International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 02 | Feb 2021 www.irjet.net p-ISSN: 2395-0072

CONVERSION OF SQL QUERY TO NATURAL LANGUAGE

Dhairya Timbadia

Undergraduate Student, Rajiv Gandhi Institute of Technology, Andheri, Mumbai

---

Abstract - *This project aims to develop a system which converts a natural language statement into SQL query to retrieve information from respective database. The natural language input statement taken from the user is passed through various Open NLP 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 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 Oracle SQL database.*

***Key Words*: Natural language Processing, Query, Tokenization, Parts Of Speech Tagging, Stemming and Lemmatization.**

INTRODUCTION

Natural Language Processing is a subfield of Artificial Intelligence used to build intelligent computers that can interact with the human being like a human being. It bridges the man-machine gap. The main purpose of Natural Language Query Processing is for the interpretation of the English sentences by computer. In spite of all the challenges, it is being used widely for research purpose. Natural Language Processing can be used to access the database by asking questions in Natural Language and getting the required results. Asking questions in natural language to databases is a very convenient and easy method of data access, especially for users who do not have knowledge about the complicated database handling query languages such as SQL (Structured Query Language).

There are many challenges in the conversion of natural language query to SQL query like ambiguity which means that one word can have more than one meaning. In this case, one word maps to more than one sense. Another challenge is the formation of complex SQL query and next challenge is about Discourse knowledge in which immediately preceding sentence affects the interpretation of next sentence for example if the user enters SELECT and INSERT query at the same time, then such a case is not understandable by the system.

Literature Review

The problem we address is a subcategory of a broader problem; natural language to machine language. SQL is opportunistic for its distinctive, high level language and close connection to the underlying data. We utilize these characteristics in our project. SQL is tool for manipulating data. To create a system which can generate a SQL query from natural language we need to make the system which can understand natural language.

Most of the research done until now solves this problem by teaching a system to identify the parts of speech of a particular word in the natural language which is called tagging. After this the system is made to understand the meaning of the natural query when all the words are put together which is called parsing. When parsing is successfully done then the system generates a SQL query using proper syntax of Oracle SQL.

Existing System

The existing approach is to generate the query from the knowledge of SQL manually. But certain improvement done in recent years helps to generate more accurate queries using Probabilistic Context Free Grammar (PCFG). The current implemented standard is QuePy and similar, disjoint projects like them. These projects use old techniques; QuePy has not been updated in over a year. The QuePy website has an interactive web app to show how it works, which shows room for improvement. QuePy answers factoid questions as long as the question structure is simple. Recent research such as SQLizer presents algorithms and methodologies that can drastically improve the current opensource projects. However, the SQLizer website does not implement the natural English to query aspect found in their 2017 paper. We wish to prove these newer methods.

Research Gap

The following are some of the types of inputs that are not presently handled by our system. 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 class- room’ or ‘classroom number’ as an n-gram. Hence, the mapping fails

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). Our system struggles with column value references in the natural language. It can hang trying to find the match to the column value to a word in the schema.

Problem Statement and Objectives

The objective of our project is to generate accurate and valid SQL queries after parsing natural language using open source tools and libraries. Users will be able to obtain SQL statement for the major 5 command words by passing in an English sentence or sentence fragment. We wish to do so in a way that progresses the current open source projects towards robustness and usability. This project makes use of natural language processing techniques to work with text data to form SQL queries with the help of a corpus which we have developed. En2sql is given a plain English language as input returns a well-structured SQL statement as output.

Methodology

1. Data Collection

With the domain level knowledge of SQL we will create a corpus which will contain words which are synonymous the SQL syntax to SELECT, LIMIT, FROM, etc. This is common among the open source projects we have seen. Many of the open source projects we have inspected use such keywords, thus coming up with a generous keyword corpus will be easy. If our English to keyword mapping results are not desirable, we may use an online thesaurus API. An Oracle SQL database will be constructed with data from the public Yelp SQL Database [13]. We chose the yelp dataset because it is fairly large, has a good amount of tables, and we have some domain level knowledge about Yelp already. This data will be used as a corpus and for testing. The corpus will be constructed from the table names, column names, table relationships, and column types. The database corpus will be used in an unsupervised manner to keep the program database agnostic. A set of substructure queries will be used as a starting point for the queries. The natural language tokens will be matched to these.

2. Solution Structure

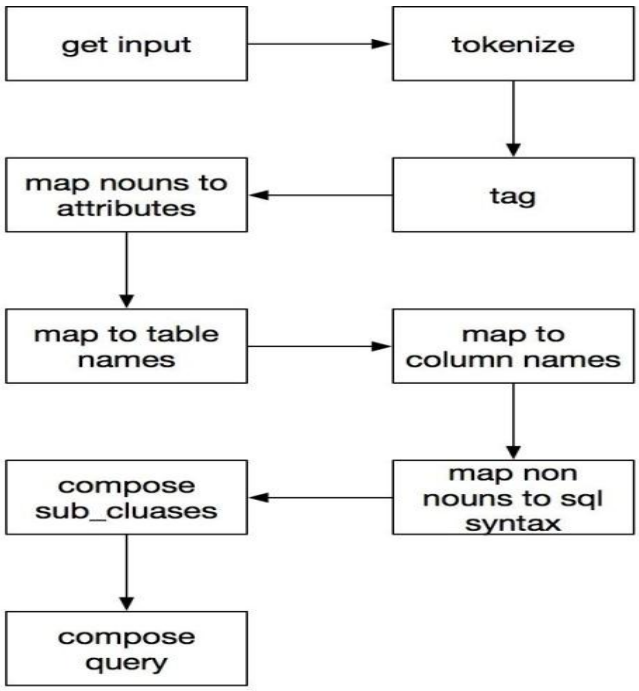

Figure 1 Solution Structure

Algorithm Design

Following will be our algorithm

1. Scanning the database: Here we will go through the database to get the table names, column names, primary and foreign keys.

2. Input : We will take a sentence as a input from the user (usinginput.txt)

3. Tokenize and Tag : We will tokenize the sentence and using POS tagging to tag the words

4. Syntactic parsing: Here we will try to map the table name and column name with the given natural query. Also, we will try to identify different attributes of thequery.

5. Filtering Redundancy: Here we will try to eliminate redundancy like if while mapping we have create a join requirement and if they are not necessary then we remove the extra table.

6. Query Formation: Here we will form a complete SQL query based on MySQLsyntax.

7. Query Execution: Here we will execute the query on database to getresults.

Results and Discussion

1. Implemented Algorithm's Pseudo-code

We propose a system which looks to overcome the shortcomings of existing system that gets a natural language sentence as an input, which is then passed through various phases of NLP to form the final SQL query.

2. Tokenize and Tag

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

3. 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 using a pre-created corpus of words. 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.

4. 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 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 lemmatizer 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.

5. 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.

6. 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:

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>.

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.

CONCLUSIONS

This project has given us a great opportunity to come up with a solution for writing tedious queries. This project though helps resolving basic queries but with time it can made powerful to handle complex queries, normalization and also can be extended for NoSQL. We were able to learn and implement NLTK, cosine, tf-idf of python3. We have got accuracy around 30-50% in basic queries.

REFERENCES

1. A Natural Language Database Interface Based on a Probabilistic Context Free Grammar, IEEE International Workshop on Semantic Computing and Systems 978-0-7695-3316-2/08 $25.00 © 2008 IEEE DOI 10.1109/WSCS.2008.14

2. Domain Specific Query Generation from Natural Language Text, The Sixth International Conference on Innovative Computing Technology (INTECH 2016) 978-1-5090-2000-3/16/$31.00 ©2016 IEEE

3. Generic Interactive Natural Language Interface to Databases (GINLIDB), Proceedings of the WSEAS International Conference on Evolutionary Computing ISSN : 1790-5109 ISBN : 978-960-474-067-3

4. Natural Language Interface to Database Using Modified Co-occurrence Matrix Technique, 2015 International Conference on Pervasive Computing (ICPC) 978-1-14799-6272-3/15/$31.00(c)2015 IEEE

5. Natural language to SQL Generation for Semantic Knowledge Extraction in Social Web Sources, Indian Journal of Science and Technology, Vol 8(1), 01-1, January 2015 ISSN (Online) : 0974-5645 ISSN (Print) : 0974-6846 DOI : 10.17485/ijst/2015/v811/54123

6. Natural Language Query Processing Using Semantic Grammar, Gauri Rao et al.

/ (IJCSE) International Journal on Computer Science and Engineering Vol.02, No.02, 2010, 219-223 ISSN 0975-3397

7. SQLizer : Query Synthesis from Natural Language, Proc. ACM Program. Lang., Vol. 1, No. OOPSLA, Article 63. Publication date : October 2017

8. Synthesizing Highly Expressive SQL Queries from Input-Output Examples, PLDI'17, June 12-23, 2017, Barcelona, Spain ACM.

978-2-4503-4988-8/17/06…$15.00
http://dx.doi.org/10.1145/3062341.3062365

9. "Sqlizer API." Easily Convert Files into SQL Databases | SQLizer. N.p., n.d. Web. 27 Feb. 2018.

10. Machinalis. "Quepy." A Python Framework to Transform Natural Language Questions to Queries. N.p., n.d. Web. 27 Feb. 2018.

11. "Natural Language Toolkit¶." Natural Language Toolkit - NLTK 3.2.5 Documentation. N.p., n.d. Web. 27 Feb. 2018.

12. Alarfaj, Salah. "SalN3t/NlpSQL." GitHub. N.p., 11 Dec. 2017. Web. 27 Feb. 2018.

13. Yelp. "Yelp SQL Dataset." Yelp Dataset. N.p., n.d. Web. 27 Feb. 2018.