Natural Language Interface for Databases in Hindi Based on Karaka Theory

Aanchal Kataria
M.Tech Scholar
Department of Computer Science and Applications
Kurukshetra University
Kurukshetra, Haryana, India

Rajender Nath, PhD
Professor
Department of Computer Science and Applications
Kurukshetra University
Kurukshetra, Haryana, India

ABSTRACT
Data retrieval from the database requires knowledge of database language like SQL. Hence people with no knowledge of database language may find it difficult to access the database. Moreover, in recent time there is a growing demand for non-expert users to query relational database in a more natural language encompassing linguistic variables and terms. This triggered the idea of developing natural language interface to database where a user can query the database using natural language. And it relieved the persons from the burden of learning database language like SQL. In this paper, architecture of interface for converting natural language sentence in Hindi into the equivalent SQL query based on Computational Paninian Framework and Karaka theory is proposed. The interface has been evaluated experimentally and is found efficient.

Keyword
Hindi Language Query, Natural Language, Natural Language Interface for Database (NLIDB), Query, Structured Query Language (SQL).

1. INTRODUCTION
One of the major sources of information is database. Almost all applications need to access information from database that requires knowledge of database languages like SQL. A person having no knowledge of Structured Query Language (SQL) may find himself or herself handicapped while querying the database for data retrieval. Hence for overriding the complexity many researchers came with the idea to use natural language in place of SQL, which can be ideal for non-technical users who want to access database. This idea of using natural language for data retrieval has prompted the development of interface called Natural Language Interface for Database (NLIDB). The interface acts as a kind of communication channel between the non-professional users and the database. Main objective of NLIDB is to accept the natural language sentence in language the user is comfortable with and then apply the processing steps required and after conversion the user is provided with equivalent SQL query. The interface conceals all the inherent complexity involved in information retrieval based on unqualified user queries.

In this paper, Natural Language Interface for Database in Hindi is proposed using computational paninian grammar framework that uses karaka theory so that without any ambiguity the interface answers the user more efficiently. The presented interface supports simple, conditional and join queries. Moreover intimate the users about the query status in case no such data exists which users want to fetch.

The rest of the paper is organized as follows: Section 2 presents the literature review of NLIDB. Proposed work is presented in section 3 along with the architecture of the interface. Section 4 describes the results and experiments of proposed interface. Conclusion along with future work is presented in section 5.

2. LITERATURE REVIEW OF NLIDB
The field of NLIDB has gained a remarkable expansion in the last few decades. Numerous attempts have been made by many researchers to build the interface till now. Different approaches and frameworks were used by different interfaces in order to enhance the efficiency of the NLIDB interface.

The LUNAR was the first system that was informally introduced in 1971 [1], which answered questions about samples of rocks, which were brought back from the moon. The performance of LUNAR was very impressive and it could easily handle 90% of requests without any error [2]. Hendrix et.al in [3] proposed a system called as LADDER, which was designed for US navy ships that used linguistics synchronic linguistics to interrupt down inquiries to question a distributed data. The system LADDER was enforced in LISP [4]. Warren and Pereira in [5] proposed one of the best-known NLIDBs of the early eighties known as CHAT-80. It was implemented entirely in Prolog. It was used to transform English questions into Prolog expressions, which were evaluated on the Prolog database. The CHAT-80 was very impressive, efficient and sophisticated kind of system.

PRECISE [6] system was developed at the University of Washington and David Koin in 2004 by Alex Armanasu, Ana-Maria Popescu, Oren Etzioni, and Alexander Yates. The database of PRECISE was in the form of a relational database, which used SQL as the query language. PRECISE was being tested on two database domains out of which the first one was the ATIS domain and second one was GEOQUERY domain. El-Mouadib et.al introduced GINLIDB called as Generic Interactive Natural Language Interface to Database in 2009 [7]. It was designed by the use of UML and developed using visual basic .NET 2005. The system had two major components: Linguistic handling component and SQL constructive component. Linguistic handling component controlled the natural language query correctness and SQL constructive component generated the required SQL statement. The main advantage of this system was that it analyzed the given query with both syntactic and semantic merits to improve the correctness.

Neelu Nihalani et.al in 2009 [8] proposed intelligent layer for flexible querying in databases, based on semantic grammar approach. In this paper an intelligent layer was developed which can be incorporated with existing database system. This proposed interface employed a set of predefined training sets. The main benefit of these training sets was that they could be expanded or appended when the intelligent information system discovered some new knowledge.

Abhijeet Gupta et.al in [9] introduced NLIDB interface using the Computational Paninian Grammar Framework, which used two approaches syntactic followed by semantic approach. This system could be adapted to any domain by people having basic knowledge of the domain.
Axita Shah et.al in 2013 [10] introduced, NLKBIDB interface based on pattern matching and syntax based approach due to which interface was not only able to answer syntactically correct queries but also incorrect natural language queries. NLKBIDB took natural language query as an input and generated an output in tabular format if and only if generated SQL query was valid. On the basis of testing done, their result showed 53% increments in accuracy but the provided interface was domain dependent.

Mohit Dua et.al proposed a Hindi Language Graphical User Interface to Database Management system in 2013, which used pattern-matching approach [11]. The paper mainly discussed how a Hindi language sentence is mapped to equivalent SQL query. The proposed architecture consisted of four main components i.e. Tokenizer, Mapper, Query Generator and Database Management System. This interface supported selection, updating and deletion type of queries. The proposed interface was domain specific and passed testing of more than 100 queries.

D-HIRD: Domain –Independent Hindi Language Interface to Relational Database was proposed by Kumar Rajender et.al in [12]. The main feature of this interface was its domain independency, using pattern-matching approach. They introduced a domain-identifier component in database, which identified the domain by using knowledge base, which made it domain independent. The interface was able to correctly identify the domain from the query and translated it into SQL, which was executed on the database. The interface has been tested for 80 queries for each type of database.

3. PROPOSED NLIDB INTERFACE

In [11], the interface proposed was based on pattern matching approach in which command, table name, fields name and condition were predicted using knowledge base where each such fields were stored which could occur in Hindi language query provided by the user. By matching each token to the stored token SQL query was generated but in case table name and fields name were same, the interface had faced ambiguity problem as interface was not able to decide which was the table name and which was field name. Moreover, due to so many tokens stored for converting the Hindi query into SQL query the response time was high. Also this interface was domain specific i.e., works for single domain only.

To address the problems stated above, a new NLIDB interface in Hindi is proposed. It uses Karaka theory in order to find the relation between the table name and attribute name. The architecture of the proposed interface consists of six modules: Parser, Case Solver, Graph Generator, Query Translator, Knowledge Base and Query Executor (see Figure 1). All the modules along with their functioning are discussed as follows:

Parser: This module is responsible for providing tokens for a given Hindi language sentence submitted by the users with the help of shallow parser. The main function of the parser is to remove undesirable words, reporting corresponding base word and POS type for each token. Tokens here can be any of the following type i.e., can be table name, attribute name, condition, value, or any unwanted type of word. The output of this module is set of tokens after the removal of undesirable token.

For example in a given Hindi query "**उन विद्यार्थियों का नाम और शहर बताओ जिनके अंक** 79 **है**", there are 11 tokens but the parsed output "**उन विद्यार्थी नाम और शहर बता जो अंक** 79" have 9 tokens i.e., '**का**', '**है**' tokens are removed by shallow parser.

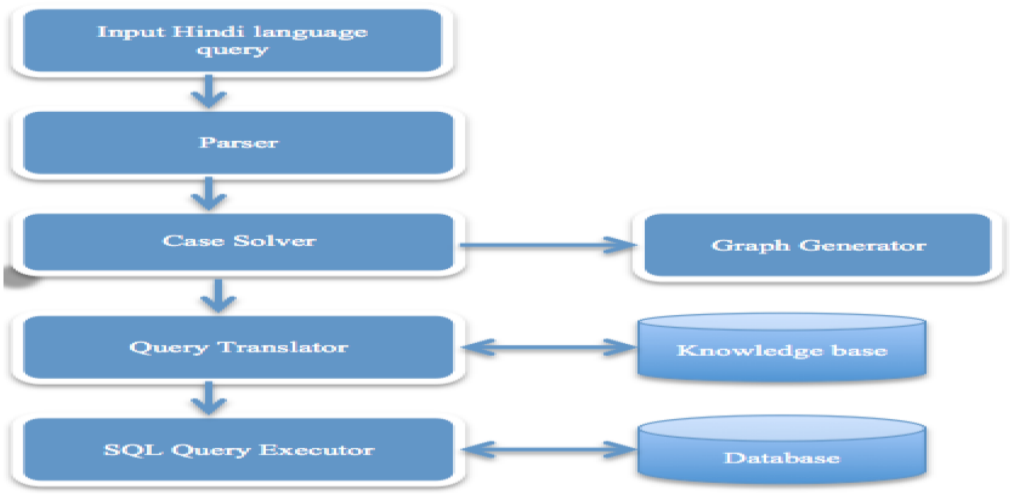


Figure 1: Architecture of Proposed NLIDB

As the output of this module provide us number of useful tokens along with their base word and POS type for example for the token '**बताओ**' the base word is '**बता**' and its POS type is verb which helps in predicting the command for the query. The parsed output of the file acts as the input for the Case Solver module.

Case Solver: Even after the removal of undesirable tokens in the parser module, there are still some tokens left which are not of any use. For example: '**उन**', '**सभी**', '**बारे**', '**में**' etc. These tokens are removed here. In this module after comparing the given Hindi language query and output provided by the parser module, a new Hindi language query is formed, which consists of base words. The new query formed is: "**विद्यार्थी का नाम और शहर बता जो अंक** 79".

Further processing is made in this query for the retrieval of command, table name, attribute name and conditional part if any exists. Following are the steps for the retrieval of required information in the translation of Hindi query into SQL query: (a) Command Retrieval (b) Table Name Retrieval (c) Attribute Name Retrieval (d) Conditional Part Retrieval.

(a) Command Retrieval: As the parsed output from the shallow parser predicts about the POS type so token with the POS type as verb is treated as the command, which is '**बता**' in the taken example. This information about the command is saved in the set.

(b) Table Name Retrieval: Now the tokens of the new formed Hindi language query is matched for the case symbols of the karaka theory i.e., **'का', 'के'** etc. After analyzing the structure of Hindi Language query it is found that the words before the case symbols talks about the entity, about which users wants to retrieve information. So the token before the case symbols represents the table name and it is stored in a set. In the example taken, **'विद्यार्थी'** is the name of the table.

(c) Attribute Name Retrieval: After getting the table name, the next step is to find out the attribute name. There can be a case the list of tokens of Hindi language may contain single or multiple attribute name or it may not contain even a single attribute name. Now for the retrieval of attribute name, the tokens after the case symbols are searched as it is found in karaka theory that attribute names are followed by the case symbols. The tokens are searched for attribute name until the field end token is found which is **'और'** in case of more than one attribute. There can also be a case that the attribute names are separated by "," and then by field end token **'और'** in that case also tokens are matched one position more until we get field end token. In case, no attribute name is specified explicitly then '*' is set as default column name. Now at this point of time all the attribute names are predicted and stored in a set. Here in the example it is resolved that **'नाम', 'शहर'** are the attribute names.

(d) Condition Part Retrieval: After noticing the structure of many Hindi language queries it is assumed that there exists some words that always come before the start of condition. These words here are called as condition start tokens which are stored in a list. The Hindi language tokens are now matched for the condition start token which can be '**जिनका**', '**जिनके**', '**जिसका**', '**जिसके**' etc base word for all these is '**जो**'. In case '**जो**' token is found, this confirms the existence of the condition field. Now the tokens after the condition start token are searched for the retrieval of condition attribute name and for their corresponding attribute value separated by the relational operators. In case more than one condition then it must be joined by logical operators, which is '**या**' and '**और**'. So for the first conditional part retrieval the tokens before the logical operators are matched and stored in a set and for the other conditional part retrieval the tokens after the logical operator is matched and stored.

For example: relational operator here is '=', condition attribute is '**अंक'** and its corresponding condition value is 79.

The main point to be noted here is that all the information retrieved about the query to be formed is still in linguistic form that is in the form provided by the user. This information is mapped to English tokens for SQL query in the subsequent modules.

Graph Generator: The case solver module provides a set for the command, table name, attribute name and conditional part if exists. This module is responsible for generating the graph by creating vertexes and edges showing the relationship between the command, table name, attribute name and conditional part. The graph for this particular query is shown in Figure 2.

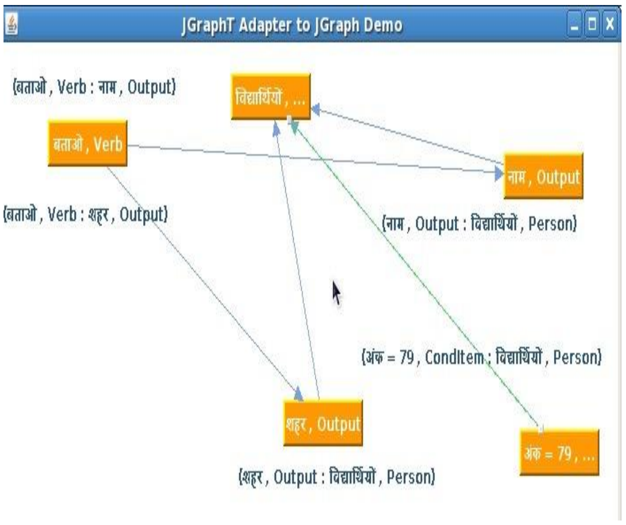

JGraphT Adapter to JGraph Demo
{बताओ , Verb : नाम , Output}
विद्यार्थियों , ...
बताओ , Verb
नाम , Output
{बताओ , Verb : शहर , Output}
{नाम , Output : विद्यार्थियों , Person}
{अंक = 79 , CondItem : विद्यार्थियों , Person}
शहर , Output
अंक = 79 , ...
{शहर , Output : विद्यार्थियों , Person}

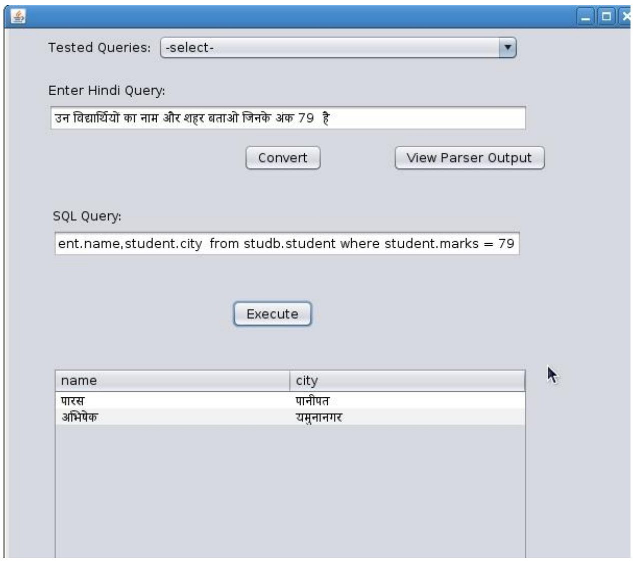
Figure 2: Generated Graph from Graph Generator

Query Translator: This is the module where main conversion from the natural language into the equivalent SQL query takes place. Query translator module process the output of case solver module which is set containing information about the command, table name, attribute name and condition attribute if exists. This module performs all the mapping of Hindi tokens using the knowledge base database for converting Hindi tokens into equivalent English tokens required for generating SQL query. After the mapping step, the required SQL query is generated.

So final SQL query generated corresponding to given example is: '**select student.name, student.city from studb.student where student.marks=79**'.

After all the above steps, query translator yields SQL query as output, which is fed to the SQL query executor module.

SQL Query Executor: SQL query executor takes SQL query as input from the query translator and yields the result set after fetching the data from the database. It generates the output in tabular form if data is available else message showing 'Data not found' is displayed as shown in Figure 3. Moreover this module is also responsible for storing the Hindi language query, its corresponding SQL query and the status of the query in database so that successful executed queries can be selected later easily from the combo box if users want to execute them again.



Enter Hindi Query:

उन विद्यार्थियों का नाम और शहर बताओ जिनके अंक 79 है

SQL Query:

```
ent.name,student.city  from studb.student where student.marks = 79
```

name	city
पारस	पानीपत
अभिषेक	यमुनानगर

Figure 3: Executed Query

Knowledge Base: Knowledge base stores all the Hindi tokens and their corresponding English tokens required for converting the Hindi language query into equivalent SQL query. This also stores the types of the tokens i.e., whether it is table name, attribute name, command, logical operator, relational operator or any value.

The Hindi tokens in query translator module also uses knowledge base for extracting their corresponding English tokens required for generating SQL query.

As the proposed architecture of the NLIDB uses the Karaka theory to find the relation between table name and attribute name so, interface is able to answer the users queries more efficiently and without any ambiguity. Moreover, the interface makes use of shallow parser for parsing the Hindi language query, which not only parse the sentence but also predicts the base word. This base word helps in reducing the mapping time as instead of storing each natural language token which represents the same meaning, its corresponding base word is stored in knowledge base. So at the time of mapping interface needs to search few tokens stored in knowledge base that leads to increase in speed of interface and reduction of response time. In this way, interface converts the Hindi language query into equivalent SQL query. The proposed interface is portable to new domains without many changes and without any expert help.

4. EXPERIMENTAL EVALUATION

The proposed NLIDB interface is implemented using Java Swings as front end with MySQL as backend. The interface is tested with 85 different queries. A partial list of queries is shown in Table 1. When a Hindi query is given to interface by user, the interface provided its corresponding standard SQL query as shown in Table 1. For evaluating the performance of NLIDB interface, the translation success rate is used which is defined as (No. of queries answered/Total no. of queries)*100. When tested on 85 queries experimentally it is found that the proposed NLIDB is giving 92.4% translation success rate, which is quite good and promising.

Table 1. Translated Queries

Query in Hindi	SQL Query Generated by NLIDB
सभी विद्यार्थी बताओ	select * from studb.student
उन विद्यार्थियों के विभागनाम और शहर बताओ जिनके अंक 79 से कमयाबराबर है	select student.department_name, student.city from studb.student where student.marks<= 79
उन विद्यार्थियों के शहर बताओ जिनका नाम पारस हो	select student.city from studb.student where student.name ='पारस'
सभी विभागों के नाम बताओ जिनकी संख्या 2 से ज्यादा या संख्या 1 है	select department.name from studb.department where department.id > 2 or department.id = 1

5. CONCLUSION AND FUTURE WORK

This paper has proposed a NLIDB architecture based on innovative computational pannian grammar framework that uses Karaka theory. The interface accepts users queries in Hindi and converts them into equivalent standard SQL query. The interface supported queries that include logical operators, joining of tables, selection of single or multiple columns and relational operators. The proposed interface is domain portable. In order to test the proposed NLIDB, it has been implemented using Java Swings as front end with MySQL as backend. By testing it on 85 queries, it has been found that the proposed NLIDB has given 92.4% translation success rate, which is quite good and promising. The interface failed to answer syntactically incorrect queries but this failure rate can be reduced by re-formation of the query.

Future work will consists of extending the capabilities of existing interface by enhancing it to accept queries in other languages. Moreover, types of queries that interface can handle can also be added. Also, the interface can be made platform independent.

6. REFERENCES

[1] Sujatha B, Viswanadha S.R, Shaziya H, (2012), "A Survey of Natural Language Interface to Database Management System", International Journal of Science and Advance Technology, vol. 2(6).

[2] Nihalani N., Silakari S., Motwani M., (2011), "Natural language interface for database: a brief review", International Journal of Computer Science, vol. 2(8), 600–608.

[3] Hendrix G., Sacrdoti E., Sagalowicz D., Slocum J., (1978), "Developing a Natural Language Interface to Complex Data", ACM Transactions on Database Sytems, vol. 3(2), 105-147.

[4] Sujatha B,ViswanadhaS.R,(2014),"A Flexible and Efficient Natural Language Query interface to databases", International Journal of Computer Science and Information

Technologies, vol. 5(5), 6464-6467.

[5] Warren D., Pereira F., (1982), "An efficient and easily adaptable system for interpreting natural language queries in Computational Linguistics", vol. 8, 3-4.

[6] Popescu Ana-Maria, Armanasu Alex, Etzioni, Ko David, Yates Alexander, (2004), "Modern Natural Language Interface to Database: Composing Statistical Parsing with Semantic Tractabilty", COLING

[7] El-Mouadib A. Faraj, Zubi S. Zakaria, Almagrous A. Ahmed, El-Feghi S. Irdess, (2009), "Generic Interactive Natural Language Interface to Database", International Journal of Computers, vol. 3(3).

[8] Nihalani Neelu, Silakari Sanjay, Moywani Mahesh, (2009), "Design of Intelligent layer for Flexible Querying in Database", International Journal on Computer Science and Engineering, vol. 1(2), 30-39.

[9] Gupta Abhijeet, Akula Arjun, Malladi Deepak, Kukkadapu Purneeth, Ainavolu Vinay, Snagal Rajeev, (2012), "A Novel Approach Towards Building a Portable NLIDB System Using the Computational Paninian Grammar Framework", International Conference on Asian Language Processing, 93-96

[10] Shah Axita, Pareek Jyoti, Patel Hemal, Namrata, (2013), "NLKIDB-Natural Language and Keyword Based Interface to database", International Conference on Advances in Computing, Communication and Informatics, 1558-1576

[11] Dua Mohit, Kumar Sandeep, Virk Singh Zorawar, (2013), "Hindi Language Graphical Interface to Database Management System", International Conference on Machine Learning and Applications, 549-553.

[12] Kumar Rajender, Dua Mohit, Jindal Shivani, (2014), "D-HIRD: Domain-Independent Hindi Language Interface to Relational Database", International Conference on Computation of Power, Energy, Information and Communication, 843-847.