Generating SQL queries from Natural Language

Nihal Surendra Parchand
np9603@rit.edu

Rohit Kunjilikattil
rk4447@rit.edu

Viraj Chaudhari
vc6346@rit.edu

1. MOTIVATION

In a world where data has become more valuable than oil, advancement in any data related fields are becoming more and more significant. And in any field that deals with a lot of data, data retrieval is still the starting point for almost every task. Data retrieval requires writing long SQL queries which can be tedious and time-consuming. To save this time, we aim to implement a system that can investigate the English language occurrences, i.e. sentences or sentence fragments and transform it into an SQL query to extract data from a database. Our goal is to create a robust and generalized end-to-end system which will allow the user to perform data retrieval using the basic English language, which is a lot more convenient than having to write the entire SQL query.

2. INTRODUCTION

We have implemented two approaches for this project. Our first approach will be to use our knowledge of SQL to create a corpus and identify all variations of the SELECT SQL commands. We will then map these token with appropriate Parts-Of-Speech (POS). Then we search the input queries for words related to the table names, column names, etc based on word similarities. As the input words may not exactly match the database attributes or table names, we will use techniques like stemming and lemmatization to get the root word and match it properly. The accuracy of the parser will depend upon the correctness of the mapping in the created corpus. Also SQL queries have a standard template which we will use to properly identify the components in the query. We found that the lemmatization and stemming process was not enough to match all possible variations of the column and table names.

For the second approach, we have created a context free grammar in order to parse the natural language queries and convert them to SQL queries. The inspiration for the grammar is from the sql0.fcfg and sql1.fcfg which are the feature context free grammars mentioned in the NLTK library.

2.1 Data Set Description

The world database used in this project was collected from the mysql documentation https://dev.mysql.com/doc/index-other.html.[2] The database consists of 3 tables:

- **country** - Information about different countries of the world. Its attributes are:
  Code (char) , Name (char), Continent (enum), Region (char), SurfaceArea (float), IndepYear (smallint), Population (int), LifeExpectancy (float), GNP (float), GNPOld (float), LocalName (char), GovernmentForm (char), HeadOfState (char), Capital (int), Code2 (char)
  Total number of countries are 239.

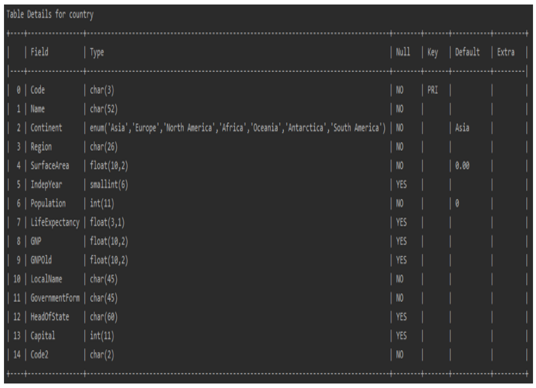
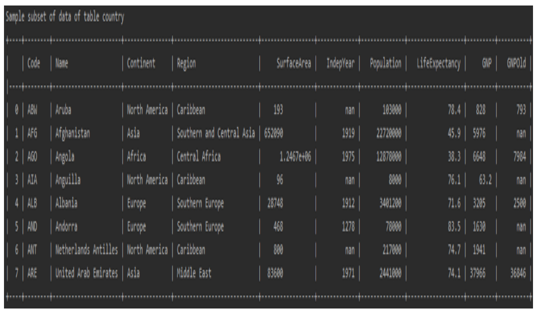


Table Details for country

	Field	Type	Null	Key	Default	Extra
0	Code	char(3)	NO	PRI		
1	Name	char(52)	NO			
2	Continent	enum('Asia','Europe','North America','Africa','Oceania','Antarctica','South America')	NO		Asia	
3	Region	char(26)	NO			
4	SurfaceArea	float(10,2)	NO		0.00	
5	IndepYear	smallint(6)	YES			
6	Population	int(11)	NO		0	
7	LifeExpectancy	float(3,1)	YES			
8	GNP	float(10,2)	YES			
9	GNPOld	float(10,2)	YES			
10	LocalName	char(45)	NO			
11	GovernmentForm	char(45)	NO			
12	HeadOfState	char(60)	YES			
13	Capital	int(11)	YES			
14	Code2	char(2)	NO			

Figure 1: Country Table



Sample subset of data of table country

	Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy	GNP	GNPOld
0	ABW	Aruba	North America	Caribbean	193	nan	103000	78.4	828	793
1	AFG	Afghanistan	Asia	Southern and Central Asia	652090	1919	22720000	45.9	5976	nan
2	AGO	Angola	Africa	Central Africa	1.2467e+06	1975	12878000	38.3	6648	7984
3	AIA	Anguilla	North America	Caribbean	96	nan	8000	76.1	63.2	nan
4	ALB	Albania	Europe	Southern Europe	28748	1912	3401200	71.6	3205	2500
5	AND	Andorra	Europe	Southern Europe	468	1278	78000	83.5	1630	nan
6	ANT	Netherlands Antilles	North America	Caribbean	800	nan	217000	74.7	1941	nan
7	ARE	United Arab Emirates	Asia	Middle East	83600	1971	2441000	74.1	37966	36846

Figure 2: Country Subset

- **city** - Information about some of the cities in those countries. Its attributes are:
  ID (int), Name (char), CountryCode (char), District

(char), Population (int)
Total number of cities are 4079.

Table Details for city

	Field	Type	Null	Key	Default	Extra
0	ID	int(11)	NO	PRI		auto_increment
1	Name	char(35)	NO			
2	CountryCode	char(3)	NO	MUL		
3	District	char(20)	NO			
4	Population	int(11)	NO		0	

Figure 3: City

Sample subset of data of table city

	ID	Name	CountryCode	District	Population
0	1	Kabul	AFG	Kabol	1780000
1	2	Qandahar	AFG	Qandahar	237500
2	3	Herat	AFG	Herat	186800
3	4	Mazar-e-Sharif	AFG	Balkh	127800
4	5	Amsterdam	NLD	Noord-Holland	731200
5	6	Rotterdam	NLD	Zuid-Holland	593321
6	7	Haag	NLD	Zuid-Holland	440900
7	8	Utrecht	NLD	Utrecht	234323

Figure 4: City Subset

- **countrylanguage** - Languages spoken in each country. Its attributes are:
  CountryCode (char), Language (char), IsOfficial (enum), Percentage (float)
  Total number of countrylanguage are 984.

Table Details for countrylanguage

	Field	Type	Null	Key	Default	Extra
0	CountryCode	char(3)	NO	PRI		
1	Language	char(30)	NO	PRI		
2	IsOfficial	enum('T','F')	NO		F	
3	Percentage	float(4,1)	NO		0.0	

Figure 5: Country Language

3. IMPLEMENTATION

This section describes the implementation of both the approaches. The approach for the application is described in following section 3.1,3.2 and 3.3.

3.1 Non-CFG Approach

Word Structure Handling: All the text input is converted to lowercase characters to handle the mismatch of the words and maintain uniformity.

Sample subset of data of table countrylanguage

	CountryCode	Language	IsOfficial	Percentage
0	ABW	Dutch	T	5.3
1	ABW	English	F	9.5
2	ABW	Papiamento	F	76.7
3	ABW	Spanish	F	7.4
4	AFG	Balochi	F	0.9
5	AFG	Dari	T	32.1
6	AFG	Pashto	T	52.4
7	AFG	Turkmenian	F	1.9

Figure 6: Country Language Subset

Removing Stop Words:
Removed Stop words and escape characters so as the relevance of the input is preserved and we can focus on the important query information rather focusing on the irrelevant data. We have done this with the Regex functions and NLTK libraries available in the python module and simply discarding those words.

Tokenization:
It is a process of splitting the input or sentence into meaningful tokens or units. So we have tokenized the data with the help of the NLTK libraries available in the python module.

Handling Ambiguity with POS Tagging:
In this step, we are Part-Of-Speech Tagging the input provided to us with the Stanford POS Tagger library available in the Python module.[3] In this we are mainly focusing on the noun tags as the column and table names are generally considered as nouns.

Lemmatization and Syntax Parsing:
After the above performed steps,now we want to map the input query words to the words in our database, to fetch the correct query and obtain the results. So have performed lemmatization of the input text and will match the word in the database by maintaining all possible words for that word in the dictionary.

We have loaded the database and have made a GUI for user input. After which the user input is taken and the removal of stop words, tokenization and POS tagging and lemmatization of the input are achieved using NLTK libraries.[4]

Formation of Query from Natural Language:
In this step, after all the input text data is cleaned and pre-processed, the text will be converted into the SQL query by tagging each relation properly and result is displayed on the GUI.

3.2 FCFG Approach

One of the other approaches we have used for natural language to SQL conversion is using the nltk library to generate simple FCFGs to parse a natural language sentence and convert it to SQL. This approach works best for small databases. But large databases with many tables and columns lead to the creation of a very large FCFG which could become complex. Our aim would be to generalize the creation of a grammar similar to the sql0.fcfg[1] grammar from the nltk library so that it can create this grammar just by parsing the database. Then by using this generated grammar, we can generate the corresponding SQL query.

This approach that we are trying to implement follows the simple procedure.

- Get the details of the database and the database schema which will be used to form the grammar.
- Write the initial part of the grammar which includes rules for the column and table name.
- Parse the entire database and map all possible values in the database and write these new rules to the grammar.
- Using the final generated grammar, parse the natural language query and generate the SQL query.
- Run the generated SQL query and display the results.

The grammar generated using is as follows:-

```
% start S

S[SEM=(?sel + ?cn + FROM + ?tn)] -> SEL[SEM=?sel] CN[SEM=?cn]
TN[SEM=?tn]
S[SEM=(?sel + ?cn + FROM + ?tn + ?where + ?c)] -> SEL[SEM=?sel]
CN[SEM=?cn] TN[SEM=?tn] WH[SEM=?where] C[SEM=?c]
```

Figure 7: CFG Example 1

```
OP[SEM=(?op1 + ?op2)] -> OP[SEM=(?op1)] OP[SEM=(?op2)]
OP[SEM=''] -> 'to' | 'than' | 'in'
OP[SEM='='] -> 'is' | 'equal'
OP[SEM='>'] -> 'above' | 'more' | 'greater' | 'larger' | 'bigger'
OP[SEM='<'] -> 'below' | 'less' | 'lesser' | 'smaller'
```

Figure 8: CFG Example 2

The start of the grammar begins with S. There are two types of SELECT statements that we have implemented. The first one is the basic select statement which is of the format SELECT column_name FROM table_name. The second select statement is of the format SELECT column_name FROM table_name WHERE condition.

The CN describes the rules for column name. It can either have one column name or more than one column names separated by commas.

The C tells us about the grammar built for parsing conditions. We have implemented the grammar to incorporate one or more than one conditions. Our grammar also supports queries that have values in some range with the help of BETWEEN.

For accessing different values provided by the user, we converted the values to a different value so that it could parse any number and retrieve the results from the SQL database.

The OP defines the rules for operators like less than (<), more than (>), and equal(=). [6]

3.3 GUI

After performing either of the discussed approaches, then comes the section where user interacts with the application. In this section, description of how user will interact with the application is given and what results will be displayed is described with help of figures.

GUI for User Query Search:
The user will be provided with a GUI to enter the query in a natural language form. Once the query has been entered, the user can press the convert button. This will convert the natural language sentence to its corresponding SQL query and then execute the query on the database.[5]

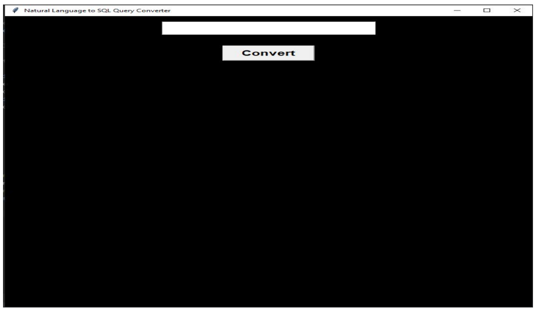


Figure 9: User Search GUI

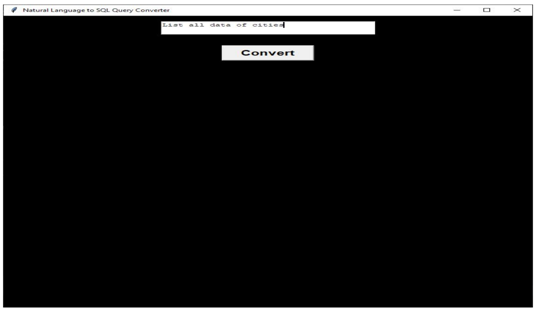


Figure 10: User Query 1

Query Execution:
Here the results of the query will be retrieved as will be displayed on the GUI, after the query formation procedure.

4. CONCLUSIONS

Although, first approach of maintaining dictionary and generating SQL queries is more flexible but lemmatization not completely handle the multiple synonyms of the word or the operators used in the SQL query dynamically, for that

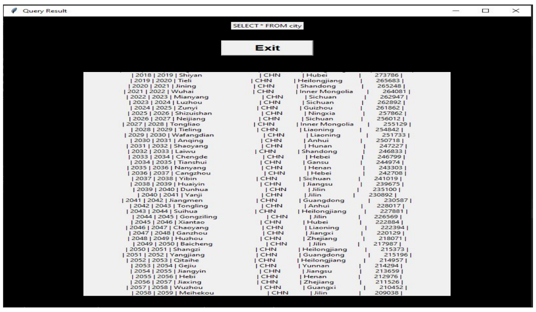


Figure 11: Query Result 1

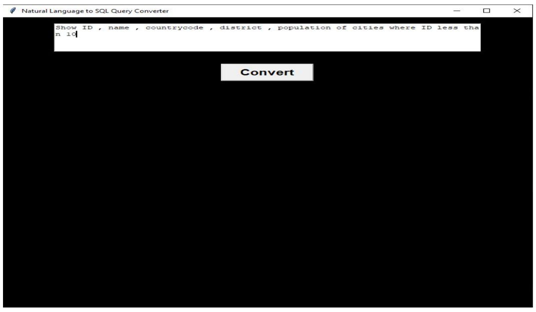


Figure 12: User Query 2

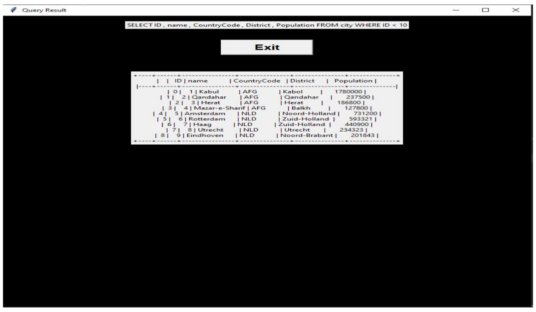


Figure 13: Query Result 2

Figure 14: User Query 3

we still have to maintain a dictionary. So, CFG approach is more rigid and specific to database but provide results as it can be used for multiple type of SQL queries, and for the other databases data, we just have to change the column and table names.

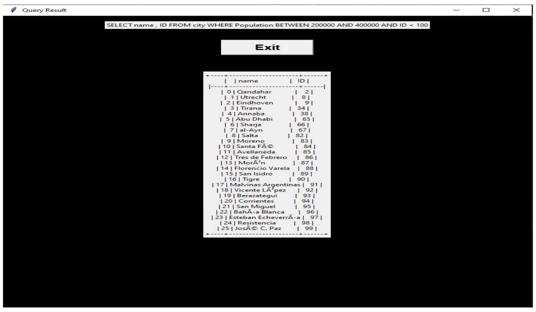


Figure 15: Query Result 3

Figure 16: Sample Search

Figure 17: Sample Result

5. LESSONS LEARNED

While doing this project, we found out new techniques apart from Tokenization and Part of Speech tagging, which can help us handle the natural language problems. The techniques like Stemming and Lemmatization, helps us to get the root word with the help of Lemmatization where Stemming removes the suffix part of the word. Also we learned that context free grammars (CFG) use in creating an expert system with the help of the rules which can be defined in CFGs.

6. FUTURE WORK

The future work for this project would include implementing the conversion of natural language to SQL for INSERT, DELETE and UPDATE commands. Another enhancement that could be done is to generalize the code more ensure that majority of the grammar is written automatically. Future work for the first approach could be finding a better way to map variations of all words in a better way.

7. REFERENCES

[1] Natural language toolkit¶.

[2] Setting up the world database :: 2 installation.

[3] The stanford nlp group.

[4] Stemming and lemmatization with python nltk.

[5] Python gui - tkinter, Jun 2017.

[6] S. B. Bhattacharya. How to match integers in nltk cfg?, Mar 1965.