An Algorithm for Solving Natural Language Query Execution Problems on Relational Databases

Enikuomehin A.O., Okwufulueze D.O.
Dept. of Computer Science,
Lagos State University
Lagos, Nigeria

***Abstract*— There continues to be an increased need for non-experts interaction with databases. This is essential in their quest to make appropriate business decisions. Researchers have, over the years, continued to find a methodology that bridges the gap that exist between information need and users satisfaction. This has been the core in studies related to natural language information retrieval. In this paper, we understudy the existing methodology and develop a model to extend the proposition of (a) Bhardwaj et al where a MAPPER was developed and implemented on student database and (b) Nihalani et al. where an integrated interface was used on relational databases. We present a time saving executable algorithm that satisfies needed conditions required to retrieve results of natural language based queries from relational databases. Results of the experiment shows that the performance index of the algorithm is satisfactory and can be improved upon increasing the metadata table of the relational database. This is a sharp diversion from the keyword based search that has dominated most commercial databases in use today. The implementation was deployed in PHP and the retrieval time has compared favorably with earlier deployed models. We further propose the extension of this work in the areas of inculcating some fuzzy constraints to handle uncertainty and ambiguity which are inherent in human natural language.**

Keywords- Relational Database; Interface; Natural Language; Query; SQL.

I. INTRODUCTION

Research work on developing a flexible Natural Language Interface for Relational Databases has experienced expansion at a very high rate [1]. This has led to continuous research on natural language interfaces and query execution related issues. However, the attention received in this area has not led to significant and commensurate improvement in the existing models for natural language information retrieval essentially in the areas related to development of human useable interfaces. This complexity has been linked with the discreetness required for information extraction from relation databases by the autonomous use of Structured Query Language (SQL). SQL (Structured Query Language) is the formal querying language for relational databases. This is an expert language that is; users need to learn a specific syntax to initiate an appropriate query. In contrast, most business individuals are not experts in this domain and have causes to relate with the relational databases. Obviously, there is a need for this category of users to interact consistently with the content of the relational databases. This paper discusses some of the approaches that had been introduced to enable users query the database using their natural languages rather than SQL. These developed approaches enable database queries to be performed by users with little or no SQL querying abilities. However, some of the systems developed so far are not flexible enough to deal with the complexity associated with human users. Such earlier propositions force the user to adhere to strict grammatical rules when formulating queries. For appropriate usable results to be achieved, queries must be well posed against the relational database. The NLIDB will assist users to reformulate a natural language query into an appropriate SQL. The use of NLIDB has experienced rapid growth and continues to enjoy great support in terms of research and contributions.

If the above holds, one wonders why it is necessary to put some and energy in studying this process with the level of attention received. The answer is simple: the information seeking task becomes more complex and the available number of information object increases. This increment is being experienced by the day with the continuous exponential growth of the internet. This consideration clearly establishes that the existing tools for SQL generation may not be appropriate for some strictly defined domains; we therefore propose an algorithm that is flexible for extension to handle the information growth. In Enikuomehin et al [9], a proposal for handling natural language queries in LANLI was proposed. The resulting implementation performed considerably better than existing commercial interfaces however the time of execution has been a concern to researchers. The formalism involves that non SQL experts could pose a query which runs through a preprocessor. We advance on this proposition to save time and present a direct executable algorithm for natural language retrieval

II. BACKGROUND

Relational Databases (*a collection of data items organized as a set of tables for easy storage, manipulation and retrieval of data*) are becoming ubiquitous as there continues to be an increased need for people - mostly laypeople – to query databases and gain access to information. There is hardly any existing institution today that does not make use of a relational database in managing the massive amount of data the institution deals with. Such cases can be made for Government, Education, Religion, and Business amongst others. These relational databases however can be accessed using formal methods, which require a great deal of learning on the part of the user. This requirement is actually

challenging because, a user who is a novice in the methods used to access a database will find it really difficult to gain access to important information he/she may need at the moment. For example, consider a situation where an expert in database access could not perform his/her duties due to technical incompetence in the formulation of SQL queries. In the early generation of computers, a lot of skills, gotten from a formal and rigorous training in computer usage was required to operate the computer. Subsequent generations dealt with this rather difficult demand of an expert operator, and brought about an era where the less experienced could also operate the computer. To access a database, user must make use of a formal language which the relational database understands. One of such a formal language used to communicate with a database is SQL (Structured Query Language). The use of SQL requires some level of expertise, such expertise are normally acquired after due training. This paper presents a simple and easy-to-use natural language interface to enable less non technical users to have the capability to retrieve information from the relational database.

III. Some Related Earlier Works

Research in Natural Language Interface for Relational Databases began as far back as the 20th century. Since then the study and interest has continued to grow tremendously such that the area has become the most active in Human-Computer Interaction. The first Natural Language Interface for Relational Databases appeared in the 1970s*[2]*, the NLIDB system was called LUNAR*[]*. After the development of the first NLIDB, many were built which were supposed to be an improvement on the apparent flaws of LUNAR. The presentation and acceptance of LUNAR was huge. The reason for such huge success with NLIDBs includes the fact that there are real-world benefits or payoffs that can be derived from this area of study, other fact is that the earlier experimented domain was a single domain where execution of non complex systems are easy and easily adaptable. Same feet were not achieved in the area of using complex databases. [3] we highlight below, the development of some NL interfaces.

A. Lunar (1971)[4]

Man had accomplished the complex task of both having a physical presence on the moon and that of positioning satellites in space that can bring results from observations done on the moon. Information of rock samples brought back from the moon, for example, chemical information were stored in a database, while literature reference on various samples were stored in another database. LUNAR helped provide answers to queries about any of the two information about a rock sample by the use of these databases. LUNAR had linguistic limitations and was able to handle 78% of user-requests.

B. Philiqa [Philips Question Answering Machine](1977)[5]

This system works by having a clear-cut distinction of the syntactic parsing and semantics of the user-defined query. It has three layers of semantic understanding:

a. English Formal Language

b. World Model Language

c. Database Language

Together, these three layers work to answer user-defined queries. Users did not achieve so much acceptance as the earlier developed LUNAR.

C. Ask (1983)[6]

Ask was a complete information management system with an in-built database and the ability to communicate with multiple external databases using several computer applications which are accessible to users through the user's natural language query. Learning is the ability of a system to experience change based on a certain experience with an input such that it can perform an activity better and more efficiently next time. Since ASK had the ability to be taught new concepts by the user during conversation with the user, it can be said that ASK was a learning system.

D. Team (1987)[7]

TEAM was an NLIDB whose developers concerned themselves with portability issues, as they wanted it to be easily implementable on a wide range of systems without compatibility issues. It was designed to be easily configured by database administrators with no knowledge of NLIDB. These feet affected the functionality of TEAM.

E. Precise (2004)

PRECISE introduced the concept of Semantically Tractable Sentences which are sentences whose semantic interpretation is done by the analysis of some dictionaries and semantic constraints.

It was developed by Ana-Maria Popescu, Alexander Yates, David Ko, Oren Etzioni, and Alex Armanasu in 2004 at the University of Washington [8].

When a natural language query is given to PRECISE, it takes the keywords in the sentence of the query, and matches the keywords to corresponding database structures. This, in fact is the major strength of PRECISE. PRECISE does this matching in two stages. The first is to narrow down the possible keywords using the Maximum Flow algorithm which finds a feasible, constraint-satisfying flow through a Flow Network having just a single source and a single sink, such that the flow is maximum; where a flow network is a directed graph in which each edge has a capacity and each edge receives a flow. By using the Maximum Flow algorithm, the maximum number of keywords is obtained, thereby increasing the chance of the natural language sentence to be accurately transformed to a formal SQL query as there will be enough keywords to compare with the PRECISE dictionary. The second stage is to analyse the syntactic structure of the sentence. PRECISE also has its own limitations.

Generally, some major flaws have been common to these interfaces and their ability to handle natural language processing. Users' feedback system has not been thoroughly handled in existing systems. Such systems learn when the user prompts command such as save text on the interface. This is worsened by the fact that, though they are considered as a NLI, their knowledgebase has been a concern in recent times such that can only get results that keyword based. The area of natural language that can be handled by NLIDBs is just

a small subset, and this subset is even difficult to define due to Natural language complexity and the existence of ambiguity.

IV. Overview Of The Proposed System

In Enikuomehin et al[9], an NLI for RDB was developed. In that work, the system developed was named LANLI where a set of operations is defined on a Local Appropriator. The local appropriator allows for both semantic and syntactic tree generation for query execution. The highlight of the proposition is that the matching algorithm would have been generated before the query formulating tree is used. The advantage is in the area of effective retrieval due to accurate tree formation for both the database and the query. An additional feature of the system is the use of a knowledge dictionary like table where the Natural Language presented by users is assumed to have some knowledge interpretation. Same is similar to the work of NIHALIA et al[10] where an interface was designed on plain relational database. The common factor in the above schedule is that they both operate as a query executor that does not require any formal syntactic presentation. In the implementation of the proposed algorithm, the following process is undergone:

- ✓ User's natural language queries are accepted as input to a given natural language interface.
- ✓ Data-transformation of the natural language query into a formal SQL query is performed by an underlying program without the knowledge of the user.
- ✓ The SQL query is then delivered to the relational database.
- ✓ The result of the query produced by the database is accepted and transformed back into expressions in the user's natural language by the underlying program.(this is the reverse operation of 3 define above).
- ✓ This transformed result is then displayed to the user as output.

The system can be integrated into the module of existing commercial systems. The steps outlined above are necessary for an efficient operation of an NLIDB. For experimental purposes, the lecturer- course database of the department of computer science is used a case study for the implementation of the algorithm. The system is a combination of a database and set of tables resident in it. This work introduces the use of corpus in areas other than the strict information retrieval domains. The execution process can be classed into phases and presented as follows:

A. Input To The Natural Language Interface

To use an NLIDB, there must be a point of interaction between the user and the system. This point of interaction must be able to accept data (query in this case) in a form expressed in the natural language of the user, and it must be able to produce output in the same natural language format.

Because it is a point through which users can communicate with the system using their natural language, it is therefore called a Natural Language Interface. It should be noted that for the purpose of this paper, the natural language used is English Language. Thus the Natural Language Interface to be used in this work is one that accepts English Language queries as input.

B. Transformation Of Natural Language Query To Sql

Natural language is the language used for communication by humans. This language is immediately understood intuitively by humans without any further interpretation. However, to carry on conversation with any component of the computer system such as a database, one must make use of some formal language which requires some special kind of rigorous learning process for anyone to have a mastery over it. Expressions in this rather artificial language must conform to some unambiguous syntactic rules for there to be a meaningful conversation between the human and the computer system. Interaction with a database requires the use of a formal language, whose expressions, unlike natural language expressions, contain no ambiguities. Several Database Management Systems DBMS) have their corresponding language used to interact with them. The database used in this project is the Relational Database. To interact with a relational database, the language to be used is Structured Query Language (SQL). Since the natural language interface collects natural language expressions as input, this input has to be converted to a corresponding SQL expression before the database could understand the query of the user. Therefore, there must exist a program or application whose job is to retrieve the natural language input from the Natural Language Interface, and do some transformation works on it to convert it to an equivalent SQL query.

This application should be able to split the natural language query into its constituent tokens, and through comparisons with the contents of the corpus, it should be able to single out keywords in the statement. With the use of these keywords, and the use of a knowledge base (If-Then knowledge base as used in this paper), the user's query should be able to be parsed semantically, enabling the formulation of a corresponding SQL query which will then be passed to the database. The use of a knowledge base implies that the system will be domain dependent, thus it has to be reconfigured for any new database system on which it is implemented. The SQL query resulting from the transformation performed on the natural language query will have to be passed from the application to the database system itself. This transfer is possible if there is an interface between the application and the database system. This interface is usually inbuilt as a class or subroutine in many programming languages. Thus the language used for the application must possess the capability to connect to the database. After processing the SQL query, the RDBMS returns a result, this result set, occurring in less-human-understandable format, should be manipulated, to enable presentation in a natural language format. This is done by the intermediate application program between the interface and the database. In the human-readable format, the results are then ready to be presented to the user.

V. Design And Structure Of The Proposed Nlidb

Five steps are taken in the use an NLIDB and are described below:

A. *User's Natural Language Queries Are Accepted As Input To The Natural Language Interface:*

The interface actually is the first thing a user should encounter. Then the user gets started with the system by entering a query in his/her natural language

B. *A Data-Transformation Of The Natural Language Query Into A Formal Sql Query Is Performed By An Underlying Program:*

In this stage, it will be observed that the natural language query of the user is fed into the underlying application program, which in turn transforms the user's natural language query into an appropriate SQL query. Thus, there exist an interface between the Natural Language Interface and the underlying application program. This interface is responsible for presenting the natural language query from the user to the application. This interface is for the sake of this project called **NL-Application Program Interface (NLAPI)**.

C. *The Sql Query Is Then Delivered To The Relational Database:*

After the transformation of natural language query into Structured Query Language, the application program having first established a connection to the relational database, will now transfers the SQL query to the RDBMS. The connection established between the application program and the database is made possible by the help of another interface called **Application Program-Database Interface (APDI)**. This interface does the presentation of the corresponding SQL query produced by the application program to the RDBMS.

D. *The Result Of The Query Produced By The Database Is Accepted And Transformed Back Into Expressions In The User's Natural Language By The Underlying Program:*

This process is performed by the application program. The application program receives the result of the SQL query, and transforms it back into a form easily understandable by a human user.

E. *This Transformed Result Is Then Displayed To The User As Output:*

The interface here can be viewed as a reverse automated machine that displays the output of the search process. This makes the entire database search a cycle-like process.

VI. An Algorithm For Implemetation

The first thing to be done with a user's query, is to tokenize the words in the user's queries into the words found in the corpus and the requests tables of the database. This tokenization of words is done in such a way that erroneous repetitions are eliminated. The algorithm for the execution is given as:

```
query=the user's query ;
tok=getTheFirstToken(query) ;
 i=0; j=0;
while (tokenStillExists(query)) {
        if(existsInCorpus(tok)){
                toUse[i]=tok;//This array contains
words found in the user's query and also in the
corpus
                i++;
        }
        if(existsInRequests(tok)){
                reqWord[j]=tok;
        term[j]=TColumnInRequest(tok);
                //TColumnInRequest(tok) is
                the value in the t column of
                requests table for the word
                r=tok
                j++;
        }
        tok= nextToken(query);
}//End of while loop
removeDuplicate(toUse);//Removes duplicates
from the user's query
removeDuplicate(reqWord);//Removes
duplicates from the array of non-entity-reference
terms in the array reqWord[].
removeDuplicate(term);//Removes duplicates from
the array of requested data in the array term[].
```

Now that the user's query have been tokenized and separated into different sets. It must be noted that the user's query now tokenized into the array to Use can contain a combination of general and specific words.

The general words in the array to Use is thus stored in the array G and the specific words in toUse are stored in the array S. This results in four different cases for which the execution of the query differ. These cases are:

sizeOf(G)==0 and sizeOf(S)==0

sizeOf(G)==0 and sizeOf(S)!=0

sizeOf(G)!=0 and sizeOf(S)==0

sizeOf(G)!=0 and sizeOf(S)!=0

A knowledge base is created that caters for any one of the above situations; however, a brief discussion is given here to demonstrate what happens in any of the cases.

1. In the case where sizeOf(G)==0 and sizeOf(S)==0, that is, there are no general and specific words in the arrays G and S, this means that the query of the user does not contain any word in the corpus, thus the query is invalid. This message will be shown to the user.

2. In the case where sizeOf(G)==0 and sizeOf(S)!=0, that is, there are no general words but there are specific words in the query of the user, then two cases arise from this:

sizeOf(reqWord)==0

sizeOf(reqWord)!=0

In the case where sizeOf(reqWord)==0, there exists non-entity-reference words in the user's query, this would lead to the production of an SQL query that selects only the data requested by the user from the csc table, else, a general collection of data is selected from csc table for the data item(s) in the set of specific words S.

In fact, for any of the remaining cases:

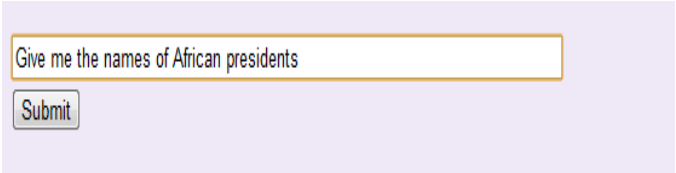
sizeOf(G)!=0 and sizeOf(S)==0

sizeOf(G)!=0 and sizeOf(S)!=0

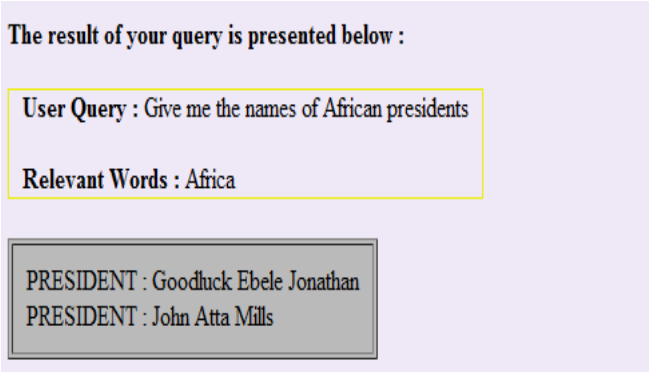
it is tested whether sizeOf(reqWord)==0 or sizeOf(reqWord)!=0, and the codes of the knowledge base found in the intermediate application program does the necessary operations using techniques in both syntactic and semantic parsing to transform the user's query into a corresponding SQL query.

VII. Implementation And Results

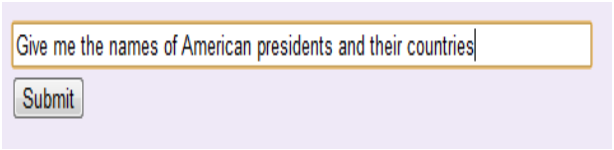
The proposed system is implemented as a web-based application. Thus the languages used include HTML, CSS, JAVASCRIPT, PHP,SQL, while the database used is MySQL as stated earlier. The system answers most of the questions posed to it by the user in natural language. The system enables a user to get information about subject of interest by typing the text in its natural language form. Below is a snapshot of some search carried out to test the performance of the result. a student or lecturer by just typing the latter's phone number or any identifying data for that matter. The snapshots below show some correct inputs and their associated results for the testing on correct inputs.



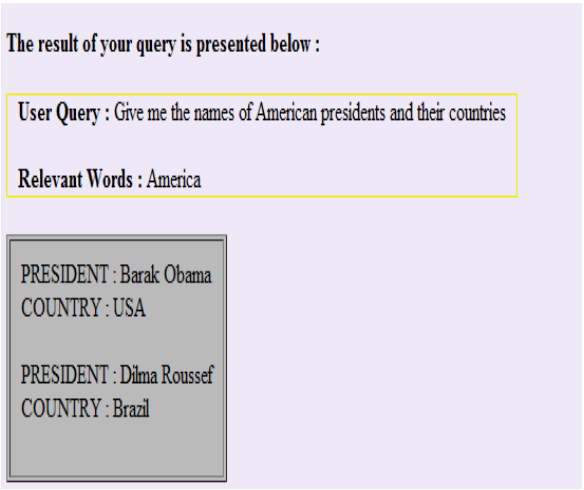
Query 1



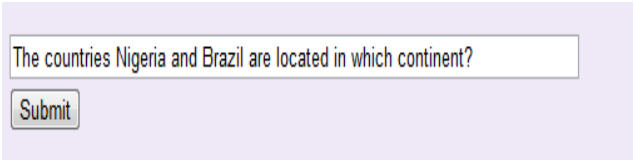
Result 1



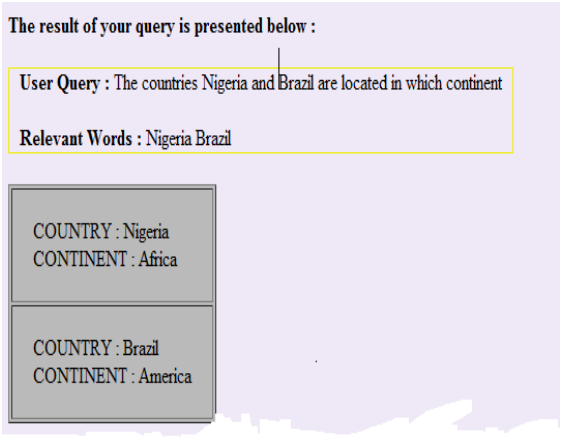
Query 2



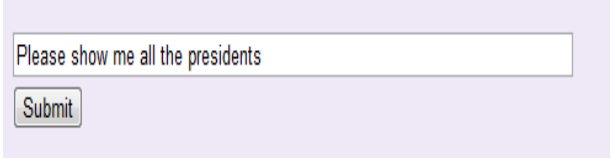
Result 2



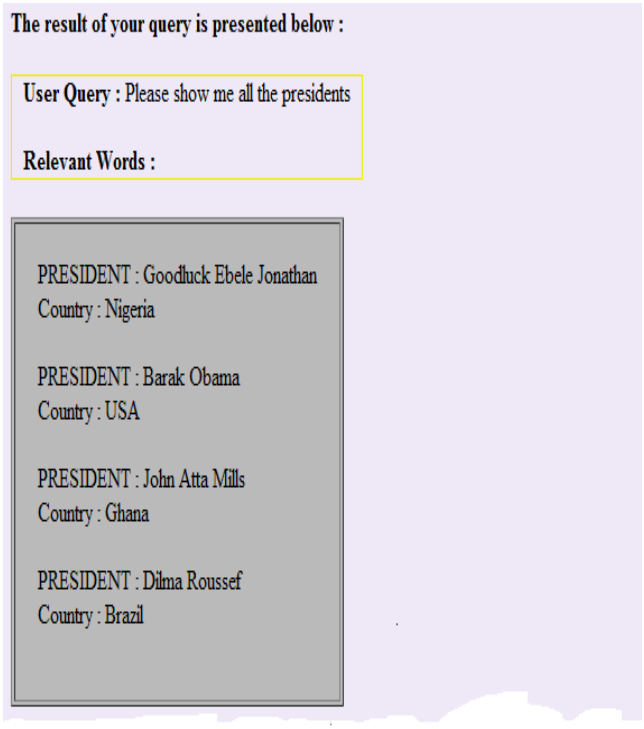
Query 3



Result 3

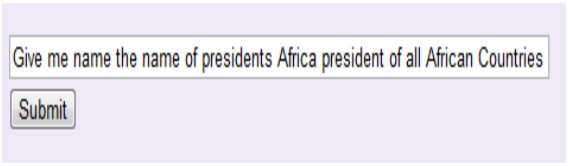


Query4

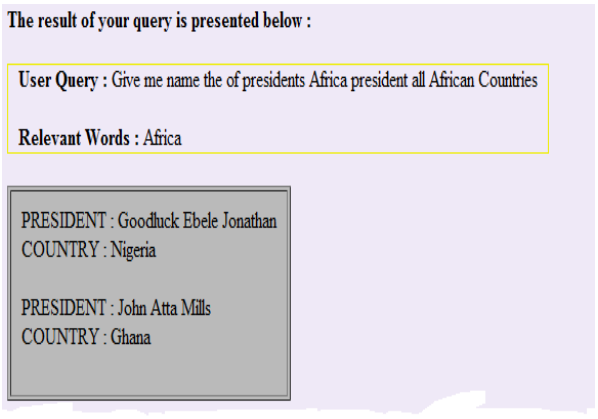


Result 4

The following snapshots below show how cases of grammatically incorrect queries are handled. These errors might arise from the fact that the user has poor grammatical abilities when it comes to the use of English Language [10], or the user forgets that a particular word had been typed once earlier, and then proceeds to type it again.



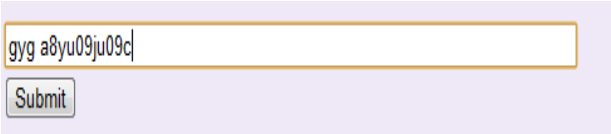
Query 5



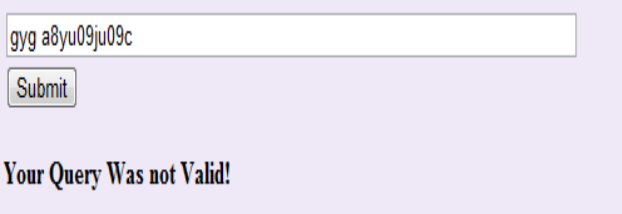
Result 5

In a case where a user just enters an arbitrarily random query, that makes no sense whatsoever in the English Language, the NLIDB should not crash, rather, it should neatly handle this error and show an appropriate message flagging off that error.

The snapshot below shows the result of this idea.



Query 6



Result 6

VIII. Discussions

The flexibility of the Natural Language Interface for Relational Databases is of great importance since it is almost unavoidable for users to make either typographical errors or input out-rightly wrong queries altogether.

A flexible NLIDB should be able to get along somehow with these errors as neatly as possible. This means that there shouldn't be any query whatsoever that could crash the NLIDB.

Flexibility of an NLIDB also makes the computer appear intelligent. This is the main goal of the field of Artificial Intelligence, as a branch of Artificial Intelligence, Natural Language Processing (NLP), and attempts to make human-computer interaction as easy as possible *[4]*. From the experimental results presented above, it is clear that the developed NLIDB is indeed flexible as intended.

It is this flexibility that this project seeks to accomplish and experimentation with random queries have yielded a very high efficient performance rate. The developed NLIDB has its own limitations. These limitations include the following:

- Domain Dependence: The NLIDB is meant to be implemented on a particular Relational Database domain, if it is to be moved to another RDBMS domain, it will have to be reconfigured for that domain. This is one limitation.
- Selective Query Domain: The NLIDB does not answer ALL the questions users may have about different countries of the world. For example, questions on civil issues of different countries will not be answered, as they are beyond the scope of the NLIDB, albeit, such questions can be answered if

words describing such civil issues are included in the corpus.

Despite these limitations, the developed NLIDB have proven to have a high performance rate when it comes to the queries posed to it from its query domain.

The limitations of the developed NLIDB, as stated earlier are as follows:

- Domain Dependent (the goal of most researchers is to design a domain independent NLIDB)[11].
- Limited on Query Domain

However, despite these limitations, the developed NLIDB have proven to have a high performance rate when it comes to the queries posed to it from its query domain, as demonstrated in the previous section on implementation and testing, by experimentations with random selection of queries.

IX. CONCLUSION

Natural language has been successfully to perform a full knowledge based semantically conscious search on relational database. This is the intent of this work. The paper showed how a modelled algorithm can be used to create a user friend non expert search process. The modularity of sql conversion was also shown.

Proposal was implemented on a departmental database however the interest in this work is not the size of the corpus but the time of execution of any unit query. Our proposed model has been able to intelligently process users request in a reasonable human useable format. The implemented result shows that the time is considerable better than earlier propositions and shall thus be upheld.

The research in this area is still ongoing and many interesting additions will be made in the future especially in the area of uncertainty in user information request definition.

REFERENCES

[1] P. Reis, N. Mamede, and J. Matias, "Edite - A Natural Language Interface to Databases: a New Dimension for an Old Approach", Proceeding of the Fourth International Conference on Information and Communication Technology in Tourism, Edinburgh, Scotland, 1997.

[2] M. Minock, "A phrasal approach to natural language access over relational databases", proceedings of the 10th International Conference on Applications of Natural Language to Information Systems, Alicante, Spain, 2005, pp. 333-336

[3] Adam: Student Debt Advisor, Convagent Ltd, Manchester, UK, 2001, Available at: http://www.convagent.com/convagent/adam3.aspx

[4] W. Woods, R. Kaplan, and B. Webber, "The Lunar Sciences Natural Language Information System", Final Report, Technical Report 2378, Bolt Beranek and Newman Inc., 1972

[5] R.J.H., Scha., "Philips Question Answering System PHILIQA1", In SIGART Newsletter, no.61. ACM, New York, (February 1977)

[6] W. Wahlster, H. Marburger, A. Jameson, and S. Busemann. Over-answering Yes-No Questions: Extended Responses in a NL Interface to a Vision System. In: Proc. of the 8th IJCAI, pp. 643–646, Karlsruhe, FRG, 1983

[7] B.J. Grosz, "TEAM: A Transportable Natural-Language Interface System", In Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California, (1983), pp 39–45

[8] A.M. Popescu, O. E. , and H. Kautz," Towards a Theory of Natural Language Interfaces to Databases " University of Washington Computer Science Seattle, WA 98195, USA.

[9] A. Enikuomehin, J.Sadiku, A new Architecture for NLIDB Using Local Appropriator Engine for SQL Generation, International journal of Advance research in computer science, 2012.

[10] N Nihalani et al ," An Intelligent Interface for Relational Databases", IJSSST, Vol. 11, No. 1, ISSN: 1473-804x online, 1473-8031 print, p30.

[11] R. Ahmad "Efficient Transformation of a Natural Language Query to SQL for Urdu", Proceedings of the Conference on Language & Technology 2009, p53.