AUTOMATED SQL QUERY GENERATOR BY UNDERSTANDING A NATURAL LANGUAGE STATEMENT

Amit Pagrut, Ishant Pakmode, Shambhoo Kariya, Vibhavari Kamble and Yashodhara Haribhakta

Department of Computer Engineering and Information Technology College of Engineering Pune Wellesley Road, Shivajinagar, Pune, Maharashtra, India

ABSTRACT

This project aims to develop a system which converts a natural language statement into MySQL query to retrieve information from respective database. The system mainly focuses on creation of complex queries which includes nested queries with more than two-level depth, queries with aggregate functions, having clause, group by clause and co-related queries which are formed due to constraint on aggregate function. The natural language input statement taken from the user is passed through various OpenNLP natural language processing techniques like Tokenization, Parts of Speech Tagging, Stemming and Lemmatization to get the statement in the desired form. The statement is further processed to extract the type of query, the basic clause, which specifies the required entities from the database and the condition clause, which specifies constraints on the basic clause. The final query is generated by converting the basic and condition clauses to their query form and then concatenating the condition query to the basic query. Currently, the system works only with MySQL database.

1. INTRODUCTION

Almost all applications in today's world make use of collected data to fulfill the intended requirements and to enhance their functionalities. The main objective remains the efficient storage and fast retrieval of this data. Databases provide a better provision and are the most suitable solution which addresses these objectives. The relational databases provide a structured way to store the huge collection of data and provide real-time accessibility. Relational database management system is representation of domain entities and their respective attributes in the tabular form.

Many organizations and social networking sites make use of relational databases for storage and analysis. In this modern world, everyone aims to be more dynamic in terms of information gathering, retrieval and sharing using existing systems of database storage, but are not enough well versed with the underlying technology. Users need to learn the underlying database language and database properties to generate queries. Natural language is used in day-to-day life, and if information sharing can be made easy with the use of natural language, it will reduce the cost of learning and understanding the technology used for it.

NLIDB(Natural Language Interface to Database Systems) is the provision under which the natural language is used to interact with databases. There are many existing NLIDB systems which allows user to work with databases using their own languages, although the research on these systems started a few decades ago, there is still no perfect system that fulfills all the objectives of NLIDB.

This system is a type of NLIDB system which concentrates on formulating complex queries, along with simple queries. In this project, we take the user natural language query input in English language through the graphical user interface. In Java, OpenNLP library provides various natural language processing modules. The input is then processed through various Natural language Processing processes like tokenizing the sentence by the space delimiter, mapping each token with its Parts Of Speech(POS) tag using POS Tagger, lemmatizing each token to convert it into its basic form, and these lemmas and tags are stored. The converted statement is now broken

DOI: 10.5121/ijnlc.2018.7301

down into two parts : Basic clause and Condition clause. The Basic clause identifies the attributes used to form the query and the Condition clause identifies the constraints on these attributes and are mapped to their respective attribute. A Table Linking Algorithm(TLA) is incorporated to find the links between different tables of the query. The basic query and condition query is then joined together to form the final resultant query.

2. RELATED WORK

[1] Lunar Involved a system that answered questions about Rock Samples brought back from the moon. Two databases were used, the chemical analyses and literature references. **Lifer/Ladder,** it was designed as a natural language interface to database of information about US Navy ships.

It used a semantic grammar to parse Questions and query a distributed database.**EasyAsk2**, Also known as English wizard, is a commercial application that offers both keyword and natural language search over relational databases.**EQ**, Stands for English query. It was implemented by Microsoft. It creates a model, collecting database objects (tables, fields, joins) and semantic objects (Entities).

(Siasar et al., May 2008) [2] gave an insight on how the machine understands natural language. They proposed an expert system taking into account the concepts of syntactic and semantic knowl- edge. They also suggested a selection algorithm to select most appropriate query from the sug- gested possible query.

Mrs. Neelu Nihalani , Dr. Sanjay Silakari , Dr. Mahesh Motwani, March 2011[is an introduction to Intelligent Database System and Natural Language Interface to databases. It gives a brief overview of NLIDB subcomponents and also explains NLIDB architectures and various approaches for the development of NLIDB systems.

Akshay G. Satav, Archana B. Ausekar, Radhika M. Bihani, Mr Abid Shaikh, March 2014 [3] provides result to users for any type of query accurately and efficiently, even if any user make spelling mistake the system will autocorrect the spelling and experimental result will be shown. They used a query mapping algorithm where query of any form in English language mapped according to syntax of SQL query that provides user accurate data from the database after execution of the query.

Alessandra Giordani and Alessandro Moschitti **[4]**used linguistic dependencies and metadata to build sets of possible SELECT and WHERE clauses and then exploit again the metadata to build FROM clause enriched with meaningful joins. Finally, combining all the clauses they got the set of all possible SQL queries. The algorithm can be recursively applied to deal with complex questions, requiring nested SELECT instructions.

(Garima Singh, Arun Solanki, September 2016) [5]combined Artificial Intelligence (AI) and Linguistics to develop programs helped to understand and produce information in a natural language. The NLIDBS are built for to optimize the search results and produce information with more accuracy. This paper extends the existing work further by processing more complex queries along with ambiguity removal.

K. Javubar Sathick *and* A. Jaya, *2014* **[6]**provides an user friendly interface between end user and the database for easy access of social web data from different web sources such as facebook, twitter and linkedIn etc. by deriving a query translator which take natural language statement as input.R-tool is used to collect the data from social web sources.

Abhijeet Gupta and Rajeev Sangal [7] In this paper, they introduced an aggregation processing framework, which can handle different types of aggregation operations in a natural language query, including direct quantitative as well as indirect qualitative aggregations, and those which combine quantifiers or relational operators with aggregations.

(Hu Li and Yong Shi, 2010) [8]This paper presents the framework based on ontology techniques to implement a portable NLIDBs that makes it easier to migrate from one domain to another.

(Ashish Palakurthi, Ruthu S. M, Arjun R. Akula and Radhika Mamidi, September 2015) [9] uses a statistical classifier trained on manually prepared data. They report their results on three different domains also shows how the system can be used for generating a complete SQL query. They also used Conditional Random Fields (CRF) for classifying the explicit attributes in a natural language query to different SQL clauses.

(Abhijeet R. Sontakke, Amit Pimpalkar, July 2015) [10] have developed the rule based system which accepts Hindi language as query and gives output in Hindi languageonly.

(Nandan Sukthankar, Pranay Deshmukh 2016) [11]The research focuses on incorporating complex queries along with simple queries irrespective of the database.The system accommodates aggregate functions, multiple conditions in WHERE clause, advanced clauses like
ORDER BY, GROUP BY and HAVING. The system handles single sentence natural language inputs, which are with respect to selected database.

3. OUR APPROACH

3.1. SYSTEM ARCHITECTURAL LAYOUT

Figure 1.0 gives the simple architecture of the system. It consists of tokenization, lemmatization of tokens to convert them into their basic forms called lemmas, parts of speech tagging of each lemma, using OpenNLP APIs. The input natural language statement goes through these above mentioned steps and forms parsed data. Each keyword in the parsed data is processed through semantic analysis and relation mapping which makes use of data dictionaries (contains database schema information like synonyms of SQL clause words, aggregation words, database attributes and table names). The synonyms are found using WordNet data dictionary in NLTK.

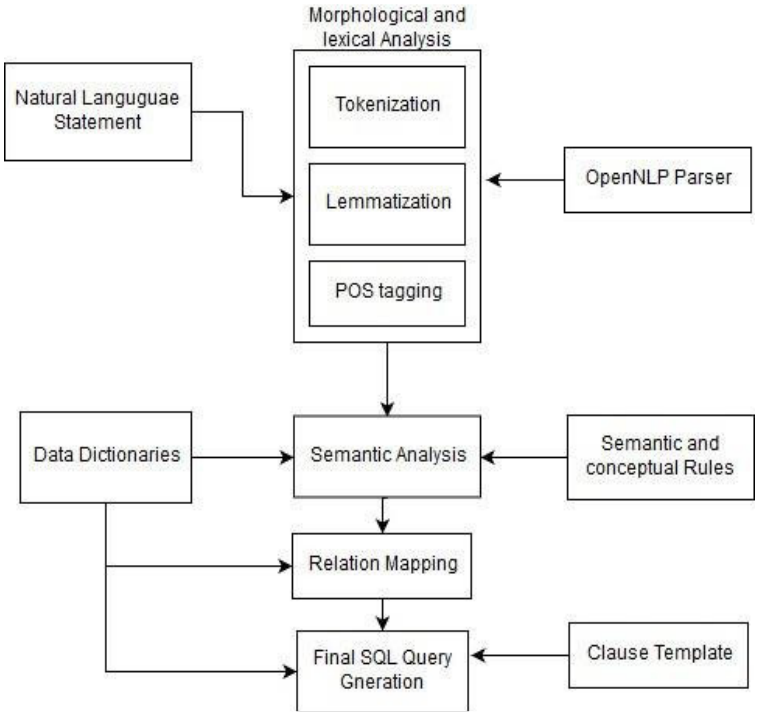

Figure 1. System Architecture

The processed data is then used to generate final MySQL query by using semantic conceptual rules and clause templates(query templates e.g. SELECT_FROM_WHERE etc.)

3.2 MORPHOLOGICAL AND LEXICAL ANALYSIS

3.2.1 Tokenization

Tokenization refers to the breakdown of the natural language sentence into tokens with _white space‘ as delimiter. This step is required to convert each token into its root form in the next step.

3.2.2 Lemmatization

Lemmatization is the process of converting a word into its root form. The tokens from the earlier step are converted into their respective root form e.g (‘employees‘ to ‘employee‘, ‘saw‘ to ‘see‘) to extract their actual meaning so as to incorporate the processing algorithms on them. These converted tokens are calledlemmas.

3.2.3 PARTS OF SPEECH (POS) TAGGING

The lemmas are mapped to their Parts Of Speech tags so as to derive their semantic meaning for further processing using the Parts of Speech tags. For example, the Parts of Speech tags are used to find the nouns and verbs in the input sentence, and further help to find the tables and attributes in the query, considering the fact that table and attribute names are nouns and verbs.

Ex.1. Find the names of employees with salary greater than 50000. Here,

Lemmas	Parts of Speech tags
show	VBZ
the	DT
name	NN
of	IN
employee	NN
with	IN
salary	NN
great	JJ
than	IN
50000	CD

3.3CONDITION CLAUSE EXTRACTION

3.3.1 Determination of attribute values

The attribute values are derived from the user query with the help of the Parts of Speech tags provided by lexical analysis stage. The Parts of Speech tags are checked for tags like ―NNP‖ for string values, as attribute values in the database are usually proper nouns and ―CD‖ for integer values. Ex.2. Show the names of employees whose salaries are greater than 50000 and department is Computer Science. Here, 50000 is CD and ‘Computer Science‘ is NNP.

3.3.2 Mapping relational operators to attributevalues

The attribute values are related to their respective attribute through the relational operators like equal to, is greater than, etc. These operators define the condition or the constraint on the attribute, and would contribute in making the query more descriptive. The operator symbol for each operator is stored in advance. The mapping is stored as => RELATIONAL OPERATOR — OPERATOR SYMBOL — ATTRIBUTE VALUE. In Ex.2, 50000 is mapped to '>' and 'Computer Science' is mapped to '='.

3.3.3 Mapping of attribute values to the respective attributes

The above extracted attribute values are linked to their respective attributes using the attribute mapping algorithm, which searches for the most probable attribute to be mapped to the value. The attribute names are replaced by their original names in the database. This mapping is stored as => ATTRIBUTE TABLE - ATTRIBUTE NAME - OPERATOR SYMBOL - ATTRIBUTE VALUE.
In Ex.2, the mapping is => *{*salary, salary, >, 50000*}*, *{*department, depart name, =, 'Computer Science'*}*

3.3.4 Check for aggregation function on theattribute

The query is processed to check if the mapped attributes have an aggregation function applied on them, by extracting the synonyms of the aggregation words(MINIMUM, MAXIMUM, COUNT, AVERAGE, SUM) and then mapping the actual aggregation function word with the attribute.

3.3.5 Removing the condition clause part from theQuery

The condition clause part is removed from the query to obtain the basic clause required to process the remaining part of the SQL query. In Ex.2, the remaining part of the query is => show the name of employee.

3.4 BASIC CLAUSE EXTRACTION

The basic clause will determine the ―SELECT‖ part of the SQL query. The select clause of the SQL query contains the names of the attributes, which are to be extracted from the user query. After the extraction and removal of the condition clause part from the user query, the basic clause part is evaluated and the required entities are mapped and stored. The execution of the basic query part will be done using of the following algorithms:

3.4.1 Table Linking Algorithm(TLA)

In TLA, the tables present in the database schema are represented as a data structure entity and has following properties:

- tableName
- allAttributes
- foreignKeys
- primaryKey

The main focus of the algorithm is to find a path between two referenced tables, the source table and the destination table.

The algorithm proceeds in the following fashion:

- First, the data structure references of the source and destination tables are retrieved. All the foreign keys present in the source table are obtained. Now, for each foreign key, the key's

referencing table is linked to the source table, with the use of the foreign key reference and the fact that the attribute would be present in both tables, this path is appended to the available paths. The referencing table now becomes the source table and now, source and destination tables are fed to the algorithm to find a path between them, making the algorithm recursive. If the referencing table at some point matches the destination table, the path till now plus the path appended between the current source and destination table becomes the final path.

- If the source table doesn't contain any foreign key attributes, the source and destination tables are swapped, and then they are passed again to the algorithm, but with an indicator that the tables have been swapped. If the indicator indicates that the tables are swapped, the path between those tables is reversed to obtain the correct path and then appended to the available paths.
- If the source and destination tables don't contain any foreign key attributes, a common table linking both the source and destination table is found out, by considering the fact that the common table would contain at least one common attribute of both tables. Now the path between source table and common table is first appended and the path between the common table and the destination table is appended later and this path is appended to the available paths. Out of the found common tables, only one has the actual semantic of both the joining tables and only the query generated through this common table is correct. Ex.3, Find employees in Human Resource department, While finding path between employee and department tables, depart emp and depart manager are tables which has foreign keys of both. So, two paths will be generated by system, but by semantic only path with depart emp is correct.

3.4.2 NATURAL JOIN Algorithm

NATURAL JOIN is used in the FROM clause when the SELECT clause contains attributes which are to be extracted from different tables. Ex.3 Show salary and names of employee. Here, attribute 'salary' is from 'salary' table and 'name' is from 'employee' table. The tables to be joined using NATURAL JOIN are tables of the attributes present in the SELECT clause as well as the tables of the attributes present in the WHERE clause. The WHERE clause tables are retrieved from the Condition Clause Extraction stage, whereas the tables of the SELECT clause are found out and the link between these tables is to be identified using the Table Linking Algorithm, in the following manner:

- The tables of the SELECT clause are linked with each other by passing them one by one in the Table Linking Algorithm and finding their intermediate tables in the path joining the tables. The tables obtained are stored in a set.
- Now, each table from the SELECT clause is to be linked with each table of the WHERE clause, in the similar above fashion, and the tables identified are appended to the set.
- Now, the resultant tables in the set are used to form the FROM clause of the SQL query, using the 'NATURAL JOIN' keyword between them.

3.5 THE AGGREGATION FUNCTION EXTRACTION

The aggregated functions provide the constraints on the result set of the SQL query. The aggregate functions are handled by an RDBMS independent layer called Aggregation Extraction Layer (AEL). The steps in AEL are:

- The synonyms of the aggregated words are extracted from the user query and are mapped with their actual aggregation function.
- The query is then processed to relate the aggregate words with their respective attributes and stored.

- The query is further processed to find if there is any constraint on the aggregation function. If present, the constraint is filtered and mapped to the corresponding aggregate function.
- The final mapping is stored as => [Constraint on Aggregation, Aggregation function, Attribute name].
- The GROUP BY clause is then formulated by taking the attributes other than the aggregate attribute from the SELECT clause (if any).

3.6. CORRELATED QUERY EXTRACTION

Correlated query is formed whenever there is a constraint on the aggregation functions like MAX or MIN. The correlated query is formed by deriving the constraint value mapped to the attribute and formulating the SQL query in the below givenform:

SELECT T1.attribute FROM table AS T1 where (N - 1) = (SELECT COUNT (DISTINCT T2.attribute) FROM table AS T2 WHERE T1.attribute <operator> T2.attribute)
Where, N => constraint value of the aggregation function

<operator> => depends on the type of aggregation function (Minimum or Maximum)

3.7 LIMIT AND ORDER BY CLAUSES

The LIMIT clause is formed when there is a constraint on the number of records to be shown in the result set of the SQL query. The query is processed to find LIMIT value to which the LIMIT clause word is mapped and stored.

The ORDER BY clause is formed when there is need of sorting the records of the result set in ascending or descending order according to one or more attributes. The ORDER BY clause is formulated by extracting the synonyms of the clause words and then mapping them to the actual ORDER BY clause words like ASC or DESC, for ascending and descending respectively.

4. RESULTS

We have tested our system on a synthesized corpus of natural language statements related to employee database. Employee Database contains 6 tables. The system has tested with around 50 queries with single sentence natural language inputs. The accuracy comes out be 82%. Fallowing are the example of inputs given to the system

4.1 SYSTEM OUTPUT

4.1.1 Simple Query

Query: Find the salary of employee whose id is 4

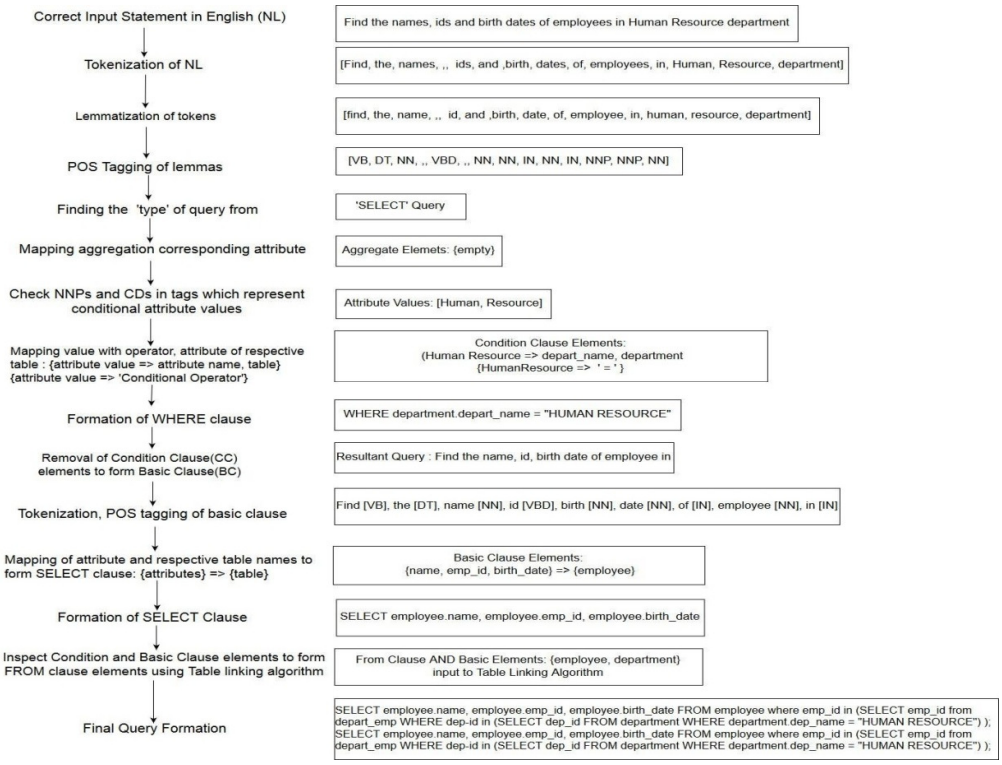

Figure 2. Flow of execution

Figure 2 shows the implementation steps by which the natural language statement is converted into Final MySQL query.

In this query, ‘salary‘ and ‘employee‘ tables are linked together using the Table Linking Algorithm. Here, the WHERE clause is formed by using the integer value ‘4‘.

Output: SELECT salary.salary FROM salary WHERE emp _id IN (SELECT emp id FROM employee WHERE employee.emp_id = 4)

Query: Show the department manager with department name Techonology

In this query, ‘depart manager‘ and ‘department‘ tables are joined together using Table Linking Algorithm. Here, the WHERE clause is formed using literal value ‘TECHNOLOGY‘.

Output: SELECT * FROM depart manager WHERE dep_id IN
(SELECT dep_id FROM department WHERE department.dep_name = ‘TECHONOLOGY‘)
Query: Show all employees
Output: SELECT * FROM employee

4.1.2 Queries with same semantics

Query 1: Show the salaries of Human Resource department employees

Query 2: Determine the salaries of employees whose department is Human Resource Query 3: Choose the salaries of Human Resource department

Query 4: Find the salaries of employees if they are in Human Re- source department

The system is able to handle the queries with same semantics up to some level. The above queries have same semantics. Therefore, they form the same result query. One out of the two queries

given by the system is correct due to the fact given in theTable Linking Algorithm.

Output 1: SELECT salary.salary FROM salary WHERE emp_id IN (SELECT emp id FROM employee WHERE emp _id IN (SELECT emp_id FROM depart_manager WHERE dep_id IN (SELECT dep_id FROM department WHERE department.dep_name = 'HUMAN RESOURCE')))
Output 2: SELECT salary.salary FROM salary WHERE emp_id IN (SELECT emp _id FROM employee WHERE emp_ id IN (SE LECT emp_id FROM depart_emp WHERE dep_ id IN (SELECT dep_id FROM department WHERE department.dep_name = 'HUMAN RESOURCE')))

4.1.3 Queries with advanced condition clause

Query: Show the id of employee, birth date of employees with department name computer science and names are Amit and Alex

Output 1: SELECT employee.emp_id, employee.birth_date FROM employee NATURAL JOIN department NATURAL JOIN salary NATURAL JOIN depart_manager WHERE department.dep_name = 'COMPUTER SCIENCE' AND employee.name = 'AMIT' AND employee.name = 'ALEX
Output 2: SELECT employee.emp_id, employee.birth_date FROM employee NATURAL JOIN department NATURAL JOIN salary NATURAL JOIN depart_emp WHERE department.dep_name
= 'COMPUTER SCIENCE' AND employee.name = 'AMIT' AND employee.name = 'ALEX'

4.1.4 Queries with aggregation functions

Query 1: show the name of employee whose salary is maximum

In this query, the aggregation function 'MAX' is mapped to 'salary'. The aggregation function is in the Condition clause elements and therefore it is used to form the WHERE clause.

Output: SELECT employee.name, salary.salary FROM employee NATURAL JOIN salary WHERE salary.salary IN (SELECT MAX(salary.salary) FROM salary)

Query 2: Show the maximum salary and employee id, employee name, title of employee whose id is 4

In this query, aggregation function 'MAX' is mapped to 'salary'. But, the aggregation function is in the Basic clause elements and therefore it is used to form the SELECT clause.

Output: SELECT MAX (salary.salary), employee.emp_id, employee.name, title.title FROM salary NATURAL JOIN employee NATURAL JOIN title GROUP BY employee.emp_id, employee.name, title.title HAV- ING employee.emp_id = 4

4.1.5 Queries with BETWEEN and LIKE clause

Query Determine the birth date of employees having salary between 40000 and 50000

Output: SELECT employee.birth date FROM employee WHERE emp id in (SELECT emp id FROM salary WHERE salary.salary BETWEEN 40000 AND 50000)

Query 2: Show the title of employee where employee name starts with Sham
Output: SELECT title.title FROM title WHERE emp_id IN (SELECT emp_id FROM employee WHERE employee.name LIKE 'SHAM'

4.1.6 Correlated Queries

Query: Find the 4th maximum salary

Here, aggregation function 'MAX' will be mapped to 'salary' but aggregation has constraint on it and the query is formed according to the format given in Correlated queryExtraction.

Output: SELECT T1.salary FROM salary as T1 WHERE 3 = (SELECT COUNT(DISTINCT T2.salary) FROM salary as T2 WHERE T1.salary > T2.salary)

4.1.7 Queries with LIMIT and ORDER BY clause

Query: show the first 5 names of employee in ascending order of salary

Output: SELECT employee.name FROM employee NATURAL JOIN salary ORDER BY salary.salary ASC LIMIT 5

5. LIMITATIONS

- Queries which have another query embedded in it. e.g Find the department manager of employees whose employee id is greater than Alex employee id. Here we are unable to find correct condition clause elements as it is another query.
- Query ‖ Who is Alex's department manager‖, here Alex will be mapped to department manager but query semantic demands to find department manager of employee Alex
- Queries with qualitative quantifiers can not be processed in the system e.g. Find the youngest employee. Here, 'youngest' is the qualitative quantifier and the system is not able to map it with birth date of employee.

6. CONCLUSION AND FUTURE SCOPE

The user of the system will be able to give a natural language input statement and formulate the following types of MySQL queries:

- Generate nested queries with more than two- leveldepth.
- Formulate aggregate queries along with HAVING and GROUP BY clause.
- Generate correlated queries having constraints on aggregation function.
- Generate queries with NATURALJOIN.
- Generate queries with LIMIT and ORDER BY clauses.
- Generate queries with BETWEEN, LIKE and RANGE clauses.

The system can be further developed by incorporating the following future works, which are yet to be implemented in the present system. The development on the points mentioned in future work is in progress.

- The system can be developed to handle qualitative quantifiers in the user query.
- A recursive algorithm can be designed to handle advanced nested queries which have an independent SQL query in the condition clause.
- Machine Learning algorithms can be incorporated to determine the most efficient query amongst all the possible SQL queries for a user query.
- There can be a situation where the user query has more than one se- mantic. An algorithm can be designed to formulate a SQL query with respect to each derived semantic.
- The system uses only MySQL database. It can be broadly developed to accept other database or unstructured database systems.

REFERENCES

[1] William Woods, Ronald Kaplan, and Bonnie Webber, (01 1972) ―The lunar science natural language information system: Final report‖.F Siasar djahantighi, Mohammad Norouzifard, Seyed Hashem Davarpanah, and M. H. Shenassa, ―Using natural language processing in order to create sql queries‖, (2008).

[2] Mr Abid Shaikh Akshay Satav Archana B. Ausekar, Radhika M. Bihani, ―A proposed natural language query processing system‖, (2014).

[3] Giordani Alessandra and Moschitti Alessandro, (2012) ―Generating sql queries using natural language syntactic dependencies and metadata‖, in NLDB.

[4] Arun Solanki Garima Singh, ―An algorithm to transform natural language into sql queries for relational databases‖, (2016).

[5] Javubar Sathick K. and Jaya A., (2015) ―Natural language to sql generation for semantic knowledge extraction in social web sources‖, Indian Journal of Science and Technology, Vol. 8, No. 1.

[6] Abhijeet Gupta and Rajeev Sangal, (December 2013) ―A novel approach to aggregation pro- cessing in natural language interfaces to databases‖, in Proceedings of the 2013 International Conference on Natural Language Processing, CDAC Noida, India, International Institute of Information TechnologyHyderabad.

[7] Li Hu and Shi Yong, (Feb 2010) ―A wordnet-based natural language interface to relational databases‖, in 2010 The 2nd International Conference on Computer and Automation Engi- neering (ICCAE), Vol. 1, pp. 514–518.

[8] Ashish Palakurthi, Ruthu S. M., Arjun R. Akula, and Radhika Mamidi, ―Classification of attributes in a natural language query into different sql clauses‖,(2015).

[9] Amit Pimpalkar Abhijeet R. Sontakke, ―A rule based graphical user interface to relational database using nlp‖.

[10] Nandan Sukthankar, Sanket Maharnawar, Pranay Deshmukh, Yashodhara Haribhakta, and Vibhavari Kamble, ―nquery - a natural language statement to sql query generator‖, (2017).