RESEARCH ARTICLE OPEN ACCESS

Converting Natural Language Query to SQL Query

Buddhaditya Rath
Computer Engineering, University of Mumbai, Mumbai – 400098.
Email: adityaarath97@gmail.com

--************************----------------------------------

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. We investigate avenues of using natural English utterances - sentence or sentence fragments - to extract data from an SQL, a narrower inspection of the broader natural language to machine language problem. We intend to contribute to the goal of a robust natural language to data retrieval system.

--************************----------------------------------

I. 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.

II. LITREATURE SURVEY

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.

III. SURVEY 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 open source 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.

IV. LIMITNG EXISTING SYSTEM OR 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 3.

```
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.

V. PROBLEM STATEMENT & OBJECTIVES

PROBLEM STATEMENT: 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.

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 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.

VI. SCOPE

We will be implementing SELECT, INSERT, DELETE, UPDATE query and WHERE clauses. We hope to implement more clauses such as join, aggregate, order by, limit, etc but cannot commit to these more challenging due to their added complexity and time constraints. The research willstart by focusing on statements(“get / find number of employees”) and then extend to questions (“Who is Bob?”). The statements are easier as it gives more keywords to interpolate the SQL query structure. There are many relations databases. While their SQL syntax is similar, it can differ for more complicated queries. We will focus on Oracle SQL as it is an open source database with a large user base.

VII. PROPOSED SYSTEM

The proposed approach aims to use knowledge of SQL to create a corpus which will help to identify

SQL command words i.e. SELECT, INSERT, DELETE, UPDATE and map the tokens with appropriate POS. Word similarities will be calculated with the input tokens to the database schema (table names, column names, data) to insert table names, column names, and data comparisons into the query.

VIII. DATA ANALYSIS & DISCUSSION

DATASET: We will create our own corpus by scanning the schema for the table name, column name, column types, key relations, and the data. This will be schema specific dataset. Another corpus will contain all the necessary elements to build the query. It will contain probable words for Select, Insert, Delete, Where which will help to form a query based on the input. We also use Stanford's POS corpus and the WordNet corpus vianltk.

SETUP: For implementing En2SQL you will need following packages. Python3, NLTK, pymysql, Stanford's POS Tagger [14], Oracle MySql, and the Yelp SQL Dataset [13]. First you will need to set up a user for the MySQL database, then upload the Yelp SQL data to the database. We include the POS tagger with the code. Run the requirements.txt (via pip3) file to install the python package requirements (nltk and pymysql). Update the database connection details in the db.config.py file. Input natural language queries into input.txt, one per line. Run the main.py file.

RESULT & 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:

i. Find the student name where instructor name is 'Crick'.
SELECT DISTINCT student.stud name
FROM instructor INNER JOIN advisor
ON instructor.ID = advisor.instID
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.

ii. 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.

iii. 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 7

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.

iv. Give the department name where maximum salary of instructor is greater than50000.

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.

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

vi. 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.

vii. o Publish in alphabetic order the names of all instructors.o Give names of all the instructors in alphabetical order.o 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.

vii. 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.totcred) 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.

ABNORMAL CASE EXPLAINATION:
Some input table name, column name consist of underscore, short forms due to which it becomes unusual and hard for it to distinguish amongst stop word, normal word. So we have to add it to corpus or explicitly mention it before using it. The accuracy while generating queries shows a minute fluctuations. Some English statements are very less informative. For example: Who isBob? This question if asked for a huge database creates an ambiguity to find the correct answer. It sometimes gives and correct output and sometimes it gives a vague output.

IX. 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 an input from the user (using input.txt)

3. Tokenize andTag: 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 created 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

X. DETAILS OF HARDWARE & SOFTWARE REQUIREMENTS

Hardware Requirements • 4GB RAM. • 10 GB HDD. • Intel 1.66 GHz Processor Pentium 4

Software Requirements • Windows XP/Vista or higher • Python 3.6.3

XI. DETAILED DESIGN

Language Our project uses Python 3.6. Python has many readily available and proven open source libraries. All our required libraries support Python 3.6.Tools used NLTK3 library for python will be used for input stemming. This library serves as a toolkit for computational linguistics. Following is a list of the modules we will be using. Token module provides basic classes for processing individual elements of text, such as words, or sentences. NLTK Tokenizer is used to tokenize.

XII. 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.

XIII. SOLUTION STRUCTURE

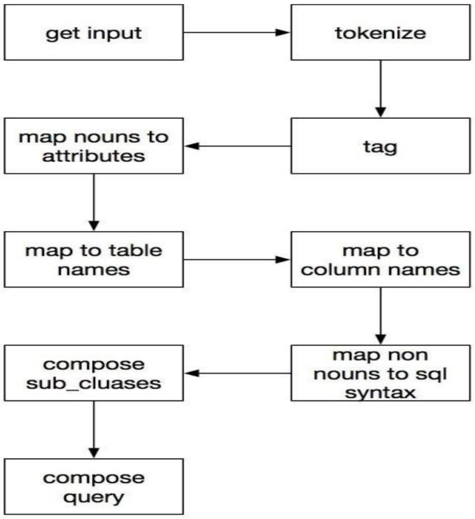

Figure 1.1: Solution Structure

XIV. OUTPUT AND TESTING

The program with output a structure SQL query that runs on the database and attempts to answer the input question or statement. The output is displayed to standard output as well as into output.txt.

To test our code we will first create a schema specific corpus which will contain data related to table, column name and column data. Another corpus will contain data related to query command SELECT. And then we will be giving it a general natural language statement to test it. It will take input of the natural language and then will make use of the two corpuses and thus will output a SQL query. We will take the output query and run it against the MySQL Yelp Database, testing the feasibility of the query. After it runs, we will take the resulting data and compare it to our expected results. For the last test, we will inspect the query for correctness making sure a wrong query does not return the correct data. We will need to construct a set of natural English with expected output pairings. If the query can pass the first two automated tests, then it will need to be hand inspected for correctness. If all three tests pass, the query is correct. With this testing methodology we will construct an accuracy for the program.

XV. RESULT & DISCUSSION

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.

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.

ANALYZE TAGGED TOKENS

Based on the tagged tokens of earlier step, the noun map and verb list are 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. Inany 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", "me an": "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.

MAP TO TABLE NAMES & 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.

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 algorithms 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.

FORM THE FINAL QUERY & EXECUTE

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.

XVI. DISCUSSION-COMPARITIVE STUDY

We wanted to analyse the custom bag of words using LSTM based RNN network and verify how the performance of system changes. Instead of using trivial NLP techniques like NLTK, custom corpus we wanted to use Stanford NER library which contains all the known words with tags. There are many steps to improve the work we have done. A using thesaurus to match input tokens not only to the table and column names but their synonyms as well would greatly increase our accuracy and chances a natural language token would be matched to a correct corpus name. 20 The next changes to be implemented are better column value matching. Currently the system has a hard time matching an input token to a column value (as opposed to a column or table name). After parsing all table and column names, we can use the LIKE MySQL syntax with %% to find column values that contain the input token. We would only evaluate nouns in this case. We could count the number of rows that contain the token and use the column with the highest count. We have not addressed abbreviations in this project. Simply, a corpus of English abbreviations would be used to map from common words to abbreviations and vise versa. This project does not allow for user input after a failed query. The future scope of this project will look at prompting the user for correct input token to corpus token mappings to build up a thesaurus on the database and improve its performance. It also does not address non-natural column or table names, studentName or student_name for example. In this case we could split on camel case and underscores respectively add keep then nouns in the new word set. The input tokens can be matched against the set of words. Before the code is useful, tests need to be written and the coding style needs to be updated.

XVII. CONCLUSION

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.

ACKNOWLEDGMENT

I, Buddhaditya Rath, wish to state that the work embodied in this synopsis titled" Conversion of Natural Language Query to SQL Query" forms our own contribution to the work carried out. I declare that this written submission representsideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from

the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

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 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

[4] 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

[5] 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

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

[7] 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…http://dx.doi.org/10.1145/3062341.3062365