is affiliated?

Tokens: [in] [which] [department] [scholar] [Adam] [Green] [is] [affiliated].

4.1.2 Identifying concepts, predicates, objects, and attributes. The second step matches NL query elements with the stored triples in order to identify their semantic nature (concepts, predicates, objects, or attributes). While matching, the system temporarily gets the elements being matched, stemmed. This avoids the problem of not finding a match because an

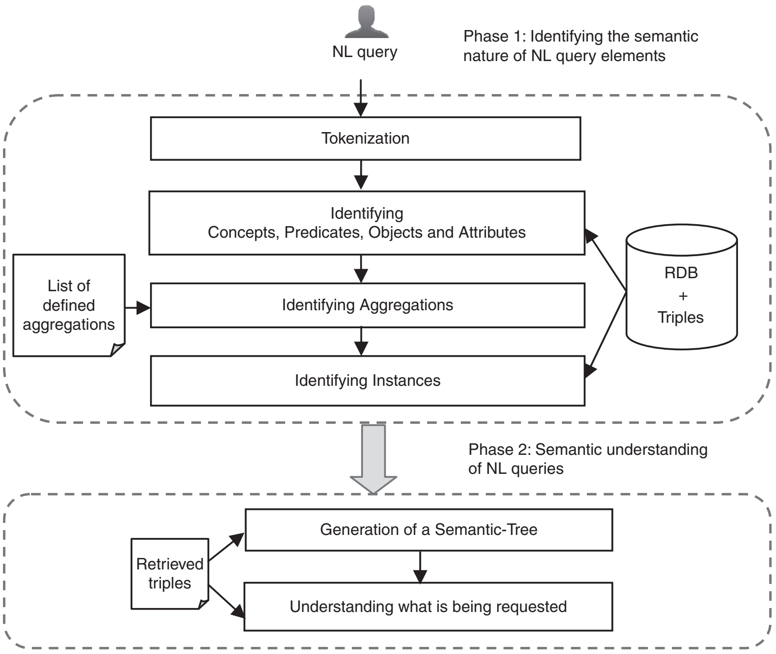

NL query
Phase 1: Identifying the semantic nature of NL query elements
Tokenization
Identifying Concepts, Predicates, Objects and Attributes
List of defined aggregations
Identifying Aggregations
Identifying Instances
RDB + Triples
Phase 2: Semantic understanding of NL queries
Retrieved triples
Generation of a Semantic-Tree
Understanding what is being requested

Figure 5.
System for intelligent understanding of NL queries

element is not a root word and is using suffixes, prefixes, etc. Once done, the system retrieves triples in which a match is found, but only those that are relevant to the NL query (see Figure 6). A set of filtering conditions is used to eliminate non-relevant triples as follows:

- if an NL query element matches on a triple of "Triple_type" synonym or enriched (see Table II): do not retrieve it;
- when two elements match on the same triple: retrieve this tuple only once; and
- if among the retrieved triples there exist a triple(s) in which the subject and object are matched with NL query elements: do not retrieve the other triples where the subject or object is not matched.

In Figure 6, the element scholar is identified as a student using the synonymous enriched triple. According to the filtering conditions, the triples $<$ student same as scholar $>$ and

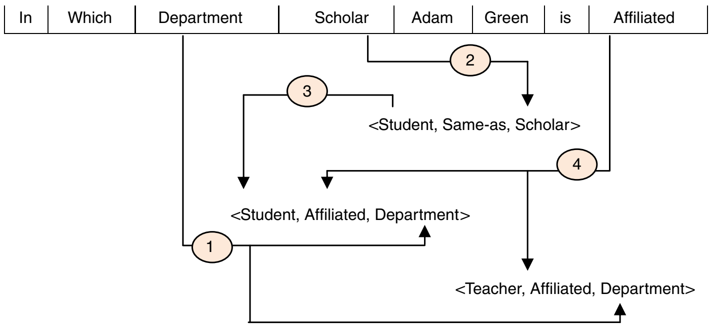

In
Which
Department
Scholar
Adam
Green
is
Affiliated
2
3
<Student, Same-as, Scholar>
4
<Student, Affiliated, Department>
1
<Teacher, Affiliated, Department>

Figure 6.
The process of filtering triples in order to keep only relevant triples for the NL query

< Teacher, affiliated, department > will not be retrieved. The only retrieved triple is < Student, Affiliated, Department > where NL query elements have matched on the subject, predicate, and object.

In order to facilitate the process of matching and reduce its execution time, the system ignores stopwords (e.g. and, is, in, of, etc.) and questioning words "Q-W" (what, which, who, etc.).

4.1.3 Identifying aggregations. After the identification of concepts, predicates, and objects, this step identifies if aggregations are used in the NL query. Users are more likely to use aggregations in their NL queries and a successful NLIDB system should be adapted to handle them. In our work we provide a solution for processing aggregations by dividing them into categories (see Table III).

We define an aggregation as four-tuples (C_{agg}, $Term_{agg}$, Fun_{agg}, $Attr_{agg}$) where:

C_{agg}: defines the category of the aggregation.

$Term_{agg}$: defines the term used for an aggregation.

Fun_{agg}: defines the function to be applied for the aggregation.

$Attr_{agg}$: defines the attribute in which the function will be applied.

Category 1: are aggregations that should specify at least one value.

- Example: who are the students having a mark more than 15 in math course?

Category 2: are aggregations that do not specify values.

- Example: what is the average mark of student Adam in all courses?

Category 3: are combined aggregations used for the same concept/attribute.

- Example: which students have a mark above average in philosophy course?

Aggregations should specify an attribute ($Attr_{agg}$). Regarding the first three categories the attribute is either specified in the NL query or is defined by an expert using this form:

$$if\ (concept) \wedge Term_{agg} \rightarrow Attr_{agg} \tag{1}$$

For instance, let us consider the following example: Who are the best students? Here, it is unclear for the system on which attribute the aggregation "best" should be applied. Therefore, an expert should define earlier attributes for such aggregations:

if (Student) ∧ Best →Mark).

Category 4: are aggregation that are supposed to return the number of a count:

- Example: how many students are working on a project?
- The $Attr_{agg}$ of this category are primary keys.

C_{agg}	$Term_{agg}$	Fun_{agg}
Category 1	Superior, above, Upper than […]	>
	Inferior, under […]	<
Category 2	Average, Medium […]	AVG
	Best, Highest […]	MAX
	Worse, lower […]	MIN
Category 3	Above average […]	(>) AVG
	Below average […]	(<) AVG
Category 4	How many, More than […]	COUNT
Category 5	$Term_{agg}$	Filter (Value)

Table III. Categorization of aggregations

Category 5: are aggregations that are dedicated to a specific domain, and are always defined by an expert using this form:

$$if\ \left(Term_{agg}\right) \rightarrow \left[Concept, Attr_{agg}\right]\ (Values) \tag{2}$$

Let us consider the following two examples to explain the usage of such aggregations:

- Example 1: which are the valuable projects?

  Predefined second form: *if (Valuable) → Supplier (IBM)*

  Using this filled form the system understands that the aggregation "valuable" is used for the attribute "supplier" of the concept "project". Additionally, the instance "IBM" is provided to be used for filtering assuming that valuable projects are supplied by "IBM".

- Example 2: what are the technical branches?

  Predefined second form: if (Technical) → (Course, Branches) (Math, Physics, Electronics)

  In this example, the system understands that the aggregation "technical" is to be used on the column "branches" (see Figure 1). Moreover, the filtering here is performed using three instances (Math, Physics, and Electronics).

After the identification of any possible aggregations, the remaining non-identified elements in the NL query can represent instances and are identified in the next step.

4.1.4 Identifying instances. Any NLIDB should accurately identify instances used in an NL query. Otherwise, the system will generate SQL queries without proper WHERE clauses. Identifying instances can be ambiguous and problems arise (Pazos *et al.*, 2013). There are two main problems that should be handled:

- Problem 1 (When the value involves two columns):

  Example: which student Josef Silver is supervising?

  The value "Josef Silver" involves two columns "FirstName" and "LastName" so if the system searches for this value in only one column it will not find a match or will return wrong results.

- Problem 2 (When the value is constituted by two or more words):

  Example: which student is working on NL processing using enriched semantics?

  The value "natural language processing using enriched semantics" includes many words and can eventually confuse the system while trying to find a match.

In our work we use the following solution:

(1) If non-identified elements exist before or after a concept/attribute/predicate we combine them.

  Example: which student is working on NL processing using enriched semantics?

  Combined: (on NL processing using enriched semantics)

(2) Track triples of "Triple_type.Rule7" of the concepts in the retrieved triples.

  Example: < Teacher has FirstName >, < Teacher has LastName > < Department has name >

(3) Generate triple-based instance < attribute, value, instance >:

 e.g. < FirstName, value, Albert > , < FirstName, value, Josef > […]

 e.g. < LastName, value, Silver > , < LastName, value, Adam > […]

 e.g. < name, value, department of physics > , < name, value, department of computer science > […]

(4) Combine attributes that may define a concept:

 e.g. < FirstName+LastName, value, Albert Adam > , < FirstName+LastName, value, Josef Silver > […]

(5) Compare the temporal instance based triples with the value of the NL query.

The solution of the first problem is handled by combining attributes that may define a concept (e.g. < Teacher has FirstName+LastName >). The system returns only those with a perfect match. In the second problem the system does not look for a perfect match but accepts matches with high similarity.

4.2 Semantic understanding of NL queries

The second phase generates a semantic tree using the identified elements of the NL query. The semantic tree serves as a support to understand what is being requested and how. By doing this, NL queries are less likely to be misunderstood.

4.2.1 Creation of a semantic tree. A semantic tree organizes the identified elements by their semantic nature in a tree with three levels: concept levels, attribute levels, and instance levels (see Figure 7). Creating a semantic tree promotes understanding NL queries and helps generating correct SQL queries.

While generating a semantic tree it occurs to have some missing elements. These elements are seen as gaps (Carbonell and Hayes, 1983). In our work, we fill the gaps using the retrieved triples (see Figure 8). For instance, we use these three examples followed by their generated semantic trees:

Example 1: in which department scholar Adam Green is affiliated?
Example 2: which student Josef Silver is supervising?
Example 2: what is the name of teachers living in address Street 17?
The system has two ways to fill gaps:

(1) If gaps are by the side where an instance is provided we use the retrieved triples to fill gaps (e.g. in Figure 8(b) Level 3 "Josef Silver" has Level 1 and 2 filled).

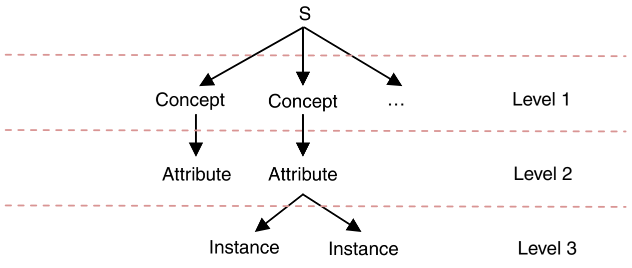


Figure 7.
Representation of NL query elements in a semantic tree

(2) If a concept is requested without specifying an attribute we use an attribute that can define the concept such as identifiers, names, designation, etc. (e.g. in Figure 8(a) department is filled in Level 2 with attribute "name").

4.2.2 Understanding what is being requested. Once the semantic tree is generated the system uses it to understand what exactly is being requested in the NL query and how. There are three main cases for understanding what elements are requested.

- Case 1 (A semantic tree with one concept): if a semantic tree reaches Level 3 this means that there is one attribute or more with instances (e.g. "Street 17" of Figure 8(c)). In this case the requested element is the attribute without instances. The generated SQL would have the following form. (Note that if the semantic tree does not reach Level 3 the SQL query will not use a WHERE clause.)

  Generated SQL:
  SELECT Level 2 (attribute without instance)
  FROM Level 1
  Where Level 2 (attribute with instance) = "Level 3"

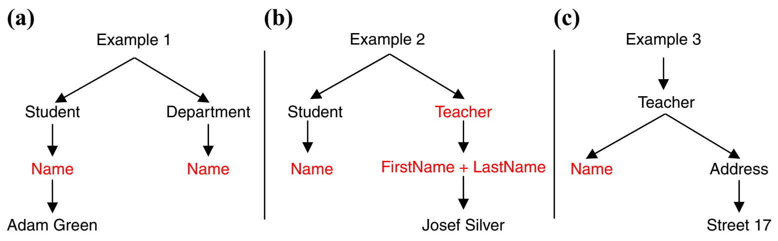
- Case 2 (A semantic tree with two concepts): if a semantic tree has two concepts (see examples 1 and 2 of Figure 8) and reaches Level 3 the requested element is the attribute without instances. In the other case, if the semantic tree does not reach Level 3, the SQL query will not use AND clause.

  Generated SQL:
  SELECT Level 2 (of the concept without Level 3)
  FROM Level 1 (concept without Level 3)
  WHERE Id. Level 1(concept with Level 3) = Id. Level 1(of the concept without Level 3)
  AND Level 2(concept with Level 3) = "Level 3"

- Case 3 (A semantic tree with three concepts or more): if a semantic tree has three concepts and reaches Level 3 the requested element is the concept after a Q-W and without instances (see Figure 9).

For instance consider this example: retrieve the average scholars who study at least one course and are affiliated to department of physics?

The attribute "name" of the concept "course" does not have an instance and is not requested. Instead, it is used as a condition for retrieving the desired scholars. However, our system returns the name of course in which students fulfil the condition



Note: Filled gaps are underlined and represented in a different colour

Figure 8.
Examples of representing different NL queries in a semantic tree

to provide a rich answer. The generated semantic tree for this example is represented in Figure 9.

```
Generated SQL:
SELECT student. name, course.name, AVG(Mark) AS Average FROM Grade
WHERE id-student IN
(
select student.id-student from course
join teaching on course.id-course = study.id-course
join student on teaching.id_student = student.id-student
where student.affiliated-to = "physics"
)
GROUP BY student. name, course.name
```

In this SQL query we use the names of the tables instead of levels to make it understandable. This SQL example also shows the usage of aggregations and this makes the query uses a GROUP BY clause as well. The SELECT clause uses the attributes without Level 3 (see Figure 9). The IN clause helps when manipulating data from three or more tables, and the WHERE clause is used when there exists a Level 3 in the semantic tree.

In case the three cases do not apply, the system returns triples of any identified element, especially when an NL query is not well formulated. The users can then understand database semantics and reformulate queries beyond the scope of the database. The next section presents an evaluation to our work.

5. Evaluation

In the section, we evaluate the robustness and effectiveness of our work by evaluating the capacity of our system to understand NL queries and also the capacity to deal with complexities and how semantics play an important role for system accuracy. Finally we discuss the reusability of our proposed method, followed by a comparison evaluation between related work and ours.

5.1 Capacity to understand NL queries

Our system is based on the usage of semantic triples to understand user sentences. These are a key solution towards a flexible and intelligent system. The reason is that the human language is full of irregularities. Thus, it is important to understand semantically each NL user element using enriched semantics.

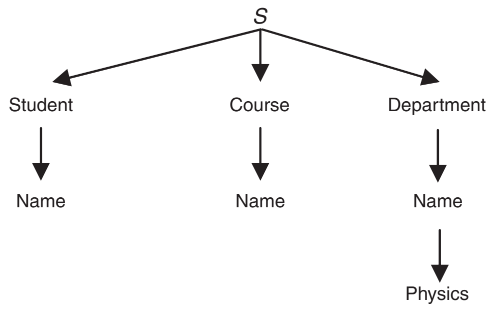


Figure 9.
A semantic tree with three concepts

In the following, we demonstrate the capacity of our system to understand NL queries with ambiguous content using semantic triples. Let us consider the following two NL sentences and suppose that "employee" is a database table with no links with tables "student" and "project":

- NL query 1: who are the employees and students that are working on project NL Processing?
- NL query 2: who are the teachers and employees supervising student Albert Adam and work in the Department of Physics?

First of all, the system retrieves triples to check the semantics between the concepts detected in the NL query.

Retrieved triples of the first NL query 1:	Retrieved triples of the second NL query 2:
<Student, works_on, project>	<Teacher, supervises, student>
<Employee, affiliated_to, department>	<Teacher, affiliated-to, department>
	<Employee, affiliated_to, department>

In the first sentence, the system deduces that employees do not work on projects. The system generated the semantic tree (see Figure 10(A)), and returns the triple that regards employees to the user $<$ employee affiliated_to deparment $>$.

In the second NL query sentence, the system understands that the employee does not supervise any student. However, the system understands also that an employee is affiliated to a department. Hence, the system generates two semantic trees (see Figure 10(B1), (B2)) and uses the instance of department "physics" as a condition to filter in both semantic trees.

In addition to having the ability to understand user sentences using semantic triples, our method benefits from using synonyms, and additional enriched concepts. This increases the capacity of the system to understand user sentences. If desired, synonymous concept from other languages can also be added to enable multilingual NL processing.

5.2 Capacity to deal with complexities

Here we evaluate how our system deals with the complexities, especially when generating a semantic tree. First is it important to mention that in our work we assume that the relational database should be in the 3NF. Otherwise, the generated semantic tree may be very complex and ambiguous. In case of dealing with complex databases, our system will still detect every NL query element by its semantic nature before using it in a semantic tree. Preventing thus,

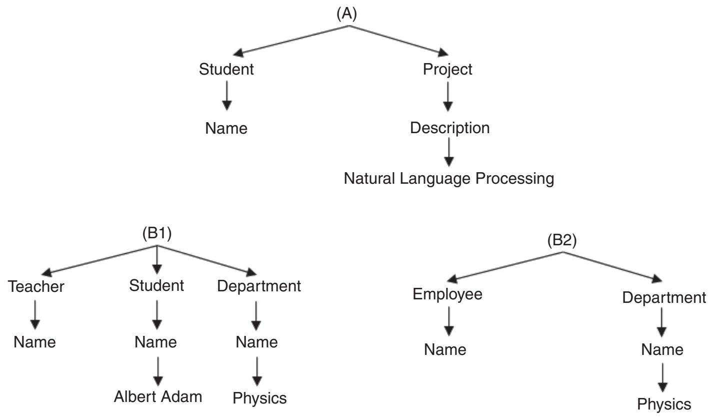


Figure 10.
Special cases of generating semantic trees

any ambiguous representations. The only case of a complex semantic tree occurs when this last has many branches (many detected concepts). Such semantic-tree complexities are handled using semantic triples as illustrated in Figure 10. However, another different case may occur as follows. Let us consider the following two NL queries:

- NL query 1: which student is taught by at least one teacher and is affiliated to the Department of Computer Science?
- NL query 2: who are the students and teachers affiliated to the Department of Computer Science?

These two NL queries are requesting different things, but have generated the same semantic tree (see Figure 11).

Retrieved triples for NL query 1 :	Retrieved triples for NL query 2:
<Student affiliated_to department>	<Student affiliated_to department>
<Student taught_by teacher>	<Teacher affiliated_to department>
<Teacher affiliated_to department>	

In the first NL query sentence, there exist a triple between student and teacher < student, taught_by, teacher > which is retrieved using the predicate "taught_by". Hence, it is used as a condition for student (the triple < Teacher, affiliated_to, department > is deleted). The reason is that the condition of department cannot be applied to it. Additionally, a condition of Level 2 (as the branch of teacher) can be applied only to one concept. But a condition of Level 3 (as the branch of department) can be applied to more than one concept.

For the case of the second NL query sentence, there exist two triples and teacher is not used as a condition for student. Hence, the system applies the condition of department for both student and teacher. This shows the capacity of the systems to understand the meaning of NL queries with the same generated semantic tree. As a consequence the accuracy of the system increases considerably.

5.3 *Accuracy*

The accuracy of an NLIDB system is its ability to answer correctly NL queries. From this point, we calculate the accuracy in NL query using two metrics:

(1) Precision: the number of NL queries correctly translated to SQL, divided by the number of NL queries that are correctly and incorrectly translated to SQL queries.

(2) Recall: the number of NL queries correctly translated to SQL, divided by the total number of (correctly translated + incorrectly translated + not translated) NL queries.

The first measure of precision is for showing the accuracy of the system when the relevant results are returned compared to the irrelevant ones.

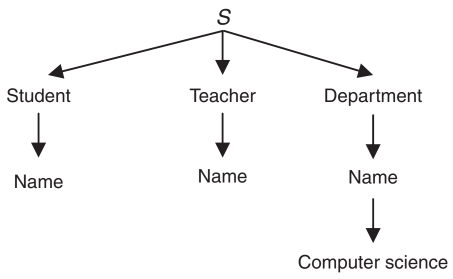


Figure 11. Same generated semantic tree for two different NL queries

Recall shows how many relevant results are returned. Table IV illustrates the obtained rates of precision and recall when the second enrichment step is considered and when it is not.

The reusability of our approach is discussed in the next.

5.4 Reusability

Our approach is based on a modular system using different modules that can be customized separately (reverse engineering, enrichment, storage, and NL processing). In other words, if a system needs to be modified or improved, it will require fewer efforts. Therefore, our system is easier for customization from this point of view. The generalization of our proposed method is discussed in the following points:

- Reverse engineering process: this process is independent of any NL and can be applied to any relational database in the 3NF. This makes the first module easy to generalize).
- Enrichment using a domain ontology: this process which is regarded as the second enrichment step is dependent on the dictionary used for synonyms (i.e. wordnet) and the language of the domain ontology. That is why this step is not mandatory in our system and can be avoided. Although, it is preferable to keep it in order to increase system accuracy in understanding user sentences.
- NL processing part: the techniques of our NLIDB system are based on using a semantically enriched database. If a database is already enriched our NLIDB attempts to generate a semantic tree using its semantics. This makes the processing techniques highly domain independent and easy to generalize, customize with other database domains. In Table V we demonstrate a comparison study between the approaches of the literature and our work.

Table IV.
Precision and recall rates when answering NL queries

Number of NL queries	Correctly translated queries	Incorrectly translated queries	Queries not understood (with elements out of scope)	Precision %	Recall %
Results when the second enrichment step is not considered					
100	71	18	11	79,77	71
Results when the second enrichment step is considered					
100	89	6	5	93,68	89

Table V.
Comparison study

	Characteristics						
	Sentence analysis						
Approach	Morphological	Syntactical	Semantic	Using external sources	Multilingual and Synonyms	Semantic retrieval	Generalization
Pattern matching systems	✓	✓					Very hard
Syntax-based systems	✓	✓					Hard
Semantic grammar systems	✓	✓	✓				Hard
Intermediate representation language-based systems	✓	✓					Easy
Our system	✓	✓	✓	Domain ontology	✓	✓	Easy

This comparison is based on different criterion to show whether an approach assumes using a morphological, syntactical, or semantic processing of an NL query. It shows also the external sources used for the approach; the support of synonymous concepts and multilingual queries; and finally it shows the capacity to be generalized or used to other databases.

6. Conclusions

The main objective of this research is to address the challenges of providing NLIDB systems with new processing techniques to overcome the limits of the existing systems. The idea of our approach consists in extracting database semantics. A second enrichment step is applied to add additional concepts and synonyms. We first applied a reverse engineering process to extract the relevant concepts and roles. The result of the reverse engineering process is an ontology represented as triples. However, the extracted ontology is poor semantically and needs to be enriched further. This motivates our choice to enrich the ontology and make it richer with more concepts and synonyms. All the obtained semantics are then stored locally on the same relational database. Our NLIDB system uses the stored semantics in order to have a better and intelligent understanding of NL queries. We introduce the principles of using a semantic tree which organizes the semantically identified NL query elements, which facilitates the generation of SQL queries. As a consequence, our system is more flexible and robust and easier to port and to use with other databases. On the other hand, semantics are provided for user's ambiguous NL queries. Therefore, the users can understand the coverage of the database and reformulate more precise NL queries. Our future work consists in improvin our system capabilities, especially in the case of understanding sentences with negation. In addition, an interactive system is preferable for the users.

References

Astrova, I. (2004), "Reverse engineering of relational databases to ontologies", *Proceedings of the 1st European Semantic Web Symposium (ESWS), Springer, Berlin Heidelberg, 10-12 May*, pp. 327-341.

Astrova, I. and Kalja, A. (2006), "Mapping of SQL relational schemata to OWL ontologies", *Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications*, *Elounda, 18-20 August*.

Astrova, I. and Stantic, B. (2005), "An HTML forms driven approach to reverse engineering of relational databases to ontologies", *Databases and Applications*, *Innsbruck*, *14-16 February*, pp. 246-251.

Ahmad, R., Abid, M. and Ali, R. (2009), "Efficient transformation of natural language query to SQL", *Proceedings of 2nd Conference on Language and Technology*, pp. 53-60.

Androutsopoulos, I., Ritchie, G.D. and Thanisch, P. (1995), "Natural language interfaces to databases-an introduction", *Natural Language Engineering*, Vol. 1 No. 1, pp. 29-81.

Benslimane, S., Malki, M., Rahmouni, M. and Rahmoun, A. (2008), "Towards ontology extraction from data-intensive web sites: an HTML forms-based reverse engineering approach", *The International Arab Journal of Information Technology*, Vol. 5 No. 1, pp. 34-44.

Broekstra, J., Kampman, A. and Van Harmelen, F. (2002), "Sesame: a generic architecture for storing and querying RDF and RDF schema", *Proceedings of the First International Semantic Web Conference*, *Number 2342 in Lecture Notes in Computer Science*.

Carbonell, J.G. and Hayes, P.J. (1983), "Recovery strategies for parsing extragrammatical language", *Computational Linguistics*, Vol. 9 Nos 3-4, pp. 123-146.

Damljanovic, D. (2011), "Natural language interfaces to conceptual models", PhD thesis, Language Resources and Evaluation, The University of Sheffield, Sheffield.

Dehainsala, H., Pierra, G. and Bellatreche, L. (2007), "Ontodb: an ontology-based database for data intensive applications", DASFAA. D-O, available at: http://ontoware.org/swrc/swrc_v0.3.owl

Florencia-Juárez, R., Rangel, R.A.P. and Morales-Rodríguez, M.L. (2014), "Using semantic representations to facilitate the domain-knowledge portability of a natural language interface to databases", in Castillo, O., Melin, P., Pedrycz, W. and Kacprzyk, J. (Eds), *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, Springer International Publishing, pp. 681-693.

Giordani, A. and Moschitti, A. (2010), "Semantic mapping between natural language questions and sql queries via syntactic pairing", *International Conference on Applications of Natural Language to Information Systems*, pp. 207-222, doi:10.1007/978-3-642-12550-8_17.

Gupta, A., Akula, A., Malladi, D., Kukkadapu, P., Ainavolu, V. and Sangal, R. (2012), "A novel approach towards building a portable nlidb system using the computational paninian grammar framework", *International Conference on Asian Language Processing (IALP), IEEE, November*, pp. 93-96.

Hainaut, J.-L. (2002), "Introduction to database reverse engineering", LIBD Lecture Notes, University of Namur, avialable at: www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf (accessed October 2005).

Hallett, C. (2006), "Generic querying of relational databases using natural language generation techniques", *Proceedings of the Fourth International Natural Language Generation Conference, Association for Computational Linguistics*, *July*, pp. 95-102.

Hamaz, K. and Benchikha, F. (2012), "Semantic enrichment of relational databases", in Sobh, T. and Elleithy, K. (Eds), Second International Workshop on Advanced Information Systems for Enterprises (IWAISE), IEEE, November, Dordrecht, pp. 15-19.

Hamaz, K. and Benchikha, F. (2013), "From relational databases to ontology-based databases", *ICEIS, Angers*, *July*, pp. 289-297.

Jing, L., Li, M., Lei, Z., Jean-Sébastien, B., Chen, W., Yue, P. and Yong, Y. (2007), "SOR: a practical system for ontology storage, reasoning", *VLDB 2007, 33rd Very Large Data Bases Conference, University of Vienna*, *23-27 September*, pp. 1402-1405.

Johannesson, P. (1994), "A method for transforming relational schemas into conceptual schemas", *Proceedings of the Tenth International Conference on Data Engineering IEEE Computer Society, Washington, DC, 14-18 February*, pp. 190-201.

Kashyap, V. (1999), "Design and creation of ontologies for environmental information retrieval", *Proceeding of the 12th Workshop on Knowledge Acquisition, Modeling and Management, Alberta*, *8-23 April*, pp. 1-18.

Khamis, R. (2010), "An approach for developing natural language interface to databases using data synonyms tree and syntax state table", in Sobh, T. and Elleithy, K. (Eds), *Innovations in Computing Sciences and Software Engineering*, Springer, Dordrecht, pp. 509-514.

Li, F. and Jagadish, H.V. (2014), "NaLIR: an interactive natural language interface for querying relational databases", *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, ACM*, *June*, pp. 709-712.

Martin, P., Appelt, D.E., Grosz, B.J. and Pereira, F. (1986), "TEAM: an experimental transportable natural-language interface", *Proceedings of 1986 ACM Fall Joint Computer Conference, IEEE Computer Society Press*, *November*, pp. 260-267.

Minock, M. (2010), "C-phrase: a system for building robust natural language interfaces to databases", *Data & Knowledge Engineering*, Vol. 69 No. 3, pp. 290-302.

Pazos, R.R.A., González, B.J.J., Aguirre, L.M.A., Martínez, F.J.A. and Fraire, H.H.J. (2013), "Natural language interfaces to databases: an analysis of the state of the art", in Castillo, O., Melin, P. and Kacprzyk, J. (Eds), *Recent Advances on Hybrid Intelligent Systems*, Berlin, pp. 463-480.

Pierra, G., Dehainsala, H., Aït-Ameur, Y. and Bellatreche, L. (2005), "Base de données à base ontologique: principes et mise en oeuvre", *Ingénierie des Systèmes d'Information*, Vol. 10 No. 2, pp. 91-115.

Popescu, A.M., Armanasu, A., Etzioni, O., Ko, D. and Yates, A. (2004), "Modern natural language interfaces to databases: composing statistical parsing with semantic tractability", *Proceedings of the 20th International Conference on Computational Linguistics, Association for Computational Linguistics*, *August*, pp. 41-47.

Ramanathan, S. and Hodges, J. (1997), "Extraction of object-oriented structures from existing relational databases", *ACM SIGMOD Record*, Vol. 26 No. 1, pp. 59-64.

Robin, C.R. and Uma, G.V. (2010), "A novel algorithm for fully automated ontology merging using hybrid strategy", *European Journal of Scientific Research*, Vol. 47 No. 1, pp. 074-081.

Shah, A., Pareek, J., Patel, H. and Panchal, N. (2013), "NLKBIDB-Natural language and keyword based interface to database", *International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE*, *August*, pp. 1569-1576.

Stojanovic, N.L., Stojanovic and Volz, R. (2002), "Migrating data-intensive web sites into the semantic web", *Proceedings of the 17th ACM Symposium on Applied Computing*, *Madrid*, *10-14 March*, pp. 1100-1107.

Thomas, R.G. (1993), "Towards principles for the design of ontologies used for knowledge sharing", International Workshop on Formal Ontology", International Workshop on Formal Ontology, Padova, March, pp. 1-22.

Wilkinson, K., Sayers, C., Kuno, H. and Reynolds, D. (2003), "Efficient RDF storage and retrieval in Jena2", *Proceedings of the First International Conference on Semantic Web and Databases*, *Aachen*, *7-8 September*, pp. 120-139.

Zdenka, T. (2010), "Relational database as a source of ontology creation", *Proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 135-139.

Further reading

Jean, S., Aït Ameur, Y. and Pierra, G. (2006), "Querying ontology based databases the ontoql proposal", *Proceedings of the 18th International Conference on Software Engineering & Knowledge Engineering*, *San Francisco, CA*, *5-7 July*, pp. 166-171.

Corresponding author
Kamal Hamaz can be contacted at: hamaz.kamal@gmail.com