nQuery - A Natural Language Statement to SQL Query Generator

Nandan Sukthankar, Sanket Maharnawar, Pranay Deshmukh
Yashodhara Haribhakta, Vibhavari Kamble
Department of Computer Engineering and Information Technology
College of Engineering Pune
Wellesley Road, Shivajinagar, Pune, Maharashtra, India
http://www.coep.org.in
{nandans16, sanketmaharnawar, pranay.s.deshmukh}@gmail.com
{ybl.comp, vvk.comp}@coep.ac.in

Abstract

In this research, an intelligent system is designed between the user and the database system which accepts natural language input and then converts it into an SQL query. 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. The research currently concentrates on MySQL database system. The natural language statement goes through various stages of Natural Language Processing like morphological, lexical, syntactic and semantic analysis resulting in SQL query formation.

1 Introduction

Today, virtually every relational database management system (RDBMS) uses Structured Query Language (SQL) for querying and maintaining the database. Users accessing relational databases need to learn SQL and build queries in the right syntax for retrieving the data. It becomes a big hurdle for all those who are not technically knowledgeable in this domain to write the queries in SQL. It would be very convenient if the relational database system can be queried using natural language like English.

Natural language processing (NLP) is the ability of a computer program to understand human speech as it is spoken. While natural language may be easy for people to learn and use, it has been proved to be hard for a computer to master. Despite such challenges, natural language processing is regarded as a promising and important endeavor in the field of computer research.

nQuery will translate natural language queries into SQL before retrieving data from database. It will deal with single sentence inputs given by the user using a particular database. The system mainly focuses on data retrieval but also provides the facility to convert DML natural language statements to SQL. However, the system will output queries which can be used for querying the MySQL database system only. The aim of the system is to reduce the complexity of database querying. The approach our system uses, extracts certain keywords from the natural language statement and goes through various steps of Natural Language Processing. This system focuses on table mapping, attribute mapping and clause tagging to generate the resultant query.

2 Related Work

Over the years, certain systems which focus only on a particular database have been built to serve a particular purpose. (Woods, 1973) developed a system called LUNAR, that answered questions about rock samples brought back from the moon. LIFER/LADDER designed by (Hendrix, 1978) was designed as a natural language interface to a database of information about US Navy ships. The system could only support simple one-table queries on a specific database.

Some of the recent developments try to build a complete system which can generate various types of queries. An expert system was proposed by (Siasar et al., May 2008) using the concepts of syntactic and semantic knowledge. They have also suggested a selection algorithm to select most appropriate query from the suggested possi-

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics- Student Research Workshop, pages 17–23
Vancouver, Canada, July 30 - August 4, 2017. ©2017 Association for Computational Linguistics
https://doi.org/10.18653/v1/P17-3004

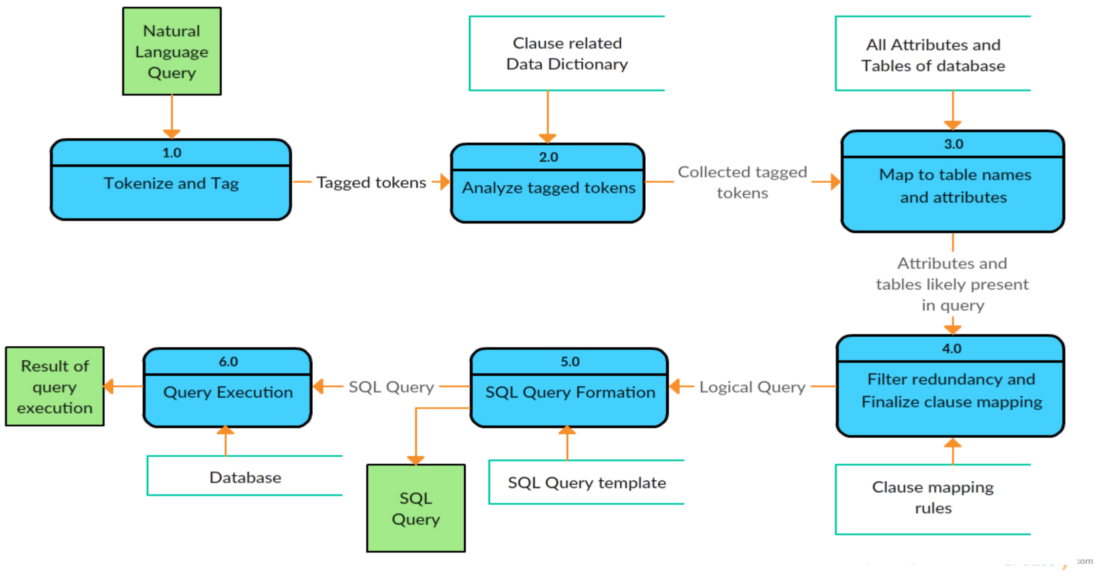


Figure 1: nQuery - Data Flow Diagram

ble query. (Rao et al., 2010) have put forth a system in which simple queries along with basic implicit queries are covered. (Chaudhari, November 2013) implemented a system which handled simple queries and aggregate functions using a prototype approach. Both the above methods have not handled multiple tables and advanced clauses. . (Ghosh et al., 2014) built on the development done by (Chaudhari, November 2013) and developed an automatic query generator which took natural language input in the form of text or speech. It provided support for simple nested queries and aggregate functions. The system handled sentences which explicitly mention the attribute names as they are in the Database. Our system handles the problem by a specific substring algorithm we have developed. We have looked to do the tasks which the above systems do in a more efficient way by building a different type of algorithm relying on conditional substring matching to map the words to attributes and tables. We also go beyond by including various different types of queries.

A different type of approach was used by (Reinaldha and Widagdo, 2014) in which the different kinds of questions which can be asked when a database is to be queried were analyzed. They have made use of semantic rules to find out dependencies among the words present in the question asked. (Palakurthi et al., 2015) provides information about the types of attributes and classification features. They describe how different kinds of attributes are handled differently when they occur in sentences. We handle explicit attributes and certain types of implicit attributes in sentences. (Ghosal et al., March 2016) proposed a system which worked well on simple queries involving multiple tables. But the data dictionaries used for the system are limited and the grammar is hard coded. (Kaur and J, Jan 2016) emphasized on simple queries and basic JOIN operations. However, the system does not accommodate advance clauses like aggregate functions, GROUP BY and HAVING Clauses. Our system incorporates advanced clauses along with all the simple queries and generalizes well on different databases. (Singh and Solanki, 2016) proposed an algorithm to convert natural language sentence to SQL queries. They used verb lists, noun lists and rules to map attributes and tables to the words in the sentence. The system also handled ambiguity among the inputs. We have tried to use concepts discussed in this algorithm like noun and verb lists in order to develop our algorithm.

From the above literature survey, we were able to get a fair idea of the work carried out in this field of research. The shortcomings of the referred papers and applications along with the future work mentioned motivated us to take up this research. The increasing importance of Natural Language Processing lured us towards learning these concepts. The system we propose looks to go beyond the accomplished work.

3 System Design

As we have seen from the literature survey, every system had limitations. We propose a system which looks to overcome the shortcomings of the existing systems. Our system gets a natural language sentence as an input, which is then passed through various phases of NLP to form the final SQL query. Refer Fig 1 for the data flow diagram and Fig 2 for the running example.

3.1 Tokenize and Tag

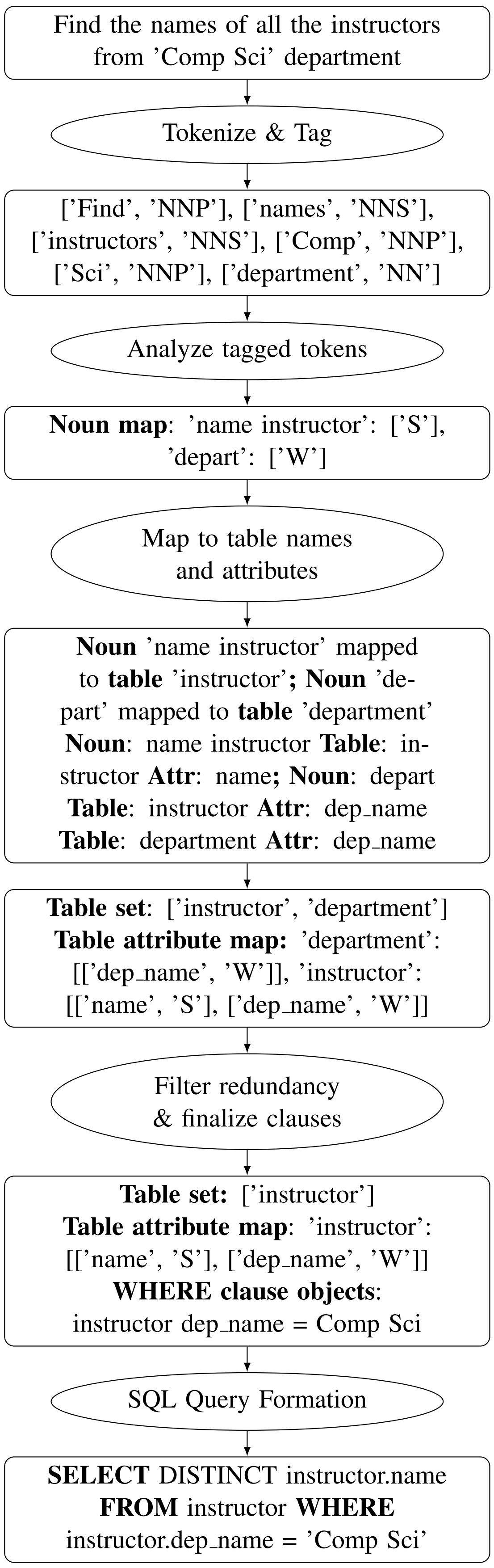
The input natural language query gets split into different tokens with the help of the tokenizer from 'NLTK' package. The tokenized array of words is tagged according to the part-of-speech tagger. All processes following this step use these tagged tokens for processing.

3.2 Analyze tagged tokens

Based on the tagged tokens of earlier step, the noun map and verb list is prepared through one iteration over the tokens. The tokens corresponding to aggregate functions are also mapped with their respective nouns. The decision whether the natural language statement represents a data retrieval query (SELECT) or a DML query (INSERT, UPDATE, DELETE) is taken at this stage with the help of certain 'data arrays' for denoting type of query. For example, when words like 'insert' and its certain synonyms appear in the input, the type of query is 'INSERT' and so on. In any type of query, the tentative tags 'S' (SELECT), 'W' (WHERE), 'O' (ORDER BY) are mapped to the nouns indicating the clauses to which they belong. For this, we have designed 'data dictionaries' for different clauses. These data dictionaries consist of the token-clause term pair, for e.g. aggregate clause data dictionary is "number": "COUNT", "count": "COUNT", "total": "SUM", "sum": "SUM", "average": "AVG", "mean": "AVG". Thus, if any of these tokens is encountered, it is likely to have aggregate clause and accordingly the nouns are tagged with the clause tag.

3.3 Map to table names and attributes

Using the noun map and verb list, the table set is prepared, which will hold the tables that are needed in the query to be formed. This is based on the fact that the table names are either nouns or verbs. The noun map is used to find the attributes which are needed in the final query. The



Find the names of all the instructors from 'Comp Sci' department
Tokenize & Tag
['Find', 'NNP'], ['names', 'NNS'], ['instructors', 'NNS'], ['Comp', 'NNP'], ['Sci', 'NNP'], ['department', 'NN']
Analyze tagged tokens
Noun map: 'name instructor': ['S'], 'depart': ['W']
Map to table names and attributes
Noun 'name instructor' mapped to table 'instructor'; Noun 'depart' mapped to table 'department' Noun: name instructor Table: instructor Attr: name; Noun: depart Table: instructor Attr: dep_name Table: department Attr: dep_name
Table set: ['instructor', 'department'] Table attribute map: 'department': [['dep_name', 'W']], 'instructor': [['name', 'S'], ['dep_name', 'W']]
Filter redundancy & finalize clauses
Table set: ['instructor'] Table attribute map: 'instructor': [['name', 'S'], ['dep_name', 'W']] WHERE clause objects: instructor dep_name = Comp Sci
SQL Query Formation
SELECT DISTINCT instructor.name FROM instructor WHERE instructor.dep_name = 'Comp Sci'

Figure 2: Algorithm with example

attributes, the table associated with the attribute and the clause tag are stored in an attribute-table map which is used in the final stage of query formation. This is done using the string matching algorithm that we have implemented in our system. The words in the input sentence need not exactly be as they are in the database. The stemmer and lemmatiser are applied on the words before they are matched using our string matching algorithm. The data obtained during this step i.e. table set and attribute-table map, is most likely to be in the final query, however, it might be refined later.

3.4 Filter redundancy and finalize clauses of the query

Using the various data dictionaries defined, the system has already decided which clauses are likely to exist in the final query and has mapped the data to the clauses. But, some of the data has to be finalized at this stage. The data related to GROUP BY and HAVING clause is collected using the previous data and the basic rules of SQL. For example, if aggregate function is compared to a constant, i.e. 'MAX(salary) > 40000', then 'HAVING' clause has to be used instead of 'WHERE' clause.

As mentioned in the earlier step, the refinement of data must be done. Here, the redundant tables and attributes are removed using some filter algorithms. For example, one of the algorithm filters the table and their corresponding attributes which are a subset of some other table in table set. i.e. if table set has [table1, table2] and table1 has attributes [a1, a2] and table2 has [a1, a2, a3] after the previous steps, then table2 is enough to represent all the attributes required and hence table1 is removed. There are various other algorithms applied in order to filter the results and finalize the table set and table-attribute map.

3.5 Form the final query and execute

Currently, as our system handles only MySQL queries, the templates used for the query formation will be according to the MySQL syntax. According to the type of query selected in the second stage of the process (Analyze tagged tokens), the appropriate template is chosen.
The template is selected from the following:

1. For data retrieval queries (SELECT):
   - SELECT <select clause>
     FROM <tables>
     WHERE <where clause>
     ORDER BY <order by clause >
     GROUP BY <group by clause>
     HAVING <having clause>
     LIMIT <limit clause>.
2. For data manipulation queries (INSERT, UPDATE, DELETE):
   - INSERT INTO <insert clause>
     VALUES <values clause>
   - UPDATE <update clause>
     SET <set clause>
     WHERE <where clause>
   - DELETE FROM <delete clause>
     WHERE <where clause>

Based on the data about various clauses collected from earlier steps and the information about attributes and tables stored in the attribute-table map, the final query is formed by filling in the information into the appropriate template. Depending on the clause data collected from earlier steps, corresponding <> are filled.
Depending on the relation between multiple tables, the decision of INNER JOIN or NATURAL JOIN is taken. For example, if there are two tables. If these two tables have one common attribute and is named the same in both, then there is NATURAL JOIN between the tables. But if the common attribute is named differently in the two tables, then there is INNER JOIN between the tables. The final query is as shown in Fig 2.

4 Results and Analysis

The corpus that can be used to test our system is not readily available and is dependent on a database. Hence, we have tested our system on a synthesized corpus of natural language statements related to a bank and a university database. The university and bank database consists of 11 and 6 tables respectively. However, system can work on any complex database. The natural language statement has to be a single sentence. The system has been evaluated on a corpus of around 75 natural language statements of university database and around 50 related to bank database. The accuracy of the system is found out to be around 86%. The system gives the same SQL query as the output when the same natural language statement is represented in different ways. If the system fails to generate SQL query corresponding to any natural language statement, an error message is displayed. These are a few results given by the system on the university corpus:

1. Find the student name where instructor name is 'Crick'.

   - SELECT DISTINCT student.stud_name
     FROM instructor
     INNER JOIN advisor
     ON instructor.ID = advisor.inst_ID
     INNER JOIN student
     ON student.ID = advisor.stud_ID
     WHERE instructor.name = 'Crick'

   In this database, the tables 'student' and 'instructor' are linked through the table 'advisor'. So, we can see that this query deals with multiple tables which are joined by INNER JOIN.

2. Find all student name whose credits are between 90 and 100 and department name is 'Finance' or 'Biology'.

   - SELECT DISTINCT student.stud_name
     FROM student
     WHERE (student.tot_cred
     BETWEEN '90' AND '100') AND (
     student.dep_name = 'Finance' OR
     student.dep_name = 'Biology')

   The above query showcases multiple conditions within the WHERE clause. This query also involves use of BETWEEN clause and logical clauses like AND, OR.

3. List all student names whose credits are 50 in decreasing order of credits.

   - SELECT DISTINCT student.stud_name
     FROM student
     WHERE student.tot_cred = '50'
     ORDER BY student.tot_cred DESC

   Another type of query is the one involving sorting its result based on some attribute. For this purpose, the query uses the ORDER BY clause to sort the results in decreasing order.

4. Give the department name where maximum salary of instructor is greater than 50000.

   - SELECT DISTINCT
     instructor.dep_name
     FROM instructor
     GROUP BY instructor.dep_name
     HAVING
     MAX(instructor.salary) >'50000'

   In SQL, when an aggregate function is compared to constant, like in this case maximum of salary is compared to 50000, then the query involves use of HAVING clause instead of a WHERE clause. Also, whenever HAVING is used, the results are supposed to be grouped by the attributes in the SELECT clause.

5. Give the department name where salary of instructor is greater than average of salary.

   - SELECT DISTINCT
     instructor.dep_name
     FROM instructor
     WHERE instructor.salary >
     (SELECT AVG(instructor.salary)
     FROM instructor)

   This query showcases a special case of nested queries. Whenever an attribute is compared to the result of an aggregate function, i.e. in this case salary greater than average of salary, we have to use nested query.

6. Find the course taught by Crick.

   - SELECT DISTINCT teaches.course_id
     FROM teaches NATURAL JOIN instructor
     WHERE instructor.name = 'Crick'

   Till now, we have seen cases in which an attribute associated to the value is mentioned in the natural language statement. In this case, we handle cases where attribute is not mentioned. We find out the most appropriate attribute for the given value.

7. (a) Publish in alphabetic order the names of all instructors.
   (b) Give names of all the instructors in alphabetic order.
   (c) Give instructors names in ascending order.

   - SELECT DISTINCT instructor.name
     FROM instructor
     ORDER BY instructor.name ASC

   As seen in this example, there can be multiple ways of representing the same natural language statement.The system gives the same

SQL query as the output when the same natural language statement is represented in different ways.

8. Insert a student whose id is 5, name is Josh, department name is Physics and credits are 150.
   - INSERT INTO student
     (student.ID, student.stud_name,
     student.dep_name, student.tot_cred)
     VALUES
     ('5' , 'Josh' , 'Physics' , '150')

   In addition to the data retrieval queries, our system also provides a natural language interface to insert data into the database. Other DML queries such as UPDATE and DELETE are also provided by the system.

5 Limitations

The following are some of the types of inputs that are not presently handled by our system.

1. Find the capacity of the classroom number 3128 in building Taylor
   - SELECT *
     FROM classroom
     WHERE classroom.capacity = '3128'
     AND classroom.building = 'Taylor'

   In this particular example, the system fails to decide whether to take 'capacity of classroom' or 'classroom number' as an n-gram. Hence, the mapping fails.

2. Who teaches Physics?
   - SELECT *
     FROM department
     WHERE
     department.dep_name = 'Physics'

   In this example, the implicit query module of our system is able to map Physics to 'department name' attribute from table 'department'. But it fails to identify that 'who' refers to a person (an instructor).

6 Comparison and Conclusion

Similar existing systems:

1. Complex Queries are not handled very well.
2. Only a few types of aggregate functions have been taken care of.
3. No system has incorporated HAVING, GROUP BY and other clauses.
4. Many systems were specific use systems which were limited to a certain database.
5. No system till date incorporates such a wide range of queries.

Our System:

- The system is currently capable of generating
  1. Simple queries.
  2. Complex queries involving natural and inner joins.
  3. Aggregate functions in queries.
  4. Advanced 'WHERE' clauses.
  5. ORDER BY, GROUP BY, HAVING and LIMIT clauses.
  6. Basic implicit queries.
  7. DML Queries like INSERT, UPDATE and DELETE.
- The system works irrespective of the selected MySQL database.
- No system till date incorporates such a wide range of queries.

7 Future Work

The following points are not yet incorporated in the system and are hence left as future work. The development on the points mentioned in future work is in progress.

1. It is possible that a natural language statement can result in multiple SQL queries.
   Machine learning can be incorporated to choose the most efficient query.
2. This system only considers MySQL database system. It can be expanded to work for any other database system or unstructured databases.
3. More efficient algorithm to handle implicit queries can be developed.
4. Only single sentence natural language input is handled. Multiple sentences which result in a single query can be incorporated.
5. Neural methods can be used to solve the problem of indecisiveness of n-grams.

References

Pranali P. Chaudhari. November 2013. Natural language statement to sql query translator. *International Journal of Computer Applications (0975-8887)* 82(5).

Prof. Debarati Ghosal, Tejas Waghmare, Vivek Satam, and Chinmay Hajirnis. March 2016. Sql query formation using natural language processing (nlp). *International Journal of Advanced Research in Computer and Communication Engineering* 5(3).

Prasun Kanti Ghosh, Saparja Dey, and Subhabrata Sengupta. 2014. Automatic sql query formation from natural language query. *International Journal of Computer Applications (0975-8887), International Conference on Microelectronics, Circuits and Systems (MICRO-2014)* .

Hendrix. 1978. Lifer / ladder .

Prabhdeep Kaur and Shruthi J. Jan 2016. Conversion of natural language query to sql. *International Journal of Engineering Sciences and Emerging Technologies* .

Ashish Palakurthi, Ruthu S. M., Arjun R. Akula, and Radhika Mamidi. 2015. Classification of attributes in a natural language query into different sql clauses. *IBM Research, Bangalore, India* .

Gauri Rao, Chanchal Agarwal, Snehal Chaudhry, Nikita Kulkarni, and Dr. S.H. Patil. 2010. Natural language query processing using semantic grammar. *International Journal on Computer Science and Engineering* 2(2):219–223.

Filbert Reinaldha and Tricya E. Widagdo. 2014. Natural language interfaces to database (nlidb): Question handling and unit conversion. *IEEE* .

F. Siasar, M. Norouzifard, S. H. Davarpanah, and M. H. Shenassa. May 2008. Using natural language processing in order to create sql queries. *Proceedings of the International Conference on Computer and Communication Engineering 2008* .

Garima Singh and Arun Solanki. 2016. An algorithm to transform natural language into sql queries for relational databases. *Selforganizology* 3(3):100–116.

W. A. Woods. 1973. Progress in natural language understanding-an application to lunar geology. *National Computer Conference* pages 451–460.