Database Schema Independent Architecture for NL to SQL Query Conversion

Saima Noreen Khosa
NCBA&E
saimakhosa@yahoo.com

Muhammad Rizwan
Khwaja Fareed University of Engineering and IT
rizwan2phd@gmail.com

Abstract

NL (Natural Language) to SQL (Structured Query Language) query conversion overcomes communication gap between databases and non-technical user. It is very easy way to write question in NL and system automatically convert this question into a SQL query and show result to the user. Here we propose the database schema independent architecture for NL to SQL query conversion. User is free to give input in its own way in English. System is not restricted user to give input in a specific pattern. Existing proposed solutions are usually schema naming structure dependant and does not apply other than their specific schemas, but we propose a generic architecture which is independent of any particular schema and rather work on all schemas uniformly within the defined scope of the proposed architecture.

1. Introduction

For many long time computer programmers try to overcome the communication gap between causal user and computer. In most of places computer are used for sorting data and retrieval (according to the need and requirement of the user). During last decades there is lots of work done in NL (Natural Language) to SQL (Structured Query Language) conversion. NL is a natural language which is used in a daily life for the communication between human. And SQL is a query Language which is used to retrieve data from relational databases. The process of conversion of NL to SQL is divided into step by step levels. For example morphology deals with the meaning of smallest part of word. Lexical Level deals with sentence structure and tokenization. Here we also deal with the possible meaning and extract the one which is suitable for this use programmatic knowledge. Semantic and syntactic deals with one the limit of sentence but in some scenario the exact meaning comes beyond the limit of one sentence.

2. Related work

In this section we review the existing work regarding NL to SQL conversion.

2.1. Midway Query Generation

An already proposed system changes English statement enter by novice user in all midway query. So users can choice a one of that intermitted query [1] to find which one is more close to her requirement. After selection of final requirement, system fairs a SQL query. If any error occurs in system then user often frizzed. To overcome this problem writer gives a recommendation framework in future. Understanding of any language by machine is a difficult task. For this purpose use parsing rules which convert any natural language statement into computer understandable statement.

Generally NLP has following steps to process Natural language: first one is morphological analysis in which every single word is analyzed and split punctuation also. Then Syntactic analysis was checks the rules of language. If sentence are syntactically incorrect it will rejected. After syntax semantic are checked. Sometimes impact of previous sentence come on the upcoming sentence, it called Discourse integration and in last Pragmatic analysis phase comes. System facilitate user insert and delete value.[1]

2.2. Opportunity of modifies the system

Some systems give opportunity to user that they extend or modify system in any other language as English, German, and Greek. Under discussed system allow user to enter input near the SQL format. System gets three inputs. Firstly get SQL schema. Writer gives a specific pattern for make a file of SQL schema for the system. Second is SQL keys file. This file contains the relation between word phrases with SQL keys words. There is also a specific pattern to write this key file. Third input to the system is user query. User query enters also in a reserved given pattern. This is near to SQL query format, but not exact. [2]

After entering first two inputs in the system, Lexical Analyzer analyzes them. After that Lexical Analyzer give its output to the Syntax Analyzer. User query is also an input of Syntax Analyzer. Syntax Analyzer checks both inputs. After checking final process was held and SQL query is generated. Writer also makes it feasible about the extension process of the system. In which, system is extended easily in other languages also. For this proved a Key words file to the system. This file maps word phase of that language with SQL key words. And the remaining process are same as discussed earlier.[2]

2.3. Conversion of Urdu question into SQL

NLIDBs are one of the mechanisms which accomplish this task. User give input to NLIDB in its daily routine language and the answer is also given in same language. Authors discuss NLIDB for Urdu language. Algorithm discuss in this paper is efficiently maps the given Urdu question into SQL queries. Discuss algorithm implemented in c#.NET and test on student Information System and Employee Information System [3].

In this paper writer discuss some requirement of natural language interface to database. The query placed by user is either a request or question. Presented system evaluation the query must match one given category. It is also important to identify the rule of parameter in query. Parameters are table attribute values. Division of sentence is important for the understanding of computer. The parts of sentence are called tokens. To divide the sentence into tokens in called token formulation. After token formulation the process come is syntactic markers to make query semantically correct it is important to define and remove syntactic markers. It is important to extract the necessary parameters, for the successful translation of query. These parameters are table name, attributes and value.

Parser identifies the parameters and constructs. Construction of dictionary is important which keep the synonyms of columns and tables names. Inclusion of synonyms makes it possible for user to write a sentence in different natural way. The main propose of the constructed system is to track the correctness of query semantically. For this propose constructed a semantic dictionary. [3]

2.4. Natural Language Interface for DB

Generally computers are used to storing data and retrieve data according to the need requirement of the user. For the retrieval of data from the data base user employed SQL Language. If anyone works in relational databases then he/she must know SQL query structure how to write a query in SQL. Causal users don't know how to write query in database. Here writers proposed a Web Database System which creates an ease for novice user to access data from the database. Without having knowledge about SQL or internal structure of database, user just write query in natural Language and the system convert it into SQL query. Retrieve data and show result in natural language. This system gives a suitable answer for single database. This system restricted user to give input in a specific pattern. For example users enter a question system check that words are comprised in data dictionary of system if yes then further proceed and if not then show a massage to the user enter a right question. This is relevant to the database or data dictionary. After this, entering a correct form of input, system creates tokens of the input and removes extra words. There are some rules already defined. After removing extra words system mapping with that rules when rules are mapped a SQL query is created and run in the system and retrieve a required data from the database. And show arranged result to the user. [4]

3. Scope of Proposed System

In this paper we propose a generic system for NL to SQL conversion. Firstly we present the scope of the system for better understanding of proposed system architecture which is presented later on. As we know that scope of the system is very important to accurately understand the working of the system. Our system would work best by fulfilling the following constraint / scope w.r.t database and its schema:

1. System accepts only English language query.
2. Table names and column names should be complete English words within database schema. It should not be abbreviation or short word as it decreases the efficiency and accuracy of proposed system.
3. If any schema entity name is multiword, it should be connected with underscore.
4. Multiword name should have maximum two words.
5. Columns name should be unique within the schema.
6. System can accept only single line input query which can be extendable in future work.

4. Proposed System architecture

First we present architectural diagram of the proposed system as under:

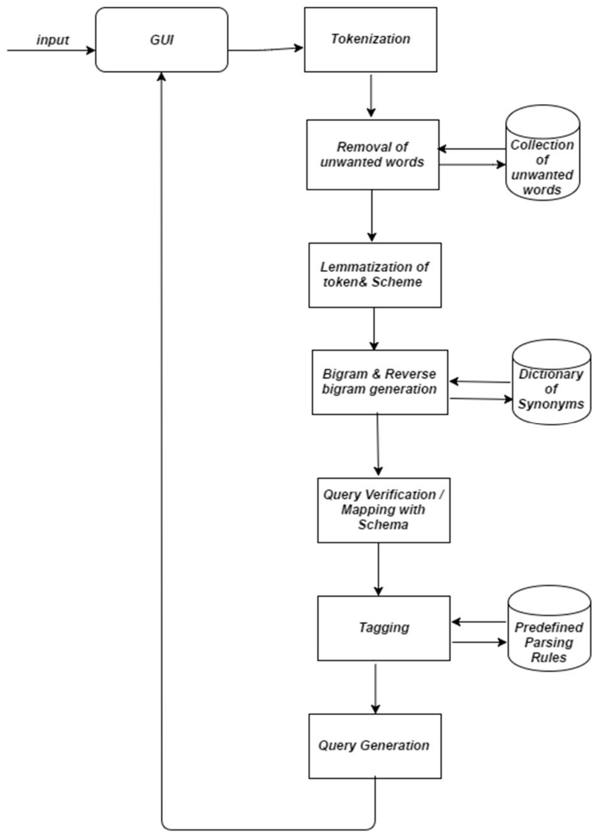
input
GUI
Tokenization
Removal of unwanted words
Collection of unwanted words
Lemmatization of token& Scheme
Bigram & Reverse bigram generation
Dictionary of Synonyms
Query Verification / Mapping with Schema
Tagging
Predefined Parsing Rules
Query Generation

Figure 1: Schema Independent NL to SQL System Architecture

As we can see the Figure 1, proposed system architecture is divided in this sequence of steps mention as follows:

- Tokenization
- Removal of unwanted words
- Lemmatization of query tokens as well as schema entities
- Generation of word bi-gram entities and their reverse combination
- Query verification / Mapping with schema
- Tagging
- Query generation

5.1. Tokenization

In tokenization phase we break text stream into single word tokens. Upcoming phases require morphology processing so we need to break up user input query into lexemes or single word tokens so that we could deal with each token separately. At this stage we not only tokenize the user input but also the schema entities name. Tokenization is very essential part of NLP. It helps to understand semantics of smallest parts of a sentence. Entered query spilt into single word tokens, for further processing. Consider the following example user input:

"Find out employees name whose salary is equal to 5000"

After tokenization, we will have following tokens with their corresponding indexes.

Table : 5.1

Token	Find	Out	Employees	Name	Whose	Salary	is	equal	To	5000
Index	1	2	3	4	5	6	7	8	9	10

The reason for generating index is that later on we can refer these tokens w.r.t to their index and this approach would also help to implement this system as well.

5.2. Removal of unwanted words

User enters query in natural language (we consider English language; other languages are not in the scope of this paper). After tokenization, the proposed system removes the unwanted words in the question from the user query. Only those words would remove which have no semantic importance regarding NLP to SQL query conversion. After filtering unwanted words e.g. and, or, is etc which should be predefined in "Escape Word Dictionary" this phase gives output as under for the same query cited in previous phase.

Table. 5.2

Token	Find	Out	Employees	Name	Whose	Salary	equal	5000
Index	1	2	3	4	5	6	8	10

Here token number 7 and 9 has been removed as they are escaped words.

5.3. Lemmatization of tokens and schema

As it is very difficult to exactly match the words with different forms so we need to perform lemmatization before any verification with schema. Lemmatization means to cut off the last part of the word which occur in different shapes & spellings, so that all form of the word within particular text would be same.

e.g.

am, are, is → *be*

Car, Cars, Car's → *Car*

The target schema keywords also need to convert in lemmatized form so that the verification process in the upcoming verification phase would give us the maximum accuracy. Now by applying the lemmatization on Table 5.2, the output table would be as under:

Table :5.3

Token	Find	Out	Employee	Name	Whose	Salary	equal	5000
Index	1	2	3	4	5	6	8	10

Form above table we can see that Employees converted to Employee after lemmatization, and rest of the words already in lemmatized form, so there is no change in these words.

5.4. Generation of Bi-Gram and Reverse Bi-Gram

In our scope we already mentioned that schema entities having two words attributes or table name are allowed. So in the upcoming verification phase we also need to match bi-gram query tokens with bi-gram database schema entities tokens. We will generate dictionary of Bi-gram and their reverse combination. The bi-gram tokens of synonyms would also be generated. The wordnet can be very handy for this task[8]. e.g.

Table 5.4 .Bi-Gram from Query Tokens

find_out	out_employee	employee_name	name_whose	whose_salary
1_2	2_3	3_4	4_5	5_6

salary_equal	equal_5000
6_8	8_10

Table 5.5. Reverse Bi-Gram of Query Tokens

out_find	employee_out	name_ employee	whose_ name	salary_ whose
2_1	3_2	4_3	5_4	6_5

equal_ salary	5000_ equal
8_6	10_8

5.5. Query verification / mapping with Schema

In this phase we have to verify schema entities tokens (table names and column name) as per the output of previous phase with the query tokens. We also extract synonyms of each lemma and make lemma dictionary of use input. We then generate bi-gram and their reverse combination lemma and add in the dictionary too. After that we verify each token Bi-grams and their combination with the lemmatized schema. So that we can identify the columns and tables name in the user input. We tag each column and table in the input which is exactly matched with the lemmatized schema.

5.6. Tagging

After verification we tag lemmatized tokens for the use of next phase and to identify context of that particular token with schema. Tagged can be done with respect to three ways:

i. Attribute or Column tag
ii. Table tag
iii. Index Number

Attribute tag shows that word is an attribute or column in that schema, table tag shows that word exist in the schema as table. Index number shows the indexing number in tokenized array. e.g

Table 5.6

Token	*Tags*
Employee_Name	Attribute
	Employee (Table Name)
	3_4 (index number)

Table 5.7

Token	*Tags*
Salary	Attribute
	Employee (Table Name)
	6 (index number)

From table 5.6 & 5.7, we can see that we assign each uni-gram and bi-gram token to their corresponding tags after verification. These tagged_tokens would be the output of this phase and would help to make actual SQL query fragments against the user input in upcoming phase. We also tag numbers as "cardinal number".

There are many scenarios in natural text which can be tagged / identified by using state-of-the-art NLP algorithms and tools such as Apache Open NLP [5], Stanford NLP [6], LingePipe [7] which provides implementation of state-of-the-art algorithms of tokenization, POS tagging, sentence splitting, named entity extraction and different NLP task. Form future point of view, we can also tag named entities (date time and place), conditional words and booster words as the field of NLP has significant success and accuracy in these areas.

5.7 Query Generation

5.7.1. Fragment Creation

In this phase different fragments of SQL query are generated against the natural language query.

SQL query "where" part is extracted and matched from each fragments and their corresponding rules. In this phase, we create fragments which consist of tagged words and suffix and prefix of tagged word. We do not include suffix and prefix if it is another tag word. Tag word in the query which refers to the column or table of target schema.

5.7.2. Query Parts Creation

In this phase we try to create different query parts from fragments. FROM parts is identify based on table tagged tokens. If multiple tables exist then we try to join them based on some common attribute between two successive tables. If no table exists then we try to match attribute in all table of schemas.

"Select" part of SQL query is extracted based on all attribute tagged tokens. If there is no tagged attribute then * is considered as select part. The "select" and "from" clause can be extracted by the help of tagged attributes and tagged tables.

5.7.3. Parsing Rules Mapping

We create some morphological rules. We map these rules with tagged input. We extract different fragment from the input which match according the rules. There are different types of fragments like WHERE clause, SELECT clause, FROM clause. The following table shows the rules for "where" clause of SQL query.

Table 5.8

	Scenario	SQL condition	*<A> stands for attribute <CN> stands for cardinal number <BW> stands for booster word*
1	<A><CN>	<A> = <CN>	
2	<A>greater than<CN>	<A> > <CN>	
3	<A>less than<CN>	<A> < <CN>	
4	<BW><A>	<BW><A>	
5	<A>equal to<CN>	<A> = <CN>	

From table 5.8, we can see that what possible parsing rules can be extracted from user natural text input.

According to each fragment and its type we generate SQL fragments. By combing all SQL fragments we can finally generate complete SQL query.

6. Conclusion

This paper shows that the concept of NL to SQL query conversion in broader scope w.r.t multiple database schemas support because other existing system depend on specific single schema which narrow their scope but our proposed system works schema independently. System gets input in plain English language. User is not restricted to follow any pattern. Advancement in this field can give benefit to large number of novice user or businessman who wants to directly explore their organization databases without technical knowledge of SQL.

In future work we try to modify the architecture so that it might have multilingual support and more sophisticated morphological rules with latest NLP advancement which can map more SQL fragment like "group by" & "order by" etc as well.

7. References

[1] Bhadgale, Anil M., et al. "Natural language to SQL conversion system." IJCSEITR 3.2 (2013): 161-6.

[2] Papadakis, Nikos, Pavlos Kefalas, and Manolis Stilianakakis. "A tool for access to relational databases in natural language." Expert Systems with Applications 38.6 (2011): 7894-7900.

[3] Ahmad, Rashid, Mohammad Abid Khan, and Rahman Ali. "Efficient Transformation of a Natural Language Query to SQL for Urdu." Proceedings of the Conference on Language & Technology. 2009.

[4] Alexander, Rukshan, Prashanthi Rukshan, and Sinnathamby Mahesan. "Natural Language Web Interface for Database (NLWIDB)." arXiv preprint arXiv:1308.3830 (2013).

[5] OpenNLP, Apache. "a Machine Learning Based Toolkit for the Processing of Natural Language Text." URL http://opennlp. apache. org (Last accessed: 2016-09-18).

[6] Manning, Christopher D., et al. "The Stanford CoreNLP Natural Language Processing Toolkit." ACL (System Demonstrations). 2014.

[7] Carpenter, Bob, and Breck Baldwin. "Text analysis with LingPipe 4." LingPipe Inc (2011).

[8] Pedersen, Ted, Siddharth Patwardhan, and Jason Michelizzi. "WordNet:: Similarity: measuring the relatedness of concepts." Demonstration papers at HLT-NAACL 2004. Association for Computational Linguistics, 2004.