Abstract—Databases have become ubiquitous. Almost all IT applications are storing and retrieving information from databases. Retrieving information from the database requires knowledge of technical languages such as Structured Query Language (SQL). However, majority of the users who interact with the databases do not have technical background and are intimidated by the idea of using languages such as SQL. This has led to the development of Natural Language Database Interfaces (NLDBIs). NLDBI allows the user to query the database in a natural language. This paper highlights on architecture of new NLDBI system, its implementation and discusses on results obtained.

The system architecture of natural language database interface developed is given in Fig. 1, which depicts the layout of processes included in converting NL query into a syntactical SQL query to be fired on the RDBMS.
To process a query, the first step is speech tagging, followed by word tagging. The second step is parsing the tagged sentence by a grammar. The grammar parser analyzes the query sentence according to the tag of each word and generates the grammar tree/s. Finally, the SQL translator processes the grammar tree to obtain the SQL query.

The paper is based on a unique concept of processing user natural language into a technical form so as to access the data from higher end data storage. NLDBI is a system that allows users to access a database in natural language and has been a popular field of study. Suppose we consider a properly normalized database. Now if the user wishes to access the data from the table, he/she accesses the tables in his/her language.

### III. GRAMMAR AND PARSING

Consider a sentence $w_1w_2w_3...w_m$ (ignoring punctuations), and each string $w_i$ in the sequence stands for a word in the sentence. The grammar tree of $w_1w_m$ can be generated by a set of predefined grammar rules; usually more than one grammar tree may be generated. The formalizing capability of grammar help in describing most sentence structures and built efficient sentence parsers.

A parser is one of the components in an interpreter or compiler, which checks for correct syntax and builds a data structure (often some kind of parse tree, abstract syntax tree or other hierarchical structure) implicit in the input tokens. The parser often uses a separate lexical analysis to create tokens from the sequence of input characters. Parsers may be programmed by hand or may be semi automatically generated (in some programming language) by a tool (such as Yacc) from a grammar written in Backus-Naur form.

The SQL translator generates query in SQL. Using grammar the parse tree is obtained from the input statement. The leaves of the parse tree are translated to corresponding SQL. Fig. 2 depicts the processing of English input statement to generate SQL query. The entire process involves tagging of input statement, apply grammar and semantic representation to generate parse tree, analyze the parse tree using grammar and translating the leaves of the tree to generate corresponding SQL query.

The database tables considered are EMP (empid, empname, salary, edepid, address, post, mobilenno), DEPT (deptid, deptname, deptloc, dcapacity) and PROJECT (pid, pname, epid). From the input NL statement, to generate parse tree the grammar written based on database tables is:

- **WhatKeyBank** → for | of | with | is | where | whose | having | in | on
- **AAnTheBank** → a | an | the
- **empid** → integer | id | number
- **empname** → string | name
- **salary** → integer | salary | income | earning
- **mgrid** → integer | manager | boss | superior
- **edeptid** → integer | id | number
- **deptid** → integer | id | number
- **deptname** → string | name
- **deptloc** → string | location
- **dcapacity** → integer | capacity
- **EmpTable** → employee | worker | person | emp | employees | emps | workers | persons
- **ProjectTable** → project | projects
- **DeptTable** → department | dept | dpt | departments | depts

The experimental work is to design an interface for generating queries from natural language statements/questions. It also consists of designing a parser for the natural language statements, which will parse the input statement, generate the query and fire it on the database. The experimental work will understand the exact meaning the end user wants to go for, generate a *what-type* sentence and then convert it into a query and handover it to the interface. The interface further processes the query and searches for the database. The database gives the result to the system which is displayed to the user. The following modules were developed.

- **An Interface:** It allows the user to enter the query in NL, interact with the system during ambiguities and display the query results.
- **Parsing:** Derives the Semantics of the statement given by the user and parses it into its internal representation, to convert NL input statement into what-type question for selection of data.
Query Generation: It generates a query against the user statement in SQL and passes on to the database.

The algorithm designed is put as mention below:

Algorithm 1 Generation of parse trees from NL statement using grammar:

1. Read Input Statement S
2. for each word $w_i$ from S do
   3. if ($w_i$ in Grammar $G$) then
      4. Add $w_i$ to Symbol Table $ST$
      5. end if
   6. end for
7. for each $w_i$ from $ST$ do
   8. Add $w_i$ to parse tree $T$ for $\textit{What-type}$ questions/s
   9. end for
10. Display $\textit{What-type}$ question/s $Q$
11. Read Input $Q$
12. for each $w_i$ from $Q$ do
13. if ($w_i$ in $G$) then
14. Add $w_i$ to parse tree $T$ for SQL-query
15. end if
16. end for
17. Display SQL-query

A. Scope of the Experimental Work

1. To work on a Relational database (RDBMS), one should know the syntax of the commands of that particular database software.
2. The Natural language processing is done on statements written in English language.
3. NL Input from the user is converted in the form of $\textit{what-type}$ questions only.
   For example: What is salary of employee with name Nikhil
4. A limited Data Dictionary is used where all possible words related to a particular system are included. The Data Dictionary of the system need to be regularly updated with words that are specific to the particular system.
5. Ambiguity among the words is taken care of while processing the natural language.
6. All the names in the input natural language statement have to be in double quotes.
   For example: Address of emp “Vivek”
7. Data dictionary used are: EMP, DEPT and ROJECT

IV. EXPERIMENTAL RESULTS

The system implemented was tested for variety of NL statements under various categories and the results obtained were satisfactory under the known constraints. The results were categorized based on the generation of unambiguous parse tree, ambiguous parse trees with two and three parse trees.

A. Generation of ambiguous parse trees.

The Fig. 3 shows the typical category of generating ambiguous parse trees.

1. The user expects the salary and department name of employee with id 2 and accordingly the statement that he gives to the system may be as under:
   “Salary and department name of emp of id 2”
2. The result generated depicts the ambiguous parse trees were the system is not able to identify the expected meaning of the statements. Instead it generates more than one parse trees leading two different meanings.
   For example:
   i. What is salary and department name of employee with id 2
   ii. What is salary and department name of department with id 2
3. The user here can interact to remove the ambiguity by choosing the appropriate options.
4. The SQL query is generated by the system which further fired on to the database to obtain the results as shown in Fig. 4 as employee salary – “12000” and department name – “Management”.

Fig. 4 Result for NL statement input to system.
V. CONCLUSION

The NL statement is converted into machine understandable form such as SQL. The NLDBI system is tested for more than 75 different NL input statements and the system works satisfactorily. The advantage of NLDBI system is that it works on a Relational database and removes ambiguities. So far, our NLDBI system considers selection of data and performing primitive queries onto the database and JOIN operation with some constraints. The next step of research is to optimize grammar to accommodate more complex queries.

ACKNOWLEDGMENT

I express my deep sense of gratitude and appreciation towards my research guide Prof. G. A. Patil for his continuous inspiration and valuable guidance in throughout my dissertation work.

REFERENCES

[1] In-Su Kang, Jae-Hak J. Bae, Jong-Hyeok Lee “Database Semantics Representation for Natural Language Access.” Department of Com Computer Science and Engineering, Electrical and Computer Engineering Division Pohang University of Science and Technology (POSTECH) and Advanced Information Technology Research Center (AITrc), 2002.


