A Comparative Study of Natural Language Interfaces to Databases
Aanchal Kataria1 Dr. Rajender Nath2
1M.Tech Scholar 2Professor
1,2Department of Computer Science and Applications
1,2Kurukshetra University, Kurukshetra, Haryana

Abstract— One of the most important sources of information, which plays a significant role in day-to-day life, is database. But using database languages like SQL for dealing with database is always a difficult task, which lead to development of natural language interface for databases. Natural language interface is step towards the development of an intelligent interface for making it easier for the novice users to access the database. This paper reviews the various NLIDB techniques referred in the literature and suggests future research.

Key words: Natural Language, Natural Language Interface, Natural Language Interface to Database (NLIDB), Structured Query Language (SQL)

I. INTRODUCTION
People around the world need to deal with database for data retrieval for which database languages like SQL is required. To write SQL query one need to have knowledge of formal query language. But everybody is not able to write SQL queries. For overriding the complexity many researches have turned out to use Natural Language (NL) i.e. English, French, Tamil, and Arabic etc. instead of SQL. One area in which natural language interfaces are powerful enough to be effective is database query system. The database can be consulted and appropriate response can be generated. Much research is still going on in natural language database interface.

NLIDB provides so many advantages to the users like knowledge of physical structure of data is not required as to query in formal language one must have knowledge of location of data where it is store, but in case of NLIDB no requirement is needed. Formal languages

Used for querying are difficult to learn and master at least by non-computer specialists. But with use of natural language interface for database user is not required to learn an artificial communication language. Despite of having so many advantages, NLIDB has certain drawbacks. The linguistic capabilities of NLIDBs are not clear to the users. Already existing NLIDBs can only deal with limited subset of natural languages. Users may find it very difficult to understand what kind of query, the NLIDB can or cannot deal with. NLIDB users are often misled by the interface ability to process natural language and they assume that the system is intelligent, that it has common sense, or that it can deduce facts, while most of the NLIDBs have no reasoning capabilities. Natural language is claimed to be too ambiguous for human-computer communication. This may leave user with multiple answer of same question.

II. REVIEW OF EXISTING WORK
Prototype for NLIDB had appeared in late sixties and early seventies. Since then, numerous attempts have been made to build useful natural language interface. Querying databases in natural language is very convenient and easy method of data access especially for common users who do not understand complex database query language.

The LUNAR was the first system was informally introduced in 1971 (1), which answered questions about samples of rocks, which were brought back from the moon. LUNAR system had two databases, one for chemical analysis and other for literature references. 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 used linguistics grammar technique that interleaves grammar a linguistics method. The question respondent was completed via parsing the input and mapping the break down tree to a data question. The system LADDER used to depend on a three-bedded style. The first part of the system was for casual communication Access to Navy data (INLAND), which accepts queries throughout a communication and produces an issue to the data. The queries from the midland unit of measurement directed to the intelligent data Access (IDA) that was the second a part of LADDER. The third a part of the LADDER system was for File Access Manager (FAM). The function of FAM was to look after the position of the generic files and manage the access to them. 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. The database used in this consisted of facts (that is oceans, major seas, major rivers and major cities) about 150 countries of the world and a small set of English language vocabulary that was enough for querying the database.

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. It had presented the idea of semantically tractable sentences that could be converted to a unique semantic interpretation by analyzing some lexicon and semantic constraints. 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-Moudabid 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. First one was Linguistic handling component controlled the natural language query correctness as far as the grammatical structure and the
possibility of successful transformation to SQL statement and second one was SQL constructive component generated the required SQL statement, opened a connection to the database in use, executed the generated SQL statement and returned the query's result to the user. 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. Initially, the flexible queries from users in their natural language were submitted to intelligent layer and this layer converted the amorphous query into structured SQL query. The query formed was executed and results were presented to the user. It enabled the design of a knowledge based self-learning based system i.e., the values obtained from the user, which aided the selection of appropriate SQL query, when the same flexible query was issued in the future. 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. But the proposed system was developed for particular domain only thus making it domain dependent.

Abhijeet Gupta et.al in (9) introduced NLIDB interface using the Computational Paninian Grammar Framework, which used two approaches syntactic followed by semantic approach. The developed system used four modules for the conversion of natural language query into SQL query: Normalization Module, Syntactic Parsing Module, Semantic Mapping Module and Query Generation Module. The syntactic parser used the Computational Paninian Grammar (CPG) Framework; it had advantages over usual dependency trees because the dependency relations were closer to semantics. At the same time because these were syntactic relations, it was possible to build robust wide coverage syntactic parsers easily. The parse tree so produced was mapped to semantic relations using domain specific semantic frames. The latter process was domain specific and saved us from the need to build any general-purpose semantic parser, which was error prone. Moreover this system could be adapted to any domain by people having basic knowledge of the domain.

Axita Shah et.al in 2013 (10) introduced, NLKIDB 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. NLKIDB took natural language query as an input and generated an output in tabular format if and only if generated SQL query was valid. This had Natural Language Agent that took natural language query as an input then it passed to the Lexical Analyzer, Syntax Analyzer and Semantic Analyzer. Natural Language agents sent analyzed tree structure to the SQL Generator if and only if Syntax of Natural language user query was valid. Keyword Based Agent looked after the grammatical incorrect syntax queries. It applied mapping of tokens with the generated knowledge base. If token found then it was forwarded to the SQL generator else user was informed to reform the natural language query. On the basis of testing done, their result showed 53% increments in accuracy but this provided system was domain dependent.

Dua Mohit 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 system supported selection, updating and deletion type of queries. The system proposed by them 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 system 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 make system domain independent. The system was divided into two modules: language processing module and database module. Language Processing Module performed all language related functions like tokenization, parsing, part of speech tagging. Database module was responsible for executing the SQL query. It has three components: Domain identifier, SQL query generator and SQL query executor. The system was able to correctly identify the domain from the query and translated it into SQL, which was executed on the database. This system has been tested for 80 queries for each type of database.

III. COMPARATIVE STUDY OF EXISTING NLIDB INTERFACES

There has been large number of research works introducing the theories and implementations of NLIDBs. On the basis of review of existing work in section II the NLIDB interfaces can be further categorized on the basis of approach used and on the basis how NLIDB related to the domain knowledge of database. There are mainly four kinds of NLIDBs approaches- Pattern Matching based approach, Syntax based approach, Semantic Grammar based approach and Intermediate Representation Language based approach (13).

A. Pattern Matching Based Approach

The first type of framework was based on pattern matching. A typical application of this type of framework was SAVVY (14). Many of the systems were depend on pattern matching to directly map the user input to the database. Formal List Processor (FLIP) was an early language for pattern-matching based on LISP structure worked on the bases that if the input matched one of the patterns then the system was able to build a query for database. The main advantage of the pattern-matching approach was its simplicity. Systems using this approach required no parsing and interpretation modules, moreover systems were quite easy to implement. Also, pattern-matching systems usually manage to come up with some reasonable answer, even if the input given was out of the range of sentences that the patterns were designed to handle. But the patterns matching based systems were too shallow and hence often lead to bad failures.

Weizenbaum in 1966 (15) developed a first dialog system – ELIZA that was based upon the pattern matching approach. It was written in a language called MAD
(Michigan Algorithm Decoder), using a package called SLIP (Symmetric List Processor), which Weizenbaum originally developed in 1963.

B. Syntax Based Approach
This approach was basically used in application-specific database systems. A database query language must be provided by the system so that the mapping from parse tree to the database query could be accomplished (13). It was usually difficult to devise mapping rules that will transform directly the parse tree into some expression in a real-life database query language like SQL. The main advantage of using syntax-based approaches was that they deliver detailed information about the structure of a sentence.

Syntax-Based Approach was proposed in LUNAR system in (16) which answered about rock samples brought back from the moon. In this system a parsing algorithm was used to generate a parse tree depending on user’s queries. It could not handle ungrammatical sentences and was not flexible.

C. Semantic Grammar Based Approach
In semantic grammar based approach based interfaces, the question answering was still done by parsing the input and mapping the parse tree to a database query. The difference, in this case, was that the grammars categories did not necessarily correspond to syntactic concepts. Semantic information about the knowledge domain was hard-wired into the semantic grammar that is why systems based on this approach were very difficult to port to other knowledge domains for which a new semantic grammar has to be written whenever the NLIDB was configured for a new knowledge domain.

Warren and Pereira in (5) introduced system using this approach known as CHAT-80. CHAT-80 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 used to be very impressive, efficient and sophisticated kind of system.

D. Intermediate Representation Language Based Approach
Some intermediate representation languages could be used to convert the statements in natural language to a known formal query language. Due to the difficulties in syntax-based approach in translating a sentence into a general database query languages, the intermediate representation systems were proposed. Moreover, because the logic query language was independent from the database, it could be easily ported to different database query languages as well as to other domains, for example to other expert systems and operating systems.

Androustopoulos et.al in (17) introduced MASQUE/SQL system using this approach, which used this language as a front-end language for relational database that could be reached through SQL.

All the NLIDB discussed above can be classified into two sub categories on the basis of domain knowledge of database: Domain-dependent NLIDBs and Domain Independent NLIDBs (18).

Domain Dependent NLIDB needs to know the particularities about the underlying domain entities and restrictions imposed on them. This NLIDB can be any of following three types i.e., Non-Reconfigurable NLIDB, Reconfigurable NLIDB and Auto-Reconfigurable NLIDB.

Non-Reconfigurable NLIDB is designed ad-hoc bases for a particular database. Reconfigurable NLIDB can be used for other domain as well with little more modification. Auto-reconfigurable NLIDB which is the most interesting from a cost-saving perspective (19,20) as it allows NLIDBs that are knowledgeable about the underlying domain data, to provide more accurate information, error messages, etc. and at the same time this enables non-technical users to connect to multiple databases without the need for manual reconfiguration.

Domain-independent NLIDBs allowed the user to write queries in a natural language and that do not store any knowledge about the underlying domain, they simply translated NL queries into SQL queries and executed them against the underlying database (2). Since the system does not know anything about the domain, it is not able to warn the user about conceptual errors in the query (entity–property mismatch, data type mismatch, etc.) and therefore the error catching happens in the database, thus, making the system slower and less user-friendly. An example of NLIDB system in this category was PRECISE (6).

Table I summarize various existing NLIDB interfaces, which are based on different approaches and provides certain contributions to the field of NLIDB along with certain scope of improvements that can be made in these work.

Neelu Nihalani et.al (18) proposed the interface, which was domain dependent thus, making it useful for only domain for which it was developed. Abhijeet Gupta et.al (19) developed interface, which used two parsers, which made whole processing slow. Moreover it did not support natural queries having quantifiers and aggregation operators. The interface introduced by Axita Shah et.al (20) was only able to answer those syntactically incorrect queries for which the pattern was stored. For the remaining queries the interface failed to answer. Amisha Shingala et.al (21) introduced interface, which was able to handle natural language query pertaining to non-Overlapping domain only. So, it leads to bad failure if operated with other domain, which made it too specific. The interface given by Mohit Dua et.al (8) and Rajender Kumar et.al (9) had done mapping of each generated token with the tokens of lexicons. Due to so many stored lexicon tokens the searching time was more, which lead to high response time.

<table>
<thead>
<tr>
<th>Published Work</th>
<th>Approach Used</th>
<th>Domain Dependency</th>
<th>Language of Communication</th>
<th>Remarks</th>
</tr>
</thead>
</table>
Axita Shab et.al in 2013 [10] | Pattern Matching and Syntactic | Domain Dependent | English | Provides results for both syntactically correct and incorrect queries.

Amish a Shingal a and Paresh Virp a in 2013 [21] | Intermediate Language | Domain Dependent | English | Can handle natural language query pertaining to non-overlapping domain.


Rajender Kumar et.al in 2014 [12] | Pattern Matching | Domain Independent | Hindi | System can be ported to different database without any extra cost.

Table 1: Techniques Used in Papers

IV. CONCLUSION AND FUTURE WORK

Relational data models are widely used for construction of database system and hence the demand of natural language interfaces to database system arises. The importance of this system lies in the fact that it uses natural language for querying the database. Natural language interface for database is introduced in this paper for this reason. This paper has discussed the existing work on NLIDBs and has compared them on various parameters such as approach used, domain dependency and language of communication. On comparison it has been found if they use double parsers for better accuracy in efficient and slow. If they are domain-dependent then they become quite specific and cannot be used for other domains without major modifications. The area of NLIDB is very active and application oriented. Researchers in this area need to address the issues discussed above in their future work.

REFERENCE


