Speech to SQL Generator - A VoiceBased Approach

Abstract:

Recent developments in speech recognition system and natural language processing have given rise to a new generation of powerful voice-based interfaces.
In this proposed system, a user interacts with the system through a voice-based user interface to fetch the desired results. To make the system smooth and interactive, the query is based on casual human speech / conversation. The spoken query undergoes many steps to arrive at the final results.
It includes synonym table which is used to convert the spoken query into SQL keywords. This proposed work will give the good results for simple and complex queries.
The accuracy of the system depends on the complexity of the database.
This system will show that the voice-based user interface is an effective means of querying and fetching data from the stored database.

Keywords: NLP, Speech, Speech to SQL, SQL, SQL generator, Synonym table.

Introduction:

Speech recognition has become an essential part of a system with the fast growing technology. These interactive applications should be able to process the human communicating language queries.

Natural Language Processing is the field of study that focuses on the interactions between human language and computers.

It sits at the intersection of computer science, artificial intelligence, and computational. The spoken query undergoes many steps to arrive at the final results.
Recent advances in voice-based interfaces, like Apple's Siri, Google assistant. When used with database systems, voice-based interfaces provide an effective way to query and fetch data. A major advantage over classical query interfaces or even touch-based visual interfaces is that voice-based interfaces are completely hands-free.

Literature:

Kumar *et al.* [6] proposed system which uses the knowledge of underlying database and generate lex file automatically, which will be used while tokenizing the words involved in English text query and since lex file contains underlying database information like column and table names. So, automatic generation of lex file helps in making the system database independent. In first phase of this work, speech is converted into text, Second phase analyses

the text whether it is syntactically correct or not based on grammar rules for valid queries, In third phase text is mapped into an intermediate query using lexer, parser and syntax directed translation, In fourth phase we extract the SELECT clause and WHERE clause from the intermediate query, In fifth phase we find all the required tables to form the FROM clause and thus SQL query is formed, In sixth phase formulated SQL query is fired to database and result is obtained. This System has been checked for single tables and multiple tables and it gives correct result if the input query is syntactically consistent with the Syntactic Rules.
Salma Jamoussi *et al.* [7] proposed statistical approach which constitutes the most used method for resolving the speech understanding problem. For this, they use a Bayesian network for unsupervised classification, called Auto Class and we expose three methods for the vector representation of words, these representations aim to help the Bayesian network to build up efficient concepts. We test this method on two applications data and we compare the Bayesian network performances with those obtained by the Kohonen maps and the K-means algorithm. Then, we will describe the last stage of our understanding process, in which we label the user requests and we generate the associated SQL queries. We use a speech recognition system to be able to treat sentences given in their signal forms. The system takes either speech or text as the query input.

Blunschi *et al.* [8] proposed SODA. Search Over Data warehouse enables a Google-like search experience for data warehouses by taking keyword queries of business users and automatically generating executable SQL. The key idea is to use a graph pattern matching algorithm that uses the metadata model of the data warehouse. The key idea of SODA is to use a graph pattern matching algorithm to generate SQL based on simple key words. Our experiments- with both synthetic data as well as with a large data warehouse of a global player in the financial services industry- show that the generated queries have high precision and recall compared to the manually written gold standard queries. As part of their future work they will evaluate the impacts of using DBpedia for matching keyword queries against various synonyms found in our classification. Since the use of DBpedia will naturally increase the number of possible query results- the query complexity, we will study more advanced ranking algorithms. Furthermore, the current GUI of SODA could be extended in several ways to engage the user in selecting and ranking the different results
Hendrix [4] proposed a system which was called LIFER / LADDER in 1978 which was one of the first good database NLP systems. It was designed as a natural language interface to a database of information about US Navy ships. This system, as described in a paper by Hendrix, used a semantic grammar to parse questions and query a distributed database. The LIFER / LADDER system could only support simple one-table queries or multiple table queries with easy join conditions.

Methodology:

The system accepts the spoken query as its input and it is sent to speech recognition engine, the output of that phase will be the input text query which is in the mixed format. The correct input query is extracted and further sent to tokenisation. Tokenisation is the method of splitting the sentence into individual words and storing it in the list. Unwanted tokens are removed after storing it in the list. The tokens are mapped with the pre stored synonym database which contains the words with its synonyms. The refined text is then sent to text translator. Text translator contains clause extractor and mapper. Through which an intermediate query is being generated and tokens are mapped with the table name and attribute. The result of this phase is the SQL query. This SQL query is operated on the database and correct results is being displayed on the interface. The SQL query will be displayed on to the command prompt.

The proposed model is composed of two modules, which allows user to communicate with the system. The primary module that is typically used whenever a user starts a new conversation with the system is the query module. The input for this module is called a query request. Basic Query Requests: In its current version, the system supports simple non-aggregation and aggregation query requests using one of the aggregation types (COUNT, SUM, AVG, MAX and MIN). In the current version of the system, basic query request can only contain one result column but allows multiple where clauses, and also multiple group-by columns. It is more convenient to the user when using a voice-based interface where results are displayed on the screen based on the speech request.

The most basic query request is of the following form:

Get the {column}(s) of {table}(s)?

What is the {aggregation} {column}(s) of {table}(s) ?

Here, {column}(s) {table}(s) are elements of the database schema, while {aggregation} refers to one of the spoken variants of the standard aggregation functions: i.e., "total", "average", "minimum", and "maximum".

In the following, we consider how where and group-by clauses can be formulated in a query request.

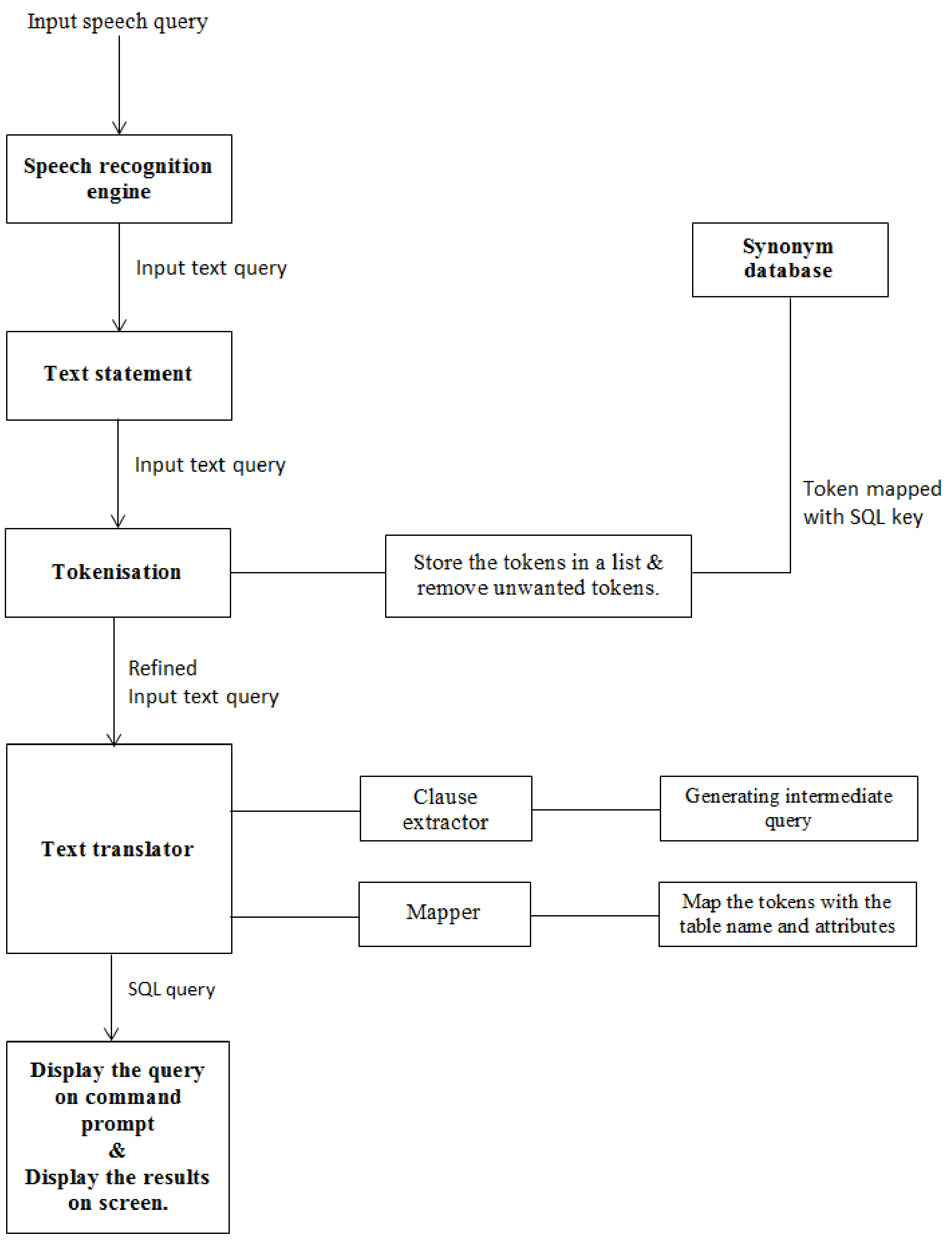
Multiple where Clauses: Where clauses can be added to a basic query request by appending:

Where, {table}(s) {column}(s) {Comparator}{value}?

Here, {column}(s) and {table}(s) refer to elements of the database schema, while {Comparator} and {value} refer to the comparison operator "is", "is equal to", "is greater than", "is less than", etc., and the value to compare the column against, respectively. When referring to a column the {table}(s) phrase is optional in all query requests. For example, two equivalent query requests that use a simple where clause is:

Get title of course with student whose name is Shravan?

In the system, multiple where clauses are also supported. Inner join operation takes place. The primary and foreign keys are compared to remove the redundant bits and correct result is being obtained.



Input speech query
Speech recognition engine
Input text query
Text statement
Input text query
Tokenisation
Store the tokens in a list & remove unwanted tokens.
Synonym database
Token mapped with SQL key
Refined Input text query
Text translator
Clause extractor
Generating intermediate query
Mapper
Map the tokens with the table name and attributes
SQL query
Display the query on command prompt & Display the results on screen.

Algorithm:

Step 1. Accept the input from the user in the form of speech.
Step 2. The speech is converted into text by using speech recognition engine.
Step 3. The correct form of the statement is saved and other statements are discarded.

Step 4. Divide the input query and store it in a list, i.e. tokenising the input query statement.
Step 5. Remove the unnecessary token from the list like, the, an, etc.
Step 6. Map the tokens with the table name and attributes of the database.
Step 7. Now find the tables which will contain the pair of ((attribute which do not belong to the table in the query), (other attributes present in the table in the query)).
Step 8. For a natural join, find out the common attribute and form the inner query. Then form the outer query according to the different conditions. Merge both of them and generate the final query.
Step 9. For a simple query, generate the final query by checking the all conditions.
Step 10. Obtain the conditions of the where clause and other clause such as comparators, aggregate function and relational operators, add these to the final query.
Step 11. Print the final query on command prompt and display results on UI.

Results and Discussion:

The results are drawn based on the query execution rate. The system is been tested on a college database containing 2, 3, 4 tables. The interface is the voice based interface which allow you to start speaking after a click on the button and 'done' is the word to end the speech. The results are displayed on the UI and the query is displayed on the command prompt.

The system is tested for 3 different database varying from 2 to 4 tables.

The efficiency of the system reduces as the number of tables increases.

The following types of queries are tested and results are being listed,
Type 1 - Simple query operation.
Type 2 - Join operation with no condition.
Type 3 - Join operation with multiple conditions.
Type 4 - Join operation with comparators.
Type 5 - Having and like clauses.
Type 6 - Multiple Join with multiple where conditions

References:

J. Weizenbaum, "ELIZA - A computer program for the study of natural language communication between man and machine," Communications of the ACM, vol. 9, no. 1, pp. 36-45, January 1966.
T. Winograd, "Procedures as a representation for data in a computer program for understanding natural language," *MIT AI Technical Report 235*, February 1971.
B.-B. Huang, G. Zang, and P. C.-Y. Sheu, "A natural language database interface based on probabilistic context free grammar," *IEEE International Workshop on Semantic Computing and Systems*, Huangshan, China, 14-15 July 2008.

G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, “Developing a natural language interface to complex data,” in ACM Transactions on Database Systems, vol. 3, no. 2, pp. 105-147, June 1978.

N. T. Dang, and D. T. T. Tuyen, “Natural language question answering model applied to document retrieval system,” World Academy of Science, Engineering and Technology, vol. 51, pp. 36-39, 2009.