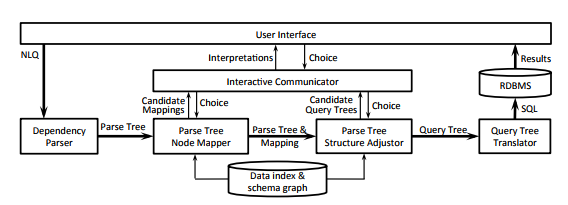
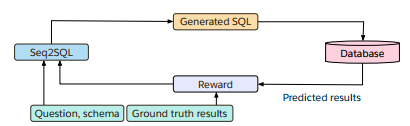
NLIDB Approach

**Recent Research:**



Fei Li, H.V. Jagdish – Their approach consists of generating multiple parse trees based on natural language query, and then presenting the user with options. After selecting options, parse tree is adjusted according to the user input and SQL is generated. (Simple approach)



SEQ2SQL – Uses LSTM model to generate queries, check queries against database and learn from the reward (comparison of result with ground truth) (Machine Learning approach)

**Current Approach**

Consists of using ElasticSearch + Python 2.7 to generate SQL queries.

On receiving the Natural Language Query, we use POS tags to determine the key words in the input.

These POS tags (or their synonyms) are likely to be the table/column in our dataset.

We use ElasticSearch to do an indexed search on the key words identified.

SQL generation takes place by filling in a standard template:

SELECT \_\_\_\_\_\_\_\_\_\_\_ FROM \_\_\_\_\_\_\_\_\_\_\_

WHERE \_\_\_\_\_\_\_\_\_\_\_ GROUP BY \_\_\_\_\_\_\_\_\_\_\_

HAVING \_\_\_\_\_\_\_\_\_\_\_ ORDER BY \_\_\_\_\_\_\_\_\_\_\_

Example: How many mascots?

Mascot is identified as column (POS tag is Noun). It is searched in Elasticsearch which returns a document consisting of column\_name, tableid, column\_index and column\_value. The tableid refers to the table name in WikiSQL. After select column, where\_column, from\_table and other phrases are identified they are clubbed under one query. Code will return all possible queries as per the data fetched from back end.

Sample ES document which is returned on searching:

{

"\_index": "final\_data",

"\_type": "final\_data",

"\_id": "AV-1G1y1lJTtyVfQ90P4",

"\_score": 8.878173,

"\_source": {

"column\_value": "Titans Hot Dogs Tigers Zebras Bronchos ",

"column\_index": 2,

"tableid": "2-18871102-3",

"column\_name": "Mascot"

}

Technical Steps [mostly in Parser.py ]

1. Sentence received, get POS tags
2. Search each word for column match in ES
3. Get tables for column match
4. Classify column as select or where column (first column identified is always select column)
5. Any words before matched column is select phrase and any words after is from phrase.
6. In select phrase, words are identified to see if any function needs to be applied
7. In where phrase, filter conditions are identified.
8. Join all phrases together in one final query
9. Run above steps [4-9] for each table received in step 3
10. Return list of queries generated

Assumptions:

1. NLQ needs to be in active voice form
2. Only 1 select column is present
3. Simple queries (no join queries implemented yet)
4. Only metadata search implemented, hence any data entity in a query (generally in the where phrase) is replaced by 00V.

Project Setup

Requirements:

1. Python 2.7
2. Flask (can install with pip)
3. ElasticSearch 5.6.3 as of writing this document
4. Kibana ( ES GUI for simplicity, not really part of project) 5.6.3
5. You may need to run pip install for several requirements as and when prompted

Setup ElasticSearch:

1. <https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-windows.html>

Download and extract ES from the zip file in a preferred directory

1. <https://www.elastic.co/downloads/past-releases/kibana-5-6-4>

Download and extract Kibana from zip file. (Make sure Kibana and ES version are same)

1. In kibana root directory, go to /config/kibana.yml and make the following changes :

elasticsearch.url: <http://localhost:9200> (line 21). Default ES port number is 9200, if you have changed it, you will need to make necessary changes in above yml file.

1. Go to localhost:5601 to confirm Kibana is working. Go to Dev tools to query ES.

Importing Metadata

Run the es\_import\_data.py file.

Running the project

To run the project, run python run.py command in root directory. Open localhost:5000 to enter natural language query and see generated SQL queries.

Potential Additions:

1. Implement full data search and take intersection of results
2. Ideas taken from TalkingDB project (which is not implemented, just ideas are posted)

[Import your data](https://github.com/TalkingDB/TalkingDB/wiki/Import-your-data-to-TalkingDB) to database. [Tell the topics your data is about](https://github.com/TalkingDB/TalkingDB/wiki/Tell-the-topics-your-data-is-about). Whether it is about cars? or food? or what?

Program will ask you multiple choice questions relating to your data. [Answer](https://github.com/TalkingDB/TalkingDB/wiki/Tell-the-topics-your-data-is-about) them. This trains the program and makes it intelligent to answer queries about your data.

Database was stored in a [graph data](https://en.wikipedia.org/wiki/Graph_database) format by TalkingDB. While storing your data TalkingDB had tried to understand your data based on the questions it asked you. TalkingDB came up with these questions automatically based on the topics you told it (TalkingDB automatically brought lots of knowledge of those topics from Wikipedia).

1. Bigram
2. NER

-Vocabulary can be built using table names, column names, and their synsets

1. How to solve pragmatist problem (map first\_name to people/person)
2. Manual walkthrough (problem of estimate rollout time)

Ultimate Aim:

LSTM model