

# Question to Query: Converting Human Language to DBMS Query



Yashvi Thakkar, Faiz Palwala, Utsav Vyas, Krati Agarwal, RajeshKannan Regunathan

**Abstract:** In this paper a method has been proposed keeping in the mind the need for systems that could generate structured queries from normal language keeping in mind that the user has no prior knowledge of database query language. A novel method which aims at aiding analyst who aren't well versed with codes, but need quantitative outputs to analyze, predict and alert the business or market. A python model is used, which aims at converting any sentence typed in English to a query provided that such tables and database is present for query processing. Tree tagging is used here to relate words typed in to SQL query syntax. Any sentence typed in by analyst, it further annotated by parts of speech and lemmas. A list of generic words and stop words is used while parsing the input the sentence and tagging it. Query is generated by simultaneously removing the stop words, mapping the keywords with the one's used in structured query language. The generated query comes out in form of a JSON file.

**Keywords :** Complex SQL generation, Natural Language Processing, Query parsing, Structured query language ,Tree tagging.

## I. INTRODUCTION

We live in a data savvy world, today with over a billion connected devices today the amount of data produced daily has been increasing exponentially. Every device generates data which can be analyzed to give multiple insights into its environments. This data is mined to get invaluable insights to the connected world. Most of this data goes into relational databases like Oracle databases, MySQL, etc.. Not all of this data though, is used for analytical purposes. Now, when we think of any Multi National Company querying and handling data in said Databases, they can just choose to employ the expertise needed to use SQL. But in this new age, with the growth of smart phones, and with the idea that anyone can be an entrepreneur, which results in more and more people from different backgrounds trying to run a business on a digital platform. This involves using their data to make informed decisions regarding their businesses for making corrective

measures or better policy decisions. This ultimately requires them to query and use the data that they have acquired and have the permission to use for their economic gains.

Similarly, an accountant may need to query a database regularly as a part of their job but may not be fluent in SQL query formation. Despite the meteoric rise in the popularity of data science, most people do not have adequate knowledge to write SQL and query their databases for their own data. They lose out on advantages like better risk prediction and analysis, customer requirement understanding and flexibility to change in this dynamic corporate world. Moreover, most of these people are unable to find adequate time to learn and understand SQL and to be proficient in it, as it takes a sufficient amount of time. To add to that a lot of them cannot afford to employ someone who is an expert in database management. Even for SQL experts, writing similar queries, again and again, is a tedious task and is not what they are expected to do or what they expect their job to be. Due to this fact, the vast amount of data available today cannot be effectively accessed especially due to the inefficiencies in database management. A solution like the one we are suggesting will help bring, complex data analytics to the masses, by removing the flood gates in syntaxes and querying via natural language interfaces to databases. The goal is to allow you to talk to your data directly using human language. The user is not expected to have any knowledge of SQL or database management for the matter, he/she is just expected to be used to writing queries in a structured manner. The user is expected to type in his query in natural language and the output is the required SQL query. This query can be used to retrieve the relevant data. Thus, these interfaces should help users of any background to easily query their databases and analyze a vast amount of data and use it for further business decisions.

## II. LITERATURE SURVEY

IlyaSutskever uses pattern matching techniques to give an output sequence after reading an input sequence. She proposes to use a simple sequence to sequence (Seq2Seq) model. The input sequence is mapped to a fixed dimensional vector using multi-layered Long short term memory network or LSTM taking in inputs one at a time. The output sequence is then obtained by the fixed dimensional vectors using another LSTM. The input sequence ends with a <eos> token, which indicates the model to start decoding. [1]Rajender Kumar and MohitDua worked with a controlled language and used it to generate SQL queries. Using controlled Hindi language helps eliminate ambiguity and reduces the complexity of queries that the system can process.

Revised Manuscript Received on December 30, 2019.

\* Correspondence Author

**Yashvi Thakkar**, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore

**Faiz Palwala**, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore

**Utsav Vyas**, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore

**Krati Agarwal**, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore

**RajeshKannan Regunathan**, School of Computer Science and Engineering Vellore Institute of Technology, Vellore

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an [open access](http://creativecommons.org/licenses/by-nc-nd/4.0/) article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

The system uses a morphological analyser coupled with a word group analyser to obtain the keywords from the Hindi query (controlled). This is followed by pattern matching for finding the type of keyword. [2] Debarati Ghosal, Tejas Waghmare, Vivek Satam and Chinmay Hajirnis proposed a system that uses AI as in an expert system for converting simple NL Query to a SQL Query. It gives the users all the probable intermediate queries, the user is expected to select the appropriate intermediate query. The system then generates a SQL query from the selected intermediate one. In the end, this query is executed and the user gets the output. [3] Arun and Garima proposed a system that makes use of rules which have pre defined structure sets used in Natural Language Processing. These sets have the capability to expand and include new knowledge. These include noun set, verb set, escape word set, relation set, ambiguity check set, etc. The approach builds on NLQ Interface to Database System (NLIDBS) with the ability to tackle complex queries and ambiguity removal, and is named NL Converter to SQL Converter. The proposed system achieves a better accuracy and precision score and also has a lower error rate. [4] Abhilasha Kash, Satish Kumble, Aishwarya Bodkhe, Mrunal Joshi in their paper, introduced an automated system for converting NL Query to SQL Query. The user can enter the query in the text format or in the form of speech and that sentence will be split into words based on the empty spaces. Taking all the words (tokens) it removes all the tokens when compared with the ignored list. The selected words (tokens) are mapped with the dictionary, other synonyms and word type and finally result in forming the approximate query and also adding if any particular conditions which are specified by the user. Once the query is generated it is executed and the output is returned. They designed this system for the Training and Placement cell officers who predominantly work on student databases but don't have much knowledge about SQL. [5] Rodolfo A. Pazos R, Marco A. Aguirre L, Juan J. González B, José A. Martínez F, Joaquín Pérez O and Andrés A. Verástegui O proposed a new methodology for processing the natural language query in their paper. They consider a NLIDB and divide the whole system into different functionality layer translation modules. Firstly it takes the Natural Language Query and lexical tagging the key words in it. Then it generates the syntax tree for the tagged words in the canonical order and further generating the complete semantic graph for it. Thus, converting to the simple SQL query and processing it for the output. [6] Darryl Jon Mocek, Keister Li, Jonathan Micheal Levine have proposed a system having data display device which accepts the user selected database command and process it to produce the natural language translation. This is displayed on the device screen and the user is provided with the additional option of adding some additional commands at any point of time and interpret the command on the SQL window and generate the output of the query. If the given commands have any error, the system will generate an error message and also helps in providing assistance for the database query. It can be used in multiple devices like machine control functioning in industries, etc. [7] Ian M. Bennett has proposed a system which uses natural language routines to identify word phrases which are presented in the user based query and provide context of it. This helps in loading the approximate grammar or dictionaries and map the word phrases with them. Thus, converting it into the query with more potential. In order to identify the answer for the

query, the routine compares the words phrases which are present in the query and maps it with potential query answer pairs. This turns out to give the best effective solution almost every single time. [8]

### III. METHODOLOGY

In this paper the base language is defined using query itself, for test purposes English is considered. The proposed tool will then extract, or for a more technical term parse through the user input with the help of python parsing library to extract meaningful lexicons, it can be related to semantic analysis process in the NLP paradigm. The structure along with the necessary information is stored, a language store/thesaurus in form of a CSV file is used to find a match between keywords extracted from the user and the entities within the SQL, which results in selection on the type of query be it select, delete, average, sum, count, max, min. A second search follows to determine the table in query. All the previous found information is finally structured in form of query (i.e. syntactic analysis), the query is verified by a dry run execution, upon confirmation it is output in the form of a JSON file. Fig 1 describes the system flow which the model proposed in the paper follows. It aims at simplifying the conversion process and attaining maximum possible accuracy. The steps are as follows:

#### A. Extracting Meaning

The idea is to recover only the meaningful words in the sentence entered by the user. Indeed, it is important to be able to subsequently perform a correspondence (matching) between certain words entered by the user and entities of SQL. For this purpose, therefore, the common phrases are preserved. I.e. can be the name of a table or a column, but also the proper names, numbers, adjectives, etc. Empty words are filtered out according to their class (prepositions, pronouns, determinants, etc.) and rooting (stemming) of the remaining words is performed. Considering the sentence: How old are the students whose first name is Jean? The filter should return the elements: " age, student, first name, Jean". The word order is preserved and is important in future steps.

#### B. Recovery of the database architecture

This step involves resolving the structure of the database that we are querying in hopes of knowing the entities (columns, tables, primary and secondary keys, etc.) to be used in a second search with the words extracted from the user request during the section 3.1. This paper is suggesting two approaches to achieve this result. The first method is to collect the information needed by querying the database using SQL queries like "SHOW TABLES, SHOW COLUMNS, DESCRIBE, etc.". The second method is to analyze the backup or database creation file.

#### C. Coupling to a thesaurus

As the study by (Mohite & Bhojane, 2014) points out, if the user does not enter a correctly orthograph or whose vocabulary used is not strictly the same as that of the database, no match between his sentence and elements of the base will be found and no relevant results will be returned. Therefore, it is important to maximize the number of words that will result in a relevant match between a key word of the input request and an element of the DB.

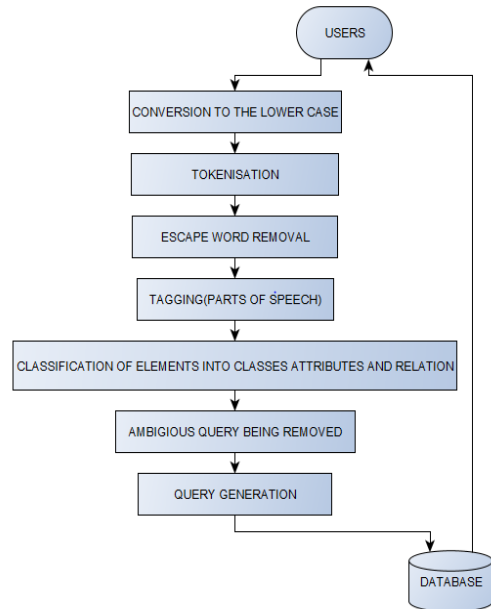
To this end, in parallel with the process mentioned in the previous sections, a thesaurus is loaded. For each word, we have access to a table of concepts containing all the words of the language by which it can be substituted. For example, the words "students" and "student" represent the same concept. A concept is an idea, a meaning represented by a word or a group of words. From then on, a concept is represented by a lexical meaning word as well as all possible substitution words contained in its table. The purpose of this translator is to make the query of a database accessible to a person who does not know either the structure and keywords (names of tables and columns) and therefore being able to use a synonym for a word used in the base instead of the word itself. It is therefore more judicious to represent a word by a concept, a table of all the words by which it can be substituted (a table of its synonyms, including him) rather than only by himself. This way to query the table the user may enter the word student.

**D. Cutting demand**

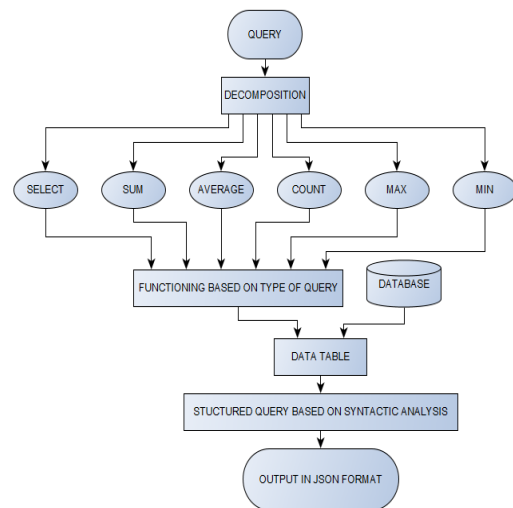
At this stage of the process, each key word of the request entered by the user is extracted. Along with a list of synonyms for each of these keywords. The idea now is to look for a match between the key words of the application (or their synonyms) and the entities of the database in order to perform a segmentation of demand according to the correspondences found and thus to know at best the structure of the request to generate. When matching, all words are put in lowercase and all diacritic characters (accents etc.) are standardized. Each keyword found is tagged according to whether it is a column or a table the database being queried, or something else still unknown at the moment.

**E. Determining the structure of the request & query generation**

Then, in each of the segments obtained during the division (section 3.4), we analyze the key words tagged until unknown present. These words can be factors of presence of a query called counting, algebraic calculations, of a negation, etc., or even of a value on which make a constraint. In this way, if a word referring to the count as for example "how many" is found in the first segment of the sentence, the one corresponding to the SELECT, the system identifies the request to be generated as being a count request, that is to say a SELECT COUNT (\*), works the same way. In this paper we address only the inner joins (INNER JOIN). Two types of inner joins exist, implicit and explicit. When there is a selection or a constraint on a column that is not part of the FROM table, ie when the table to which the targeted column belongs is not mentioned in the sentence entry is an implicit join. In this case, make a join between the table of the target column and the table FROM which is specified in the sentence. In the case of an explicit join, the table on which we must do join is specified directly in the sentence.



**Fig. 1. Flow Chart**



**Fig.2. Query Processing**

**IV. PSUEDOCODE**

**A. Query Generation**

```

START
Select Class based on call from parser
column = selection[0]
column_type = selection[1]
if else ladder to build column string
Select if else ladder based on dB dependency
Concatenate to column string
Loop in range (0,len(column))
if i == (len(columns) - 1)
    string = string + str(column(columns[i]))
else
    string = string + str(column(columns[i])) + ', '
  
```



# Question to Query: Converting Human Language to DBMS Query

```
return query type type and string
Value returned checked in SQL console:
if Error
    Go back to Parser.py
else
    Func Print Json
funcprint_json // adding spaces specific to query type
Print output.json
STOP
```

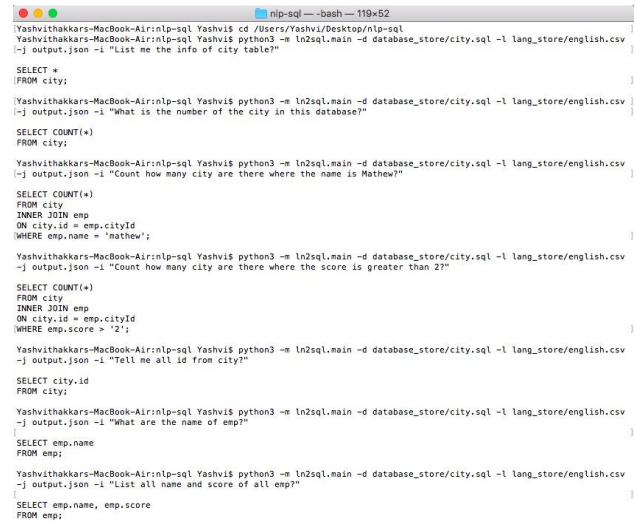
## B. Database creation

```
START
Import data
Table (columns, equivalences)
while (True)
    INSERT column name
    INSERT number of columns
    RETURN columns
    ADD columns
GET equivalences
    ADD equivalences
    isequivalences (word)
        if words in equivalences
            return True
        else
            return False
GET primary key
    primary keys->[]
    ADD primary keyif column in columns
        column name->primary key column
    SET primary key
GET foreign key
    foreign keys->[]
    ADD primary key(column name, foreign table, foreign
column)
        if column in columns
            SET foreign key as foreign table and foreign column
as foreign column
    if columns (False)
        columns = []
    if equivalences (False)
        equivalences = []
STOP
```

## V. IMPLEMENTATION

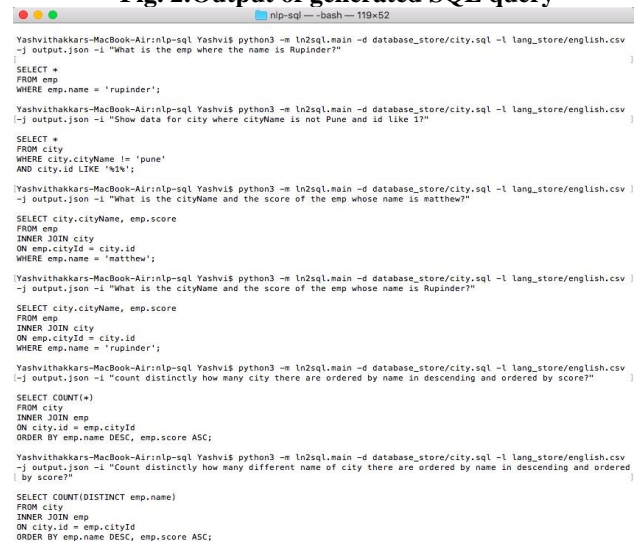
After performing the above mentioned steps, the following screenshots were obtained. In this paper we have discussed, how natural language can be processed and converted to SQL queries as follows. Figure 2, 3, and 4 have all possible combinations of prompts that any HR/analyst can use and fetch the following outputs as obtained. The prompts that covered here include basic SQL commands which include data query and data control language which includes the following commands: SELECT, ORDER BY, COUNT, SUM, AVG, MIN, MAX, INNER JOIN, DISTINCT. The model converts natural language accurately into structured queries. Table 1 shows a few input questions and the output

generated by the proposed model. It accurately converts human language into structured query language.



```
nlq-sql -- bash -- 119x52
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ cd /Users/Yashvi/Desktop/nlp-sql
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "List me the info of city table?"
SELECT *
FROM city;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "What is the number of the city in this database?"
SELECT COUNT(*)
FROM city;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count how many city are there where the name is Matthew?"
SELECT COUNT(*)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
WHERE emp.name = 'matthew';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count how many city are there where the score is greater than 2?"
SELECT COUNT(*)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
WHERE emp.score > '2';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Tell me all id from city?"
SELECT city.id
FROM city;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "What are the name of emp?"
SELECT emp.name
FROM emp;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "List all name and score of all emp?"
SELECT emp.name, emp.score
FROM emp;
```

Fig. 2. Output of generated SQL query



```
nlq-sql -- bash -- 119x52
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "What is the emp where the name is Rupinder?"
SELECT *
FROM emp
WHERE emp.name = 'rupinder';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Show data for city where cityName is not Pune and id like 17"
SELECT *
FROM city
WHERE city.cityName != 'pune'
AND city.id LIKE '%17%';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "What is the cityName and the score of the emp whose name is matthew?"
SELECT city.cityName, emp.score
FROM emp
INNER JOIN city
ON emp.cityId = city.id
WHERE emp.name = 'matthew';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "What is the cityName and the score of the emp whose name is Rupinder?"
SELECT city.cityName, emp.score
FROM emp
INNER JOIN city
ON emp.cityId = city.id
WHERE emp.name = 'rupinder';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count distinctly how many city there are ordered by name in descending and ordered by score?"
SELECT COUNT(*)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
ORDER BY emp.name DESC, emp.score ASC;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count distinctly how many different name of city there are ordered by name in descending and ordered
by score?"
SELECT COUNT(DISTINCT emp.name)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
ORDER BY emp.name DESC, emp.score ASC;
```

Fig. 3. Output of generated SQL query



```
nlq-sql -- bash -- 119x52
ORDER BY emp.name DESC, emp.score ASC;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count distinctly how many different name of city there are ordered by name in descending and ordered
by score?"
SELECT COUNT(DISTINCT emp.name)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
ORDER BY emp.name DESC, emp.score ASC;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "What are the distinct name of city with a score equals to 8?"
SELECT DISTINCT emp.name
FROM city
INNER JOIN emp
ON city.id = emp.cityId
WHERE emp.score = '8';
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count how many city there are ordered by name?"
SELECT COUNT(*)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
ORDER BY emp.name ASC;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$ python3 -m ln2sql.main -d database_store/city.sql -l lang_store/english.csv
- j output.json -i "Count how many city there are ordered by name in descending order and ordered by score in ascending
order?"
SELECT COUNT(*)
FROM city
INNER JOIN emp
ON city.id = emp.cityId
ORDER BY emp.name DESC, emp.score ASC;
Yashwithakkars-MacBook-Air:nlq-sql Yashvi$
```

Fig. 4. Output of generated SQL query



**Table 1. Test cases with the output generated**

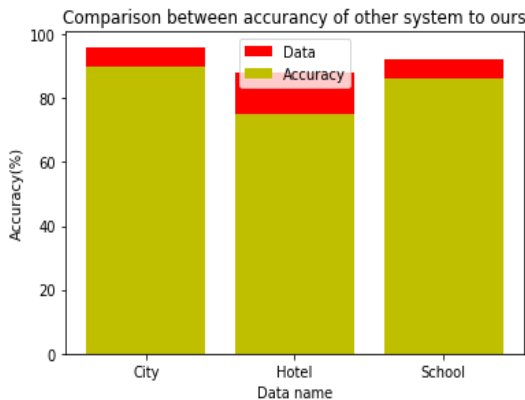
S No.	Test Cases	Output
1	What is the number of the city in this database?	SELECT COUNT(*) FROM city;
2	Count how many city there are where the name is Matthew?	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId WHERE emp.name = 'Matthew';
3	Count how many city there are where the score is greater than 2?	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId WHERE emp.score > '2';
4	Tell me all id from city.	SELECT city.id FROM city;
5	What are the name of emp?	SELECT emp.name FROM emp;
6	List all name and score of all emp.	SELECT emp.name, emp.score FROM emp;
7	What is the emp with the name is Rupinder?	SELECT * FROM emp WHERE emp.name = 'Rupinder';
8	Show data for city where cityName is 'Pune Agra'	SELECT * FROM city WHERE city.cityName = 'pune agra';
9	Show data for city where cityName is not Pune and id like 1.	SELECT * FROM city WHERE city.cityName != 'pune' AND city.id LIKE '%1%';
10	What is the cityName and the score of the emp whose name is Matthew?	SELECT city.cityName, emp.score FROM emp INNER JOIN city ON emp.cityId = city.id WHERE emp.name = 'Matthew';
11	What is the cityName and the score of the emp whose name is Rupinder?	SELECT city.cityName, emp.score FROM emp INNER JOIN city ON emp.cityId = city.id WHERE emp.name = 'Rupinder';
12	Count distinctly how many city there are ordered by name in descending and ordered by score?	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name DESC, emp.score ASC;
13	Count distinctly how many different name of city there are ordered by name in descending and ordered by score?	SELECT COUNT(DISTINCT emp.name) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name DESC, emp.score ASC;

## Question to Query: Converting Human Language to DBMS Query

14	What are the distinct name of city with a score equals to 9?	SELECT DISTINCT emp.name FROM city INNER JOIN emp ON city.id = emp.cityId WHERE emp.score = '9';
15	Count how many city there are ordered by name	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name ASC;
16	Count how many city there are ordered by name in descending order and ordered by score in ascending order	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name DESC, emp.score ASC;
17	Count how many city there are ordered by name in descending order and ordered by score?	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name DESC, emp.score ASC;

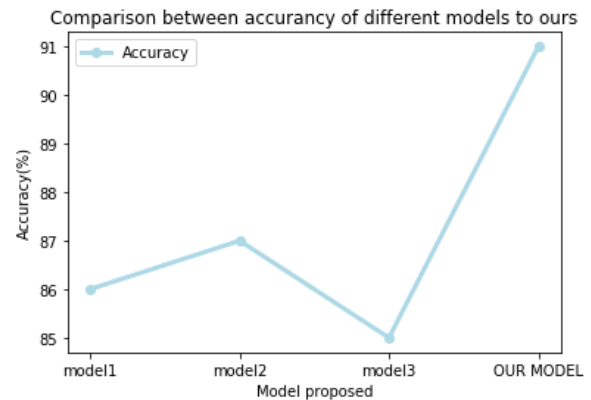
### VI. RESULT AND DISCUSSION

The database that has been used in order to test the proposed model has three tables – City, School and Hotel. City table has attributes like id, cityName, cityId, Score where id is the primary key. Table School has two attributes – id, idClass, classroom, idProf, idField with idClass as the primary key and id as the foreign key from city table. Table Hotel has idRoom, clientName, age, phoneNo and address with idRoom as the primary key. A lot of questions with all possible synonyms like – “What is Mathew’s score?” or “How much did Mathew score?” or “Find Mathew’s score?” In the graph below (figure 5) the performance of our system on three different data sets which are city data, hotel data and school data has been measured and is visualized through image that in all the three datasets our system is much more accurate than the previous system (the difference is highlighted with the help of red color and previous model performance is shown with yellow color). Thus the accuracy percentage obtained for these three cases will be 94, 88 and 92 respectively giving an average of 91.33 percent as shown in the graph.



**Fig. 5. Accuracy Graph showing number of correctly generated queries**

The graph below (figure 6) shows the accuracy rate of different model, as from the image we can see that the model 1 have the accuracy of 86%, second model (model 2) showcasing little better result have the accuracy rate of 87%, third model (model 3) having little less accuracy as compared to the previous model but “OUR MODEL” has an average accuracy rate of 91.3% which is far better than the previous models, hence proving its quality.



**Fig. 6. Accuracy comparison**

### VII. CONCLUSION

This paper successfully deals with nested queries apart from generating data control and query language. This application could help millions of analyst who face trouble in fetching useful information due to inability to code or waste time in learning it. In future works, this model can be extended to deal with mute constraints, keeping the verbs as key words. In addition, it is intended to detect the language of the request entered by the user to use a thesaurus related to this language and adjust the rules according to the language

so as to make a robust system with languages other than English. To conclude, although perfectible, this approach makes it possible to query any SQL database with an appreciable accuracy, thus meeting the fixed portability objectives, while keeping performance in the average of applications already and covering a wide range of selection operations.

## REFERENCES

1. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014), "Sequence to sequence learning with neural networks", In Advances in neural information processing systems, pp. 3104-3112.
2. Rajender Kumar, Mohit Dua "Translating Controlled Natural Language Query into SQL Query using Pattern Matching Technique", IEEE 2014.
3. Prof. Debarati Ghosal, Tejas Waghmare, Vivek Satam, Chinmay Hajirnis "SQL query formation using natural language processing", International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 3, March 2016.
4. Singh, G., & Solanki, A. (2016). An algorithm to transform natural language into SQL queries for relational databases.
5. Rodolfo A. Pazos R. Marco A. Aguirre L. (2016) Comparative study on the customization of natural language interfaces to databases
6. Abhilasha Kate, Satish Kamble (2018), Conversion of Natural Language Query to SQL Query, IEEE Xplore digital library
7. Bennett, I.M., Bennett Ian M, 2010. Systems for natural language processing of sentence based queries. U.S. Patent Application 12/559,347
8. Mocek, D. J., Li, K., & Levine, J. M. (1999). U.S. Patent No. 5,924,089. Washington, DC: U.S. Patent and Trademark Office.
9. Kovács, L. (2009). SQL generation for natural language interface. Journal of Computer Science and Control Systems, 2(18), 19-22.
10. Kaur, S., & Bali, R. S. (2012). SQL generation and execution from natural language processing. International Journal of Computing & Business Research ISSN (Online), 2229-6166.
11. M. Auli, M. Galley, C. Quirk, and G. Zweig. Joint language and translation modeling with recurrent neural networks. In EMNLP, 2013.
12. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

## AUTHORS PROFILE



**Yashvi Thakkar** is currently pursuing BTech (final year) in Computer Science and Engineering at Vellore Institute of Technology, Vellore notably worked on Spot A Spot - Using Single Shot Multibox Detector, Springer AISC series and is interested in fields of business analytics and data science.



**Faiz Palwala**, is pursuing B.Tech Computer Science at Vellore Institute of Technology. He is passionate in tech pursuing projects in Data Sciences and NLP, notably worked on an Alexa Skill to bring knowledge base of services like Wolfram to the masses in a conversational form.



**Utsav Vyas** is a final year student from Vellore Institute of Technology, Vellore. He counts Mathematics, Statistics, Machine Learning and Java as his favorite subjects. He also has an avid interest in the field of data analytics and data visualization.



**Krati Agarwal** is B.Tech final year Computer Science student from Vellore Institute of Technology, Vellore. Being strong in mathematics from childhood and loving statistics on the other hand, she has developed a lot of interest in Data Science and its related fields. She is very much interested in learning new technology and working on new ideas. She also has participated in more than 20 hack-a-thons.

