I.J. Intelligent Systems and Applications, **2016, 2, 35-44**
Published Online February 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2016.02.05

A Model of a Generic Natural Language Interface for Querying Database

Hanane Bais[1], Mustapha Machkour[2] and Lahcen Koutti[3]
Information Systems and Vision Laboratory, Department Computer Sciences, Faculty of Sciences, Ibn Zohr University, Agadir, Morocco
E-mail: [1]baishanan@gmail.com, [2]machkour@hotmail.com, [3]lkoutti@yahoo.fr

***Abstract*—**Extracting information from database is typically done by using a structured language such as SQL (Structured Query Language). But non expert users can't use this later. Wherefore using Natural Language (NL) for communicating with database can be a powerful tool. But without any help, computers can't understand this language; that is why it is essential to develop an interface able to translate user's query given in NL into an equivalent one in Database Query Language (DBQL).

This paper presents a model of a generic natural language query interface for querying database. This model is based on machine learning approach which allows interface to automatically improve its knowledge base through experience. The advantage of this interface is that it functions independently of the database language, content and model. Experimentations are realized to study the performance of this interface and make necessary corrections for its amelioration

***Index Terms*—**Databases, Natural Language Processing (NLP), Database Query Language (DBQL), Intermediate XML logical Query (IXLQ), Extended Context Free Grammar (ECFG), XML Schema, Auto Generator of Syntactic Rules (AGSR), Module of Natural Language Query Definitions (MNLQD), Machine Learning.

I. Introduction

Natural Language Processing (NLP) is one of the most active techniques used in Human-Computer Interaction. It is a branch of Artificial Intelligence (AI) that is used for information retrieval, machine translation and linguistic analysis. The main objective of NLP is to allow communication between human and computers without memorizing commands and complex procedures [1, 6]. In other words, NLP allows computer to understand NL. Moreover the NL is easy to learn and use.

One of the classic problems in the field of NLP which particularly has recently attracted the attention of the research community in this area is the Natural Language Database Interface (NLDBI).

The objective of NLDBI is to extract information from Database using NL [3, 2]. In this sense, the database user doesn't need to have expertise in programming language to access data from database. Traditionally, people are used to work with a form, but their anticipations strongly depend on the capabilities of this form. Wherefore the using of NLDBI can provide powerful improvements to the use of data stored in a database. It offers to a large number of users of database a simple, uniform and unlimited access to data without learning any DBQL.

The remainder of the paper is organized as follows. In section 2 we give an overview of existing work showing their advantages and limitations. Section 3 presents a brief description of the proposed system. Section 4 details the architecture of the system. Section 5 reports the experimental results. Finally, section 6 presents the conclusions and possible extensions of this work.

II. Related Work

Many researches have been done in the field of NLP and one of the most important successes of NLP since it started is NLDBI systems. The success in this area is due of both the actual helps coming from NLDBI systems and NLP that works very well in a single-database domain. In general, databases deal with small enough domains that ambiguity problems in NL can be resolved successfully [4].

The earliest research has been started since1960s [5, 6, 2]. Since this date several systems have been created. . BASEBALL and LUNAR (1972), appeared in late sixties, were the first operational NLDBI. The BASEBALL system was designed to answer questions about baseball games [5, 2]. LUNAR Contained chemical analysis of moon rocks. It uses an Augmented Transition Network [7, 2]. However, these systems were Non-Reconfigurable system. They were designed for a particular domain and thus could not be easily modified to interface other different databases.

By late 1970s, various research prototypes were implemented, like LIFER/LADDER (1978). It was developed for information about US Navy ships and was considered one of the first good NLDBI systems [8]. The system uses semantic grammar techniques that include syntactic and semantic processing. Although, systems based on semantic grammars proved difficult to interface to different application domains since a different grammar had to be developed if LADDER would be used with a different application domain. Another system namely CHAT-80 (1980) is one of the most referenced NLP systems in the eighties. This system was implemented in Prolog. In which English query is converted into prolog expressions, which were evaluated against the Prolog database. The code of CHAT-80 was widely circulated and formed the basis of several other experimental systems such as MASQUE (i.e. Modular Answering System for Queries) [3]. A modified version of The MASQUE is MASQUE/SQL [10, 2] system. It translates the NL query into an intermediate logic representation, and then translates the logic query into SQL. But this system has some Shortcoming. If the SQL query fails, the system does not identify which part of the query produced the failure. It is also domain dependent and must be configured for other knowledge domains.

Recently many NLDBI are developed such as the PRECISE system (2004) and NALIX (Natural Language Interface for an XML Database) (2006). PRECISE is developed at the University of Washington. It is one of the best examples based on approaches interesting to the Design of NLDBIs that are database independent [10]. By combining the latest advances in statistical parsers with a new concept of semantic tractability, PRECISE becomes easily highly reconfigurable system. Therefore PRECISE is able to perform impressively in semantically tractable questions. However the system had the problem of treatment nested structures.

NALIX is the first generic interactive Natural language query interface to XML database (extensible markup language). This system is developed at the University of Michigan. The processes of transformation used in NALIX are done in three phases: generating a Parse tree, validating the parse tree, and translating the parse tree to An XQuery expression. So NALIX can be classified as syntax based system. The system uses Schema-Free XQuery as the database query language. The advantage of Schema-Free XQuery is that it is not necessary to map a query into the exact database schema, since it will automatically find all the relations given certain keywords [12].

In this paper we cope with tree issues present in many NLDBIs. The first one is that some NLDBIs are based on intermediate representation [13]. In these the NLQ is translated into a logical query and this latter is translated into DBQL. But not all forms of logical query are independent of database language. The second problem exists in systems that use syntactic parsing in which the syntax rules describing the terminal symbols are domain-dependent [15].So to be ported to another domain these rules must be manually changed. The last problem is that many NLDBI don’t improve the waiting time for translating the questions already processed. In the next sections we propose some solutions to resolve these problems.

III. Proposed System

In this section we present a brief description of our system. the proposed idea consist of the implementation of a generic natural language query interface for relational database based on machine learning approach .The system architecture is based on intermediate representation language. Firstly, the NL query is parsed syntactically, and then the parser tree translated by the semantic analyzer into an Intermediate XML Logical Query (IXLQ). Then The IXLQ is translated to an expression in the DBQL such as SQL, and evaluated against the database system. By using the intermediate language approach, our system is divided into two parts. One part starts from a natural language query to the generation of IXLQ. The second part starts from a logical query until the generation of DBQL. The idea to express logical Queries in XML form has the advantage of being independent of both the database language and natural language. Consequently the system is independent of the database language, content and model (Relational, Relational-Object, Object, XML, and so on).

For parsed natural language query, our system uses two types of syntactic rules. The first type includes the syntactic rules linking non-terminal symbols (non-leaf nodes in the parse tree) and the corresponding semantic rules are domain-independent rules (i.e. they can be used in any application domain). These rules are described using an extended context free grammar (ECFG). The second type is the syntactic rules linking terminal symbols (leaf nodes) are domain-dependent rules; these rules are generated automatically by an Auto Generator of Syntactic Rules (AGSR). With this generator our system becomes learning and generic system and then it automatically improves through experience its knowledge base.

By using some syntactic and semantic rules the user can ask a question with multi-queries. Our system treats this question as a complex query, divides it to elementary queries and displays the result corresponding to each query.

In our system we use a Module of Natural Language query definitions (MNLQD). This module contains a set of classes of IXLQ. Each class represents a logical interpretation of many natural language queries. The integration of this part in our system is going to minimize the waiting time for translating the questions already asked by the user. In the following sections of this article, we will clearly describe this module.

IV. System Architecture

The process of transformation of natural language query cited above is illustrated in the Figure 1, which represents the architecture of our system.

As illustrated in figure 1, the proposed architecture is divided into tree modules: the linguistic component module, the database knowledge component module and MNLQD.

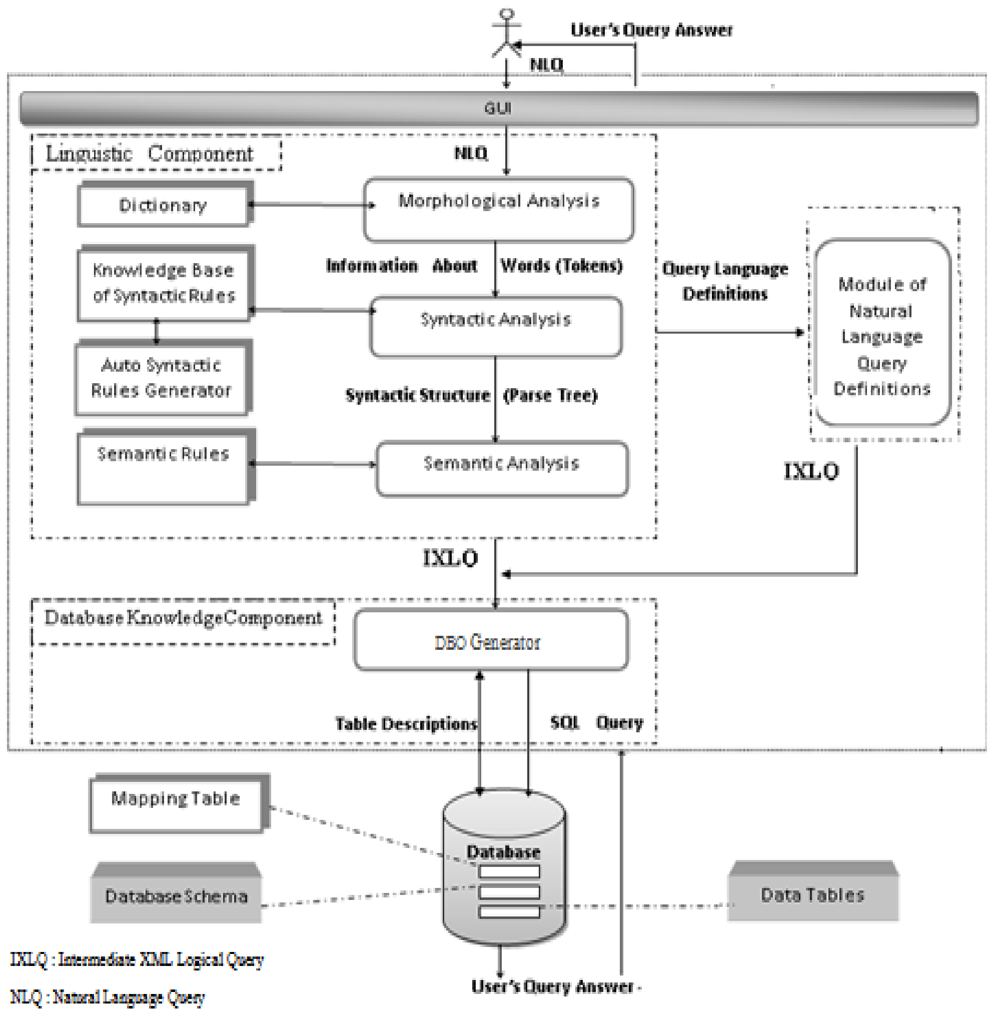

User's Query Answer
NLQ
GUI
Linguistic Component
NLQ
Dictionary
Morphological Analysis
Information About Words (Tokens)
Knowledge Base of Syntactic Rules
Syntactic Analysis
Auto Syntactic Rules Generator
Syntactic Structure (Parse Tree)
Semantic Rules
Semantic Analysis
Query Language Definitions
Module of Natural Language Query Definitions
IXLQ
IXLQ
Database Knowledge Component
DBQ Generator
Table Descriptions
SQL Query
Mapping Table
Database
Database Schema
Data Tables
IXLQ : Intermediate XML Logical Query
NLQ : Natural Language Query
User's Query Answer

Fig.1. System Architecture

The first module controls the linguistic aspect, where the natural language Query is submitted a many analysis operations (morphological, syntactic and semantic analysis). At the end of this procedure, we obtain an intermediate XML logical query (IXLQ). This expression corresponds to the XML interpretation of the initial natural language Query. The second module is used to translate IXLQ to DBQ expression. This latter is sent the to the database jet for producing the answer. In the next we explain in depth how these modules work.

By separating between linguistic component module and the database knowledge component module is the guarantee that the system can be ported to another relational database.

The Module of Natural Language Query Definitions (MNLQD) to reduce the waiting time for translating questions already processed.

A. Linguistic Component

The linguistic component performs three analyses: morphological, syntactic and semantic explained as follows:

1) Morphological Analysis

The purpose of the morphological analysis is to segment the text into individual units that called tokens and determines the different characteristics of these units. This process is performed by the following functions:

- **Token analyzing**: this function is used to split the input sentence in primitive units called tokens, which is considered as a single logical unit.
- **Spelling checker**: this function ensures that each token is in the system dictionary, if this is not the case, then spell checking is performed or a new word is added to the system vocabulary.
- **Ambiguity reduction**: this function minimizes the ambiguity in a sentence to simplify the task of the next analysis by replacing several words or symbols with canonical internal words.
- **Tagger**: this function determines the grammatical category of each token.
- **Morpheme**: this function is used to determine the morpheme of each token.

2) Syntactic Analysis

After the morphological analysis stage, the syntactic analysis is used to show how the words of query entered by the user are related to each another. This transaction will enable our system to know the syntactic structure of the request and explain the dependency relationships between different words. This is done by applying a set of syntactic rules. These rules constitute a formal grammar, which describe the grammatical structures of the request. Our system uses the Extended Context-Free Grammar (ECFG) [14], presented in the figure2.

- S→ QU'AUX_V OBS (CONDITION) (ORDER) (CONJ S)
- S→ OBS (CONDITION) (ORDER) (CONJ S)
- S→ VP OBS (CONDITION) (ORDER) (CONJ S)
- VP → V (PRON)
- AUX_V →IS | ARE |WANT
- QUE→WHO | WHAT | WHERE | WHICH
- OBS → OB (CONJ OBS)
- OB → NP
- NP → (DET) ADJ_EXPR (CONJ NP)
- NP → (QUANT) (POSADJ) (DET) N (CONJ NP)
- NP → NP PP (CONJ NP)
- QUANT → ALL | ANY | EVERY
- PRON → ME | US | THEM | HER
- POSADJ → MY| YOUR| HIS |HER| ITS | OUR
- PP → PREP NP
- PREP
- → OF | IN | AT | TO |ON
- ADJ_EXPR → (ADJ) NP
- CONDITION→COND OP (CONJ CONDITION)
- COND → WHERE | WHOSE | WHOM | HAVING | WITH
- OP →OB SYMBOL VALUE
- SYMBOL→ IS | = | > | >= | < | <= | <> |IN | LIKE
- ORDER → ORD NP (CONJ ORDER)
- ORD→ ORDER BY | SORTED BY | ACCORDING TO
- CONJ → AND | OR

Fig.2. Extended Context-Free Grammar used by the system

The syntax rules represented in the above ECFG are linking non-terminal symbols and are domain-independent rules. The syntax rules linking terminal symbols are domain-dependent. These rules are generated by an Auto Syntactic Rules Generator (ASRG).

This ASRG is based on machine learning approach, we use This approach due to its experienced considerable growth in recent years, and its interactions with the NLP are increasingly close and frequent, therefor it allows the system to automatically improve through experience of its knowledge base [11].the aim of ASRG is to check whether all the syntactic rules necessary to parse the user query exist in the system knowledge base. If not, it detects automatically the necessary syntactic rules; it creates and adds them to the knowledge base. This part of our system will help to adapt its knowledge base with user requests and therefore it can function regardless of the database domain and it will be generic system.

The result of syntactic analysis is a description of the syntactic structure of natural language query in the form of a derivation tree or parse tree (see Figure 3). A parse tree is composed of nodes and branches; each node can be a root node, a branch node or a leaf node. An interior node of a parse tree is a phrase and is called a non-terminal of the grammar, and a leaf node is a word and is called a terminal of the grammar [16].

The figure 3 displays the parse tree representing the syntactic structure of the NLQ "Show me the address of client whose age > 25 and name is 'AHMED' ":

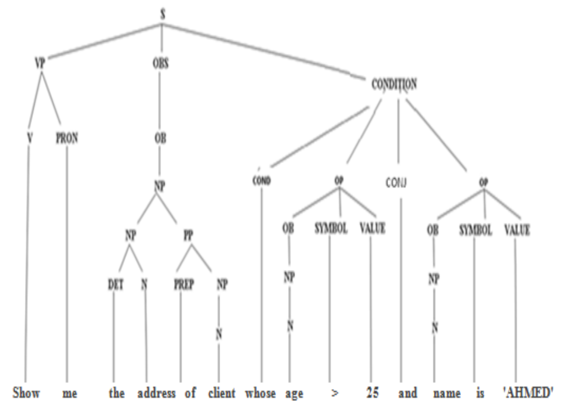


Fig.3. Parser tree of the NLQ "Show me the address of client whose age > 25 and name is 'AHMED'"

3) *Semantic Analysis*

The overall objective of the semantic analysis is to assign a logical meaning to the parse tree created by the syntactic analysis. This is done by applying a set of semantic rules, which are used to translate the parse tree to a logical query.

Each syntactic rule defined in Figure 2 has a corresponding semantic rule. For this reason the translation process is called rule-by-rule style [17]. In our system we use a particular model of semantic rules. Table1 shows some examples of semantic rules with their corresponding syntactic rules:

Table 1. Semantic rules with their corresponding syntax rules

Semantic rule	corresponding syntactic rule
<attribute > pre <object>	NP PREP NP
< attribute 1> cc < attribute 2 > pre <object>	NP CONJ NP PREP NP
< attribute > pre <object1> cc < object2 >	NP PREP NP CONJ NP
<attribute> pre <object> symbol <attribute value>	NP PREP NP SYMBOL VALUE

We have already mentioned in the previous paragraph that the application of semantic rules on the parse tree produces the IXLQ. In xml, we can define the structure of IXLQ by the following XML Schema:

```
<xs:schema elementFormDefault="qualified">
 <xs:element name="REQUEST" type="REQUEST"/>
  <xs:complexType name="REQUEST">
   <xs:sequence>
    <xs:element name="SELECT" type="SELECT"/>
    <xs:element name="COND" type="COND"
minOccurs="0"  maxOccurs="unbounded"/>
    <xs:element ref="ORDER" type="ORDER"
minOccurs="0"  maxOccurs="1"/  >
   </xs:sequence>
 </xs:complexType>
 <xs:complexType name="SELECT">
   <xs:sequence>
    <xs:element name="OBJECT" type="OBJECT"
minOc curs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
 <xs:complexType name="COND">
   <xs:sequence>
    <xs:element name="OBJECT" type="OBJECT"/>
    <xs:element name="SYMBOL"   type="xs:string"/>
    <xs:element name="VALUE" type="xs:string"
minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="ORDER">
    <xs:sequence>
  <xs:element name="OBJECT" type="OBJECT"
minOccurs="1" >
   </xs:sequence>
</xs:complexType>
<xs:complexType name="OBJECT">
  <xs:sequence>
    <xs:element name="ATTRIBUT" type="ATTRIBUT"
minOccurs="1" maxOccurs="unbounded"/>
    <xs:element name="Name"   type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ATTRIBUT">
   <xs:sequence>
    <xs:element name="AGGREGA" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="Name"   type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

The following IXLQ displays the logical query associated to the parse tree of the NLQ "Show me the address of client whose age >25 and name is 'AHMED' " (see Figure 3):

```
<REQUEST>
  <SELECT>
    <OBJECT>
       <NAME> client </NAME>
        <ATTRIBUT>
          <NAME> address </NAME>
         </ATTRIBUT>
    </OBJECT>
  </SELECT>
  <COND>
      <OBJECT>
        <ATTRIBUT>
          <NAME> age </NAME>
        </ATTRIBUT>
    </OBJECT>
    <SYMBOL> > </SYMBOL>
     <VALUE> 25 </VALUE>
  </COND>
   <COND>
     <OBJECT>
       <ATTRIBUT>
         <NAME> name </NAME>
       </ATTRIBUT>
      </OBJECT>
       <SYMBOL> is </SYMBOL>
       <VALUE> AHMED </VALUE>
  </COND>
</REQUEST>
```

During the generation of the IXLQ, the system uses a Module of Natural Language Query Definitions (MNLQD). This module is composed of a set of classes. Each class contains multiple queries in natural languages that have the same logical interpretation.

The integration of MNLQD in our system has multiple advantages. It becomes able to reuse the already processed query and then reduces the query response time. The function of this module is based on a classification method that's used to classify natural language queries according to their logical interpretation (IXLQ). The graph in Figure 4 describes the operation of MNLQD:

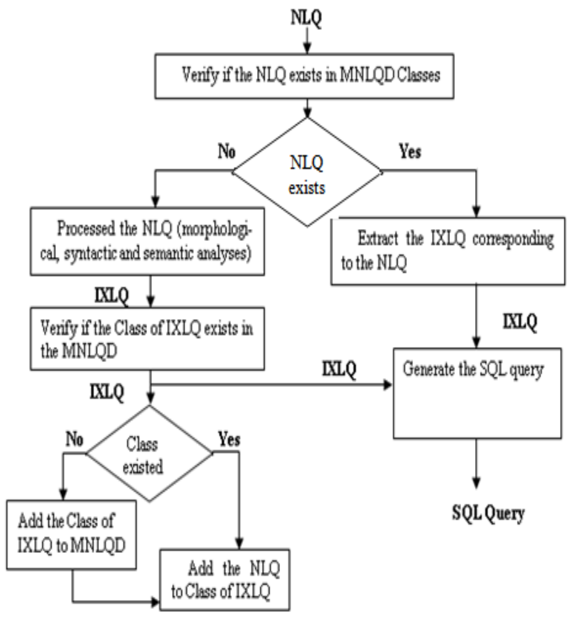

NLQ
Verify if the NLQ exists in MNLQD Classes
No
NLQ exists
Yes
Processed the NLQ (morphological, syntactic and semantic analyses)
Extract the IXLQ corresponding to the NLQ
IXLQ
Verify if the Class of IXLQ exists in the MNLQD
IXLQ
IXLQ
Generate the SQL query
IXLQ
No
Class existed
Yes
Add the Class of IXLQ to MNLQD
Add the NLQ to Class of IXLQ
SQL Query

Fig.4. The operation of MNLQD

B. Database Knowledge Component

The database knowledge component consists of two parts: DBQ generation and DBQ execution.

1) DBQ Generation

The task of the DBQ generation is to translate the IXLQ created by the semantic analyzer into SQL. By mapping each element of the logical query to its corresponding clause in the SQL query. The DBQ generation consists of four phases. Each phase manipulates only one specific part of the SQL query. The concatenation of the results of the four phases constructs the final SQL query.

The first phase deals with the part of the logical query that corresponds to the names of attribute for building the SELECT clause. The second phase, builds the FROM clause by selecting the portion of the logical query that is mapped the table name or a group of table names. The third phase extracts the conditions of selection from of the logical query to construct the WHERE clause. At the fourth phase, we select the portions of the logical query that corresponds to the order of presenting the result of SQL query (i.e. ORDER BY clause). Each one of these phase is followed by a test which consists to verify if the name of tables and attributes, extracted from logical query, are valid or exist in dictionary of database. If it's not the case, the system uses a domain specific dictionary (mapping table) which stores the synonyms of table and attribute names. The mapping table helps the user to write his query with different natural language sentences.

2) DBQ Execution

Once the DBQ is generated it will be executed by the Database Management System (DBMS), and then, displays the answers returned in tabular form.

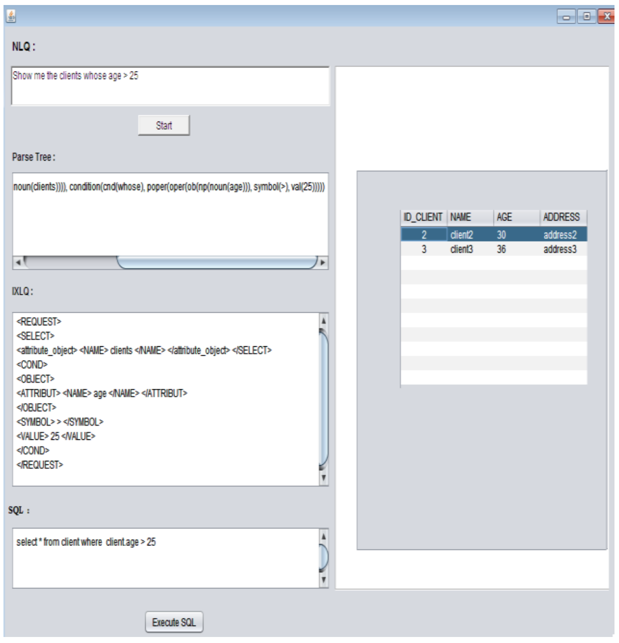
V. System Results

In this section, we present some results of our proposed system. The user can ask the same question with different ways (often more than sixty ones) using different query verbs such as: Show, Find, Tell, Search, Give, List and Display. Also it's possible to put a natural language question without query verb. Each of these questions can be written in eight different syntactic manners (see figure 2).

The interface represented in figure 5 shows the translation of the NLQ: "Show me the clients whose age > 25" into a SQL query. It composes by four textboxes: The first textbox used to enter the user NLQ. The second textbox shows the parse tree of the NLQ entered by the user. While the third textbox displays the logical interpretation of the parse tree. Finally the fourth textbox shows the SQL query. This interface has also two buttons: button "Start" for start the translation processes and button "Execute SQL" for execute the SQL query, and then, displays the answers returned in tabular form.

The following tables show a list of variety of NLQ that are successfully translated and executed by our system.

The first section of these NLQs is queries without projection and selection. In this type of query the user doesn't specify any attribute and any conditions. The table 2 presents some example of these queries.



Show me the clients whose age > 25

noun(clients)))), condition(cnd(whose), poper(oper(ob(np(noun(age))), symbol(>), val(25)))))

```
<REQUEST>
<SELECT>
<attribute_object> <NAME> clients </NAME> </attribute_object> </SELECT>
<COND>
<OBJECT>
<ATTRIBUT> <NAME> age </NAME> </ATTRIBUT>
</OBJECT>
<SYMBOL> > </SYMBOL>
<VALUE> 25 </VALUE>
</COND>
</REQUEST>
```

```
select * from client where  client.age > 25
```

ID_CLIENT	NAME	AGE	ADDRESS
2	client2	30	address2
3	client3	36	address3

Fig.5. The system interface

Table 2. NLQ without projection and selection

Natural language query	Generated SQL	Comment
Give projects	SELECT * FROM project	Many ways to ask the same question. All these NLQ belong to the same MNLQD class
Give all projects		
Give me our projects		
Give me all projects		
Give me projects		
Give our projects		
Give me all our projects		
Give all our projects		
Projects?		
What are our projects?		
All our projects?		
list all our clients	SELECT * FROM customer	Use of the Synonymous of table name
list all our customers		
Show me all our clients and projects	SELECT * FROM customer	Question with multi-queries
	SELECT * FROM project	

The table 3 displays some example the second section of NLQs, which are queries with projection and without selection. In This type of query the user identify some attribute, but he doesn't specify any conditions

Table 3. With Projection and without Selection

Natural language query	Generated SQL	Comment
give me the names of our employee	SELECT employee.name FROM employee	query with one attribute
give me all our clients names		
Names, age and sex of student	SELECT student.name, student.age, student.sex FROM student	query with many attributes
display me all our student names and age and sex		
What are our projects labels	SELECT project.designation FROM project	Using the Synonyms of attribute name
Find all the names of clients and projects	SELECT customer.name FROM customer	Question with multi-queries
	SELECT project.disignation FROM project	

Table 4. Queries with selection

Natural language query	Generated SQL	Comment
display all clients whose names are "Hanane"	SELECT * FROM customer where customer.names =‘Hanane’	Conditioned query without selected attributes
all our client whose names are "Hanane" or "Mustapha"	SELECT* customer FROM customer where customer.names in(‘Mustapha’, ‘Hanane’)	
display all invoice whose sum is between 1000 and 20000	SELECT * FROM invoice where invoice.amount between 1000 and 20000	
Search all customers whose names ends with p	SELECT * FROM customer where customer.names like ‘ percentp’	
What are our Clients names and sex whose address is "Hay Dakhla Agadir"?	SELECT customer.name, customer.sex FROM customer where customer.address =‘Hay Dakhla Agadir’	Conditioned query with specific Selected attributes
display all sum of invoice where sum is more than 1000	SELECT invoice.amount FROM invoice where invoice.amount > 1000	
Give me the addresses of clients where age greater than or equal to 25 and name is "Hanane"	SELECT customer.address, customer.sex FROM customer where customer.names =‘Hanane’ and customer.age >= 25	
give me the names of all our clients and projects and show me the clients whose age is greater than 20	SELECT * FROM customer	Question with multi-queries
	SELECT * FROM project	
	SELECT * FROM customer where customer.age > 25	

The third section of NLQs is queries with selection; these queries can be with projection or without projection. In this question the user defined some specific criteria. Table 4 depicts some examples of conditioned query.

The last section of questions is a query with aggregate function. Table 5 depicts some examples of those queries:

Table 5. Query with aggregate function

Natural language query	Generated SQL
Give me the number of clients whose names are "Hanane"	SELECT COUNT (*) AS NB_client FROM customer where customer.names =‘Hanane’
Count me all our project	SELECT COUNT (*) AS NB_project FROM project
Display me the totality of invoice amount where amount of invoice is older than 1200	SELECT SUM(invoice.amount) AS SUM_INVOICE_AMOUNT FROM invoice where invoice.amount >1200
Display me the average of invoice amount	SELECT AVG(invoice.amount) AS AVG_INVOICE_AMOUNT FROM invoice
Give me the smallest amount of invoice	SELECT MIN (invoice.amount)AS MIN_INVOICE_AMOUNT FROM invoice
show the max amount of invoice where sum is less than 1000	SELECT MAX (invoice.amount)AS MIN_INVOICE_AMOUNT FROM invoice where invoice.amount < 1000
What is the invoice with the max amount?	SELECT * FROM invoice where invoice.amount in (SELECT MAX (invoice.amount) FROM invoice)

We have tested the performance of system by a set of **13493**NLQ, which created using a program. To classify the result obtained with this experiment, we use the decision tree show in figur6.

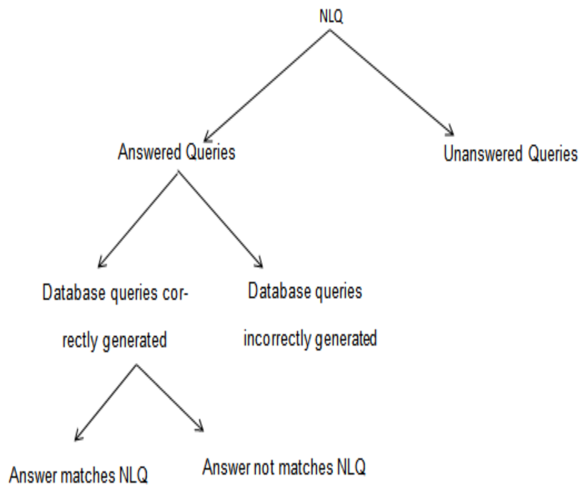


Fig.6. Decision tree

We said that query is correctly generated if the Database queries produced is syntactically correct. The different results obtained by our test were tabulated in table 6.

Table 6. The results obtained by the first test

NLQ	Answered Queries		Unanswered Queries
13493 (100 percent)	12622 (93.5480 percent)		871 (6.4508 percent)
	DBQ Correctly generated	DBQ Incorrectly generated	
	11956 (94.7948 percent)	666(5.2051 percent)	

The Table 6 shows that our system gave an answer of 94.79 percent in the 13493 queries. For 6.4508 percent of queries, the system didn't give any answer. 11956 NLQs are correctly converted into DBQ. While 666 queries produce incorrect database queries.

To find explanation for errors, we have examined the output of the NLQ and obtained the result presented at Table 7.

Table 7. The result of errors explanation

	Error in Incorrectly generated (666)	Error in Unanswered queries (871)
Error in Morphological analysis	151 (22.9832 percent)	64 (7.3478 percent)
Error in syntactic analysis	111 (15.5251 percent)	807 (92.6521 percent)
Error in Semantic analysis	389 (58.2085 percent)	0 (0 percent)
Error in Generation of database query	15 (2.2831 percent)	0 (0 percent)

From Table 7, we observe that the Morphological analysis produced 7.34 percent of errors in unanswered queries and 22.983 percent of errors in incorrectly generated queries. Generally, one of the reasons of these errors is that in some NLQ the system considers some verb like noun. For Example in the query "count all clients whose ages > 50" the token "count" is considered as a noun not a verb.

We also observe that syntactic analysis has resulted around 15.52 percent of errors in the 666 queries that are incorrectly generated and 92.6521 percent of Errors in unanswered queries. These errors occurred because the parser generates some parse trees that don't match the initial NLQ. For instance, in the query "Display me the sum of ages and names of clients", the parser considered the chunk “sum of age” refers to the age object and the chunk “names of client” refers to the client object, whereas both of them refer to one object: "clients".

The errors in semantic analysis represent 58.20 percent of the total errors found in database queries that are incorrectly generated. The cause of these errors is that the semantic analysis doesn't convert correctly the parser tree into IXLQ. As example in the sentence: "give me the ages and names of clients and employees", the parse tree demonstrates that the "names" and "ages" refer to "client" and "employee", however the IXLQ result of this parse tree shows that "name" and "age" refer just to "client".

The errors relate to the generation of database query has just been 2.28 percent. These errors can be avoided at program level.

For the Queries that are correctly generated, not all of the generated DBQ match NLQs. The Table 8 displays the number of DBQ matches NLQ and the numbers of DBQ don't match NLQ.

Table 8. DBQ matches NLQ

	DBQ matches NLQ	DBQ not matches NLQ
Number of queries (11956)	10900	1056
percent	91.1676 percent	8.8323 percent

As presented in Table 8, we show that 91, 99 percent of database queries that are correctly generated match NLQ, while 8.83 percent of answers not match NLQ.

In the figure 7 present the decision tree decorated by values obtained by the experimentations.

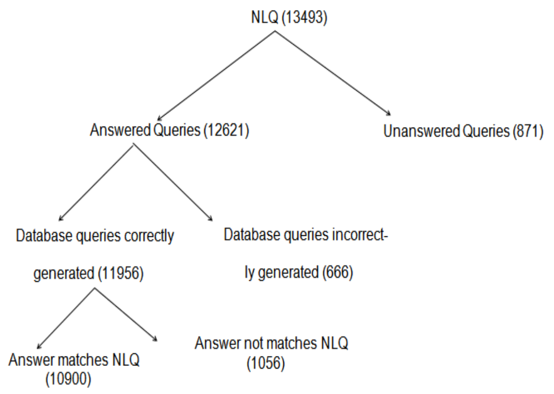

NLQ (13493)
Answered Queries (12621)
Unanswered Queries (871)
Database queries correctly generated (11956)
Database queries incorrectly generated (666)
Answer matches NLQ (10900)
Answer not matches NLQ (1056)

Fig.7. Decision tree decorated by obtained values

The table 9 shows the results for runtime minimization of the some NLQ when they are executed on a notebook with AMD C-50 processor 1.00 GHz and 2GO of RAM

Table 9. Results for runtime minimization

	runtime of query asked in first time (s)	runtime if query is already asked (s)	performance achieved
queries without projection and selection	7.80	0.26	96.6 percent
queries with projection and without selection	8.85	0.43	95.14 percent
queries without projection and with selection	9.92	1.55	84.37 percent
queries with projection and selection	10.76	1.917	82.24 percent
Query with aggregate function	10.452	1.87	82.10 percent

VI. Conclusion and Future Works

This research paper presents a study on constructing a generic natural language query interface for database based on machine learning approach. The main objective of this system is to allow communication between database and its users using natural language. The system has the capabilities to translate natural language query into an equivalent SQL query after processing through multiple steps and generates answers in tabular form. The primary advantages of this system are that it's independent of the database language, domain and model and it's able to automatically improve its knowledge base through experience.

The results show that the techniques used by our system can produce reasonable answers for very important types of natural language queries also the integration of MNLQD in our system provides important advantages. It reduces the runtime for translating the questions already asked by the user.

As future work we intend to continue to solving more complex queries and using other mechanism for better performance. Also we would like to construct a generic multi-langue query interface for database.

References

[1] Gauri Rao, et al., 'Natural language Query Processing Using Semantic Grammar', in International Journal on Computer Science and Engineering, Vol. 02, pp.219-223, 2010.

[2] Avinash J. Agrawal, Dr. O. G. Kakde, 'Semantic Analysis of Natural Language Queries Using Domain Ontology for Information Access from Database', in I.J. Intelligent Systems and Applications, 12, pp. 81-90, 2013.

[3] N. Nihalani, S. Silakari, M. Motwani., 'Natural language interface for database: a Brief review', in International Journal of Computer Science Issues 8 (2), pp.600-608, 2011.

[4] Jasmeen Kaur, Bhawna chauhan and Jatinder Kaur Korepal, 'Implementation of Query Processor Using Automata and Natural Language Processing', in International Journal of Scientific and Research Publications, Vol. 3, Issue 5, pp.1-5, 2013.

[5] Androutsopoulos, G.D. Ritchie, and P. Thanisch, 'Natural Language Interfaces to Databases – An Introduction', in Journal of Natural Language Engineering 1 Part 1, pp.29-81,1995.

[6] Huangi, Guiang Zangi, Phillip C-Y Sheu, 'A Natural Language database Interface based on probabilistic context free grammar', in IEEE International workshop on Semantic Computing and Systems, 2008.

[7] Woods, W. A. Progress in natural language understanding: An application to LUNAR geology. AFIPS Natl. Computer. Conj: Expo. Conference Proc. 42, pp.441-450, 1973.

[8] Hendrix, G.G., Sacerdotal, E.D., Sagalowicz, D., Slocum, J. 'Developing a natural language interface to complex data' in ACM Transactions on database systems, 3(2), pp.105-147, 1978.

[9] Androutsopoulos, I., Ritchie, G.D., and Thanisch, P, 'MASQUE/SQL - An Efficient and Portable Natural Language Query Interface for Relational Databases', in Proceedings of the 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems,1_4 June1993 Edinburgh, Scotland, pp.327-330, 1993.

[10] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko and Alexander Yates., 'Modern Natural Language Interfaces to Databases', Composing Statistical Parsing with Semantic Tractability COLING, 2004.

[11] Y.W. Wong,' learning for semantic parsing using statistical machine translation techniques', in Technical Report UT-AI-05-323, University of Texas, Austin, 2005.

[12] Y. Li, H. Yang, H.V. Jagadish, 'NALIX: an interactive natural language interface for querying XML', in Proceedings of the International Conference on Management of Data, pp.900-902, 2005.

[13] Minock, M, 'C-Phrase: A System for Building Robust Natural Language Interfaces to Databases', in Journal of Data Engineering (DKE), 69(3), pp.290-302.

[14] Jurgen Albert, Dora Giammarresi and Derick Wood. (2001), 'Normal form algorithms for extended context-free grammars', in Theoretical Computer Science 267, pp.35-47, 2010.

[15] C. Manning, H. Schütze, 'Foundations of Statistical Natural Language Processing', MIT Press, Cambridge, 1999.

[16] Luis Tari, Phan Huy Tu, Jorg Hakenberg, Yi Chen, Tran Cao Son, Graciela Gonzalez and Chitta Baral, 'Parse Tree Database for Information Extraction', in IEEE transactions on knowledge & data, engineering, 2010.

[17] P. Reis, J. Matias and N. Mamede, 'Edit – A Natural Language Interface to Databases': A New Dimension for an Old Approach', in Proceedings of the Fourth International Conference on Information and Communication Technology in Tourism (ENTER' 97), Edinburgh, 1997.

Authors' Profiles

Bais Hanane received her Master's degree in Computer Science and Network Systems in 2013 from Departments of Mathematics and Computer Science, Faculty of Science, University Ibn Zohr, Agadir, Morocco. she is currently a Ph.D. candidate of the Ibn Zoher University, Agadir, Morocco. her research interests include DataBase system, natural language processing and artificial intelligence.

Mustapha. Machkour is a professor of higher education, department of Mathematics and computer Sciences, Ibn Zohr University, Agadir, Morocco. Member of Laboratory of Computer Systems and Vision, Faculty of Science, Ibn Zohr University, Agadir Morocco. Current research interests include Image processing, Data security in Information systems, multimedia information, DataBase System, logic and artificial intelligence.

Lahcen Koutti is currently a Professor at Department of Computer Science in Faculty of Science, Ibn Zohr University, Agadir, Morocco. He received the Ph.D. degree in computational physics in 1999 from University Paul Verlaine, France and the Habilitation degree in 2010, from Ibn Zohr University, Morocco. He was a software engineer in a multinational company "ALDATA". His research interests include Artificial Intelligence and Computer Vision. Koutti is a member of the Computer Systems and Vision Laboratory.