IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Revue d'Intelligence Artificielle
Vol. 33, No. 6, December, 2019, pp. 393-401
Journal homepage: http://iieta.org/journals/ria

Method and Apparatus for Querying Relational and XML Database Using French Language

Hanane Bais*, Mustapha Machkour

Information Systems and Vision Laboratory, Department Computer Sciences, Faculty of Sciences, Ibn Zohr University, Agadir 80000, Morocco

Corresponding Author Email: hanane.bais@edu.uiz.ac.ma

https://doi.org/10.18280/ria.330601

Received: 29 August 2019
Accepted: 5 November 2019

Keywords:
intelligent interface, natural language processing, Backus-Naur form, machine learning, linguistic operations

ABSTRACT

The extraction of the information from database systems requires the formulation of queries using database query languages, such as Structured Query Language (SQL). This formulation needs the knowledge of the model and the structure of the database. However, non-expert users cannot write such queries. This is why a lot of works have been developed to request the database in natural language. Historically, most of these works were carried out for English language and they were designed for a specific database model. Some of them function independently of database domain. For the French language, all existing contributions are dependent on database domain and model. The aim of this paper is to present a model of an intelligent interface to query databases using the French language. A simulation model was established through the using of linguistic operations and machine learning. The results indicate that the proposed interface functions independently of the database domain and model (relational and XML) and it can translate a very important number of French natural language query into a database query. The findings of this research may serve to improve the interaction between non-expert French users and databases.

1. INTRODUCTION

In the modern world of computing, several platforms have been designed to allow humans to interact with computers by using natural languages, such as recognition systems, question answering systems and Natural Language Interfaces to Databases (NLIDB). The primary objective of NLIDB systems is to allow users to communicate with the database in the same way they communicate with each other, without the need to memorize commands and complex procedures. Hence, non-expert users don't need to learn any artificial language to query database, only natural language is enough [1, 2]. Traditionally, these users are used to work with forms, but their anticipations strongly depend on the capabilities of these forms. However, the using of NLIDB offers a uniform, simple and unlimited access to data without having skills in the field of databases.

Building an NLIDB for a specific domain is fairly easy and more robust. However, the specificity of NLIDB's field of application prevents its use outside the field for which it was developed. To overcome this limit, several methods have been proposed. These methods assure NLIDBs to move from one domain to another, without using new resources while preserving its performance.

Generally, most of the NLIDBs proposed so far operate independently of domain. However, all these NLIDBs are designated for a specific database model.

In this work we propose a generic NLIDB which functions independently at the same time of domain and model of databases. The proposed interface translates French Natural Language Query (FNLQ) into Database Query (DBQ). The translation process is based on linguistic operations. These operations assist us in extracting from the FNLQ the significant information which adequately represents what the user is looking for.

Due to the difficulties encountered during the direct transformation of the FNLQ towards in DBQ, our system uses an approach based on the intermediate representation of the language. The idea of this approach is to first map the FNLQ to an intermediate logical query that represents an unambiguous interpretation of the FNLQ. The logical query obtained is then translated into a DBQ to be evaluated by the database system to display the expected result.

The remainder of this paper is organized as follows. First, we give an overview of related work, followed by the detailed architecture of the system that we have developed. Then, we show a list of FNLQs that are successfully translated and executed by our system with comments on the results of the experiment. Finally, we present the conclusions and some possible perspectives of this work.

2. RELATED WORK

The majority of NLIDB contributions in the nineties focused on querying relational databases through the use of natural language instead of SQL. However, these systems are designed for a specific application domain.

Androutsopoulos et al. developed an extended version of the MASQUE system [3], called MASQUE/ SQL [4]. This system can interface all commercial databases that support the SQL language. MASQUE/ SQL answer user's questions in English by generating SQL. MASQUE/ SQL can interface any database system that supports SQL code.

One of the intelligent tutoring systems designed to relational database is named SQL-Tutor [5, 6]. Smart tutoring systems

are computer programs that attempt to give students new skills. They are called intelligent because they try to help students as a human guardian [7].

For XML database, NALIX (Natural Language Interface for an XML Database) [8] is a generic and interactive interface, developed at the University of Michigan by Li et al. in 2006. The database used for this system is an XML database (Extensible Markup Language, with Schema-Free XQuery as the DBLQ. This language is designed primarily to retrieve information from XML databases. Its role in relation to XML databases is similar to that of SQL to relational databases.

Generally English is the main language of many countries. Also, it is the secondary language in most multilingual countries. For this, the majority of existing ILNBDs respond to requests written in English. However, other ILNBDs are proposed to access the information stored in a database via the formulation of requests in other languages, such as Spanish [9] Urdu [10], Chinese [11, 12] and Hindi [13, 14] languages.

Furthermore, the most of the NLP projects done so far for the French language were involved with many topics, such as French word knowledge engineering [15] document auto-indexing [16] and information extraction [17]. Only a few of them have dealt with NLIDB.

The first one is the Edite system [18]. Edite is a multi-langue (Portuguese, French, English, and Spanish) natural language interface to relational databases. It translates natural language queries about tourism resources into SQL queries. Edite functions independently of database domain. The second system presents the semantic analysis of queries written in French language and it dedicates for object-oriented database [19].

In general, all the French NLIDB cited above is designed for databases of a particular domain and they interface a particular database model. Furthermore, there is no NLIDB designed to the both of XML and relational database. In that, the aim of this works is to design and implement a NLIDB for querying relational and XML database using the French language. The techniques used by the system help it to function independently of both database domains. Also, it can be easily extended to other models.

3. PROPOSED SYSTEM

Due to the difficulties of directly transform the FNLQ into DBQ, the architecture of the proposed system is based on the intermediate representation language approach. The idea of this approach is to map the FNLQ firstly into a logical query, written in XML. Then, translate this query into a DBQ and submitted it to the database system. By expressing the logical query in XML form, the proposed interface can function independently of the database domain and model (Relational and XML). Figure 1 shows the proposed architecture [20].

The proposed architecture is constituted of two units:

• Linguistic processor: in this later, the FNLQ is submitted to many analyses operations (morphological, syntactic and semantic). We obtain at the end of this procedure the logical interpretation of the FNLQ.

• Database knowledge generator: it used to translate the logical query resulted by the linguistic module into a DBQ. The system generates SQL query for the relational database and XPATH query for the XML database.

The separation of the linguistic processor and database knowledge generator makes the proposed system can function independently of database domains [21].

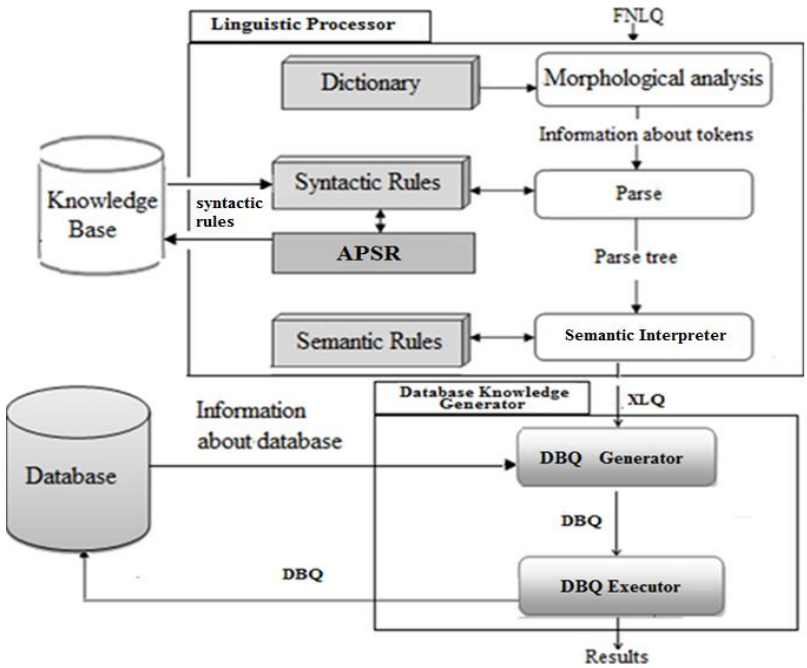

Figure 1. Proposed architecture

3.1 The linguistic processor

The linguistic processor is an important phase in the process of translating the FNLQ into DBQ. In this part, the FNLQ is submitted to many analyses operations: morphological, syntactic and semantic.

3.1.1 Morphological analysis

The morphological analysis is in charge of reading the FNLQ, dividing it into primitive elements called tokens and returning information about each token. This process is performed using the following functions:

(1) Ambiguity reduction: this function helps to reduce the

ambiguity in the FNLQ, by replacing several words or symbols with canonical internal words, as presented in Table 1.

(2) Token analyzing: this function is used to divide the FNLQ into primitive units called tokens. This later is considered as a single logical unit in the FNLQ.

(3) Spelling checker: by using this function, we ensure that each token is in the dictionary used by the interface if this is not the case, the spell checking is performed or a new word is added to the system vocabulary.

(4) Part-of-Speech Tagging: this function determines the grammatical category of each token.

(5) Morpheme: this operation is used to determine the morpheme or the radical of each token.

In Figure 2, we present an example of using the above functions. The FNLQ in this example contents some spelling errors to the impact of spelling checker.

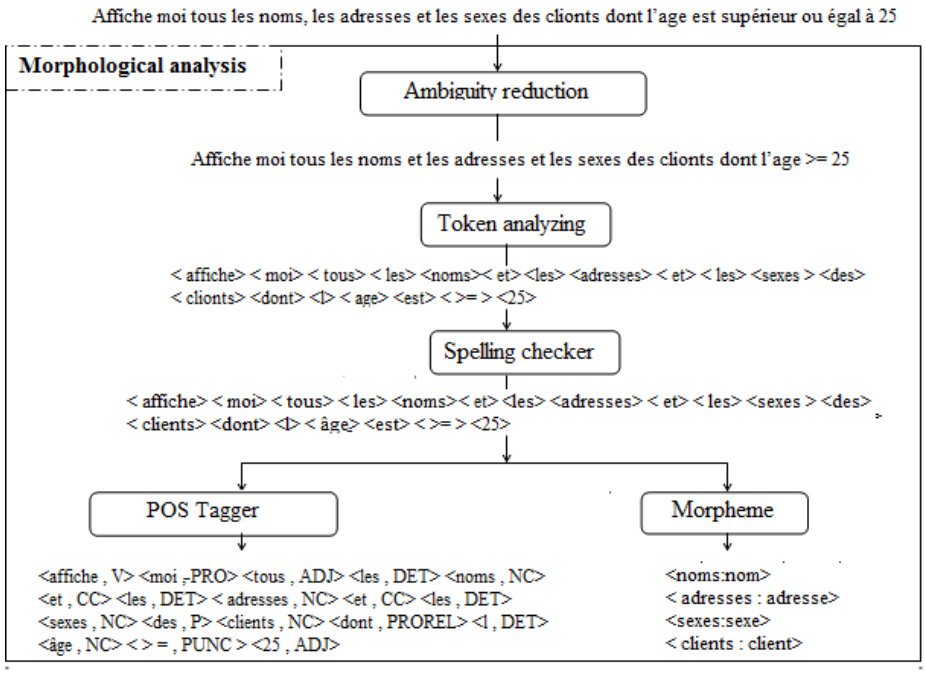

Figure 2. Example of using morphological analysis functions.

3.1.2 Parser

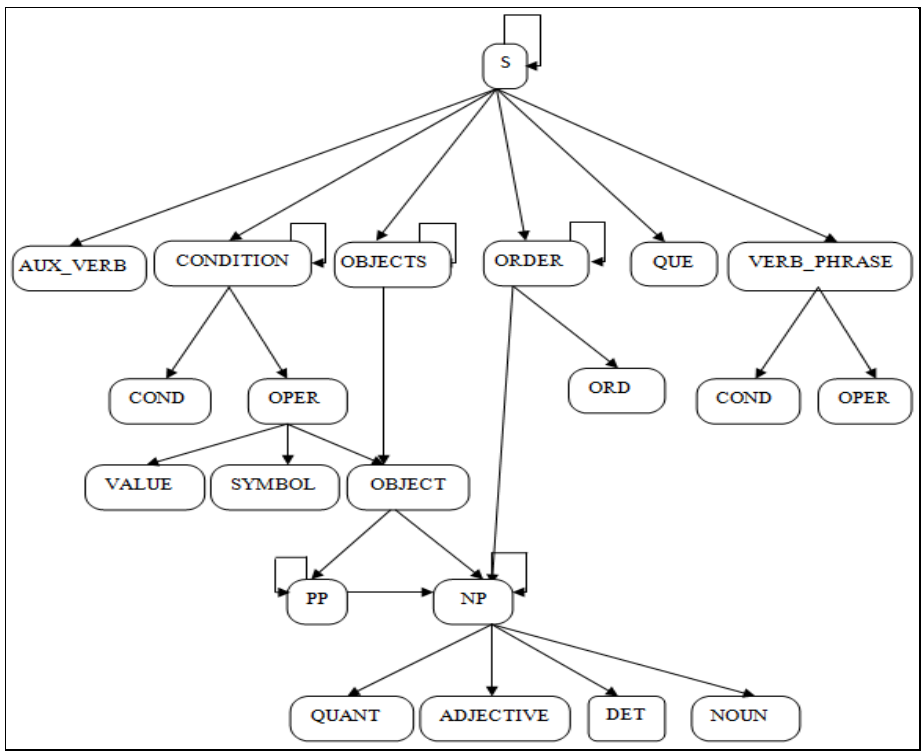

Figure 3. Graphical representation of BNF

The parser or syntactic analysis creates the syntactic structure of the FNLQ, which is a parse tree [22]. It shows how words in the FNLQ relate to each other.

The function of the syntactic analysis is based on a set of syntactic rules having the form S → {E 1 ... En}, in which the sequence of expressions {E 1 ... En} can be replaced (rewritten), when analysis, by a new unique identifier S. These rules describe the possible grammar structures of the FNLQ and constitute a formal grammar named Backus-Naur Form (BNF) defined by:

BNF = (N, T, R, and S) where:

(1) N: a set of non-terminal symbols.

(2) T: is a set of terminal symbols.

(3) R: is a set of context-free productions.

(4) S: is the start symbol used to represent the FNLQ.

The BNF is graphically schematized in Figure 3. The elements of this grammar include:

(1) S: sentence;

(2) NP: noun phrase;

(3) PP: prepositional phrase;

(4) DET: determiner

The implementation of the grammar was done by a Prolog Knowledge Base (KB) as a defined logic program. The defined word means that the KB consists of only defined clauses. A defined clause is a rule that represents each of the rules of the BNF used.

Generally, the BNF used has two types of rules:

• *Domain independent rules*: these rules have non-terminal symbols on the right side. The following rule is an example:

S → OBJECTS [CONDITION] [ORDER] [CON S]

• *Domain-dependent rules*: only have terminal symbols on the right side. The following rule is an example:

NOUN → Client

Using rules that are domain dependent requires these rules to be generated whenever the proposed system interfaces with a new domain. For this purpose, we use an Automatic Producer of Syntax Rules (APSR). The operation of APSR is based on machine learning approach which consists in automatically producing all new rules necessary to parse the FNLQ [23]. It has two roles:

• *Verification*: it verifies if all the syntax rules necessary to analyze the FNLQ exist in the *KB*.

• *Learning*: it detects, creates, and adds missing rules to the *KB*. This process is outlined in the next algorithm.

Algorithm APSR
Input:
FNLQ an French Natural Language Query
Output
A set of syntactic rules SR= {(R_i), $1 \le i \le n$};
Begin
  Split the FNLQ into a set of tokens W = {(t_j: GC_j), $1 \le j \le m$}
  where GC_j is Grammatical Category of the token t_j;
 For each token (t_j: GC_j) $\in$ W loop
Create the syntactic rule R_i correspond to the token t_j ;
  If $R_i \notin KB$
    add R_i to KB;
  End if
 End loop
 Return SR ;
End APSR

The use of the APSR enables the system to improve its *KB* automatically through experience and adapt it with user's requests.

Figure 4 shows the parse tree corresponds to the following FNLQ:

"Affiche moi tous les noms et les adresses et les sexes des clients dont l'âge >= 25"

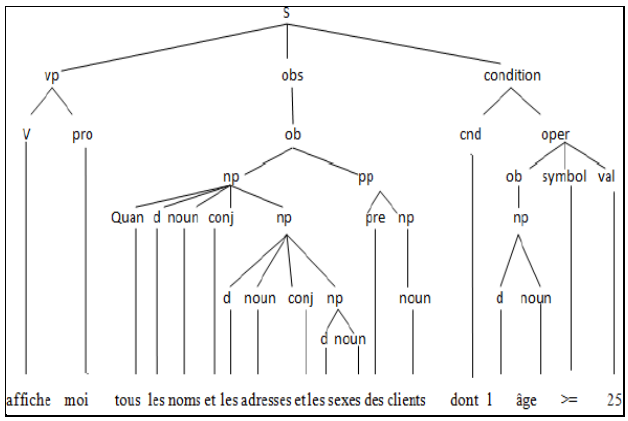

Figure 4. Example of parse tree

3.1.3 Semantic interpreter

The purpose of the semantic interpreter is to assign meaning to the parse tree created by the parser. This is done through the application of a set of semantic rules and is accomplished in two steps. In the first step, the system filters the parse tree to obtain an abstract parse tree. This latter does not represent all the details appearing in the parse tree, but it keeps the necessary parts for the production of the logical query.

Filtering involves removing superfluous nodes that don't add meaning to the parse tree, such as linking elements (such as "et"), separators (such as commas), and parentheses. Figure 5 presents the abstract parse tree produced from the parse tree shown in Figure 5.

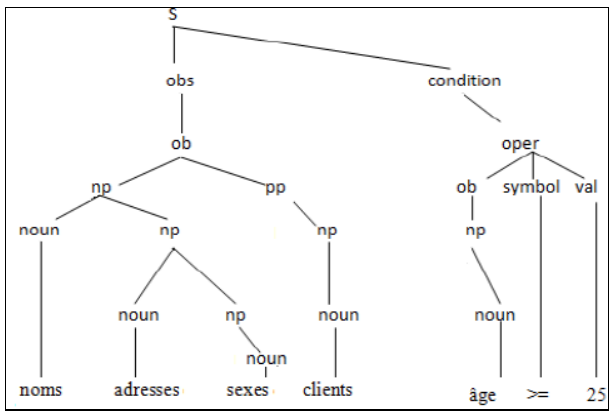

Figure 5. Example of abstract parse tree

After filtering the parse tree and obtaining the abstract parse tree, the semantic interpreter uses other semantic rules to generate the logical interpretation of FNLQ, which we call XLQ (XML Logical Query). We decide to express the logical query in XML for the reason that:

• XML is an inter-tool language that ensures the reuse and verification of models.

• XML does not depend on any language; the logical query can be easily translated to different DBQs (SQL, XPATH,

etc.).

• The logical query in XML displays information about attributes and their values.

In XML, we define the structure of XLQ by the following XML schema (Figure 6):

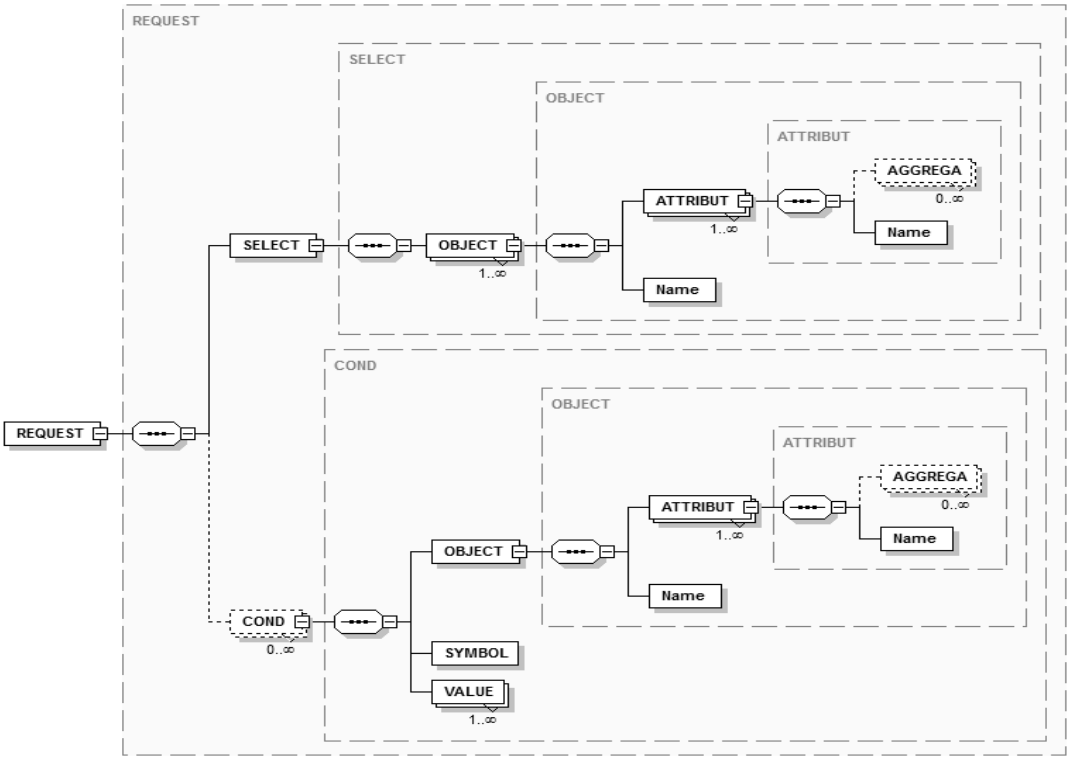

Figure 6. XML schema of XLQ

The following XLQ represents the logical interpretation of the FNLQ:

"Affiche moi tous les noms et les adresses et les sexes des clients dont l'âge >= 25":

```
<QUERY>
  <SELECT>
     <OBJECT>
       <NAME> client </NAME>
       <ATTRIBUT>
          <NAME> noms </NAME>
        </ATTRIBUT>
        <ATTRIBUT>
             <NAME> adresses </NAME>
        </ATTRIBUT>
        <ATTRIBUT>
             <NAME> sexes </NAME>
        </ATTRIBUT>
    </OBJECT>
  </SELECT>
  < CONDITION>
    <OBJECT>
       <ATTRIBUT>
            <NAME> âge </NAME>
       </ATTRIBUT>
    </OBJECT>
     <SYMBOL> >= </SYMBOL>
    <VALUE> 25 </VALUE>
  </ CONDITION >
 </ QUERY>
```

The XLQ is the final result of the linguistic processor. It will be used by the Database Knowledge Generator to generate the DBQ. This operation will be detailed in the following paragraph.

3.2 Database knowledge generator

The Database Knowledge Generator is used to generate the final DBQ. This is done by mapping each part of the XLQ to its corresponding clause in the DBQ.

The process of the DBQ generation has three phases. Each one of them manipulates a particular part of the XLQ. In the first phase, the system deals with the part that corresponds to the names of the attribute in XLQ to build the SELECT clause. In the second phase, it produces the FROM clause by selecting the part of the XLQ that represents to the table name or a group of table names. Finally, it extracts the conditions to build the WHERE clause.

By concatenating the results of the previous phases, the system constructs the DBQ. A test function follows each phase to verify if the name of tables and attributes, extracted from the XLQ, exists in database dictionary. If it is not the case, the system uses a domain specific dictionary called mapping table. This table stores synonyms of names of tables and attributes. The using of mapping table helps the user to put its FNLQ with different ways without the need to know the exact name of tables and attributes. Once the DBQ is generated, the DBQ executor sends it to the Database Management System (DBMS) and displays the returned responses in tabular format for SQL queries and in hierarchical format for XPATH queries.

4. SYSTEM RESULTS

The interface in Figure 7 displays the result of translating of the FNLQ:

"Affiche moi tous les noms et les adresses et les sexes des clients dont l'âge >= 25":

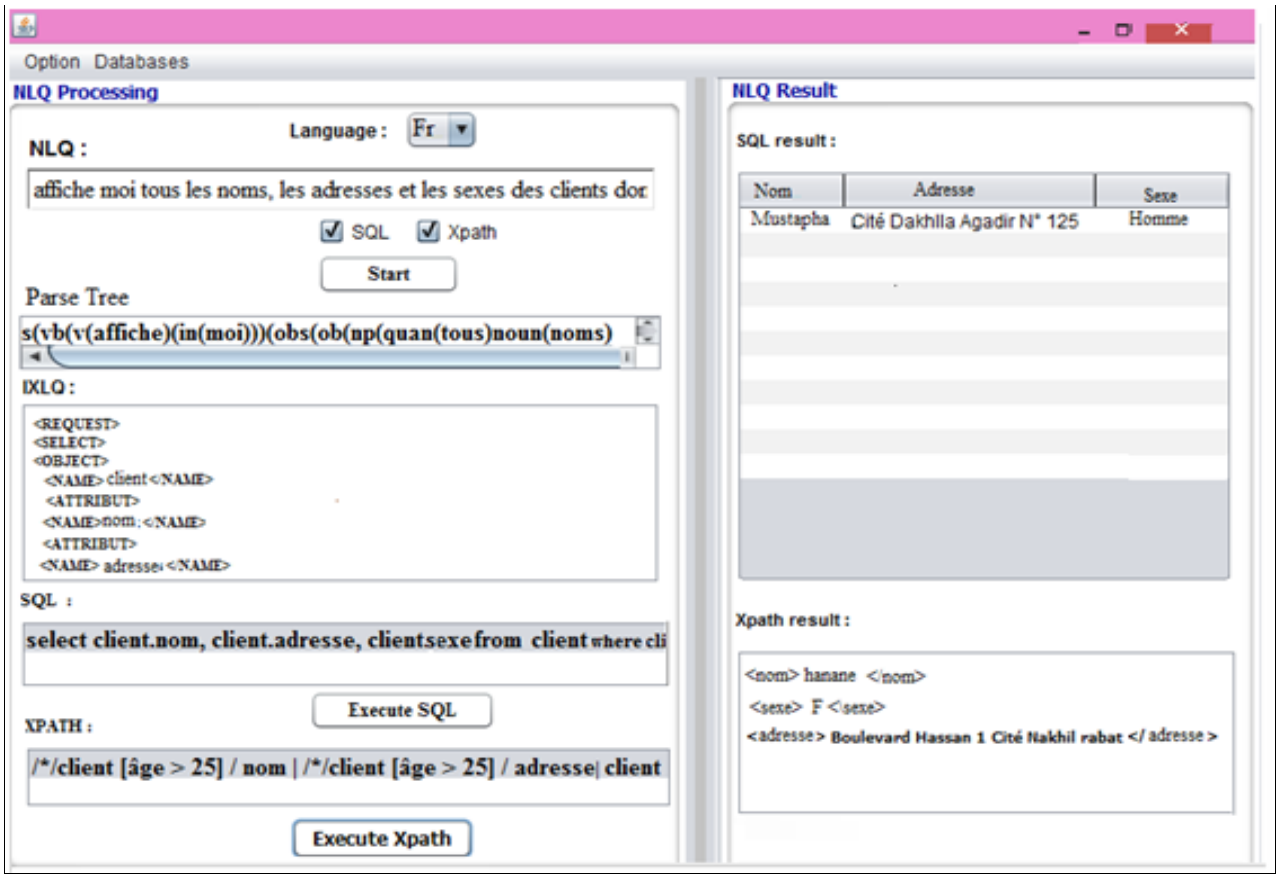

Figure 7. System interface

Table 1. Queries without projection and selection

FNLQ	SQL	XPATH
Affiche les clients Affiche tous les clients Affiche-moi nos clients Affiche-moi tous les clients Affiche-moi les clients Affiche nos clients Affiche-moi tous nos clients Affiche tous nos clients Clients?	SELECT * FROM client	*/client/*
Liste tous nos salariés Liste tous nos employés	SELECT * FROM salarié	/*/ salarié /*
Montre-moi tous nos clients et projets	SELECT * FROM client SELECT * FRO projet	/*/client/* /*/projet/*

In the following tables, we present several types of FNLQs that have been translated and successfully executed by the system. These queries cover different areas of databases. We have classified these requests into four categories.

The first category concerns queries without projection and selection. These queries don't contain any attribute or selection condition specification. Table 1 shows examples of queries in this category.

Table 2 presents examples of the second category of FNLQs. These queries use only projection where we specify certain attributes without any conditions.

The third category of FNLQs deals with requests with projection and selection. In these queries, we specify attributes and conditions. Table 3 shows some examples of these queries.

Table 4 illustrates the last category of FNLQs. These queries contain aggregation functions.

Table 2. FNLQS with projection and without selection

FNLQ	SQL	XPATH
Donne-moi les noms des étudiants	Select étudiant.nom from étudiant	/*/ étudiants /nom
Montre-moi tous les noms, âges et adresses des employées	Select salarié.nom, salarié.âge, salarié.adresse from salarié	/*/ employées /nom |/*/ employées /âge |/*/ employées /adresse
Quels sont tous les noms et les adresses des clients	Select client.nom, client.adresse from client	/*/client/nom |/*/client/adresse
Cherche tous les noms des employés et des clients	Select client.nom from client Select salarié.nom from salarié	/*/client/nom /*/employé/nom
Trouve les noms des clients et les montants des factures	Select client.nom from client Select facture.montant from facture	/*/client/nom /*/facture/montant

Table 3. FNLQS with projection and selection

FNLQ	SQL	XPATH
Montre tous les étudiants dont le nom est "hanane"	Select * from étudiant where étudiant.nom = 'hanane'	/*/ étudiant [nom = “hanane”] /*
Tous les clients dont le nom est "hanane" ou "Fatima"	Select* client from client where client.nom in(‘mustapha’, ‘hanane’)	/*/client [nom = “hanane”]/*| /*/client [nom=”Mustapha”]/*
Montre tous les enseignants avec l’âge est entre 28 et 40	Select * from enseignant where enseignant.âge between 28 and 40	/*/ enseignant [âge > 28] [âge<40]/*
Quels sont les noms des employées dont l’adresse est "Agadir dakhla"?	Select salarié.nom, salarié.âge from salarié where salarié.adresse =‘Hay dakhla Agadir’	/*/employée [adresse = "tan tan hay ljadid"] /nom|/*/employée [adresse = "tan tan hay ljadid"] /âge
Cherche les adresses des clients avec âge est supérieure ou égale à 26 et le nom est "hanane"	Select client.adresse from client where client.âge <= 26 and client.nom = 'hanane'	/*/client [âge <= 26] [nom = "hanane"] /adresse

Table 4. FNLQS with aggregate function

FNLQ	SQL	XPATH
Donne-moi le nombre des fournisseurs dont le nom est "Hanane"	Select count (*) as NB_fournisseur from fournisseur where fournisseur.nom =‘hanane’	/*/count (fournisseur [nom = "hanane"])
Montre-moi la moyenne des âges des clients	Select avg (client.âge) as avg_client_age from client	Avg (/*/client/âge)
Donne-moi le minimum âge des étudiants	Select min (étudiant.âge) as min_ étudiant _age from étudiant	Min (/*/étudiant /âge)
Affiche le maximum âge des clients dont l’âge est inférieur ou égale à 40	Select max (client.âge)as min_client_age from client where client.âge < 40	Max (/*/client [âge < 1000] /âge)
Quels sont les employés avec le maximum âge?	Select * from employée where employée.âge in (select max (employée.âge) from employée	/*/ employée [âge=max (/*/employée /âge)]/*

In order to evaluate the performance of the proposed system and make the necessary corrections to improve it, we have carried out experiments.

The first experiment is to check the number of FNLQs for which the system generates a response. To do this, we tested 1300 FNLQs. The results obtained by this experiment are presented in Figure 8 below.

According to Figure 8, from 1300 FNLQs the system generated SQL queries for 1223 FNLQs, a percentage of 94.07%, and XPath queries for 1205 FNLQs, a percentage of 92.69%. However, some of these responses are not correctly generated, implying that a query is correctly generated if it is syntactically correct.

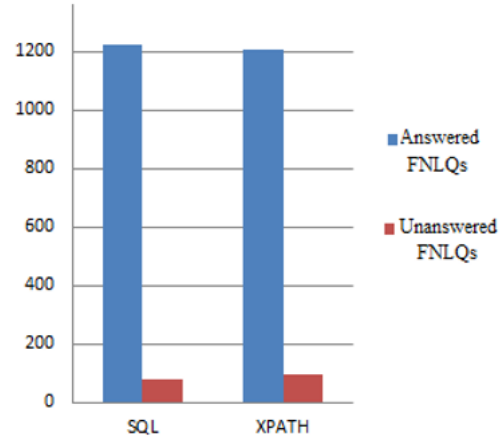

Figure 8. Answered\Unanswered FNLQs

In order to evaluate the number of correctly generated queries, we performed a second experiment. This latter is based on the different queries for which we have answers in the first experiment. The different results obtained are presented in Figure 9 below.

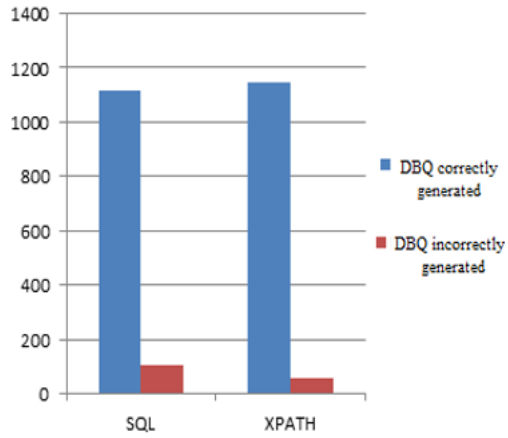

Figure 9. DBQ correctly / incorrectly generated

As shown in Figure 9, 1223, 1118 of the FNLQs are correctly converted to SQL queries, and 1147 of these queries are correctly translated to XPATH queries.

To locate errors responsible for unanswered queries and those that are incorrectly generated, we examined the outputs of these queries and constructed Figure 10.

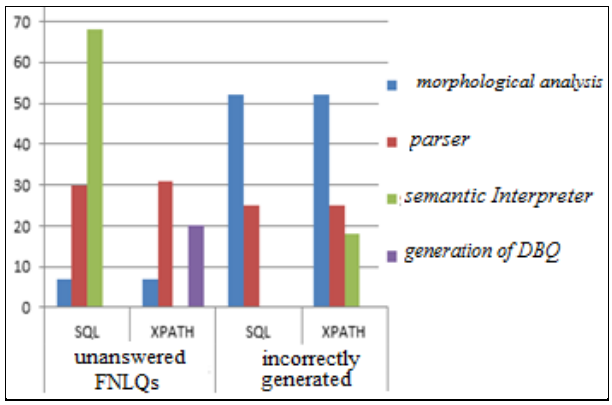

Figure 10. The result of errors explanation

From Figure 10 the errors found are committed at the level of the FNLQs analysis and the generation of the DBQ request. We classified them in four categories:

• Errors due to morphological analysis: From figure 10, we see that the morphological analysis is the cause of 6.66% of FNLQs that don't have SQL query and 12.06% of SQL queries generated incorrectly. This is also the cause of 6.64% of XPATH queries that are incorrectly generated. One of the main sources of these errors is that in some FNLQs the tagger function does not affect some tokens their correct grammatical function.

• Errors due to Parser: This review also shows that the Parser is the cause of 32.46% of the FNLQs that don't have SQL queries and 28.57% of the SQL queries generated incorrectly. In addition, this parser is responsible for 53.44% of incorrectly generated XPATH requests and 26.31% of FNLQs that don't have XPATH requests. Generally, one of the sources of these errors is that the parser generates parser trees that don't reflect the interactions between the different words constituting the natural language query.

• Errors due to semantic Interpreter: The errors produced by the semantic Interpreter are 64.76% of the incorrectly generated SQL queries and 19.94% of the FNLQs that don't have an XPATH query. The source of these errors is that semantic Interpreter does not correctly convert the parser tree to the exact XLQ.

• Errors due to the generation of DBQ: The errors due to the generation of DBQ are 34.48% of FNLQs that don't have XPATH requests.

The evaluation of some DBQ that are correctly generated shows us that these queries don't always correspond to the FNLQ. The object of the last experiment is to show the number of DBQ that match and don't match the FNLQs. This is the subject of the following Figure 11.

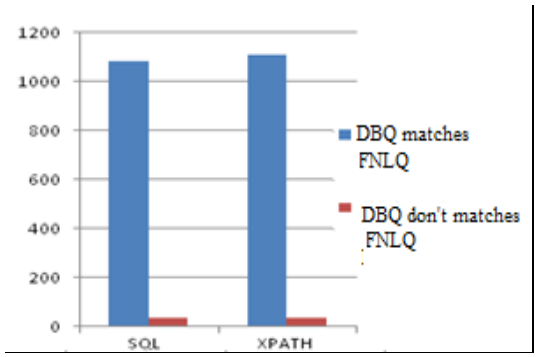

Figure 11. DBQ matches\ don't matches FNLQ

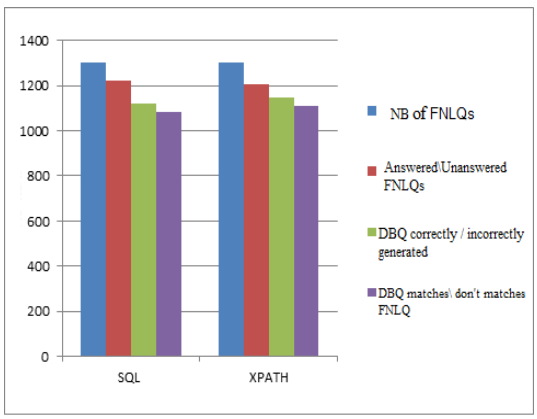

Figure 12. Summary of the results obtained

The graph in Figure 12 shows a summary of the results obtained.

5. CONCLUSION

The ultimate objective of this study is to propose a generic model of Natural Language Interfaces for querying databases using the French language. The function of this interface is based essentially on linguistic operations. Applying these operations help the proposed interface to extract the linguistic information needed to translate the FNLQ to a DBQ.

The main advantages of this interface are that it operates independently of database model and domain and that it is able to automatically extend its Knowledge Base through experience.

As future works, we will continue to test the capacity of our system with other database models: relational object, object oriented, NoSQL. Also, we attend to increase the level of human-computer interaction by allowing our system to access a database via voice requests; Furthermore, we want to use agents to improve the system knowledge base.

REFERENCES

[1] Agrawal, A.J., Kakde, O.G. (2013). Semantic analysis of natural language queries using domain ontology for information access from database. International Journal of Intelligent Systems and Applications, 5(12): 81-90. http://dx.doi.org/10.5815/ijisa.2013.12.07

[2] Shah, A., Pareek, J., Patel, H., Panchal, N. (2013). NLKBIDB-natural language and keyword based interface to database. In 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Mysore, India. http://dx.doi.org/10.1109/ICACCI.2013.6637414

[3] Auxerre, P. (1986). MASQUE modular answering system for queries in English-programmer's manual. Technical Report AIAI/SR/11, Artificial Intelligence Applications Institute, University of Edinburgh.

[4] Moore, R.C. (1981). Problems in logical form. Sri International Menlo Park Ca Artificial Intelligence Center, 19th Annual Meeting of the Association for Computational Linguistics, Stanford, California, USA, pp. 117-124. http://dx.doi.org/10.3115/981923.981957

[5] Mitrovic, A., Ohlsson, S. (2016). Implementing CBM: SQL-tutor after fifteen years. International Journal of Artificial Intelligence in Education, 26(1): 150-159. http://dx.doi.org/10.1007/s40593-015-0049-9

[6] Mitrovic, A., Martin, B. (2000). Evaluating the effectiveness of feedback in SQL-tutor. In Proceedings International Workshop on Advanced Learning Technologies. IWALT 2000. Advanced Learning Technology: Design and Development Issues, Palmerston North, New Zealand, New Zealand, pp. 143-144. http://dx.doi.org/10.1109/IWALT.2000.890591

[7] Knowles, S. (1999). A natural language database interface for SQL-tutor.

[8] Li, Y.Y., Yang, H.H., Jagadish, H.V. (2005). NaLIX: An interactive natural language interface for querying XML. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 900-902. http://dx.doi.org/10.1145/1066157.1066281

[9] Range, R.A.P., Gelbukh, A., Barbosa, J.J.G., Ruiz, E.A., Mejía, A.M., Sánchez, A.P.D. (2002). Spanish natural language interface for a relational database querying system. In International Conference on Text, Speech and Dialogue, 2448: 123-130. https://doi.org/10.1007/3-540-46154-X_16

[10] Ahmad, R., Khan, M.A., Ali, R. (2009). Efficient transformation of a natural language query to SQL for Urdu. In Proceedings of the Conference on Language & Technology, pp. 53-60.

[11] Li, Z.J., Li, J.K., Ning, W.X. (2015). Research on Chinese natural language query interface to database based on syntax and semantic. In Applied Mechanics and Materials, 731: 237-241. http://dx.doi.org/10.4028/www.scientific.net/AMM.731.237

[12] Meng, X.F., Zhou, Y., Wang, S. (1999). Domain knowledge extracting in a Chinese natural language interface to databases: NChiql. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, 1574: 179-183. https://doi.org/10.1007/3-540-48912-6_25

[13] Kataria, A., Nath, R. (2015). Natural language interface for databases in Hindi based on karaka theory. International Journal of Computer Applications, 122(7): 39-43. https://doi.org/10.5120/21716-4841

[14] Nanda, G., Dua, M., Singla, K. (2016). A Hindi question answering system using machine learning approach. In 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), New Delhi, India, pp. 311-314. http://dx.doi.org/10.1109/ICCTICT.2016.7514599

[15] Bourigault, D., Aussenac-Gilles, N., Charlet, J. (2004). Construction de ressources terminologiques ou ontologiques à partir de textes Un cadre unificateur pour trois études de cas. Revue d'Intelligence Artificielle, 18(1): 87-110.

[16] Haddad, H. (2003). French noun phrase indexing and mining for an information retrieval system. In International Symposium on String Processing and Information Retrieval, 2857: 277-286. http://dx.doi.org/10.1007/978-3-540-39984-1_21

[17] Deléger, L., Grouin, C., Zweigenbaum, P. (2010). Extracting medication information from French clinical texts. In MedInfo, 160(Pt 2): 949-953.

[18] Reis, P., Matias, J., Mamede, N. (1997). Edite-A Natural Language Interface to Databases A new dimension for an old approach. In Information and Communication Technologies in Tourism, pp. 317-326. http://dx.doi.org/10.1007/978-3-7091-6848-6_33

[19] Hemerelain, B., Belbachir, H. (2010). Semantic analysis of natural language queries for an object oriented database. Journal of Software Engineering and Applications, 3(11): 1047-1053. http://dx.doi.org/10.4236/jsea.2010.311123

[20] Bais, H., Machkour, M., Koutti, L. (2016). A model of a generic natural language interface for querying database. International Journal of Intelligent Systems and Applications, 8(2): 35-44. http://dx.doi.org/10.5815/ijisa.2016.02.05

[21] Tari, L., Tu, P.H., Hakenberg, J., Chen, Y., Son, T.C., Gonzalez, G., Baral, C. (2010). Parse tree database for information extraction. IEEE Transactions on Knowledge & Data, Engineering.

[22] Essalmi, F., Ayed, L.J.B. (2006). Graphical UML view from extended Backus-Naur form grammars. In Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06), Kerkrade, Netherlands, pp. 544-546. http://dx.doi.org/10.1109/ICALT.2006.1652498

[23] Bais, H., Machkour, M., Koutti, L. (2016). Querying database using a universal natural language interface based on machine learning. In 2016 International Conference on Information Technology for Organizations Development (IT4OD), Fez, Morocco, pp. 1-6. http://dx.doi.org/10.1109/IT4OD.2016.7479304