|| Volume 5 || Issue 9 || September 2020 || ISSN (Online) 2456-0774

INTERNATIONAL JOURNAL OF ADVANCE SCIENTIFIC RESEARCH AND ENGINEERING TRENDS

NATURAL LANGUAGE TEXT TO SQL QUERY BY PYTHON WRAPPER

Mrityunjay kr[1], Niraj Kumar [2], Prof Prabodh Nimat[3]

Dept. Computer Engineering , Dr. D.Y Patil Institute of technology, Pimpri, Pune[1,2,3]

sinhamrityunjay@gmail.com,nirajkumarsinghnks17@gmail.com

Abstract: Databases are increasingly common and are becoming increasingly important in actual applications and Web sites. They often used by people who do not have great competition in this domain and who do not know exactly their structure. This is why translators from natural language to SQL queries are developed. Unfortunately, most of these translators are confined to a single database due to the specification of the base architecture. In this paper, we propose a method to query any database from any language. We evaluate our application on databases and we also show that it supports more operations than most other translators of these translators are confined to a single database due to the specification of the base architecture. In this paper, we propose a method to query any database from any language. We evaluate our application on databases and we also show that it supports more operations than most other translators.

Keywords: *Databases, SQL Query, English to SQL Translator, Natural language interfaces to database, TreeTagger.*

I INTRODUCTION

For several years, databases (DB) have become inevitable for all Websites or applications managing large amounts of information, such as user accounts (banks, transport agencies, social network, video games, etc.).The Internet has gradually become democratized and popularized, but databases still remain abstract for many people. Some positions do not require any computer training or in data administration still require working closely with the databases, as in accounting or secretarial work for example. It is with the aim that a person, having no competence in the field database management, cannot directly administer one, but at least understand how it works, interact with it and perform simple

II OBJECTIVE:

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 command words by passing in a sentence or sentence fragment. We wish to do so in a way that progresses the current open source projects towards robustness and usability.

III RELATED WORK

A.Natural language processing for speech synthesis

A system and the method interact with networked objects, via a computer using the utterance, speech processing and natural language processing. A data definition file relates networked objects and a speech processor. The speech processor searches a first grammar file for a matching phrase for the utterance, and search a second grammar file for a matching state if the coordinating expression isn't found in the principal sentence structure record. The system also include a natural language processor to search a database for a matching entry for the matching phrase. The natural language processing is the computerized approach to analyzing text and being a very active area of research and development. This is based on the text to tasks on it (interrogation, addition, deletion), which translators of the language, natural to database query language have emerged.

B. A bit of progress in language modeling.

speech conversion in which the text data is first input to the system. It uses high levels of modules for speech synthesis. It uses sentence segmentation which deals with punctuation mark with simple decision tree [2].

Our perplexity diminishment are maybe the most noteworthy detailed contrasted with a baseline[5] .

IV IMPLEMENTATION

System Design

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.

1.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 [14]. All processes following this step use these tagged tokens for processing.

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 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.

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 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.

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.

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):

1.1. 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):

2.1. INSERT INTO <insert clause> VALUES <values clause>

2.2. UPDATE <update clause> SET <set clause> WHERE <where clause>

2.3. 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 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.

ADVANTAGES

1.The tool can deal with any language, so long as it has its configuration file (i.e. a file with the keywords of the language).

2.Language configuration files can be found in lang/

directory. The files are CSV files. Each line represent a type of keywords. Anything before the colon is ignored. Keywords must be separated by a comma.

3.We can build our own language configuration file following the English and French templates.

DISADVANTAGES

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’

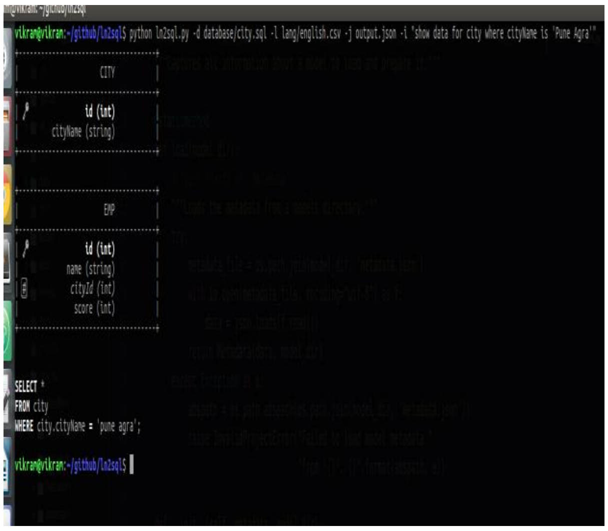
In this particular example, the system fails to decide whether to take ‘capacity of class- room’ or ‘classroom number’ as an n-gram. Hence, the mapping fails

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).

V RESULT:



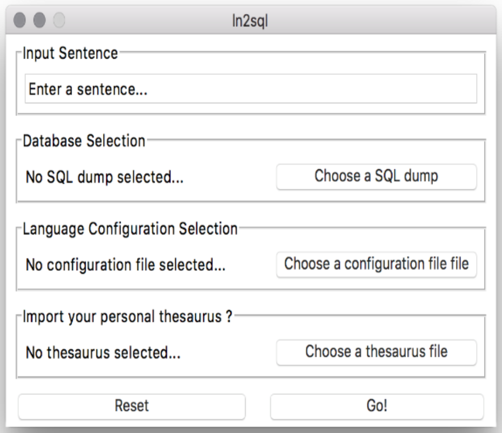
```
vikram@vikram:~/github/ln2sql$ python ln2sql.py -d database/city.sql -l lang/english.csv -j output.json -i "show data for city where cityName is 'Pune Agra'"
+------------------------------+
|                  CITY        |
+------------------------------+
|              id (int)        |
|       cityName (string)      |
+------------------------------+

+------------------------------+
|                   EMP        |
+------------------------------+
|              id (int)        |
|           name (string)      |
|           cityId (int)       |
|            score (int)       |
+------------------------------+

SELECT *
FROM city
WHERE city.cityName = 'pune agra';

vikram@vikram:~/github/ln2sql$
```

OUTPUT



VI CONCLUSION

The ability to find query for any simple language statement make it more useful for people with no knowledge of database. The approach itself can revolutionize and make the work easier. In addition, provision is made to detect the language of the request entered by the user in order to use a dictionary of synonyms relating to this language and to adjust the rules according to the language to thus make the robust system for languages other than English. To conclude, although it can be improved, this approach allows you to query any SQL database, thus meeting the portability objectives set, while keeping performance within the average of the applications already existing and covering a wide range of selection operations.

REFERENCES

1.TO THEXANDER R., R U KS HA N P. & M AHESAN S. (2013). Natural Language Web Interface for Database (NLWIDB). In CoRR .

2.A NDROUTSOPOULOS I., RITCHIE G. & THANISCH P. (1995). Natural Language Interfaces to Databases - An Intro-

duction. In Journal of Natural Language Engineering , 1 , 29–81.

3.C HANDRA Y. (2006). Natural Language Interfaces to Databases . PhD Thesis. University of North Texas, USA.

4.C HAUDHARI P. P. (2013). Natural Language Statement to SQL Query Translator. In International Journal of Computer Applications , 82 (5), 18–22.

5.C HEN W. (2014). Parameterized Spatial SQL Translation for Geographic Question Answering. In Semantic Computing (ICSC), 2014 IEEE International , p. 23–27.

6.C IMIANO P. & M INOCK M. (2009). Natural Language Interfaces: What is the Problem ? - A data-driven quantitative analysis. In 14th International Conference on Applications of Natural Language to Information System (NLDB) , 5723 .

7.D ESHPANDE A. K. & D EVALE PR (2012). Natural Language Query Processing Using Probabilistic Context Free Grammar. In International Journal of Advances in Engineering and Technology , 3 , 568–573.

8. D JAHANTIGHI F. S., N OROUZIFARD M., D AVARPANAH YEAR D S. & SHEY N / ASSTo Mr. H. (2008). Using Natural Lan- guage Processing in Order to Create SQL Queries. In Proceedings of the International Conference on Computer and communication engineering. Communication Engineering , p. 600–604.

9.G IORDANI A. & M OSCHITTI A. (2009). Semantic Mapping Between Natural Language Questions and SQL Queries via Syntactic Pairing. In Natural Language Processing and Information Systems , 5723 , 207–221.

10. G IORDANI A. & M OSCHITTI A. (2012). Translating Questions to SQL Queries with Generative Parsers Discriminati- vely Reranked. In COLING 2012: Posters , p. 401–410.