Natural Language Query Parser using First Order Logic for Querying Relational Databases

B. Sujatha
Assistant Professor
Department of CSE
Osmania University
Hyderabad, India

S. Viswanadha Raju
Professor
Department of CSE
JNTUH University
Jagtial, India

ABSTRACT
Relational database management systems are mostly used for effective representation and retrieval of data. For the user, it is hard to learn the database interface language to deal with various operations on databases. Hence there is a need to construct a bridge between natural language query and database understandable query which is a major challenge. In this paper, we have proposed a Natural Language Parser for Natural Language Interface to customer database. The parser converts the Natural Language query into First order Logic and then the First order logic query is converted into structured query. This paper also addresses the word sense disambiguation problem using ontologies and n-grams. The lexical meaning of the natural language query can be captured with n contiguous characters or words of the query. The proposed system is able to handle extraction, insertion, deletion and updation queries. It is also able to process join, conditional, single and multiple column retrieval queries. The performance of the system is measured using precision, recall and F-measure. The results are progressive.

Keywords
Natural Language Query, First Order Logic, Structured Query, Precision, Recall, F1-measure

1. INTRODUCTION
Database systems are used since 1970s for the storing various kinds of data for different purposes such as commercial and personal needs. Though there are many types of architectures for database design like object oriented, object based, file based, hierarchical based and network based, the predominant designing of databases follow relational database architecture to store the data by using various types of storage devices. In relational databases, the data is stored using tables. The table contains set of rows and columns. Each column represent and attribute and each represents the instance of the data for a set of attributes. The data can be manipulated using various operators with fixed set of keywords by following a set syntax rules. By learning this structured query language one can extract the required data from the whole set of data, can also perform various operations such as update, manipulate and deletion of the data.

The Relational database management systems are more popular based on the characteristics like its robustness and flexibility, high performance, scalability, data security and protection and flexible data maintenance. Above all these advantages, it allows to index, perform aggregation, filtering and sorting can be done on the data using structured query language.

There are some disadvantages with relational databases. To perform operations on the data which is stored on databases, it is required to learn the structured query language. Hence , the naive user who knows only the natural language can not directly access the required information from the databases. To come out from these limitations, it is required to design a tool which can understand the requirements of the naive user through natural language query, convert the natural language query into an equivalent structured language query. Then the obtained structural query is used to access the required information from the databases. This kind of tool ins termed as Natural Language Interface to Databases or NLIDB system. Thus, the NLIDB system take the input as natural language query and converts it into a structures language query and returns the desired information to the naive user.

The designing of a NLIDB system for various languages and for different underlying databases is attempted by various researchers since five decades. But, designing of an most suitable NLIDB systems with high accuracy, precision and recall is still an open research problem which need to be addressed. The various earlier developed NLIDB systems focused on particular databases. There is need of designing a generic NLIDB system which can address the robustness and scalability of the applications. It is required to attempt the problem of portability to customize a NLIDB system to a other language and to other set of datasets designed for various domains. The efficiency of conventional NLIDB systems depend mostly on domain experts capabilities and linguistic features of the natural language.

In this paper, it is focused on designing a NLIDB system to overcome the various issues such as portability to different languages and to access the required information independent of the underlying database. It also required maintains the scalability and robustness of the system. To achieve this objective, the system is designed with general purpose syntactic parser. In this paper, it is proposed a system in which the natural language query is parsed using First Order Logic and the parsed query is converted into SQL query. The designed system maintains a high accuracy 84% for customer database.

2. RELATED WORK
There are many designing models are proposed in the literatures in the field of NLIDB such as pattern matching systems, syntax based systems, semantic based grammar systems and intermediate representation of languages system.

The pattern matching systems takes input as a set of rules and sample set of pattens. Based on the inputted word of sentence with natural language, it will be compared with the predefined patterns [1]. If there is a match between the input and predefined pattern then an action will be generated and these generated actions will be stored in the database. The response given to the user is based on the action generated. This kind of systems are limited to specific databases. The accuracy of the system is depend on the complexity of the patterns used to

train and based on the set of rules used to train the system [2]. The NLIDB system SANVY is a good example for pattern-matching systems [3].

The syntax based systems takes the user query as input and parse the given input syntactically. The parse tree generated for the input query is overlapped with the one structured query of the database expressed using structured query language. LUNAR is a best example for syntax based NLIDB systems [4]. In these systems, the grammar rules are derived to match the various user questions with syntactic structures [5]. This system is used to answers the questions on rocks which were collected from the moon. With the corrections in the dictionary errors, the performance of the system has increased [8].

In the semantic grammar system, the parse is simplified by eliminating unimportant nodes or by combining two or more nodes into one node. The complexity of structured query can be reduced in semantic grammar system. Semantic grammar systems are more simpler when compared with syntax based systems. But these systems need to be trained with a prior knowledge of the various elements of a domain. PLANES and LADDER are the good examples for Semantic grammars systems [6,7].

In many NLIDB systems, the natural language query is transformed into an intermediate logical query. The logical query is represented using a meaningful representative language such as first logic language or Boyce codd normal form. This kind of representative languages, represents the meaning of the users queries in high order level of concepts. These concepts are independent from the structure of the database. This representative query is then transformed into an expression in the structured query language which can extract the relevant data from the databases.

In the intermediate representation of natural language systems, the natural language query is inputted to the system. This query is processed for syntax rules using a parser. Based on the set of syntax rules of a natural language, it generates a parse tree. By using the semantic rules of semantic interpreter module, the generated parse tree is translated into an intermediate logic query. In the semantics rule, left hand side of the syntax rule contains the logic expression of the constituent where as right-hand side of the syntax rule is a function of the logic expressions of the constituents. The logic expressions represents the words which are corresponds to lexicon. To get the required information from the database, the logic query is to be transformed into a structured query which is supported by the underlying Database Management System. MASQUE/SQL is an example of intermediate representation language systems [7].

By using semantic grammar techniques which interleaves semantic and syntactic processing in distributed databases, LADDER system is used to parse natural language questions to database understandable queries [7]. The another NLIDB system implemented using the language called Prolog was CHAT-80. This system transforms the natural language inputted English queries into Prolog expressions. These Prolog expressions are evaluated using the Prolog database. ROBOT which was a prototype of a NLIDB system named INTELLECT which was a commercial natural language interface to database systems [9]. ASK is the another NLIDB system which allows the users to train the system with new words and concepts while inter actioning with the system. By using the system, it is possible to make interactions with various external sources such as external databases, chating, Facebook, twitter, email programs and many other applications.

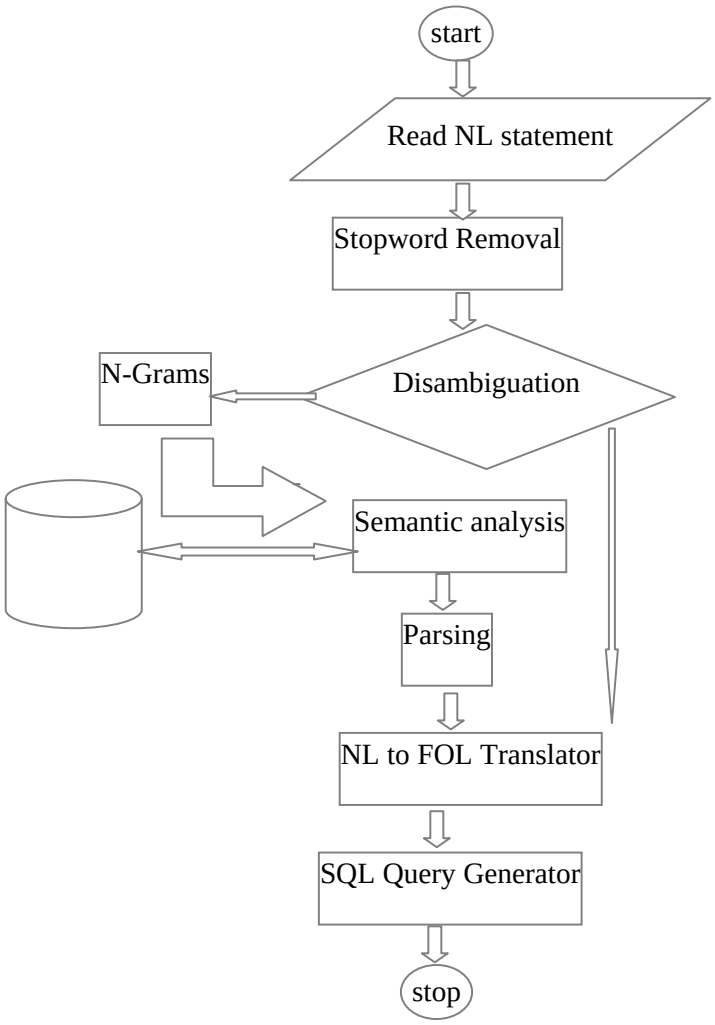
Generic Interactive Natural Language Interface to Databases (GINLIDB) was designed by the using UML and developed using Visual Basic.NET. The system was a generic system and it works for underlying suitable database and knowledge base [10]. SynTactic Analysis using Reversible Transformations (START) is also another Natural Language System. It was the first Web-based question answering system. It was available online and continuously operating till now [11]. It utilizes various language Dependant functions such as parsing, semantic analysis, word sense dis-ambiguous, natural language annotation for appropriate information segmentation and presentation for the user [12].

JUPITER was a NLIDB system to know the weather information worldwide. The user can pose a question to the system in their native language to forecast the weather information over the telephone. The Oracle Structured Query Language SQL can be learned by the students using the NLIDB system called SQL-Tutor. If the student asked the new questions by typing at terminal then also, the SQL-Tutor can answer the question by using the existing knowledge [13]. KUQA system divides the query based on possible answer and after that it uses NLP techniques and also WorldNet to identify the answers which suitable to its corresponding category. But, this system can not handle any linguistic information [11]. QuALiM another NLIDB system designed based on complex syntactic structure which were based on certain syntactic description question patterns [11].

3. NATURAL LANGUAGE QUERY TRANSLATION

The database query specified by the user is gone through several stages of processing to ultimately get convert into SQL query. The obtained SQL query is forwarded to the Database Management Systems for the query execution and the results are presented to the user. All of the conversion process is abstract to the user whose only job is to specify the natural language query to the system. The system then responds to the user with the acquired results.

The flowchart for Natural Language Query Translation is as follows:



The natural language query goes through the following phases:

A. Stop words Removal and Stemming

B. POS tagging and Resolving Disambiguation

C. Parsing

D. Converti Natural Language query to First Order Logic

E. FOL to SQL transformation.

This section covers the first three phases such as stopword removal and stemming, POS tagging and resolving dis-ambiguous and Parsing the natural language query. The remaining two phases are covered in sections 4 and 5.

3.1. Stopword removal and Stemming

A stop word is a word which is of little value in a user's request statement. Stop words when ignored does not affect the search and also saves time in unnecessary processing. The words from the user defined request statement are discarded if they belong to stop list. Stop words removal saves both time and space at the preprocessing stage.

Stemming is one of the important features of information retrieval. The main idea is to remove any additionally attached suffixes or prefixes to a word. Words have different morphological variants, reducing the variants to its root form is done through stemming. The stemming operation is optionally applied as per need to target the root word. Stemming can be achieved using any of the stemming algorithms such as dictionary look up stemmers, porter stemming algorithm.

3.2. POS tagging and Resolving Disambiguation

The tagging is the process of automatic allotment of descriptors. A descriptor is also knows as a tag. A tag may be used to identify part of speech, semantic information etc. Thus tagging can also be said as a form of classification of information.

Part of Speech tagging can be defined as the task of assigning parts of speech to the words in a given text. It is simply knows as POS Tagging. Parts of speech are nouns, pronouns, verbs, adjectives, adverbs, preposition, conjunction, and interjection. Interjections are rarely found in the database query and if occurred, it is ignored since it is a member of stop list. A POS tagger is a program that does its job using information from the dictionary, lexicon or rules.

After POS tagging, the system is left with the information of the words that may relate either to the CPVbase tables or attributes or fields. Any other word that does not relate to any of it may be resolved by mapping with the ontology for

CPVbase. When disambiguation arises then the preceding or succeeding words are analyzed with the candidate word or the other aspects such as linguistic features of the word are examined.

The words are divided using N-grams and when complexity to interpret increases the level of N-grams is increased and the words in the group are analyzed. The synonymous words may also occur that can be replaced with the system's terminology by finding its correct mapping using the ontology.

3.3. Parsing

The query containing only the keywords meant to be processed is parsed. The grammar rules that are applied for parsing are explained below. The main aim is to convert the user's database query into First Order Logic. To achieve this parsing need to be done using few of the grammar rules. The First order Logic statement can be then converted to SQL Query. The following grammar defines when the predicate is a mathematical operation:

If the phrase is of the form : <p1> M <p2>

where <p1>, <p2> denotes preceding and succeeding predicates respectively. M indicates a scientific measure.

4. PARSER USING FIRST ORDER LOGIC FOR CUSTOMER DATABASE

The conversion process of Natural language to First Order Logic is carried out using the formula

[<S1>] [→] [<QEXP>]<S2>

S1 indicates the natural language statement and S2 denotes the First order Logic statement containing Quantifiers given by QEXP. The problem in converting the natural language statement to FOL rises at the NL side, since it does not follow any clear semantic. The conversion is an ad-hoc process. The main format of the conversion algorithm can be simply stated as it follows the syntax of FOL and incorporates the natural language statement such that the intention of the statement is retained.

The FOL formulas rely on the following set:

1) *Variables:* The semantics of an FOL depends on a Domain D. The set of variables represents elements of D.The set of variables can be denoted symbols such as {x, y, z,..., x_1, x_2,…}.

Mathematically it can be stated as, If there is an Assignment function A, then A: V→D. That is, the assignment functions maps every variable to an element of the domain D.

2) *Quantifiers:* The most quantifiers used are '∀' read as "for all" and '∃' called as "there exists". The "for all" is a universal quantifier and "there exists" is an Existential quantifier. The quantifiers are used to quantify what assignments can be used for the variables.

3) *Predicate Symbols:* The predicates represent the characteristics of the elements or relation among element of D.

4) *Function Symbols:* They denote function in the Domain D, denoted as f, g, h, or can be as plus, multiply etc.

5) *Constant Symbols*: They represent the specific element from D.

6) *Connectives:* The connectives used to connect the atomic formulas constructed using above element. The commonly used connectives are ∧ (AND), ∨ (OR), ⇒ (IMPLIES), ~ (negation: NOT).

The functions and predicates can define their ARITY that specifies how many arguments it can take. Arity is specified below a predicate or a function. For example a function defined as fn indicates that the function has arity of 'n', meaning that it can have n arguments.

According to FOL a term is defined as a

(I) If T is a family of terms such as {t1, t_2,…..t_n }, then every Variable is a term, represented as $x \in V \rightarrow x \in T$

x is any entity and V is a set of variables.

(ii) Every Constant is a term, such that $c \in C \rightarrow c \in T$

c is any element and C is the set of Constants.

(iii) For every 'f' of arity 'n', that is $f_n \in F$, where F is a set of function symbols and the set {t1, t_2,…..t_n }$\in$ T then the following expression also holds true: f_n(t1, t_2,…..t_n) $\in$ T

The mapping of predicates, constants and function sets is done by a function called as Interpretation function (I). The interpretation specifies what does each of the constituent stands for in the domain. The mapping by the interpretation function is done as follows,

$p \in P \rightarrow p^I$

$f \in F \rightarrow f^I$

$c \in C \rightarrow c^I \in D$

where p, f, c are elements from the sets of Predicate (P), functions (F) and Constants (C). All of the above specifications define the vocabulary of the first order logic system.

FOL formulas can de denoted as P(t1, t2,…..tn) , also called as atomic formulas , (M∨N), M⇒N, ~M , where M and N are FOL formulas constructed using functions, predicates or constants. The following specifies the algorithm to convert the NL Query into FOL:

Step 1: Identify the nouns, verbs, adjectives from the given NL query. And denote the predicates or functions symbols.

Step 2: Indicate the arity of each of the symbols.

Step 3: Specify the quantifiers of the variables.

Step 4: Build the Atomic formulas for the predicates.

Step 5: Classify the atoms that belong to the same group that will be joined using Connectives.

Step 6: Join the atoms of each group using the logical connectives.

Step 7: State the left and right hand side of the implications.

Step 8: Designate the connectives between the formulas of the group and spot the next level of formulas if any and go to step 7.

Step 9: Assign the quantifiers at the right places in the obtained formula to generate the FOL formula.

The FOL formulas obtained thus can be further forwarded to the process of converting it to the SQL query that is examined in the next section.

5. FOL TO SQL TRANSFORMATION

The NLIDB system constructed intends to solve natural language queries based on the select statement of the SQL language. If the syntax of the SQL's Select statement is observed then it requires column list to be displayed from the specific table list based on some given condition. Hence the elements that are needed to form a SQL select statement is:

(1) Column List

(2) Table List

(3) Condition

The obtained FOL formula specifies all the above elements in its Statement. Thus the SQL query can be formulated using the information interpreted using the FOL statement. All the theoretical concepts shown are applied on few natural language queries taken into consideration as sample queries and the working of the above algorithms is illustrated below. The NL statements are randomly selected querying the customer database. The flowchart constructed presented above show the complete course of Natural language query translation to SQL query formulation.

6. EXPERIMENTAL RESULTS AND DISCUSSIONS

6.1. Database Description

The database consists of customer, product, vendor, invoice table for illustration. All of the tables are related to each other, pertaining to a relational database model. So, CPVbase can be defined as:

CPVbase = (Customer, Invoice, Product, Vendor). Customer table can be defined as

$C= \{\forall n/ \exists c_k, w(n), x(n), y(n),z(n),\}$

where n represents a customer entity and ck is unique to every, primary key of customer table and w, x, y and z are the functional variables of n, representing information related to n, as name, area code, phone and balance respectively. A product table is given as

$P= \{\forall m/ \exists p_k, p_i(m), \text{where } 1\leq i \leq s ,v_k \}$

'm' is a product entity . pk is the primary key and pi(n) denotes product's free variables, s represents the field size of the product. vk illustrates foreign key from Vendor table. Vendor table can be defined as below

$V=\{\forall d/ \exists v_k, v_i(d), \text{where } 1\leq i \leq s \}$

'd' is a vendor entity. v_k is the primary key and $v_i(n)$ denotes vendor table's free variables, s represents the field size of the vendor. An entry into invoice table represents that a purchase transaction is at execution and is given by:

$I=\{\forall e/ \exists I_k, I_i(e), \text{where } 1\leq i \leq s , p_k , c_k \}$

'e' is an invoice entity . I_k is the primary key and $I_i(n)$ denotes Invoice table's free variables, s represents the field size of the Invoice. p_k , c_k are the foreign key references from the Product and the Customer table.

6.2. Evaluation measures

The attainment of relevant information by the user as per the natural language query in English gives the retrieval efficacy. The precision is the measure of retrieved results that are relevant to the need, evaluated using the fraction of relevant documents retrieved to the total number of documents retrieved. Mathematically it can be expressed as

$$Precision = \frac{CorrectlyAnsweredQueries}{AnsweredQueries} \quad (5.1)$$

Recall measures the relevant results retrieved as per the user statement. That is the fraction of relevant documents retrieved to the total number of relevant documents present in the system.

$$Recall = \frac{CorrectlyAnsweredQueries}{TotalNumberofQueries} \quad (5.2)$$

based on the precision and recall measurements, the system was tested for a random of 100 queries, giving the result tabulated displayed in table 5.1

Table 5.1 Implementation Results

Total Queries : 100
Answered Queries: 97
Unanswered Queries: 3
Correct Results: 84
Wrong Results: 13
Precision: 84/97 = 86.5%=0.86
Recall: 84/100 = 84.0%=0.84

The results shows that the system offers a recall rate of 0.84 which means that it has 84% Probability of generating correct responses to the user queries. This proves the effective and optimal working of the system. The result is determined by taking portion of queries and is obtained as presented in the table 5.2.

The table 5.2 contains system data generated by testing using different amount of queries and obtained the counts of correctly answered queries (correct_q), wrongly answered queries(wrong_g) and unanswered queries due to improper mapping or no corresponding record (unans_q) with their respective measures of precision and recall. The precision and recall is decreasing if irrelevant queries are observed. Thus the precision and recall can be well defined if the data search is acquired with maximum relevant terms in the query.

Table 5.2: Precision and Recall for varying number of Queries

No. of queries	Correct_q	Wrong_ q	Unans_q	Precision	Recall
20	18	2	0	0.9	0.9
40	35	3	2	0.92	0.875
60	52	6	2	0.89	0.866
80	70	9	2	0.897	0.875
100	84	13	3	0.86	0.84

7. CONCLUSIONS AND FUTURE SCOPE

The research aimed at developing an interface that eases the work of the naive user to formulate a database request and generate appropriate responses. The system vitally uses the ontology constructs, Parsing rules and FOL logic to extract the requisite information in forming a standard database Query. The system is flexible and can be adapted to any of the Database management systems or a relational database management system. EFFCN is a domain independent and highly portable system. It uses the semantics and syntactic knowledge to generate the correct match of the input statement's SQL query. Using the power of ontology and enhanced parsing mechanisms to filter query up to a refined level where it incorporates needed information as per the user. Compared to which the EFFCN system gives a success rate of 84% and high precision of 86.5%.

The NLIDB system future growth is directed towards improving the success rate by applying concepts of neural networks, machine learning parsing techniques and the use of SQL standard aggregate functions such as average, min and max along with the operator precedence concepts. The analysis of the system from the perspective of abbreviations and the temporal queries also needs careful interpretation along with the complex restrictions of FOL logic.

8. REFERENCES

[1] Mrs. Neelu Nihalani, Dr. Sanjay Silakari and Dr. Mahesh Motwani, "Natural Language Interface for Database: A Brief Review", IJCSI International Journal of Computer Science Issues, vol. 8, no. 2, pp. 600-608, Mar. 2011.

[2] T. Johnson, "Natural Language Computing-The Commercial Applications", The Knowledge Engineering Review, vol. 1, no. 3, pp. 11-23, 1984.

[3] Androutsopoulos, G.D. Ritchie and P. Thanisch, "Natural Language Interface to Databases-An Introduction", Department of Computer Science, University of Edinburgh, King‟s Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland, U.K. , Mar. 1995.

[4] W.A. Woods, R.M. Kaplan and B.N. Webber, "The Lunar Sciences Natural Language Information System: Final Report", BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.

[5] C.R. Perrault and B.J. Grosz, "Natural Language Interfaces", Exploring Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Mateo, California, 1988, pp. 133-172.

[6] D.L. Waltz, "An English Language Question Answering System for a Large Relational Database", Communications of the ACM, pp. 526-539, 1978.

[7] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data", ACM Transactions on Database Systems, pp. 105-147, 1978.

[8] W. Woods, "An experimental parsing system for transition network grammars in Natural Language Processing", Algorithmic Press, New York, USA, 1973.

[9] L.R.Harris,"Experience with INTELLECT: Artificial Intelligence Technology Transfer", The AI Magazine, pp. 43-50, 1984.

[10] Faraj A. El-Mouadib, Zakaria S. Zubi, Ahmed A. Almagrous and Irdess S. El-Feghi, "Generic Interactive Natural Language Interface to Databases (GINLIDB)", International Journal of Computers, vol. 3, no. 3, 2009.

[11] "START Natural Language Question Answering System". [Online].Available: http://start.csail.mit.edu/

[12] M. Joshi, R. A. Akerkar, "Algorithms to improve performance of Natural Language InterfaceInternational Journal of Computer Science & Applications, vol. 5, no. 2, pp. 52-68, 2008.

[13] Seymour Knowles and Tanja Mitrovic, "A Natural Language Interface For SQL-Tutor", Nov. 5, 1999.