RESEARCH ARTICLE OPEN ACCESS

Pattern based approach for Natural Language Interface to Database

Niket Choudhary*, Sonal Gore**
*(Department of Computer Engineering, Pimpri-Chinchwad College of Engineering, Savitribai Phule Pune University, Pune, India)
** (Department of Computer Engineering, Pimpri-Chinchwad College of Engineering, Savitribai Phule Pune University, Pune, India)

ABSTRACT
Natural Language Interface to Database (NLIDB) is an interesting and widely applicable research field. As the name suggests an NLIDB allows a naive user to ask query to database in natural language. This paper presents an NLIDB namely Pattern based Natural Language Interface to Database (PBNLIDB) in which patterns for simple query, aggregate function, relational operator, short-circuit logical operator and join are defined. The patterns are categorized into valid and invalid. Valid patterns are directly used to translate natural language query into Structured Query Language (SQL) query whereas an invalid pattern assists the query authoring service in generating options for user so that the query could be framed correctly. The system takes an English language query as input, recognizes pattern in the query, selects one of the before mentioned features of SQL based on the pattern, prepares an SQL statement, fires it on database and displays the result.
Keywords **-** natural language interface, pattern based, relational database, SQL

I. INTRODUCTION

Databases are an essential component for any enterprise. They contain huge volume of valuable data. To handle this large volume of data a powerful system called Database Management System (DBMS) is used. One of the most important features provided by a DBMS is Structured Query Language (SQL). It is used to store, retrieve and process structured data. But use of SQL restricts a naive user to retrieve his desired data. To overcome this problem, many researchers are continuously working on the concept of Natural Language Interface to Database (NLIDB). An NLIDB takes input a query in natural language, translates it into SQL and fires it on the database [1]. The NLIDB is a branch of more comprehensive subject namely Natural Language Processing (NLP). NLP is concerned with creating an easy and user friendly environment to interact with computer without requiring some programming language skills. Through NLP one can interact with computer in his natural language.

Many interesting theories and approaches have been developed so far about how to develop an NLIDB with improved accuracy [2], able to handle more natural language expressions [3,4] and able to guess real requirement of user who has not properly asked the query [5]. In this paper a novel approach for NLIDB is proposed. The proposed system is given the name Pattern based Natural Language Interface to Database (PBNLIDB). The systems developed so far are able to handle simple queries and join between two tables. In PBNLIDB, other than providing more knowledge of SQL functionalities to the translation function, features such as error detection and query authoring at query asking phase are employed. PBNLIDB allows a non-SQL expert user use more functionalities of SQL than provided by any other systems developed so far. In the proposed system patterns are defined for SQL features like simple query, aggregate function, relational operator, short-circuit logical operator and join. These patterns are categorized into valid and invalid patterns. Natural language queries containing valid patterns are mapped to their corresponding SQL template and thus translated into an SQL query. Queries containing invalid patterns are processed by query authoring service and options are presented to the user assisting him in framing correct query.

In this article contents are organized as follows. Section II explores some of the earlier NLIDB systems; section III describes the proposed work; finally section IV concludes the paper with some opportunities of future work.

II. RELATED WORK

In recent times, there have been rising demands by non-expert computer users to query relational databases in natural language. Actually, research in NLIDB was started in the decade of 1960 [1]. The Lunar Science Natural Language Information System (LSNLIS) [6] was the first system based on the concept of NLIDB. It was actually a question-answering system. It was developed for the

geologists who were studying about rocks on moon. The information was obtained by the Apollo missions. It was a waste of time and cost to teach the geologists the programming skill to process and retrieve data. Similarly LADDER was an NLIDB developed for US Naval ships [7]. Then in late seventies RENDEZVOUS System [8] appeared. This system first presented the use of paraphrasing and clarification dialog with the user in case the system was not able to parse the input. In the eighties CHAT-80 [9] was one of the most referenced NLP systems. It was implemented in Prolog and the database was consisted of world facts like oceans, major seas, major rivers and major cities of 150 countries. It also consisted of a small English vocabulary package required to process the query. ASK, developed in 1983, was the system that was able to work upon multiple databases simultaneously. NALIX [10], developed in 2005, is a natural language interface to XML. PRECISE [2] presented an interesting idea of making interaction with user more human like. In this if user asks one question and then asks another similar question by only changing the values then he does not need to ask the complete question. He can only ask partial question and remaining words are taken from previous question. For example if user first asks a query "who is the author of Algorithms?" then while asking second query he does not need to ask complete query, he can only ask "Database?". It will automatically be taken as "who is the author of Database?". Generic Interactive NLIDB (GINLIDB) [11] came with a component namely database adaptor which allowed the system to interact with multiple DBMS tools. It is used to set the environment according to the DBMS tool in use. One very recent NLIDB namely AskMe [12] presented a feature query-authoring service which helps a user to frame the query properly so that it can be validated before getting fired on database. It does not wait for the query to be fired on database and get error if any. It identifies error while framing the query.

It is difficult to say which existing interface is the best but through the study; the problem that we found is that they do not support most of the features of SQL. This made us to work in this direction. We developed the interface with the intention that it supports more features of SQL than provided by any other system developed so far.

III. PROPOSED SYSTEM

The proposed system is an extension of natural language interface for CINDI virtual library [3]. The only difference is that their idea of templates is replaced with more sophisticated Expert System which is responsible for converting English language query into SQL query. Architecture of proposed NLIDB namely PBNLIDB is shown in Fig. 1. Initially hyponyms, hypernyms and synonyms of all table and column names are found out using WordNet and stored in the knowledge base. Database Administrator can further add or remove words to or from the knowledge base. This pre-processing is done only once. This pre-processing is done only once or when there is a change in the database schema. Knowledge base allows a user to use similar words rather than the exact words present in the database. For example the words "income" or "salary", both are allowed in a query even if only "salary" is one of the column names and not "income".

At run time initially an English language query is accepted. This query is syntactically parsed and tagged using Link Parser. Then it is tested semantically- whether the query asked is relevant to the database or not. Finally the most important component Expert System is used to translate the asked query into SQL query. Fig. 2 shows internal modules of the Expert System. In the Expert System patterns of various features of SQL like simple query, aggregate function, relational operator, short circuit logical operator and join among multiple tables are defined. These features are discussed in the following sections taking the example of an employee database. The E-R diagram of employee database is shown in Fig. 3.

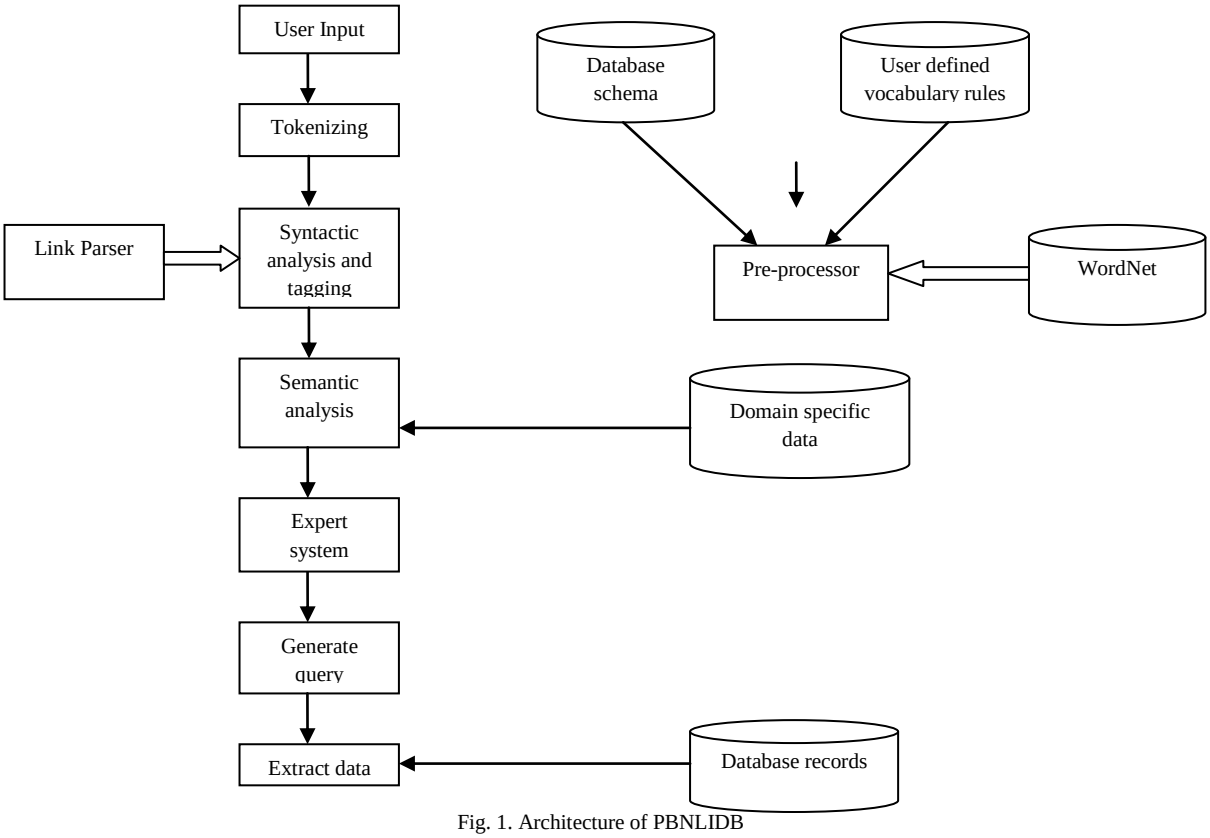

Fig. 1. Architecture of PBNLIDB

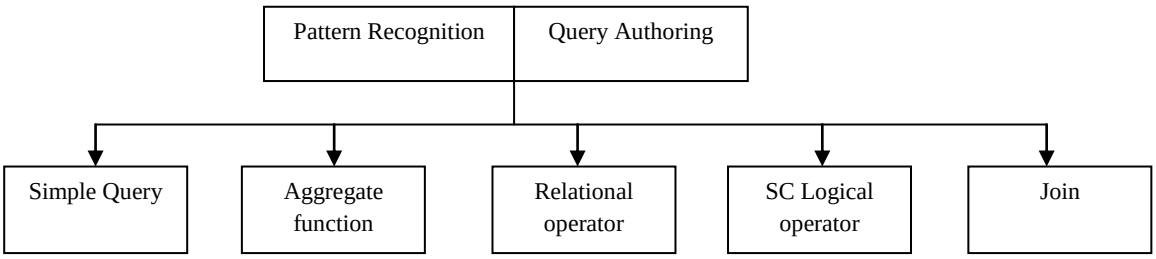

Fig. 2. Internal Modules of Expert System

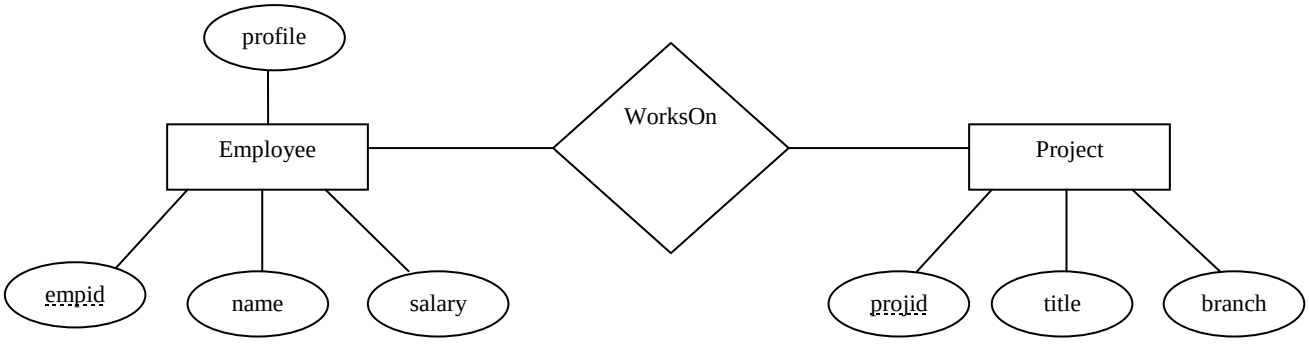

Fig. 3. E-R diagram of an employee database

Once pattern of these functions is identified, one of these features is selected, an appropriate SQL statement is prepared, fired on database and result is displayed to the user. To begin with the discussion of features included in PBNLIDB, first refer to Table I for various acronyms used for defining patterns.

TABLE I. Acronyms used for defining Patterns

Sr. No.	Acronyms	Meaning
1.	DC	Desired Column
2.	TB	Table Name
3.	AF	Aggregate Function
4.	PC	Predicate Column
5.	RO	Relational Operator
6.	VL	Value
7.	LO	Short circuit logical operator

An SQL template containing all the acronyms mentioned in table 1is as shown below:
select <AF><DC> from <TB> where <PC1> <RO1> <VL1> <LO1> <PC2> <RO2> <VL2> <LO2> ...

3.1 Simple Query

This includes all those queries in which only TB or DC with their TB are mentioned. Following are few examples of simple queries based on employee database.
Example 1: List details of all employees.
Here "details" tells to select all column names and employee is identified as TB. So the SQL query generated is
SELECT * FROM EMPLOYEE
Example 2: Show the names and salaries of all employees.
In this example name and salary are identified as DC and employee as TB. So SQL query generated is
SELECT NAME,SALARY FROM EMPLOYEE
Thus valid patterns for simple query are- {TB}, {DC, TB} and invalid pattern is {DC}.
Invalid pattern {DC} indicates that asking only column names without their table name is not allowed and user is asked to further clarify the query by selecting a table name from many, produced as options by query authoring service.

3.2 Queries with aggregate function

Various aggregate functions supported by our system are count, sum, avg, min, max and distinct. Actually more than these can be seen in a DBMS tool but in our interface, only the aforementioned aggregate functions are included. Few examples mentioned below are the queries that require aggregate function:
Example 1: What is the number of employees working in the industry?
Here AF is count, DC is name and TB is employee. Name is identified as DC because it serves as default attribute [3] for the table. So the SQL query generated is
SELECT COUNT(NAME) FROM EMPLOYEE
Example 2: What is the total salary of all employees?
Similar to previous example here SQL query generated is
SELECT SUM(SALARY) FROM EMPLOYEE
"total" in the query is mapped to SUM using the knowledge base.

In the above examples one aggregate function, column name and table name is mentioned. If column name is not mentioned then default attribute is taken as column name. Aggregate function like sum cannot be applied onto a column with string data type. In such a situation again query authoring service informs the user to select one of the columns of that table whose data type is integer or float. Thus based on this observation valid patterns for aggregate function are- {AF,TB}, {AF,DC,TB} and invalid patterns include {AF}, {AF,DC}. In this feature there cannot be more than one AF, DC and TB.

3.3 Queries with Relational Operator

The relational operators processed by SQL are as follows:
Equal to (=),
Not equal to (!=),
Greater than (>),
Greater than or equal to (>=),
Lesser than (<) and
Lesser than or equal to (<=)
Consider following examples to understand the valid and invalid patterns for this functionality.
Example 1: Which employee has salary greater than 25000?
Here DC is name (as no columns are mentioned so default attribute is taken), TB is employee, PC is salary, RO is > and VL is 25000. Thus SQL query generated is
SELECT NAME FROM EMPLOYEE WHERE SALARY > '25000'.
Example 2: What is the empid and name of employee whose salary is greater than 25000?
In this case DC are empid and name, TB is employee, PC is salary, RO is > and VL is 25000. Thus SQL query generated is
SELECT EMPID,NAME FROM EMPLOYEE WHERE SALARY > '25000'

The above example indicates that a query requiring relational operator functionality must have at least one TB, one PC, one RO and one VL.

Thus valid patterns for using relational operator are- {DC,TB,PC,RO,VL}, {TB,PC,RO,VL} and invalid patterns are- {PC}, {RO}, {VL}, {PC,RO}, {PC,VL}, {RO,VL}, {DC,PC}, {DC,RO}, {DC,VL}, {TB,PC}, {TB,RO}, {TB,VL}, {DC,TB,PC}, {DC,TB,RO}, {DC,TB,VL}, {DC,PC,RO}, {DC,PC,VL}, {DC,RO,VL}, {TB,PC,RO}, {TB,PC,VL}, {TB,RO,VL}, {DC,TB,PC,RO}, {DC,TB,PC,VL}, {DC,TB,RO,VL}.

3.4 Query with short-circuit logical operator

The short circuit logical operators are "AND" and "OR". They are used in predicate section of an SQL query for applying more than one constraint. Consider following examples for identifying the need of short-circuit logical operators.

Example 1: Who is the employee whose salary is more than 20000 and profile is manager?

In this example DC is name (default attribute), TB is employee, PC are salary and profile, RO are > and = , VL is 20000 and manager and LO is "and". So the SQL query generated is

SELECT NAME FROM EMPLOYEE WHERE SALARY > '20000' AND PROFILE = 'MANAGER'

Example 2: Who is the employee whose salary is more than 20000 and less than 50000?

Here DC is name, TB is employee, PC is salary, VL is 20000 and 50000, RO are > and < and LO is "and". Thus SQL query generated is

SELECT NAME FROM EMPLOYEE WHERE SALARY > '20000' AND SALARY < '50000'

Example 3: Who is the employee whose salary is more than 20000 and less than 50000 and profile is either manager or team leader?

In this case DC is name, TB is employee, PC are salary and profile, RO are >, <, = , VL are 20000, 50000, manager and team leader. The SQL query generated is

SELECT NAME FROM EMPLOYEE WHERE SALARY > '20000' AND SALARY < '50000' AND PROFILE = 'MANAGER' OR PROFILE = 'TEAM LEADER'

Observing the above examples it is clear that at least one TB, more than one PC with their RO and VL or one PC with more than one RO and VL has to be present separated by "and" or "or".

Thus valid patterns for short-circuit logical operator are- {TB,PC,RO,VL,LO}, {DC,TB,PC,RO,VL,LO}. For invalid pattern mainly count of various entities in the set are considered. For example the number of PC and VL must be equal if there are distinct PCs are used. For one PC multiple VL are possible. The count of LO is one less than the number of VL. In a query there can be many PC with different count of their VL. To handle such situation their position in the query is considered. To identify VL of PC1 all VL present between PC1 and PC2 are taken. Similarly for VL of PC2 all VL between PC2 and PC3 are taken and so on. For final PC, PC3, VL coming after PC3 and before end of the query are taken as VL for PC3. In this case there are no special patterns are defined, instead the count and position of various PC,RO and VL are considered and valid and invalid patterns of relational operators are used with LO as one more data of the set.

3.5 Queries with join

Join is used to fetch records from multiple tables. Following are the examples considered for identifying the valid and invalid patterns of Join operation.

Example 1: Which employee is doing the project of ABC Bank?

Here there are two different TB are used- one is employee and another is Project. In this case, the relationship "WorksOn" is also taken by default. The SQL query generated is

SELECT EMPLOYEE.NAME FROM EMPLOYEE,WORKSON,PROJECT WHERE EMPLOYEE.EMPID = WORKSON.EMPID AND WORKSON.PROJID = PROJECT.PROJID

Example 2: Which employee is doing the project of ABC Bank in Mumbai branch and salary is 2000?

Here extra conditions are put other than default as in the previous example. Extra conditions include PC as project.branch and employee.salary, RO as = and =, VL as Mumbai and 2000. Thus SQL query generated is

SELECT EMPLOYEE.NAME FROM EMPLOYEE,WORKSON,PROJECT WHERE EMPLOYEE.EMPID = WORKSON.EMPID AND WORKSON.PROJID = PROJECT.PROJID AND PROJECT.BRANCH = 'MUMBAI' AND EMPLOYEE.SALARY = '20000'

Observing the above examples we get that a query in this group contains more than one TB plus it may contain extra predicates. Without any extra predicate default SQL template is used but presence of extra predicate makes it follow the rules of relational operator or short circuit logical operator as well.

Thus briefly explaining, a set of valid and invalid patterns of various features of SQL are defined in the Expert System. Query Authoring module of Expert System is used to provide hints to the user in case the query asked contains an invalid pattern. Hints are provided based on valid pattern of the feature for which invalid pattern is identified. The sets formed in both valid and invalid group of all features are unique. Thus there is no conflict in identifying a feature based on the pattern identified from the English language query.

IV. CONCLUSION AND FUTURE WORK

In summary, the proposed system is an extension of natural language interface for CINDI virtual library. For translation of English language query into SQL query their templates based module is replaced with more sophisticated Expert System. The intent of using patterns is to support more complexities in the translation function and making the interface more aware of the features of SQL. Currently, the proposed system supports simple query, aggregate function, relational operator, short-circuit logical operator and join. For each of these features, a set of valid and invalid patterns are defined. These patterns are unique throughout the features. In case, any invalid pattern is identified, query authoring service provides options to the user containing valid patterns for that feature. Short circuit logical operator and join work in collaboration with the patterns of relational operator. There is no need to define lengthy and duplicate patterns for short circuit logical operator and join. Our next target will be to make the interface aware of other features of SQL like clauses (group by and order by), keywords used in predicate like BETWEEN, NOT IN, IN etc., nested query, varieties of join, union, intersection, difference and much more. More attention will be made towards developing interface in such a manner that a new feature could use the patterns or functionalities of its previous features.

REFERENCES

[1] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch, Natural Language Interfaces to Databases – An Introduction, *Journal of Natural Language Engineering 1 Part 1*, 1995, 29–81.

[2] A. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates, PRECISE on ATIS: Semantic Tractability and experimental results, *Proceedings of the National Conference on Artificial Intelligence – AAAI*, 2004, 1026–1027.

[3] N. Stratica, L. Kosseim and B.C. Desai, Using Semantic Templates for a natural language interface to the CINDI virtual library, *Data & Knowledge Engineering Journal 55 (1)*, 2004, 4–19.

[4] H. Young, and S. Young, A data-driven spoken language understanding system, *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, *vol. 1*, 2003, 583–588.

[5] J.L. Vicedo, and A. Ferrandez, Importance of pronominal anaphora resolution in question answering systems, *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 2000, 555–562.

[6] W.A. Woods. Progress in natural language understanding: An application to lunar geology, *AFIPS '73: Proceedings of the June 4–8, national computer conference and exposition, ACM*, 1973, 441–450.

[7] G. Hendrix, E. Sacrdoti, D. Sagalowicz, and J. Slocum, Developing a natural language interface to complex data, *ACM Transactions on Database Systems, vol 3*, 1978, 105–147.

[8] E.F. Codd, Seven steps to rendezvous with the casual user, *IFIP Working Conference Data Base Management*, 1974.

[9] D. Warren, and F. Pereira. An efficient and easily adaptable system for interpreting natural language queries, *American Journal of Computational Linguistics, vol. 8*, 1982, 3-4.

[10] Y. Li, H. Yang, H.V. Jagadish, NALIX: an interactive natural language interface for querying XML, *Proceedings of the International Conference on Management of Data*, 2005, 900–902.

[11] P.R. Devale, and A. Deshpande, Probabilistic context free grammar: an approach to generic interactive natural language interfaces to databases, *Journal of Information, Knowledge and Research in Computer Engineering, vol. 1*, 2010, 52–58.

[12] M. Llopis, A. Ferrandez, How to make a natural language interface to query databases accessible to everyone: An example, *Computer Standards and Interacts, Elsevier*, 2012, 470-481.